ZÁPADOČESKÁ UNIVERZITA V PLZNI

FAKULTA ELEKTROTECHNICKÁ

Katedra aplikované elektroniky a telekomunikací

BAKALÁŘSKÁ PRÁCE

Systém pro navigaci v rozsáhlých budovách a areálech na platformě android

Lukáš Fuchman **2014**

Abstrakt

Práce se zabývá problematikou navigační aplikace v rozsáhlých budovách a areálech na platformě Android. Následně pak její tvorbou. Aplikace umožňuje navigovat v podlažích dle zadaných místností. V aplikaci je použitá standardní XML databáze pro vyhledávání kanceláří.

Klíčová slova

Navigace v budovách, mapový systém, Android, mobilní aplikace.

Abstract

The subject of this bachelor thesis is application for navigation in vast buildings and complexes. Primary output of this thesis is application for Android platform which meets functional criteria. Application allows to navigate to the predefined rooms in individual levels. For search function it uses standard XML database..

Key words

Navigation in buildings, map system, Android, mobile aplication

Prohlášení

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

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

.....
podpis

V Plzni dne 6.6.2014 Jméno příjmení

Obsah

Seznam symbolů a zkratek

- Java IDE (Integrated Development Environment) - softwarová aplikace, umožňující snadněji psát a ladit programy v jazyce java
- API (application programming interface) - rozhraní pro programování aplikací
- DP (Density-independent pixel) - virtuální pixel používané při definování rozhraní
- XML (Extensible Markup Language) - značkovací jazyk
- SVG (Scalable Vector Graphics) - grafický vektorový formát popisující dvourozměrnou grafiku pomocí XML

Úvod

V některých rozsáhlých budovách, zejména v počátku je složité se zorientovat. Nový či občasný návštěvník musí obvykle zdlouhavě či vyptáváním dohledávat trasu. Pro takové návštěvníky je zaměřena aplikace v této práci. Práce rozebírá problematikou navigování v uzavřených budovách a tvorbou navigační aplikace pro platformu Android.

Text se nejprve zaobírá všeobecnou problematikou navigací pod platformou Android a verzováním Androidu. Poté se přesune k samotné realizaci mobilní aplikace jako je struktura a uživatelské rozhraní v aplikaci. Dále se zabývá optimálním výpočtem trasy a problematikou vykreslování map i tras. V neposlední řadě je v práci řešena XML databáze kanceláří.

Pro navigaci mezi budovami je rozebíráno rozhraní Google Map Android API V2 a je ukázáno jeho použití v aplikaci. Pro správu map je v práci popsána multiplatformní aplikace, sloužící především k popisu map a zjednodušení generování XML souboru.

1 Operační systém android

Android je komplexní operační systém pro mobilní zařízení od společnosti Google. Je založený na linuxovém jádře, navržen především pro dotykové mobilní zařízení, jako jsou chytré telefony či tablety. Dnes jej můžeme zahlédnout i na jiných zařízeních, jako jsou televize [1].

Celková popularita Androidu je vysoká, na chytrých telefonech zastupoval Android ke dni 28. května 2014 celkem 80,2 % a podle výzkumu IDC worldwide mobile phone tracker bude Android v roce 2018 sloužit na 77,6 % všech chytrých telefonech [2].

1.1 Vývoj aplikací

Na vývoj aplikací je Googlem doporučený a podporovaný nástroj Eclipse [3], založený na Java IDE. Eclipse je open source nástroj, který nám umožňuje kompletní vývoj aplikace, debugování, simulaci různých verzí androidu a velikosti obrazovek, návrh grafického rozhraní pomocí XML apod. [4].

1.2 Verzování

První beta Androidu vyšla v listopadu roku 2007, první komerční verze Androidu (Android 1.0) vyšla v září roku 2008. Od té doby vývoj pokročil o 19 verzí.

V následujícím odstavci jsou vypsány jednotlivé verze, seřazené chronologicky společně s API. Od verze Androidu 1.5 se začaly k verzím přidružovat názvy zákusků, jdoucích abecedně [5].

Android 1.0 (API level 1)
Android 1.1 (API level 2)
Android 1.5 Cupcake (API level 3)
Android 1.6 Donut (API level 4)
Android 2.0 Eclair (API level 5)
Android 2.0.1 Eclair (API level 6)
Android 2.1 Eclair (API level 7)

Android 2.2–2.2.3 Froyo (API level 8)

Android 2.3–2.3.2 Gingerbread (API level 9)

Android 2.3.3–2.3.7 Gingerbread (API level 10)

Android 3.0 Honeycomb (API level 11)

Android 3.1 Honeycomb (API level 12)

Android 3.2 Honeycomb (API level 13)

Android 4.0–4.0.2 Ice Cream Sandwich (API level 14)

Android 4.0.3–4.0.4 Ice Cream Sandwich (API level 5)

Android 4.1 Jelly Bean (API level 16)

Android 4.2 Jelly Bean (API level 17)

Android 4.3 Jelly Bean (API level 18)

Android 4.4 KitKat (API level 19)

1.3 Četnost výskytu verzí

Následující tabulka ukazuje četnost výskytu jednotlivých verzí, měřených podle přístupů na obchod Google Play, aktuální k datu 1. května 2014. Data pro statistiky jsou získávána v cyklu sedmi dní, přístupem na Obchod, na který se lze připojit od verze Androidu 2.2 a výše.

Verze	Název Androidu	API	Zastoupení
2.2	Froyo	8	1%
2.3.3 - 2.3.7	Gingerbread	10	16.2%
3.2	Honeycomb	13	0.1%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	13.4%
4.1.x	Jelly Bean	16	33.5%
4.2.x		17	18.8%
4.3		18	8.5%
4.4	KitKat	19	8.5%

Tab. 1 Četnost výskytu verzí Androida [6]

Z tabulky lze vyčíst, že android 4.0.3 a výše má na zařízeních zastoupení 82,7 %. Android od verze 2.2 do 4.0 má jen 17,3 % a vzhledem k malé, až téměř žádné distribuci, podíl zastoupení klesá.

V mobilní aplikaci, je minimální podpora od verze Androidu 3.0 Honeycomb (API level 11), jednak z důvodu výše uvedeném, ale hlavně z důvodu přímé podpory grafického rozhraní.

2 Problematika tvorby navigační aplikace

2.1 Podpora různých typů obrazovek

Android umožňuje používat široké spektrum různých velikostí obrazovek a rozlišení. Každé zařízení je osazené jinou obrazovkou, a aby bylo možné zobrazovat aplikace korektně, na všech zařízeních správně, je zde zaveden parametr DP [7].

Jeden DP představuje jeden skutečný pixel při rozlišení 16 px/dpi. Koresponduje tedy s fyzickou velikostí obrazovky [8]. Zařízení si podle velikosti obrazovky, kterou zná, přepočítá DP na pixely. Na různých typech obrazovek bude tedy DP vždy na stejném místě.

Obrázky se pro různé hustoty obrazovek řeší pomocí modifikátoru popsaného v tabulce níže [7]. Každý obrázek je v aplikaci nahrán ve více rozlišeních s odpovídající kvalifikací a obvykle se volí ta nejbližší.

Kvalifikace	Popisek
ldpi	Zdroje pro nízkou hustotu (ldpi) obrazovek (~120 dpi).
mdpi	Zdroje pro střední hustotu (mdpi) obrazovek (~160 dpi). (Základní hustota.)
hdpi	Zdroje pro vysokou hustotu (hdpi) obrazovek (~240 dpi).
xhdpi	Zdroje pro extra vysokou hustotu (xhdpi) obrazovek (~320 dpi).
nodpi	Zdroje pro všechny hustoty. Jedná se o nezávislé zdroje. Systém u takto označených zdrojů nemění velikost, bez ohledu na velikost obrazovky.
tvdpi	Zdroje, které jsou někde mezi mdpi a hdpi; zhruba 213 dpi. Primárně určené pro televizory.

Tab. 2 Modifikátory pro různé typy obrazovek [7]

2.2 Zobrazení Map

Na uchování a následné vykreslování map je vhodné použít vektorové uchování dat. Důvodem je snadná editace a rychlá případná aktualizace mapy. Dále odpadá nutnost používat kvalifikátory a tedy nutnost v aplikaci udržovat mapy ve více rozlišeních.

V mobilní aplikaci je použit vektorový formát SVG. Je vhodný především pro jeho jednoduchost, rozšířenost a jeho podporu. SVG uchovává dvourozměrnou grafiku, kterou popisuje pomocí značkovacího jazyka XML [9].

2.3 Google mapy

Jako venkovní mapu lze v aplikacích použít Google Maps Android API. Jedná se o API s přístupem na servery Google Maps, dokáže tedy stahovat mapové podklady, do kterých dokáže vkládat značky zadaných lokací, křivky polygony i bitmapovou grafiku [10]. Největší výhodou Google Maps Android API je její schopnost zjistit lokaci mobilního zařízení [11]. Lze jí tedy s výhodou použít k navigaci, mezi budovami v areálu.

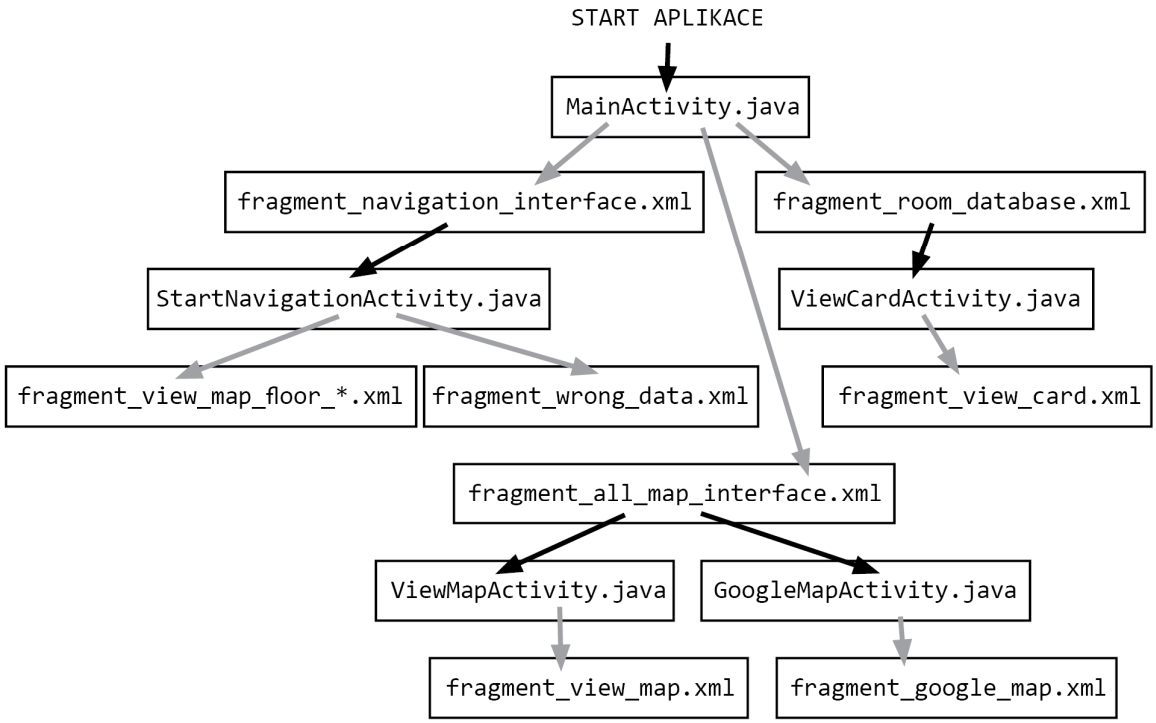
2.4 Zjišťování polohy uvnitř budovy

Zjištění polohy uvnitř budovy je velice problematické, až skoro nemožné. GPS signál není uvnitř budovy zachytitelný. U WIFI signálu vznikají interference, které znemožňují přesně a jednoduše určit polohu. Proto se v aplikaci výchozí poloha určuje zadáním dat uživatelem.

3 Struktura Aplikace

Kód aplikace je rozdělen do dvou balíčků `fuchmanl.navigacevbudovach.activity` a `fuchmanl.navigacevbudovach.jclass`. V prvním balíčku jsou, jak vypovídá název jen aktivity [12]. V druhém balíčku jsou pomocné třídy, které se v případě potřeby volají z balíčku prvního.

Uživatelské rozhraní využívá layouty [13] popsané ve značkovacím jazyce XML. Layouty v aplikaci představují převážně fragmenty, které jsou v kódu podle potřeby volány avykreslovány.



Obr. 1 Struktura aplikace

Na obr. 1 je znázorněna struktura aplikace. Šedé šipky znázorňují možnost spuštění fragmentu a černé šipky možnost spuštění aktivity.

4 Uživatelské rozhraní v aplikaci

4.1 Aktivita MainActivity.java

Základní kámen aplikace je třída `MainActivity.java` s metodou `onCreate()` [14], jejíž ukázka kódu je níže, celý kód metody lze najít v příloze A. Jedná se o FragmentAktivitu [15], která je v `AndroidManifest.xml` popsána jako hlavní, a tak se při spuštění aplikace spustí jako první.

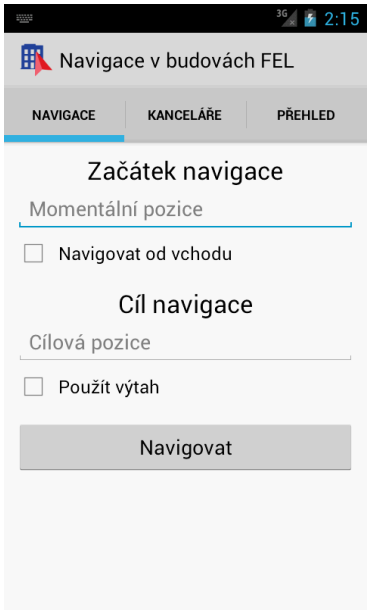
```
AppSectionsPagerAdapter mAppSectionsPagerAdapter;
ViewPager mViewPager;

@Override
protected void onCreate(Bundle savedInstanceState) {
	super.onCreate(savedInstanceState);
	setContentView(R.layout.activity_main);

	mAppSectionsPagerAdapter = new AppSectionsPagerAdapter(
			getSupportFragmentManager());

	final ActionBar actionBar = getActionBar();
	actionBar.setDisplayHomeAsUpEnabled(false);
	actionBar.setNavigationMode(ActionBar.NAVIGATION_MODE_TABS);

	mViewPager = (ViewPager) findViewById(R.id.pager);
	mViewPager.setAdapter(mAppSectionsPagerAdapter);
```

V tomto kódu se nadefinuje třída FragmentPagerAdapter [16], ViewPager [17] a ActionBar [18]. `FragmentPagerAdapter`, viz kód níže, vrací data, která používá ViewPager, jako je název, počet fragmentů a fragmenty samotné, celý kód lze najít v příloze B. ViewPager slouží k přesunu mezi fragmenty přejetím po obrazovce.

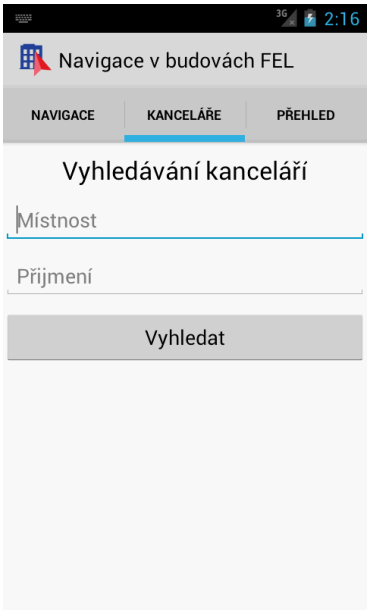
```
@Override
	public Fragment getItem(int i) {
		switch (i) {
		case 0:return new ViewNavigationFragment();
		case 1:return new ViewRoomDatabaseFragment();
		case 2:return new ViewMapSectionFragment();
		default:return null;
		}
	}
	@Override
	public int getCount() {
		return 3;
	}
	@Override
	public CharSequence getPageTitle(int position) {
```

```
            switch (position) {
            case 0:
                  return NAVIGACE;
            case 1:
                  return KANCELARE;
            case 2:
                  return PŘEHLED;
            default:
                  return "nic";
            }
}
```

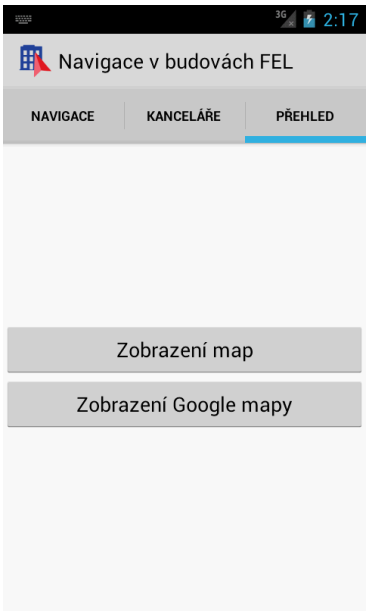
Toto rozhraní bylo zvoleno pro jeho intuitivnost a rychlost. Spustí-li se aplikace, rovnou se načte první fragment - obr. 2, do kterého lze rovnou zadat data. Obr. 3 a obr. 4 ukazují zbývající dva fragmenty. Takovýto systém rozhraní je velice efektivní a často se v aplikacích využívá [19].



Obr. 2 Uživatelské rozhraní pro navigaci



Obr. 3 Uživatelské rozhraní pro vyhledávání v kancelářích



Obr. 4 Uživatelské rozhraní pro přehled o aplikaci

4.1.1 Fragment pro navigaci

Níže je ukázka XML kódu pro fragment (obr. 2), `fragment_navigation_interface.xml`, celý kód fragmentu lze najít v příloze C. Fragment slouží k získání dat od uživatele a následné předání přes Intent do aktivity `StartNavigationActivity.java`.

ScrollView [20] je Layoutový kontejner pro View [21]. Pokud je View větší než fyzická oblast obrazovky, ScrollView obalí View a umožní nám view posouvat po obrazovce. V tomto případě je ScrollView velice důležitý, pokud se vysune klávesnice pro zadávání dat, fyzická plocha displeje se zmenší a ScrollView tedy umožní nalistovat i na spodní položky.

LinearLayout slouží k uspořádání View do řádky nebo do sloupce [22].

TextView, EditText a CheckBox přirozeně definují následující tři prvky, needitovatelný text, editovatelný text a zaškrtávací box. Tyto tři prvky se opakují dvakrát jen s jiným textem, ukázkový kód ukazuje jen první tři.

```
<?xml version="1.0" encoding="utf-8"?>
<ScrollView xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/scrollView1"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent" >

    <LinearLayout
        android:layout_width="match_parent"
```

```
android:layout_height="wrap_content"
android:orientation="vertical" >

<TextView
    android:id="@+id/start_string"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="center_horizontal"
    android:layout_marginTop="10dp"
    android:text="@string/start_string"
    android:textAppearance="?android:attr/textAppearanceLarge" />
```

4.1.2 Fragment pro vyhledávání v kancelářích

Fragment je vyobrazený na obr. 3, má název `fragment_room_database.xml` a slouží k získání dat od uživatele, které předává aktivitě `ViewCardActivity.java`. Tento fragment používá stejné View jako Fragment pro navigaci, proto následující kód ukazuje jen objekt, který je navíc, a to button [23].

```
<Button
    android:id="@+id/room_button_find"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_marginTop="10dp"
    android:layout_marginLeft="10dp"
    android:layout_marginRight="10dp"
    android:text="@string/room_button_find" />
```

4.1.3 Fragment pro přehled o aplikaci

Fragment je vyobrazený na obr. 4, má název `fragment_all_map_interface.xml` a slouží hlavně jako informační. Odkazuje na dvě aktivity `ViewMapActivity.java` a `GoogleMapActivity.java`. První slouží k informačnímu zobrazení všech map, druhá slouží k zobrazení Google mapy.

V tomto fragmentu je nechán prostor pro případné další rozšiřování aplikace. Například by se zde ještě mohlo nechat zařadit tlačítko o aplikaci, které by spouštělo informační aktivitu obsahující dané aplikace.

4.2 Aktivita StartNavigationActivity.java

Aktivita používá stejné rozhraní jako `MainActivity.java`, jediný rozdíl je v dynamickém přidělování rozsahu fragmentu, podle počtu vykreslovaných pater.

5 Zobrazení map a tras v aplikaci

Jak bylo řečeno výše, pro mapy bude použit formát SVG. Bohužel, mapové podklady se podařilo získat jen pro budovu FEL [24]. Z tohoto důvodu je aplikace zaměřena hlavně na tuto budovu. Pokud bychom získali mapové poklady i z ostatních budov, lze je do aplikace přidat obdobným způsobem jako současné mapy.

5.1 Načtení SVG

Pro načítání formátu SVG je použita knihovna třetí strany AndroidSVG, která umožňuje poměrně jednoduše a elegantně formát SVG zobrazit [25].

Nejprve se musí knihovna do aplikace importovat. Poté do layoutu, kde chceme vykreslit SVG, se vloží níže popsaný element [26]. Ukázkový kód je vyjmutý z layoutu `fragment_view_map_floor_1.java`, který slouží k vykreslení prvního patra. V XML atributu `svgimageview:svg` je zadána cesta k SVG. Ostatní atributy elementu jsou totožné jako u ImageView [27]. Každé patro má svůj vlastní layout s odpovídající cestou k SVG souboru.

```
<com.caverock.androidsvg.SVGImageView
        android:id="@+id/image_view_svg_1"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        svgimageview:svg="@drawable/fel_1" />
```

Následně můžeme fragment s tímto elementem vykreslit běžným způsobem. Pro ukázku je níže uvedený kód z `StartNavigationActivity.java`, kde XMLPath je pomocná třída ze `fuchmanl.navigacevbudovach.jclass`, přičemž je z ní použita metoda `getFloorMap`, která vyžaduje jako parametr číslo a vrací cestu k XML souboru odpovídající danému patru.

Konečně, `LayoutInflater` konktretizuje daný XML soubor, který metoda pomocí return vrací. `counterFragment` je pomocná vnitřní proměnná, která počítá počet vykreslených fragmentů.

```
XMLPath xmlpath = new XMLPath();

@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container,
                         Bundle savedInstanceState) {

      View rootView = inflater.inflate(
```

```
            xmlpath.getFloorMap(importRoom[counterFragment]),
            container, false);

            counterFragment++;
            return rootView;
        }
}
```

5.2 Vykreslení trasy

Po vykreslení mapy ve formátu SVG je potřeba ještě vykreslit trasu odpovídající zadání od uživatele.

Algoritmus popsaný v kapitole 6 vyhledá odpovídající trasu a uloží jí do ArrayListu [28], ze kterého se vykresluje trasa.

5.2.1 Layout pro vykreslení trasy

V každém layoutu daného patra, je kromě SVG mapy přidán ještě element ImageView [27], viz kód níže, který má nastavený atribut background na plnou průhlednost. Rozložení layoutu je zvoleno `RelativeLayout` [29] a oba elementy jsou nastavené na roztažení po celé ploše. Tím zapříčiníme, že se oba elementy překrývají, ale element ImageView z důvodu jeho průhlednosti není vidět.

```
<ImageView
   android:id="@+id/draw_line_image_view_1"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:background="#00000000" />
```

5.2.2 Třída pro fyzické vykreslení trasy

Do průhledného elementu z předchozí podkapitoly, se vykresluje trasa pomocí pomocné třídy `DrawRoute.java` z balíčku `fuchmanl.navigacevbudovach.jclass`, jejíž ukázka kódu je uvedena níže a celý kód lze najít v příloze D. Metoda `DrawRoute` přebírá ArrayList, který je naplněn body trasy. Metoda `draw` pak jednoduše ArrayList prochází a mezi body trasy vykresluje červenou čáru, představující trasu.

```
@Override
```

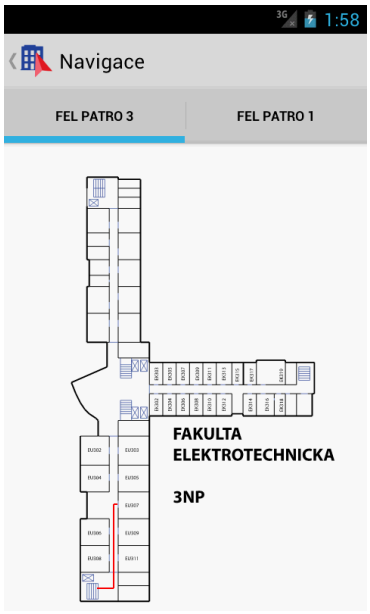
```
public void draw(Canvas canvas) {

	Paint paint = new Paint();
	paint.setColor(Color.RED);
	paint.setStrokeWidth(2);

	if (pointsRoute.isEmpty() == false) {

		for (int i = 0; i < pointsRoute.size() - 1; i++) {
			canvas.drawLine(Integer.parseInt(pointsRoute.get(i).x),
					Integer.parseInt(pointsRoute.get(i).y),
					Integer.parseInt(pointsRoute.get(i + 1).x),
					Integer.parseInt(pointsRoute.get(i + 1).y),paint);
		}
	}
}
```

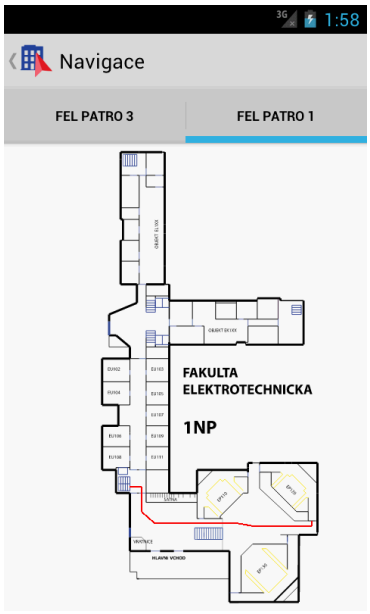
5.2.3 Přizpůsobení trasy velikosti obrazovky

SVG mapa se při vykreslování na obrazovku roztáhne automaticky, ale u bodů trasy, které jsou vztažené k původní velikosti SVG mapy, to tak není. Proto jsou body trasy taktéž přepočítány na novou velikost mapy.

Nejdříve se tedy musí zjistit velikost vykreslené mapy. Zde vystává značný problém, zjistit velikost lze až po vykreslení fragmentu. Nelze tedy jednoduše vykreslovat trasu při tvorbě fragmentu.

K vykreslení trasy jsou zvoleny dvě metody, první je `onWindowFocusChanged` a druhá `onTabSelected`. První metoda je volána, když jsou v okně vyvolány změny [30], proto se zavolá při přechodu vykreslování fragmentu s mapou. Druhá metoda se zavolá, když uživatel v uživatelském rozhraní vstoupí do jiné záložky [31]. Trasa je tedy vykreslená při přesunu do druhého fragmentu.





Obr. 5 Ukázka navigování z EU307 do EP120 Obr. 6 Ukázka navigování z EU307 do EP120

V metodách z předchozího odstavce je již možné zjistit velikost fragmentu a tedy přepočítat a vykreslit mapu, viz ukázka kódu níže. Celý kód lze najít v příloze E. Podmínka kontroluje proměnné typu boolean, sloužící jako kontrola chyb a blokuje vykreslování. `findViewById` hledá podle id identifikátoru view v XML [32], v našem případě za pomocí metody `getXMLFragmentViewMapFloorSvg` ve třídě `XMLPath.java` vracíme indentifikátor na SVG mapu. `getMeasuredHeight` a `getMeasuredWidth` vrací velikost fragmentu, ve kterém je vykreslená SVG mapa [33], `getIntrinsicHeight` a `getIntrinsicWidth` vrací originální velikost SVG mapy [34].

```
        if (hasFocus == true && wrongData == false && onlyOnce == false) {
        onlyOnce = true;
        imageview =
        findViewById(xmlpath.getXMLFragmentViewMapFloorSvg(importRoom[0]));

        workHeight = ((ImageView) imageview).getMeasuredHeight();
        worktWidth = ((ImageView) imageview).getMeasuredWidth();

        workHeightOrig = ((ImageView) imageview).getDrawable().getIntrinsicHeight();
        worktWidthOrig = ((ImageView) imageview).getDrawable().getIntrinsicWidth();
}
```

Pomocí čtyř velikostí popsaných v předešlém odstavci, metoda `calcScreenPosition` přepočte body trasy na odpovídající velikost. Protože známe velikost plochy, do které je vykreslena mapa, ale neznáme její přesné rozměry, musíme zavést předpoklad, že mapa bude

vždy roztažena na výšku bez mezer, což je vzhledem k charakteru obrazovek téměř vždy. Výpočet pak může být následující:

$$S_m = \frac{V}{V_o} S_o \tag{5.1}$$

Rovnice (5.1) představuje výpočet šířky mapy. Kde V je výška mapy po vykreslení; V_o je originální výška mapy a S_o je originální šířka mapy.

$$S_{mz} = \frac{S - S_m}{2} \tag{5.2}$$

Rovnice (5.2) vypočte, v jaké šířce mapa začíná, kde S je šířka mapy po vykreslení.

$$X = S_m + \frac{X_o \cdot V}{S_o} \tag{5.3}$$

Rovnice (5.3) vypočte novou velikost pro souřadnici x, kde X_o je originální souřadnice x.

$$Y = \frac{Y_o \cdot V}{V_o} \tag{5.4}$$

Poslední rovnice (5.4) přepočte velikost pro souřadnici y, kde Y_o je originální souřadnice y.

Níže popsaný kód ukazuje metodu `calcScreenPosition`, kde jsou předchozí rovnice implementovány. Celý kód metody lze najít v příloze F.

```
for(int i = 0; i < pointsRouteSt.size(); i++){
      pointsRouteSt.get(i).x = Integer.toString(worktWidthStart +
((Integer.parseInt(pointsRouteSt.get(i).x)*worktWidthPicture)/worktWidthOrig));
                  pointsRouteSt.get(i).y =
Integer.toString(((Integer.parseInt(pointsRouteSt.get(i).y)*workHeight)/workHeight
Orig));
}
      drawroute = new DrawRoute(pointsRouteSt);
      return drawroute;
```

5.3 Implementace Google mapy

Google mapy se do aplikací implementují podobně jako AndroidSVG. V layoutu se nadefinují, viz ukázkový kód níže a poté se v aplikaci layout zavolá.

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/map"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    class="com.google.android.gms.maps.SupportMapFragment">
```

5.3.1 Oprávnění Google mapy

Aby Google mapa mohla fungovat, potřebuje jisté náležitosti. Jedna z nich jsou specifická oprávnění, která se definují v AndroidManifest.xml [35]. Umožňují aplikaci přistupovat např. k internetu [36]. Google mapy potřebují oprávnění ukázané v kódu níže.

```
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission
android:name="com.google.android.providers.gsf.permission.READ_GSERVICES" />
    <!-- External storage for caching. -->
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
    <!-- My Location -->
    <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
```

5.3.2 Certifikace aplikace pro použití google mapy

Pro přístup na servery Google map je potřeba přidat API klíč do aplikace. Klíč se získá pomocí Google APIs Console podepsáním digitálního certifikátu, čímž obdržíme soukromý jedinečný klíč [35].

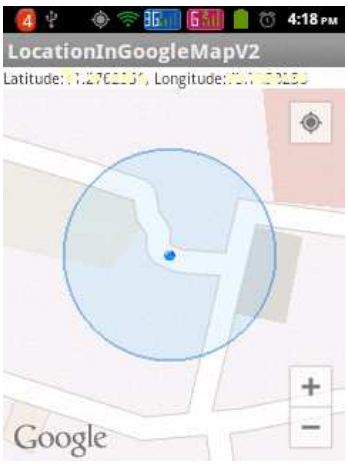
Bohužel i přes značné úsilí se pro tuto aplikaci nepodařilo získat tento digitální podpis. Při přesném postupu podle dokumentace se povedl získat digitální klíč, který byl vyplněn

do AndroidManifestu, viz kód níže. I přesto se aplikace nemůže připojit k mapovým serverům. Možností proč čím je tak způsobeno může být více: Nedodržení postupu a tedy opomenutí důležitého kroku, špatné a vygenerování a podobně. Tudíž nebylo dále pokračováno v práci s Google mapami.

```
<meta-data
      android:name="com.google.android.maps.v2.API_KEY"
      android:value="AIzaSyCvGBr5in13NK2yYBR7lhXTtnxj3mrXQy4" />
```

V případě správné podepsání aplikace by se nadále pokračovalo nadefinováním markeru jednotlivých budov do mapy, které je velice jednoduché, viz ukázkový kód převzatý z dokumentace [37]. Poté samotnou lokalizaci uživatele, viz obr. 7, u které do Manifestu stačí přidat příslušné oprávnění.

```
private GoogleMap mMap;
mMap = ((MapFragment) getFragmentManager().findFragmentById(R.id.map)).getMap();
mMap.addMarker(new MarkerOptions()
        .position(new LatLng(10, 10))
        .title("Hello world"));
```



Obr. 7 Zobrazení aktuální polohy v Google Map Android API V2 [38]

6 Vyhledávání trasy v aplikaci

Veškeré body jsou uloženy v XML databázi, odkud si program body načítá a poté s nimi následně pracuje. Soubory XML jsou generovány v multiplatformní aplikaci popsané v poslední kapitole.

6.1 Popis bodů trasy

Pro každé patro je samostatný XML soubor, níže je ukázkový výtah kódu z prvního patra. Celý kód z pvního patra lze najít v příloze G. Body jsou rozdělené do čtyř kategorií, body místností, body schodiště, body výtahu a body trasy. Každá kategorie má dvě souřadnice *x* a *y* představující pixely vztažené k SVG mapě. Kromě bodů trasy mají body i kategorii bodu sousednosti. Ty říkají, k jakému bodu na trase jsou přidruženy, sousednost je důležitá pro algoritmus vykreslující trasu. Body místností mají navíc jméno místnosti.

Body místností mohou být uspořádány libovolně, tedy na pořadí nezáleží. Naopak je to u ostatních kategorií. Body trasy musí dodržet dané pořadí, na směru tedy záleží. U bodu schodů a výtahu záleží na pořadí tak, že schody nebo výtah v každém patře musí mít stejnou pozici. Musí tedy být nad sebou.

```
<!-- Body místností a hl. vchodu -->
<room_point><name_room>EU111</name_room><X>186</X><Y>431</Y><neighbor><n_X>181</n_
X><n_Y>431</n_Y></neighbor></room_point>

<room_point><name_room>EU109</name_room><X>186</X><Y>403</Y><neighbor><n_X>181</n_
X><n_Y>403</n_Y></neighbor></room_point>

 <!-- Body schodů. Záleží na pořadí, schody na sebe navazují. -->
<stairs_point><X>192</X><Y>257</Y><neighbor><n_X>192</n_X><n_Y>245</n_Y></neighbor
></stairs_point>

<!-- Body výtahu -->
<elevator_point><X>160</X><Y>450</Y><neighbor><n_X>236</n_X><n_Y>520</n_Y></neighb
or></elevator_point>

<!-- Body trasy. Záleží na pořadí, trasa se vykresluje postupně. -->
<route_point><X>414</X><Y>525</Y></route_point>
<route_point><X>360</X><Y>525</Y></route_point>
<route_point><X>334</X><Y>525</Y></route_point>
<route_point><X>311</X><Y>520</Y></route_point>
<route_point><X>192</X><Y>245</Y></route_point>
```

6.2 Parsování XML

Pro parsování XML je zde použita knihovna `org.xmlpull.v1.XmlPullParser` [39]. Ukázka kódu uvedeného níže patří do metody parsující body schodiště a demonstruje tak použití parsování. Celý kód metody lze najít v příloze H. Do proměnné `eventType` se pomocí metody `next` ukládá tag z XML dokumentu; struktura ve while tag porovnává a hledá shodu. Pokud shodu najde, metoda `nextText` přečte hodnotu a zapíše ji do arrayListu, který se používá k uložení bodů potřebných pro vykreslení trasy.

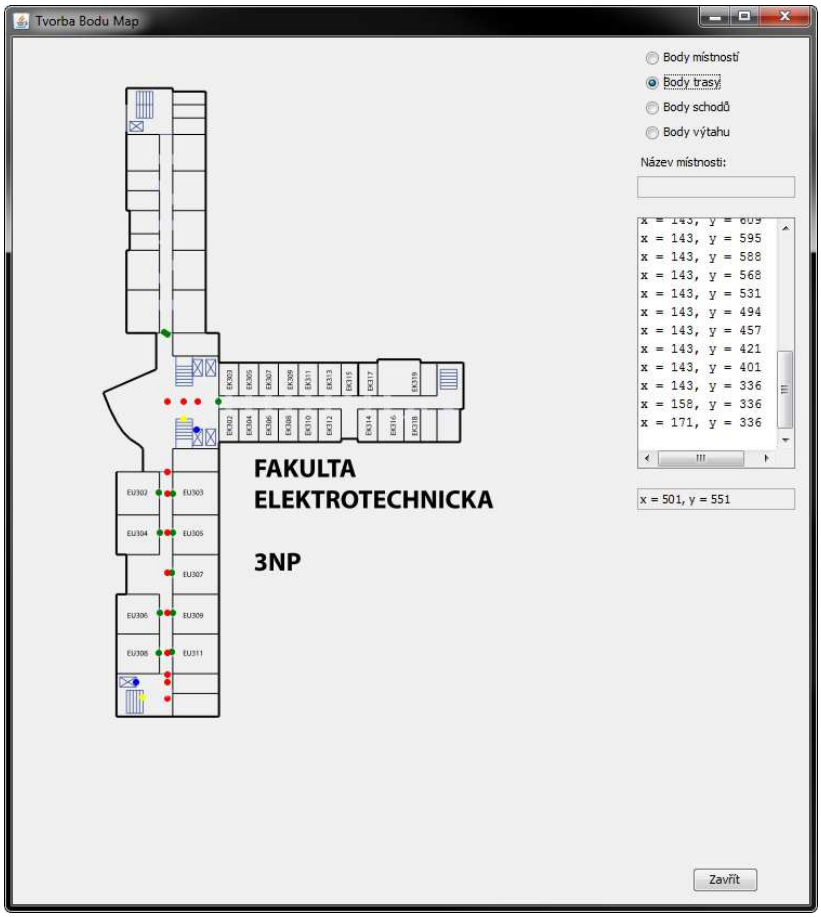
```
eventType = parser.next();
while (eventType != XmlPullParser.END_DOCUMENT) {

	if (eventType == XmlPullParser.START_DOCUMENT) {
	}

	if (eventType == XmlPullParser.START_TAG) {

		String name = parser.getName();

		if (name.equals(STAIRSPOINT)) {
			currentPoint = new myPointStair();
			reallyIn = true;

		} else if (currentPoint != null && reallyIn == true) {
			if (name.equals(POINT_X)) {
				currentPoint.x = parser.nextText();

			} else if (name.equals(POINT_Y)) {
				currentPoint.y = parser.nextText();
}
```

Zde ukázaná metoda parsování je obdobně používaná pro body místností i pro body trasy. Parsování pro body výtahu není v aplikaci implementováno, ačkoliv body v XML jsou připravené. Parsovací metoda by v tomto případě byla prakticky totožná, jen arrayList se naplní body výtahu místo schodiště.

6.3 Algoritmus pro vyhledávání trasy

Algoritmus se snaží být co nejúspornější, z důvodu rychlého vykreslení i na výpočetně slabších zařízeních. V každém patře, je pomocí bodu trasy popsána jedna trasa (červené body viz obr. 8), na kterou jsou pomocí bodu sousednosti napojené okolní body místností, schodů a výtahů. Body sousednosti tedy vždy odpovídají nějakému bodu trasy, viz obr. 8; červené body jsou body trasy, zelené body místností, žluté body schodů a modré body výtahu. Nejprve se zjistí, kde leží místnosti, připojí se na trasu a body mezi nimi dají výslednou trasu.

Pokud je potřeba navigovat mezi patry, určí se nejbližší schody a trasa naviguje k nim, př. od nich.



Obr. 8 Body přidružené k mapě třetího patra

Program realizující algoritmus pro výpočet trasy je rozdělený na dvě části. První a jednodušší část se provede, pokud se vyhledává jen v jednom patře, druhá část pak pokud bude trasa přes více pater. Níže bude popisována převážně druhá část, která je složitější.

6.3.1 Vyhledávání trasy mezi patry

Trasa se vyhledává zároveň s parsováním, důvod je prostý, pokud zadaná trasa neodpovídá, nebo se trasa vykresluje krátká, není potřeba parsovat celý dokument a tedy zbytečně využívat výpočetní výkon zařízení. V následujících odstavcích bude odkazováno na metody, z nichž některé zde z důvodu velkého rozsahu nebudou uvedeny, lze je najít na přiloženém CD.

Nejprve zjistíme odpovídající patra zadaných místností. K tomu slouží třída `WorkWithString.java` obsažená v balíčku `fuchmanl.navigacevbudovach.jclass` a její metoda `occupiedFloor`. Tato třída vyhledá a vrátí první číslo ze zadané místnosti, které představuje patro. Způsobem popsaným výše, toto číslo použijeme k načtení odpovídajícího XML souboru, obsahujícího body daného patra.

Jako první se parsují body místnosti pomocí `parseXMLRoomPoint`, k zadané místnosti uživatelem se přiřadí její skutečné a sousední souřadnice pomocí metody `findMatchesRoom`. Je-li vyhodnoceno navigování mezi více party, je volána i metoda pro parsování schodiště. Obě metody uloží body do arrayListu.

Následně probíhá porovnávání s body trasy metodou `findRouteMoreFloor`, jejíž ukázka je uvedena níže a celý kód lze najít v příloze I, která je volána během parsování bodu trasy. Dlouhá podmínka v kódu hledá shodu souřadnic sousednosti u bodu místnosti s body trasy. Pokud je taková shoda nalezena, nahodí se proměnná savePoint, která dovoluje ukládat trasu. Pokud je nalezena druhá shoda, která tedy znamená konec trasy, proměnná savePoint se shodí a body se přestanou ukládat. Tímto způsobem se tedy uloží jen trasa mezi body místností př. mezi body schodiště a místnosti nebo naopak. Zbývá už jen na začátek a konec přidat samotný bod místnosti či schodiště, aby trasa byla kompletní.

Celý systém popsaný výše se provede dvakrát, tedy pro každé patro zvlášť a body se uloží separátně.

```
if ((currentPoint.x.equals(pointsSrairSt.get(position).neighborX) &&
       currentPoint.y.equals(pointsSrairSt.get(position).neighborY))||
       (currentPoint.x.equals(pointsRoom.get(0).neighborX) && currentPoint.y
                           .equals(pointsRoom.get(0).neighborY))) {

       if (savePoint == true) {
              savePoint = false;
              pointsRouteSt.add(currentPoint);
              pointsRouteSt.add(addFirstRoomSt(currentPoint));
       } else {
              savePoint = true;
              pointsRouteSt.add(addFirstRoomSt(currentPoint));
       }
}
if (savePoint == true) {
       pointsRouteSt.add(currentPoint);
}
```

6.3.2 Vyhodnocení nebližšího schodiště

Z více schodišť v budově musí aplikace určit to nevhodnější, nejlépe schodiště, které vede k celkové nejkratší trase.

Aby aplikace nemusela počítat všechny možné trasy a ty následně porovnávat, což by bylo velice zdlouhavé, jsou vzdálenosti schodiště propočítány ještě před samotným vykreslováním trasy.

Algoritmus představuje metoda `shortestRoad` viz kód níže, používá Pythagorovu větu a je volána jen při vyhledávání trasy ve více patrech. Celý kód metody lze najít v příloze J. V každém patře spočte vzdušné vzdálenosti od místnosti ke schodům, které sečte. Tento výpočet provede pro všechny možné schodiště a schodiště s nejmenším výsledkem a tedy nejkratší trasou uloží do proměnné, která po zbytek vykreslování trasy udává pozici schodiště.

```
for (int i = 0; i < pointsSrairSt.size(); i++) {
      if(i < pointsSrairEnd.size()){
            x = Math.abs(Integer.parseInt(pointsRoom.get(0).x) -
Integer.parseInt(pointsSrairSt.get(i).x));
            y = Math.abs(Integer.parseInt(pointsRoom.get(0).y) -
Integer.parseInt(pointsSrairSt.get(i).y));
            resultSt = Math.sqrt((x*x)+(y*y));

            x = Math.abs(Integer.parseInt(pointsRoom.get(1).x) -
Integer.parseInt(pointsSrairEnd.get(i).x));
            y = Math.abs(Integer.parseInt(pointsRoom.get(1).y) -
Integer.parseInt(pointsSrairEnd.get(i).y));
            resultEnd = Math.sqrt((x*x)+(y*y));

            if (result > (resultSt + resultEnd)) {
                  result = (resultSt + resultEnd);
                  position = i;
            }
      }
}
```

Algoritmus jako matematický vzoreček by vypadal takto:

$$Z = \sqrt{|x_s - x_e|^2 + |y_s - y_e|^2}\,, \tag{6.1}$$

kde x_s představuje souřadnici x místnosti, x_e souřadnici x schodů, y_s souřadnici y místnosti a y_e souřadnici y schodů. Rovnice (6.1) se aplikuje na obě patra a následný součet udává celkovou vzdálenost ke schodům.

Tento Algoritmus je velice rychlý, jeho slabost může nastat ve velice složitých budovách, kdy vzdušná cesta bude sice nejkratší, ale skutečná cesta ke schodům velice strukturovaná a nepřímá.

V případě implementování funkce navigování k výtahům místo schodům, může zůstat tento algoritmus nezměněn.

7 Ošetření zadaných dat v aplikaci

7.1 Ošetření při spouštění Intent

Při spuštění nové aktivity, aplikace předává data ze staré aktivity do nové pomocí Intent [40]. Už na této úrovni, před spuštěním nové aktivity se kontrolují uživatelské data. Kontroluje se, zda uživatel vyplnil správně všechna data. Pokud se tak nestalo, nová aktivita se nespustí a aplikace varuje uživatele pomocí `Toast` [41].

Toto ošetření se provádí v aktivitě `MainActivity.java`. Níže je ukázka kódu při předávání dat z `fragment_room_database.xml` do `ViewCardActivity.java`. Celý kód pak lze najít v příloze K.

```
if(roomName.equals("") == false || name.equals("") == false){
	startActivity(intent);
}else{
	Toast.makeText(getActivity(), WARRINGROOM,Toast.LENGTH_SHORT).show();
}
```

7.2 Ošetření řetězců

Uživatel může zadat korektní data, ale rozličného formátování, proto jsou zadané řetězce, ale i řetězce z databáze ošetřeny.

Metody na ošetření řetězců se nachází ve třídě `WorkWithString.java`, která je v balíčku `fuchmanl.navigacevbudovach.jcass`. Jedná se o metodu `deleteDiacritics`, která má za úkol odstranit z řetězce diakritiku a `deleteSpaces`, která odstraňuje z řetězce mezery.

Metoda `deleteDiacritics` jejíž úkázka kódu je níže, vyhledává v řetězci znak s diakritikou uložený v charovém poli a nahrazuje ho znakem bez diakritiky taktéž uloženým v charovém poli na stejné pozici. Celý kód metody je v příloze L.

```
char[] diacritics = {'á', 'ä', 'č', 'ď', 'é', 'ě', 'í', 'ľ', 'Í', 'ň', 'ó', 'ö',
'ő', 'ô', 'ř', 'ŕ', 'š', 't', 'ú', 'ů', 'ü', 'ű', 'ý', 'ž'};
char[] noDiacritics = {'a', 'a', 'c', 'd', 'e', 'e', 'i', 'l', 'l', 'n', 'o', 'o',
'o', 'o', 'r', 'r', 's', 't', 'u', 'u', 'u', 'u', 'y', 'z'};

for(int i = 0 ;i < charArrayString.length;i++){
	for(int j = 0 ;j < diacritics.length;j++){
		if(charArrayString[i] == diacritics[j]){
			charArrayString[i] = noDiacritics[j];
```

```
            }
        }
}
```

Metoda `deleteSpaces` viz kód níže, funguje obdobně jako předchozí metoda. Rozdíl je takový, že pokud v řetězci najde mezeru, vytvoří nový řetězec, který má o jedna menší velikost a do tohoto řetězce překopíruje starý řetězec bez mezery. Celý kód metody lze najít najít v příloze M.

```
if (charArrayString[i] == ' ') {
     char[] charArrayStringHelp = new char[charArrayString.length-1];

      for (int j = 0; j < i; j++) {
            charArrayStringHelp[j] = charArrayString[j];
      }

      for (int j = i; j < charArrayString.length - 1; j++) {
            charArrayStringHelp[j] = charArrayString[j + 1];
      }
```

8 Databáze v aplikaci

Databáze v aplikaci je vytvořena pomocí značkovacího jazyka XML z důvodu snadné správy dat a zachování integrity aplikace.

V databázi se vyhledává parsováním, které je popsané v kapitole 6.2. Vyhledávání běží v aktivitě `ViewCardActivity.java` a po prohledání vykreslí fragmenty `fragment_view_card.xml` s výsledky. Kritéria pro vyhledávání jsou buď jméno, místnost nebo obojí, přičemž aplikace prohledá databázi a vykreslí veškeré nalezené shody.

Pro ukázku je níže uveden kód jedné položky `<room>` z databáze. Položka `<title>` obsahuje místnost, která je porovnávána při vyhledávání podle místnosti, položka se vypíše do hlavičky fragmentu; `<name>` je položka, podle které se porovnává vyhledávání při zadání příjmení, tato položka se nevykresluje, slouží jen pro vyhledávání. Ostatní položky už svým názvem vypovídají, jaká data obsahují. Tyto položky slouží už jen k vykreslení.

```
<room>
  <title>EK 602</title>
  <name>KROPÍK</name>
  <full_name>Ing. Petr  KROPÍK, Ph.D.</full_name>
  <workplace>KTE</workplace>
  <email>pkropik@kte.zcu.cz</email>
  <phone>4639</phone>
</room>
```

9 Multiplatformní aplikace pro údržbu mobilní aplikace

Multiplatformní aplikace je realizována v programovacím jazyce Java v IDE NetBeans [42]. Slouží především pro usnadnění definování mapových bodů pro mobilní aplikaci. Multiplatformní aplikace načítá mapy ve formátu SVG a v reálném čase generuje XML soubor s body.

9.1 Ovládání multiplaformní aplikace

Načítání XML souboru není vyvedeno na grafické rozhraní, cesta souboru se tedy musí měnit v kódu programu. K tomu slouží třída `Value.java`, kód je uveden níže. Proměnná `FILEPATHSVG` udává cestu a název mapy, `FILEPATHXML` pak cestu a název generovaného XML souboru.

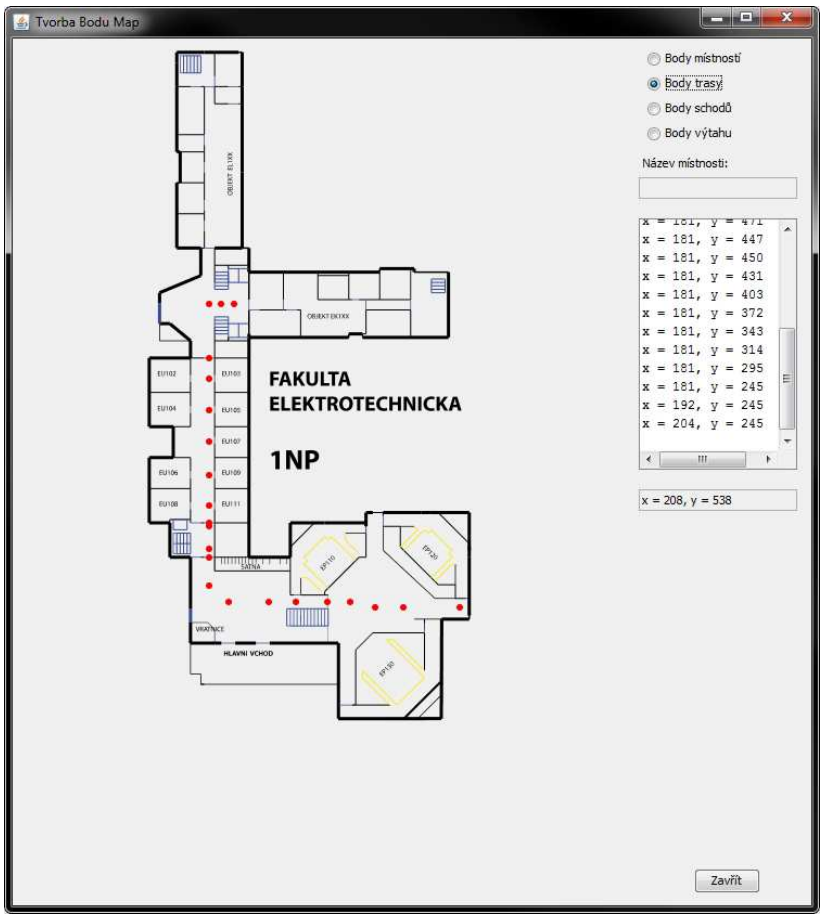
```
public class Value {
    private final String FILEPATHSVG = "src/fel_4.svg";
    private final String FILEPATHXML = "src/fel_4.xml";

    public String getFilePathSVG(){
        return FILEPATHSVG;
    }

        public String getFilePathXML(){
        return FILEPATHXML;
    }
}
```

Vybráním radiobuttonu *body místností* viz obr. 9, se aktivuje položka *Název místnosti*, kam se před nakliknutím bodu napíše název místnosti. Pokud tak neučiníme, je nutno název doplnit ruční editací XML souboru. Body místností se vždy definují na dveře.

Body trasy se definují od spodní části mapy nahoru postupně, jak jsou za sebou. Algoritmus pro vykreslování trasy vykresluje body postupně, a tedy při nedržení postupnosti bude vykreslená trasa graficky přeskakovat. Body trasy by měly být umístěné tak, aby se na ní mohly snadno navázat ostatní body. Dále pak v takové hustotě od sebe, aby aplikace nemusela parsovat zbytečně velké množství bodů, ale zároveň dostatečný počet bodů na líbivé vykreslení. Body jsou mezi sebou propojovány úsečkou, bude-li tedy použito příliš málo bodů, trasa může působit nevzhledně.

Body schodů a výtahu mají obdobný princip jako body místností, musí navazovat na body trasy, jen nemají svůj název. Jak již bylo popsáno dříve, schodiště a výtahy musí v XML zaujímat vždy stejnou polohu jako schodiště nebo výtah na obdobné pozici v jiném patře. Algoritmus vykreslování trasy si tak spojí, které schodiště či výtah je navzájem propojené. Při složitější budově s velkým a rozmanitým počtem schodišť a výtahů, je nutno přidávat tagy a jednotlivé schody a výtahy tak kategorizovat.

S každým přidáním nového bodu v grafickém rozhraní, se přidá bod i do XML souboru na poslední řádek. Generování bodu do souboru tak probíhá průběžně při zadávání trasy pomocí grafického rozhraní aplikace a paralelně lze provádět případné přímé ruční změny v souboru XML.



Obr. 9 Grafické rozhraní multiplatformní aplikace

Po správném nadefinování bodů je nutné vygenerovaný XML soubor doeditovat ručně, ukázka vygenerovaného kódu souboru je ukázána v kapitole 6.1. Program nepřiřazuje body sousednosti, ale jen prázdnou strukturu připravenou k vyplnění:

```
<neighbor><n_X> </n_X><n_Y> </n_Y></neighbor>
```

Do položek `<n_X>` a `<n_Y>` je nutno vyplnit odpovídající bod trasy. Po doplnění bodu sousednosti je již XML soubor použitelný.

Je-li nakliknut nějaký bod nepřesně, při editaci XML souboru je možné upravením souřadnic takový bod srovnat. Pro správnou orientaci v souřadnicovém systému aplikace slouží v aplikaci poslední rámeček, který v reálném čase ukazuje pozici kurzoru.

Mapy do aplikace jsou takto předělané do pátého patra. Ostatní patra lze předělat stejným způsobem. Kanceláře a místnost EL a EK jsou v XML zapsána jako EL* a EK*, kde hvězdička znamená číslo patra. Takto je voleno z důvodu nedostupnosti přesného seznamu místnosti a neúplné mapy. Místnosti EL* a EK* směřují ke dveřím celkového vstupu.

Závěr

Operační systém Android je velice rozšířený a v budoucnu tomu nejspíše nebude jinak. Android nabízí dobré možnosti pro vývojáře a velice dobrou podporu. Z těchto důvodů se jeví jako ideální platforma na vývoj aplikací toto typu.

Vytvořená aplikace obsahuje navigaci mezi podlažími dle zadaných parametrů. Dále obsahuje databázi, ve které jsou obsažené kanceláře budovy FEL, a umožňuje v nich snadno vyhledávat. Jako největší problém při vývoji se projevila špatná dostupnost mapových pokladů, které jsou k dispozici jen pro budovu FEL. Aplikace je tedy zaměřená především na tuto budovu, avšak kód v aplikaci je dostatečně univerzální pro přidání další budovy.

Aplikace umožňuje navigovat podle zadání startovní a cílové místnosti, nebo od vchodu do cílové místnosti. Pro navigaci mezi patry výtahem místo schodů je aplikace efektivně připravena. Algoritmus na vykreslování trasy probíhá velice rychle i na pomalejších zařízeních.

Aplikace obsahuje Google Map Android API V2, která umožňuje stahovat mapy z Google map serveru a zjišťovat aktuální pozici zařízení. Toto API lze snadno využít k navigaci v areálu. Bohužel vzhledem k nepovedenému pokusu aplikaci podepsat, nemá aplikace přístup k mapovým serverům, a tak je v této práci řešení popsáno teoreticky. V případě úspěšného podepsání aplikace lze snadno implementovat GPS modul, značky budov v mapě apod.

K aplikaci byla vytvořena multiplatformní aplikace pro tvorbu a správu map. Umožňuje jednoduše definovat body mapy pro použití v mobilní aplikaci. Není však naprosto automatická a pro definování velkého objemu map je doporučeno aplikaci nadále vyvíjet pro zefektivnění práce.

Jak multiplatformní aplikace, tak mobilní aplikace je připravena pro případný následující vývoj, který je pro maximální funkčnost a nasazení aplikací do oběhu doporučený.

Seznam literatury a informačních zdrojů

[1] *Wikipedia*. *Android (operating system)* [online]. [cit. 1.6.2014]. Dostupné z: http://en.wikipedia.org/wiki/android_(operating_system)

[2] *Phonearena. What will the smartphone market look like in 2018? IDC says it knows* [online]. ©2014 [cit. 1.6.2014]. Dostupné z: http://www.phonearena.com/news/What-will-the-smartphone-market-look-like-in-2018-IDC-says-it-knows_id56615

[3] *Eclipse* [online]. ©2014 [cit. 1.6.2014]. Dostupné z: http://www.eclipse.org

[4] *Developer android. Developer Tools* [online]. [cit. 1.6.2014]. Dostupné z: http://developer.android.com/tools/index.html

[5] *Wikipedia*. *Android version history* [online]. [cit. 1.6.2014]. Dostupné z: http://en.wikipedia.org/wiki/Android_version_history#Android_1.5_Cupcake_.28API_level_3.29

[6] *Developer android. Dashboards* [online]. [cit. 1.6.2014]. Dostupné z: https://developer.android.com/about/dashboards/index.html?utm_source=ausdroid.net

[7] *Developer android. Supporting Multiple Screens* [online]. [cit. 2.6.2014]. Dostupné z: http://developer.android.com/guide/practices/screens_support.html#support

[8] *Zdroják. Vyvíjíme pro Android: Suroviny, Intenty a jednotky* [online]. ©2014 [cit. 2.6.2014]. Dostupné z: http://www.zdrojak.cz/clanky/vyvijime-pro-android-suroviny-intenty-a-jednotky/

[9] *Wikipedia*. *Scalable Vector Graphics* [online]. [cit. 2.6.2014]. Dostupné z: http://en.wikipedia.org/wiki/Scalable_Vector_Graphics

[10] *Developer android. Introduction to the Google Maps Android API v2* [online]. [cit. 2.6.2014]. Dostupné z: https://developers.google.com/maps/documentation/android/intro

[11] *Developer android. Location Data* [online]. [cit. 2.6.2014]. Dostupné z: https://developers.google.com/maps/documentation/android/location

[12] *Developer android. Activities* [online]. [cit. 2.6.2014]. Dostupné z: http://developer.android.com/guide/components/activities.html

[13] *Developer android. Layouts* [online]. [cit. 2.6.2014]. Dostupné z: http://developer.android.com/guide/topics/ui/declaring-layout.html

[14] *Developer android. Fragment.* [online]. [cit. 2.6.2014]. Dostupné z: http://developer.android.com/reference/android/app/Fragment.html#onCreate(android.os.Bundle)

[15] *Developer android. FragmentActivity.* [online]. [cit. 3.6.2014]. Dostupné z: http://developer.android.com/reference/android/support/v4/app/FragmentActivity.html

[16] *Developer android. FragmentPagerAdapter.* [online]. [cit. 3.6.2014]. Dostupné z: http://developer.android.com/reference/android/support/v4/app/FragmentPagerAdapter.html

[17] *Developer android. Using ViewPager for Screen Slides.* [online]. [cit. 3.6.2014]. Dostupné z: http://developer.android.com/training/animation/screen-slide.html

[18] *Developer android. Action Bar.* [online]. [cit. 3.6.2014]. Dostupné z: http://developer.android.com/guide/topics/ui/actionbar.html

[19] *Developer android. Creating Swipe Views with Tabs.* [online]. [cit. 3.6.2014]. Dostupné z: http://developer.android.com/training/implementing-navigation/lateral.html

[20] *Developer android. ScrollView.* [online]. [cit. 3.6.2014]. Dostupné z: http://developer.android.com/reference/android/widget/ScrollView.html

[21] *Developer android. View*. [online]. [cit. 3.6.2014]. Dostupné z: http://developer.android.com/reference/android/view/View.html

[22] *Developer android. LinearLayout.* [online]. [cit. 3.6.2014]. Dostupné z: http://developer.android.com/reference/android/widget/LinearLayout.html

[23] *Developer android. Button.* [online]. [cit. 3.6.2014]. Dostupné z: http://developer.android.com/reference/android/widget/Button.html

[24] PŘEDOTA, Pavel. *Mapový systém pro orientaci v budově (areálu) pro mobilní telefony.* Plzeň 2012. Diplomová práce. Západočeská univerzita. Fakulta elektrotechnická. Katedra technologií a měření.

[25] *AndroidSVG.* [online]. [cit. 3.6.2014]. Dostupné z: https://code.google.com/p/androidsvg/

[26] *AndroidSVG. How to use SVGImageView*. [online]. [cit. 3.6.2014]. Dostupné z: https://code.google.com/p/androidsvg/wiki/SVGImageView

[27] *Developer android. ImageView*. [online]. [cit. 4.6.2014]. Dostupné z: http://developer.android.com/reference/android/widget/ImageView.html

[28] *Docs oracle. Class ArrayList<E>*. [online]. [cit. 4.6.2014]. Dostupné z: http://docs.oracle.com/javase/7/docs/api/java/util/ArrayList.html

[29] *Developer android. Relative Layout.* [online]. [cit. 4.6.2014]. Dostupné z: http://developer.android.com/guide/topics/ui/layout/relative.html

[30] *Developer android. Window.Callback.* [online]. [cit. 4.6.2014]. Dostupné z:

http://developer.android.com/reference/android/view/Window.Callback.html#onWindowFocusChanged(boolean)

[31] *Developer android. ActionBar.TabListener.* [online]. [cit. 4.6.2014]. Dostupné z: http://developer.android.com/reference/android/app/ActionBar.TabListener.html

[32] *Developer android. Activity.* [online]. [cit. 4.6.2014]. Dostupné z: http://developer.android.com/reference/android/app/Activity.html#findViewById(int)

[33] *Developer android.* [online]. [cit. 4.6.2014]. Dostupné z: http://developer.android.com/reference/android/view/View.html#getMeasuredHeight()

[34] *Developer android. Drawable*. [online]. [cit. 4.6.2014]. Dostupné z: http://developer.android.com/reference/android/graphics/drawable/Drawable.html

[35] *Developer android. Google Maps Android API v2.* [online]. [cit. 4.6.2014]. Dostupné z: https://developers.google.com/maps/documentation/android/start#getting_the_google_maps_android_api_v2

[36] *Developer android. Using Permissions.* [online]. [cit. 4.6.2014]. Dostupné z: http://developer.android.com/guide/topics/security/permissions.html

[37] *Developer android. Markers.* [online]. [cit. 4.6.2014]. Dostupné z: https://developers.google.com/maps/documentation/android/marker

[38] *Wptrafficanalyzer*. [online]. [cit. 4.6.2014]. Dostupné z: http://wptrafficanalyzer.in/blog/showing-current-location-in-google-maps-using-api-v2-with-supportmapfragment/

[39] *Developer android. XmlPullParser.* [online]. [cit. 4.6.2014]. Dostupné z: http://developer.android.com/reference/org/xmlpull/v1/XmlPullParser.html

[40] *Developer android. Intent.* [online]. [cit. 5.6.2014]. Dostupné z: http://developer.android.com/reference/android/content/Intent.html

[41] *Developer android. Toast.* [online]. [cit. 5.6.2014]. Dostupné z: http://developer.android.com/reference/android/widget/Toast.html

[42] *Netbeans.* [online]. [cit. 5.6.2014]. Dostupné z: https://netbeans.org

Přílohy

Příloha A - Metoda OnCreate z aktivity MainActivity.java

```
public class MainActivity extends FragmentActivity implements
		ActionBar.TabListener {

	AppSectionsPagerAdapter mAppSectionsPagerAdapter;
	ViewPager mViewPager;

	@Override
	protected void onCreate(Bundle savedInstanceState) {
		super.onCreate(savedInstanceState);
		setContentView(R.layout.activity_main);

		mAppSectionsPagerAdapter = new AppSectionsPagerAdapter(
				getSupportFragmentManager());

		final ActionBar actionBar = getActionBar();
		actionBar.setDisplayHomeAsUpEnabled(false);
		actionBar.setNavigationMode(ActionBar.NAVIGATION_MODE_TABS);

		mViewPager = (ViewPager) findViewById(R.id.pager);
		mViewPager.setAdapter(mAppSectionsPagerAdapter);
		mViewPager.setOnPageChangeListener(new ViewPager.
		SimpleOnPageChangeListener() {
			@Override
			public void onPageSelected(int position) {
				actionBar.setSelectedNavigationItem(position);
				}
			});

		for (int i = 0; i < mAppSectionsPagerAdapter.getCount(); i++) {
			actionBar.addTab(actionBar.newTab()
					.setText(mAppSectionsPagerAdapter.getPageTitle(i))
					.setTabListener(this));
		}
}
```

Příloha B - Metoda AppSectionsAdapter z aktivity MainActivity.java

```
public static class AppSectionsPagerAdapter extends FragmentPagerAdapter {

		public final String NAVIGACE = "navigace";
		public final String KANCELARE = "kanceláře";
		public final String PŘEHLED = "přehled";

		public AppSectionsPagerAdapter(FragmentManager fm) {
			super(fm);
		}

		@Override
		public Fragment getItem(int i) {
			switch (i) {
			case 0:
				return new ViewNavigationFragment();
			case 1:
				return new ViewRoomDatabaseFragment();
			case 2:
				return new ViewMapSectionFragment();
			default:
				return null;
			}

		}

		@Override
		public int getCount() {
			return 3;
		}

		@Override
		public CharSequence getPageTitle(int position) {
			switch (position) {
			case 0:
				return NAVIGACE;
			case 1:
				return KANCELARE;
			case 2:
				return PŘEHLED;
			default:
				return "nic";
			}
		}
}
```

Příloha C - Fragment fragment_navigation_interface.xml

```
<?xml version="1.0" encoding="utf-8"?>
<ScrollView xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/scrollView1"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent" >

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="vertical" >

        <TextView
            android:id="@+id/start_string"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_gravity="center_horizontal"
            android:layout_marginTop="10dp"
            android:text="@string/start_string"
            android:textAppearance="?android:attr/textAppearanceLarge" />

        <EditText
            android:id="@+id/start_room_string"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_marginLeft="10dp"
            android:layout_marginRight="10dp"
            android:hint="@string/start_room_string_hint" />

        <CheckBox
            android:id="@+id/from_entrance_checkBox"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_marginLeft="10dp"
            android:layout_marginRight="10dp"
            android:text="@string/check_from_entrance" />

        <TextView
            android:id="@+id/end_string"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_gravity="center_horizontal"
            android:layout_marginTop="10dp"
            android:text="@string/end_string"
            android:textAppearance="?android:attr/textAppearanceLarge" />

        <EditText
            android:id="@+id/end_room_string"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_marginLeft="10dp"
            android:layout_marginRight="10dp"
            android:hint="@string/end_room_string_hint" />

        <CheckBox
            android:id="@+id/use_elevator"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_marginLeft="10dp"
```

```
            android:layout_marginRight="10dp"
            android:text="@string/check_use_elevator" />

        <Button
            android:id="@+id/start_navigate"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_marginLeft="10dp"
            android:layout_marginRight="10dp"
            android:layout_marginTop="10dp"
            android:text="@string/start_navigate_string" />
    </LinearLayout>

</ScrollView>
```

Příloha D - Třída DrawRoute.java

```
public class DrawRoute extends Drawable {

	ArrayList<myPointRoute> pointsRoute;
	final int OFFSET = 2;

	public DrawRoute(ArrayList<myPointRoute> pointsRoute) {
		this.pointsRoute = pointsRoute;
	}

	@Override
	public void draw(Canvas canvas) {

		Paint paint = new Paint();
		paint.setColor(Color.RED);
		paint.setStrokeWidth(2);

		if (pointsRoute.isEmpty() == false) {

			for (int i = 0; i < pointsRoute.size() - 1; i++) {
				canvas.drawLine(Integer.parseInt(pointsRoute.get(i).x),
						Integer.parseInt(pointsRoute.get(i).y),
						Integer.parseInt(pointsRoute.get(i + 1).x),
						Integer.parseInt(pointsRoute.get(i + 1).y),
paint);
			}
		}
	}

	@Override
	public int getOpacity() {
		return 0;
	}

	@Override
	public void setAlpha(int alpha) {
	}

	@Override
	public void setColorFilter(ColorFilter cf) {
	}
}
```

Příloha E - Metoda onWindowFocusChanged z aktivity StartNavigationActivity.java

```
@Override
public void onWindowFocusChanged(boolean hasFocus) {

	super.onWindowFocusChanged(hasFocus);
	View imageview;
	DrawRoute drawroute;

	if (hasFocus == true && wrongData == false && onlyOnce == false) {
		onlyOnce = true;
		imageview = findViewById(xmlpath
				.getXMLFragmentViewMapFloorSvg(importRoom[0]));

		workHeight = ((ImageView) imageview).getMeasuredHeight();
		worktWidth = ((ImageView) imageview).getMeasuredWidth();

		workHeightOrig = ((ImageView) imageview).getDrawable()
				.getIntrinsicHeight();
		worktWidthOrig = ((ImageView) imageview).getDrawable()
				.getIntrinsicWidth();

		drawroute = calcScreenPosition(importRoom[0]);

		imageview = findViewById(xmlpath
			.getXMLFragmentViewMapFloorDrawLine(importRoom[0]));
		((ImageView) imageview).setImageDrawable(drawroute);

		drawRouteEnd = true;
	}
}
```

Příloha F - Metoda calcScreenPosition z aktivity StartNavigationActivity.java

```
public static DrawRoute calcScreenPosition(int position){
	DrawRoute drawroute;

	double scale = (((double)workHeight) / ((double)workHeightOrig));
	int worktWidthPicture = (int)(scale * worktWidthOrig);
	int worktWidthStart = (worktWidth - worktWidthPicture)/2;

	if (position == importRoom[0]) {
		for(int i = 0; i < pointsRouteSt.size(); i++){
			pointsRouteSt.get(i).x = Integer.toString(worktWidthStart +
((Integer.parseInt(pointsRouteSt.get(i).x)*worktWidthPicture)/worktWidthOrig));
			pointsRouteSt.get(i).y =
Integer.toString(((Integer.parseInt(pointsRouteSt.get(i).y)*workHeight)/workHeight
Orig));
		}
		drawroute = new DrawRoute(pointsRouteSt);
		return drawroute;
	}else{
		for(int i = 0; i < pointsRouteEnd.size(); i++){
			pointsRouteEnd.get(i).x = Integer.toString(worktWidthStart +
((Integer.parseInt(pointsRouteEnd.get(i).x)*worktWidthPicture)/worktWidthOrig));
			pointsRouteEnd.get(i).y =
Integer.toString(((Integer.parseInt(pointsRouteEnd.get(i).y)*workHeight)/workHeigh
tOrig));
		}
		drawroute = new DrawRoute(pointsRouteEnd);
		return drawroute;
	}
}
```

Příloha G - XML popis bodů prvního patra; soubor fel_1.xml

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<points>
<!-- Body místností a hl. vchodu -->
<room_point><name_room>EU111</name_room><X>186</X><Y>431</Y><neighbor><n_X>181</n_
X><n_Y>431</n_Y></neighbor></room_point>
<room_point><name_room>EU109</name_room><X>186</X><Y>403</Y><neighbor><n_X>181</n_
X><n_Y>403</n_Y></neighbor></room_point>
<room_point><name_room>EU107</name_room><X>186</X><Y>372</Y><neighbor><n_X>181</n_
X><n_Y>372</n_Y></neighbor></room_point>
<room_point><name_room>EU105</name_room><X>186</X><Y>343</Y><neighbor><n_X>181</n_
X><n_Y>343</n_Y></neighbor></room_point>
<room_point><name_room>EU103</name_room><X>186</X><Y>314</Y><neighbor><n_X>181</n_
X><n_Y>314</n_Y></neighbor></room_point>
<room_point><name_room>EU108</name_room><X>164</X><Y>431</Y><neighbor><n_X>181</n_
X><n_Y>431</n_Y></neighbor></room_point>
<room_point><name_room>EU106</name_room><X>164</X><Y>403</Y><neighbor><n_X>181</n_
X><n_Y>403</n_Y></neighbor></room_point>
<room_point><name_room>EU104</name_room><X>164</X><Y>343</Y><neighbor><n_X>181</n_
X><n_Y>343</n_Y></neighbor></room_point>
<room_point><name_room>EU102</name_room><X>164</X><Y>314</Y><neighbor><n_X>181</n_
X><n_Y>314</n_Y></neighbor></room_point>
<room_point><name_room>EL1</name_room><X>181</X><Y>194</Y><neighbor><n_X>181</n_X>
<n_Y>245</n_Y></neighbor></room_point>
<room_point><name_room>EK1</name_room><X>218</X><Y>245</Y><neighbor><n_X>204</n_X>
<n_Y>245</n_Y></neighbor></room_point>
<room_point><name_room>EP110</name_room><X>261</X><Y>514</Y><neighbor><n_X>261</n_
X><n_Y>520</n_Y></neighbor></room_point>
<room_point><name_room>EP120</name_room><X>414</X><Y>517</Y><neighbor><n_X>414</n_
X><n_Y>525</n_Y></neighbor></room_point>
<room_point><name_room>EP130</name_room><X>379</X><Y>543</Y><neighbor><n_X>360</n_
X><n_Y>525</n_Y></neighbor></room_point>
<room_point><name_room>Hlavni1</name_room><X>199</X><Y>559</Y><neighbor><n_X>199</
n_X><n_Y>520</n_Y></neighbor></room_point>
<!-- Body schodů. Záleží na pořadí, schody na sebe navazují. -->
<stairs_point><X>192</X><Y>257</Y><neighbor><n_X>192</n_X><n_Y>245</n_Y></neighbor
></stairs_point>
<stairs_point><X>164</X><Y>471</Y><neighbor><n_X>181</n_X><n_Y>471</n_Y></neighbor
></stairs_point>
<stairs_point><X>252</X><Y>534</Y><neighbor><n_X>236</n_X><n_Y>520</n_Y></neighbor
></stairs_point>
<!-- Body výtahu -->
<elevator_point><X>160</X><Y>450</Y><neighbor><n_X>236</n_X><n_Y>520</n_Y></neighb
or></elevator_point>
<elevator_point><X>204</X><Y>264</Y><neighbor><n_X>204</n_X><n_Y>245</n_Y></neighb
or></elevator_point>
<!-- Body trasy. Záleží na pořadí, trasa se vykresluje postupně. -->
<route_point><X>414</X><Y>525</Y></route_point>
<route_point><X>360</X><Y>525</Y></route_point>
<route_point><X>334</X><Y>525</Y></route_point>
<route_point><X>311</X><Y>520</Y></route_point>
<route_point><X>290</X><Y>520</Y></route_point>
<route_point><X>261</X><Y>520</Y></route_point>
<route_point><X>236</X><Y>520</Y></route_point>
<route_point><X>199</X><Y>520</Y></route_point>
<route_point><X>181</X><Y>505</Y></route_point>
<route_point><X>181</X><Y>479</Y></route_point>
<route_point><X>181</X><Y>471</Y></route_point>
```

```
<route_point><X>181</X><Y>450</Y></route_point>
<route_point><X>181</X><Y>447</Y></route_point>
<route_point><X>181</X><Y>431</Y></route_point>
<route_point><X>181</X><Y>403</Y></route_point>
<route_point><X>181</X><Y>372</Y></route_point>
<route_point><X>181</X><Y>343</Y></route_point>
<route_point><X>181</X><Y>314</Y></route_point>
<route_point><X>181</X><Y>295</Y></route_point>
<route_point><X>181</X><Y>245</Y></route_point>
<route_point><X>192</X><Y>245</Y></route_point>
<route_point><X>204</X><Y>245</Y></route_point>
</points>
```

Příloha H - Ukázka parsování z metody parseXMLStairsPoint

```
eventType = parser.next();
while (eventType != XmlPullParser.END_DOCUMENT) {

	if (eventType == XmlPullParser.START_DOCUMENT) {
	}

	if (eventType == XmlPullParser.START_TAG) {

		String name = parser.getName();

		if (name.equals(STAIRSPOINT)) {
			currentPoint = new myPointStair();
			reallyIn = true;

		} else if (currentPoint != null && reallyIn == true) {
			if (name.equals(POINT_X)) {
				currentPoint.x = parser.nextText();

			} else if (name.equals(POINT_Y)) {
				currentPoint.y = parser.nextText();

			} else if (name.equals(POINT_NEIGBOR_X)) {
				currentPoint.neighborX = parser.nextText();

			} else if (name.equals(POINT_NEIGBOR_Y)) {
				currentPoint.neighborY = parser.nextText();
				reallyIn = false;

				if (i == 0) {
					pointsSrairSt.add(currentPoint);
				} else {
					pointsSrairEnd.add(currentPoint);
				}
			}
		}
	}
	eventType = parser.next();
}
```

Příloha I - Metoda findRouteMoreFloor z aktivity StartNavigationActivity.java

```
private void findRouteMoreFloor(myPointRoute currentPoint, int counter) {
	if (counter == 0) {
		if ((currentPoint.x.equals(pointsSrairSt.get(position).neighborX) &&
currentPoint.y.equals(pointsSrairSt.get(position).neighborY))||
(currentPoint.x.equals(pointsRoom.get(0).neighborX) && currentPoint.y
.equals(pointsRoom.get(0).neighborY))) {

			if (savePoint == true) {
				savePoint = false;
				pointsRouteSt.add(currentPoint);
				pointsRouteSt.add(addFirstRoomSt(currentPoint));
			} else {
				savePoint = true;
				pointsRouteSt.add(addFirstRoomSt(currentPoint));
			}
		}
		if (savePoint == true) {
			pointsRouteSt.add(currentPoint);
		}
	} else {
		if ((currentPoint.x.equals(pointsSrairEnd.get(position).neighborX) &&
currentPoint.y.equals(pointsSrairEnd.get(position).neighborY))||
(currentPoint.x.equals(pointsRoom.get(1).neighborX) && currentPoint.y
.equals(pointsRoom.get(1).neighborY))) {

			if (savePoint == true) {
				savePoint = false;
				pointsRouteEnd.add(currentPoint);
				pointsRouteEnd.add(addFirstRoomEnd(currentPoint));

			} else {
				savePoint = true;
				pointsRouteEnd.add(addFirstRoomEnd(currentPoint));
			}
		}
		if (savePoint == true) {
			pointsRouteEnd.add(currentPoint);
		}
	}
}
```

Příloha J - Metoda shortestRoad z aktivity StartNavigationActivity.java

```
private void shortestRoad(){
	double resultSt,resultEnd;
	double result = Double.MAX_VALUE;
	int x,y;

	for (int i = 0; i < pointsSrairSt.size(); i++) {
		if(i < pointsSrairEnd.size()){
			x = Math.abs(Integer.parseInt(pointsRoom.get(0).x) -
Integer.parseInt(pointsSrairSt.get(i).x));
			y = Math.abs(Integer.parseInt(pointsRoom.get(0).y) -
Integer.parseInt(pointsSrairSt.get(i).y));
			resultSt = Math.sqrt((x*x)+(y*y));

			x = Math.abs(Integer.parseInt(pointsRoom.get(1).x) -
Integer.parseInt(pointsSrairEnd.get(i).x));
			y = Math.abs(Integer.parseInt(pointsRoom.get(1).y) -
Integer.parseInt(pointsSrairEnd.get(i).y));
			resultEnd = Math.sqrt((x*x)+(y*y));

			Log.d("MOJEINFO","nejkratsi: " + resultSt);
			if (result > (resultSt + resultEnd)) {
				result = (resultSt + resultEnd);
				position = i;
			}
		}
	}
}
```

Příloha K - Metoda onClick z fragmentu ViewRoomDatabaseFragment

```
public void onClick(View view) {
	Intent intent = new Intent(getActivity(),ViewCardActivity.class);

	EditText editText = (EditText)getActivity().findViewById(R.id.room_search);
	String roomName = editText.getText().toString();
	intent.putExtra(ViewCardActivity.INTENTROOM, roomName);

	editText = (EditText)getActivity().findViewById(R.id.room_name_search);
	String name = editText.getText().toString();
	intent.putExtra(ViewCardActivity.INTENTNAME, name);

	if(roomName.equals("") == false || name.equals("") == false){
		startActivity(intent);
	}else{
		Toast.makeText(getActivity(), WARRINGROOM,Toast.LENGTH_SHORT).show();
	}
}
```

Příloha L - Metoda deleteDiacritics z třídy WorkWithString.java

```
public String deleteDiacritics(String string){
	string = string.toLowerCase();
	char[] charArrayString = string.toCharArray();

	char[] diacritics = {'á', 'ä', 'č', 'ď', 'é', 'ě', 'í', 'ľ', 'Í', 'ň', 'ó',
'ö', 'ő', 'ô', 'ř', 'ŕ', 'š', 'ť', 'ú', 'ů', 'ü', 'ű', 'ý', 'ž'};
	char[] noDiacritics = {'a', 'a', 'c', 'd', 'e', 'e', 'i', 'l', 'l', 'n',
'o', 'o', 'o', 'o', 'r', 'r', 's', 't', 'u', 'u', 'u', 'u', 'y', 'z'};

	for(int i = 0 ;i < charArrayString.length;i++){
		for(int j = 0 ;j < diacritics.length;j++){
			if(charArrayString[i] == diacritics[j]){
				charArrayString[i] = noDiacritics[j];
			}
		}
	}

	String newString = String.valueOf(charArrayString);
	return newString;
}
```

Příloha M - Metoda deleteSpaces z třídy WorkWithString.java

```
public String deleteSpaces(String string) {
	char[] charArrayString = string.toCharArray();

	for (int i = 0; i < charArrayString.length; i++) {

		if (charArrayString[i] == ' ') {
			char[] charArrayStringHelp = new char[charArrayString.length -
1];

			for (int j = 0; j < i; j++) {
				charArrayStringHelp[j] = charArrayString[j];
			}

			for (int j = i; j < charArrayString.length - 1; j++) {
				charArrayStringHelp[j] = charArrayString[j + 1];
			}

			charArrayString =
Arrays.copyOf(charArrayStringHelp,charArrayStringHelp.length);
		}
	}

	String newString = String.valueOf(charArrayString);
	return newString;
}
```