**VŠB - Technická univerzita Ostrava**

**Fakulta elektrotechniky a informatiky**

# Bakalářská práce

**2010** **Juraj Vaněk**

**VŠB - Technická univerzita Ostrava**

**Fakulta elektrotechniky a informatiky**

**Katedra informatiky**

# Systém pro podporu dotazníkových průzkumů

# Questionnaire Survey Support System

**2010** **Juraj Vaněk**

## Poděkování

Na tomto místě bych rád poděkoval svému vedoucímu bakalářské práce, Ing. Davidu Ježkovi Ph.D. za rady a pomoc při odstraňování chyb.

## Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě dne 7. 5. 2010

_____

Juraj Vaněk

## Abstrakt

Tato práce obsahuje krátký úvod do tvorby aplikací pro technologii Google Android a zabývá se návrhem a implementací systému pro podporu dotazníkových průzkumů. Tento systém se skládá ze dvou částí. První část tvoří mobilní aplikace, která je určena pro tvorbu a vyplňování dotazníků. Druhá část – server – zase umožňuje zálohu nastavení jednotlivých zařízení, zpracování a úpravu sesbíraných dat.

Systém byl vytvářen s důrazem na mobilní část systému a využití výkonu současných chytrých telefonů. Mobilní část umožňuje vytvoření prakticky celého nastavení přímo na samotném zařízení, které lze poté pomocí serveru distribuovat mezi ostatní zařízení. Z mobilního zařízení se také můžeme pomocí internetového prohlížeče dostat přímo na server a tam spravovat jednotlivá zařízení a zpracovávat sesbíraná data. Server tak tvoří místo pro zálohu nastavení jednotlivých zařízení, úložiště sesbíraných dat a obstarává výkon, potřebný pro zpracování těchto dat.



## Klíčová slova

Google Android, Open Handset Alliance, SQL, SQLite, XML, J2EE, Struts 2, Hibernate, Spring, Java, MySQL, MVC, MySQL, Eclipse, Netbeans, chytrý telefon, mobilní aplikace, webová aplikace, systém pro podporu dotazníkových průzkumů



## Abstract

This document contains a short introduction to creating applications for system Google Android (dedicated for embedded systems) and deals with design and implementation of system for support questionnaire surveys. This system consists of two parts. The first part - mobile application - is designed for creating and completing the questionnaires. The second part - server - allows backup settings of mobile devices, manages those devices, collected data and process those collected data.

The system was developed with an emphasis on the mobile part of the system and utilizes the performance of the current smart phones. Mobile part allows the creation of whole settings directly on the device itself, which can then use the server to distribute those settings among other devices. The mobile device also can use a browser to get directly to the server and then manage each device and process the collected data. Server is the place for a backup of device settings, storage of collected data and provides a performance for processing such data.

## Keywords

Google Android, Open Handset Alliance, SQL, SQLite, XML, J2EE, Struts 2, Hibernate, Spring, Java, MySQL, MVC, MySQL, Eclipse, Netbeans, smartphone, mobile application, web application, Questionnaire Survey Support System

## Seznam použitých zkratek

HTML – Hypertext Markup Languague

JSP – Java Servlet Pages

J2EE – Java Enterprise Edition

MVC – architektura Model-View-Controller

SDK – Software Development Kit

SQL – Structured Query Languague

XML – eXtensible Markup Languague

# Obsah

# 1 Úvod

Tato práce obsahuje krátký úvod do tvorby aplikací pro technologii Google Android a zabývá se návrhem a implementací systému pro podporu dotazníkových průzkumů. Účelem těchto průzkumů je získávat názory a fakta od respondentů a na základě těchto dat získat informace, které mohou být důležité například pro vedení marketingových kampaní, nebo sledování veřejného mínění.

Navrhovaný systém by měl doplňovat ostatní možnosti sběru dat a měl by sloužit ke zmenšení režie a nákladů na straně tazatelů – mobilní aplikace je určena pro běh na platformě Google Android, která je open source projektem a data jsou uchovávána elektronicky (sníží se tedy spotřeba papíru a zjednoduší samotné zpracování získaných dat).

Systém byl vytvářen s důrazem na mobilní část systému a využití výkonu současných chytrých telefonů. Mobilní část umožňuje vytvoření prakticky celého nastavení přímo na samotném zařízení, které lze poté pomocí serveru distribuovat mezi ostatní zařízení. Z mobilního zařízení se také můžeme pomocí internetového prohlížeče dostat přímo na server a tam spravovat jednotlivá zařízení a zpracovávat sesbíraná data. Server tak tvoří místo pro zálohu nastavení jednotlivých zařízení, úložiště sesbíraných dat a obstarává výkon, potřebný pro zpracování těchto dat.

# 2 Technologie Google Android

## 2.1 Krátká historie platformy

Dne 5. listopadu 2007 byl oficiálně představen systém Android, postavený na platformě Linux. Pomocí tohoto open source projektu chce společnost Google konkurovat komerčním mobilním operačním systémům. Na jejím vývoji se podílí Open Handset Alliance - skupina složená z 34 firem, do které patří společnosti jako Motorola, Samsung Electronics, T-Mobile, LG Electronics, Intel, nVidia nebo Texas Instruments. Největší podíl na vývoji ale měl Google, který již delší dobu prohlašoval, že by se rád orientoval i na oblast mobilních technologií. V roce 2005 v tichosti koupil společnost Android Inc., kterou založil Andy Rubin, který stál například za projektem WebTV. Díky investicím společnosti Google došlo k urychlení vývoje tohoto systému.

Ještě v listopadu 2007 byla představena raná verze SDK. Následující měsíc byla na podporu vývoje aplikací pro tuto platformu spuštěna Android Developer Challange I - soutěž o nejlepší aplikaci běžící na této platformě. Bylo vybráno 50 aplikací, z nichž každá obdržela 25000 dolarů. Vývojářům těchto aplikací byla také o něco dříve zpřístupněna nová verze SDK.

První mobilní zařízení s touto platformou bylo představeno operátorem T-Mobile 23. září 2008 v New Yorku. Jednalo se o smartphone Dream společnosti HTC, která je jedním ze

zakládajících členů Open Handset Aliance. 14. října už T-Mobile registroval předobjednávky na asi 1,5 milionů kusů tohoto zařízení. V prodeji se objevil v druhé polovině října.

27. října byl spuštěn další projekt na podporu vývoje aplikací pro tento systém - Android Market (obdoba AppStore od společnosti Apple). Vývojáři tak dostali k dispozici prostředí, kde mohou nabízet své výtvory, a které mohou hodnotit a komentovat jejich uživatelé. Vývojáři se mohou zaregistrovat za poplatek 25 dolarů a poté nabízet své programy - na rozdíl od AppStore - přímo bez ověřování. Od začátku roku 2009 je možno na Android Marketu prodávat aplikace za poplatek, z něhož vývojáři dostávají 70 procent.

Od oficiálního představení se Open Handset Alliance se postupně rozšířila o firmy jako je například Sony Ericsson, AsusTek, Toshiba nebo Vodafone a koncem roku 2009 měla tato platforma již 11% podíl na trhu s mobilními aplikacemi, přestože se tento OS nacházel jen na čtrnácti modelech mobilních telefonů. Podle studie společnosti ABI Research by se tento podíl měl do roku 2014 zvýšit až na 23%.

## 2.2 Architektura

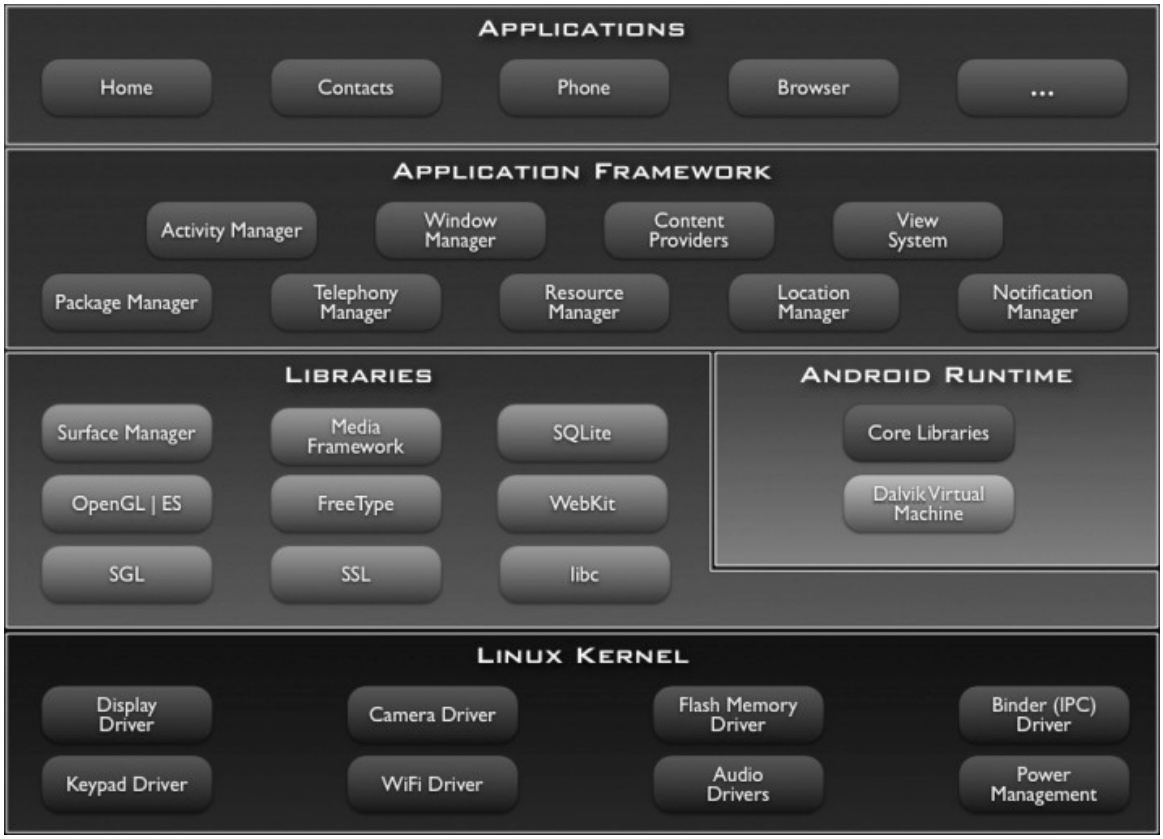
Architekturu této platformy můžeme vidět na Obr. 1 - Architektura systému Google Android. Jádro celého systému tvoří Linuxový kernel verze 2.6, který se stará o bezpečnost, správu paměti a procesů, o přístup k sítím apod. Vytváří abstraktní vrstvu mezi hardwarem a softwarovým vybavením.

Nad touto vrstvou se nachází běhové prostředí Androidu. Každá aplikace napsaná pro Android běží ve svém vlastním procesu jako instance virtuálního stroje *Dalvik*. Jedná se v podstatě o Javovský virtuální stroj, optimalizovaný pro mobilní aplikace. *Dalvik* spouští soubory ve formátu **.dex* (*Dalvik executable*) který je optimalizován tak, aby zanechával co nejmenší paměťovou stopu. Třídy jsou nejprve kompilovány do Javy a poté pomocí nástroje *dx* transformovány do formátu **.dex*. Hlavní knihovny systému obsahují většinu funkcí klasické Javy.

Platforma dále obsahuje soubor knihoven napsaných v jazycích *C/C++*, které jsou používány různými částmi platformy. V těchto knihovnách se nachází například funkcionalita pro práci s 2D (*SGL*), 3D grafikou (*OpenGL ES 1.0*), různými médii (*MPEG4*, *H.264*, *MP3*, *AAC*, *AMR*, *JPG* a *PNG*…) a obsahuje i relační databázi *SQLite*.

Nad běhovým prostředím a knihovnami se nachází aplikační framework, obsahující soubor služeb a systémů. Zde patří například *Resource Manager*, pomocí něhož aplikace získávají přístup k řetězcům, grafice, souborům vzhledu, apod., nebo *Activity Manager*, starající se o životní cyklus aplikace.

Na nejvyšší vrstvě se nacházejí samotné aplikace, které jsou buď součástí samotného systému (například SMS klient nebo kalendář), nebo aplikace nainstalované samotným uživatelem.




**Obr. 1 - Architektura systému Google Android (Zdroj: developer.android.com)**

## 2.3 Komponenty

Na rozdíl od jiných systémů, aplikace platformy Android nemají jediný vstupní bod spouštějící aplikaci (obdobu metody *main()*). Aplikace se zde skládá ze skupiny komponent, které, pokud budeme chtít, mohou využívat i jiné aplikace. Nebo naopak, my můžeme použít v naší aplikaci komponentu obsaženou v jiné aplikaci.

Existují čtyři základní typy komponent – *Activity*, *Service*, *Content provider* a *Broadcast receiver*.

### 2.3.1 Activity

Komponenta *Activity* je nejpoužívanější, v podstatě představuje uživatelské rozhraní. Můžeme mít *Activity* obsahující například seznam obrázků, další *Activity* starající se o editaci obrázku, apod.

### 2.3.2 Service

*Service* neobsahuje na rozdíl od *Activity* uživatelské rozhraní, ale může běžet na pozadí po dobu neurčitou (např. přehrávání hudby). Můžeme s ní komunikovat pomocí jí poskytnutým rozhraním.

### 2.3.3 Broadcast Receiver

Tato komponenta na rozdíl od ostatních nedělá nic, jen naslouchá celoplošným oznámením (např. přijetí SMS) a na vybraná oznámení nějakým způsobem reaguje – může například spustit námi napsanou aplikaci.

### 2.3.4 Content Provider

Tato komponenta poskytuje přístup ostatním aplikacím k datům uloženým naší aplikací. Data mohu být uložena v souborech, v databázi, nebo nějakým jiným námi definovaným způsobem (např. stahována z nějakého umístění na internetu).

### 2.3.5 Práce s komponentami

*ContentProvider* je aktivován pomocí požadavku z objektu typu *ContentResolver*. Zbývající tři komponenty aktivujeme pomocí zpráv zabalených v objektu *Intent*. Při volání komponent *Activity* a *Service* musí obsahovat název volané komponenty a data potřebná pro práci dané komponenty, při volání komponenty *BroadcastReceiver* musí obsahovat název akce, která byla vykonána.

Spuštění *Activity* můžeme provést pomocí metod *Context.startActivity(Intent I)* a *Context.startActivityForResult(Intent I)*. Po jejich zavolání systém Android zavolá metodu *onCreate()* volané *Activity*. Metodu *Context.startActivityForResult(Intent I)* využijeme tehdy, pokud od volané *Activity* vyžadujeme nějaký výsledek. Pak musíme implementovat metodu *onActivityResult()*, která bude zavolána po návratu z dané *Activity*.

Komponenty *Service* spouštíme pomocí metody *Context.startService(Intent I)* a pomocí metody *Context.sendBroadcast()* a jí podobným můžeme vytvořit celoplošné oznámení, na která mohou reagovat komponenty typu *Broadcast receiver*.

Ukončování nepotřebných komponent provádíme hlavně pomocí k tomu určených metod jejich rodičovských tříd (u *Activity* metoda *finish()* a u *Service stopSelf()*). Lze je ukončit i jinými způsoby.

## 2.4 Manifest

Tento XML soubor je dalo by se říci jádrem celé aplikace. Nachází se v kořenovém adresáři. Pomocí něj systém Android rozpozná, jaké komponenty daná aplikace obsahuje. Je zde také definována startovací komponenta – komponenta, která je spuštěna jako první při spuštění aplikace.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
      package="android.socioresearch" android:versionCode="1" android:versionName="1.0.0">

     <uses-permission android:name="android.permission.INTERNET" />

    <application android:icon="@drawable/icon" android:label="@string/app_name">
        <activity android:name=".Socioresearch android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>

        <activity android:name=".system.Registration" android:label="@string/app_name"/>
              ...
    </application>
</manifest>
```

*Ukázka struktury manifestu*

V tagu *application* se nachází deklarace všech komponent obsažených v aplikaci. Definujeme je pomocí tagu *action*. Tento tag musí obsahovat atribut s názvem dané komponenty (název třídy implementující danou komponentu). Pro každou komponentu můžeme definovat takzvaný *Intent filter*. Tento filtr umožňuje systému Android vyvolat nejvhodnější komponentu bez toho, aby znal její název (u jiné verze potřebné aplikace mohla být například potřebná komponenta přejmenována a díky tomuto filtru ji můžeme zavolat bez toho, abychom tento nový název znali).

Startovací komponenta musí obsahovat speciální filtr, který ji určuje jako spouštěcí:

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

Dále můžeme v manifestu například definovat pomocí tagu *uses-permission* jednotlivá oprávnění (např. oprávnění pro přístup na internet, pro přístup ke kontaktům mobilního zařízení apod.).

## 2.5 Zdroje

Zdroje jsou data neobsahující kód (například lokalizované řetězce, obrázky, videa, XML soubory definující uživatelské rozhraní). Existuje několik druhů zdrojů. Jednotlivé typy zdrojů se nacházejí v pro ně určených podadresářích adresáře *res*. Všechny soubory zdrojů jsou zakompilovány do výsledného souboru aplikace.

Vybrané typy zdrojů:

*res/drawable/* - Obrázky používané v aplikaci

*res/layout/* - Soubory XML obsahující definici uživatelských rozhraní komponent

*res/values/* – Soubory XML s lokalizovanými řetězci (*strings.xml*), předdefinovanými barvami (*colours.xml*) apod.

Ke zdrojům přistupujeme z kódu pomocí automaticky generované třídy *R.java*. Pro každý typ zdroje zde existuje vnitřní třída. V té se nacházejí „odkazy“ na jednotlivé zdroje ve formě konstant typu *int*:

```
public final class R {
        public static final class drawable {
                public static final int icon=0x7f020000;
        }
        public static final class id {
                public static final int alert_text=0x7f05003b;
                public static final int b_continue=0x7f050017;
                public static final int b_edit_group=0x7f050027;
        …
```

*Ukázka třídy R.java*

## 2.6 Použití zdrojů

Uživatelské rozhraní *Activity* můžeme nastavit pomocí zavolání metody *setContentView()*, kde jako argument zadáme „odkaz“ na soubor XML obsahující layout (např. *setContentView(R.layout.user_interface)*). Pomocí metody *findViewById()* zase můžeme z UI

vytáhnout potřebný objekt a dále s ním pracovat – můžeme například vytáhnout objekt typu *Button*:

```
Button b = (Button) findViewById(R.id.b_continue);
```

Nastavit jeho text:

```
b.setText(R.string.edit_user);
```

A posluchače:

```
b.setOnClickListener(new View.OnClickListener() {
    public void onClick(View v) {
      …
```

U nastavování textu můžeme vidět použití třídy *R.java* – pomocí ní ze zdrojů vybereme potřebný řetězec.

## 2.7 Uživatelské rozhraní

Uživatelské rozhraní lze vytvářet dvěma způsoby. Buď způsobem uvedeným v předchozí kapitole, nebo klasicky vytvářením jednotlivých objektů. Tyto dva způsoby lze kombinovat, můžete definovat hlavní části UI v XML souboru a poté je dynamicky upravovat a přidávat nové objekty. V XML souborech definujících uživatelská rozhraní lze použít pouze třídy dědící z třídy *View*.

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

*Ukázka XML souboru definujícího uživatelské rozhraní*

Menu dané *Activity* dostupné pod tlačítkem menu tvoříme pomocí přetížení metody *onCreateOptionsMenu(Menu m)* třídy *Activity*.

```
@Override
    public boolean onCreateOptionsMenu(Menu menu) {
        super.onCreateOptionsMenu(menu);
               menu.add(0, PREV_ID, 0, R.string.previous_question);
               menu.add(0, NEXT_ID, 0, R.string.next_question);
               …
```

*Ukázka vytvoření menu*

Výběr položky z menu zase ošetříme pomocí přetížení metody *onOptionsItemSelected(Item I)*.

```
    @Override
    public boolean onMenuItemSelected(int featureId, MenuItem item) {
        switch(item.getItemId()) {
               case PREV_ID:
               …
```

*Ukázka metody ošetřující výběr položky z menu*

## 2.8 Přístup ke zdrojům dat

### 2.8.1 Síť

K sítím přistupujeme stejným způsobem jako v klasické Javě, nesmíme však zapomenout naší aplikaci explicitně v souboru manifestu nastavit oprávnění pro přístup na internet (pomocí tagu <*uses-permission android:name="android.permission.INTERNET" />*).

### 2.8.2 Soubory

Systém Android má restriktivnější přístup k práci se soubory, soubor vytvořený v jedné aplikaci nelze jen tak použít v jiné. Pro načítání a zápis dat poskytuje Android dvě speciální metody – *openFileInput(String nazev_souboru)* a *openFileOutput(String nazev_souboru, int mod_zapisu)* - obě vracejí *File Input/Output Stream*. Zda budou moci ostatní aplikace přistupovat k souboru nebo ne, nastavujeme pomocí druhého atributu metody *openFileOutput()*. Máme několik možností:

*MODE_PRIVATE* – výchozí mód, k souboru může přistupovat jen tato aplikace

*MODE_APPEND* – jako výchozí mód, ale data zapíše na konec souboru

*MODE_WORLD_READABLE* – soubor mohou číst i ostatní aplikace

*MODE_WORLD_WRITABLE* – soubor mohou ostatní aplikace číst i do něj zapisovat

### 2.8.3 Shared preferences

Shared preferences nám umožňují uchovávat data, která mohou použít ostatní komponenty naší aplikace. Můžeme tak do nich například při přihlášení vložit jméno aktuálního uživatele a poté ho v jiné komponentě nějakým způsobem využít.

K souboru reprezentujícímu shared preferences se dostaneme pomocí metody *getSharedPreferences(String nazev, int mod)* (jednotlivé módy přístupu jsou stejné jako u souborů). Data pak získáváme a zapisujeme pomocí metod typu *getString(), setString(), getBoolean(),* apod.

### 2.8.4 Content provider

Další možností přístupu k datům je využití komponent *typu Content Provider*, které nám poskytují ostatní aplikace nainstalované na zařízení. Systém Android nám tímto způsobem například poskytuje přístup ke kontaktům nebo přijatým zprávám.

### 2.8.5 Databáze

K permanentnímu uchovávání většího množství dat nám zase poslouží databáze *SQLite*. K databázi přistupujeme pomocí objektu třídy *SQLiteOpenHelper*. Přístup k databázi získáme zavoláním metody *getWritableDatabase()*, ta nám vrátí objekt typu *SQLiteDatabase*. Pomocí tohoto objektu už můžeme provádět k tomu určenými metodami jednotlivé operace (metody q*uery(), update(), delete()*, apod.). Při dotazech získáváme data zabalená do objektu typu *Cursor*. Z něj můžeme získat jak samotná data, tak informace o počtu získaných řádků, názvy vrácených sloupců apod.

# 3 Analýza

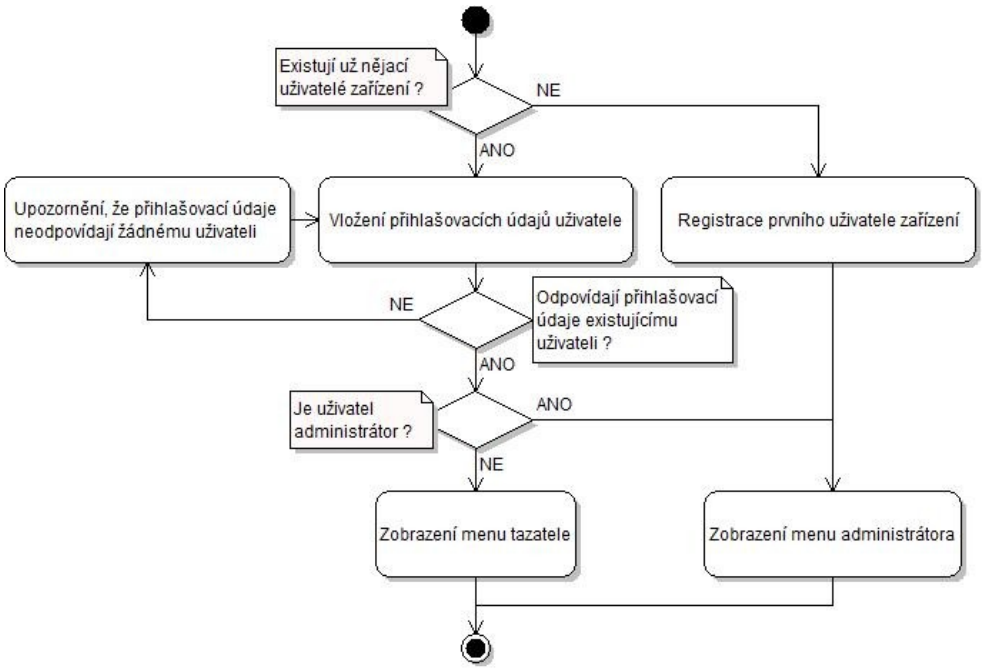
## 3.2 Mobilní aplikace

Mobilní aplikace by měla umožňovat správu uživatelů daného zařízení, zařazovat je do skupin a podle nich určovat které formuláře může daný uživatel používat. Dále by měla obsahovat možnost odeslání sesbíraných dat na server. Měla by také umět zálohovat nastavení zařízení na server a opětovně načíst toto nastavení zpět. Nakonec by také měla umožňovat vytvářet a editovat formuláře přímo na zařízení.

V aplikaci budou existovat dva typy uživatelů – administrátor a tazatel. Administrátor bude mít přístup ke všem funkcím aplikace, tazatel bude moci využívat pouze vyplňování dotazníků a jejich následné odeslání na server.

### 3.1.1. Autentizace

Protože se ze začátku v aplikaci žádní uživatelé nenacházejí, je potřeba vyřešit vytvoření prvního uživatele (obr. 2). Tento uživatel by měl být administrátor, aby mohl po svém vytvoření vytvářet ostatní uživatele zařízení. První uživatel by také neměl jít smazat, aby nedošlo k situaci, že budou smazáni všichni uživatelé a nebudeme se moci do aplikace přihlásit. Přihlašování uživatelů by mělo probíhat klasicky pomocí uživatelského jména a hesla.




**Obr. 2 – Aktivitní diagram autentizace uživatele v mobilní aplikaci**

### 3.1.2 Správa uživatelů

Administrátor by měl mít přehled o všech uživatelích, měl by je moci vytvářet, upravovat a mazat. U uživatelů bychom měli uchovávat jejich přihlašovací údaje (přihlašovací jméno, heslo), informaci o tom, zda je daný uživatel administrátor a také informaci o tom, v jaké skupině se uživatel nachází. Pokud nebude skupina vyplněna, bude automaticky vložen do výchozí.

### 3.1.3 Dotazníky

Administrátor by měl moci na zařízení vytvářet a editovat dotazníky. Dotazníky by měly jít editovat, pouze pokud ještě neexistují žádná sesbíraná data (Obr. 4). Pokud by totiž došlo ke změně struktury dotazníku, došlo by také ke ztrátě kontextu mezi jednotlivými otázkami formuláře a dříve sesbíranými daty. Při smazání dotazníku také dojde ke smazání sesbíraných dat – bez znalosti kontextu obsaženého ve formuláři jsou nám poté sesbíraná data na nic.

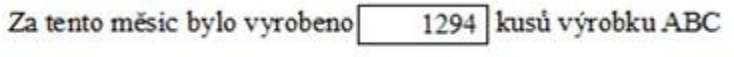
Kvůli omezení velikosti displeje mobilního zařízení může nastat problém se zobrazením složitějších otázek dotazníků. Každý složitější dotazník lze ale rozložit na jednodušší otázky – potřebujeme tedy určit jednotlivé typy těchto atomárních otázek. Existují v podstatě tři tyto základní typy:

- Vkládání informací – např. jméno, příjmení, datum narození, apod.
- Výběr jediné možnosti – například možnost výběru pohlaví (muž/žena), nebo škály (například otázka „Vyberte prosím na stupnici (od 1 do 7) jak moc jste spokojeni s naším výrobkem (kde 1 je velmi spokojeni a 7 velmi nespokojeni“).
- Výběr více možností – například otázka „Které z následujících firem zabývajících se výrobou produktu ABC znáte?“

U jednotlivých voleb každého typu otázky také musí existovat možnost vkládání textu (například vyplnění jiné možnosti než uvedené). Pomocí těchto základních otázek lze rozložit větší, hůře zobrazitelné otázky do několika samostatných podotázek.

#### *3.1.3.1 Struktura dotazníku*

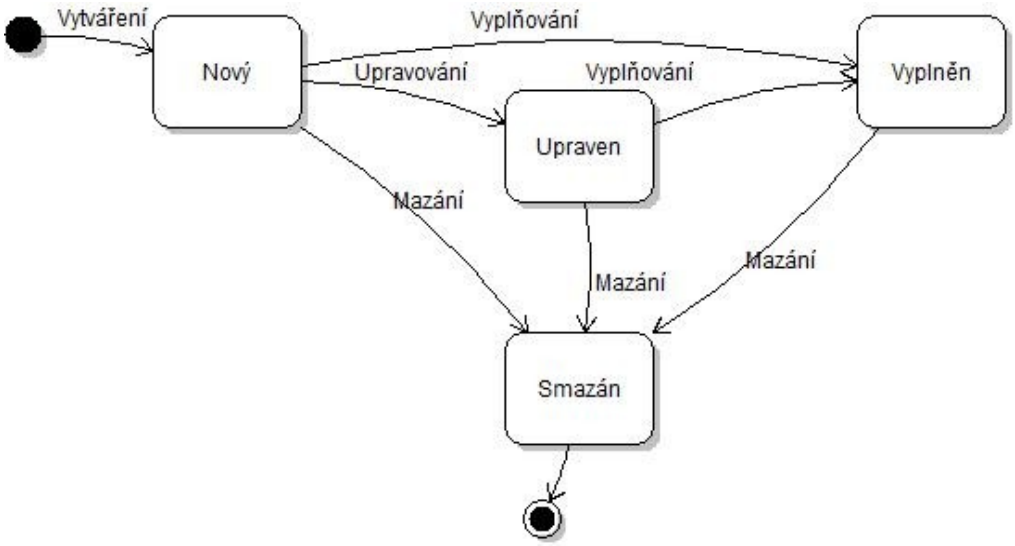
U každého dotazníku by měl být uveden nějaký úvod – popis účelu daného dotazníku. Dále by měl samozřejmě obsahovat jednotlivé otázky. Každá otázka bude obsahovat text otázky, její typ (z tří výše uvedených) a jednotlivé možnosti. Každá tato možnost bude obsahovat text dané možnosti a informaci o tom, zda bude obsahovat možnost vkládání textu. Text dané možnosti by měl být rozdělen do dvou částí, které budou obalovat případné vkládání textu. Např. u možnosti, kterou můžeme vidět na Obr. , je řetězec „Za tento měsíc bylo vyrobeno” první část a „kusů výrobku ABC“ část druhá.




**Obr.3 – Ukázka otázky, která obsahuje textový vstup**

### *3.1.3.2 Tvorba dotazníku*

Při tvorbě dotazníku musí mít uživatel možnost vytvořit celou strukturu dotazníku (viz výše) – zadat jeho název, úvod a vkládat jednotlivé otázky. U vkládání otázek musí mít možnost výběru typu otázky. Aplikace musí umožňovat libovolný pohyb po struktuře dotazníku a také vkládání a mazání otázek v libovolném místě. Stejně tak by měl mít možnost libovolného pohybu v možnostech otázky a jejich mazání a vkládání. Při tvorbě možnosti otázky musí být vložen „prefix“ a „postfix“ otázky a musí být také možno určit, zda bude vytvářená volba obsahovat vkládání textu.




**Obr. 4 – Stavový diagram dotazníku**

### *3.1.3.3 Ukládání vyplněných dat*

Kromě samotných dat, vyplněných v dotazníku by měly být uchovávány i informace o uživateli, který data vyplnil a také datum a čas vyplnění (to nám může pomoci při samotném zpracování dat – např. změna názorů dotazovaných osob během určité časové periody, apod.).

### *3.1.3.4 Vyplňování dotazníku*

Každý tazatel musí mít možnost výběru dotazníku z jemu dostupných, aby mohl vybrat ten, který chce vyplnit. Na začátku vyplňování musí být také zobrazen úvod vybraného dotazníku – může například obsahovat osnovu, pomocí které může tazatel seznámit dotazovaného s účelem

výzkumu. Poté bude následovat samotné vyplňování jednotlivých otázek dotazníku. Na konci vyplněného dotazníku musí mít tazatel možnost rozhodnout se, zda vyplněný dotazník uloží, nebo ne. Tazatel by také měl vidět, kolik už v ten daný den vyplnil dotazníků.

### 3.1.4 Skupiny uživatelů

Uživatelé budou rozděleni do jednotlivých skupin, které jim budou umožňovat přístup k různým podmnožinám formulářů. Každý uživatel bude moci být umístěn pouze v jedné skupině, každá skupina bude obsahovat několik formulářů. V zařízení by měla také existovat výchozí skupina, do které bude automaticky vložen první vytvořený uživatel a také uživatelé, u kterých ještě není jisté, do jaké skupiny je chceme umístit.

Administrátor bude moci tyto skupiny vytvářet, editovat a mazat. Pokud se budou při mazání skupiny ve skupině nacházet uživatelé, budou automaticky přesunuti do výchozí skupiny.

#### *3.1.4.1 Přidávání uživatelů do skupin*

U editace skupiny by měla být možnost výběru a odstraňování jednotlivých uživatelů skupiny. Administrátor by měl mít možnost vybrat uživatele ze seznamu uživatelů výchozí skupiny a vložit jej do potřebné skupiny a stejně jednoduše jej z dané skupiny odstraňovat. Všichni uživatelé odstranění ze skupin budou automaticky vloženi zpět do výchozí skupiny.

#### *3.1.4.2 Přidávání dotazníků do skupin*

V možnostech editace skupiny musí být také obsažena možnost vkládání a mazání dotazníků dané skupiny. Postup by měl být stejný jako u spravování uživatelů skupiny – ze seznamu všech dotazníků vybere administrátor ten, který chce do skupiny vložit. A naopak, při mazání bude vybraný dotazník ze seznamu dotazníků skupiny odstraněn a vložen zpět do skupiny výchozí.

### 3.1.5 Data na mobilním zařízení

Nyní provedeme celkový souhrn toho, co budeme muset na mobilním zařízení trvale uchovávat.

#### *3.1.5.1 Uživatelé*

U uživatelů potřebujeme uchovávat přihlašovací údaje (jméno a heslo), informaci o tom, zda je uživatel administrátor a název skupiny, ve které se uživatel nachází.

#### *3.1.5.2 Skupiny*

U skupiny nám stačí znát pouze její název. Informace o tom, kteří uživatelé do ní patří, je již obsažena u uživatelů.

#### *3.1.5.3 Formuláře*

U formuláře musíme uchovávat jeho název, úvod, a samotné otázky s jednotlivými možnostmi. Nesmíme také zapomenout na vztah formulářů a skupin. Musíme přesně vědět, jaké formuláře se ve vybrané skupině nachází.

#### *3.1.5.4 Sesbíraná data*

Sesbíraná data budou navazovat na jednotlivé formuláře. U nich potřebujeme uchovávat informaci o tom, který uživatel daný dotazník vyplnil a kdy k vyplnění došlo. Dále zde musí být obsažena vyplněná data. Ty musí odpovídat struktuře vyplňovaného dotazníku.

## 3.2 Server

### 3.2.1 Úvod

Server by měl umožňovat zálohu nastavení jednotlivých zařízení, ukládání sesbíraných dat, jejich správu a možnost zpracování těchto dat. S webovou aplikací budou pracovat dva typy uživatelů – administrátor a klasický uživatel. Administrátor bude moci využívat veškeré funkce serveru, obyčejný uživatel bude pracovat pouze se sesbíranými daty. Bude moci mazat vybraná sesbíraná data a formuláře, určovat, která data budou použita při zpracování dat a nakonec i tyto data zpracovávat.

### 3.2.2 Přihlášení

Protože na serveru budou existovat dva typy uživatelů, je potřeba je nějakým způsobem rozlišit a autentizovat. Je také potřeba vyřešit počáteční stav, kdy se v systému ještě nenachází žádný uživatel – na serveru by se měl nacházet nějaký výchozí. Uživatelé se budou na server přihlašovat klasicky pomocí uživatelského jména a hesla. Po ověření přihlašovacích údajů musíme zjistit, zda je přihlášen obyčejný uživatel nebo administrátor a tím mu umožnit přístup k funkcím, na které má nárok.

### 3.2.3 Správa uživatelů serveru

Uživatele serveru bude moci spravovat pouze administrátor. U uživatelů musíme znát informace nutné k autentizaci – přihlašovací jméno, heslo a dále informaci o tom, zda je uživatel administrátor. Měly by být také uchovávány kontaktní informace, ty ale nejsou nutné.

Administrátor by měl mít možnost vidět všechny uživatele najednou – nejlépe v seznamu, a moci vybrat toho, u kterého má v úmyslu upravovat informace, nebo kterého chce smazat. Ve správě uživatelů musí být také obsažena funkce vytváření uživatele serveru.

### 3.2.4 Správa zařízení

Administrátor bude také moci spravovat profily zařízení, která se budou na server přihlašovat. Přihlašování bude probíhat stejným způsobem jako u uživatele – pomocí názvu zařízení a hesla.

Musí mít možnost - stejně jako u spravování uživatelů – vidět všechna zařízení najednou a co nejjednodušším způsobem upravovat jejich údaje, mazat je a vytvářet nové profily zařízení.

Protože je samotné nastavení vytvářeno už na zařízení, stačí, když budeme na serveru toto nastavení moci pouze prohlížet. Může sloužit například ke kontrole, na jakých zařízeních se už daný formulář nachází apod. Administrátor by měl mít možnost vidět uživatele, formuláře a skupiny zařízení. U jednotlivých skupin zařízení by měl mít také možnost prohlížet uživatele a formuláře, které do nich patří.

### 3.2.5 Správa sesbíraných dat

Ve správě sesbíraných dat musí být možno mazat nepotřebná data a určovat, která data budou použita při zpracování a která ne. Musí umožňovat výběr co nejmenší, nejpřesnější potřebné množiny vyplněných dotazníků u všech existujících formulářů. To znamená, že musíme mít možnost zadat podmínky pro ta data, která mají všechny formuláře stejná – zařízení, na kterém byl formulář vyplněn, id uživatele, který jej vyplnil a nakonec čas, kdy došlo k vyplnění.

### 3.2.6 Zpracování dat

Zpracování dat na serveru by mělo být co nejuniverzálnější. Mělo by umožňovat získat ze sesbíraných dat maximum informací a přitom musí být jednoduché na obsluhu. Při zpracovávání dat musíme nějakým způsobem určit výsledek, který nás zajímá a podmínky, které musí výsledek splňovat.

Zpracovávány budou data, jejichž strukturu známe – jsou rozděleny podle otázek a jejich jednotlivých voleb. Uživatel, který bude tato data zpracovávat, by měl vidět přesné znění otázek a jejích voleb a pomocí těchto informací se rozhodnout, která data ho zajímají a jakými daty má být výsledek podmíněn.

Nejmenší segment dat, pomocí kterého můžeme určit tuto podmínku, je v dotazníku jedna volba otázky. To samé platí i o vzhledu výsledku, který zpracováním dat chceme získat. Proto by měl mít uživatel možnost u každé otázky vybrat volby, které se mají zobrazit ve výsledku (např. kolik procent dotazovaných, kteří odpovídají podmínkám, používá operátora XXX) a také vybrat volby určující podmínku výsledku (výsledek musí například splňovat to, že dotazovaní byli pouze muži ve věku od 20 do 30 let).

Pokud vše shrneme do několika bodů, musí systém zpracování dat obsahovat:

- Zobrazení jednotlivých otázek dotazníku
- Zobrazení voleb jednotlivých otázek
- Výběr voleb otázky, které mají být obsaženy ve výsledku
- Výběr voleb otázky, které budou reprezentovat podmínky

## 3.3 Spojení

Nyní je potřeba definovat způsob, jakým budou mezi sebou obě části systému komunikovat. Bude mezi nimi probíhat několik druhů komunikace. Po vytvoření nastavení zařízení jej budeme muset nějakým způsobem zálohovat na server. Pak jej zase musíme nějakým způsobem získat zpět. Operace odeslání sesbíraných dat na server musí být od předchozích dvou operací oddělena, protože možnost odeslání sesbíraných dat musí mít i tazatel. Také nesmíme zapomenout na samotnou autentizaci zařízení u serverové části systému.

### 3.3.1 Autentizace zařízení

Přihlášení zařízení na server bude prováděno, jak již bylo uvedeno výše (viz Správa zařízení), pomocí názvu zařízení a hesla. Tyto údaje bude moci na zařízení nastavovat pouze administrátor zařízení.

### 3.3.2 Odeslání nastavení

Na server by měly být kvůli záloze odeslány všechny údaje, které se na zařízení momentálně nacházejí – uživatelé, skupiny uživatelů a formuláře. Tato data budou na serveru uložena a musíme nějakým způsobem rozlišovat, ze kterého zařízení pocházejí.

### 3.3.3 Příjem nastavení

Pokud se na serveru nachází již zálohované nastavení, mělo by mít zařízení možnost toto nastavení získat zpět. Během přenosu musí být původní nastavení zálohováno – pokud by totiž

během spojení, nebo samotného ukládání nastavení do zařízení došlo k chybě, musí být původní nastavení obnoveno. Staré nastavení bude po úspěšném dokončení této operace definitivně smazáno.

### 3.3.4 Odeslání sesbíraných dat

Odesílání sesbíraných dat by mělo být provedeno co nejjednodušeji, stisknutím jednoho tlačítka, tak aby neměl případný tazatel možnost odesílaná data nějakým způsobem upravovat. U přenosu sesbíraných dat by měly být, kromě samotných dat, přenášeny i jednotlivé formuláře, které reprezentují kontext sesbíraných dat a které se ještě na serveru nemusí nacházet. Aby nedocházelo ke zbytečnému opětovnému odesílání starých dat, budou úspěšně odeslaná data ze zařízení smazána.

# 4 Návrh

## 4.1 Mobilní aplikace

Mobilní aplikace bude vytvořena pro běh na mobilní platformě *Google Android*. Aplikace se zde skládá ze samostatných komponent a datovou vrstvu reprezentuje relační databázový systém *SQLite*. Uživatelská rozhraní jsou zde z větší části deklarována pomocí XML souborů.

Při návrhu tedy musíme definovat jednotlivé komponenty, operace nad daty prováděné těmito komponentami a uživatelská rozhraní těchto komponent.

Ve velké části komponent se nachází stejné metody, přepsané z rodičovských tříd. Proto jsou shrnuty zde, aby nebyly zbytečně popisovány vícekrát:

*onCreate* – spouštěcí metoda, obdoba metody *main*. Většinou je zde nastaven vzhled komponenty (načtený ze souboru XML).

*onCreateOptionsMenu* – zde je vytvořeno menu komponenty a nastaveny jednotlivé položky.

*onMenuItemSelected* – tato metoda ošetřuje reakci na výběr položky z menu

*onListItemClick* – zde je vždy ošetřena událost kliknutí na položku v seznamu

### 4.1.1 Datová vrstva

Data, která budeme používat, musíme nějakým způsobem uchovávat. Většina dat bude uložena v následujících tabulkách, reprezentujících celkové nastavení zařízení. Samotné formuláře ale budeme muset uchovávat jiným způsobem, protože nikdy předem nevíme, jakou bude mít daný formulář strukturu. Budou tedy ukládány do samostatných souborů.

#### *4.1.1.1 Uživatelé*

Informace o uživatelích bude uchovávat tabulka *Users*. Budou v ní obsaženy informace o přihlašovacích údajích – jméno a heslo. Dále se zde budou nacházet informace o tom, do jaké skupiny uživatel patří a zda je administrátor. Jednotlivá uživatelská jména musí být unikátní.

#### *4.1.1.2 Skupiny*

Informace o uživatelských skupinách bude obsahovat tabulka *Groups*. Zde bude uchováván pouze název skupiny. Jednotlivé názvy musí být unikátní.

#### *4.1.1.3 Formuláře*

Seznam formulářů bude obsažen v tabulce *Forms*. Zde budou také pouze uvedeny názvy. Jednotlivé názvy musí být unikátní.

Do výše uvedených tabulek musel být kvůli implementaci přidán navíc atribut *_id* typu int – je potřebný u komponent, které rozšiřují třídu *ListActivity* a která je používána pro zobrazení požadovaných dat ve formě seznamu.

#### *4.1.1.4 Vztah mezi skupinami a formuláři*

Protože jedna skupina může obsahovat více formulářů a jeden formulář může být uveden ve více skupinách, je potřeba vytvořit vazební tabulku *Use*, která obsahuje informace o jednotlivých vazbách.

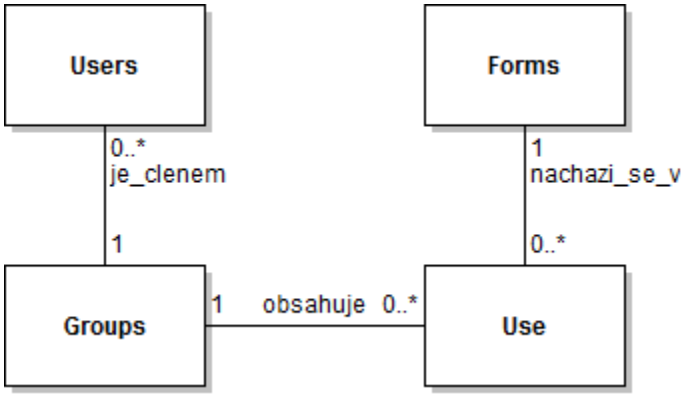
#### *4.1.1.5 Sesbíraná data*

Sesbíraná data jednotlivých dotazníků budou uchovávána v samostatných tabulkách. Každá tato tabulka ponese název dotazníku, ke kterému náleží. Tabulky budou vygenerovány na míru podle počtu otázek a počtu možností daného dotazníku. Tyto tabulky budou mít několik společných základních atributů:

- *user_id* – ID uživatele, který data vyplnil
- *date* – datum vyplnění dotazníku. Hodnota je typu *int* (místo *Date*) kvůli rychlejšímu porovnávání a vyhledávání dat na mobilním zařízení, a tím i menšímu zatěžování systémových prostředků.
- *time* – čas vyplnění dotazníku. Také zde je použit typ *int* místo *Date*.

Tabulka dotazníku dále obsahuje atributy *Q_q_ch*, kde *q* je číslo otázky a *ch* číslo možnosti dané otázky. Obsah tohoto atributu je typu *char*, protože nevíme, jaký typ dat v té určité otázce budeme ukládat. Vyhodnocení vyplněných dat je už věcí zpracování dat na straně serveru.

Ve výsledku tedy získáme strukturu, kterou můžeme vidět na obr. 5.




**Obr. 5 – Struktura databáze mobilní aplikace**

#### *4.1.1.6 Lineární zápis entit*

**primární klíč**, *cizí klíč*, *cizí i primární klíč*

| | |
|---|---|
| Users | {**_id**, name, pass, admin, *groupid*} |
| Groups | {**_id**, groupname} |
| Use | {*form_id*, *group_id*} |
| Forms | {**_id**, formname} |
| <formname> | {*user_id*, date, time, …} |

#### *4.1.1.7 Datový slovník*

| Název atributu | Datový typ | Rozsah | Klíč | Null | Popis |
|---|---|---|---|---|---|
| **Users** | | | | | |
| _id | int | | A | N | Id uživatele |
| name | char | 64 | N | N | Jméno uživatele, musí být jedinečné |
| Pass | char | 128 | N | N | Heslo |
| admin | int | 1 | N | N | Indikuje, zda je daný uživatel administrátor zařízení, nebo ne |
| *groupid* | int | | N | N | Defaultně 0 – výchozí skupina |
| **Groups** | | | | | |
| _id | int | | A | N | Id skupiny |
| Groupname | char | 64 | N | N | Název skupiny |
| **Use** | | | | | |
| *form_id* | int | | A | N | Id formuláře |
| *group_id* | int | | A | N | Id skupiny |

| **Forms** | | | | | |
|---|---|---|---|---|---|
| _id | Int | | A | N | Id formuláře |
| formname | Char | 64 | N | N | Název formuláře |
| **<formname>** název formuláře, ke kterému náleží tato tabulka) | | | | | |
| *user_id* | Int | | A | N | Id uživatele, který formulář vyplnil |
| Date | Int | | A | N | Datum vyplnění formuláře – ve formátu rrmmdd |
| Time | Int | | A | N | Čas vyplnění formuláře – ve formátu hhmmss |
| <Q_*q*_*ch*> | Char | 256 | N | A | Obsah vybrané možnosti otázky (q je id otázky a ch id možnosti). Obsah závisí na typu otázky. |

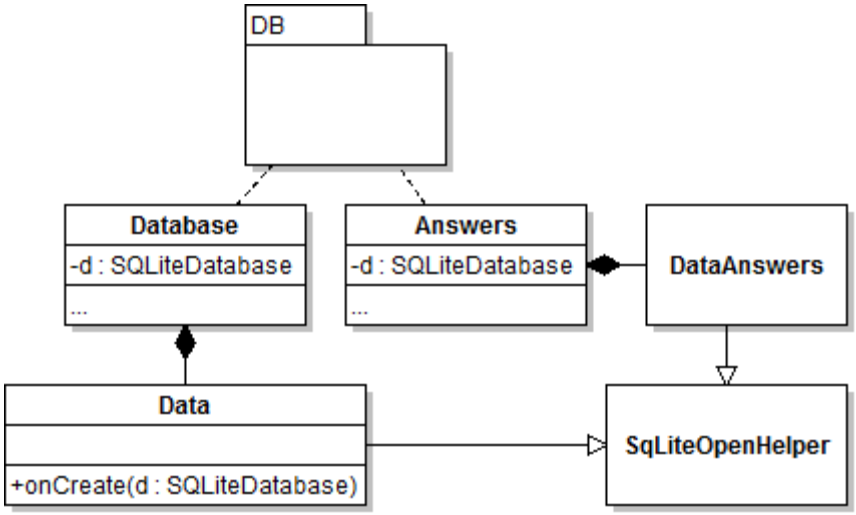
### 4.1.2 Ukládání dat

Nyní potřebujeme definovat, jakým způsobem budou všechna výše uvedená data ukládána a jak s nimi budeme moci pracovat. K tomuto účelu bude využit relační databázový systém *SQLite*.

Kvůli lepšímu přehledu bude databáze rozdělena do dvou částí – v první se bude nacházet celkové nastavení zařízení, jehož strukturu známe (tabulky uživatelů, skupin a formulářů) a v druhé data, která se nám podařilo sesbírat (tabulky jednotlivých formulářů).

Jednotlivé části budou reprezentovat samostatné třídy. Ve třídě *Database* se bude pracovat se samotným nastavením aplikace – informacemi o uživatelích, skupinách a formulářích. Zde se bude pracovat pouze s obsahem jednotlivých tabulek, vytvořených při prvním spuštění. Ve třídě *Answers* budou zase obsaženy tabulky vyplněných dat, náležející k jednotlivým formulářům. Během následujících částí návrhu budeme v těchto třídách postupně definovat metody, které budou provádět potřebné operace nad daty.

K databázím bude přistupováno pomocí objektu *SqLiteDatabase*, který bude použít v obou třídách. Tento objekt nám umožňuje provádět všechny potřebné operace nad jednotlivými tabulkami. Vytvoření, správu verzí databází a přístup k objektu *SqLiteDatabase* budou v obou třídách obstarávat vnořené třídy, nazvané jako *Data*, respektive *DataAnswers*, rozšiřující třídu *SQLiteOpenHelper*. Ve třídě *Data*, vnořené ve třídě *Database*, musí být ošetřeno vytvoření tabulek nastavení, pokud ještě neexistují. K tomu nám poslouží překrytá metoda *onCreate* třídy *SQLiteOpenHelper* – ta je zavolána vždy, když se na zařízení databáze s požadovaným názvem ještě nenachází. Nesmíme také zapomenout do nově vytvořené tabulky reprezentující skupiny vložit výchozí skupinu. Výslednou strukturu můžeme vidět na obr. 6.




**Obr. 6 – Diagram, zobrazující třídy, určené pro obsluhu databáze mobilní aplikace**

### 4.1.3 Výchozí stav aplikace

Nejprve potřebujeme vytvořit tzv. spouštěcí komponentu – to je komponenta, která je vždy spuštěna jako první při spuštění samotné aplikace (zde reprezentována třídou *SocioResearch*). Po instalaci se zde ještě nebudou nacházet žádní uživatelé. Spouštěcí komponenta bude tedy kontrolovat, zda se na zařízení už nějaký uživatel nachází. Pokud ještě ne, musíme ošetřit vytvoření tohoto uživatele. Tento uživatel bude automaticky administrátor (aby mohl vytvářet další uživatele) a bude vložen do výchozí skupiny. Potřebujeme tedy získat zbývající vlastnosti uživatele – uživatelské jméno a heslo. Heslo by mělo být pro jistotu zadáno dvakrát – jedná se o vytvoření prvního uživatele, a pokud by došlo při jeho zadávání k překlepu, už bychom se nemuseli podruhé do aplikace dostat. Po získání odpovídajících údajů bude vytvořen nový uživatel a poté bude zobrazena komponenta, umožňující přístup k potřebným funkcím – *programMenu*. K výše uvedené registraci bude sloužit komponenta *Registration*.

Menu bude pro každý typ uživatele jiné:

Menu administrátora bude umožňovat přístup k následujícím funkcím:

- správa uživatelů
- správa skupin
- správa formulářů
- vyplňování dotazníků
- odeslání sesbíraných dat
- zálohování nastavení na server / příjem nastavení ze serveru

Tazatel bude mít zase přístup pouze k následujícím funkcím:

- vyplňování dotazníků
- odeslání sesbíraných dat

Pro nastavení každé verze menu bude sloužit speciální metoda (*setAdminMenu* a *setUserMenu*).

Musíme také vytvořit způsob, pomocí kterého upozorníme uživatele v případech, kdy dojde k nějaké chybě (například zadání dvou rozdílných hesel při tvorbě prvního uživatele). K tomu využijeme komponentu *Alert* rozšiřující třídu *AlertDialog*. Ta bude pomocí dialogového okna zobrazovat požadovaný text upozornění.

Nyní musíme určit, které operace nad daty budeme provádět. Zde budeme pracovat pouze s uživateli - budeme definovat nové metody ve třídě *Database*. Potřebujeme zkontrolovat, kolik uživatelů se už na zařízení nachází (metoda *getNumberOfUsers*) a musíme také nějakým způsobem vytvořit nového uživatele (metoda *createUser*).

### 4.1.4 Autentizace

Autentizace bude prováděna pomocí klasického přihlašování. Zde pouze potřebujeme získat přihlašovací údaje – uživatelské jméno a heslo. K přihlášení nám poslouží nová komponenta - *Login*, ve které dojde k ověření poskytnutých údajů a v případě úspěchu dojde k zobrazení patřičného menu uživatele (komponenta *programMenu*).

Ve třídě *Database* vytvoříme novou metodu, určenou pro autentizaci uživatele – *loginUser*. Potřebujeme také zjistit, zda je uživatel administrátor, nebo ne – výše uvedená metoda bude tedy vracet tři hodnoty:

0 – není administrátor

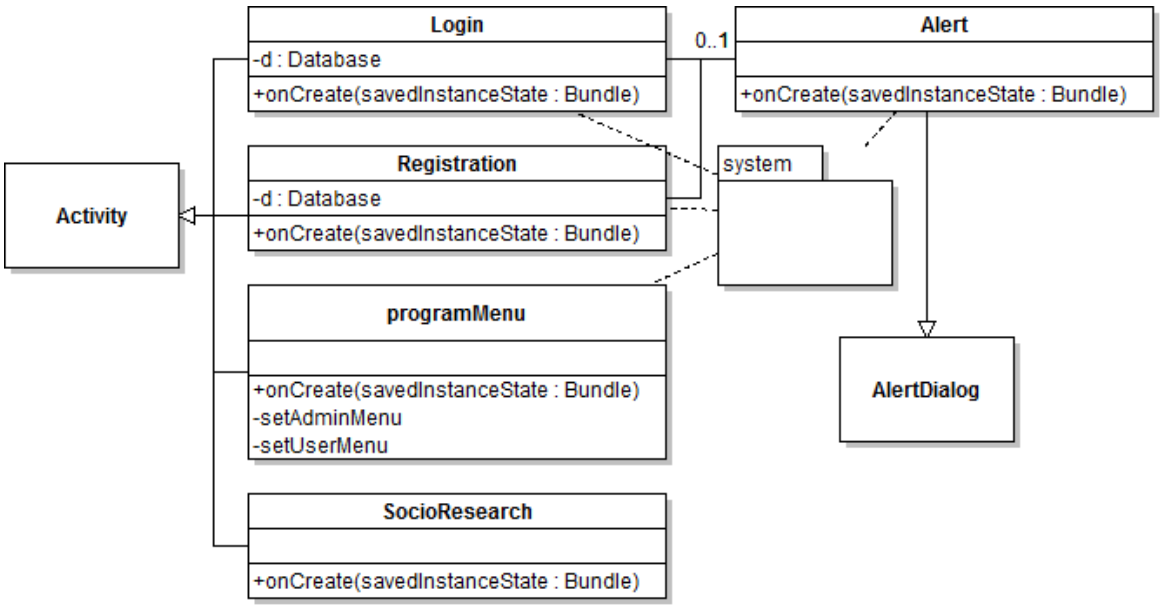
1 – je administrátor

-1 – přihlášení se nezdařilo

Strukturu všech výše uvedených tříd můžeme vidět na obr. 7.

### 4.1.5 Správa uživatelů

Ve správě uživatelů potřebujeme mít přehled o všech uživatelích – musíme nějakým způsobem zobrazit jejich seznam. K tomu bude využita komponenta rozšiřující třídu *ListActivity*. Z databáze bude načten seznam všech uživatelů a v této komponentě zobrazen. Pro načtení potřebných dat poslouží metoda *fillData*. Tato data budou načtena pomocí nové metody ve třídě *Database – fetchAllUsers*.




**Obr. 7 – Třídní diagram zobrazující třídy potřebné pro registraci, přihlášení uživatele a zobrazení uživatelského menu**

#### *4.1.5.1 Vytvoření nového uživatele*

Komponenta, umožňující vytvoření uživatele musí obsahovat textové vstupy pro všechny vlastnosti uživatele – uživatelské jméno, heslo a id skupiny, do které patří. Informace o tom, zda je uživatel administrátor, bude zobrazena pomocí políčka, které lze zaškrtnout. Po potvrzení údajů tuto komponentu již potřebovat nebudeme – opět se automaticky zobrazí seznam uživatelů. Pro tvorbu uživatele v databázi bude využita již definovaná metoda *Database.createUser*.

#### *4.1.5.2 Editace uživatele*

Pro editaci uživatele bude použita stejná komponenta jako pro jeho tvorbu – při jejím spuštění se zde ale již budou nacházet vyplněny všechny jeho vlastnosti, načtené z databáze pomocí metody f*illData*. Ty budeme moci editovat. Stejně jako u tvorby uživatele, ani zde ji již po potvrzení upravených údajů nepotřebujeme – bude tedy opět zobrazen seznam uživatelů.

Pro načtení údajů o uživateli, které budeme moci upravovat, poslouží nová metoda třídy *Database – fetchUser.*

Protože uživatele rozlišujeme pomocí id (kvůli komponentě *ListActivity*), ale při zobrazení seznamu uživatelů – jejich přihlašovacích jmen – musí být uživatelé unikátní, musíme vždy zkontrolovat, zda nově vytvářený (nebo editovaný) uživatel s daným jménem již neexistuje. Pro tuto kontrolu vytvoříme metodu *Database.userExist*. K podobnému problému dochází i u skupin uživatelů – při editaci uživatele musíme vidět, v jaké skupině se nachází, ale samotné id skupiny nám nic neřekne. Proto musíme pomocí tohoto id zjistit její název – k tomu nám poslouží

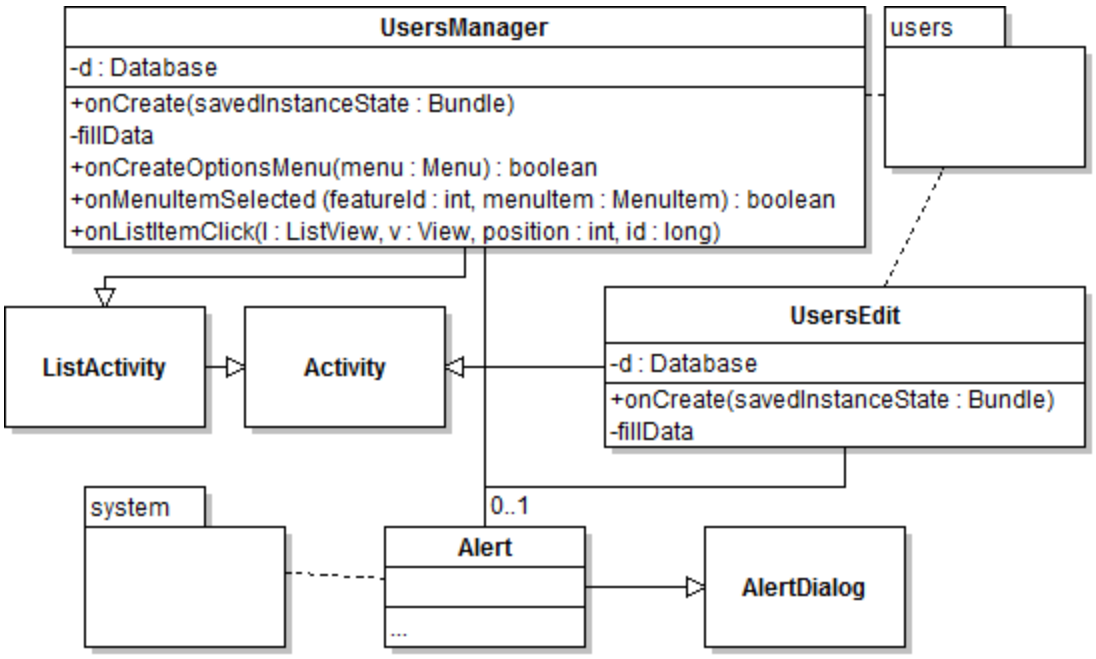
nová metoda *Database.getUserGroup*. Při potvrzování nového (nebo upraveného) nastavení také musíme zkontrolovat, zda daná skupina existuje (nemůžeme vložit uživatele do neexistující skupiny) – pro tuto kontrolu opět vytvoříme novou metodu – *Database.groupExist*. Nakonec nesmíme zapomenout na operaci, která upraví uživatele v databázi – ta bude reprezentována metodou *Database.updateUser*.

#### *4.1.5.3 Smazání uživatele*

Smazání uživatele bude provedeno pomocí speciální položky v menu. Po jejím výběru bude smazán z databáze. Tuto funkci bude obstarávat komponenta, zobrazující seznam uživatelů – po smazání potřebujeme opět zobrazit aktualizovaný seznam uživatelů. Smazání uživatele z databáze bude provedeno pomocí metody *Database.deleteUser*.

Nyní musíme tyto části spojit do funkčního celku. Základ tvoří seznam uživatelů. Zde budou použity všechny tři výše uvedené funkce. Přístup k prvním dvěma bude vytvořen pomocí menu komponenty, určené pro zobrazení seznamu uživatelů. K editaci vybraného uživatele se zase dostaneme pomocí události kliknutí na vybraného uživatele.

Ve výsledku tedy získáme dvě komponenty, které budeme používat. Jedna sloužící pro zobrazení všech uživatelů (*UsersManager*) a druhá pro jejich tvorbu a editaci (*UsersEdit*). Celou strukturu můžeme vidět na obr. 8.




**Obr. 8 – Třídní diagram správy uživatelů mobilní aplikace**

### 4.1.6 Dotazníky

#### *4.1.6.1 Správa dotazníků*

Stejně jako u správy uživatelů, i zde musíme mít přehled o všech dotaznících, nacházejících se na zařízení. K tomu účelu bude i zde použita komponenta rozšiřující třídu *ListActivity***.** Stejné jsou zde i operace, které potřebujeme nad dotazníky provádět. I zde bude pro přístup k operacím tvorby a smazání dotazníku využito menu komponenty a editace bude prováděna kliknutím na požadovaný formulář.

#### *4.1.6.2 Tvorba dotazníku*

U nového dotazníku musíme definovat jeho název, úvod, jednotlivé otázky a jejich možnosti. U každého dotazníku se může lišit počet otázek a jejich voleb, co ale mají všechny stejné, jsou vlastnosti název a úvod. Tyto budou spravovány v samostatné komponentě, podobné té, která je použita pro tvorbu a editaci uživatele. Zde ale budeme pracovat pouze s dvěma již uvedenými vlastnostmi. V menu komponenty se budou nacházet dvě možnosti – uložení aktuálního formuláře a zobrazení otázek formuláře.

Dále potřebujeme vytvořit otázky formuláře. Zde musíme mít možnost zadat text otázky a možnost vybrat její typ. Text otázky bude zadáván do klasického textového pole a výběr typu bude proveden pomocí tří uživatelských prvků typu *Radio Button*, reprezentujících jednotlivé typy.

Pro tvorbu možnosti dané otázky bude sloužit další komponenta – zde budeme zadávat text možnosti (její prefix a postfix) a pomocí zaškrtávacího políčka určovat, zda bude tato možnost obsahovat textový vstup, nebo ne.

Uživatel by měl mít možnost libovolného pohybu v rámci celého vytvářeného/editovaného dotazníku. Potřebujeme tedy zajistit pohyby vpřed/vzad mezi jednotlivými otázkami a stejně tak v rámci možností samotné otázky. K tomu nám poslouží menu jednotlivých komponent. U komponenty pro tvorbu/editaci otázky se musí v menu nacházet položky pro přesun na předchozí/další otázku, vytvoření nové otázky, smazání aktuální otázky, přesun na možnosti aktuální otázky a samozřejmě také uložení celého formuláře

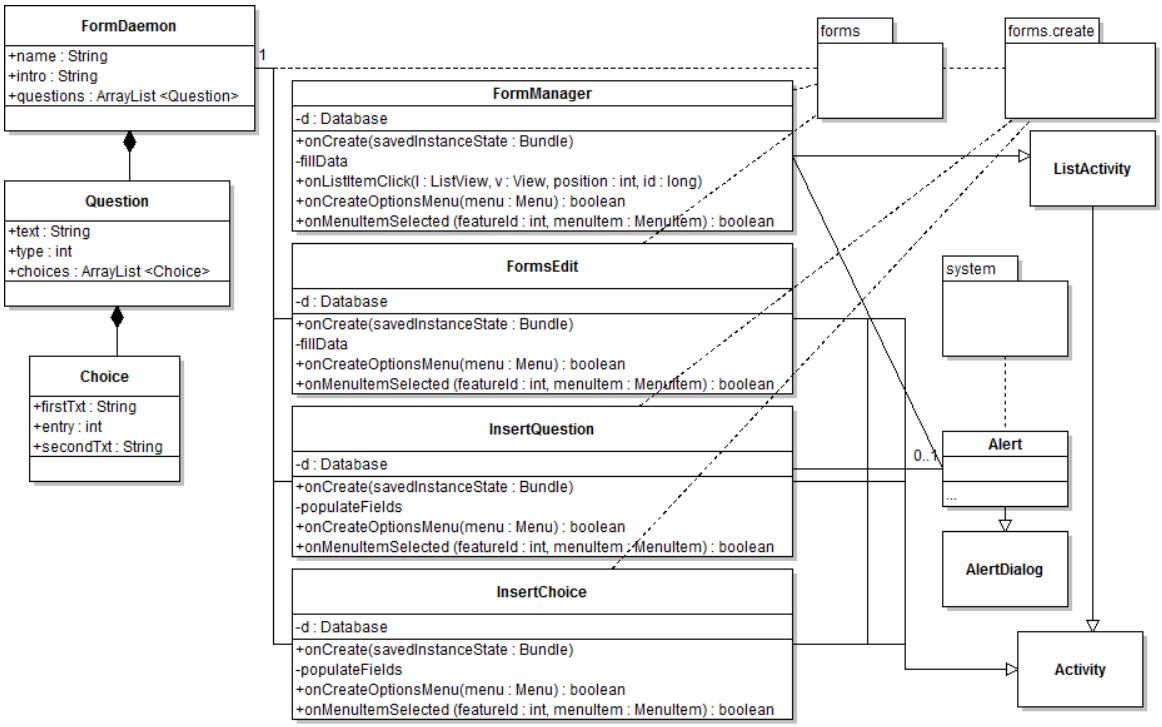
Strukturu dotazníku bude reprezentovat samostatná třída. Ta bude obsahovat název dotazníku, úvod a seznam otázek. Ty budou reprezentovány třídou *Question*. Zde se bude nacházet text otázky, její typ a seznam možností. Ty budou také reprezentovány samostatnou třídou – *Choice*. V ní se zase bude nacházet informace o tom, zda možnost obsahuje vkládání textu a samotný text možnosti – její prefix a postfix (viz Obr.).

Ve výsledku tedy budeme mít jednu třídu pro zobrazení seznamu dotazníků (*FormsManager*), jednu pro editaci názvu a úvodu dotazníku (*FormsEdit*), další třídu pro tvorbu/editaci otázky (*InsertQuestion*), pak jednu pro tvorbu/editaci možnosti otázky (*InsertChoice*) a nakonec třídy, reprezentující celý formulář (*FormDaemon*, *Question* a *Choice*). Celou strukturu můžeme vidět na obr. 9.

Metody *populateFileds*, nacházející se v komponentách *InsretQuestion* a *InsertChoice* jsou určeny pro případné načtení textů formuláře, který editujeme. Metody *fillData* komponent *FormsEdit* a *FormsManager* poslouží zase k načtení potřebných dat z databáze.

Pro načtení všech dotazníků poslouží metoda *Database.fetchAllForms*, pro načtení jednoho *Database.fetchForm a* pro smazání *Database.deleteForm.* Pro kontrolu, zda již existuje dotazník se zadaným názvem (obdobně jako u uživatele) poslouží *Database.formExist* a nakonec pro samotné vytvoření dotazníku bude určena metoda *Database.createForm*. Také zde nesmíme zapomenout na aktualizaci upraveného dotazníku (metoda *Database.updateForm*).

O operace, které souvisejí s obsahem dotazníku (s jeho otázkami a možnostmi) se postará třída *FormDaemon*. Výsledné třídy můžeme vidět níže (obr. 9).




**Obr. 9 – Třídní diagram tvorby a správy dotazníků**

### 4.1.7 Skupiny uživatelů

Stejně tak jako v předchozích případech, i zde bude pro zobrazení seznamu (v tomto případě skupin, nacházejících se na daném zařízení) použita komponenta rozšiřující třídu *ListActivity* (třída *GroupsManager*). Funkce vytváření, editace a mazání skupiny budou také implementovány obdobným způsobem jako v předchozích případech.

V komponentě, určené pro editaci/vytváření skupiny (*GroupsEdit*) musí být umístěn textový vstup pro název skupiny. V rámci správy skupin musíme mít také přístup k uživatelům a formulářům upravované skupiny.

I ve výše uvedených komponentách budou metody *fillData* sloužit pro načtení požadovaných dat.

U jednotlivých seznamů musíme mít možnost odstranit požadovanou položku a ze seznamu volných uživatelů/formulářů zase vybrat tu, kterou chceme do dané skupiny vložit.

Pro zobrazení položek využijeme již vytvořené komponenty – zobrazení uživatelů obstará komponenta *UsersManager* a formulářů skupiny *FormsManager*. Do těchto komponent budou doimplementovány další stavy:

U komponenty *UsersManager*:

- Seznam uživatelů skupiny – zde bude místo seznamu všech uživatelů načten seznam uživatelů skupiny. Kliknutím na vybraného uživatele bude daný uživatel ze skupiny smazán a vložen do skupiny výchozí. Bude také upraveno menu – bude v něm obsažena pouze jedna položka – Vložení uživatele do skupiny. Při výběru této položky bude opět spuštěna komponenta *UsersManager*, ale nacházející se v následujícím stavu.

- Seznam všech volných uživatelů – zde budou zobrazeni uživatelé, kteří se momentálně nachází ve výchozí skupině. Bude zde použita pouze jediná funkce a to výběr a vložení potřebného uživatele do aktuální skupiny. K operaci dojde kliknutím na vybraného uživatele. Poté bude automaticky zobrazen seznam uživatelů skupiny s již nově vloženým uživatelem.

U komponenty *FormsManager* budou implementovány v podstatě stejné stavy, jako u komponenty *UsersManager*, ale budeme pracovat s formuláři:

- Seznam formulářů skupiny – bude načten seznam formulářů skupiny a stejně jako u předešlé komponenty bude pomocí kliknutí na vybraný formulář daný formulář smazán. V menu se bude také nacházet jediná položka – Vložení formuláře do skupiny. Po jejím výběru bude zobrazena aktuální komponenta v následujícím stavu.

- Seznam všech volných formulářů – protože se formulář může, na rozdíl od uživatele, nacházet ve více skupinách, budou zde zobrazeny všechny formuláře zařízení. Funkčnost tohoto stavu bude obdobná jako u stavu komponenty *UsersManager* – Seznam všech volných uživatelů. Po výběru požadovaného formuláře bude daný formulář vložen do aktuální skupiny a komponenta se automaticky vrátí do předchozího stavu a zobrazí seznam s aktuálně vloženým formulářem.

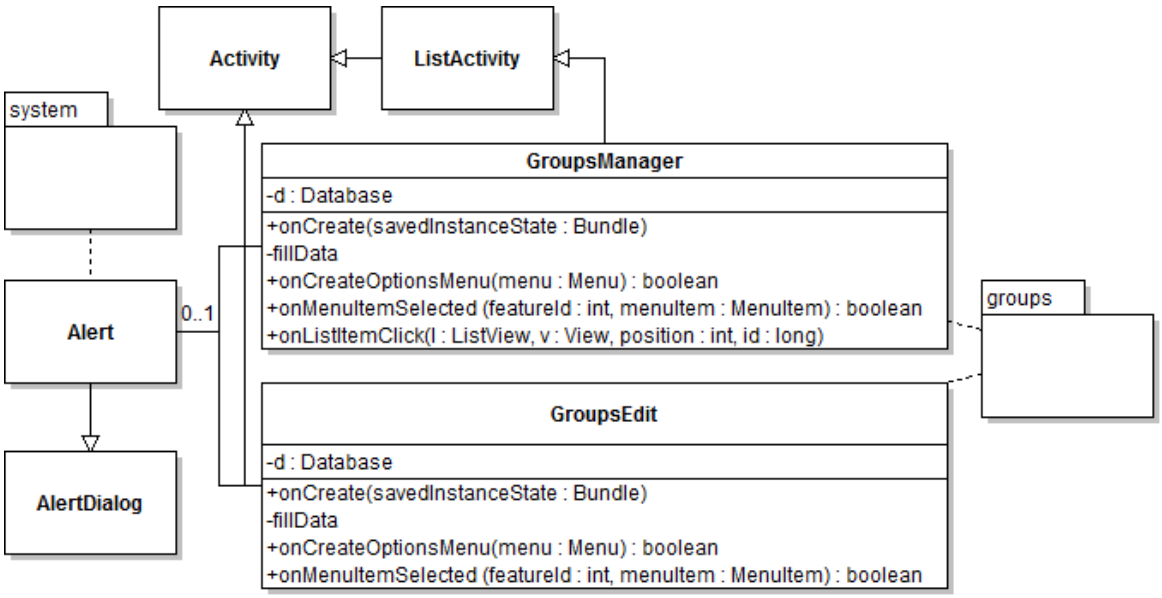
Nakonec nám zbývá definovat potřebné operace nad daty. I zde budeme muset získat seznam všech skupin (*Database.fetchAllGroups*), skupinu vytvářet (*Database.createGroup*), mazat (*Database.deleteGroup*), upravovat (*Database.updateGroup*) a získat skupinu pro editaci (*Database.fetchGroup*).

Pro ošetření uživatelů, kteří se skupinou souvisejí, vytvoříme metodu *Database.updateUserGroup*. Ta nám poslouží pro smazání uživatele ze skupiny (uživatel bude přemístěn do výchozí) a také při upravování názvu skupiny (změnu názvu budeme muset provést u všech uživatelů dané skupiny).

Dále potřebujeme načíst uživatele dané skupiny (metoda *Database.fetchUsersOfGroup* – ta bude také využita pro zobrazení uživatelů ve výchozí skupině) a její formuláře (metoda *Database.fetchFormsOfGroup*).

Ještě musíme ošetřit práci se vztahem mezi formulářem a skupinou, reprezentovaný pomocí tabulky *Use*. Zde potřebujeme vkládat formulář do skupiny (*Database.insertFormToGroup*)a smazat jej z ní (*Database.deleteFormFromGroup*). Při smazání formuláře také nesmíme zapomenout smazat všechny vazby na ni – k tomu využijeme metodu *Database.deleteFormFromUse*. To samé musíme provést u smazání skupiny (metoda *Database.deleteGroupFromUse*).

Třídy, potřebné pro správu skupin můžeme vidět na obr. 10.




**Obr. 10 – Třídní diagram správy skupin**

### 4.1.8 Vyplňování dotazníků

Nyní musíme navrhnout samotné vyplňování dotazníků. Každý uživatel musí mít možnost výběru formuláře, který chce vyplnit. Dostupné formuláře jsou dány členstvím daného uživatele v určité skupině. Potřebujeme tedy zobrazit seznam formulářů skupiny, ve které se tazatel nachází. K tomu opět využijeme komponentu *FormsManager* (viz Správa dotazníků) a vytvoříme v ní nový stav:

Výběr formuláře dané skupiny – tento stav se nijak zásadně neliší od stavu *Seznam formulářů skupiny*, ale na rozdíl od něj zde budeme pomocí kliknutí na vybranou položku daný formulář otevírat.

Po výběru požadovaného dotazníku potřebujeme zobrazit jeho úvod – k tomu nám poslouží nová komponenta *OpenForm*. Zde bude zobrazen název dotazníku, jeho úvod a pomocí tlačítka se bude tazatel moci dostat na otázky formuláře.

O zobrazení jedné otázky a její vyplnění se postará další komponenta – *FillForm*. Zde bude zobrazen text otázky a podle jejího typu zobrazeny jednotlivé možnosti pomocí odpovídajících ovládacích prvků uživatelského rozhraní a také případné vstupy pro vkládání textu. Na další otázku se dostaneme opět pomocí tlačítka.

Nyní potřebujeme při přechodech mezi jednotlivými otázkami vyplněná data nějakým způsobem uchovávat. Protože známe počet otázek a množství jejich možností, ale neznáme přesně typ dat, která budou vkládána, pro jejich dočasné uložení využijeme dvojrozměrné pole typu *String*, kde bude jeho délka odpovídat počtu otázek a délky jednotlivých položek tohoto pole počtu možností odpovídající otázky.

Ještě musíme rozlišit to, zda je na dané pozici pole vyplněn výběr (je pouze zaškrtnuto nějaké políčko), nebo byl do případného textového vstupu vybrané možnosti zadán nějaký text. To provedeme pomocí speciálního řetězce, který bude určovat to, že došlo pouze k výběru možnosti. Zadávání tohoto řetězce musí být při vkládání případného textu zakázáno.

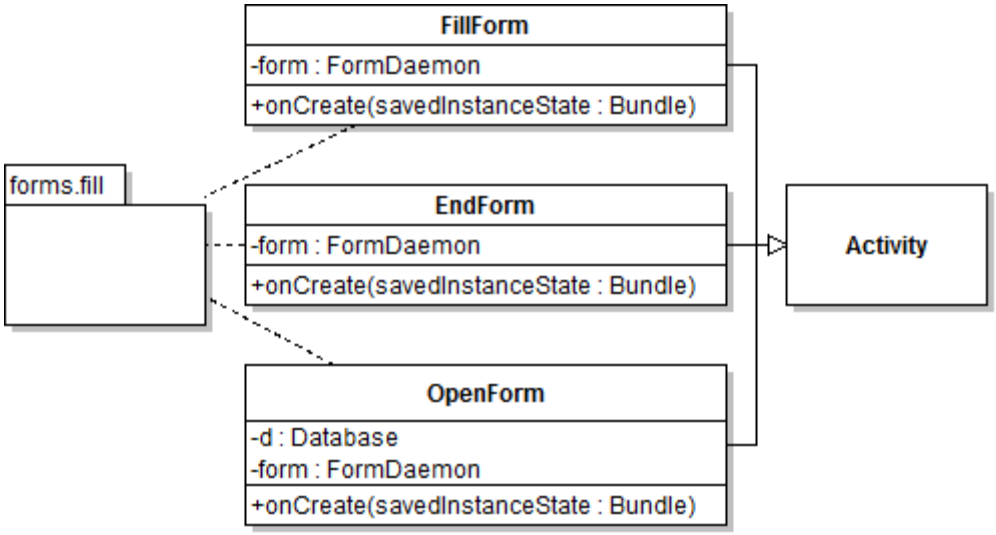
Po vyplnění všech otázek se dostaneme na konec formuláře. Ten bude reprezentován další komponentou – *EndForm*. Zde bude mít tazatel možnost na výběr, zda daný formulář uloží nebo ne, a zda bude chtít začít vyplňovat další dotazník, nebo bude chtít přejít zpět do hlavního menu, reprezentovaného komponentou *programMenu*.

Všechna potřebná data, jako texty otázek a možností budou získány pomocí objektu typu *FormDaemon* reprezentujícího formulář.

Při potvrzení uložení vyplněného dotazníku musí být provedena transformace získaných dat do formy, která nám je umožní trvale uchovat. Jak již bylo uvedeno výše, sesbíraná data budou uchovávána v samostatné databázi reprezentované třídou *Answers*, kde bude pro každý dotazník vytvořena nová tabulka.

Nejprve musíme zkontrolovat, zda tabulka daného dotazníku existuje (*Answers.tableExist*). Pokud neexistuje, musíme ji vytvořit (*Answers.createTable*). Pokud už existuje, data do ní musíme nějakým způsobem vložit (*Answers.insertRow*). Ještě nám zbývá vytvořit operaci, která nám pomůže zjistit, kolik dotazníků v ten daný den uživatel vyplnil. Tato operace bude obsažena v metodě *Answers.todayFilled*.

Na obr. 11 můžeme vidět strukturu tříd, které budeme potřebovat při vyplňování dotazníků.




**Obr. 11 – Třídní diagram vyplňování dotazníků**

## 4.2 Server

Server bude naimplementován pomocí technologie *J2EE*. Bude použit framework *Struts 2* a objektově relační mapování zajistí framework *Hibernate*.

Framework *Struts 2* implementuje části *View* a *Controller* softwarové architektury *MVC*. Pro část *View* poskytuje hlavně vlastní knihovnu uživatelských značek, část *Controller* je zase reprezentována servletem (třída *org.apache.struts.action.ActionServlet*). Ten po příchodu uživatelského požadavku tento požadavek podle požadované akce přesměruje na odpovídající třídu (rozšiřující *org.apache.struts.action*), která obsahuje samotnou aplikační logiku dané akce.

Při návrhu implementace serveru tedy musíme definovat potřebné akce, jejich umístění v odpovídajících třídách a pro zobrazení výsledku (vrstva *View*) odpovídající soubory *JSP*.

Nejnižší vrstvu architektury *MVC – Model* - framework *Struts 2* neimplementuje. Pro práci s ní tedy využijeme výše uvedený framework *Hibernate*. Ten bude reprezentovat datovou vrstvu – musíme definovat třídy reprezentující perzistentní data serveru. Operace nad těmito daty budou definovány v pomocných třídách. Díky nim dojde k oddělení aplikační logiky od datové vrstvy.

### 4.2.1 Autentizace

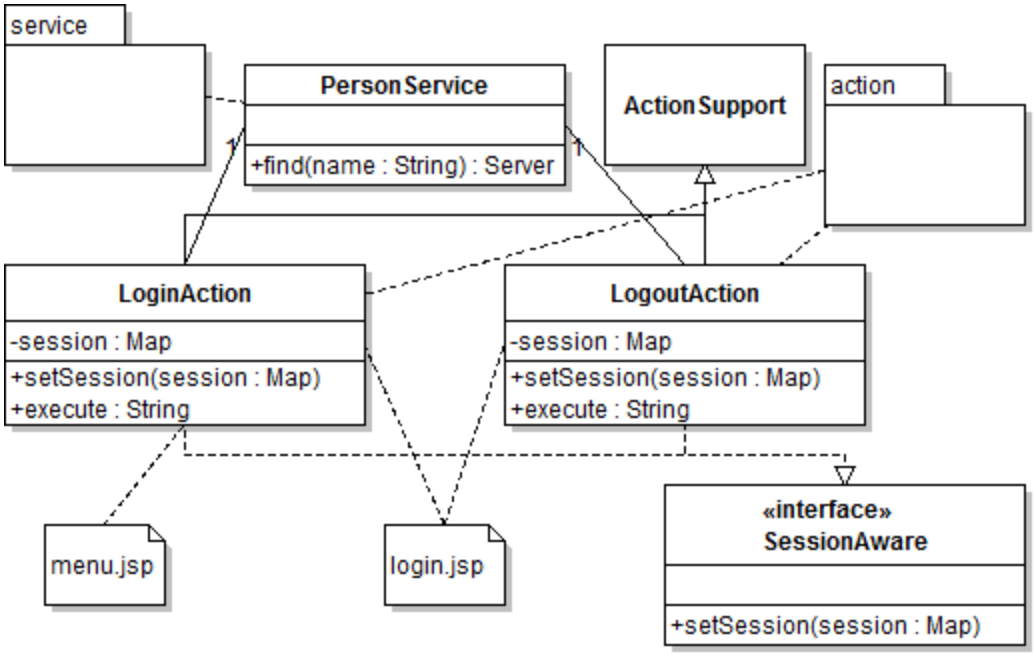
Autentizace bude prováděna pomocí uživatelského jména a hesla – potřebujeme tedy akci serveru, která bude tyto informace přijímat a která zkontroluje, zda jsou správné a zjistí, o jakého uživatele se jedná (zda o administrátora, nebo běžného uživatele). Podle toho bude zobrazeno patřičné menu, obsahující přístup k požadovaným funkcím.

Pro přihlášení bude sloužit akce *login*, která se bude nacházet ve třídě *LoginAction*. Informace o tom, že byl uživatel autentizován a o tom, že se jedná o administrátora, budeme uchovávat v sezení.

Nějakým způsobem také musíme přihlášeného uživatele odhlásit – k tomu poslouží akce *logout*. Pro ni vytvoříme samostatnou třídu *LogoutAction*. Zde budou informace vložené při akci *login* ze sezení odstraněny.

Nyní musíme definovat třídu, ve které se budou nacházet metody obsluhující operace nad uživateli. Tuto třídu nazveme *PersonService*. Do ní vložíme metodu, potřebnou pro načtení uživatele, jehož přihlašovací údaje ověřujeme – *find*.

Pro zobrazení přihlašovacího formuláře budeme potřebovat jediný JSP soubor – *login.jsp*. Všechny potřebné třídy můžeme vidět na obr. 12.




**Obr. 12 – Třídní diagram přihlášení a odhlášení uživatele na serveru**

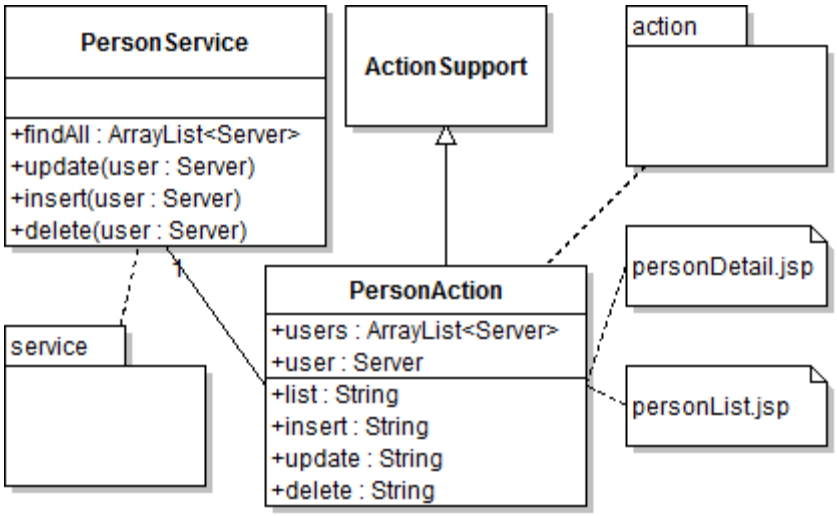
### 4.2.2 Správa uživatelů

K této funkci bude mít z menu přístup pouze administrátor. Zde potřebujeme mít přehled o všech uživatelích serveru – proto budou všichni zobrazeni ve formě seznamu (akce *list*). Při vytvoření uživatele (akce *insert*) musíme mít možnost zadat všechny potřebné údaje – k tomu využijeme formulář s potřebnými textovými vstupy. Tento formulář také využijeme při editaci (akce *update*) – zde budou navíc do textových vstupů načteny údaje uživatele, které budeme moci upravovat. Nakonec musíme uživatele nějakým způsobem smazat – k tomu poslouží akce *delete*. Po jejím vykonání bude zobrazen aktualizovaný seznam uživatelů.

Přístup k akcím *delete* a *update* bude umístěn přímo v seznamu - na řádku daného uživatele (vždy budeme pracovat s konkrétním uživatelem). Přístup k akci *insert* bude umístěn mimo seznam. Protože budou tyto akce pracovat se stejnými daty, umístíme je do společné třídy (*PersonAction*).

Do třídy, určené pro práci s uživateli (*PersonService*) přidáme metody pro zobrazení všech uživatelů (*findAll)*, vložení uživatele (*insert*), smazání uživatele (*delete*) a pro aktualizaci uživatele (*update*). Pro načtení uživatele, kterého chceme editovat, využijeme již existující metodu *find*.

Pro zobrazení potřebných dat uživateli vytvoříme dva nové JSP soubory. První z nich – *personList.jsp* - bude zobrazovat seznam všech uživatelů, druhý – *personDetail.jsp* - zase vstupní pole, které budou umožňovat vytváření a editaci uživatele. Třídy, potřebné pro správu uživatelů serveru můžeme vidět na obrázku níže (Obr. 13).




**Obr. 13 – třídní diagram správy uživatelů serveru**

### 4.2.3 Správa zařízení

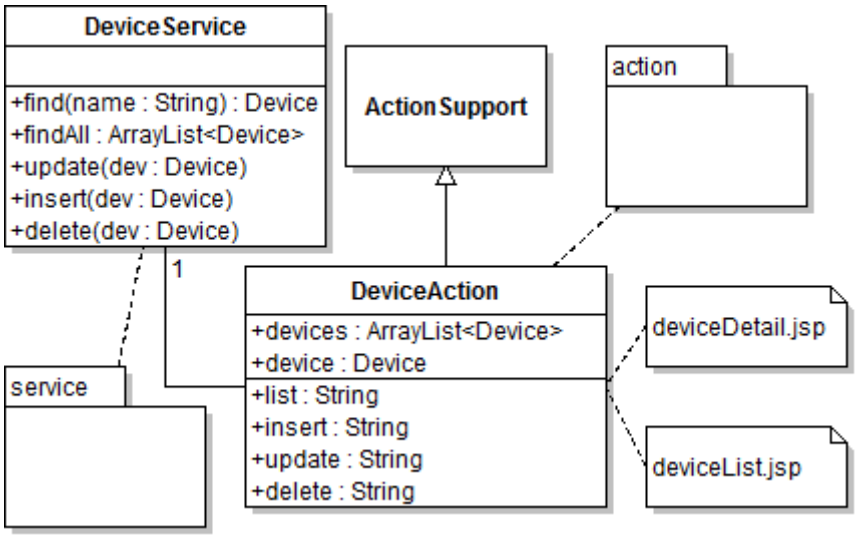
Správa zařízení bude založena na stejném principu jako správa uživatelů. Zde také potřebujeme zobrazit seznam zaregistrovaných zařízení (akce *list*). Na každém řádku bude přístup k akcím pro smazání (akce *delete*) a editaci zařízení (akce *edit*). Akce pro vytvoření zařízení (*insert*) bude umístěna mimo seznam. Všechny tyto akce umístíme do společné třídy – *DeviceAction* (budou pracovat se stejnými daty). Při smazání požadovaného zařízení musí být také smazána všechna související data – zálohované nastavení zařízení společně se sesbíranými daty.

Pro práci se zařízeními vytvoříme novou třídu – *DeviceService*. Ta bude obsahovat v podstatě stejné metody jako třída *PersonService*, které ale budou pracovat s daty zařízení – *findAll*, *insert*, *delete*, *find* a *update*.

Pro zobrazení dat budou využity dva JSP soubory – jeden pro zobrazení seznamu zařízení (*deviceList.jsp*) a další umožňující tvorbu/editaci zařízení (*deviceDetail.jsp*). Celkovou strukturu tříd můžeme vidět na obr. 14.

#### *4.2.3.1 Prohlížení nastavení zařízení*

U formuláře editace zařízení bude navíc umožněn přístup k možnosti prohlížení nastavení zařízení. Zde budeme moci prohlížet (stejně jako na zařízení) všechny uživatele (akce *users_list*), všechny formuláře (akce *forms_list*) a všechny skupiny (akce *groups_list*). Tato data budou vždy zobrazena jako seznam.




**Obr. 14 – Třídní diagram správy zařízení**

Protože tyto akce pokaždé pracují s jinými daty a vždy potřebujeme zobrazit jejich seznam, pokaždé ale s jinými objekty (které mají různé vlastnosti), umístíme tedy každou akci do samostatné třídy:

- *users_list* – třída *UsersAction*
- *groups_list* – třída *GroupsAction*
- *forms_list* – třída *FormsAction*

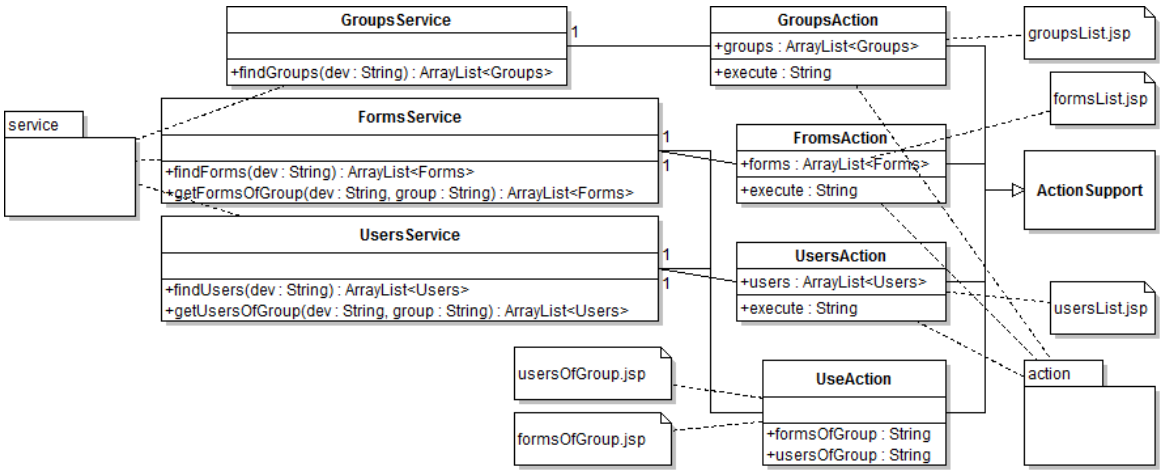
Nyní ještě potřebujeme zobrazit uživatele a formuláře určité skupiny (akce *usersOfGroup* a *formsOfGroup*). Přistupovat k těmto akcím budeme ze seznamu skupin (zobrazeného pomocí

akce *groups_list*), z řádku dané skupiny (potřebujeme zobrazit uživatele, nebo formuláře určité skupiny). Tyto akce umístíme do třídy *UseAction*.

Každá výše uvedená akce pracuje s jedním typem dat nastavení zařízení. Proto pro každý tento typ vytvoříme novou třídu, obsahující operace nad těmito daty. Pro uživatele *UsersService*, pro skupiny *GroupsService* a pro formuláře *FormsService*. Třída *UseAction* využije všechny tři třídy.

Pro zobrazení dat, nacházejících se na zařízení využijeme několik samostatných JSP souborů. *UsersList.jsp* poslouží pro zobrazení seznamu uživatelů zařízení, *formsList.jsp* pro zobrazení všech formulářů zařízení a *groupsList.jsp* bude zase zobrazovat všechny skupiny. Další dva využijeme pro zobrazení obsahu jedné skupiny. *FormsOfGroup.jsp* pro zobrazení formulářů skupiny a *usersOfGroup.jsp* pro zobrazení všech uživatelů skupiny.

Strukturu všech potřebných tříd můžeme vidět na diagramu níže:




**Obr. 15 – Třídní diagram prohlížení nastavení**

## 4.3 Spojení

### 4.3.1 Nastavení spojení

#### *4.3.1.1 Mobilní aplikace*

Nejprve musíme určit, kam se budeme připojovat. K identifikaci serveru, se kterým budeme komunikovat, nám stačí znát jeho IP adresu. U tohoto serveru se také musíme nějakým způsobem identifikovat – k tomu použijeme už výše uvedený název zařízení a heslo.

K nastavení bude sloužit speciální komponenta - *ConnectionSettings*. Zde se budou nacházet textové vstupy pro výše uvedené informace. Tyto informace také potřebujeme nějak trvale uložit – k tomu ale nevyužijeme databázi (bylo by zbytečné kvůli třem informacím vytvářet samostatnou tabulku). Tato data budou uložena pomocí funkce *Shared Preferences*, která nám

umožňuje dlouhodobě uchovávat malá množství informací, které mohou využívat i ostatní komponenty aplikace.

#### *4.3.1.2 Server*

U serveru se všechny potřebné informace nachází v informacích o jednotlivých zařízeních – potřebujeme znát pouze název zařízení a heslo, abychom mohli ověřit přihlašované zařízení.

### 4.3.2 Autentizace zařízení

#### *4.3.2.1 Mobilní aplikace*

U zařízení musíme vyřešit, jakým způsobem se budeme na server přihlašovat. Na server budeme odesílat přihlašovací údaje zařízení - název a heslo. Poté musíme počkat na odpověď serveru – zda se přihlášení podařilo, nebo ne. Protože budeme přenášet pouze přihlašovací údaje, využijeme požadavek GET.

#### *4.3.2.2 Server*

Na straně serveru využijeme již vytvořené přihlašování uživatelů. Musíme ale rozlišit, zda se právě přihlašuje uživatel, nebo zařízení. Zde bude použit speciální příznak – pokud bude u požadavku nalezen, server bude vědět, že se jedná o zařízení, provede ověření tohoto zařízení a odešle na něj potřebnou odpověď. Proto musíme na straně mobilního zařízení přidat do požadavku informaci o tom, že se jedná o zařízení.

### 4.3.3 Odeslání nastavení

#### *4.3.3.1 Mobilní aplikace*

Zde využijeme předchozí funkci – přihlášení na server. Dále už budeme odesílat pouze samotná data. Každá tabulka bude odeslána v samostatném požadavku typu POST. Protože musíme mít na serveru pevně stanovený počet parametrů, obsah jednotlivých tabulek rozdělíme podle jejich sloupců – každý parametr POST požadavku bude reprezentovat jeden sloupec. Zde musíme nějakým způsobem rozlišit data jednotlivých řádků – to bude provedeno pomocí znaku „|” - přenášet totiž budeme pouze data, ve kterých se bude tento znak vyskytovat velmi zřídka. Navíc tedy ještě budeme muset doimplementovat u jednotlivých textových vstupů aplikace kontrolu, zda nebyl tento znak zadán. Nakonec nesmíme zapomenout na přenos souborů formulářů – nejjednodušší způsob, jaký lze použít je serializace objektu formuláře, jeho odeslání a jeho následná deserializace na straně serveru.

#### *4.3.3.2 Server*

Na straně serveru musíme vytvořit novou akci, určenou pro příjem nastavení z mobilního zařízení. Zde budeme potřebovat název zařízení, které své nastavení zálohuje na server – k tomu využijeme sezení – během přihlášení do něj vložíme potřebný název. Protože ze zařízení v jednom požadavku obdržíme vždy jednu tabulku nastavení, musíme udržovat informace o tom, které tabulky jsme již obdrželi – to z důvodu, že může během přenosu dojít k chybě a obdržíme tedy jen část nastavení. Proto také nebudeme ihned data ukládat na server. K samotnému uložení dojde až po jeho úspěšném přijetí.

Protože budeme na straně serveru přijímat serializované objekty formulářů, musí být i zde umístěny identické třídy reprezentující formulář – *FormDaemon*, *Question* a *Choice*.

### 4.3.4 Příjem nastavení

#### *4.3.4.1 Mobilní aplikace - požadavek*

I zde bude opět využita funkce Autentizace zařízení. Server musíme nějakým způsobem požádat o potřebná data – k tomu nám poslouží GET požadavek. Požadovat budeme vždy jednotlivé tabulky a soubory formulářů.

#### *4.3.4.1 Server – reakce na požadavek*

Pro reakci na požadavek odeslání nastavení na zařízení vytvoříme samostatnou akci – ta podle požadovaného názvu tabulky načte potřebná data a odešle je na zařízení. Data budou odeslána ve formě jeden řádek tabulky na jeden řádek odpovědi. Jednotlivé atributy budou opět odděleny pomocí znaku „|“.

#### *4.3.4.2 Mobilní aplikace – příjem dat*

Data budeme přijímat tak, jak nám je server odeslal – jeden řádek tabulky na jednom řádku odpovědi. Z něj získáme jednotlivé atributy a vložíme je do odpovídající tabulky.

Kvůli možnosti nečekaného výpadku spojení musíme ale staré nastavení zálohovat – před samotnými požadavky o nové tabulky bude celé načteno do paměti. Nové nastavení bude nejprve načítáno do paměti. Pokud během tohoto načítání dojde k chybě, nestane se nic, pouze odstraníme zálohu a částečně obdržená data z paměti. Pokud se podaří celé nastavení úspěšně získat, dojde ke smazání starého z databáze a k pokusu o uložení nového. Pokud dojde během ukládání k chybě, bude částečně uložené nové nastavení z databáze smazáno a obnoveno původní zálohované.

Po úspěšném přenosu musí také dojít ke smazání tabulek aktuálně sesbíraných dat – ty totiž nemusí odpovídat nově načteným souborům formulářů.

Obdobným způsobem musíme také zálohovat existující soubory formulářů - načteme je jako objekty do paměti.

### 4.3.5 Odeslání sesbíraných dat na server

#### *4.3.5.1 Uchovávání sesbíraných dat na straně serveru*

Na straně mobilního zařízení se nachází data uložená v samostatných tabulkách – pro každý formulář jedna. Na straně serveru, kde bude použito objektově relační mapování *Hibernate*, nelze ale tak jednoduše vytvářet nové tabulky. Musíme proto vytvořit způsob, jakým budou tato data na straně serveru uchovávána. Potřebujeme tedy vytvořit strukturu, která bude reprezentovat různé tabulky formulářů, obsahující různý počet otázek a možností. Proto se musíme zaměřit na vlastnosti, které mají společné – název zařízení, kde byl dotazník vyplněn, název samotného dotazníku, id tazatele a datum a čas vyplnění.

Tyto společné vlastnosti bude obsahovat nový objekt, který nazveme *data*. Zde se budou nacházet všechny vyplněné dotazníky všech druhů formulářů.

Nyní musíme vytvořit pevně danou strukturu pro data vyplněných otázek. U vyplněné otázky známe to, ke kterému formuláři náleží, její id, pak id možnosti otázky, která byla vybrána a případně text, který byl u možnosti vyplněn. Z těchto vlastností můžeme získat strukturu, která bude obsahovat následující atributy:

- *id dotazníku*
- *id otázky*
- *id možnosti*
- *případný vyplněný text*

U každé otázky dokážeme rozlišit, zda obsahuje textový vstup – proto výše uvedenou strukturu rozdělíme do dvou samostatných. První z nich bude obsahovat všechny výše uvedené vlastnosti, druhá bude ochuzena o případný vyplněný text. Tímto se nám podařilo snížit objem dat, který bude nutné projít při zpracování dat. Pokud bychom totiž vše nechali v jedné struktuře, operace zpracování dat by musela procházet v podstatě všechny odpovědi všech formulářů.

Protože lze u každé vyplněné otázky navíc určit její typ, můžeme pomocí něj dvě výše uvedené struktury rozdělit na pět následujících části:

Části, které budou obsahovat vlastnost, kde bude uchováván text:

- *text_varchar* – bude obsahovat odpovědi na otázky, které obsahují pouze textový vstup
- *radio_varchar* – bude obsahovat odpovědi na otázky, u kterých lze vybrat pouze jednu možnost, a které obsahují textový vstup
- *check_varchar* – zde budou umístěny otázky, u kterých můžeme vybrat více možností, obsahující také textový vstup

Části, které nebudou obsahovat vlastnost pro uchovávání textu:

- *radio_int* – bude uchovávat pouze výběry možností u vyplněných otázek, u kterých lze vybrat pouze jednu možnost.
- *check_int* – bude uchovávat výběry možností otázek, u kterých lze vybrat více možností

Pokud tedy budeme chtít uložit sesbíraná data formuláře, který obsahuje dvě otázky s možností jediného výběru a jednu, obsahující pouze textové vstupy, vytvoříme nový objekt typu *data*, reprezentující jeden vyplněný dotazník a poté projdeme jednotlivé otázky. Podle typu otázky pak vyplněná data uložíme do odpovídající struktury – data prvních dvou otázek do struktury *radio_int* a data poslední otázky do *text_varchar*. Do vlastnosti id dotazníku vždy umístíme odkaz na odpovídající objekt typu *data (*propojující všechny vyplněné otázky jednoho vyplněného dotazníku), dále id vyplněné otázky a id vybrané možnosti a nakonec text odpovědi, pokud byl nějaký vyplněn.

#### *4.3.5.2 Mobilní aplikace*

Nyní nám zbývá přenos sesbíraných dat. Každý formulář na zařízení má svoji vlastní tabulku – opět musíme tyto data nějakým způsobem transformovat na přesně stanovený počet parametrů POST požadavku. Transformace bude provedena přesně na míru strukturám jednotlivých typů otázek nacházejících se na serveru. Budeme tedy muset postupně načítat sloupce tabulky, reprezentující jednotlivé možnosti celého formuláře, zjišťovat, jakého je typu otázka, pod kterou tato možnost patří a podle toho rozhodnout, do které výsledné struktury (reprezentované parametry požadavku POST) budou data vložena.

Protože budeme s velkou pravděpodobností přenášet velké množství dat, přenos obsahu jedné tabulky formuláře raději rozdělíme do několika požadavků typu POST. Každý tento požadavek bude reprezentovat strukturu jednoho typu otázek. Také nesmíme zapomenout odeslat dotazníky, ke kterým náleží odesílaná data – na serveru se totiž ještě nemusí nacházet a neměli bychom tedy jak ze sesbíraných dat získávat informace – neznali bychom totiž jejich kontext.

#### *4.3.5.3 Server*

Pro příjem sesbíraných dat bude sloužit nová akce serveru. Zde budou (podobně jako při příjmu nastavení) přijímány tabulky (ve formě parametrů požadavku POST), reprezentující jednotlivé typy otázek. Opět budeme muset kontrolovat, zda jsme získali všechny „tabulky". Teprve po úspěšném obdržení dojde k jejich uložení do databáze.

### 4.3.6 Shrnutí

#### *4.3.6.1 Mobilní aplikace*

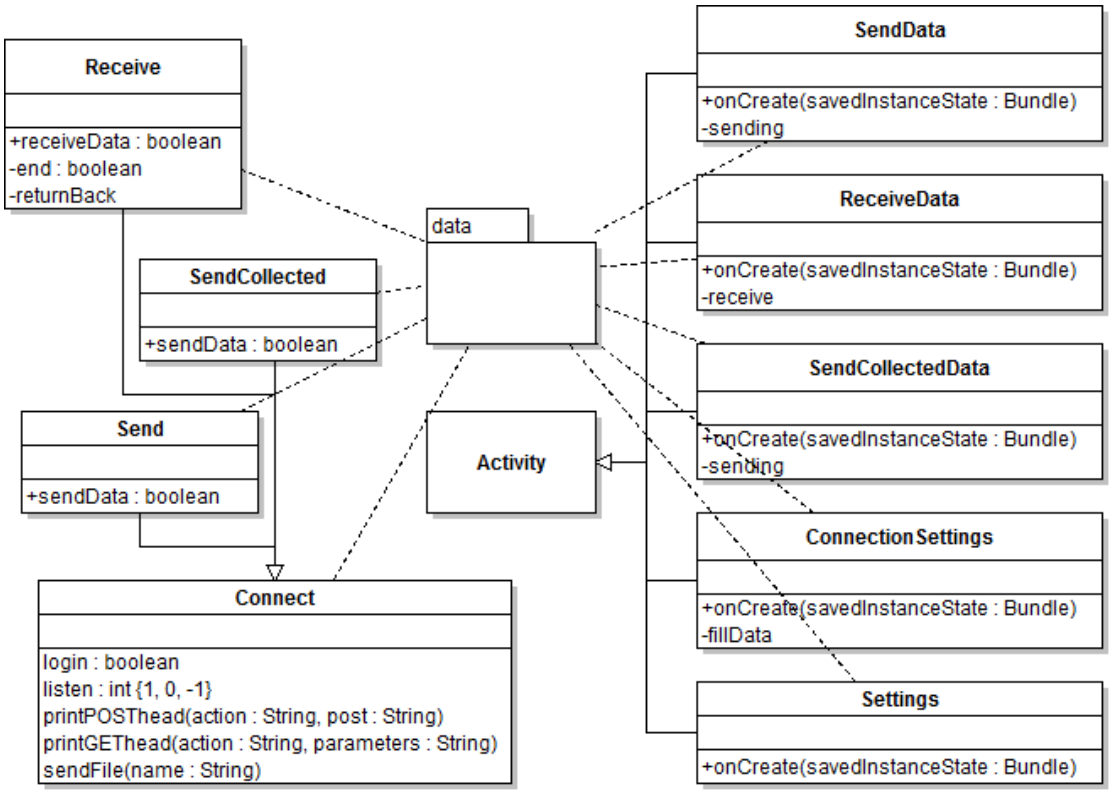
Ve výsledku tedy budeme muset vytvořit komponentu určenou pro nastavení spojení (*ConnectionSettings*), dále komponentu pro odeslání nastavení na server (*SendData*), pro příjem nastavení (*ReceiveData*) a nakonec komponentu pro odeslání sesbíraných dat (*SendCollectedData*). Protože budou funkce přihlášení na server, odesílání požadavků, naslouchání odpovědi a odesílání souborů formulářů použity ve více komponentách, umístíme je do speciální třídy *Connect*. Tři posledně uvedené komponenty budeme využívat pro informování uživatele o průběhu spojení. Samotnou obsluhu spojení umístíme do samostatných tříd (*Send*, *Receive*, *SendCollected*). Tyto třídy budou dědit ze třídy *Connect* – budeme tak moci přistupovat ke společným funkcím přímo, nebudeme muset kvůli nim vytvářet speciální objekt.

Přístup k funkci Odeslání sesbíraných dat musí mít oba typy uživatelů – bude proto přístupná přímo z hlavního menu (komponenta *programMenu*). Ostatní funkce, jako nastavení spojení, odeslání a příjem nastavení umístíme do speciálního menu – nové komponenty *Settings*. Ta bude přístupná pouze administrátorovi.

Metody *sending* a *receive* komponent *SendData*, *ReceiveData* a *SendCollectedData* budou sloužit k obsluze samotného procesu přenosu dat – buď jejich odeslání, nebo příjmu. Metoda *fillData*, stejně jako u jiných komponent, také v komponentě *ConnectionSettings* poslouží k načtení potřebných dat do UI. Metody *sendData* ve třídách *Send* a *SendCollected* poslouží k odeslání potřebných dat na server. Funkčnost třídy *Receive* bude rozdělena do následujících metod – metoda *receiveData* přijme potřebné nastavení ze serveru, v metodě *end* dojde k záloze původního nastavení a pokusu o uložení nového. Pokud se ukládání nezdaří, bude zavolána metoda *returnBack*, která se postará o obnovení nového nastavení. Ve třídě *Connect* budou zase obsaženy metody pro přihlášení (*login*), naslouchání odpovědi serveru (*listen*), pro odeslání souboru (*sendFile*) a pro odeslání potřebného požadavku (*printPOSThead* a *printGEThead*).

Operace nad uloženými daty zde budou opět ošetřeny ve třídách *Database* a *Answers*. Zde budeme potřebovat načíst všechna potřebná data všech tabulek. Zde využijeme již existující metody (*Database.fetchAllUsers, Database.fetchAllGroups* a *Database.fetchAllForms*). Chybí nám ale operace pro načtení celé tabulky *Use* – proto vytvoříme novou metodu *Database.getTableGroupUse*. Po úspěšném přijetí nastavení musíme nějakým způsobem smazat původní – k tomu bude určena metoda *Database.deleteTab* (vždy smaže požadovanou tabulku).

Také budeme muset vytvořit nové operace nad sesbíranými daty. Zde musíme získat obsah každé tabulky formuláře – k tomu nám poslouží nová metoda *Answers.fetchAllRows*. Po úspěšném odeslání tabulky formuláře tato data smažeme pomocí metody *Answers.deleteTab*. Třídy komponent a pomocné třídy, potřebné pro spojení se serverem můžeme vidět na Chyba: zdroj odkazu nenalezen.




**Obr. 16 - Diagram tříd ošetřujících spojení a se serverem a přenos dat**

### *4.3.6.2 Server*

Na serveru budeme muset vytvořit následující akce:

- *sendToServer* – sloužící k příjmu nastavení ze zařízení
- *receiveFromServer* – sloužící k odeslání požadovaného nastavení na zařízení
- *sendCollectedData* – sloužící pro příjem sesbíraných dat

Zbývá nám jedna společná funkce výše uvedených akcí – příjem a odeslání objektů formulářů. Jelikož se budou objekty formulářů nacházet vždy jeden v jednom požadavku, tuto funkci umístíme do dvou nezávislých akcí – *saveFile* a *loadFile*.

Akce *sendToServer* a *receiveFromServer* budou umístěny do stejné třídy - *SettingsAction*. Pracují totiž se stejnými daty - jedna akce je přijímá a druhá odesílá.

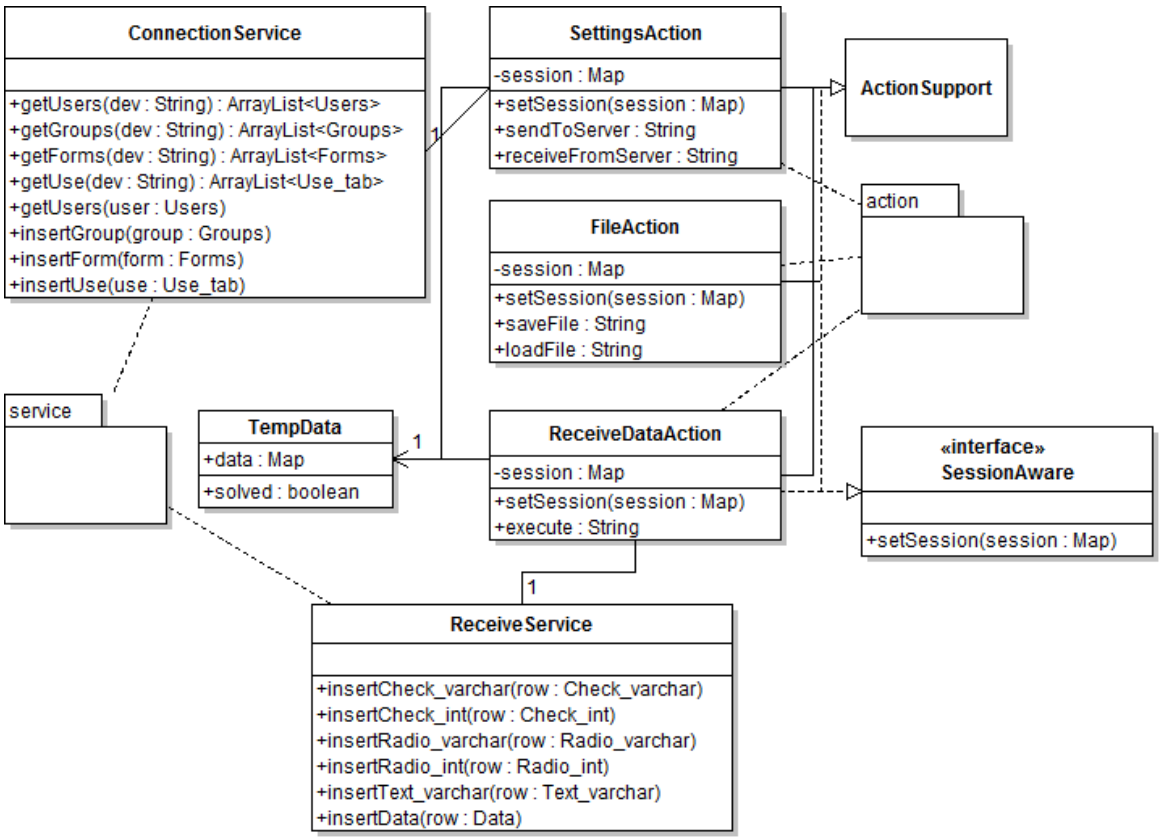
Akce pro práci s objekty formulářů budou také umístěny do samostatné třídy – *FileAction*. Zbývající akci (*sendCollectedData*) umístíme do třídy *ReceiveDataAction*.

Všechny výše uvedené třídy musí implementovat rozhraní *SessionAware* – budou totiž pracovat se sezením.

V akcích *sendToServer* a *sendCollectedData* potřebujeme ukládat částečně přijatá data (nastavení, sesbíraná data) a kontrolovat, zda již bylo přijato vše. K tomuto účelu vytvoříme třídu *TempData*. Data budou ukládána do objektu typu *Map* – k obsahu se budeme dostávat pomocí jeho názvu (názvu dočasně uložené přijaté tabulky) a kontrolovat „vyplněnost“ budeme pomocí metody *solved*.

Pro práci s formuláři zde budeme potřebovat třídy, identické s těmi na zařízení – *FormDaemon*, *Question* a *Choice*.

Operace, prováděné nad daty obstarají nové třídy – *ReceiveService* a *ConnectionService*. První bude mít na starosti operace potřebné pro třídu *ReceiveDataAction* (metody určené pro ukládání dat jednotlivých typů otázek), druhá zase operace, potřebné pro třídu *SettingsAction* (metody pro výběr dat nastavení - *getUsers*, *getGroups*, *getForms* – a pro jejich vkládání - *insertUser*, *insertGroup* a *insertForm*). Strukturu potřebných tříd můžeme vidět na třídním diagramu na Obr. 17.




**Obr. 17 – Diagram tříd na straně serveru, které mají na starosti spojení a přenos dat**

## 4.4 Data na straně serveru

Nyní vytvoříme celkový souhrn dat, se kterými budeme na serveru pracovat.

### 4.4.1 Uživatelé serveru

Data o uživatelích serveru budou uchovávány v objektech typu *Server*. Budou se zde nacházet přihlašovací údaje uživatelů – přihlašovací jméno a heslo, kontaktní email, a informace o tom, zda je uživatel administrátor.

### 4.4.2 Zařízení

Informace o zařízeních, která se starají o samotný sběr dat, budou obsaženy v objektech třídy *Devices*. Budou zde uloženy přihlašovací údaje daného zařízení – název zařízení a heslo.

### 4.4.3 Uživatelé jednotlivých zařízení

Uživatelé všech zařízení budou umístěni v objektech třídy *Users*. Zde budou uloženy všechny informace o uživatelích, tak jak jsou uloženy na jednotlivých zařízeních, doplněné o informaci o tom, ke kterému zařízení náleží – jeho název.

### 4.4.4 Skupiny uživatelů zařízení

Skupiny uživatelů všech zařízení budou zase umístěny v objektech typu *Groups*. Stejně jako u uživatelů zařízení zde bude navíc uvedena informace o zařízení, ze kterého pocházejí.

### 4.4.5 Formuláře jednotlivých zařízení

V objektech třídy *Forms* budou obsaženy formuláře daných zařízení. Také zde musí být uvedeno, z jakého zařízení pocházejí.

### 4.4.6 Vztahy mezi skupinami a formuláři zařízení

Vazební tabulky *Use* jednotlivých zařízení budou uchovávány v objektech třídy *Use_tab*. Zde také musíme uložit název zařízení, ke kterému tyto informace náleží.

### 4.4.7 Sesbíraná data

Data vyplněných dotazníků budou uložena podle způsobu uvedeného v části 4.3.5.1 Uchovávání sesbíraných dat na straně serveru. Uvedené datové struktury budou uchovány v odpovídajících třídách – *Data*, *Check_int*, *Check_varchar*, *Radio_int*, *Radio_varchar*, a *Text_varchar*.

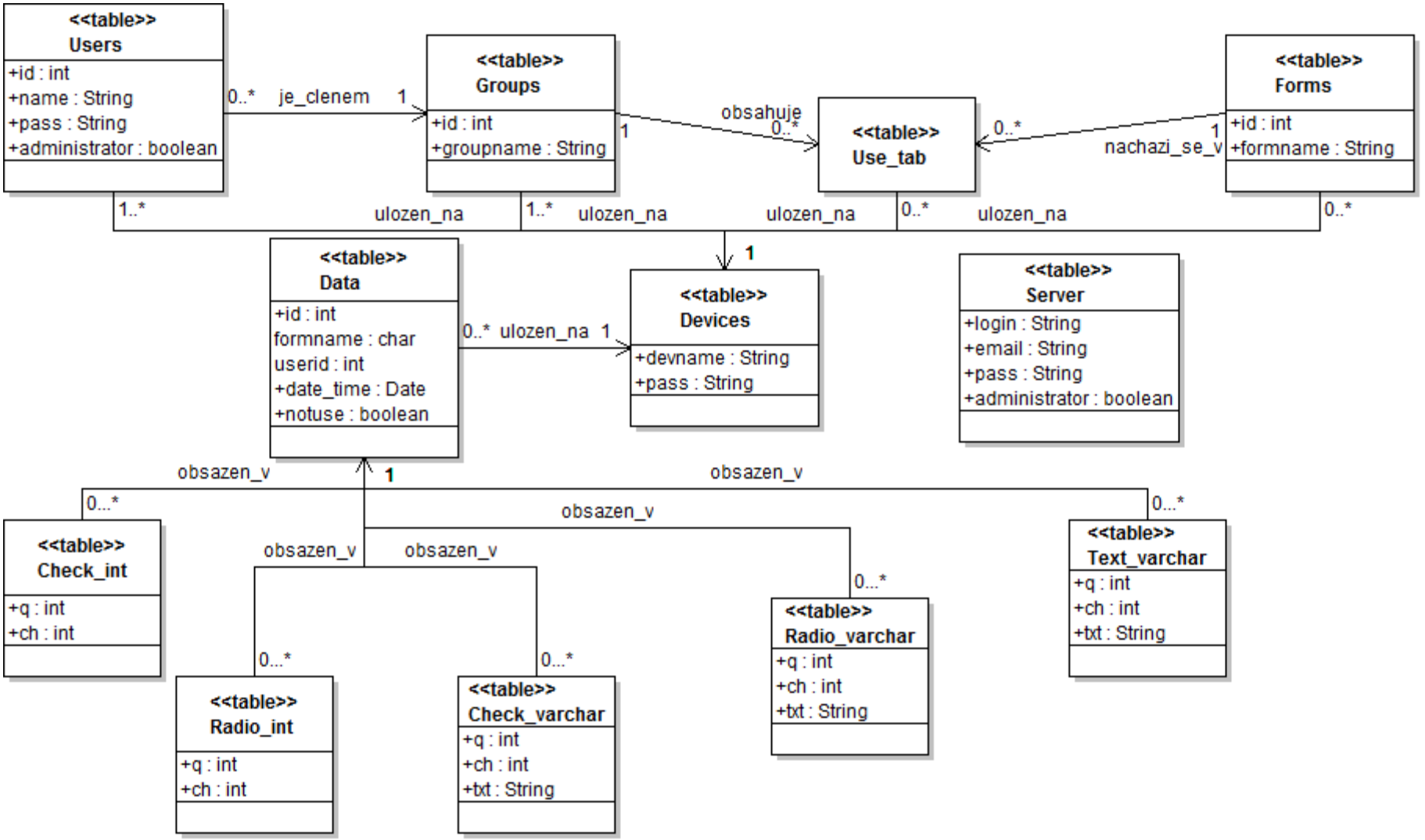
Výslednou strukturu všech potřebných tříd můžeme vidět na obr. 18.

### 4.4.8 Lineární zápis entit

Protože budou výše uvedená data ukládána v databázi, musíme definovat strukturu jednotlivých tabulek:

**primární klíč,** *cizí klíč*, *cizí i primární klíč*

Server {**login**, email, pass, administrator}
Devices {**name**, pass}
Users {*devname*, **_id**, name, pass, admin, *groupid*}
Groups {*name*, **_id**, groupname}
Use_tab {*name*, *form_id*, *group_id*}
Forms {*name*, **_id**, formname}
Data {**id**, formname, *userid*, *devname*, date_time, notuse}
Check_varchar {*id*, **q**, ch, txt}
Text_varchar {*id*, **q**, ch, txt}
Radio_varchar {*id*, **q**, ch, txt}
Check_int {*id*, **q**, ch}
Radio_int {*id*, **q**, ch}




**Obr. 18 – Třídní diagram dat, se kterými bude pracovat server**

### 4.4.9 Datový slovník

| Název atributu | Datový typ | Rozsah | Klíč | Null | Popis |
|---|---|---|---|---|---|
| **Server** | | | | | |
| **login** | char | 32 | A | N | Přihlašovací jméno uživatele |
| pass | char | 32 | N | N | Heslo |
| email | char | 256 | N | A | |
| administrator | bool | 1 | N | N | Indikuje, zda je daný uživatel administrátor, nebo ne |
| **Devices** | | | | | |
| **devname** | char | 64 | A | N | Název zařízení |
| pass | char | 64 | N | N | Heslo zařízení |
| **Users** | | | | | |
| *devname* | char | 64 | A | N | Název zařízení |
| **Id** | int | 4 | A | N | Id uživatele daného zařízení |
| Name | char | 64 | N | N | Přihlašovací jméno uživatele daného zařízení |
| Pass | char | 128 | N | N | Heslo uživatele |
| *groupid* | int | 4 | N | N | Id skupiny, do které uživatel náleží |
| admin | int | 1 | N | N | Indikuje, zda je daný uživatel administrátor, nebo ne |
| **Groups** | | | | | |
| *devname* | char | 64 | A | N | Název zařízení |
| **id** | int | 4 | A | N | Id skupiny daného zařízení |
| groupname | char | 64 | N | N | Název skupiny |
| **Use_tab** | | | | | |
| *devname* | char | 64 | A | N | Název zařízení |
| *form_id* | int | 4 | A | N | Id formuláře daného zařízení |
| *group_id* | int | 4 | A | N | Id skupiny daného zařízení |
| **Forms** | | | | | |
| *devname* | char | 64 | A | N | Název zařízení |
| **id** | int | 4 | A | N | Id formuláře daného zařízení |
| formname | char | 64 | N | N | Název formuláře |
| **Data** | | | | | |
| **id** | int | 9 | A | N | Id vyplněného dotazníku |
| formname | char | 64 | N | N | Název vyplněného formuláře |
| *userid* | int | 4 | N | N | Id uživatele, který formulář vyplnil |
| devname | char | 64 | N | N | Název zařízení |

| | | | | | |
|---|---|---|---|---|---|
| date_time | datetime | 12 | N | N | Datum a čas vyplnění (Ve formátu rrmmddhhmmss) |
| notuse | bool | 1 | N | N | Indikuje, zda má být daný formulář použit při zpracování dat, nebo ne. Výchozí hodnota je false |
| **Check_varchar** | | | | | |
| *id* | int | 9 | A | N | Id formuláře daného zařízení, ke kterému vyplněná otázka patří |
| **q** | int | 4 | A | N | Id vyplněné otázky |
| **ch** | int | 4 | A | N | Id vybrané možnosti |
| txt | char | 256 | N | A | Případný text zapsaný do možnosti |
| **Radio_varchar** | | | | | |
| Viz Check_varchar | | | | | |
| **Text_varchar** | | | | | |
| Viz Check_varchar | | | | | |
| **Check_int** | | | | | |
| *id* | int | 9 | A | N | Id formuláře daného zařízení, ke kterému vyplněná otázka patří |
| **Q** | int | 4 | A | N | Id vyplněné otázky |
| **ch** | int | 4 | A | N | Id vybrané možnosti |
| **Radio_int** | | | | | |
| Viz Check_int | | | | | |

## 4.5 Správa sesbíraných dat

Ve správě sesbíraných dat potřebujeme určovat, která data budeme a která nebudeme při zpracování dat používat. Budeme spravovat data různých formulářů, které budou mít různou strukturu. Máme však k dispozici několik vlastností, které mají všechny společné. Jedná se o název zařízení, id uživatele, který na daném zařízení dotazník vyplnil, datum a čas vyplnění a název formuláře, který byl vyplněn. Pomocí těchto vlastností musíme vytvořit systém, který nám umožní určit co nejmenší množinu dat, kterou chceme smazat, nebo u které chceme určit, zda bude při zpracování použita, nebo ne.

Protože existuje závislost mezi zařízením a id tazatele (uživatelé na různých zařízeních mohou mít stejné id), musíme určit nejprve zařízení, jehož sesbíraná data chceme spravovat. Budeme také chtít spravovat data jednoho určitého formuláře – nebudeme chtít například smazat data, která byla sesbírána v určité časové periodě na jednom zařízení, ale která se týkají různých nesouvisejících formulářů. Proto musí být po výběru zařízení ještě navíc přidána možnost výběru formuláře (nacházející se na vybraném zařízení). Tímto jsme zmenšili množinu dat, určenou pro správu. Toto budou první dvě akce správy dat.

Nyní přejdeme k další akci. Protože už známe zařízení, můžeme jednoznačně určit uživatele, který sesbíral data vybraného formuláře a která chceme spravovat. Uživatele budeme vybírat ze seznamu uživatelů daného zařízení – můžeme ho vybrat, nebo nemusíme. Dále musí tato akce obsahovat možnost zadat určitou časovou periodu – ve výsledku budeme moci díky těmto akcím určit například množinu dat určitého formuláře, která byla sesbírána určitým uživatelem na určitém zařízení v zadané časové periodě (například jeden den) a ty následně smazat.

Nyní nám zbývá definovat akce pro smazání dat a pro určení, zda budou vybraná data použita, nebo ne. Přístup k těmto akcím bude umožněn až v akci, kde již známe množinu dat, určenou podle vybraného zařízení a formuláře. V této nové akci ji můžeme ještě dále upřesnit pomocí jména uživatele a určení časové periody. Data budou upravena/smazána podle aktuálně zadaných podmínek.

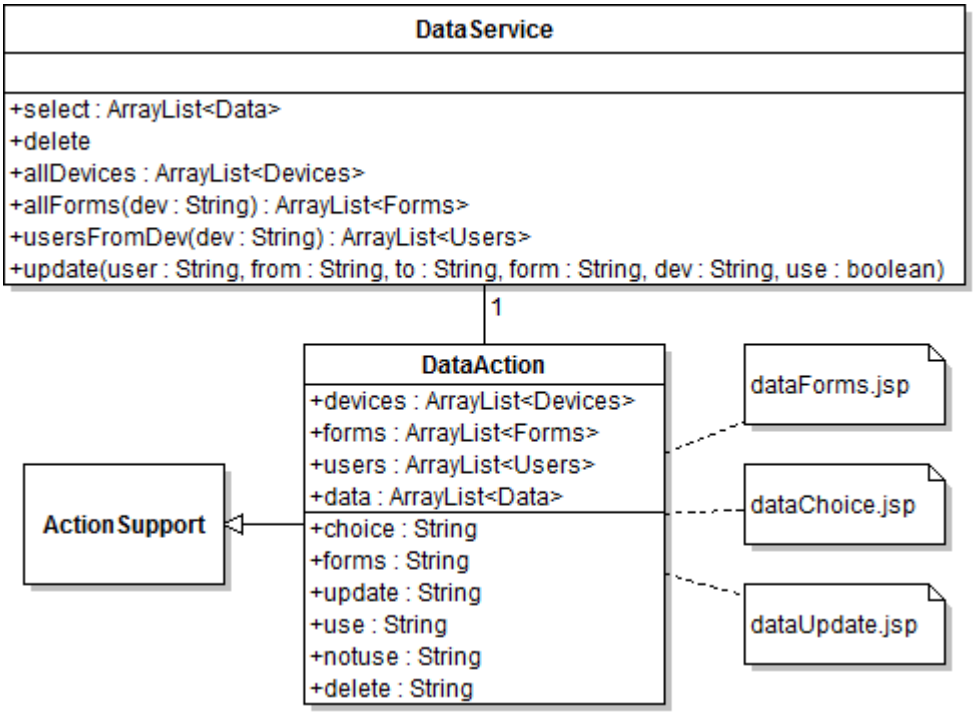
Všechny tyto akce umístíme do samostatné třídy *DataAction*. Bude se zde nacházet akce pro výběr zařízení – *choice*, pro výběr formuláře – *forms*, dále akce pro upřesnění výběru – *update* a nakonec akce, určené pro úpravu dat:

- *use* – vybraná množina dat bude při zpracování použita
- *notuse* – vybraná množina dat nebude při zpracování použita
- *delete* - vybraná množina dat bude smazána

Pro operace, potřebné při správě dat bude vytvořena nová třída – *DataService*. Metoda *update* bude sloužit k úpravě dat (nastavení toho, zda budou nebo nebudou při zpracování dat použity), metoda *select* pro výběr potřebné množiny a metoda *delete* pro smazání vybrané množiny dat. U akce *forms* potřebujeme získat seznam formulářů zařízení (k tomu nám poslouží metoda *allForms*), u akce *choice* seznam zařízení (metoda *allDevices*) a u akce *update* seznam všech uživatelů zařízení – metoda *usersFromDev*. Všechny odpovídající třídy můžeme vidět na obr. 19.

Pro zobrazení uživatelských rozhraní výše definovaných akcí využijeme následující JSP soubory:

- *dataChoice.jsp* – bude umožňovat výběr požadovaného zařízení (akce *choice*).
- *dataForms.jsp* – bude určen pro výběr určitého formuláře daného zařízení (akce *forms*)
- *dataUpdate.jsp* – bude obsahovat formulář, sloužící pro upřesnění výběru a úpravu vybrané množiny dat (akce *update*, *delete*, *use* a *notuse*).




**Obr. 19 – Třídní diagram správy sesbíraných dat**

## 4.6 Zpracování dat

Zpracování sesbíraných dat bude probíhat podobným způsobem jako jejich sběr – uživatel projde celý formulář a během tohoto procházení zadá potřebné výběry a podmínky.

Při procházení bude vždy zobrazena celá otázka a její možnosti. U každé z nich musí být možnost pro zadání výběru dat, které nás zajímají a zadání podmínky, které musí výsledek splňovat – k tomu poslouží dva textové vstupy. Data, zadaná v těchto vstupech budou ukládána do sezení pro pozdější zpracování.

Protože je zde nejmenším fragmentem dat jedna možnost, při výběru bude stačit, pokud budeme pouze zadávat číslo požadované možnosti, které je u ní uvedeno.

Ve vstupu pro zadávání podmínky zase využijeme syntaxe jazyka *SQL*, nacházející se po klíčovém slově *Where*. Podmínka bude zadávána pro právě zobrazenou otázku. V otázce můžeme mít například dotaz na věk a v možnostech bude pouze jediná, obsahující textový vstup – víme tedy, že se v sesbíraných datech pro tuto možnost nachází čísla reprezentující věk dotazovaných. Zde tedy můžeme potřebovat zadat určitý věkový rozsah – k tomu využijeme jazyk *SQL*.

Na konci máme tedy čísla jednotlivých možností otázek, pro která nás zajímá výsledek a potřebné podmínky, které musí výsledek splňovat. Pomocí těchto informací nyní musíme zpracovat data, nacházející se na serveru.

Pokud by se data jednoho formuláře nacházela v jediné tabulce (jako na zařízení), stačilo by vždy vybrat ty dotazníky, u nichž byla vybrána možnost, která nás ve výsledku zajímá, a poté tyto získané řádky „ořezat“ podle zadaných podmínek. Na serveru se ale nachází několik pevně

daných tabulek, ve kterých jsou obsaženy informace z různých formulářů – musíme tedy určit způsob, jakým za těchto podmínek získáme požadované informace.

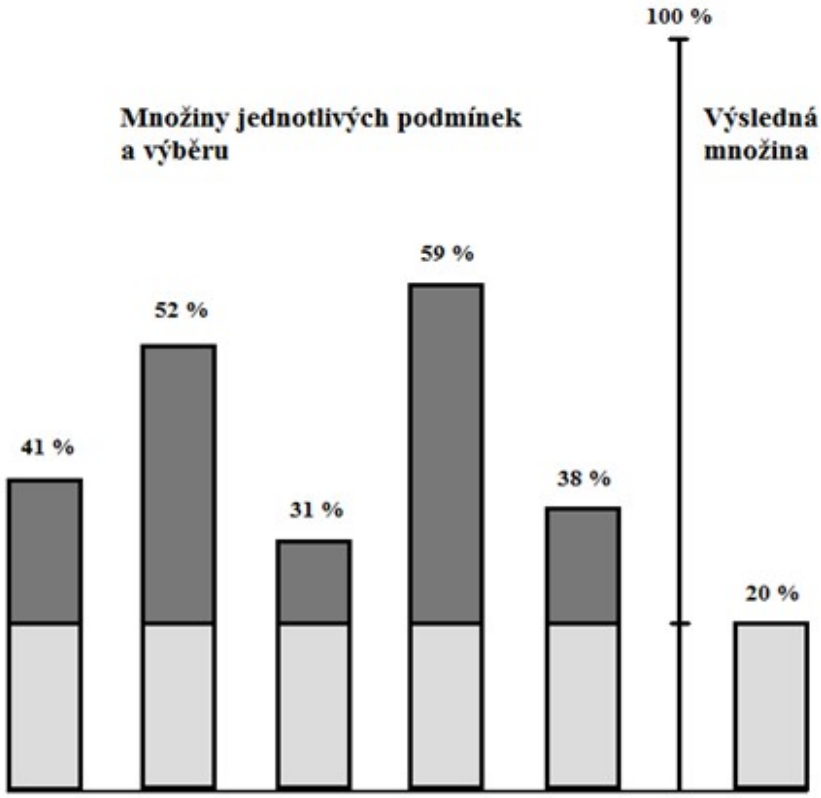
Každá z těchto tabulek odpovídá jednomu typu otázky, a protože samotné výběry a zadávání podmínek vždy probíhají pro určitou otázku, stačí pokaždé pro danou podmínku, nebo výběr provést dotaz nad odpovídající tabulkou. Poté musíme data, získaná z těchto jednotlivých dotazů nějakým způsobem sjednotit. Samotné dotazy a sjednocení lze provést až na konci pomocí spojení – postupně bychom spojili výsledky dotazů pro jednotlivé podmínky nad potřebnými tabulkami. Zde však nastává problém s velkou výpočetní složitostí. Mohli bychom mít například formulář s dvaceti otázkami a tyto otázky se budou nacházet ve třech tabulkách - museli bychom nad nimi tedy dvacetkrát provést spojení. Tento dotaz bychom museli tvořit dynamicky (pro různý počet podmínek a použitých tabulek). Zde také dochází k problému určení místa selhání. Uživatel, provádějící zpracování dat může například zadat v pěti z dvaceti otázek určité podmínky a v jedné z nich udělá chybu. To, že provedl chybu, se dovíme až na konci a navíc mu nedokážeme říci, kde přesně k chybě došlo.

Proto musíme u každé otázky formuláře provést dotaz ihned po zadání podmínky (pokud byla nějaká zadána). Pokud byla podmínka zadána špatně, můžeme okamžitě upozornit uživatele a ten ji může opravit. Výsledek každého dotazu bude poté uložen do sezení.

Na konci tedy získáme množiny dotazníků, které odpovídají jednotlivým podmínkám (viz Obr. 20). Navíc k nim přidáme množinu, kterou jsme definovali ve výběru a pro kterou chceme zobrazit výsledek (pokud nás například zajímá, kolik žen odpovídá zadaným podmínkám, musí být do výsledné množiny zahrnuty jen ty dotazníky, které vyplnily ženy). Nyní z nich musíme získat ty, které odpovídají všem podmínkám – musíme nalézt ty řádky, které se nacházejí ve všech získaných množinách. Protože jich nemůže být více, než je nejmenší samostatná množina, z výsledků tuto množinu vybereme a použijeme ji jako výchozí. Zbývá nám z této množiny odstranit ty, které se v ostatních množinách nenacházejí. Proto postupně projdeme všechny řádky nejmenší množiny a zkontrolujeme, zda se nachází ve všech ostatních množinách. Pokud se alespoň v jedné daný řádek nenachází, bude tento řádek smazán a přejdeme na kontrolu dalšího. Tímto získáme výslednou množinu, odpovídající všem podmínkám a podle její velikosti určíme výsledný procentuální podíl.

Nyní musíme naimplementovat akce serveru, potřebné k provedení celého zpracování dat. Uživatel musí mít možnost vybrat formulář, ze kterého chce zpracovávat data – potřebujeme tedy zobrazit seznam dostupných formulářů.

Po výběru potřebného formuláře by se měly zobrazit informace o něm – úvod daného formuláře a k tomu celkový počet vyplněných dotazníků. Poté by měl mít uživatel možnost přejít přímo k samotnému zpracování. Bude postupně procházet jednotlivé otázky – musíme zobrazit text otázky, její možnosti a vstupní pole pro zadání případného výběru a podmínky. Při každém potvrzení případně zadaných hodnot (při přechodu na další otázku, nebo na konec zpracování dat) musí dojít ke kontrole zadaných dat. Pokud jsou v pořádku (výběr obsahuje pouze čísla možností a dotaz pro získání množiny řádků, reprezentujících jednu podmínku neselhal), bude zpracování dat pokračovat. Pokud dojde k chybě, bude opět zobrazena původní otázka s chybovou zprávou.




**Obr. 20 – Vlevo se nachází množiny jednotlivých podmínek, napravo výsledek (množina, která odpovídá všem podmínkám)**

Na samotném konci dojde k výše uvedenému „pročištění“ výsledku (promazání nejmenší množiny) a zobrazení výsledku. Pro každý výběr zde bude zobrazen text otázky, vybraná možnost a výsledek v procentech.

Potřebujeme tedy akci pro zobrazení všech dostupných formulářů (*forms*), dále pro zobrazení úvodu (*start*), pro jednotlivé otázky (*question*) a nakonec pro zobrazení výsledku (*end*).

Všechny akce budou implementovány v jedné třídě – *MiningAction*. Ta bude implementovat rozhraní *SessionAware* – do sezení budeme totiž ukládat výsledky jednotlivých dotazů. Přístup k datům nám zajistí již existující třída *DataService*.

Pro uchovávání vyplněných vstupů pro výběr a pro podmínky vytvoříme speciální objekt – *QueryCreator*. V něm budou obsažena dvě pole se zadanými vstupy (*select* pro výběry u jednotlivých otázek a *where* pro zadané podmínky).

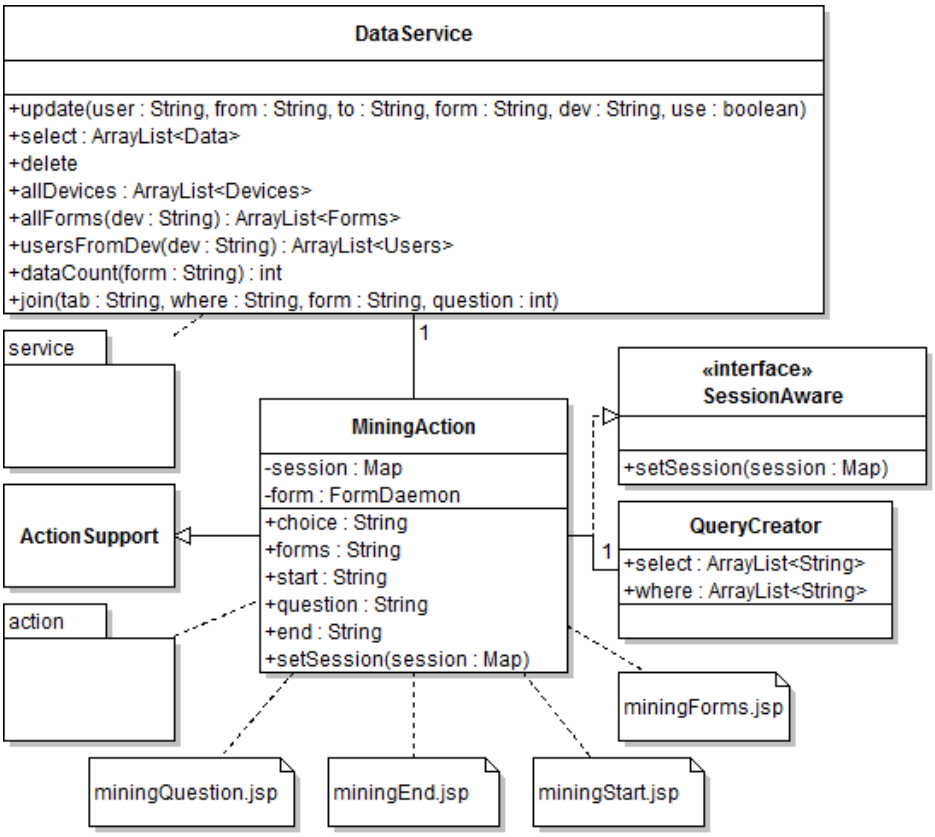
Pro zobrazení uživatelského rozhraní budou sloužit následující soubory JSP:

- *miningForms.jsp* – bude umožňovat výběr formuláře, jehož data chceme zpracovávat (akce *forms*).
- *miningStart.jsp* – zobrazí úvod formuláře (akce *start*).

- *miningQuestion.jsp* – zobrazí potřebnou otázku formuláře, spolu se vstupními poli pro zadání výběru a podmínky (akce *question*).

- *miningEnd.jsp* – poslouží pro zobrazení výsledku zpracování dat (akce *end*).

Nakonec nesmíme zapomenout definovat operace nad daty, které budeme potřebovat. Tyto operace umístíme do již existující třídy *DataService* – budeme totiž pracovat se stejnými daty jako při jejich správě. Pro výpočet výsledku potřebujeme znát celkový počet vyplněných dotazníků daného formuláře – k tomu využijeme metodu *dataCount*. Dále potřebujeme operace pro výběr dat odpovídajících jedné podmínce (metoda *join*) a pro výběr dat, odpovídajících výběru (*joinSelect*).




**Obr. 21 – Třídní diagram zpracování dat**

# 5 Implementace

## 5.1 Mobilní aplikace

Implementace mobilní části systému byla provedena pomocí vývojového prostředí *Eclipse 3.4.1*, doplněného pluginem *ADT* (*Android Development Tools*). Při tvorbě byl použit *Android SDK 1.1* a následně byla aplikace otestována i ve verzi platformy *Android 1.6*. Implementace víceméně odpovídá návrhu, proto zde již budou uvedeny odchylky od něj. Většinou byly pouze přidány nové pomocné metody a proměnné.

Textové řetězce nebyly zadávány přímo v kódu, ale je na ně vždy odkazováno pomocí třídy *R*. Všechny jsou umístěny v souboru *res/values/strings.xml.* Externí umístění veškerého textu zjednoduší případnou pozdější lokalizaci do jiných jazyků.

Uživatelská rozhraní všech komponent jsou definována pomocí odpovídajících XML souborů (složka *res/layout)*.

### 5.1.1 První spuštění

Při prvním spuštění aplikace je navíc zobrazena informace o tom, že bude následovat registrace prvního uživatele. K tomu byla v komponentě *Socioresearch* vytvořena nová metoda *welcome*. Teprve po kliknutí na tlačítko dojde k přesměrování na samotnou registraci.

### 5.1.2 Informace o aplikaci

Do hlavního menu byla přidána nová položka – *O programu*, reprezentovaná novou komponentou *About*. Obsahuje informace o aplikaci.

### 5.1.3 Nadpisy seznamů

U komponent se seznamy může dojít k dezorientaci uživatele (například při správě uživatelů skupiny jsou hned po sobě zobrazovány jejich různé seznamy, bez rozlišení, kde se uživatel právě nachází). Proto je u všech komponent, zobrazujících seznamy dat navíc při různých stavech nastaven jiný nadpis.

### 5.1.4 Tvorba a editace dotazníků

Do menu komponent *insertChoice* a *insertQuestion* byla přidána nová položka *Na konec otázek/voleb*. Umožní například uživateli, který chce přidat novou otázku na konec rozsáhlého formuláře rychlý přechod na dané místo.

### 5.1.5 Rozlišení stavů komponent

#### *5.1.5.1 Komponenta UsersManager*

Pro rozlišení stavu této komponenty je využita jediná proměnná – *type*. Obsahuje id skupiny, jejíž uživatele chceme zobrazit. Pokud je její hodnota vyšší jak -1, jsou zobrazeni uživatelé požadované skupiny (stav *Seznam uživatelů skupiny*), pokud se rovná -1, jsou zobrazeni všichni uživatelé (výchozí stav komponenty). A nakonec, pokud se rovná -2, jsou zobrazeni uživatelé, kteří se momentálně v žádné skupině nenachází (stav *Seznam všech volných uživatelů*).

#### *5.1.5.2 Komponenta FormsManager*

U této komponenty jsou pro rozlišení stavu využity dvě proměnné – *type* a *open* (typu *boolean*). Proměnná *type* zde opět reprezentuje id skupiny. Pokud je tato hodnota vyšší jak -1, budou zobrazeny formuláře dané skupiny. Toto zobrazení ale potřebujeme ve dvou různých stavech. K jejich rozlišení využijeme druhou proměnnou - *open.* Pokud je *true*, komponenta bude ve stavu *Výběr formuláře dané skupiny*. Pokud ne, bude ve stavu *Seznam formulářů skupiny*. Zbývají nám dva stavy. První z nich – výchozí stav (seznam všech formulářů zařízení) je reprezentován hodnotou -1 proměnné *type*. Zbývající (*Seznam všech volných formulářů*) je také reprezentován proměnnou *type* – tentokrát s hodnotou -2.

### 5.1.6 Práce se soubory

Protože potřebujeme nějakým způsobem načítat a ukládat formuláře (přistupujeme k nim z různých míst) a dočasně uchovávat data právě vyplňovaného dotazníku, byla vytvořena nová třída *InOut*. Obsahuje čtyři metody. Dvě pro práci s formuláři (*read* a *write*) a další dvě pro práci s dočasně uloženými daty (*readArray* a *writeArray*).

## 5.2 Server

Server byl naimplementován pomocí vývojového prostředí *netBeans IDE 6.5*. Jako databáze je zde využit systém *MySQL*. Pro implementaci byl využit framework *Struts 2* a relačně objektové mapování je zajištěno pomocí frameworku *Hibernate*. Implementace víceméně odpovídá návrhu, proto zde již budou uvedeny odchylky od něj. Většinou byly pouze přidány nové pomocné metody a proměnné.

Při tvorbě jsem vycházel ze semestrálního projektu do předmětu *Tvorba informačních systémů*. Z něj jsem převzal systém autentizace a autorizace (pomocí interceptoru *authenticationInterceptor*) a knihovnu uživatelských značek pro zobrazení tabulek *displaytag 1.2*.

Při implementaci byl navíc využit framework *Spring*, který nám zjednodušuje práci s třídami, které obsahují operace nad daty. V souboru nastavení *applicationContext.xml* jsou namapovány objekty potřebných tříd do parametrů konstruktorů tříd akcí. Nemusíme se tak zabývat jejich vytvářením přímo v kódu, framework *Spring* potřebné objekty vytvoří za nás.

### 5.2.1 Struktura projektu

Kód webové aplikace se nachází ve složce *WEB-INF/classes*:

- Třídy jednotlivých akcí jsou umístěny ve složce *action*. Zde se také nachází lokalizované řetězce projektu (*package.properties* s anglickými a *package_cs_cz.properties* s českými).
- Ve složce *model* se nachází třídy, které obstarávají namapování tabulek databáze. V této složce je také umístěna složka *keys*, obsahující třídy reprezentující složené klíče odpovídajících tabulek.
- Složka *form* obsahuje pomocné třídy.
- Složka *interceptor* obsahuje třídu interceptoru *authenticationInterceptor*.
- Ve složce *service* se nachází třídy s operacemi nad daty.

Soubory JSP jsou umístěny v adresáři *WEB-INF/classes* a nastavení webové aplikace je provedeno pomocí souborů *web.xml*, *struts.xml* a *applicationContext.xml*.

### 5.2.2 Příchod na server

Příchod na server, při kterém není požadována žádná akce, je ošetřen zobrazením „uvítacího“ souboru (nastaven v souboru *web/WEB-INF/web.xml* pomocí tagu *<welcome-file-list>*) Zde je tento soubor nazván *index.jsp* a nachází se ve složce *WEB*-INF, umístěné v kořenové složce webové aplikace. Tento soubor vždy provede automatické přesměrování na akci *login*.

### 5.2.3 Přihlášení

Pro ověření autentizace je využit interceptor *authenticationInterceptor* (deklarovaný v souboru *struts.xml* a umístěný ve třídě */WEB-INF/classes/interceptor/authenticationInterceptor.class*). Ten se postará o to, že se k funkcím serveru dostanou pouze autentizovaní uživatelé. To, zda je uživatel autentizován pozná interceptor podle toho, že je nastavena hodnota proměnné *authenticated*, nacházející se v sezení. Ta je společně s hodnotou *admin* (určující, zda je uživatel administrátor) nastavena během přihlašování. Pokud autentizován není, je zobrazen přihlašovací formulář (akce *login*) s případnou chybovou zprávou (vlastnost *error*).

### 5.2.4 Odhlášení

Při odhlášení (akce *logout* – metoda *LogoutAction.execute*) jsou ze sezení odstraněny proměnné *authenticated* a *admin*. Poté je zobrazen přihlašovací formulář *login.jsp*.

### 5.2.5 Menu

Po úspěšném přihlášení je pomocí souboru *menu.jsp* zobrazeno uživatelské menu. Podle toho, zda je, nebo není nastavena proměnná *admin*, jsou zobrazeny odpovídající položky menu. Ke kontrole dochází pomocí uživatelských značek přímo v samotném souboru *JSP*.

### 5.2.6 Seznamy dat

Pro jednodušší implementaci zobrazení seznamů dat (u akcí *person_list*, *device_list*, *data_update*, *users_list*, *groups_list*, *forms_list*, *formsOfGroup* a *usersOfGroup*) byla využita knihovna uživatelských značek *displaytag 1.2*, která ale bohužel neumožňuje lokalizaci.

### 5.2.7 Validace vstupních dat

Pro validaci vstupních dat byly využity validátory, zde reprezentované soubory XML (umístěné ve složce společně s třídami akcí). Validace je použita u akcí *person_update*, *person_insert*, *device_update*, *device_insert* a *data_update*. U prvních čtyř potřebujeme zajistit to, že budou vyplněna všechna požadovaná pole a u poslední akce zase správný formát vkládané časové periody.

### 5.2.8 Zpracování dat

Protože při samotném zpracování dat nepotřebujeme pouze uchovávat vyplněná textová pole *select* a *where*, ale i informaci o tom, o jakou otázku se jednalo a jakého typu daná otázka je (v jaké tabulce se nachází její odpovědi), byla vytvořena speciální třída *Inner*. Objekty této třídy

jsou vkládány do polí *select* a *where* třídy *QueryCreator*. Každý jednotlivý objekt třídy *Inner* vždy reprezentuje buď jeden výběr (to co nás zajímá ve výsledku), nebo jednu zadanou podmínku.

Při procházení otázek formuláře je navíc zobrazen typ otázky - ten nám umožňuje předejít tomu, aby například uživatel u typu otázky, kde lze vždy vybrat pouze jednu možnost, zadal do pole *where* text "*ch='0' and ch='1'*" (kde *ch* je jedna možnost). Výsledek by byl vždy 0%.

Byla zde také rozšířena třída *QueryCreator* – protože obsahuje informace o tom, která data nás ve výsledku zajímají (pole *select*), je využita k vygenerování obsahu stránky, na které se nachází výsledky zpracování dat.

U každé otázky byl ještě umístěn odkaz (*Na konec*), umožňující přímý přechod na konec zpracování dat (je zavolána akce *mining_end*).

### 5.2.9 Smazání souborů formulářů

Při správě dat upravujeme a mažeme množiny vyplněných dotazníků. Tyto operace se ale netýkají samotných souborů formulářů, obsahujících kontext sesbíraných dat. Může se tak stát, že se na serveru nebudou nacházet žádná data odpovídajícího formuláře, ale bude zde stále uložen jeho soubor. Musíme tedy nějakým způsobem ošetřit smazání tohoto souboru. Kvůli tomu byl upraven seznam formulářů akce *mining_forms*. Zde se totiž nachází seznam všech dostupných formulářů, nacházejících se na různých zařízeních. Byla do něj přidána operace smazání tohoto souboru (nová akce *mining_delete* - je dostupná pouze administrátorovi). Po smazání souboru také dojde ke smazání všech sesbíraných dat tohoto formuláře a všech zmínek o tomto formuláři v nastavení jednotlivých zařízení. Pokud by totiž určité zařízení požádalo o původní nastavení, nemohli bychom mu odpovídající soubor formuláře odeslat zpět.

Seznam formulářů je zde načítán ze souborového systému – jsou načteny existující soubory formulářů. Protože ale značka *<dislplay:table>* knihovny *displaytag 1.2* (využitá pro zobrazení seznamu) v jednotlivých sloupcích zobrazuje vybrané vlastnosti objektů umístěných v seznamu (ke které přistupuje podle názvu dané vlastnosti), musela být vytvořena vnořená třída *FileWrapper* obsahující jedinou vlastnost – *name*, reprezentující název souboru.

## 5.3 Spojení

### 5.3.1 Server

Protože na zařízení nepotřebujeme odesílat žádný HTML kód (metoda akce bude vracet řetězec *Action.NONE* – nebude tedy použit žádný JSP soubor), musíme využít jiný způsob pro zaslání odpovědi na zařízení. K tomu je zde využit objekt třídy *PrintWriter* (získaný pomocí *ServletActionContext.getResponse().getWriter()*). Pomocí něj můžeme na zařízení vždy odeslat jen odpověď, kterou potřebujeme.

Při implementaci byla také využita funkce relačního databázového systému *MySQL*, která automaticky transformuje poskytnutý řetězec o délce 12 znaků (obsahující čísla) na datový typ *DateTime*. To nám usnadňuje implementaci příjmu sesbíraných dat – na mobilním zařízení se totiž datum a čas nachází ve formě celých čísel. Tato vlastnost systému *MySQL* je také využita při správě dat (při získávání dat určité časové periody).

### 5.3.2 Mobilní aplikace

U mobilní aplikace bylo umístěno načítání a ukládání nastavení připojení (pomocí *Shared Preferences*) do třídy *Connect* – tato data potřebujeme hlavně pro nastavení spojení. Pro práci s nimi byly vytvořeny dvě metody – *get* a *set*. Také zde byly umístěny metody pro výpis stavu spojení – *writeInfoMessage* a *writeErrorMessage* – to proto, aby nemusely být u každé komponenty opakovaně definovány (u každé komponenty související s přenosem dat potřebujeme uživatele informovat o průběhu přenosu).

Do tříd, majících na starosti samotný přenos dat (*Send*, *Receive*, *SendCollected*) byly navíc přidány pomocné metody, které rozdělují celkový kód tříd do menších přehlednějších celků.

Došlo ale k problému při přenosu serializovaného objektu formuláře. Na straně serveru se bohužel nedařilo deserializovat objekty typu *ArrayList*, obsahující seznamy otázek a možností. Problém byl nejspíše v tom, že se na obou stranách nachází různé verze Javy. Java, která je používána při vývoji aplikací pro systém *Android* navíc není oficiální verzí společnosti *Sun Microsystems*. Proto je vždy při přenosu objekt formuláře transformován na text, odeslán, a na druhé straně znovu vytvořen.

# 6 Závěr

Celý projekt se podařilo naimplementovat podle zadání. Mobilní aplikace umožňuje vytvoření celého nastavení a všech formulářů prakticky „na koleně“, takže v podstatě nepotřebujeme přístup ke stolnímu počítači. Pomocí mobilního zařízení se také můžeme díky internetovému prohlížeči dostat na samotný server, který obsahuje jednotlivé zálohy a umožňuje distribuci námi vytvořeného nastavení mezi ostatní zařízení. Server také umožňuje zpracování námi vybraných sesbíraných dat – získané informace tak můžeme ihned vidět na zařízení.

Protože byl hlavní důraz kladen na mobilní aplikaci, je serverová část méně propracovaná. Další práce na serverové části by se měly zaměřit především na vzhled a uživatelské rozhraní. Například při výběru dat, která chceme upravit, by mohl být využit *javascript* a mohli bychom tak vybrat potřebnou množinu dat na jediné stránce. Vylepšit a zefektivnit by šel také samotný přenos dat, nebo by šly například ošetřit vstupy během zadávání podmínek při zpracování dat kvůli možnému *SQL injection*.

Při implementaci jsem si všiml veliké podobnosti architektury aplikací, psaných pro systém *Android* a webových aplikací architektury *J2EE* (s použitím frameworku *Struts 2*):

- U obou jsou definovány víceméně samostatné komponenty/akce v XML souboru nastavení (*manifest.xml*, respektive *struts.xml*).
- K těmto komponentám/akcím můžeme přistupovat samostatně - u webové aplikace se dostaneme na požadovanou stránku, pokud jsme autorizováni - u mobilní aplikace se můžeme dostat na komponentu jiné aplikace, pokud je to povoleno v souboru *manifest.xml* dané aplikace.
- Komponenty/akce obou systémů si mezi sebou mohou posílat „zprávy“ - akce webové aplikace GET a POST požadavky, komponenty mobilní aplikace zase data, zabalená v objektu třídy *Intent*.
- U webové aplikace je nastaven pomocí tagu *welcome-file* v souboru *web.xml* „vstupní“ soubor serveru, u mobilní aplikace je zase v souboru *manifest.xml* nastavena startovací komponenta.
- Stejně tak jako se v prohlížeči můžeme pohybovat zpět ke dříve navštíveným stránkám, i u mobilní aplikace systému Android se můžeme pohybovat zpět v zásobníku spuštěných komponent.
- V mobilní aplikaci také nelze definovat její celkové ukončení (lze maximálně ukončit jednu komponentu) – můžeme se pohybovat pouze směrem zpět v rámci zásobníku spuštěných komponent, stejně jako v prohlížeči ke dříve spuštěným stránkám (akcím serveru).

Tato práce je zatím největším projektem jaký jsem vytvořil a umožnila mi podívat se do větší hloubky tvorby webových aplikací, než jak by bylo možno při klasických semestrálních projektech. Mobilní část systému byla v podstatě první mobilní aplikací, jakou jsem napsal. Zjistil jsem, že se psaní aplikací pro chytré telefony už tolik neliší od psaní klasických desktopových programů.

# 7 Literatura

[1] http://www.computerworld.cz, *Deník pro IT profesionály*

[2] http://www.androidforum.cz, *Otevřená komunita pro otevřenou platformu*

[3] http://www.pcworld.cz, *Recenze, novinky a testy: Hardware, Software, Download a Internet*

[4] http://www.developer.android.com, *Android Developers*

[5] http://www.dotaznik-online.cz, *Vše o dotaznících…*

[6] http://www.linuxsoft.cz, *Linux software*

# 8 Přílohy bakalářské práce

1) **Uživatelská příručka – mobilní aplikace**, v elektronické podobě uložena na DVD
2) **Uživatelská příručka – server**, v elektronické podobě uložena na DVD
3) **Programátorská příručka – mobilní aplikace**, v elektronické podobě uložena na DVD
4) **Programátorská příručka - server**, v elektronické podobě uložena na DVD
5) **Disk DVD**, obsahující elektronické verze dokumentů bakalářské práce, zdrojové kódy, apod.