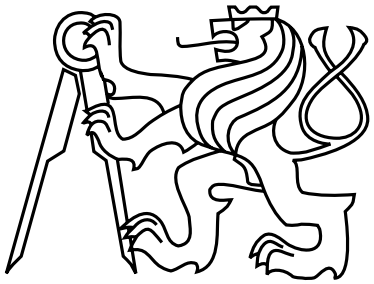
České vysoké učení technické v Praze
Fakulta elektrotechnická
Katedra počítačů



Diplomová práce

Docházkový systém

Petr Introvič

Vedoucí práce: Ing. Martin Molhanec CSc.

Studijní program: Výpočetní technika, Magisterský

Obor: Softwarové inženýrství

5. ledna 2015

České vysoké učení technické v Praze
Fakulta elektrotechnická

katedra počítačů

ZADÁNÍ DIPLOMOVÉ PRÁCE

Student: **Bc. Petr Introvič**

Studijní program: Elektrotechnika a informatika (magisterský), strukturovaný
Obor: Výpočetní technika

Název tématu: **Mobilní docházkový systém**

Pokyny pro vypracování:

Realizujte mobilní docházkový systém.
Systém bude umožňovat evidenci pracovní docházky na mobilních zařízeních. Systém bude evidovat osobu, místo, počátek a konec pracovního výkonu.
Získaná data se budou shromažďovat na centrálním úložišti. Součástí systému bude centrální aplikace pro správu shromážděných dat.
Student sám zvolí vhodnou technologii pro implementaci systému.

Proveďte analýzu, návrh a implementaci informačního systému.
Všechny části projektu budou pečlivě dokumentovány v souladu se zvyklostmi softwarového inženýrství.
Systém otestuje na jeho bezchybnou funkci.
Součástí práce bude také příručka uživatele a administrátora.

Seznam odborné literatury:

Softwarové inženýrství - K. Richta
Databázové systémy - I. Halaška
firemní manuály k jednotlivým produktům

Vedoucí: Ing. Martin Molhanec, CSc.

Platnost zadání: do konce letního semestru 2014/2015

L.S.

doc. Ing. Filip Železný, Ph.D.
vedoucí katedry

prof. Ing. Pavel Ripka, CSc.
děkan

V Praze dne 25. 9. 2013

Poděkování

Velice rád bych poděkoval Ing. Martinovi Molhancovi CSc. za společné konzultace a vedení mé práce.

Prohlášení

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

V Praze dne 4. 1. 2015 .....

Abstract

This thesis deals with creation of a modern attendance system with use of cloud services for floorbal section of team Tatran Střešovic. Aim of this work is to analyze existing solutions, design and implementation of own solution. System has to be accesible from wide area of devices.

This work describes analysis a requirements of such system, propose and also implements use of cloud database with native client for Android platform and access through web interface. Part of work is testing of system, conclusion with possible future evolution and user guide.

Abstrakt

Tato práce se zabývá návrhem moderního docházkového systému s využitím cloudových služeb pro florbalový oddíl klubu Tatran Střešovice. Cílem je analýza stávajících řešení, návrh a implementace vlastního řešení. Systém má být přístupný z co nejširšího rozsahu zařízení.

Práce popisuje analýzu a požadavky na systém, navrhuje a zároveň implementuje využití cloudové databáze s nativním klientem pro platformou Android a přístupem přes webové rozhraní. Součástí práce je testování systému, závěr s zhodnocením dalšího možného vývoje a uživatelská příručka.

Obsah

1 Úvod **1**

2 Použité metodiky a technologie **3**
2.1 Platforma Android . . . 3
2.1.1 Architektura . . . 3
2.1.2 Vývoj pro android . . . 5
2.2 Databázový backend . . . 6
2.2.1 Formát dat . . . 6
2.2.2 Ukázka práce s Firebase . . . 6
2.2.3 Bezpečnostní pravidla . . . 8
2.2.4 Hosting . . . 9
2.3 Účty Gmail a Google+ . . . 9
2.4 Metodiky . . . 9

3 Současný stav problematiky **11**
3.1 Stávající řešení . . . 11
3.2 Stávající aplikace . . . 12
3.3 Srovnání aplikací . . . 23
3.4 Vyhodnocení . . . 24

4 Úvodní studie **25**
4.1 Seznam požadavků . . . 25
4.1.1 Funkční požadavky . . . 25
4.1.2 Nefunkční požadavky . . . 27
4.2 Návrh řešení . . . 29
4.2.1 Finální návrh . . . 33
4.3 Rozdělení na procesy . . . 33
4.4 Analýza rizik . . . 33
4.4.1 Technická náročnost projektu . . . 33
4.4.2 Časová náročnost projektu . . . 34
4.4.3 Ztráta dat . . . 34
4.4.4 Neznalost a zkušenosti s technologiemi . . . 35
4.4.5 Nesplnění požadavků . . . 35
4.4.6 Bezpečnost . . . 35
4.4.7 Nepřehledný vývoj . . . 35

4.4.8 Vhodnost a použitelnost vývojových nástrojů . . . 36
4.4.9 Nedostatečné testování produktu . . . 36
4.5 Harmonogram . . . 36

5 Analýza **39**
5.1 Datový konceptuální diagram . . . 39
5.1.1 Popis entit . . . 40
5.1.2 Popis vztahů mezi entitami . . . 41
5.2 Popis případů užití . . . 42
5.2.1 Případy užití správy skupin . . . 42
5.2.2 Případy užití správy skupiny . . . 46
5.2.3 Případy užití změny přihlášení . . . 55
5.3 Analýza návrhu uživatelského rozhraní . . . 58
5.3.1 Mobilní klient . . . 58
5.3.2 Webové rozhraní . . . 59
5.3.3 Případ užití - Zobrazení skupiny . . . 59

6 Návrh a implementace **61**
6.1 Fyzický datový model . . . 61
6.2 Převod do databáze Firebase . . . 62
6.3 Mobilní klient . . . 63
6.3.1 Použité knihovny . . . 63
6.3.2 Uživatelské rozhraní . . . 71
6.4 Webové rozhraní . . . 77
6.4.1 Použité knihovny . . . 78
6.4.2 Uživatelské rozhraní . . . 78
6.5 Spolupráce přihlášených klientů . . . 80
6.6 Použité návrhové vzory . . . 81
6.6.1 Singleton vzor . . . 81
6.6.2 Observer vzor . . . 81
6.6.3 Factory method vzor . . . 82
6.6.4 ViewHolder vzor . . . 83
6.7 Struktura kódu mobilního klienta . . . 85
6.8 Struktura kódu webového rozhraní . . . 85

7 Testování **87**
7.1 Automatické testování rozhraní mobilního klienta - Espresso . . . 87
7.2 Uživatelské testování mobilního klienta . . . 88
7.2.1 Testeři . . . 89
7.2.2 Datová náročnost . . . 89
7.3 Testování webového rozhraní . . . 89

8 Závěr **91**

A Seznam použitých zkratek **95**

B Uživatelská a instalační příručka **97**
B.1 Příručka mobilní aplikace 97
B.2 Příručka webového rozhraní 97
B.3 Instalační příručka admina 100

C Obsah přiloženého CD **101**

Seznam obrázků

2.1 Architektura Androidu . . . 3
2.2 Hierarchická struktura Firebase databáze . . . 7

3.1 Role ve stávajícím řešení . . . 11
3.2 Ukázka rozhraní aplikace Attendance . . . 13
3.3 Rozhraní aplikace AttendanceManager . . . 14
3.4 Úvodní obrazovka a nastavení AttendanceTrackeru . . . 15
3.5 Attendance Tracker - Správa jednotlivých skupin a událostí . . . 16
3.6 Attendance Tracker - Generování přehledů . . . 16
3.7 Rozhraní aplikace Students . . . 17
3.8 Attendance - úvodní nastavení aplikace . . . 19
3.9 Attendance - správa účastí členů . . . 19
3.10 Rozhraní aplikace Easy Attendance School College . . . 20
3.11 Účast - rozhraní . . . 21
3.12 Účast - formát tabulky . . . 21

4.1 Seznam rolí . . . 28
4.2 První raný návrh řešení databáze . . . 31
4.3 Rozdělení verzí Androidu k 1.12.2014 . . . 32
4.4 Diagram návrhu nasazení . . . 33
4.5 Rozdělení na procesy . . . 34
4.6 Časový harmonogram procesů . . . 37

5.1 Konceptuální datový model . . . 39
5.2 Model skupin případů užití . . . 42
5.3 Diagram případů užití pro správu skupin . . . 42
5.4 Případ užití - přidání skupiny . . . 43
5.5 Případ užití - úprava skupiny . . . 44
5.6 Případ užití - smazání skupiny . . . 45
5.7 Diagram případů užití pro správu konkrétní skupiny . . . 46
5.8 Případ užití - zobrazení skupiny . . . 47
5.9 Případ užití - přidání člena . . . 48
5.10 Případ užití - úprava člena . . . 48
5.11 Případ užití - smazání člena . . . 50
5.12 Případ užití - zadání účasti . . . 54
5.13 Diagram případů užití změny přihlášení . . . 56

5.14 Případ užití - Přihlášení 56
5.15 Případ užití - odebrání práv aplikace 57
5.16 Návrh UI - zobrazení skupiny 60

6.1 Fyzický datový model databáze 61
6.2 Struktura dat ve Firebase 62
6.3 EventBus 64
6.4 Ukázka FloatLabelText 67
6.5 Životní cyklus aktivity 68
6.6 Implementace UI - Zobrazení skupin a prázdná skupina - CardView 71
6.7 Schématické zobrazení toku akcí uživatele 72
6.8 Rozvržení UI - Zobrazení skupiny 74
6.9 Ukázka UI - Zobrazení skupiny 76
6.10 Rozvržení částí ve webovém rozhraní 79
6.11 Spolupráce připojených klientů 81

B.1 Ukázky rozhraní aplikace - přihlášení, prázdná seznam skupin, přidání skupiny 98
B.2 Ukázky rozhraní aplikace - seznam skupin, prázdná data skupiny, zobrazení dat 98
B.3 Ukázky rozhraní aplikace - zadání účasti, přidání člena, úprava člena 99
B.4 Ukázka webového rozhraní 99

Seznam tabulek

3.1 Srovnání stávajících aplikací 23

4.1 Rozdělení verzí Androidu k 1.12.2014 32

Kapitola 1

Úvod

V dnešní době se většina papírových prací postupně přesouvá k počítačům a mobilním zařízením. Potřeba vést přehledně, jednoduše a pravidelně docházku se dá najít ve spoustě odvětvích. Ať už se jedná o studenty ve škole, pracovníky v zaměstnání, svěřence v zájmovém kroužku, přihlášených lidí na seminář, umělců na zkoušky představení.

Mnou vybrané téma jsem si zvolil zejména proto, že jsem dlouholetým aktivním trenérem ve florbalovém klubu. Ze začátku povinnost vést docházku jsme neměli vůbec. Po čase si však každý trenér nějakým stylem začal vést docházku (ať už pro svojí potřebu nominování hráčů dle účasti či pro kontrolu například rodičů) až vznikla potřeba vést v našem klubu celkovou docházku všech členů. Do které by měli přístup rodiče pro kontrolu, šéftrenér pro celkový dohled a trenéři pro sledování nasazení jednotlivých členů.

Cílem této práce je navrhnout systém pro vedení docházky, který by byl přístupný pro všechny, ale zároveň pohodlně použitelný pro trenéry takříkajíc z terénu. Tedy aby každý trenér na tréninku mohl rychle a jednoduše zadat docházku na nové události a aby všichni rodiče/ostatní trenéři/vedoucí mohli k této docházce jednoduše přistupovat.

Potřeba vypracování tématu

Téma práce patří k běžným činnostem a tudíž se dá předpokládat, že se tímto tématem zabývalo mnoho lidí. Proto součástí práce je i důkladná analýza stávajících řešení. Vzhledem k tomu, že práce by měla najít reálné použití minimálně v našem klubu je moje motivace na kvalitní zpracování tématu vysoká. Zkušenosti s vedením docházky v klubu (tedy zkušenosti s požadavky na takový systém) se budou hodit při návrhu nového systému. Už z podstaty využití systému naznačeného v předchozím odstavci vyplývá, že bude potřeba použití nějakého vzdáleného úložiště dat či synchronizačního systému. Využity budou pravděpodobně různé nastupující moderní technologie což vede k atraktivnosti vypracování tématu.

Struktura práce

Použité metodiky a technologie

Kapitola obsahuje popis technologií týkajících se této práce. Věnuje se platformě Android, databázovému cloudovému řešení Firebase a sociální síti Google+.

Současný stav problematiky

Tato kapitola detailně rozebírá současný stav vedení docházky v klubu a analyzuje alternativní existující řešení. Zároveň hodnotí jestli je vlastní řešení nutné.

Úvodní studie

V úvodní studii se definují požadavky na systém, navrhuje řešení systému a analyzují rizika vypracování práce. Součástí je rozdělení práce na jednotlivé dílčí procesy a rozvržení těchto procesů v čase.

Analýza

Analýza obsahuje datový konceptuální diagram systému, popisuje jednotlivé entity a vztahy mezi nimi. Rozsáhlou součástí je pak podrobný popis případů užití a rámcový návrh řešení uživatelského rozhraní.

Návrh a implementace

Obsahem kapitoly Návrh a implementace je popis datového modelu, jeho převedení do našeho databázového systému. Dále řešení mobilního klienta, implementace jeho uživatelského rozhraní, popis webového rozhraní, použití návrhových vzorů a struktura výsledného kódu systému.

Testování

Kapitola testování obsahuje popis automatické testování systému, použité nástroje a zařízení a testování s uživateli.

Závěr

Do závěru patří shrnutí práce na projektu, vyhodnocení a naznačení další možné práce na systému.

Kapitola 2

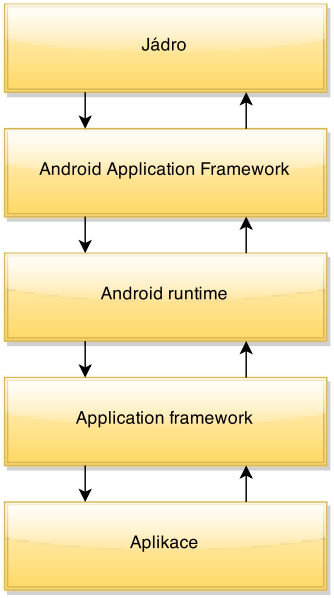
Použité metodiky a technologie

Práce zahrnuje tři hlavní části. První je mobilní klient vyvíjený pro platformu Android. Druhou cloudová databáze Firebase. Třetí částí je server s webovým rozhraním pro přístup do databáze. V této části jsou popsány principy těchto technologií.

2.1 Platforma Android

Android[5] je platforma, která vznikla zejména pro mobilní zařízení. Obsahuje operační systém založený na Linuxu, middleware, uživatelské rozhraní a aplikace. Nedílnou součástí jsou nástroje pro vývoj aplikací.

2.1.1 Architektura



Obrázek 2.1: Architektura Androidu

Architektura systému Android je rozdělena do pěti vrstev. Rozdělení ilustruje obrázek 2.1.

Jádro operačního systému

Je nejnižší vrstvou a tvoří abstraktní vrstvu mezi hardwarem a ostatním softwarem. Založeno je na Linuxu.

Android application framework

Jedná se vlastně o knihovny napsané v C/C++ a využity jsou různými částmi systému. Jedná se například o knihovny:

- media - přehrávání audio/video formátů či obrazových souborů nejběžnějších formátů (mpeg4, h.264, mp3, aac, amr, jpg, png, ...)
- SQLite - relační databázová knihovna
- OpenSSL
- knihovna webového prohlížeče
- a další

Android runtime

Tato vrstva obsahuje aplikační virtuální stroj *Dalvik*. Ten obsahuje většinu základních knihoven jazyka *Java* ve verzi *Java Standard Edition*. Neobsahuje standardní knihovny pro uživatelské rozhraní *Abstract Window Toolkit* ani *Swing*, ale vlastní knihovny uživatelského rozhraní pro *Android*.

Překlad aplikace pro Android probíhá zkompilováním *Java* kódu do *Java* byte kódu, poté se kód překompiluje pomocí *Dalvik* kompilátoru a výsledný *Dalvik* byte kód je poté na Androidu spustitelný.

Každá aplikace na *Androidu* běží ve svém vlastním procesů s vlastní instancí virtuálního stroje *Dalvik*.

Application framework

Zprostředkovává vývojářům velké množství služeb z kterých mohou být aplikace sestavené. Jedná se o prvky uživatelského rohraní, služby, přístup k hardwaru rozdělených do různých kategorií:

- balíky android.view.*, android.widget.* pro uživatelské rozhraní
- aktivity - spravování životního cyklu aktivit aplikací
- zdroje - přístup k zdrojům jako jsou řetězce, grafika, přidané soubory
- notifikace - spravování notifikací ve stavovém řádku
- a další

Aplikace

Nejvyšší vrstvu tvoří samotné aplikace. Mohou býti buďto předinstalované či dodatečně stažené z Google Play.

2.1.2 Vývoj pro android

Aplikace pro *Android* se píší v jazyce *Java*. Existují i alternativní možnosti, ale v oficiálně podporovaných vývojových prostředích Android Studio (založené na *Intelli J Idea*) a v *Eclipse* se využívá pouze *Java*. Standardně podporované verze jsou 1.6 i 1.7.

Standardní struktura projektu

- projekt
  - libs
    - dodatečné knihovny projektu
  - src
    - java
      - veškerý aplikační kód
    - res
      - drawable
      - layout
      - menu
      - values
    - AndroidManifest.xml
  - build.gradle

Zdrojový kód psaný v jazyce *Java* je ve složce *src/java* rozdělen do balíků a tříd.

Zdroje ve složce *res* mohou být rozděleny podle orientací, denzit displeje zařízení, jazyku zařízení, velikosti displejů a dalších atributů. Činí se tak přívlastkem na konci jména složky, například *layout-portrait*, *menu-en*, *menu-cs*, *values-hdpi*.

Většina zdrojů je definovaná pomocí *XML*, ať už se jedná o definici menu, animací, UI či grafiky. Využít se dají i jiné zdroje, grafika bitmapových obrázků, zvuky, či jakékoliv binární soubory.

AndroidManifest.xml je hlavní soubor, který definuje aplikace. Obsahuje vstupní body aplikace, požadované oprávnění a další.

Google Play

Aplikace pro *Android* se distribuují pomocí obchodů. Základním a hlavním obchodem je *Google Play*, který obsahuje největší množství aplikací pro *Android*. Pokud si uživatel stáhne aplikaci z nějakého obchodu, pak se obchod stará o aktualizace, počítá stažení dané aplikace a poskytuje vývojáři obsáhlé nástroje pro statistiky a ladění aplikací.

Android Material Design Guidelines

Pro Android existují oficiální příručky tvorby aplikací, které se v nejaktuálnější shrnují pod název *Material Design Guidelines*[1]. Pravidla neobsahují pouze vzhledová doporučení ale i doporučení co se funkčnosti a postupů interakce s uživatelem týče.

Pomocí těchto pravidel je velmi silně doporučeno postupovat. Aplikace takto vytvořené zapadají do celého systému *Androidu* a ač můžou designem či funkcemi se odlišovat, tak jejich užití je pro uživatele intuitivní.

2.2 Databázový backend

Firebase poskytuje realtimeovou databázi a backend jako službu. Služba poskytuje vývojáři *API*, které umožňuje ukládání dat ve stromové struktuře, která jsou automaticky synchronizovány s cloudovými servery *Firebase* a se všemi klienty, kteří naslouchají změnám v těchto datech.

K dispozici jsou knihovny pro integraci s *OS Android, iOS, Java, Objective-C, Node.js* a k databázi se dá rovněž přistupovat pomocí *REST API*.

Jedná se tedy o databázi v cloudu, ke které může vývojář skrz knihovny vzdáleně přistupovat. Databáze obsahuje pouze data a pravidla podle kterých se k ní dá přistupovat. Definování významu dat a práce s nimi náleží klientům.

Každá databáze ve *Firebase* má vlastní unikátní *URL*, pomocí které se k ní dá přistupovat. Při změně dat u klienta, se tyto změní jak v cloudu, tak i u všech aktuálně připojených zařízení.

Aplikace fungují i když nejsou zrovna připojeny k internetu, data jsou uložena lokálně a jsou synchronizována ihned jak se připojení obnoví. Nicméně je to ale doplňková funkce a využití *Firebase* počítá s většinovým přípojením k internetu.

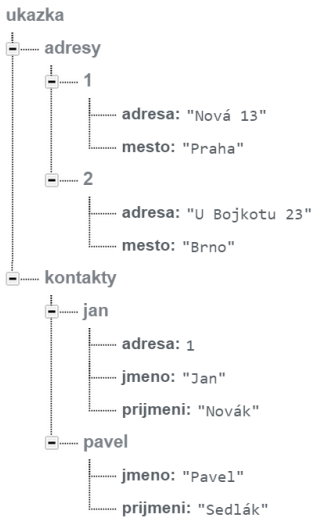
2.2.1 Formát dat

Data jsou uložena ve stromové struktuře ve formátu *JSON*. Ke každému prvku lze přistupovat pomocí cesty, která se skládá z klíčů předchůdců a klíče samotného prvku. Z principu použití klíčů objektů jako přístupových klíčů nejsou ve Firebase implicitně podporovány pole, lze je ale imitovat použitím číselných klíčů. Pole by pak bylo mapováno tak, že index prvku v poli by byl zároveň klíčem v objektu.

2.2.2 Ukázka práce s Firebase

Ukázky v této části jsou psány na klientu, v tomto případě aplikace na *Androidu*, jazykem je *Java*. Veškeré akce jsou připojeny na databázový server *Firebase* a operace provádí síťovou komunikaci a ideálně vše automaticky ukládají jak v cloudu tak u klienta.

[1] Odkaz na Material Design Guidelines



Obrázek 2.2: Hierarchická struktura Firebase databáze

Inicializace

Pro připojení k databázi je třeba vytvořit objekt *Firebase* s parametrem *URL* dané databáze.

```
1  // import trid
2  import com.firebase.client.Firebase;
3
4  // hlavni objekt pro praci s Firebase databazi
5  Firebase fbRefRoot = new Firebase("https://moje-url.firebaseio.com/");
```

Po absolutní části *URL* může následovat cesta, která bude odkazovat přímo na konkrétní část v databázi. Pro pracování pouze s *kontakty* z ukázky 2.2 bychom vytvořili objekt takto:

```
6  Firebase fbRefKontakty = new Firebase(
7      "https://moje-url.firebaseio.com/kontakty"
8  );
```

Ukládání dat

Po inicializaci pak lze data ukládat přímým voláním metod na potomcích.

```
9  // nastavi objektu "jmeno" a "prijmeni" na dane hodnoty
10 fbRefRoot.child("kontakty/jan/jmeno").setValue("Honza");
11 fbRefRoot.child("kontakty/jan/prijmeni").setValue("Novak");
```

Čtení dat

Pro čtení dat je potřeba zaregistrovat objekt, který vyvolané události zpracuje. Událost *onDataChange* je vyvolána vždy při prvotním načtení dat a poté při každé změně.

```
12    // posluchac udalosti zmeny dat
13    ValueEventListener listener = new ValueEventListener() {
14
15        // Udalost vyvolana pri zmene dat
16        // snapshot - aktualni data
17        @Override
18        public void onDataChange(DataSnapshot snapshot) {
19            // zjisteni dat z objektu
20            String jmeno = snapshot.child("jmeno").getValue(String.class),
21                prijmeni = snapshot.child("prijmeni").getValue(string.class);
22
23            // zobrazeni dat
24            System.out.println(snapshot.getKey() + " -> jmeno: " + jmeno + prijmeni);
25        }
26
27        // Udalost vyvolana pri zruseni pozadavku
28        // pri selhani sitove komunikace, pri nedostatecnych pravech
29        @Override
30        public void onCancelled(FirebaseError error) {
31            // Osetreni chyby
32        }
33    };
34
35    // Zaregistrovani objektu posluchace na zmeny dat
36    fbRefRoot.addValueEventListener(listener);
```

2.2.3 Bezpečnostní pravidla

Vzhledem k tomu, že všechny databáze jsou veřejně přístupné na vlastní *URL*, je třeba mít možnost nějak omezit práva uživatelů. *Firebase* má deklarativní jazyk pro specifikování pravidel, která jsou uložena přímo na serveru a určují bezpečnost aplikace.

Pomocí těchto pravidel lze kontrolovat přístup ke každému objektu databáze. Kaskádově se aplikují na děti daného objektu. K dispozici je i množství vestavěných funkcí a proměnných. Nejdůležitější proměnnou je *auth*, která obsahuje informace o přihlášeném uživateli.

Ukázka bezpečnostního pravidla:

```
1   {
2       "rules": {
3           "kontakty": {
4               "$user_id": {
5                   ".write": "$user_id === auth.uid",
6                   ".read" : true
7               }
8           }
9       }
10  }
```

Toto pravidlo umožňuje zápis do daného objektu pouze uživatelovi, jehož ID se rovná klíči objektu. Tedy do objektu *"kontakty/<uživatel.uid>/"*. Čtení objektu je povoleno všem.

2.2.4 Hosting

Firebase nabízí i hosting partnerstvím s firmou *Fastly*. Podporovány jsou pouze statické soubory (*HTML*, *CSS*, *Javascript*, a další), žádné dynamické stránky. Veškerá komunikace probíhá skrz *HTTPS* a *SSL*. Obsah je distribuován z *CDN*. Content Delivery Network je infrastruktura geograficky distribuovaných a spolupracujících serverů, které společně se sadou optimalizačních technik umožňují uživatelům internetových služeb (web, streamování videa, stahování souborů, apod.) rychlejší přístup k datům než v případě umístění dat na vlastním serveru nebo při využití běžné hostingové služby.

2.3 Účty Gmail a Google+

Původně emailový účet of firmy *Google* pro službu *Gmail* se postupem času vyvinul v hlavní účet pro přístup ke ve všem službám *Google* (mapy *Maps*, videoportál *Youtube*, webový disk *Drive*, reklamní systém *Adsense*, ...).

Google+ je další evolucí služeb, jedná se o sociální síť. V druhém významu pak také o účet této sociální sítě. Platforma okolo *Google+*[2] poskytuje možnosti autentizace uživatele v aplikacích právě pomocí účtu *Google+*. Při požadavku na přihlášení v aplikaci je při správné implementaci *Google+ přihlášení* uživateli nabídnut tok akcí mimo aplikaci, během kterých se přihlásí ke svému účtu. Po návratu do aplikaci tato získá autorizační token uživatele, s kterým může dále pracovat pod identitou tohoto uživatele.

OS Android je s *Google* účtem v základu úzce spojen a použití tohoto přihlášení je běžné. Kromě Androidu je podporován i *iOS* od *Apple* a pro všechny ostatní zařízení i přihlášení přes webové rozhraní.

2.4 Metodiky

Obě metodiky jsou obecně známé a tak se o nich zmiňuji jen okrajově. O obou metodikách existuje spousta článků a knih a není problém se o nich dozvědět více.

RUP

RUP (Rational Unified Process) je metodika vývoje softwaru, která definuje praktiky a postupy pro vývoj software.

Definuje fáze projektu a disciplíny, které během jednotlivých fází probíhají. Hodí se spíše pro větší projekty a tak v mé práci využívám jen disciplíny *Správa požadavků, Analýza a návrh, Implementace, Testování a nasazení.*

UML

UML (Unified Modeling Language) je standardní univerzální jazyk pro vizuální modelování systémů. UML definuje vizuální syntaxi ve formě diagramů pro zaznamenání vlastností

[2] odkaz na dokumentaci platformy Google+

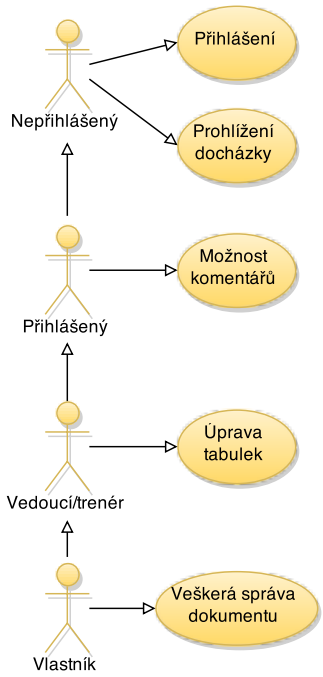
systému. Existují různé typy diagramů, v této práci jsou využity zejména: *diagram tříd*, *diagram případu užití* a další.

Kapitola 3

Současný stav problematiky

Kapitola rozebírá stávající řešení v týmu a poohlíží se po alternativním řešení. Vzhledem k tomu, že součástí má být i použitelnost na mobilních zařízeních se systémem *Android*, tak do průzkumu současným alternativ jsou zahrnuty nejlépe hodnocené aplikace z obchodu Androidích aplikací *GooglePlay*.

3.1 Stávající řešení



Obrázek 3.1: Role ve stávajícím řešení

Současným řešením vedení docházky v našem týmu je sdílená tabulka v dokumentech *Google*. Tabulka se skládá z jednotlivých sešitů pro každou kategorii. Trenéři a vedoucí druž-

stev mají povolenou editaci a vkládání komentářů. Přihlášení uživatelé mohou komentovat, nepřihlášení pak pouze prohlížet.

Funguje zde tak velmi dobře rozdělení jednotlivých rolí 3.1.

- Nepřihlášený - může prohlížet docházku
- Přihlášený - může svým jménem komentovat jednotlivé účasti
- Přihlášený trenér/vedoucí - může upravovat, měnit jednotlivé účasti, měnit tabulky, mazat dokonce celé sešity
- Vlastník (šéftrenér) - veškerá kontrola nad souborem

Z pohledu mobilní aplikace

Trenéři využívají nativní aplikaci *Tabulky Google*, která zprostředkovává přístup do docházky. Aplikace musí celý dokument (se všemi sešity, které čítají desítky členu a stovky událostí) stahovat vždy od znova a vykreslovat buňku od buňky. Vykreslování konkrétního sešitu trvá i vteřiny, je náročné na procesor. Každý trenér si sešit se svojí docházkou vede podle svého, zobrazení jsou nekonzistentní. Pro zadání nové účasti musí uživatel vytvářet nový sloupec, nastavovat ho, vyplňovat ručně účast (nutnost zadání hodnot na klávesnici na displeji). Pro každého nového člena či událost je potřeba vše stylovat, přidávat do výpočtů procentuálních účastí a přehledů. Sloupcům se musí měnit i velikost. Práce z pohledu trenéra je skrz mobilní aplikaci zdlouhavá, neefektivní a výsledek je těžce nekonzistentní.

Z pohledu webového rozhraní

Trenér si samozřejmě může data dostatečně předpřipravit pomocí webového rozhraní. Práci to rozhodně ulehčuje, nutnost nastavování nových událostí, členů či účastí ale zůstává. I na výkonnějších strojích se rozsáhlá tabulka načítá vteřiny.

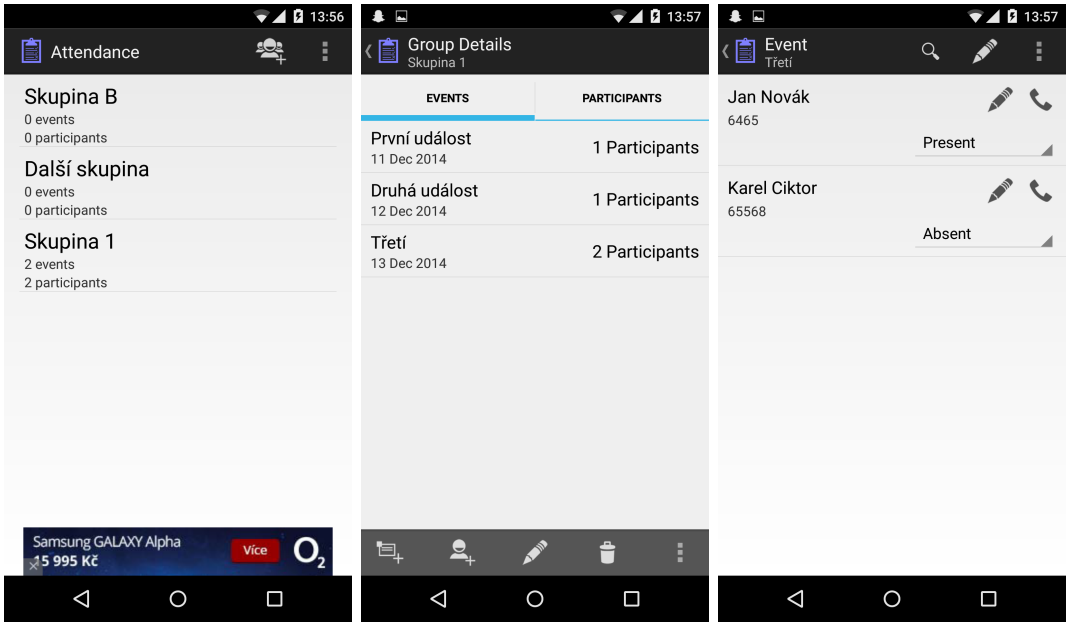
Pro uživatele, kteří prohlížejí docházku skrz web, rozdílné sešity působí nedobře, náročnost načtení tabulky odrazuje.

Shrnutí

Použití tabulek Google sice umožňuje vedení docházky, ale použitelnost je na nízké úrovni. Výsledek je nekonzistentní a rychlost užívání zdlouhavá. Z těchto důvodu se od stávajícího řešení chce upustit.

3.2 Stávající aplikace

V oficiálním obchodě Google Play existuje celá řada aplikací věnujících se spravování docházky. Do průzkumu byly zahrnuty ty, které od uživatelů měly nejlepší hodnocení.



Obrázek 3.2: Ukázka rozhraní aplikace Attendance

Attendance

Aplikace Attendance[1] stačí k jednoduchému vytváření skupin, jejich účastníků a vytváření událostí.

Obrázek 3.2 zobrazuje její rozhraní. K dispozici není žádný přehled dané skupiny. Uživatel tak nemůže jednoduše zjistit celkovou docházku účastníka, může pouze procházet po jednotlivých událostech. V konkrétní události lze pak filtrovat účastníky podle účasti.

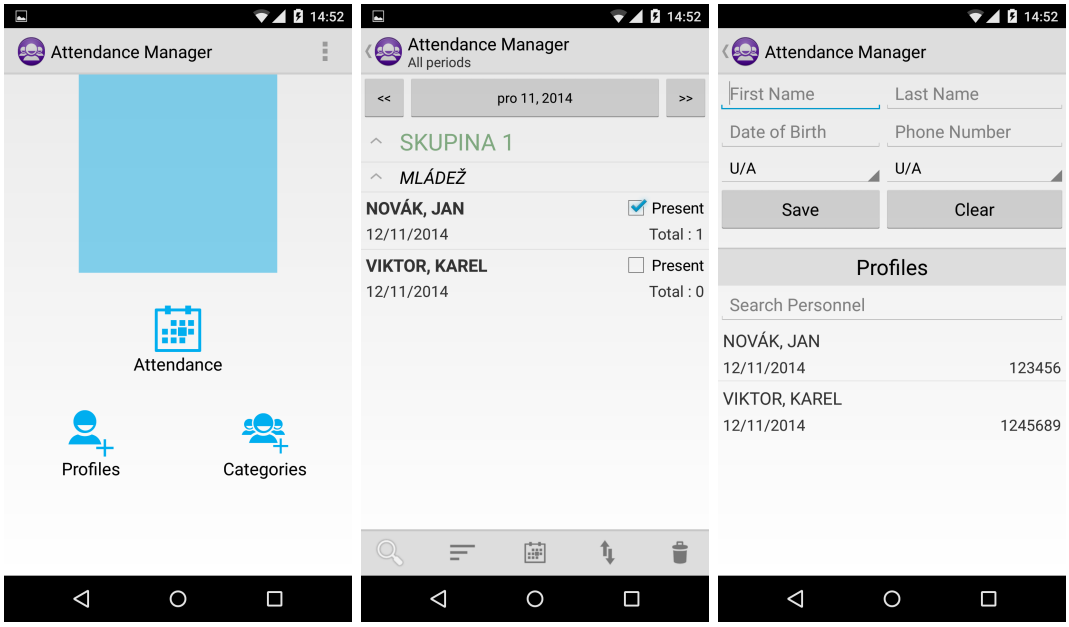
Aplikace trpí velkou chybou, kdy se při vytvoření události načtou do této data o účastnících. Při následné úpravě účastníků (změně jména účastníka, přidání či odebrání účastníka) se tyto změny už zpětně do vytvořené události nepromítnou.

Mezi další nedostatky patří zobrazování reklam, které lze odstranit za poplatek. Dále žádná nápověda, manuál pro užívání či absence češtiny.

K plusům aplikace patří možnost exportu dat do *CSV* souboru či přímo na webový disk *Google*.

Poslední aktualizace proběhla v polovině roku 2013 a dá se tak usuzovat, že již není v aktivním vývoji.

[1] https://play.google.com/store/apps/details?id=com.proj.attendance



Obrázek 3.3: Rozhraní aplikace AttendanceManager

AttendanceManager

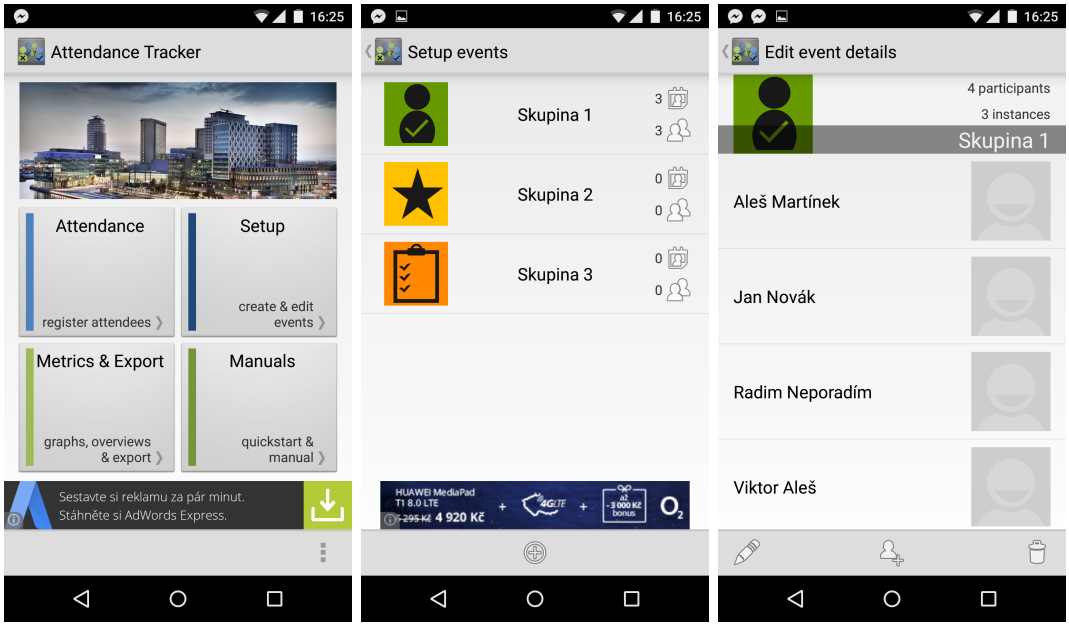
AttendanceManager[2] opět dostačuje k jednoduché správě docházky.

Používání aplikace sráží nedostatky. Uživatel musí vyplnit pole "datum narození"a "telefonní číslo"při vytváření účastníka. Seznam účastníků je globální, nerozlišený podle skupin a při větším počtu bude nepřehledný.

Opět není k dispozici žádný celkový přehled dané skupiny, pouze procházení po dnech (viz prostřední obrázek 3.3). Aplikace nerozlišuje dny kdy docházka není vedena a umožňuje pouze procházení pouze den po dni. Dále zobrazuje všechny skupiny najednou, čímž ještě více znepřehledňuje používání. Zadávání konkrétní účasti je řešeno pouhým zaškrtávacím tlačítkem, možnosti účasti jsou tak pouze "ano"či "ne". K dispozici není žádná nápověda či manuál a neobsahuje češtinu.

Od vydání v *GooglePlay* došlou pouze k dvěma minoritním aktualizacím a tak se dá předpokládat, že i vývoj této aplikace byl ukončen.

[2] https://play.google.com/store/apps/details?id=com.proappdesigns.attendancemanager



Obrázek 3.4: Úvodní obrazovka a nastavení AttendanceTrackeru

Attendance Tracker

AttendaceTracker[3] patří k propracovanějším aplikacím tohoto typu.

Na obrázku 3.4 je vidět jednoduchá úvodní obrazovka, dále nastavování skupin a správa jednotlivých účastníků ve skupině. Skupiny lze odlišovat kromě názvů i vestavěnými ikonami. Správa jednotlivých účastníků nabízí široké možnosti vytváření. Kromě účastníků "pouze v aplikaci"je možné importovat z kontaktů ze zařízení a z *Google Tabulek*. Účastníci také mohou mít přiřazenou fotku pro lepší identifikaci.

Obrázek 3.5 pak ukazuje už přímou správu účastí. Ve vybrané skupině lze vidět seznam událostí a u každé je vidět stručný přehled jaká byla účast. K dispozici je "přitomen", "nepřítomen", "nemocen"a "neznámo". Ke konkrétní účasti se dá přiřadit poznámka a taky se může označit štítkem "pozdní příchod". Události i účastníci se dají filtrovat.

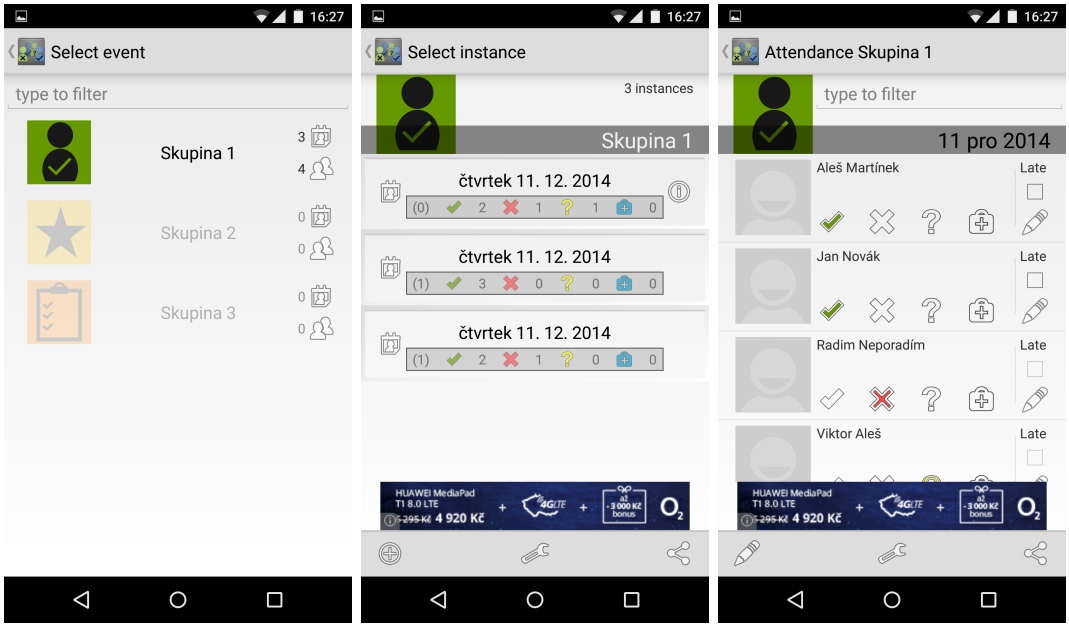
Události se řadí podle data, kdy byly vytvořeny a je to zároveň i její popisek. Toto datum se ale nedá upravit a tak není možné doplňovat docházku zpětně, což je velmi nepraktické.

Aplikace umožňuje generování přehledů a grafů ke skupině. Na obrázku 3.6 je vidět jednak přehled skupiny a také graf rozložení účastí. Generování probíhá na všech datech či na vybraném časovém rozpětí. Přehledy lze také exportovat do *Google Tabulek*.

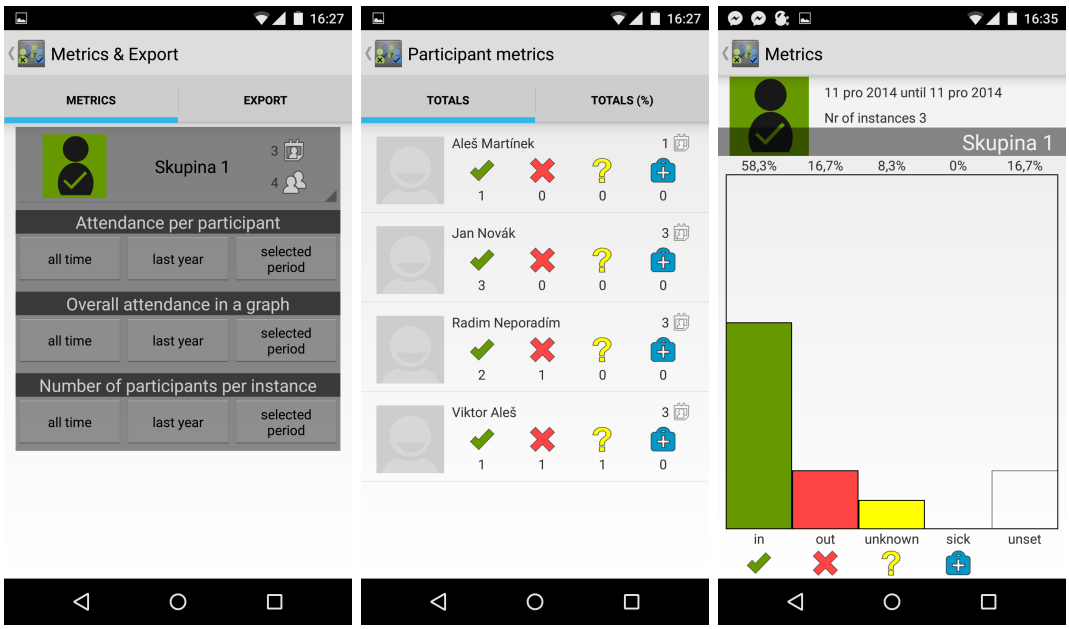
K dispozici je i záloha a obnovení všech dat aplikace. Tato data jsou v binární formě a tak se nedají využít jinde.

Na úvodním obrázku3.4 je i položka "Manuals", která obsahuje manuál a nápovědu k užívání. Díky aktivnímu vývoji a zlepšování má aplikace na GooglePlay přes 50 tisíc stažení. Zamrzí absence češtiny a prošpikování reklamami, které se dají vypnout nákupem v aplikaci.

[3] https://play.google.com/store/apps/details?id=peterman.apps.attendance



Obrázek 3.5: Attendance Tracker - Správa jednotlivých skupin a událostí



Obrázek 3.6: Attendance Tracker - Generování přehledů

Obrázek 3.7: Rozhraní aplikace Students

Students

Aplikace Students[4] se oproti předchozí řadí k nejhorším. Během testování se jí plnohodnotně nepodařilo zprovoznit, obsahuje spoustu neošetřených chyb, padá při používání diakritiky či při změně orientace displeje. Jediným jejím plusem je pokus o vzhledem unikátního prostředí, kdy vše imituje používání křídy a školní tabule (3.7).

K dispozici je manuál užití, možnost nastavit upomínky na opakující se události. Funguje zde i sdílení skupin (semestrů) mezi uživateli - jedná se ale pouze o poslání semestru druhému uživateli, ne o souběžné spolupráci na jednom semestru.

Aplikace neobsahuje češtinu (pouze angličtinu, němčinu a španělštinu), zobrazuje reklamy a umožňuje zálohu/obnovení dat na SD kartu.

[4] https://play.google.com/store/apps/details?id=com.gupte.shaunak.attendancetracker

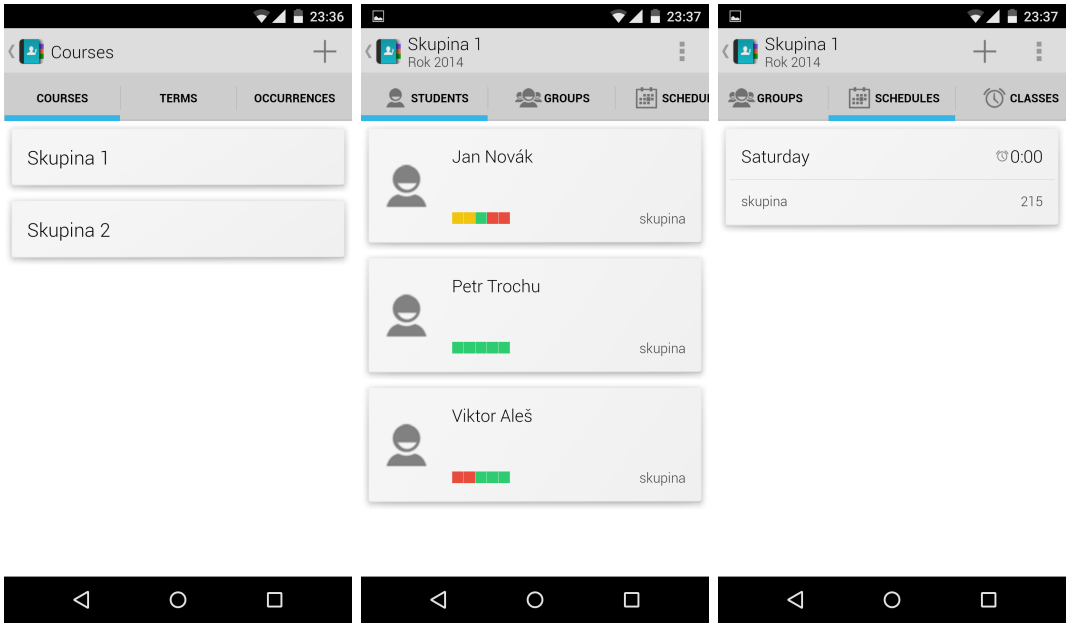
Attendance

Další povedenou aplikací je Attendance[5]. Bohužel si vynucuje rozdělení: *Předměty - Semestr - Zapsání předmětu v semestru*. Pro zapsaný předmět v semestru lze vytvořit několik seznamů účastníků. Dále se mu musí nastavit vyučovací hodiny, ty mohou být pravidelné i jednorázové. Po mírně delším nastavování se ale uživateli otevře přehledná a dobře použitelná aplikace. Na obrázku 3.8 zobrazuje obrazovky základního nastavení, obrázek 3.9 pak už konkrétní správu účastí.

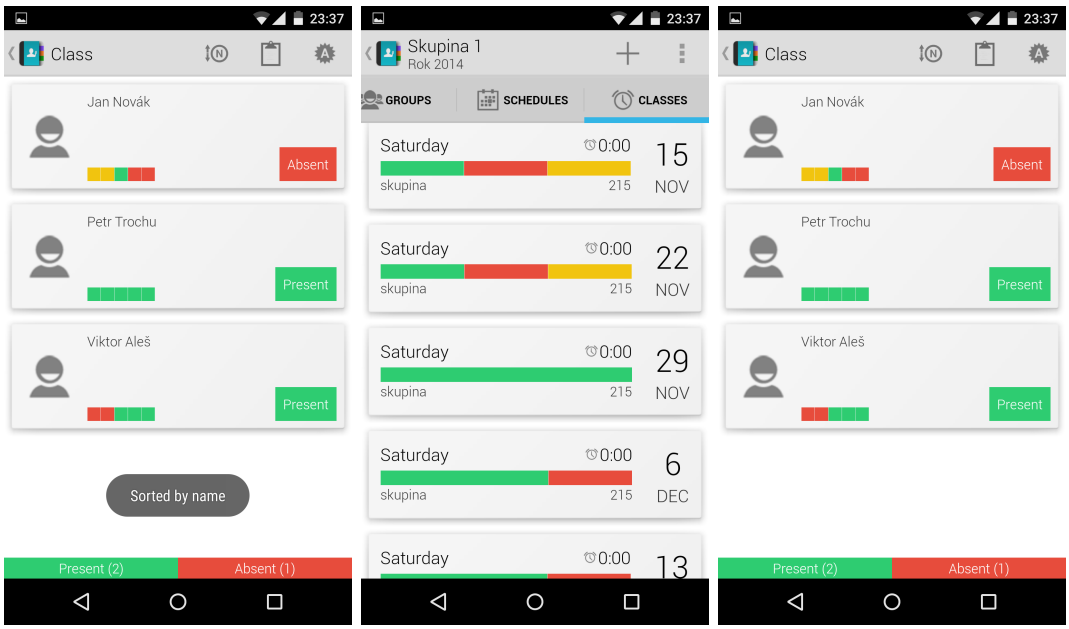
U účastníka může být nastaveno i uživatelské jméno či fotka. Vyučovacím hodinám se dá nastavit upomínka a zároveň vlastní možnosti účasti (mimo tradičních ano/ne/omluven). K dispozici je přehledný souhrn účastí, v kterém ale nelze nikterak vyhledávat či filtrovat. Dala lze z aplikace exportovat do *CSV* souboru.

Uživatelské rozhraní aplikace je čisté a přehledné, bez reklam a i českého jazyka. Prochází neustálým vývojem.

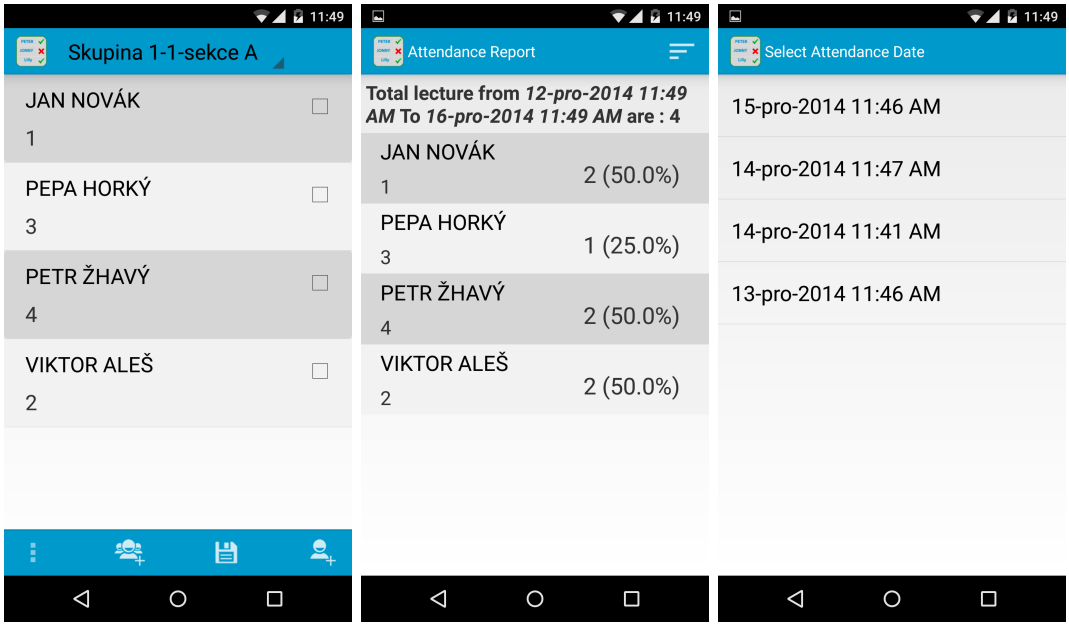
[5] https://play.google.com/store/apps/details?id=com.aor.attendance



Obrázek 3.8: Attendance - úvodní nastavení aplikace



Obrázek 3.9: Attendance - správa účastí členů



Obrázek 3.10: Rozhraní aplikace Easy Attendance School College

Easy Attendance School College

Easy Attendance School College[6] přistupuje k úvodnímu nastavení o něco jednodušeji. Stačí vytvořit skupinu/předmět a k ní přiřadit účastníky. Při vytváření nového účastníka mu musí uživatel zadat sám unikátní číslo v rámci všech skupin. Uživatel tak musí sám generovat primární unikátní klíč, což při spravování jedné dvou skupin se dá ještě zvládnout, při větším počtu účastníků se to může stát rychle velmi nepřehledné. Na obrázku 3.10 vlevo je pak vidět vybraná skupina a seznam jejich členů, která je připravena k zadání nové účasti. Pomocí zaškrtávacího pole lze určit zda-li účastník byl přítomen či ne. Po kliknutí na tlačítko uložit se zobrazí dialog pro vybrání data dané události. Plusové body, které aplikace získává na přímočarosti a rychlosti zadání nové události a jejích účastí, pak ale rychle ztrácí na zobrazení celkových přehledů.

Pokud chce uživatel vidět přehled skupiny, musí vybrat časové období od kdy do kdy má být přehled generován a poté je mu zobrazen pouze seznam celkových procentuálních účastí účastníků (3.10 uprostřed).

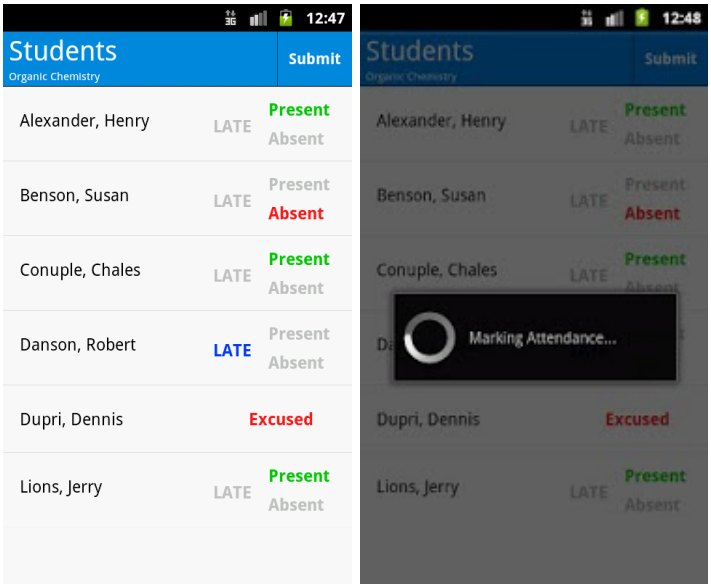
Aplikace tak nabízí sice rychlé zadávání nové účasti, ale zároveň nepřehledné celkové zobrazení. Účast je binární, pouze přítomen/nepřítomen. Není zde možnost žádných poznámek.

V placené verzi jsou navíc funkce pro export, import a sdílení dat, využívá se zde Google Disku.

V aplikaci i na jejích webových stránkách je rozsáhlá nápověda, jak ji správně používat a dokumentace.

Žádná verze neobsahuje reklamy ani český jazyk.

[6] https://play.google.com/store/apps/details?id=com.attendance.school

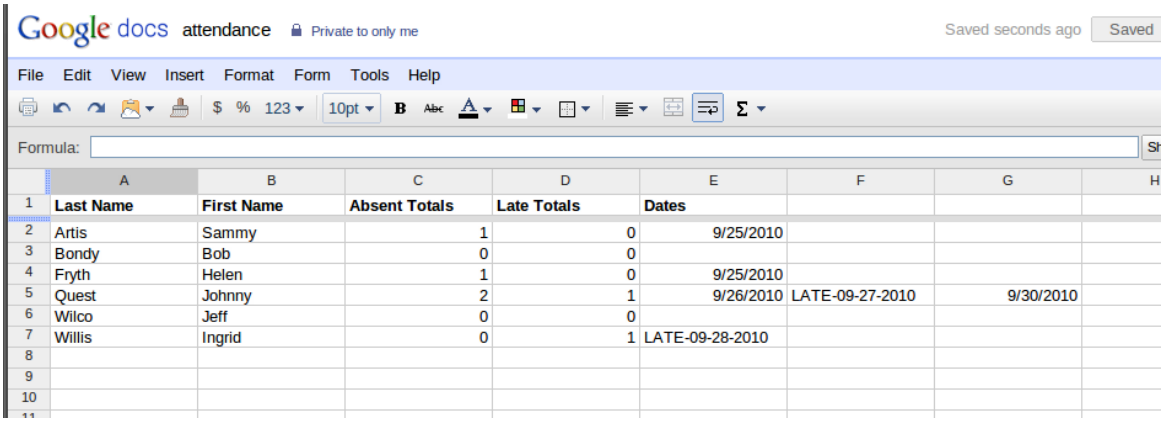


Obrázek 3.11: Účast - rozhraní

Účast

Poslední významnější aplikací v *GooglePlay* této kategorie je Účast[7] 3.11. Jedná se spíše o nadstavbu nad tabulkami *Google*.

Pro fungování aplikace je třeba aplikaci propojit s konkrétní tabulkou na webovém disku *Google*, která musí mít určitý formát. Obrázek 3.12 ukazuje tento formát a i jak pak dále aplikace do souboru ukládá data.



	A	B	C	D	E	F	G	H
1	**Last Name**	**First Name**	**Absent Totals**	**Late Totals**	**Dates**			
2	Artis	Sammy	1	0	9/25/2010			
3	Bondy	Bob	0	0				
4	Fryth	Helen	1	0	9/25/2010			
5	Quest	Johnny	2	1	9/26/2010	LATE-09-27-2010	9/30/2010	
6	Wilco	Jeff	0	0				
7	Willis	Ingrid	0	1	LATE-09-28-2010			
8								
9								
10								

Obrázek 3.12: Účast - formát tabulky

Bezpečnostním rizikem je nutnost zadání přihlašovacích údajů přímo do aplikace. Nevyužívá se zde tak tradičních přihlašovacích metod poskytnutých přímo *Googlem*, kdy by aplikace získala pouze přihlašovací token. Nicméně po zalogování se nabízí uživateli celkem

[7] https://play.google.com/store/apps/details?id=com.academics.attendance

jednoduché zadávání účasti. Vše je ale svázáno povinným formátem tabulky, uživatel zpětně nemůže upravovat účasti, pouze manuálně editovat tabulku přímo na GoogleDisku.

Dostupnost dat online je výhodou aplikace, ale zároveň i nevýhodou, když bez přístupu k internetu se k datům nedostanete.

3.3 Srovnání aplikací

Tabulka 3.1 zobrazuje srovnání aplikací, které byly zahrnuty do úvodní studie.

Vysvětlivky k jednotlivým položkám:

- **Správa skupin** - aplikace nabízí přehlednou a funkční správu skupiny, jejich přidávání, mazání a úpravu.
- **Správa členů** - aplikace nabízí přehlednou a funkční správu členů, jejich přidávání, mazání a úpravu.
- **Reklamy** - zobrazování reklam v aplikaci.
- **Import/Export** - možnost importování či exportování dat v nějaké formě
- **Platby v aplikaci** - aplikace obsahuje platby například pro odstranění zobrazování reklam, rozšířené funkce, ...
- **Čeština** - je podporován český jazyk.
- **Upomínky** - možnost nastavení upomínek pro události.
- **Aktivní vývoj** - aplikace byla v poslední době aktualizována, nebyl ukončen její vývoj a podpora.
- **Nápověda** - k dispozici je manuál či nápověda pro užití.
- **Záloha/obnovení** - data či nastavení aplikace je možno zálohovat a následně obnovit.
- **UI** - přehlednost, čistota uživatelského rozhraní, dodržení designových pokynů platformy Android, z části subjektivní.
  - 0 - nepřehledné.
  - jednoduché - aplikace je vyvedena v jednoduchém přehledném duchu.

Položka	Attendance	Attendance Manager	Atendance Tracker	Students	Attendance (2)	EASC	Účast
Správa skupin	1	1	1	0	1	0	1 (v rámci sešitu)
Správa členů	1	1	1	0	1	0	1 (v rámci sešitu)
Reklamy	1	1	1	0	0	0	0
Import/export	Export	0	Export	0	0	1 (v placené verzi)	import ze sešitu
Platby v aplikaci	0	0	1	0	0	1	0
Čeština	0	0	0	0	0	0	0
Upomínky	0	0	0	1	1	0	1
Aktivní vývoj	červen 2014	ne	1	1	1	0	1
Nápověda	0	0	1	1	1	0	1
Záloha Obnovení	1	0	1	1	1	0	0
UI	Holo	0	Holo	Jednoduchý	Holo+	Jednoduchý	Holo
Synchronizace	0	0	0	0	0	0	1 (v rámci sešitu)

Tabulka 3.1: Srovnání stávajících aplikací

  - Holo - aplikace navíc dodržuje designové pokyny vydané s verzí platformy Android 4.0 Ice Cream Sandwich .

- **Synchronizace** - k dispozici je online synchronizace dat.

3.4 Vyhodnocení

Z tabulky 3.1 je vidět, že žádná z aplikací nenabízí plnohodnotnou synchronizaci dat s cloudem. Každý uživatel si tak spravuje sám svá vlastní data a nemá pohodlnou možnost jak tato sdílet pro čtení s ostatními.

Z těchto důvodů a důvodů uvedených v předchozí části jsem se rozhodl přistoupit k vlastnímu řešení. Vlastní řešení se bude skládat z aplikace pro platformu Android, která by měla umožnit jednoduše a efektivně zadávat účasti. Druhou částí řešení pak bude webové rozhraní, která hlavně musí umožňovat přehledné zobrazení docházky. Měla by také poskytovat editaci těchto dat.

Kapitola 4

Úvodní studie

Tato kapitola se věnuje analýze požadavků na systém a návrhu vlastního řešení.

4.1 Seznam požadavků

V následující části jsou rozebrány jednotlivé požadavky, které by projekt měl splňovat. Požadavky popisují jaké funkce a vlastnosti má systém mít. Dělí se na *funkční*, které určují jaké funkce má systém mít a *nefunkční*, které popisují jaké vlastnosti má projekt mít. Popis těchto požadavků musí být srozumitelný jak zadávající straně, tak osobám co mají projekt vypracovat. Typickým představitelem funkčního požadavku je například "systém bude umožňovat zadávání nových objednávek". Příkladem nefunkčního požadavku je nutnost použití určité technologie či platformy.

Dále se požadavky dělí dle priority:

- **Vysoká** - Pro funkčnost daného projektu je splnění nezbytné a jeho realizace by měla být splněna přednostně.
- **Střední** - Projekt by měl daný požadavek splňovat.
- **Nízká** - Požadavek s nízkou prioritou obsahuje funkci či vlastnost, která pro projekt není vyloženě důležitá a jen ho doplňuje. Jejich splnění není tak povinné, může se odložit pro budoucí vývoj.

4.1.1 Funkční požadavky

- **Zobrazení skupin** - systém bude umožňovat zobrazení jednotlivých skupin.
  - Priorita - Vysoká
- **Zobrazení členů** - systém bude umožňovat zobrazení členů.
  - Priorita - Vysoká
- **Zobrazení událostí** - systém bude umožňovat zobrazení událostí.

  - Priorita - Vysoká

- **Správa skupin** - systém bude umožňovat spravovat jednotlivé skupiny.
  - Priorita - Vysoká
  - Oprávnění uživatelé mají mít přístup k funkcím pro přidávání nových skupin a k úpravě a mazání již stávajících.
- **Správa členů** - systém bude umožňovat spravovat jednotlivé členy.
  - Priorita - Vysoká
  - Do již existujících skupin může uživatel přidávat nové členy, které pak může upravovat a mazat.
- **Správa událostí** - systém bude umožňovat spravovat jednotlivé události.
  - Priorita - Vysoká
  - Do již existujících skupin může uživatel přidávat nové události, které pak může upravovat a mazat.
- **Správa účastí** - systém bude umožňovat zadávat účasti.
  - Priorita - Vysoká
  - Do konkrétní události může uživatel k jednotlivým účastníkům zadávat účast.
- **Synchronizace dat** - systém bude plně synchronizovat veškerá dat mezi zařízeními.
  - Priorita - Vysoká
  - Data skupin, členů, událostí a jejich účastí budou plně synchronizovaná mezi zařízeními a uložena na centrálním úložišti.
- **Zobrazení přehledů** - systém bude umožňovat přehledně zobrazit informace o jednotlivých skupinách.
  - Priorita - Střední
  - Zadaná data uživatele bude možno přehledně zobrazovat.
- **Zobrazení průměrné účasti členů** - systém bude umožňovat přehledné zobrazení procentuální účasti jednotlivých členů.
  - Priorita - Nízká
  - U jednotlivých členů by mělo být zobrazena procentuální účast na událostech pro jednoduchou kontrolu docházky člena.
- **Zobrazení grafů účastí členů** - systém bude umožňovat zobrazit grafový přehled účasti členů
  - Priorita - Nízká
  - Zobrazení grafů účastí může opět sloužit k analýze docházky a trendů.

- **Zobrazení průměrné účasti členů na jednotlivých událostech**
  - Priorita - Nízká
  - Pro sledování vývoje účasti na událostech je vhodné zobrazovat procentuální účast na jednotlivých událostech.
- **Import dat** - systém bude umožňovat importovat vlastní data.
  - Priorita - Nízká
  - Pro dlouhodobější práci (a pro ladění během vývoje) je možnost importovat data důležitá. Není to však funkce, bez které by se běžný uživatel neobešel.
- **Export dat** - systém bude umožňovat exportovat data.
  - Priorita - Nízká
  - Pro dlouhodobější práci (a pro ladění během vývoje) je možnost exportovat veškerá uložená data důležitá. Není to však funkce, bez které by se běžný uživatel neobešel.

4.1.2 Nefunkční požadavky

- **Mobilní klient pro Android** - Systém bude obsahovat vlastní mobilní aplikaci pro platformu Android.
  - Priorita - Vysoká
  - K dispozici musí být aplikace pro zařízení s OS Android. Aplikace musí zprostředkovávat funkční požadavky z předchozí kapitoly.
- **Webové portál** - Systém bude zahrnovat webový server s aplikací.
  - Priorita - Vysoká
  - K dispozici musí být webová aplikace nasazená na serveru. Aplikace musí zprostředkovávat funkční požadavky z předchozí kapitoly.
- **Možnost prohlížet offline** - Systém by měl být přiměřeně použitelný i bez připojení k internetu.
  - Priorita - Střední
  - I přes předpoklad, že aplikace bude nejčastěji využívána s připojením na databázový server, musí být aplikace v přiměřeném stylu použitelná i bez připojení.
- **Přehledné prostředí** - Uživatelské rozhraní systému by mělo být přehledné.
  - Priorita - Střední
  - UI by mělo být přehledné a jednoduché. Mobilní klient by měl dodržovat designové doporučení pro platformu Android.
- **Rychlost a nenáročnost** - Systém by měl být svižný, nenáročný a s nízkou odezvou

  - Priorita - Střední
  - UI by mělo být reaktivní, bez viditelných prodlev. U mobilního klienta by nemělo docházet k varování o neodpovídání aplikace. Komunikace s klientem by měla být efektivní.

- **Optimalizace pro různé typy zařízení** - Mobilní klient by měl být optimalizován pro různé velikosti a jemnosti displeje zařízení.
  - Priorita - Nízká
  - Pro Android existuje velká škála zařízení s různými velikostmi a jemnostmi displeje. Aplikace by měla být optimalizována tak, aby pokrývala většinu běžně používaných kombinací.

- **Bezpečnost** - Data můžou být upravovaná pouze oprávněným uživatelem.
  - Priorita - Střední
  - Systém bude obsahovat autentizaci a autorizaci uživatele před povolením úprav dat.

- **Použitelnost** - Systém bude obsahovat nápovědu či manuál k použití.
  - Priorita - Střední

- **Opensource knihovny** - Systém bude využívat veřejně dostupných dodatečných knihoven.
  - Priorita - Vysoká

Role a užití

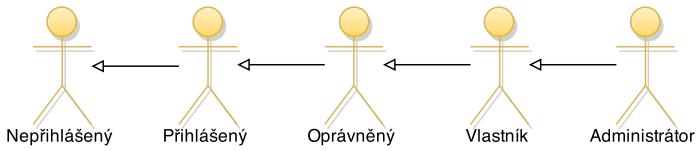
V systému bude několik typů rolí, pod kterými může uživatel vystupovat. Seznam rolí na obrázku 4.1 ilustruje hierarchické rozložení rolí.

Nepřihlášený uživatel

Základní rolí pod kterou uživatel vstupuje do systému je *Nepřihlášený uživatel*. Jeho možnosti jsou prohlížení veřejných dat bez manipulace s nimi a přihlášení se.

Přihlášený uživatel

Přihlášený uživatel může prohlížet jemu sdílená data a zakládat svá vlastní.



Obrázek 4.1: Seznam rolí

Oprávněný uživatel

Oprávněný uživatel může spravovat jeho a jemu sdílená data.

Vlastník

Vlastník má rozšířené možnosti správy vlastních dat.

Administrátor

Administrátor může upravovat veškerá data.

4.2 Návrh řešení

Projekt bude rozdělen na tři části.

- Cloudové úložiště s databází - obsahuje veškerá aktuální data.
- Mobilní klient - instalovatelná aplikace pro Android.
- Webový server - přístup k aplikaci skrz webový prohlížeč.

Databázový server

Hlavní funkcí systému má být synchronizace a přístupnost veškerých dat. Je třeba aby jakákoliv změna dat byla distribuovaná ideálně ihned dalším zařízením a aby tato data byla zároveň ukládána na cloudovém úložišti.

První návrh řešení

Jako první možnost se nabízí vyvinout vlastní řešení, které bude obstarávat jak ukládání dat na jednotlivých zařízeních, tak synchronizaci dat s hlavním úložištěm.

Pro ukládání dat na mobilním klientu byla zvolena osvědčená knihovna *GreenDAO*[1], která staticky generuje třídy do projektu, které jsou mapovány přímo do *SQLite* databáze na zařízení. Většina knihoven pro *ORM* práci s databází na Androidu využívá anotací, které jsou ale zbytečně náročné na procesor hlavně v nižších verzích Androidu[7]. *GreenDAO* využívá předgenerování si tříd, které jsou pak přímo zakomponovány do aplikace. Při spuštění aplikace tak nezatěžuje procesor nutností zpracovávat anotace.

Ukládání při použití webové aplikace se data rovnou ukládají do databáze na serveru. Použit by byl *Google App Engine*, který nabízí vlastní škálovatelné databázové řešení.

[1] domovská stránka knihovny GreenDAO

Druhou částí je pak zajištění aktualizace dat na jednotlivých zařízeních. Je potřeba, aby uživatel vždy pracoval s aktuální verzí dat a aby se na server ukládala co nejnovější data. Možnosti jaké mohou nastat při použití mobilního klienta:

- Prvotní zadání dat - uživatel nemá zatím žádná data, při práci s aplikací se mu ukládají přímo v zařízení a rovnou i na serveru.
- Klient má stejná data jaká jsou na serveru - stejný případ jako první možnost, klient pracuje, ukládá si data a posílá je na server.
- Klient má novější data než jsou na serveru - uživatel pracovat s aplikací v offline režimu, má novější data než jsou uložena na serveru. Data se nejprve aktualizují na serveru a potom s nimi klient může opět manipulovat.
- Klient má starší data než jsou na serveru - nejprve se provede aktualizace dat u klienta a poté s nimi může opět manipulovat.
- Klient č.1 aktualizuje data u sebe. Klient č.2 poté u sebe a i na serveru. Klient č.1 se připojí k serveru. Která data teď mají být aktuální?

Z výše uvedeného vyplývá, že problematika synchronizace a distribuce aktuálních dat je rozsáhlá. Po přečtení materiálů [9] [4] k této problematice jsem vypracoval prvotní návrh 4.2 jak se s tím vypořádat.

Knihovna *EventStore*[2] byla vytipována jako vhodná pro realizace vlastního řešení. Implementace se zdála ale zbytečně komplikovaná a tak bylo třeba se podívat po vhodnějším řešení.

Druhý návrh řešení

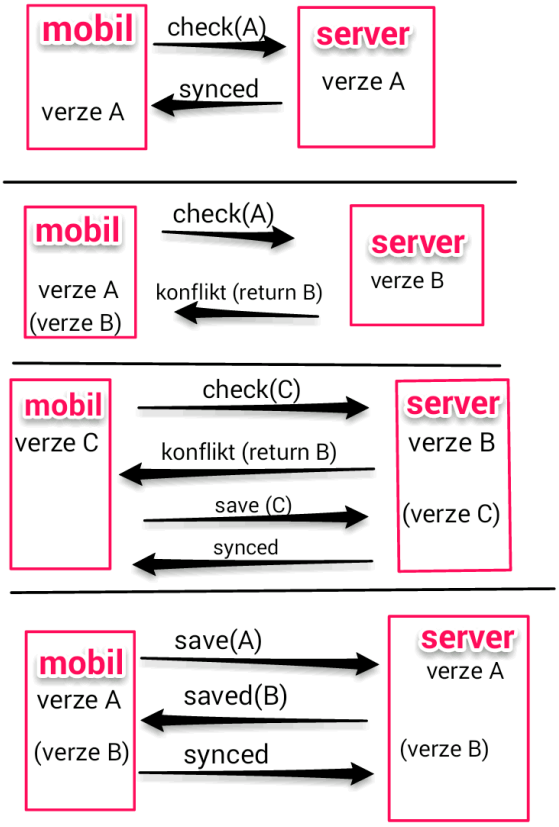
Druhou variantou bylo použití produktu, který veškeré problémy uvedené v prvním návrhu řeší. Jako nejvhodnější produkt byla vybrána služba *Firebase*, jejíž funkce je detailněji popsána v části Databázový backend 2.2. *Firebase* jednak nabízí cloudovou databázi a druhak knihovny pro vytvoření klientů, které k této databázi mají přístup. Data jsou klientům efektivně distribuována, synchronizována a knihovny taktéž zajišťují offline používání databáze v zařízení.

Tento návrh byl pro vypracování zvolen.

Webový server

Firebase zprostředkovává přístup k databázi skrz vlastní javaskriptovou knihovnu. Proto není potřeba zavádět žádný server, který by generoval dynamické stránky. Aplikace bude statická webová stránka jejíž funkce bude obsluhovat javaskriptová část přímo u klienta. Jako hosting těchto souborů bude využit hosting poskytovaný přímo službou *Firebase* (viz část Firebase hosting 2.2.4).

[2] EventStore - http://geteventstore.com/

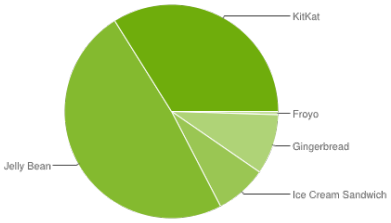


Obrázek 4.2: První raný návrh řešení databáze

Mobilní klient

První cílová skupina je 27 trenérů našeho týmu. Z průzkumu vyplynulo, že pouze dva nepoužívají mobilní zařízení s *OS Android*. Proto jako vývojová platforma pro mobilní aplikace byl zvolen právě *Android*. S touto platformou mám také největší zkušenosti. *Aplikace* pro *iOS* od *Apple* není v plánu této práce.

Android se neustále vyvíjí a tak se často objevují nové verze, které přinášejí nové funkce do systému. Ne všechny zařízení jsou na novější verze aktualizována. Je třeba aby výsledná aplikace využívala funkcí, které jsou dostupné na většině verzí a zařízení. Na obrázku 4.3 je graf který zobrazuje pokrytí jednotlivými verzemi. Tabulka 4.1 zobrazuje procentuální rozdělení verzí. Data [6] jsou získána z přístupů do *GooglePlay* během 7mi denní doby.



Obrázek 4.3: Rozdělení verzí Androidu k 1.12.2014

Verze	Označení	Verze API	Zastoupení
2.2	Froyo	8	0.5%
2.3.x	Gingerbread	10	9.1 %
4.0.x	Ice Cream Sandwich	15	7.8 %
4.1.x	Jelly Bean	16	21.3 %
4.2.x	Jelly Bean	17	20.4 %
4.3	Jelly Bean	18	7.0 %
4.4	KitKat	19	33.9 %

Tabulka 4.1: Rozdělení verzí Androidu k 1.12.2014

Přihlašování

Uživatele je potřeba v rámci systému autentizovat. Jedna varianta je opět vyvinutí vlastního řešení přihlašování uživatele, zabezpečení komunikace a vedení si informací o přihlášených uživatelích na serveru.

Druhou variantou pak použití již existujícího řešení. *OS Android* je úzce svázán s použitím *Google* účtu. Již při aktivaci nového zařízení je uživatel nucen zadat svůj vlastní (či vytvořit nový) *Google* účet, pomocí kterého pak může přistupovat do obchodu s aplikacemi *GooglePlay* a zpřístupňuje mu i další důležité funkce. Zároveň Google poskytuje službu[3] a platformu pro vývojáře[4], pomocí které se dá do vyvíjené aplikace velice snadno implementovat autentizace a autorizace skrz zmíněný *Google* účet.

Přihlašování pomocí Google účtu:

- **Přihlášení** - Uživatel se nejprve musí ve svém zařízení přihlásit ke Google účtu.
  - Zařízení s *OS Android* - zde bývá uživatel běžně přihlášen.
  - Prohlížeč *Chrome* od *Google* - v rámci celé aplikace může být opět uživatel přihlášen pod *Google* účtem.
  - Ostatní zařízení/Uživatel není k *Google* účtu přihlášen - otevře se webová stránka s výzvou k přihlášení. V prohlížeči *Chrome* a v zařízení *Android OS* s nainstalovanou aplikací *Google+* se zobrazí nativní dialogové okno pro přihlášení.
- **Autorizace** - Pokud je uživatel přihlášen pod svým *Google* účtem je potřeba ho autorizovat s vlastní aplikací. Probíhá to opět pomocí dialogu na kterém jsou uvedeny

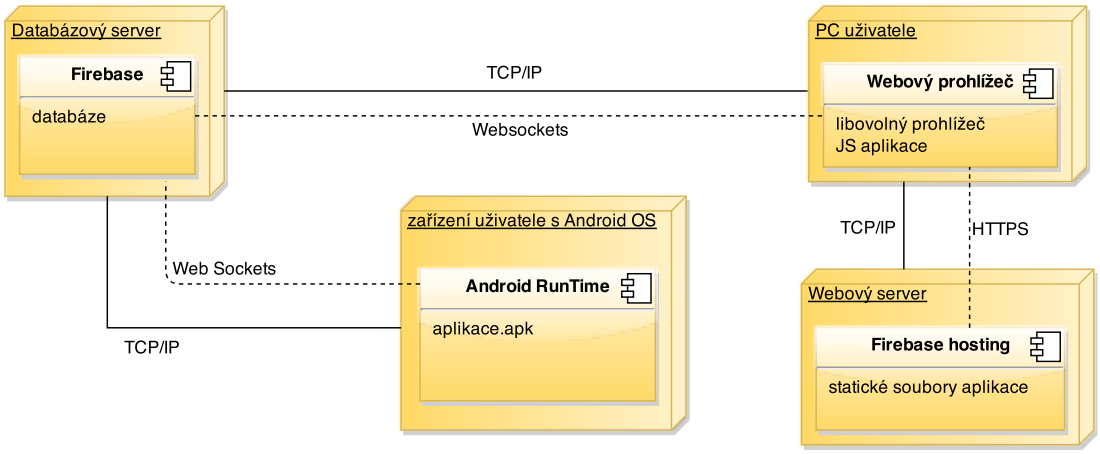
[3] dokumentace Google Play Services
[4] Google+ Platform on Android - dokumentace Google+ platformy pro Android

informace o aplikaci a jaká práva vyžaduje. Pokud už uživatel někdy autorizoval aplikaci, pak rovnou pokračuje dál.

- **Použití kdekoliv** - Pokud uživatel aplikaci autorizoval, pak jí může používat na jakémkoliv zařízení kde je pod daným účtem přihlášen.

Autorizace skrz *Google* účet lze využít i při práci s Firebase servery, proto bylo zvoleno toto řešení.

4.2.1 Finální návrh



Obrázek 4.4: Diagram návrhu nasazení

Na obrázku 4.4 je znázorněn finální diagram návrhu nasazení. Obsahuje databázový server s naší databází, ke kterému se připojuje jednak Android zařízení s nainstalovanou aplikací a druhak klient ze svého prohlížeče skrz JS aplikaci, jejíž data jsou načtena s hostinu *Firebase*.

4.3 Rozdělení na procesy

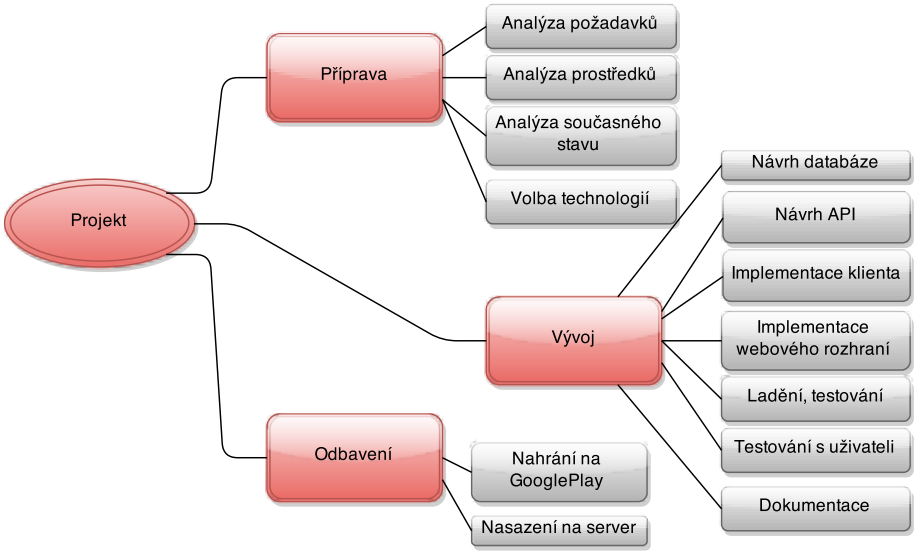
Obr. 4.5 zobrazuje rozdělení projektu na dílčí procesy.

4.4 Analýza rizik

Při realizaci každého projektu existují rizika, která mohou ovlivnit jeho úspěšnost. Tyto rizika je potřeba předem identifikovat, odhadnout pravděpodobnost jejich nastání a ohodnotit nebezpečnost.

4.4.1 Technická náročnost projektu

- Řešení některých problémových částí projektu může být komplikované a vysoko nad rámec tradiční práce. Může obsahovat implementaci složitých postupů či algoritmů a zasahovat nad rámec studijního zaměření a znalostí.



Obrázek 4.5: Rozdělení na procesy

- Nebezpečnost - Vysoká
- Pravděpodobnost - 20 %
- Efekt - Projekt se nepodaří realizovat.

4.4.2 Časová náročnost projektu

- Projekt i před jednoduchost dílčích řešení bude i tak časově náročný.
- Nebezpečnost - Vysoká
- Pravděpodobnost - 30 %
- Ač jednotlivé části projektu můžou být jednoduché na zpracování, jejich celkový počet může navýšit časovou náročnost nad původní plánovaný rozsah. Je potřeba si vše vhodně rozvrhnout v čase.

4.4.3 Ztráta dat

- Při realizace projektu dojde ke ztrátě dat. Může k ní dojít při hardwarové chybě (selže pevný disk, ...) či při softwarovém selhání (data nebudou uložena, pád aplikace, ...).
- Nebezpečnost - Vysoká
- Pravděpodobnost - 20 %
- Ke snížení tohoto rizika pomůže použití vhodných metod zálohování dat. Například uložením celého projektu do adresáře cloudového úložiště, který se automaticky synchronizuje či pravidelným přispíváním změn pomocí verzovacího nástroje.

4.4.4 Neznalost a zkušenosti s technologiemi

- Technologie zadané v požadavcích a vybrané technologie nebudou vhodné pro realizaci projektu. Nezkušenost s novými technologiemi zpomalí vývoj.
- Nebezpečnost - Vysoká
- Pravděpodobnost - 5 %
- Při vlastním zadání projektu je toho riziko nepravděpodobné. Technologie budou pravděpodobně zvolené známé či takové, které si chci osvojit.

4.4.5 Nesplnění požadavků

- Zadané požadavky nebudou všechny splněny či u nich dojde k odklonu o původního zadání.
- Nebezpečnost - Střední
- Pravděpodobnost - 15 %
- U méně prioritních požadavků může dojít k drobným změnám v pochopení a nemusí to mít na výsledek velký vliv. Odložení požadavků s nízkou prioritou na budoucí práci také projekt nemusí výrazně ovlivnit.

4.4.6 Bezpečnost

- Budou použity nedostatečně otestované a zabezpečené technologie. Výsledný systém nebude potřebně otestován a může dojít k odcizení či manipulaci s daty.
- Nebezpečnost - Vysoká
- Pravděpodobnost - 15 %
- Technologie by měly být zvoleny osvědčené a otestované. Přístup do systému a přidělení práv by mělo být dostatečně kontrolováno.

4.4.7 Nepřehledný vývoj

- Výsledný projekt by měl být řádně dokumentován a zbaven veškerých testovacích, ladících funkcí a výpisů. Kód by měl být přehledně organizován a nemělo by docházet k velkým změnám bez řádného testování a pravidelného přispívání do verzovacího nástroje.
- Nebezpečnost - Nízká
- Pravděpodobnost - 20 %

4.4.8 Vhodnost a použitelnost vývojových nástrojů

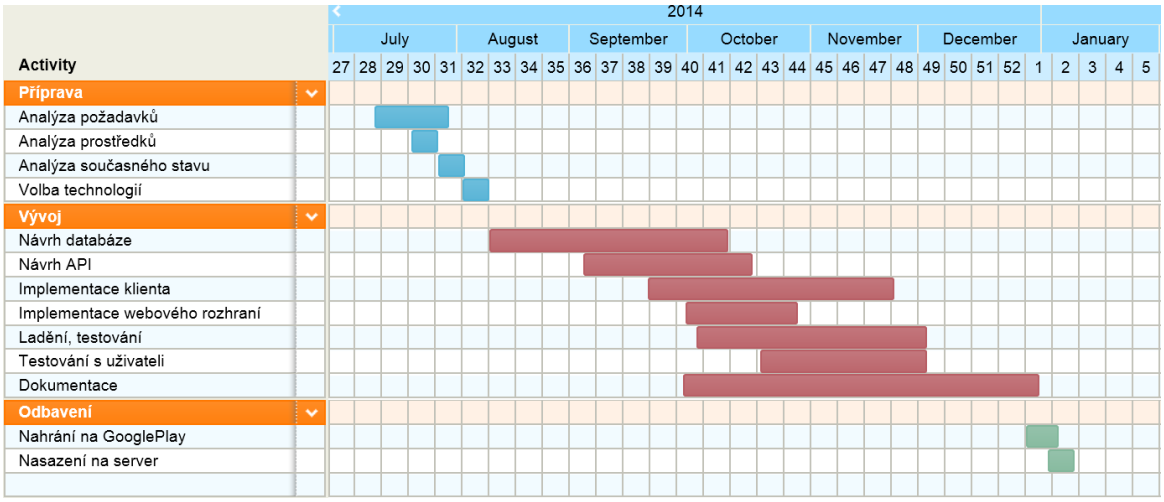
- Volba nevhodného vývojového nástroje může realizaci projektu velmi ztížit.
- Nebezpečnost - Střední
- Pravděpodobnost - 20 %
- Volbou vhodného osvědčeného nástroje lze naopak implementaci výrazně usnadnit. Většina kvalitních *IDE* obsahuje funkce pro refaktorizaci kódu, generování běžně používaných částí a možnost vlastních klávesových zkratek. Často také obsahují různé testovací nástroje, vestavěné servery a další.

4.4.9 Nedostatečné testování produktu

- Produkt nebude otestován pro všechny možné kombinace použitých zařízení a jejich konfigurací
- Nebezpečnost - Střední
- Pravděpodobnost - 35 %
- Uživatel má na výběr velké množství zařízení na kterých bude systém provozovat. Na *PC* může provozovat několik *OS* a na nich zase několik webových prohlížečů. Každý se pochopitelně chová jinak. U mobilní zařízení existuje také několik variant *OS* a *HW* konfigurací. Obsáhnout při testování všechny možné kombinace není z časového hlediska reálné a tak je potřeba otestovat produkt na hlavních majoritních kombinacích.

4.5 Harmonogram

Rozdělení projektu na procesy a naplánování těchto v čase ilustruje obrázek 4.6. Po úvodní analýze následuje hlavní část a sice vývoj, který zabere většinu času.



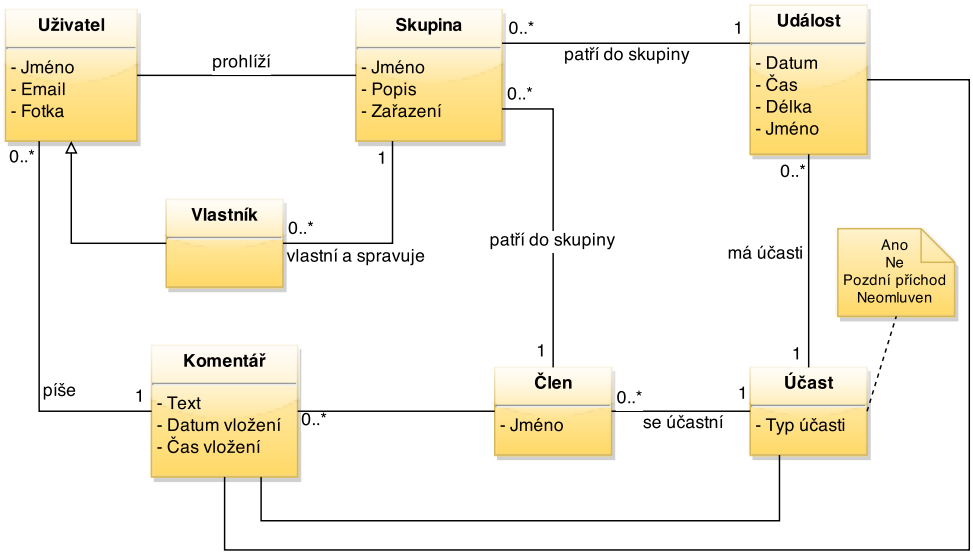
Obrázek 4.6: Časový harmonogram procesů

Kapitola 5

Analýza

V této části je popsána analýza systému. Je zde zaznamenán datový konceptuální model a rozebrány jednotlivé případy užití systému. Případy užití jsou popsány jednak slovně a také znázorněny na příslušných diagramech *UML*. Závěr kapitoly se věnuje analýze tvorby uživatelského rozhraní pro mobilní klient i pro webovou část a navrhuje rozhraní hlavní části aplikace.

5.1 Datový konceptuální diagram



Obrázek 5.1: Konceptuální datový model

Diagram 5.1 zachycuje systém z pohledu významu jednotlivých částí. Zobrazuje pouze nejnutnější atributy bez datových typů a vztahy mezi entitami.

5.1.1 Popis entit

Uživatel

Každá osoba vstupující do systému má práva pro konzumaci obsahu. Pokud má osoba vytvořený účet, pak se do systému může přihlásit a obsah i vytvářet. Identifikovaný uživatel má atributy:

- Jméno - celé jméno uživatele
- Email - identifikační email uživatele
- Fotka

Vlastník

Vlastník je vyšší role uživatele systému určena pro správu vlastních dat.

Skupina

Skupina je hlavní spravovanou a vstupní entitou systému. Veškerá data o členech a událostech rozděluje do logických celků.

- Jméno - název skupiny
- Popis - popis skupiny
- Zařazení - logické zařazení skupiny

Člen

Entita člen představuje konkrétní osobu u které chceme vést docházku.

- Jméno - jméno člena

Událost

Událost představuje konkrétní výskyt časově ohraničené akce u které chceme zaznamenat účast či neúčast členů.

- Datum, Čas - časový údaj začátku akce
- Délka - délka trvání akce
- Jméno - popis akce

Účast

Slouží k určení zda daný člen na dané události byl či nebyl. Kromě základních hodnot *přítomen* a *nepřítomen* může nabývat i dalších vlastních hodnot.

- Typ účasti - Ano, Ne, Pozdní příchod, Neomluven, ...

Komentář

Komentáře slouží k vkládání poznámek o jednotlivých událostech/členech či výskytech účasti. Ke každé instanci entity z tohoto výčtu může uživatel přidat několik komentářů.

- Text - znění komentáře
- Datum a Čas vložení - k chronologickému řazení

5.1.2 Popis vztahů mezi entitami

Uživatel - Skupina

Mezi *uživatelem* a *skupinami* je vztah, který slouží k prohlížení dat *skupin*.

Vlastník - Skupina

Vlastník může vlastnit neomezeně *skupin*. *Skupina* patří k jednomu *vlastníkovi*.

Skupina - Událost

Ke jedné *skupině* patří *událostí*. Jedna *událost* musí patřit ke konkrétní *skupině*.

Skupina - Člen

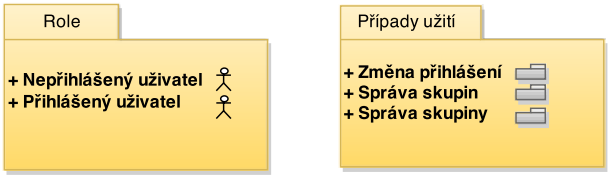
Člen musí patřit k jedné *skupině*. *Skupina* může mít několik *členů*.

Člen - Účast - Událost

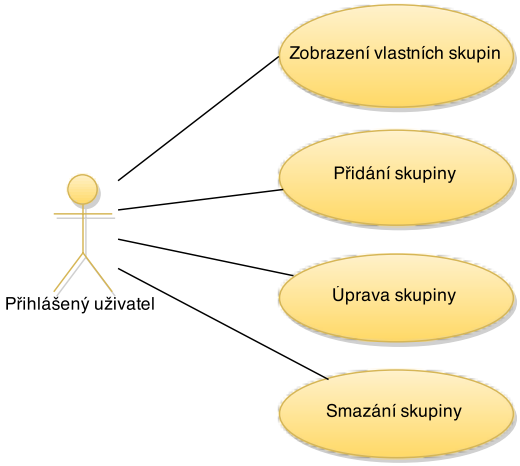
Jeden *člen* na jedné *události* musí mít vedenou pouze jednu *účast*. *Členové* i *události* můžou mít více *účastí*.

Uživatel - Komentář - Člen/Účast/Událost

Jeden *člen* může vkládat neomezeně *komentářů*. Jeden *komentář* pak musí patřit k jednomu z výčtu *Člen/Účast/Událost*.



Obrázek 5.2: Model skupin případů užití



Obrázek 5.3: Diagram případů užití pro správu skupin

5.2 Popis případů užití

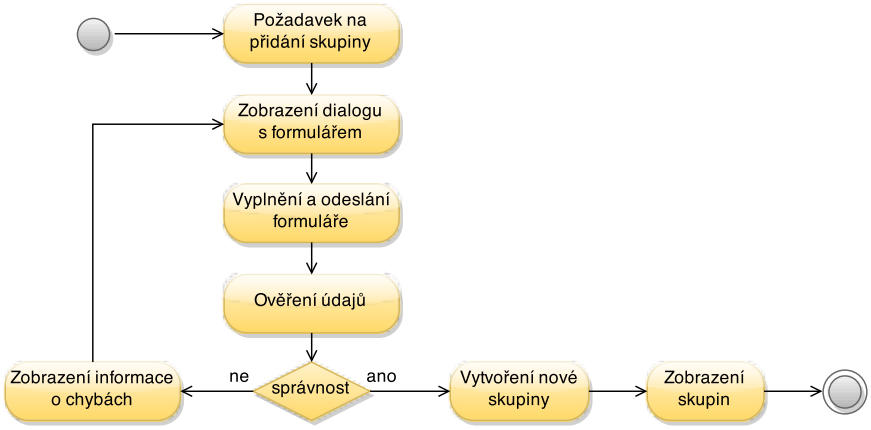
Obrázek 5.2 zobrazuje rozdělení případů užití do logických skupin a seznam rolí. Další část se věnuje jednotlivým skupinám detailněji.

5.2.1 Případy užití správy skupin

Na obrázku 5.3 jsou zobrazeny případy užití při správě skupin uživatele. Hlavním užitím je *zobrazení vlastních skupin*. A dále trojice užití pro manipulaci s konkrétními skupinami *přidání/úprava/smazání*.

Zobrazení vlastních skupin

- **Popis:** Uživatel chce zobrazit svoje spravované skupiny.
- **Vstup:** Přihlášený uživatel.
- **Hlavní scénář:**
  1. Uživatel má vlastní skupiny.
  2. Systém zobrazí jeho skupiny.
- **Vedlejší scénář:**



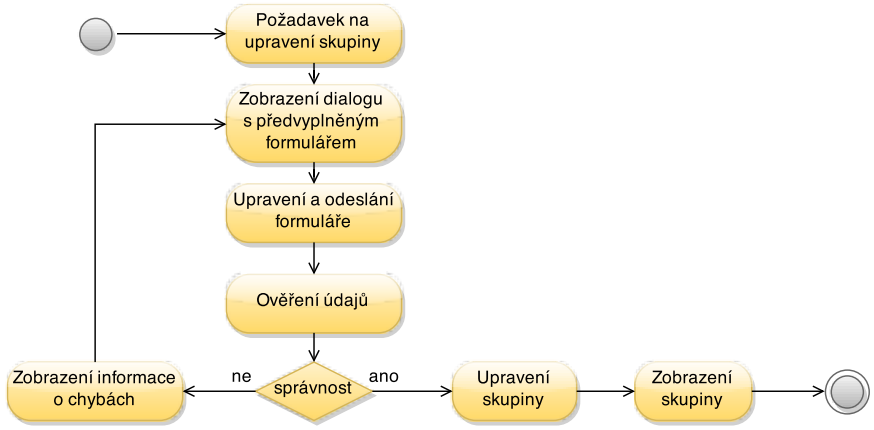
Obrázek 5.4: Případ užití - přidání skupiny

1. Uživatel nespravuje žádné skupiny.
2. Systém zobrazí výzvu k vytvoření nové skupiny.

- **Výstup:** Zobrazení spravovaných skupin.

Přidání skupiny

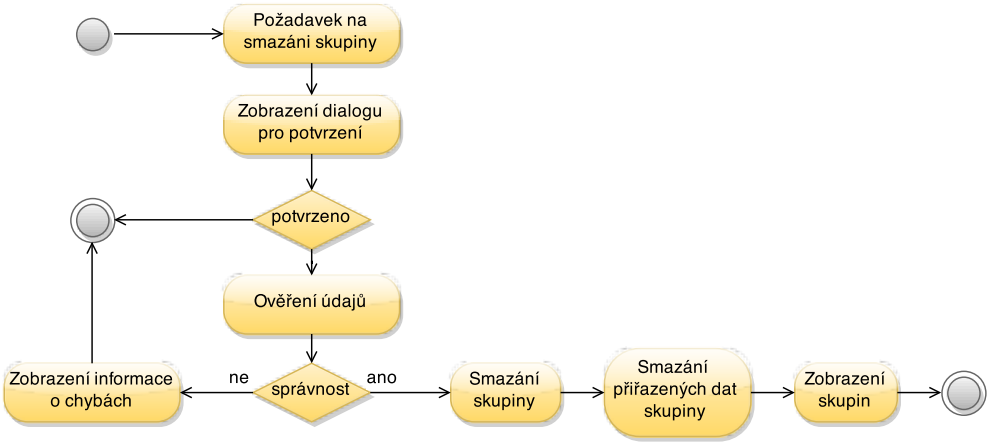
- **Popis:** Uživatel chce založit novou skupinu do systému.
- **Vstup:** Přihlášený uživatel a správně vyplněný formulář.
- **Hlavní scénář:**
  1. Uživatel vyvolá žádost při přidání skupiny.
  2. Systém zobrazí dialog s formulářem pro přidání nové skupiny.
  3. Uživatel vyplní formulář a odešle jej.
  4. Systém zkontroluje správnost a úplnost údajů.
  5. Systém uloží novou skupinu do systému.
  6. Systém zobrazí aktuální seznam skupin.
- **Vedlejší scénář:** .
  1. Nebyly zadány správné údaje.
  2. Systém informuje uživatele o chybách.
- **Výstup:** Přidání skupiny do systému a zobrazení skupin.



Obrázek 5.5: Případ užití - úprava skupiny

Úprava skupiny

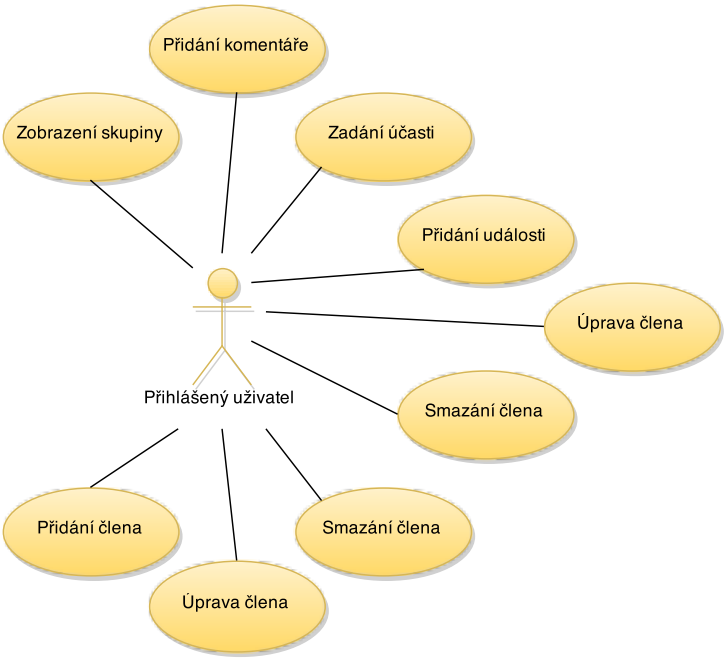
- **Popis:** Uživatel chce upravit již stávající vlastní skupinu v systému.
- **Vstup:** Přihlášený uživatel a správně vyplněný formulář.
- **Hlavní scénář:**
  1. Uživatel vyvolá žádost pro upravení skupiny.
  2. Systém zobrazí dialog s předvyplněným formulářem.
  3. Uživatel upraví formulář a odešle jej.
  4. Systém ověří vlastnictví skupiny.
  5. Systém zkontroluje správnost a úplnost údajů.
  6. Systém upraví skupinu v systému.
  7. Systém zobrazí upravenou skupinu.
- **Vedlejší scénář:**
  1. Nebyly zadány správné údaje.
  2. Systém informuje uživatele o chybách.
- **Výstup:** Upravení skupiny v systému a zobrazení skupiny.



Obrázek 5.6: Případ užití - smazání skupiny

Smazání skupiny

- **Popis:** Uživatel chce smazat vlastní skupinu ze systému.
- **Vstup:** Přihlášený uživatel a potvrzený dialog.
- **Hlavní scénář:**
  1. Uživatel vyvolá žádost pro smazání skupiny.
  2. Systém zobrazí potvrzovací dialog.
  3. Uživatel potvrdí dialog.
  4. Systém ověří vlastnictví skupiny.
  5. Systém odstraní skupinu s veškerými příslušnými daty ze systému.
  6. Systém zobrazí seznam skupin.
- **Vedlejší scénář:**
  1. Uživatel nepotvrdí odstranění skupiny.
  2. Systém zobrazí skupinu.
- **Výstup:** Odstranění skupiny a zobrazení seznamu skupin.



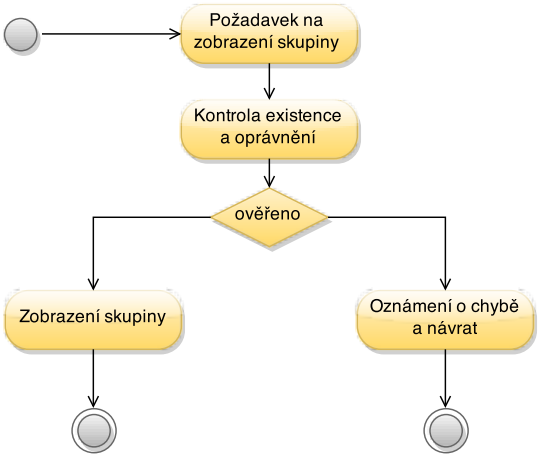
Obrázek 5.7: Diagram případů užití pro správu konkrétní skupiny

5.2.2 Případy užití správy skupiny

Na obrázku 5.7 jsou zobrazeny jednotlivé případy užití pro správu konkrétní skupiny. Hlavním užití je *zobrazení skupiny*. Dále jsou zde dvě trojice *přidat/upravit/smazat* jednou pro *členy* a jednou pro *události*. Užití *zadání účasti* pak spojuje tyto dvě entity dohromady. *Přidání komentáře* slouží k vedení si poznámek u jednotlivých částí.

Zobrazení skupiny

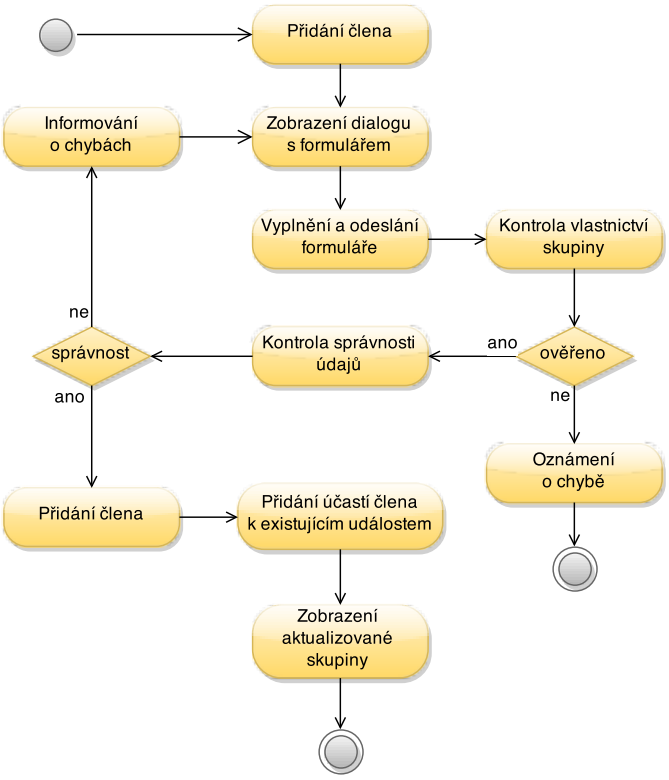
- **Popis:** Akce kdy si chce uživatel zobrazit konkrétní skupinu.
- **Vstup:** Oprávněný uživatel a ID vybrané skupiny.
- **Hlavní scénář:**
  1. Uživatel zadá požadavek na zobrazení skupiny.
  2. Systém zkontroluje existenci a oprávnění čtení skupiny.
  3. Systém zobrazí skupinu.
- **Vedlejší scénář:**
  1. Zadaná skupina neexistuje.
  2. Systém zobrazí oznámení o chybě.
- **Výstup:** Zobrazení konkrétní skupiny.



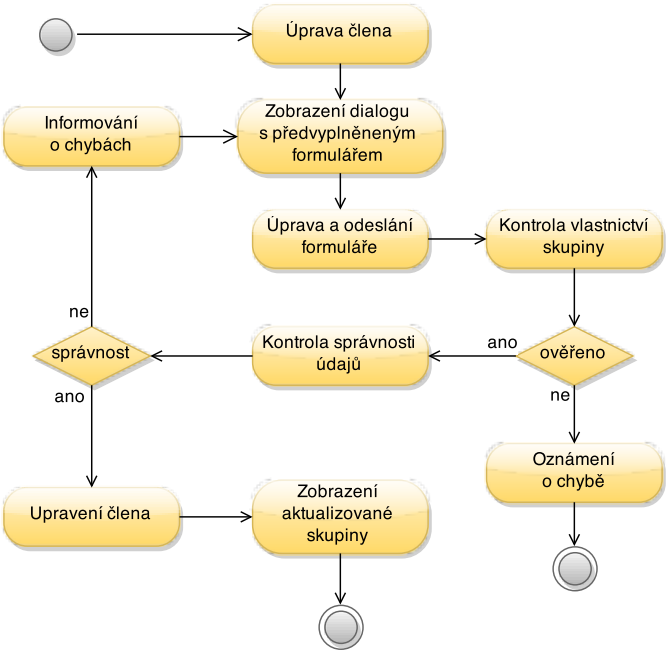
Obrázek 5.8: Případ užití - zobrazení skupiny

Přidání člena

- **Popis:** Uživatel chce přidat nového člena do skupiny.
- **Vstup:** Přihlášený uživatel. ID vybrané skupiny a správně vyplněný formulář
- **Hlavní scénář:**
  1. Uživatel vyvolá žádost o přidání člena do vybrané skupiny.
  2. Systém zobrazí dialog s formulářem pro vytvoření nového člena.
  3. Uživatel vyplní formulář a odešle jej.
  4. Systém ověří vlastnictví skupiny.
  5. Systém ověří správnost a úplnost údajů.
  6. Systém přidá člena do skupiny v systému.
  7. Systém vytvoří k existujícím událostem příslušná data o účasti člena.
  8. Systém zobrazí aktualizovanou skupinu.
- **Vedlejší scénář:**
  1. Nebyly zadány správné údaje.
  2. Systém informuje uživatele o chybách.
- **Vedlejší scénář č.2:**
  1. Byla zadána neexistující skupina či skupina bez oprávnění.
  2. Systém informuje uživatele o chybách.
- **Výstup:** Přidání člena do skupiny a její zobrazení.



Obrázek 5.9: Případ užití - přidání člena



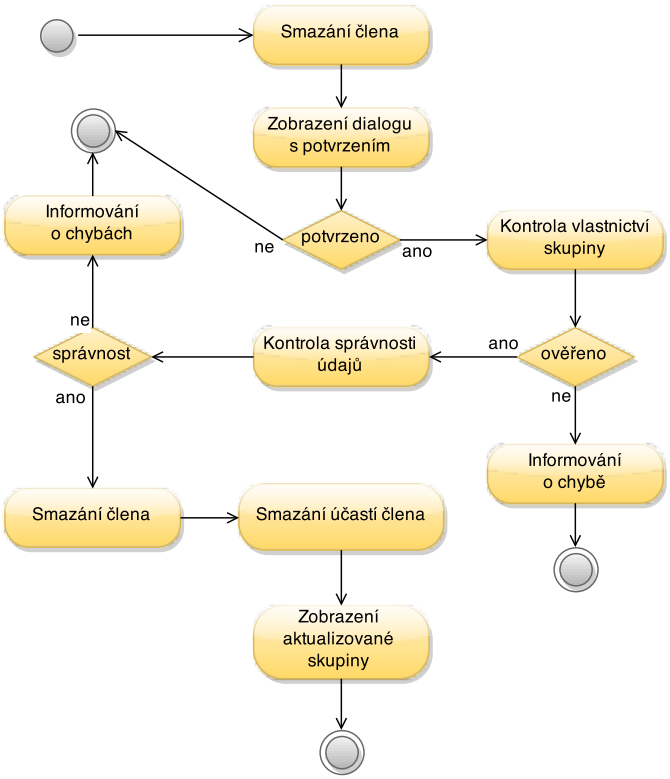
Obrázek 5.10: Případ užití - úprava člena

Úprava člena

- **Popis:** Uživatel chce upravit stávajícího člena skupiny.
- **Vstup:** Přihlášený uživatel. ID vybrané skupiny a správně vyplněný formulář.
- **Hlavní scénář:**
  1. Uživatel vyvolá žádost o úpravu člena vybrané skupiny.
  2. Systém zobrazí dialog s formulářem pro úpravu člena s předvyplněnými údaji.
  3. Uživatel upraví formulář a odešle jej.
  4. Systém ověří vlastnictví skupiny.
  5. Systém ověří správnost a úplnost údajů.
  6. Systém upraví člena skupiny v systému.
  7. Systém zobrazí aktualizovanou skupinu.
- **Vedlejší scénář:**
  1. Nebyly zadány správné údaje.
  2. Systém informuje uživatele o chybách.
- **Vedlejší scénář č.2:**
  1. Byla zadána neexistující skupina či člen anebo skupina bez oprávnění.
  2. Systém informuje uživatele o chybách.
- **Výstup:** Úprava člena skupiny a její zobrazení.

Smazání člena

- **Popis:** Uživatel chce smazat stávajícího člena skupiny.
- **Vstup:** Přihlášený uživatel. ID vybrané skupiny a potvrzený dialog.
- **Hlavní scénář:**
  1. Uživatel vyvolá žádost o smazání člena vybrané skupiny.
  2. Systém zobrazí potvrzovací dialog.
  3. Uživatel potvrdí dialog.
  4. Systém ověří vlastnictví skupiny.
  5. Systém ověří správnost a úplnost údajů.
  6. Systém odstraní člena ze skupiny.
  7. Systém odstraní související data o událostech a účastech.
  8. Systém zobrazí aktualizovanou skupinu.
- **Vedlejší scénář:**



Obrázek 5.11: Případ užití - smazání člena

1. Nebyly zadány správné údaje.
2. Systém informuje uživatele o chybách.

- **Vedlejší scénář č.2:**

1. Byla zadána neexistující skupina či člen anebo skupina bez oprávnění.
2. Systém informuje uživatele o chybách.

- **Výstup:** Smazání člena skupiny a její zobrazení.

Události

Správa událostí je prakticky totožná s předchozí částí, která se věnuje správě členů. Významově jsou rozdílné, ale při spravování totožné, až na odlišné identifikační atributy. Proto zde nejsou uváděny konkrétní diagramy. (*Přidání člena ≙ Přidání události, Úprava člena ≙ Úprava události, Smazání člena ≙ Smazání události*).

Přidání události

- **Popis:** Uživatel chce přidat výskyt události do skupiny.
- **Vstup:** Přihlášený uživatel. ID vybrané skupiny a správně vyplněný formulář.
- **Hlavní scénář:**
  1. Uživatel vyvolá žádost o přidání události do vybrané skupiny.
  2. Systém zobrazí dialog s formulářem pro vytvoření nové události.
  3. Uživatel vyplní formulář a odešle jej.
  4. Systém ověří vlastnictví skupiny.
  5. Systém ověří správnost a úplnost údajů.
  6. Systém přidá událost do skupiny v systému.
  7. Systém vytvoří k existujícím členům příslušná data o účasti na události.
  8. Systém zobrazí aktualizovanou skupinu.
- **Vedlejší scénář:**
  1. Nebyly zadány správné údaje.
  2. Systém informuje uživatele o chybách.
- **Vedlejší scénář č.2:**
  1. Byla zadána neexistující skupina či skupina bez oprávnění.
  2. Systém informuje uživatele o chybách.
- **Výstup:** Přidání události do skupiny a její zobrazení.

Úprava události

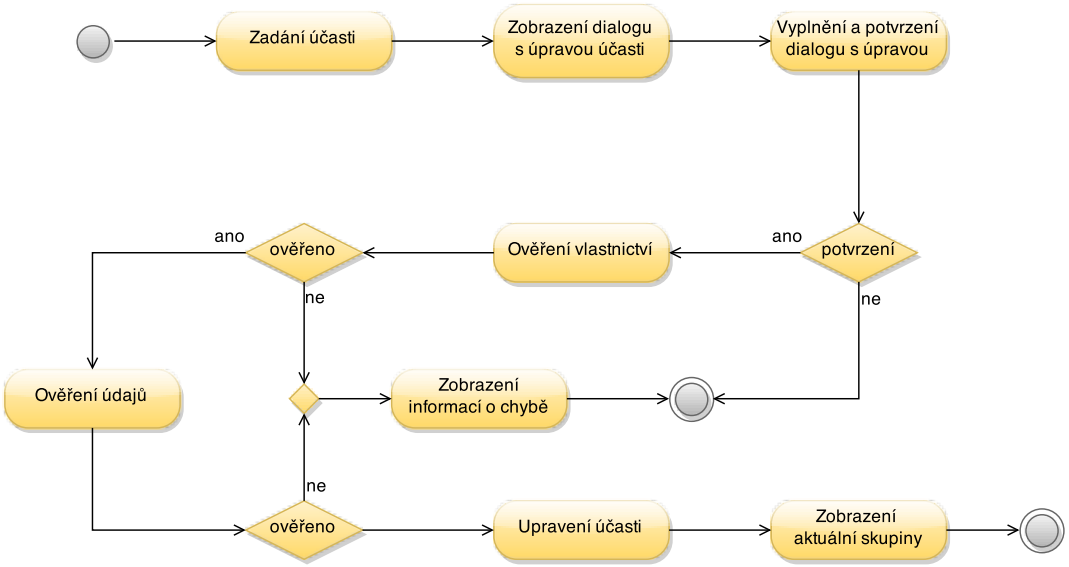
- **Popis:** Uživatel chce upravit stávající události skupiny.
- **Vstup:** Přihlášený uživatel. ID vybrané skupiny a správně vyplněný formulář.
- **Hlavní scénář:**
  1. Uživatel vyvolá žádost o úpravu události vybrané skupiny.
  2. Systém zobrazí dialog s formulářem pro úpravu události s předvyplněnými údaji.
  3. Uživatel upraví formulář a odešle jej.

4. Systém ověří vlastnictví skupiny.
5. Systém ověří správnost a úplnost údajů.
6. Systém upraví události skupiny v systému.
7. Systém zobrazí aktualizovanou skupinu.

- **Vedlejší scénář:**
  1. Nebyly zadány správné údaje.
  2. Systém informuje uživatele o chybách.
- **Vedlejší scénář č.2:**
  1. Byla zadána neexistující skupina či události anebo skupina bez oprávnění.
  2. Systém informuje uživatele o chybách.
- **Výstup:** Úprava události skupiny a její zobrazení.

Smazání události

- **Popis:** Uživatel chce smazat existující událost skupiny.
- **Vstup:** Přihlášený uživatel. ID vybrané skupiny a potvrzený dialog.
- **Hlavní scénář:**
  1. Uživatel vyvolá žádost o smazání události vybrané skupiny.
  2. Systém zobrazí potvrzovací dialog.
  3. Uživatel potvrdí dialog.
  4. Systém ověří vlastnictví skupiny.
  5. Systém ověří správnost a úplnost údajů.
  6. Systém odstraní událost ze skupiny.
  7. Systém odstraní související data o členech a účastech.
  8. Systém zobrazí aktualizovanou skupinu.
- **Vedlejší scénář:**
  1. Nebyly zadány správné údaje.
  2. Systém informuje uživatele o chybách.
- **Vedlejší scénář č.2:**
  1. Byla zadána neexistující skupina či událost anebo skupina bez oprávnění.
  2. Systém informuje uživatele o chybách.
- **Výstup:** Smazání události skupiny a její zobrazení.



Obrázek 5.12: Případ užití - zadání účasti

Zadání účasti

- **Popis:** Zadání účasti slouží k propojení konkrétní člena s konkrétní událostí a určuje, zda-li člena byl či nebyl přítomen. Účast je vždy vytvořena při vytvoření nového člena či události a musí je vždy propojovat. Uživatel má tak možnost pouze upravovat její hodnotu.
- **Vstup:** Přihlášený uživatel. Vybraná skupina a účast.
- **Hlavní scénář:**
  1. Uživatel vyvolá žádost u změnu účasti.
  2. Systém zobrazí dialog s úpravou účasti. Stávající účast je předvyplněna.
  3. Uživatel upraví hodnotu účasti a potvrdí.
  4. Systém ověří vlastnictví skupiny.
  5. Systém ověří správnost a úplnost údajů.
  6. Systém upraví hodnotu účasti.
  7. Systém zobrazí aktualizovanou skupinu
- **Vedlejší scénář:**
  1. Nebyly zadány správné údaje.
  2. Systém informuje uživatele o chybách.
- **Vedlejší scénář č.2:**
  1. Byla zadána neexistující skupina, člen či událost.

2. Systém informuje uživatele o chybách.

- **Výstup:** Změna účasti a zobrazení skupiny.

Zadání komentáře

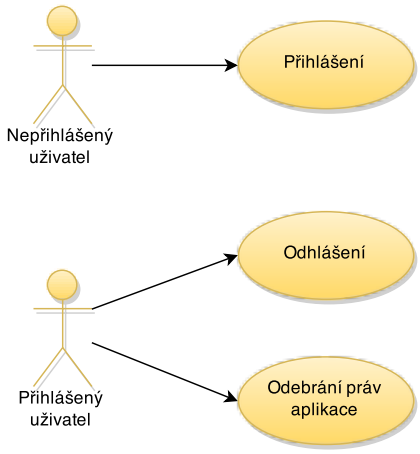
- **Popis:** Uživatel chce přidat *komentář* k nějaké části. Uživatel musí mít možnost komentovat jednotlivé části. Například: přidat komentář k *účasti* - poznámka proč daný *člen* nebyl.
- **Vstup:** Přihlášený uživatel. Vybraný člen/událost/účast.
- **Hlavní scénář:**
  1. Uživatel vyvolá žádost o přidání komentáře.
  2. Systém zobrazí dialog pro zadání komentáře.
  3. Uživatel vyplní formulář a potvrdí jej.
  4. Systém ověří práva a správnost údajů.
  5. Systém uloží komentář.
  6. Systém zobrazí skupinu.
- **Vedlejší scénář:**
  1. Byla zadána neexistující skupina či žádná data komentáře.
  2. Systém informuje uživatele o chybách.
- **Výstup:** Zadání komentáře a zobrazení skupiny.

5.2.3 Případy užití změny přihlášení

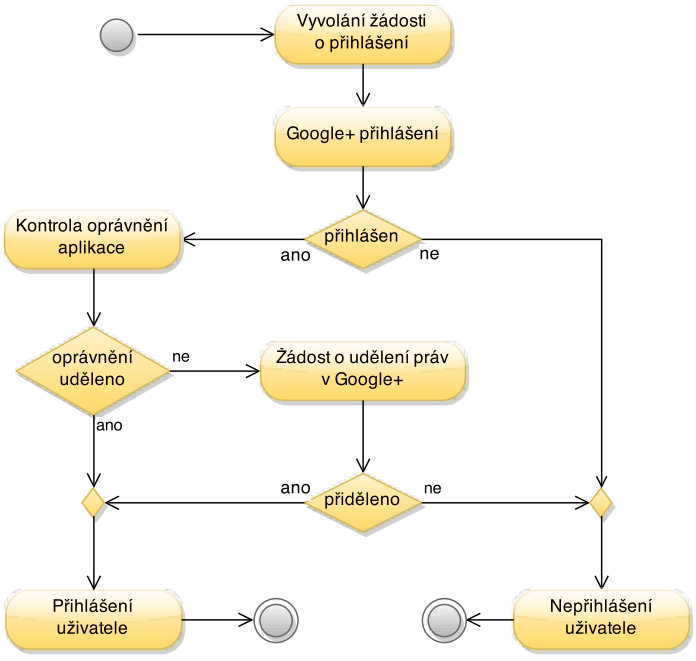
Obrázek 5.13 ilustruje užití týkající se změn přihlášení uživatele.

Přihlášení

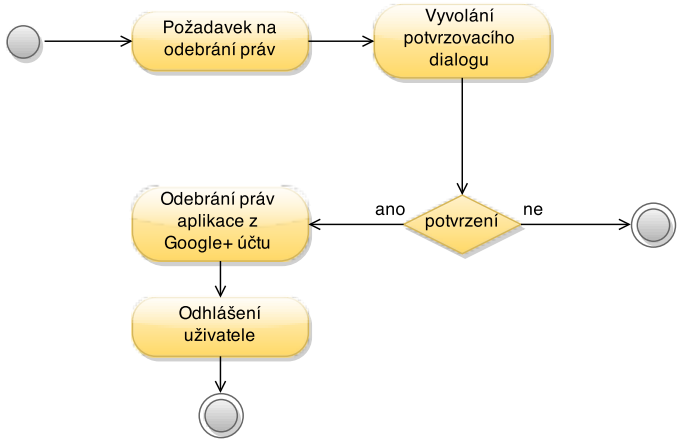
- **Popis:** Uživatel vstupuje do systému a chce provádět autorizované akce. Musí se přihlásit.
- **Vstup:** Nepřihlášený uživatel.
- **Hlavní scénář:**
  1. Uživatel zadá požadavek o přihlášení.
  2. Systém přesměruje uživatele na přihlášení skrz *Google+* účet.
  3. Uživatel je přihlášený pod *Google+* účtem.
  4. Systém zkontroluje, zda-li uživatel povolil oprávnění aplikaci u tohoto *Google+* účtu.



Obrázek 5.13: Diagram případů užití změny přihlášení



Obrázek 5.14: Případ užití - Přihlášení



Obrázek 5.15: Případ užití - odebrání práv aplikace

5. Uživatel přidělí nebo již přidělil aplikaci oprávnění.
6. Systém získá přihlášení v rámci aplikace.
7. Systém zobrazí uživateli vlastní skupiny.

- **Vedlejší scénář:**
  1. Uživatel se nepřihlásí pod *Google+* účtem.
  2. Systém zobrazí uživateli informace o chybě.
- **Vedlejší scénář č.2:**
  1. Uživatel nepřidělí aplikaci oprávnění v *Google+* účtu.
  2. Systém zobrazí uživateli informace o chybě.
- **Výstup:** Přihlášený uživatel a zobrazení jeho skupin.
- **Poznámka:** Na obrázku 5.14 je zobrazen diagram průchodu tímto případem užití. Je zde zobrazena ukončovací aktivita *nepřihlášení uživatele*. Uživatel zde skončí, pokud sice nejprve sám vyvolá přihlášení, poté ale v části přihlášení do *Google* zruší přihlášení (nepřihlásí se) či v části přidělení práv sám odmítne přidělení práv aplikaci. Tímto se aktivita přihlášení ukončí, místo běžného nabídnutí nového přihlášení (pokud jsou například zadaný špatný přihlašovací údaje) jelikož sám uživatel pokračovat v přihlášení sám odmítl.

Odebrání práv aplikace

- **Popis:** Uživatel, který autorizoval aplikaci pod svým *Google+* účtem chce její práva odebrat. Odebrání práv nesmaže jeho data.
- **Vstup:** Přihlášený uživatel.
- **Hlavní scénář:**

1. Uživatel vyvolá požadavek na odstranění práv aplikace.
2. Systém zobrazí potvrzovací dialog.
3. Uživatel potvrdí dialog.
4. Systém odstraní práva aplikace skrz *Google+ API*.
5. Systém odhlásí uživatele.

- **Výstup:** Nepřihlášený uživatel.

Odhlášení

- **Popis:** Uživatel se chce odhlásit z aplikace.
- **Vstup:** Přihlášený uživatel.
- **Hlavní scénář:**
  1. Uživatel vyvolá požadavek pro odhlášení z aplikace.
  2. Systém odhlásí uživatele skrz *Google+ API*.
  3. Systém odhlásí uživatele z aplikace.
- **Výstup:** Nepřihlášený uživatel.

5.3 Analýza návrhu uživatelského rozhraní

Je třeba navrhnout dvě různá uživatelská rozhraní, které mají stejné společné prvky a celkový dojem, ale zároveň zapadají do své platformy.

5.3.1 Mobilní klient

Na platformě Android běží aplikace celoobrazovkově, ale zároveň se počítá s menšími displeji. Aplikace může běžet buď na výšku či na šířku, může být kdykoliv ukončena. Zařízení s *OS* Android v drtivé většině obsahují dotykový displej - počítá se s dotyky uživatelových prstů - čemuž musí být i prvky uživatelského rozhraní uzpůsobeny. Pro intuitivnost *UI* je vhodné dodržovat designové vodítka Material designu[1].

Dílčí funkční části aplikace budou rozděleny do jednotlivých "obrazovek"- implementováním *Fragmentů*. Části pro úpravu objektů řešeny vyskakovacími dialogy obsahující formuláře. Hlavní část *zobrazení skupiny* bude řešeno vlastní implementací tabulky, neboť *SDK* Androidu neobsahuje dostatečně vhodnou komponentu.

Navrhnuté části UI:

- Fragment - Přihlášení
- Fragment - Seznam skupin

[1] Dokumentace Material deisgnu

- Fragment - Zobrazení skupiny
- Dialog - Úprava skupiny
- Dialog - Úprava člena
- Dialog - Úprava události
- Dialog - Zadání účasti
- Dialog - Úprava komentáře
- vlastní tabulková komponenta - Zobrazení účastí

5.3.2 Webové rozhraní

Webová část už z principu využití databáze *Firebase*, jejíž knihovny jsou javaskriptové, bude záviset právě na využití javaskriptu. Aplikace tak je navržena jako tzv "jednostránková", kdy si uživatel načte úvodní *HTML* soubor s *DOM*em aplikace a hlavní skript, který bude obsahovat naší aplikaci. Veškeré akce uživatele poté budou ovládány skrz javaskript.

HTML soubor a celé rozvržení aplikace bude rozděleno do několika kontejnerů (v *HTML* tag *DIV*) vhodně rozmístěných po stránce.

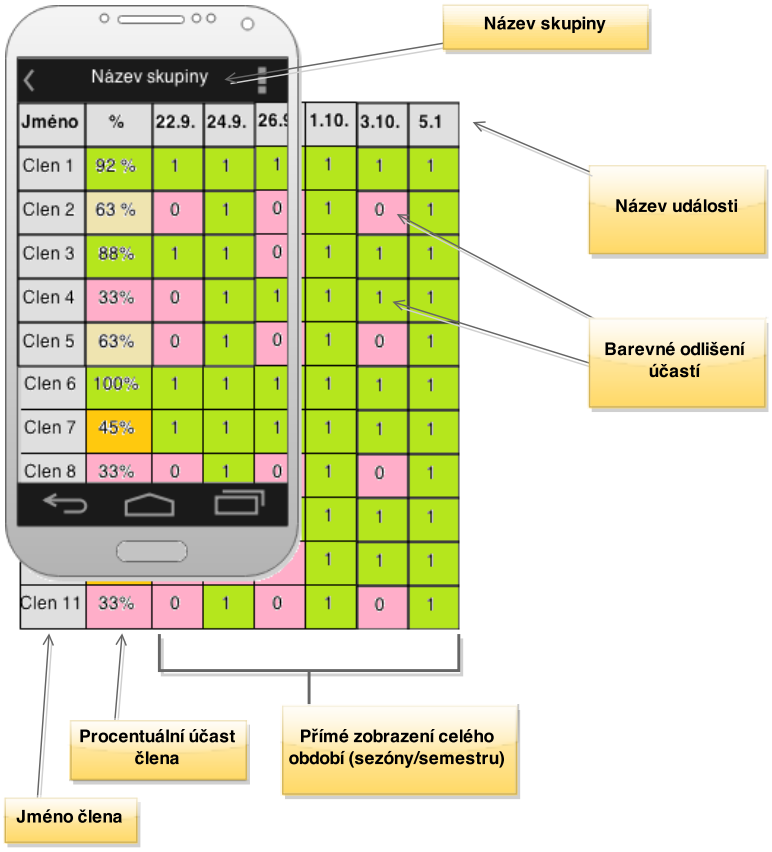
Navržené kontejnery *UI*:

- DIV s aktuálně přihlášeným uživatelem - s možností odhlášení
- DIV s aktuálně vybranou skupinou
- DIV se seznamem skupin daného uživatele
- vhodně umístěné DIVy s formuláři pro úpravu dané skupiny - úprava členů, událostí, komentářů a skupiny
- DIV s formulářem pro zadání účasti
- DIV s informacemi o vybrané skupině

5.3.3 Případ užití - Zobrazení skupiny

Na obrázku 5.16 je vyobrazen návrh části zobrazení skupiny. Toto zobrazení je kritickou částí aplikace, je třeba aby uživatel s tabulkou mohl efektivně manipulovat a dobře se orientovat. Při analýze návrhu pro mobilního klienta došlo ke zjištěním:

- Pro přehlednost účastí nesmí buňky obsahovat zbytečný text.
- Buňky účastí by měly být barevně odlišeny podle hodnoty obsažené.
- Tabulka musí mít fixní sloupec se jmény a řádek s událostmi pro snadnou orientaci v datech.
- Průměrná účast člena by měla být také barevně rozlišena.

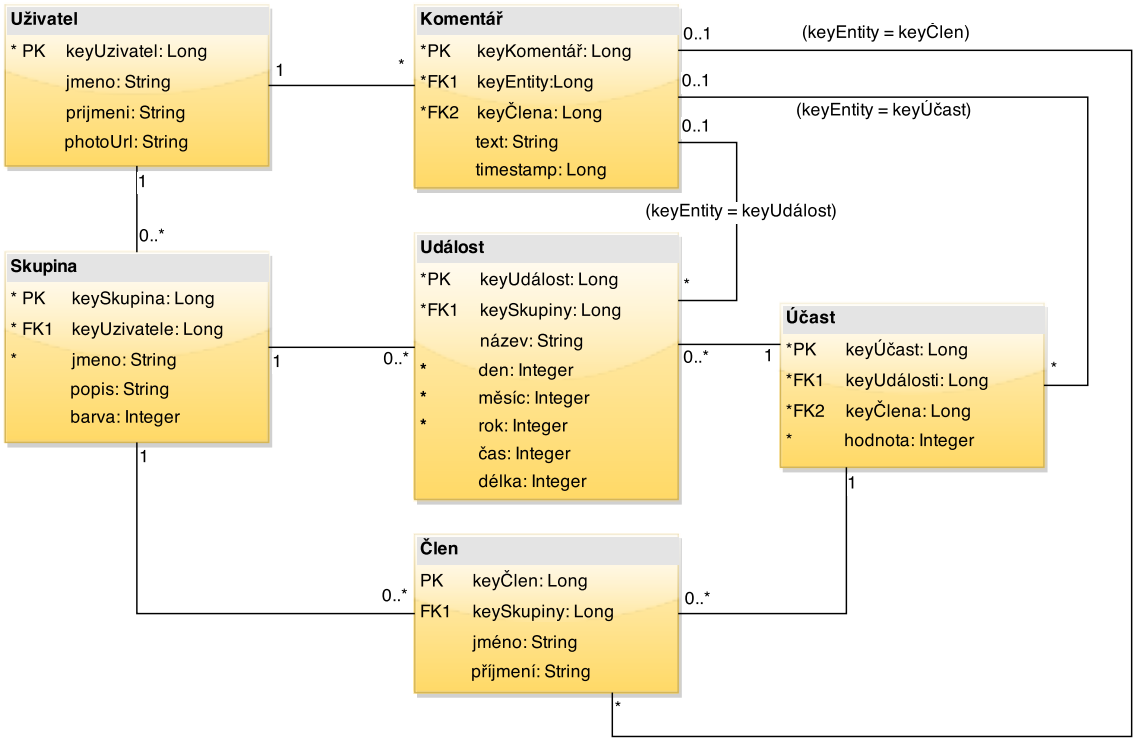


Obrázek 5.16: Návrh UI - zobrazení skupiny

Kapitola 6

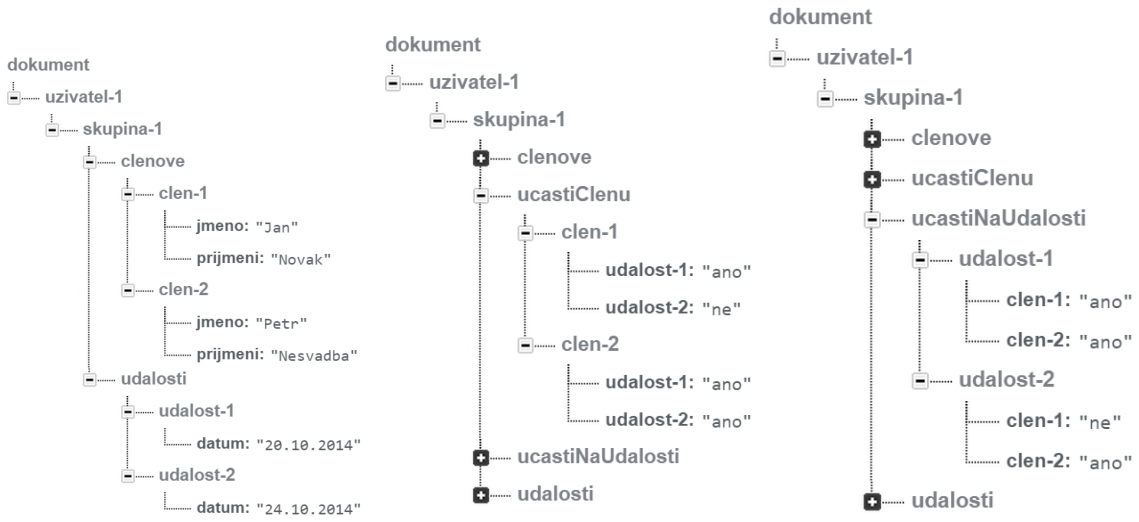
Návrh a implementace

6.1 Fyzický datový model



Obrázek 6.1: Fyzický datový model databáze

Obrázek 6.1 popisuje fyzický datový model naší databáze. Bez větších změn oproti datovému konceptuálnímu diagramu z analýzy 5.1 definuje datové typy atributů, přiřazuje primární klíče k entitám a pro vztahy mezi entitami definuje cizí klíče.



Obrázek 6.2: Struktura dat ve Firebase

6.2 Převod do databáze Firebase

Firebase není tradiční databází, svoje data si uchovává v rádoby *JSON* struktuře pro snadný přístup (popsáno v části Databázový backend 2.2). Základní datové typy měnit potřeba není, existují zde standardní typy obsažené v Javaskriptu. Řetězce *String* zůstávají, číselné hodnoty *Integer* a *Long* budou zastoupeny typem *Number*.

Důležitější částí byl převod vztahů. Vztah definovaný jedním cizím klíčem lze realizovat jednoduše tím, že daný objekt je v hierarchické struktuře potomkem objektu ke kterému patří.

Kořeny dokumentu databáze jsou tak jednotliví *uživatelé*, respektive jejich univerzální klíče získané z ID *Google+* účtu. V objektu *uživatele* jsou pak *skupiny* (jejich klíče). *Skupina* obsahuje potomka *události* (se všemi *událostmi*) a potomka *členové* (se všemi *členy*). Tímto stylem jsou realizovány vztahy *uživatel-skupina*, *skupina-událost*, *skupina-člen*.

Základní struktura dat entit *uživatel*, *skupina*, *událost*, *člen* je nastíněna na obrázku 6.2 vlevo.

Entita, která je závislá na dvou vztazích ale vzhledem k hierarchickému rozložení dat nemůže patřit do dvou různých částí zároveň. Dokumentace *Firebase*[1] vysvětluje řešení tohoto problému. Popisuje použití existence klíče ve více objektech.

Entita *účast* ve fyzickém datové modelu obsahuje právě závislosti na *události* a *členovi*. Řešením je vytvoření dvou nových uzlů:

- První uzel s názvem *účastČlenů* obsahuje klíče všech členů a každý člen pak obsahuje klíče všech událostí. V každým klíči je pak hodnota účasti.
- Druhý uzel s názvem *účastNaUdálosti* obsahuje klíče všech událostí a každá událost pak obsahuje klíče všech členů. V každým klíči je pak hodnota účasti.

[1] Článek o strukturování dat ve Firebase

Vzniká zde mírná duplicita dat, nicméně tento způsob je oficiálními zdroji doporučovaný a jiná elegantní jednoduchá varianta řešení v podstatě není.

Pokud pak chci zjistit informaci o účastech člena přistupuji k uzlu *skupina/účastČlenů/[klíčČlena]/*, který obsahuje jeho všechny účasti. Pro zjištění informace účasti na konkrétní události přistupuji do uzlu *skupina/účastNaUdálosti/[klíčUdálosti]*, který obsahuje všechny členy a jejich účasti. Danou strukturu dat popisuje obrázek 6.2 uprostřed a vpravo.

6.3 Mobilní klient

Pro vývoj mobilního klienta bylo zvoleno oficiální vývojové prostředí pro platformu Android - *Android Studio* [2]. Prostředí v nedávné době dosáhlo první stabilní verze 1.0 a obsahuje rozsáhlé množství funkcí pro vývoj software. Pokročilý editor kódu s refaktorizací, debugovací nástroje, emulátor Android zařízení, správu *SDK* a další.

Prvek uživatelského rozhraní - View

V další části se vyskytuje často slovo *prvek*. Je ním míněna část uživatelského rozhraní, která se nazývá anglicky *view*. Základní *View* je prázdná zobrazitelná komponenta. Každá část, která má být zobrazena uživateli od této třídy dědí, ať už se jedná o jednoduchou čáru (*Drawable*), základní textové pole (*TextView*) či komplexní komponentu pro výběr data (*DatePicker*).

6.3.1 Použité knihovny

V *Android studiu* se k sestavení aplikace využívá *Gradle* [3]. Pro použití externích knihoven stačí napsat do bloku závislostí v sestavovacím skriptu jejich jednoznačný název s verzí. Ukázka závislostí v aplikaci:

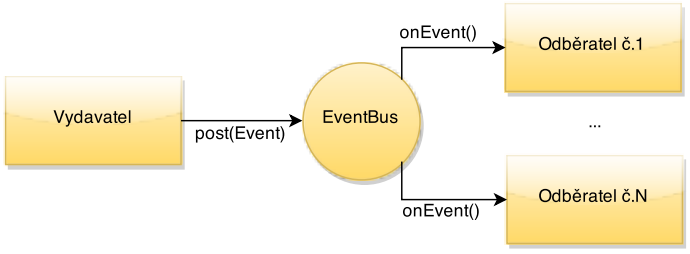
```
dependencies {
    compile 'com.android.support:appcompat-v7:21.0.0'
    compile 'com.android.support:cardview-v7:+'
    compile 'com.firebase:firebase-client-android:2.0.1'
    compile 'com.google.android.gms:play-services:6.1.71'
    compile 'com.jakewharton:butterknife:6.0.0'
    compile 'com.wrapp.floatlabelededittext:library:0.0.5'
    compile 'de.greenrobot:eventbus:2.4.0'
}
```

K identifikaci konkrétní knihovny se využívá obrácené *DNS* notace[4], která jednoznačně určuje majitele knihovny. Následuje název knihovny a číslo verze. Vše oddělené dvojtečkou.

Android Studio v základu využívá centrální repozitář *Maven*[5]. Nabízí i možnost nadefinovat si vlastní repozitáře.

[2] domovská stránka Android Studio
[3] Gradle - The Enterprise Automation Tool
[4] Odkaz na článek Reverse-DNS na wikipedia.org
[5] Centrální repozitář Maven Central Repository

Komunikace v rámci aplikace - EventBus



Obrázek 6.3: EventBus

Android poskytuje *API* ke komunikaci mezi částmi aplikace i mezi různými aplikacemi. Jedná se o *API* pro zasílání událostí[6] a dosti složité *API* pro naslouchání událostí[7]. V rámci aplikace se dá ale použít knihovna *EventBus*[8], která nabízí jednoduché a efektivní *API* pro vytvoření komunikační sběrnice mezi částmi. Komunikace je potřeba mezi aktivitami, fragmenty, vlákny a službami.

Princip *Vydavatel/Odběratel* je ilustrován na obrázku 6.3. Vydavatel předá *EventBusu* objekt určité třídy skrz statickou metodu *post(Event event);*. *EventBus* pak zaregistrované odběratele, kteří poslouchají na danou třídu objektu události, informuje o události s daným objektem. Objekty mohou naslouchat z různých kontextů a i vláken. Pomocí anotací se dá určit v jakém vlákně se má daná metoda *onEvent(EventType type)* vykonat.

Postup:

- Definování třídy události - jakákoliv třída s proměnnými a i metodami.
- Zaregistrování posluchače:
  - Instance třídy musí zavolat *EventBus.getInstance().register(this)*;
  - Třída musí definovat metodu *onEvent(EventType type) { ... }* s daným typem parametru.
- Vydavatel předá EventBusu událost - EventBus.getInstance().post(new Event(...));

Odlišení v jakém vlákně se má událost vyvolat se řeší pomocí specifických názvů metod posluchačů:

- *onEvent(EventType event);* - vyvolá se ve stejném vlákně, v jakém je událost poslána.
- *onEventMainThread(EventType event);* - vyvolá se v hlavním běžícím vlákně.
- *onEventBackgroundThread(EventType event);* - vyvolá se ve vlákně běžícím na pozadí.
- *onEventAsync(EventType event);* - vyvolá se ve vedlejším vlákně.

[6] Odkaz průvodce Intents
[7] Odkaz dokumentace BroadcastReciever
[8] Odkaz knihovna EventBus na portálu github.com

Použití *EventBus*u oproti základním nástrojům Androidu vede ke zjednodušení a zefektivnění kódu. Hlavní použití v mé aplikaci je v části přistupující do databáze a části obstarávající přihlášení uživatele. Tato operace může být časově náročná a tak by její běh v hlavním vlákně způsobil neodpovídání aplikace - vyskočení *ANR*[9]. Pomocí *EventBus*u se tak ve vedlejším vlákně provede časově náročná operace a při dokončení práce se vyvolá událost na hlavním vlákně, která pak uživatele informuje o případném úspěchu. Aplikace tak po celou dobu zůstává použitelná.

Tvorba UI - ButterKnife

Vývoj pro Android trpí nutností opakování častého kódu. Jednou takovou nutností je po nastavení rozvržení *UI* z *XML* souboru potřeba ručně vyhledat jednotlivé prvky k použití, správně je přetypovat a ideálně samozřejmě ještě provést existenci tohoto prvku. To vše se dá učinit až v metodě vytvoření životního cyklu dané komponenty.

Pro každý prvek s kterým chceme v dané komponentě pracovat musíme:

- Nadefinovat proměnnou.
- Vyhledat prvek v dané hierarchii - metody *findViewByID* či *findViewByTag*.
- Ověřit existenci prvku v hierarchii.
- Přetypovat prvek.
- Až zde ho můžeme použít.

Typická komponenta může být například formulář, s prvky pole pro zadání jména, emailu, hesla a hesla pro kontrolu. Dále tlačítky pro potvrzení a zrušení[10]. Vcelku jednoduchá komponenta s šesti prvky, kde by ale 30 řádků (6 prvků * 5 řádků) zabrala inicializace všech prvků, navíc by tyto řádky byly rozesety různě po třídě.

Knihovna *ButterKnife*[11] slouží k "vstřikování"prvků do komponenty pomocí zpracování anotací k vygenerování tohoto často používaného kódu. Stačí definovat proměnné, označit je příslušnou anotací a v hlavní části vytvoření komponenty pak pouze jednou zavolat pomocnou metodu knihovny. Kód se generuje staticky při kompilaci aplikace a tak nezvyšuje náročnost aplikace. Počet řádků nutných v tomto případě se sníží na 7 (6 prvků + 1 volání pomocné metody) a značně zpřehlední celou třídu.

ButterKnife nabízí anotace pro vložení prvku, prvků, pro jednoduché zaregistrování posluchačů událostí kliknutí (dvojklikinutí, ...).

Následující ukázka zobrazuje použití bez *ButterKnife*.

[9] ANR - Android Not Responding - stav kdy aplikace neodpovídá, o kterém informuje vyskakovací dialogové okno.

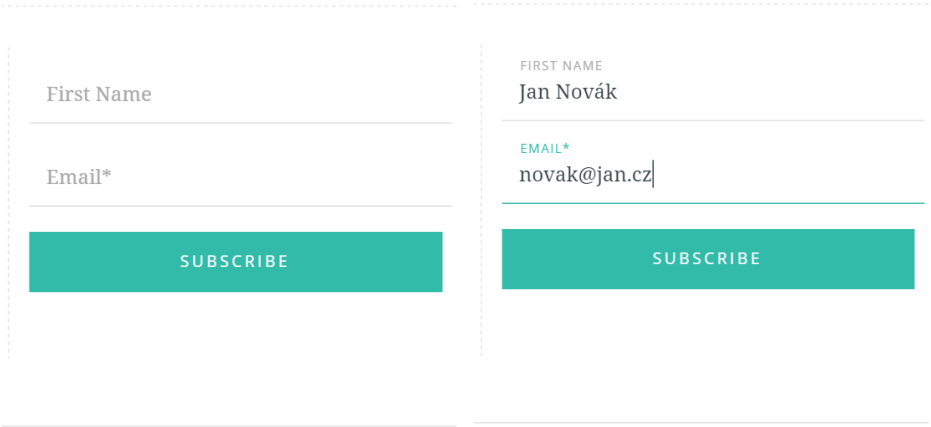
[10] Při správném návrhu by zde byly i prvky pro dynamickou nápovědu o chybách, odkaz pro zapomenutí hesla a další.

[11] knihovna ButterKnife na portálu github.com

```
class ExampleActivity extends Activity {
   private TextView tv_username;
   private TextView tv_mail;
   private Button bt_submit;

   @Override
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);

      setContentView(R.layout.simple_activity);

      tv_username = (TextView) findViewById(R.id.tv_username);
      tv_mail = (TextView) findViewById(R.id.tv_mail);
      bt_submit = (Button) findViewById(R.id.bt_submit);

      if(tv_username != null) {
         // pracovat s tv_username
      }

      if(tv_mail != null) {
         // pracovat s tv_mail
      }

      if(bt_submit != null) {
         bt_submit.setOnClickListener(new OnClickListener() {
            Override
            public void onClick(View view) {
               // pracovat s bt_submit
            }

             }
      }
   }
}
```

Je zde vidět "ukecanost" kódu - potřeba jednotlivé prvky ručně hledat a kontrolovat existenci. Dále je zde zobrazeno přiřazení funkci tlačítku. V další ukázce je již použita tato knihovna, kód je kratší a přehlednější. V úvodu třídy je definování prvků a pomocí anotace informování jak je najít. V metodě *onCreate* pak stačí zavolat pomocnou metodu *ButterKnife inject()* a knihovna se o vše postará sama.

```
class ExampleActivity extends Activity {
   @InjectView(R.id.tv_username) TextView tv_username;
   @InjectView(R.id.tv_mail) TextView tv_mail;

   @Override
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);

      setContentView(R.layout.simple_activity);

      ButterKnife.inject(this);

      // pracovat s tv_username, tv_mail, ...
   }

   @OnClick(R.id.bt_submit)
   void submit() {
      // pracovat s bt_submit
   }

}
```



Obrázek 6.4: Ukázka FloatLabelText

Správa databáze - Firebase

Použití Firebase je již naznačeno v kapitole o technologiích 2.2. V aplikaci je třída *DAO*[12] využívající vzor Singleton/Jedináček. Tato obstarává veškerou komunikaci s databází.

Třída *DAO* a veškerá práce s *Firebase* se skládá z částí:

- *Firebase* odkaz na data přihlášeného uživatele.
- Autorizace uživatele na serveru *Firebase*.
- Posluchač na změnu dat na serveru - změněná data distribuuje do aplikace skrz *EventBus*.
- Metody pro správu skupin, členů, událostí, účastí v databázi.

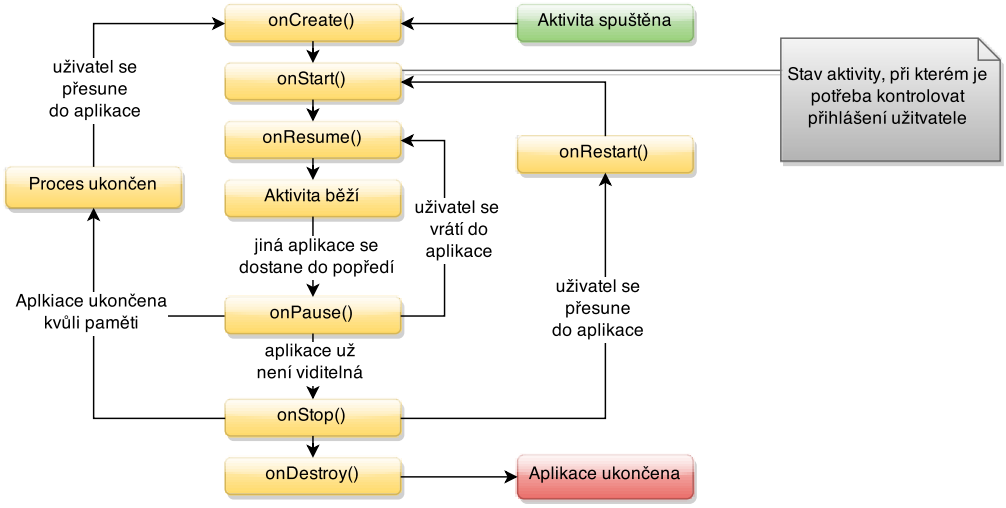
Tvorba UI - FloatLabeledEditText

FloatLabeledEditText[13] slouží k úspoře místa při tvorbě formulářů. Knihovna inspirována návrhem v článku Matt D.Smitha[3] zobrazuje nejprve popisek textového pole jako nápovědu uvnitř pole. V okamžiku kdy uživatel vyvolá interakci s polem (začne do něj psát) proběhne krátká animace, která přechodem zobrazí daný popisek pole nad ním. Obrázek 6.4 zobrazuje úvodní stav (vlevo) a stav kdy uživatel s poli již pracuje (vpravo).

Kromě úspory místa, tak dochází i k přehlednosti rozhraní. Následující úryvek zobrazuje použití knihovny. Rozhraní definované v *XML* souboru obsahuje textové pole *EditText*, které je uvnitř prvku *FloatLabeledEditText*:

```
1 <com.wrapp.floatlabeledittext.FloatLabeledEditText
2     android:layout_width="match_parent"
3     android:layout_height="wrap_content">
4
5     <EditText
```

[12] DAO - data access object - objekt pro přístup k datům.
[13] knihovna FloatLabeledEditText na portálu github.com



Obrázek 6.5: Životní cyklus aktivity

```
6         android:id="@+id/et_nazev"
7         android:hint="Nazev skupiny"
8         android:layout_width="match_parent"
9         android:layout_height="wrap_content"
10        >
11        <requestFocus />
12    </EditText>
13</com.wrapp.floatlabelededittext.FloatLabeledEditText>
```

Přihlášení - GMS Play Services

Knihovna *Google Mobile Services - Play Services* slouží pro přístup ke službám Google. V aplikaci je využíváno služeb přihlášení ke Google+ účtu a k autorizaci. Pomocí knihovny lze přihlásit uživatele a získat jeho autorizační token, který lze pak dále využívat a identifikovat uživatele jeho jménem. Tento token je třeba si udržovat aktivní. Třída *GoogleApiClient* integruje služby Google. Obsahuje metody pro správu stavu klienta:

- *connect()* - připojení klienta.
- *disconnect()* - odpojení klienta.

Dále je potřeba nastavit posluchače na změnu stavu připojení:

- *onConnected()* - klient je přihlášen.
- *onConnectionSuspended()* - přihlášení odloženo. Důvodem může být ztráta připojení či ukončení aplikace.
- *onConnectionFailed()* - přihlášení se nezdařilo. Možné důvody: Neplatný uživatel, nedostupnost *API*, překročen časový limit spojení a další.

Tyto jednotlivé stavy je potřeba vhodně propojit se stavy životní cyklu aplikace. Obrázek 6.5 ilustruje tyto stavy. Pokud se aplikace nachází ve stavu *onStart()*, pak se u instance třídy *GoogleApiClient* zavolá *onConnect*. Poté nastane buď událost *onConnected()* kdy je uživatel přihlášen (případ užití Přihlášení 5.2.3). Nebo se přihlášení nezdaří a je vyvolána buď událost *onConnectionSuspended()* či *onConnectionFailed()* dle typu chyby. Ve stavu *onStop()* je vhodné *GoogleApiKlienta* odpojit metodou *disconnect()*.

Po připojení ke službě (*onConnected*) je třeba získat i autorizační token uživatele, pro uložení v mobilním zařízení a pro autorizování uživatele na serverech *Firebase*. Při získání tohoto tokenu probíhá opět komunikace po síti a tak je třeba akci provádět ve vedlejším neblokujícím vlákně:

```
1  private void getAuthToken() {
2     // Ziskani tokenu musi probehnout na pozadi
3     AsyncTask<Void, Void, String> task = new AsyncTask<Void, Void, String>() {
4        @Override
5        protected String doInBackground(Void... params) {
6           String token = null;
7           try {
8              // Nastaveni pozadovanych opravneni: chceme Google+ prihlaseni
9              String scope = String.format("oauth2:%s", Scopes.PLUS_LOGIN);
10
11             // Pomocna metoda pro ziskani tokenu
12             token = GoogleAuthUtil.getToken(context,
13                AccountApi.getAccountName(client), scope);
14          } catch (IOException transientEx) {
15             // Sitova chyba
16          } catch (UserRecoverableAuthException e) {
17             // Nedostatecna opravneni, vyvolame zadost o~chybejici opravneni
18             Intent recover = e.getIntent();
19             startActivityForResult(recover, RC_SIGN_IN);
20          } catch (GoogleAuthException authEx) {
21             // Nepodarilo se prihlasit
22          }
23          return token;
24       }
25
26       @Override
27       protected void onPostExecute(String token) {
28          if (token != null) {
29                   // Jsme prihlaseni a mame token
30                   // Prace s~tokenem
31                   // ...
32          } else {
33            // Nemame token
34          }
35       }
36    };
37
38    task.execute(); // Spusteni ulohy
39 }
```

V tuto chvíli je uživatel přihlášen a my máme jeho autorizační token s kterým dále můžeme pracovat.

Kompatiblita - Android Support Library

Aplikace má být vyvedena pomocí aktuálních trendů a dle posledních designových doporučení. Android verze 4.x a 5.x obsahuje spoustu nových užitečných *API*, pomocí kterých se aplikace vyvíjejí jednodušeji a s více možnostmi. Na obrázku rozložení jednotlivých verzí 4.3 je vidět, že velká část zařízení není na nejnovější verze aktualizována a nejspíš ani nebude [14]. Je třeba vytvořit aplikace v moderním hávu při zachování zpětné kompatibility aby byla použitelná na co nejširším množství zařízení.

Balík *Android Support Library* je sada knihoven[15], které poskytují zpětně kompatibilní verze *API* přidaných do nových verzích Androidu. Je rozdělena podle úrovní zpětné kompatibility[16]. Knihovna obsahuje spoustu vizuálních prvků (*ActionBarCompat* - tlačítková lišta, *NavigationDrawer* - navigační prvek schovávající se po straně rozhraní aplikace, *ViewPager*, *CardView* a další) a také podpůrných tříd (*NotificationCompat* - správa notifikací, *Fragment* a *FragmentActivity* - znovupoužitelné komponenty, *RecyclerView* - prvek pro zobrazení velkého množství dat).

AppCompat

Část *AppCompat* obsahuje části pro tvorbu znovupoužitelných komponent a také podporu pro konzistenzní tlačítkovou lištu.

Fragmenty a FragmentActivity - byly představeny v Androidu 3.0 a slouží právě k tvorbě znovupoužitelných komponent. Před verzí 3.0 musela každá aktivita v aplikaci definovat odznova své chování a svůj obsah. Použitím *Fragmentů* lze do aktivit funkčnost poskládat a znovu použít. *AppCompat* navíc tuto funkci zpětně poskytuje až do verze 2.2.

ActionBar - opět představeno v Androidu 3.0, jedná se o hlavní vrchní tlačítkovou lištu, obohacenou o nadpis a menu. V Androidu 5.0 navíc přibyla možnost vlastního rozvržení *ActionBar*u a k přejmenování na *Toolbar*.

Fragmenty a ActionBar jsou tak základní komponenty v Android aplikaci, že jejich použití bylo takřka povinné.

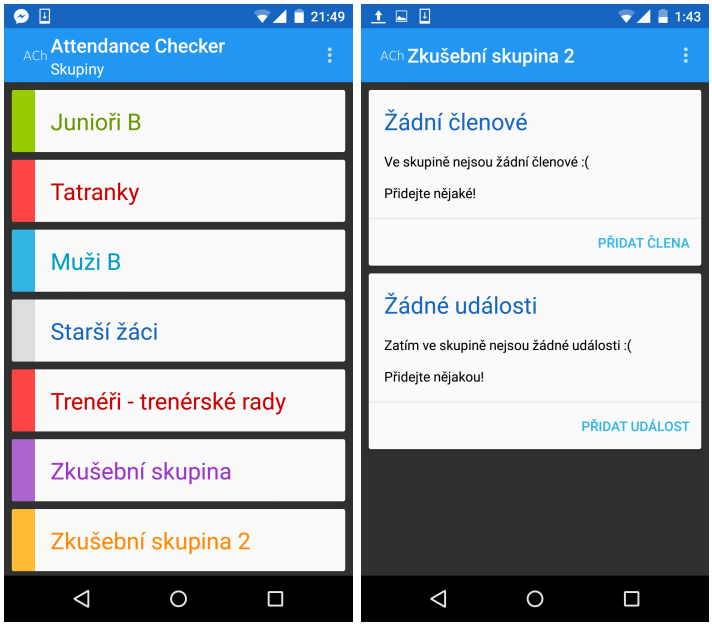
CardView

Vizuální prvek *CardView* umožňuje zobrazení informací uvnitř karet, které mají konzistentní vzhled napříč aplikacemi. Prvek umožňuje jednoduchou implementaci *Material* designu. Jedná se vlastně o kontejner s určitým stylem a vlastní *API* vytváření dalšího obsahu.

Použití je jednoduché - v *XML* souboru s rozvržením *UI* stačí požadovaný obsah obalit elementem *CardView*:

```
1 <android.support.v7.widget.CardView
2     android:layout_width="match_parent"
3     android:layout_height="wrap_content">
4
5     <TextView android:id="@+id/tv_name"
6         tools:text="Nazev skupiny"
```

[14] Na vinně jsou především výrobci mobilních zařízení, kteří aktualizaci pro konkrétní zařízení nevydají
[15] Dokumentace Android Support Library
[16] U některých částí se nedá zajistit dostatečně hluboka zpětná kompatibilita



Obrázek 6.6: Implementace UI - Zobrazení skupin a prázdná skupina - CardView

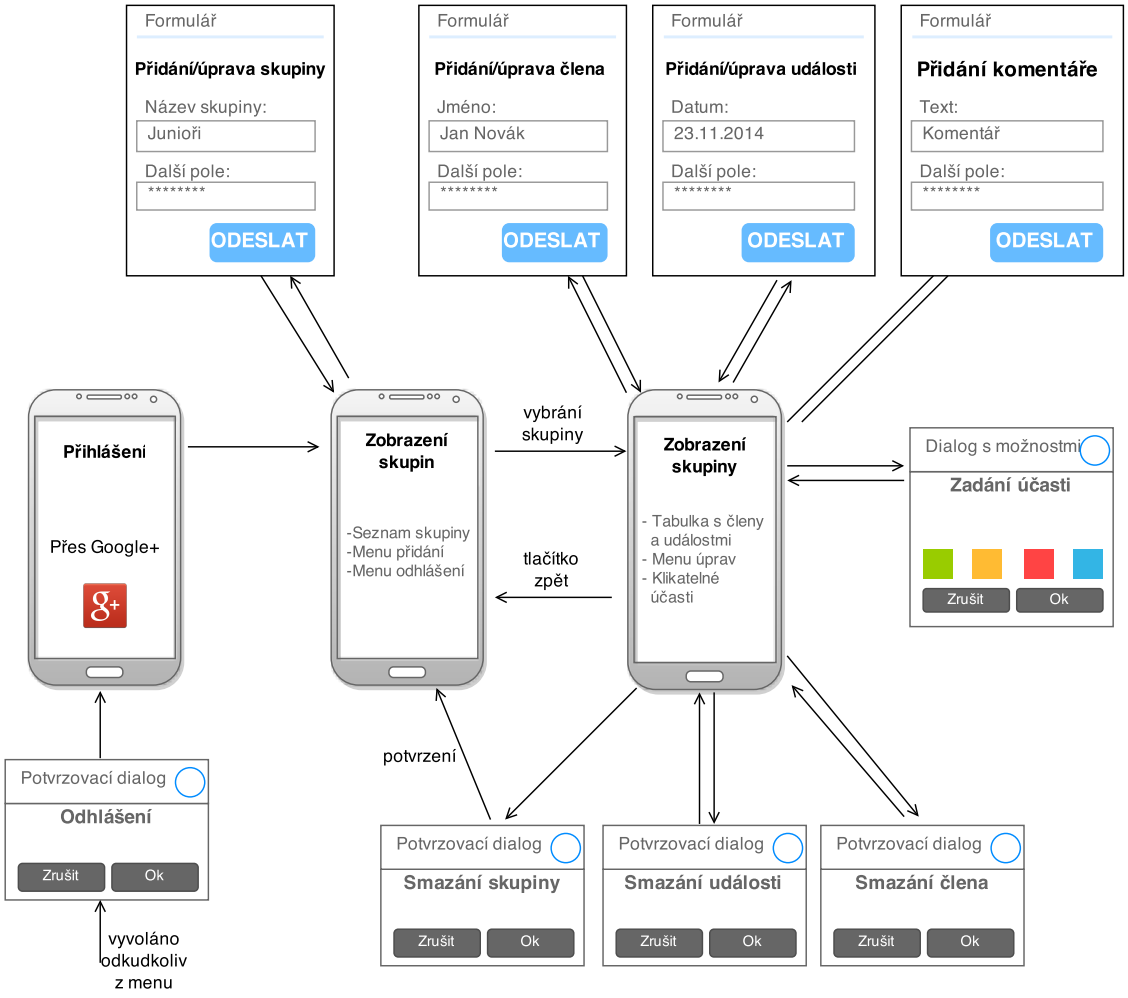
```
7           android:layout_width="wrap_content"
8           android:layout_height="wrap_content" />
9
10  </android.support.v7.widget.CardView>
```

V aplikaci je *CardView* použito pro zobrazení seznamu skupin uživatele (6.6 vlevo). Dále pak k zobrazení karet pro přidání prvního člena či události, pokud je daná skupina prázdná.

6.3.2 Uživatelské rozhraní

Schématické rozdělení jednotlivých částí uživatelského rozhraní je zobrazeno na obrázku 6.7. Aplikace se skládá z několika oken. Každé okno je realizováno vlastním *Fragmentem*.

- **Přihlášení** - Fragment s tlačítkem pro přihlášení. Tlačítko přesměruje uživatele na přihlášení pomocí knihovny z *Google+*. Po úspěšném návratu z přihlášení přesměruje na zobrazení skupin.
- **Zobrazení skupin** - Fragment obsahující seznam s vlastními skupinami. V menu *ActionBar*u položky pro přidání skupiny a odhlášení.
- **Přidání/úprava skupiny/člena/události/komentáře** - každý fragment obsahuje formulář pro přidání konkrétního objektu. Pokud je fragment vyvolán s nějakým ID parametrem, pak je objekt s tímto ID načten a formulář je předvyplněn pro úpravu tohoto objektu. Po vykonání akce dojde k návratu do zobrazení dané skupiny.
- **Smazání skupiny/události/člena** - Dialogový fragment s tlačítky pro potvrzení či zrušení akce. Po smazání skupiny dojde k návratu na seznam skupin.



Obrázek 6.7: Schématické zobrazení toku akcí uživatele

- **Zadání účasti** - Jednoduchý dialogový fragment obsahující informaci o vybraném členovi a události a tlačítky možných hodnot účasti, které v zájmu rychlosti zadávání zároveň potvrdí daný dialog.
- **Zobrazení skupiny** - Fragment, který má přehledně zobrazovat členy, události a účasti mezi nimi. Implementace je podrobněji popsána v další části.

UI zobrazení skupiny

Zobrazení skupiny má přehledně zobrazovat příslušné členy, události a kombinace účastí mezi nimi. Jako řešení bylo zvoleno zobrazení těchto dat do tabulky, kde každý řádek představuje jednoho člena a a každý sloupec jednu událost. Průsečíkem sloupce a řádku je pak účast člena na události.

Vzhledem k tomu, že počet členů ve skupině reálně bude pohybovat mezi 10-ti až 30-ti a počet událostí ve skupině se může pohybovat okolo 100ky a že mobilní zařízení mají omezenou velikost displeje je třeba aby se v datech přehledně pohybovalo a orientovalo. Byly vytipovány tři části, které tabulka musí splňovat:

- Skrolování horizontálně i vertikálně obsahem.
- Fixní řádek s názvy událostí a sloupec se jmény.
- Barevné odlišení typů účastí.
- Sestavení příslušných částí tabulky

Skrolování horizontálně i vertikálně obsahem

*SDK Android*u obsahuje komponenty pro skrolování obsahu buď vertikálně - *ScrollView* anebo horizontálně - *HorizontalScrollView*. Komponentu, která by uměla oběma směry neobsahuje, neboť to není doporučovaný postup. Kombinace obou komponent nefunguje - v tomto případě obsah buďto skroluje jedním či druhým směrem, ale ne plynule oběma směry, proto je potřeba zmíněné chování implementovat.

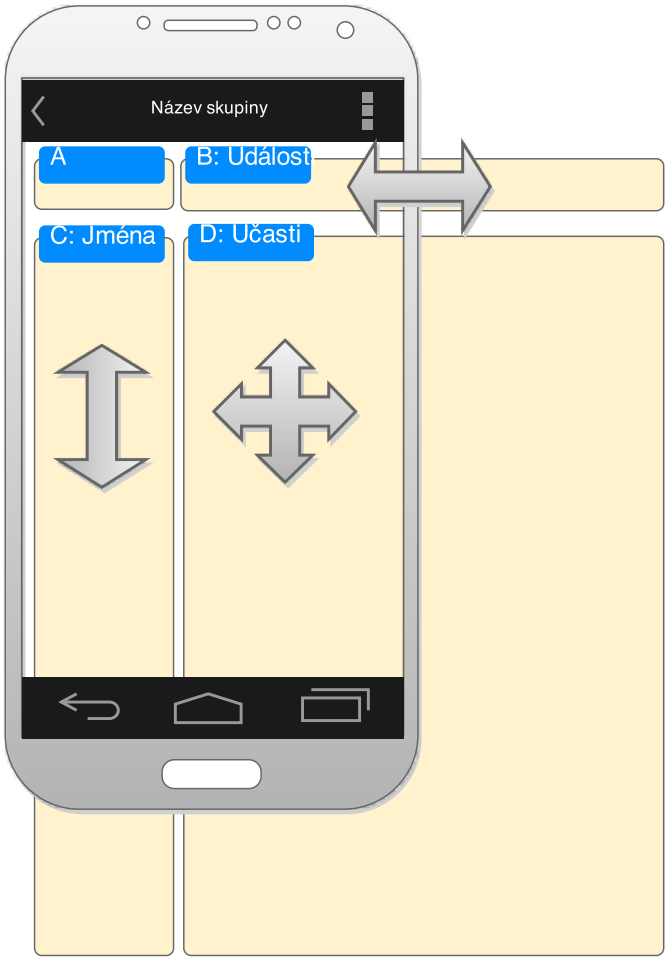
Všechny prvky *SDK* naštěstí nabízejí široké možnosti přepisu chování a existuje i spousta návodů [1] [8] jak si komponentu skrolující v obou směrech vytvořit.

Fixní řádek s názvy událostí a sloupec se jmény

Uživatel může odskrolovat do strany a v běžné tabulce by se tak mimo dosah zobrazení dostaly části pro identifikaci dané buňky. Řešením je vytvoření fixního řádku se jmény a sloupce s názvy událostí. Celý obsah ale musí být skrolovatelný a tak i tyto fixní části musí být skrolovatelné. Obrázek 6.8 zobrazuje řešení inspirované článkem [2].

Celý fragment je rozdělen na čtyři části:

- A - kontejner obsahuje pouze nehybné textové pole s řetězcem "Jméno".
- B - HorizontalScrollView - kontejner s horizontálně skrolující tabulkou, která má jen jeden řádek a její buňky jsou názvy událostí.



Obrázek 6.8: Rozvržení UI - Zobrazení skupiny

- C - ScrollView - kontejner s tabulkou skrolující vertikálně, která obsahuje sloupec s buňkami se jmény jednotlivých členů.
- D - komponenta 2DScrollView navržená v předchozí části. Skroluje v obou směrech a obsahuje jednotlivé účasti.

Fragment nyní obsahuje sloupec se jmény, řádek s událostmi a tabulku s účastmi. Každou komponentou lze skrolovat v jejím směru nezávisle. Následující kód ukazuje propojení jednotlivých částí, respektive propojení posluchačů událostí skrolování. Pokud je vyvolána uživatelem událost skrolování po části *C* se jmény, pak je i příslušně skrolováno s částí *D*. To samé platí pro část *B*. Pokud uživatel skroluje po částí *D*, pak je stav skrolování upraven pro částí *B* i *C*.

```
1  // Kontejnery jednotlivych casti
2  @InjectView(R.id.viewD) TwoDScroll view_D;
3  @InjectView(R.id.viewC) ScrollView view_C_clenove;
4  @InjectView(R.id.viewB) HorizontalScrollView view_B_udalosti;
5
6  // Vnitrni tabulky kontejneru
7  @InjectView(R.id.table) TableLayout table_D;
8  @InjectView(R.id.table_header) TableLayout table_B_udalosti;
9  @InjectView(R.id.table_names) TableLayout table_C_jmena;
10
11 @Override
12 public View onCreateView(LayoutInflater inflater, ViewGroup container,
13                          Bundle savedInstanceState) {
14    View v =  inflater.inflate(R.layout.fragment_skupina, container);
15
16    // Vlozeni kontejneru a tabulek
17    ButterKnife.inject(this, v);
18
19    // Propojeni skrolovani casti D s castmi B a C
20    viewD.setListener(new TwoDScroll.ScrollListener() {
21       @Override
22       public void onScroll(int x, int y) {
23          table_C_jmena.scrollTo(0,y);
24          table_B_udalosti.scrollTo(x, 0);
25       }
26    });
27
28    // Propojeni skrolovani casti C s hlavni casti D
29    view_C_clenove.setOnTouchListener(new View.OnTouchListener() {
30       @Override
31       public boolean onTouch(View view, MotionEvent event) {
32          table_D.scrollTo(
33             table_D.getScrollX(), view_C_clenove.getScrollY() );
34          return false;
35       }
36    });
37
38    // Propojeni skrolovani cast B s hlavni casti D
39    view_B_udalosti.setOnTouchListener(new View.OnTouchListener() {
40       @Override
41       public boolean onTouch(View view, MotionEvent event) {
42          table_D.scrollTo(
43             view_B_udalosti.getScrollX(), table_D.getScrollY() );
44          return false;
45       }
46    });
47
48    return v;
49 }
```

Obrázek 6.9: Ukázka UI - Zobrazení skupiny

Fragment nyní obsahuje řádek s názvy událostí (B), který je vždy navrchu, dále sloupec se jmény (C) vždy po levé straně a hlavní část tabulku s účastmi (D), kterou uživatel může skrolovat ve všech směrech a jejíž posun příslušně posouvá výše zmíněné fixní části (a naopak).

Barevné odlišení typů účastí

Poslední částí pro přehledné intuitivní rozhraní zobrazení skupiny bylo dostatečné odlišení účastí. Místo slov "ano", "nepřítomen", "omluven"a dalších bylo zvoleno použití zástupných znaků a odlišení decentními barvami. Každá buňka účasti musí mít stejnou velikost, aby byla zaručena stejnost celé tabulky. Obrázek 6.9 pak ilustruje přehledné zobrazení účastí, kde lze snadně identifikovat kdy jaký člen chyběl či jaká byla účast na konkrétním tréninku.

1 - Představuje účast člena na události.

0 - Představuje neúčast člena na události.

n - Představuje neomluveného člena na události.

o - Představuje omluveného člena.

Sestavení částí tabulky

Obrázek 6.9 zobrazuje implementaci tabulky. Vlevo je zobrazení tabulky při nulovém skrolování, vpravo pak výhled tabulky mírně skrolovaný dolů a vpravo.

- Buňka *příjmení* - je naprosto fixní.
- Sloupce s *příjmeními* - fixně vždy po levé straně, skrolující po ose Y. Buňky s příjmením jsou klikací, vyvolají dialog s úpravou vybraného člena.
- Řádek s nadpisy sloupců *jméno*, *průměr* a všemi *účastmi* - fixní vždy nahoře, skrolující po ose X. Názvy událostí jsou klikací, vyvolají dialog s úpravou dané události.
- Řádek s názvy *měsíce* - fixní, vždy jako druhý řádek, skrolující po ose X. Pro všechny události daného měsíce je název měsíce vypsán pouze jednou, což umožňuje co nejužší sloupce, datová tabulka tak může být více kompaktní a i se v ní lépe orientuje.

- Hlavní obsah tabulky - řádky jednotlivých členů, se sloupci se *jménem*, *průměrem* a konkrétními *účastmi*. Skrolující po ose X i Y. Účasti jsou klikací, vyvolají dialog změny účasti.

Tabulka se snaží data zobrazovat čistě bez zbytečných prvků navíc. Prezentace dat vypadá přehledně, snadno se v ní orientuje i při větším množství dat a obsahuje požadované informace.

Prázdná skupina
Pokud uživatel založí novou skupinu, pak při vstupu do ní by mu byla zobrazena prázdná tabulka. Vhodnější je napovědět s přidáním prvních dat. Implementování pomocí karet s CardView, které uživatele vybízejí k vložení dat zobrazeno na obrázku 6.6 vpravo.

Dialogy
Správa jednotlivých částí systému je řešena skrz nativní dialogy *Androidu*. Tyto dialogy mají vlastní nadpis a tlačítka potvrzení či zrušení akce. Obsahují příslušný formulář. Při úpravě dat je formulář předvyplněn stávajícími daty.

- Dialog přidání/úpravy skupiny - formulář s název skupiny, výběrem barevného rozlišení a popisem skupiny.
- Dialog přidání/úpravy člena - formulář se jménem a příjmením člena.
- Dialog přidání/úpravy události - formulář s názvem události, výběrem data a popisem.
- Dialog přidání/úpravy komentáře - formuláře s textem komentáře.
- Potvrzovací dialog smazání skupiny/člena/události - jasný stručný text smazání daného objektu.

Dialogy pro úpravu jednotlivých objektů obsahují navíc tlačítko pro smazání daného objektu. Kliknutí na dané tlačítko vyvolá varovný potvrzovací dialog.

6.4 Webové rozhraní

Pro vývoj webového rozhraní byl použit editor *Brackets* 1.0[17]. Editor je zaměřen na webový vývoj a obsahuje obsáhlé nástroje pro editaci *HTML*, *CSS* i *JS* souborů. Dále obsahuje nástroj pro živý náhled editace při propojení s prohlížečem, což je funkce která ušetří mnoho času. Vývojář má v prohlížeči otevřenou stránku s webem a při editaci se mu změny rovnou promítají v prohlížeči. Není třeba obnovovat stránku, *CSS* i *JS* změny jsou do prohlížeče promítnuty ihned, při změně *HTML DOM*u se stránka automaticky obnoví. Navíc při interakci se stránkou se v editoru zvýrazňuje právě vybraná část stránky. Dále při editaci jednotlivých prvků *DOM*u se uživateli zobrazují ve vyskakovacím okénku všechny styly které se na daný prvek aplikují a vývojář je může rovnou upravit. *Brackets* je moderní editor pro webový vývoj a spoustu operací vývojáři zjednodušuje.

[17] Brackets - http://brackets.io/

6.4.1 Použité knihovny

Ve webovém rozhraní jsou použity dvě knihovny. Firebase pro práci s databází a Handlebars pro tvorbu uživatelského rozhraní.

Správa databáze - Firebase

Použití Firebase je totožné s částí mobilního klienta, neboť knihovny jsou navrženy aby na všechn platformách fungovaly stejně. Použití Firebase je naznačeno v kapitole o technologiích 2.2. I ve webovém rozhraní je skript s třídou *DAO*[18] využívající vzor Singleton/Jedináček. Tato obstarává veškerou komunikaci s databází.

Třída *DAO* a veškerá práce s *Firebase* se skládá z částí:

- *Firebase* odkaz na data přihlášeného uživatele.
- Autorizace uživatele na serveru Firebase.
- Objekt *DAO*, který poslouchá na změny dat.
- Metody pro správu skupin, členů, událostí, účastí v databázi.

6.4.2 Uživatelské rozhraní

Obrázek 6.10 zobrazuje rozvržení *UI* webové aplikace. V *JS* aplikaci jsou tyto části generovány u klienta pomocí šablonového systému. Použití tohoto systému je popsáno v další části. Kromě použití šablon, tvořících rozhraní *UI* aplikace nijak nevybočuje z tradiční tvorby webových stránek. Obsahuje *HTML* soubor se strukturou rozhraní aplikace, soubor s definicí kaskádových stylů (*CSS*) se vhledem aplikace a soubory s javaskriptem obsahující funkční části aplikace.

Tvorba UI - Šablonovací systém Handlebars

Část uživatelského rozhraní se načítá ze vstupního *HTML* souboru. Hlavní funkční částí je ale generování zobrazení aktuálních dat načtených z databáze. Toto generování se provádí v hlavním skriptu aplikace. Jelikož skript manipuluje přímo s *DOM*em a upravuje ho, bylo vhodné použít nějaký šablonový nástroj pro ulehčení práce.

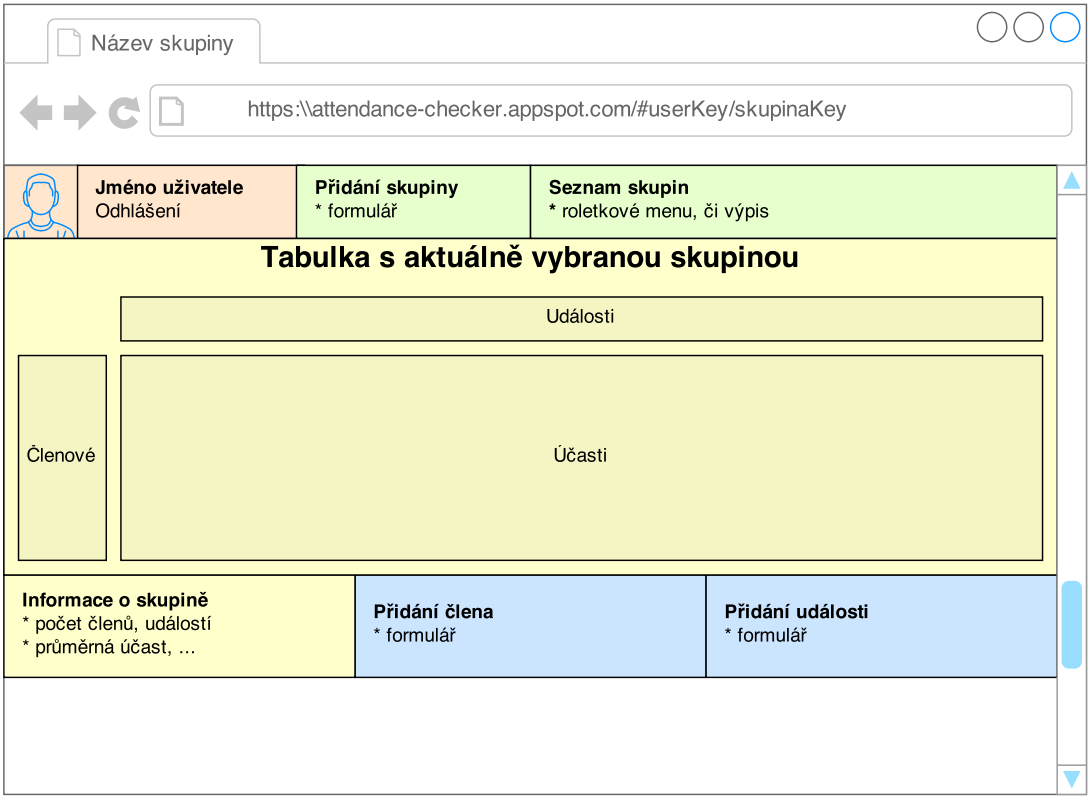
Knihovna *Handlebars*[19] umožňuje tvoření šablon jednotlivých bloků (viz obrázek 6.10) aplikace, do kterých dosazuje načtená data z databáze.

Použití šablon umožňuje oddělení části prezentace rozhraní a manipulace s daty.

Ukázka šablony se zobrazením profilu přihlášeného uživatele:

```
1 {{#if . }}
2   <img class="avatar" src="{{ google.picture}}" />
3   <div class="profil">
4     {{google.displayName}}
5     <br>
```

[18] DAO - data access object - objekt pro přístup k datům.
[19] Domovská stránka HandlebarsJS



Obrázek 6.10: Rozvržení částí ve webovém rozhraní

```
6 |     <a id="logout" href="#">Odhlasit se</a>
7 |   </div>
8 |   {{else}} Neprihlasen. <a id="login" href="#">Prihlasit se</a>
9 | {{/if}}
```

Ve skriptu pak dochází k načtení této šablony, předají se jí data a nakonec se zobrazí v příslušném kontejneru. Data v tomto případě mají formát:

```
1 | {
2 |   google:
3 |     picture: "url-obrazku-uzivatele",
4 |     displayName: "jmeno uzivatele"
5 |
6 | }
```

Pokud jsou předána data šabloně vykoná se blok *#if*, do atributu *src* tagu *img* se předá hodnota *google.picture*, do kontejneru *div.profil* se vypíše jméno uživatele. Při předání prázdných dat (uživatel není přihlášen) se vykoná blok *else*.

Kompilace šablon

Šablony jsou *HTML* soubory, které ale načítá a zpracovává knihovna *Handlebars*. Ta si *HTML* prezentaci převádí do Javaskriptové což je výpočetně náročné. *Handlebars* ale nabízí možnost zkompilování[20] šablon do oné Javaskriptové prezentace a uložení do statických souborů. Při používání aplikace se pak načtou již tyto přeložené soubory, což urychlí chod aplikace.

Prekompilace je dostupná skrz balík pro *node.js*. *Node.js* je platforma běhového prostředí pro javaskriptové aplikace. Po nainstalování balíku *Handlebars* do prostředí, lze pak spusit prekompilaci příkázem *handlebars* a parametry vstupních souborů a názvem výstupního souboru.

Kompilační příkaz:

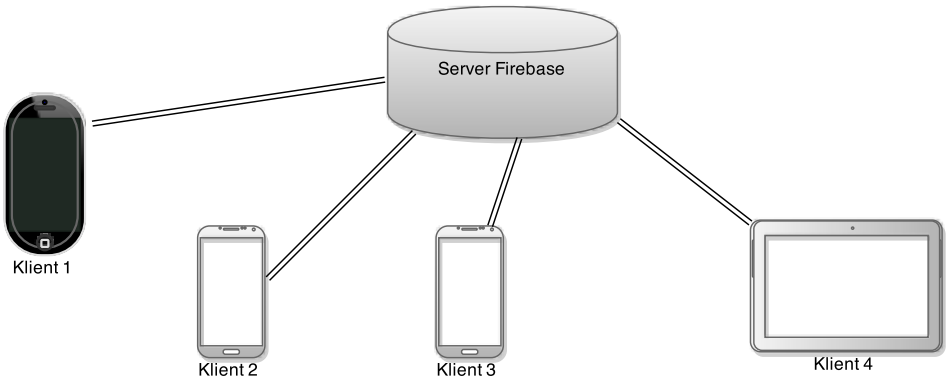
```
1 | handlebars templates/*.handlebars -f compiled.js
```

Tento příkaz zkompiluje všechny šablony ve složce *templates* a použitím přepínače *-f* je sjednotí do jednoho výstupního souboru. Šablon je v aplikaci několik, bez kompilace by prohlížeč uživatele musel načítat každou zvlášť a poté ji zpracovávat. Použitím prekompilace dochází ke snížení počtu načítaných souborů na jeden, odpadává nutnost převádění šablon a stačí použít pouze běhové prostředí *handlebars.runtime*, které neobsahuje kompilační část. Celá knihovna *handlebars* čítá 120kB, *handlebars.runtime* pouze 19.8kB což šetří množství přenesených dat.

6.5 Spolupráce přihlášených klientů

Výhodou *Firebase* a jejich knihoven je využití *WebSockets*. Pokud má uživatel spuštěnou aplikaci, pak je tato spojená se serverem *Firebase*, čili s databází a naslouchá na jakékoliv

[20] http://handlebarsjs.com/precompilation.html



Obrázek 6.11: Spolupráce připojených klientů

změny. Pokud je připojeno více klientů, pak všichni jsou ihned informování o změnách, které některý z nich provedl.

Obrázek 6.11 ilustruje přípojení různých zařízení k databázi. Pokud například klient 1 je přihlášený oprávněný uživatel a klienti 2, 3, 4 jsou uživatelé prohlížející skupinu klienta 1. Pak při změně dat u klienta 1 (například přidání nové události a zadání účastí s ní spojených) se v reálném čase promítají tyto změny na zařízeních ostatních připojených klientů. Není třeba načítat znovu stránku či obnovovat nijak data. Další výhodou je, že jsou zasílány pouze dílčí změny dat, ne všechny data, což šetří datový tok.

6.6 Použité návrhové vzory

6.6.1 Singleton vzor

Česky zvaný jedináček, je tradiční vzor programování, který řeší situaci, kdy je potřeba aby v aplikaci byla pouze jedna instance konkrétní třídy. Vzor zabezpečí, že třída bude mít pouze jedinou instanci a poskytne k ní globální přístupový bod.

Základním řešením je, že třída má privátní konstruktor (nelze jí tedy nikde jinde vytvořit), statickou privátní proměnou typu dané třídy a statickou metodu (tradičně *getInstance()* nebo *instance()*), která vrací onu privátní proměnnou. Zároveň zajišťuje vytvoření dané proměnné nebyla-li ještě inicializována.

Singleton byl v aplikaci využit pro vytvoření *DAO* třídy, která sloužila k celkové komunikaci s databází přihlášeného uživatele.

6.6.2 Observer vzor

Vzor *Observer*[21] definuje *vydavatele*, který si udržuje seznam svých *odběratelů*. Tyto *odběratele* informuje o změně svého stavu pomocí jím definovaného veřejného rozhraní. Rozhraní tak musí definovat metody pro přihlášení a odhlášení odběratele a pro informování o změně stavu.

[21] Informace o vzoru *Observer* - wikipedia a článek na itnetwork.cz

Využití tohoto vzoru v našem systému je v oddělení uživatelského rozhraní od databázových volání. Uživatel provede nějakou akci, která pracuje s databází. Tato akce se vykoná ve vedlejším vlákně. Až akce proběhne, pak *DAO* vyšle událost o potvrzení akce a příslušné částí rozhraní aktualizují svůj stav.

Konkrétní příklad:

- Uživatel přidá člena do skupiny.
- *DAO* provede akce ve vedlejším vlákně. Ta může trvat i vteřiny, v závislosti na rychlosti připojení.
- Uživatelské rozhraní je stále responzivní na akce uživatele.
- *DAO* akce skončí, *DAO* rozešle událost o provedení akce.
- Odběratelé akce aktualizují stav. V tomto případě by se aktualizoval stav počet členů ve skupině a přidá se člen do zobrazení.

Vzor je realizován pomocí knihovny *EventBus* a popsán v části *Komunikace v rámci aplikace* 6.3.1.

6.6.3 Factory method vzor

Vzor *Tovární metoda*, definuje rozhraní pro vytváření objektu, které nechává potomky rozhodnout o tom, jaký objekt bude vytvořen. Fakticky jde o metody, které vytvoří daný objekt se zadanými parametry.

Využití v naší aplikaci se nachází v části mobilního klienta u tvorby *fragmentů*. Abstraktní *fragment* je prvek uživatelského rozhraní. Třídy, které tento *fragment* rozšiřují definují jeho vlastnosti a chování. Příkladem fragmentu je *SkupinaFragment*, který obstarává chování zobrazení konkrétní skupiny, dále *SkupinyFragment*, který zase zobrazuje seznam skupin. Veškeré *dialogy* jsou také potomci *fragmentu*. *Fragment* ze své definice musí být prázdný konstruktor a tak se předání dat *fragmentu* řeší právě tovární metodou a předání dat balíkem *Bundle*.

Každý implementovaný *fragment* má tak metodu typu *newInstance*(...) s potřebnými vlastními parametry, která vytvoří daný typ *fragmentu* prázdným konstruktorem, předá vytvořenému fragmentu data pomocí *Bundle* a tento vytvořený *fragment* vrátí.

Příklad užití vzoru:

```
1  public static ClenEditDialog newInstance(String key, String jmeno) {
2    // Vytvoreni prazdnym konstruktorem
3    ClenEditDialog dialog = new ClenEditDialog();
4
5    // Nastaveni argumentu
6    Bundle data = new Bundle();
7    data.putString(ARG_KEY, key);
8    data.putString(ARG_JMENO, jmeno);
9
10   dialog.setArguments(data);
11   return dialog;
12 }
```

Android totiž při změně stavu aplikace vytváří zobrazené *fragmenty* od znova. Změna stavu může být například změna orientace displeje, návrat do aplikace z jiné, odemknutí obrazovky, příchozí hovor a další. Každý znovu vytvořený *fragment* využívá právě prázdný konstruktor. Pokud by vývojář použil vlastní rozšířený konstruktor, tak ten by se zavolal jen při jeho prvním volání, ale při znovu vytvoření samotným Androidem již, ztratila by se tak data. Aby se data neztratila, tak Android si pamatuje právě předaný *Bundle* danému *fragmentu*. V předchozí ukázce je zobrazeno využití prázdného konstruktoru a předání *Bundle* novému *fragmentu*. Data z Bundle pak může fragment znovu načíst.

6.6.4 ViewHolder vzor

ListView je prvek *UI*, který zobrazuje data ve vertikálním posunovacím seznamu. Data přichází z přidruženého *ListAdapter*u.

Vzor *ViewHolder* umožňuje přístup ke každé položce Listu bez nutnosti vyhledávání jednotlivých textových polí, tlačítek a ostatních *UI* prvků s kterými má být manipulováno. Snižuje počet volání metody findViewById, která je při skládání *UI* jedna z nejnáročnějších[22]. Tento vzor ulehčuje hlavně při rolování seznamem, díky čemuž je tento plynulejší. Činí tak uchováváním referenci na jednotlivé *UI* prvky v rámci jedné položky seznamu a tím se odstraňuje nutnost je náročně vyhledávat.

V následující ukázce je znázorněno, jak se vytváří jedna položku Listu. Volá se při každém novém zobrazení položky. (Při rolování pohledu, ...)

```
1  class MyAdapter extends BaseAdapter {
2
3    @Override
4    public View getView(int position, View convertView, ViewGroup parent) {
5       View view = mInflater.inflate(R.layout.listitem, parent, false);
6
7       // metody narocne na CPU
8       // vola se pri kazdem zobrazeni polozky!!
9       TextView tv_username = (TextView) view.findViewById(R.id.tv_username);
10      TextView tv_mail = (TextView) view.findViewById(R.id.tv_username);
11
12      // prace s tv_username a tv_mail;
13
14      return view;
15   }
16 }
```

Z této ukázky je vidět, že metoda *findViewById* se volá při každém překreslení Listu. List může obsahovat i tisíce položek, což může velmi vytížit procesor zařízení. Vzor *ViewHolder* optimalizuje toto překreslování ukládáním odkazů na jednotlivé prvky objektu přímo do objektu. Eliminuje tak opakování volání metody *findViewById*.

Použití vzoru ilustruje tato ukázka:

```
1  class MyBetterAdapter extends Base Adapter {
2
3    @Override
4    public View getView(int pozice, View convertView, ViewGroup parent) {
5       ViewHolder holder;
```

[22] odkaz na článek Testování výkonu při použití ViewHolder vzoru

```
6
7           // vola se pouze pri prvnim zobrazeni
8           if(convertView == null) {
9               convertView = mInflater.inflate(
10                  R.layout.skupina_list_item,
11                  parent,
12                  false
13              );
14
15              holder = new ViewHolder(convertView);
16
17              // ulozeni odkazu na textove pole primo do objektu
18              convertView.setTag(holder);
19          } else {
20              // nacteni odkazu z objektu - zadne narocne volani!
21              holder = (ViewHolder) convertView.getTag();
22          }
23
24          // prace s holder.et_username, holder.et_mail, ...
25
26          // ...
27
28          return convertView;
29      }
30
31      static class ViewHolder {
32          EditText et_username;
33          EditText et_mail;
34          EditText et_pass;
35
36          // vola se pouze pri prvotnim zobrazeni
37          ViewHolder(View view) {
38              this.et_username = (EditText) view.findViewById(R.id.et_username);
39              this.et_mail = (EditText) view.findViewById(R.id.et_mail);
40              this.et_pass(EditText) view.findViewById(R.id.et_pass);
41          }
42      }
43  }
```

Tento vzor byl *Googlem* doporučován při využívání ListView. Ve verzi Android 5.0 Lollipop přichází s rozšiřujícím RecyclerView[23], který tento vzor vynucuje.

Statickou třídu *ViewHolder* lze pak ještě vylepšit pomocí *ButterKnife* na:

```
1       static class ViewHolder {
2           @Inject(R.id.tv_username) EditText et_username;
3           @Inject(R.id.tv_mail) EditText et_mail;
4           @Inject(R.id.tv_pass) EditText et_pass;
5
6           ViewHolder(View view) {
7                ButterKnife.inject(this, view);
8           }
9       }
```

[23] developer.android.com - dokumentace RecyclerView

6.7 Struktura kódu mobilního klienta

Hlavní části aplikace jsou třídy pro dialogy, tříd pro testy, složkou s grafikou, složkou s rozvržením *UI*. Hlavními soubory pro zhotovení aplikace jsou *AndroidManifest.xml* a sestavovací skript *build.gradle*.

- build - zkompilované soubory
- src
  - androidTest - všechny testy
  - main
    - java
      - * adapters - Adaptéry s daty
      - * data - datové objekty
      - * dialogs - Dialogy aplikace
      - * events - události EventBusu
      - DAO.java - Třída pro komunikaci s Firebase a správou dat
      - MainActivity.java - vstupní aktivita aplikace
      - třídy SkupinaFragment, SkupinyFragment - jednotlivé fragmenty
      - TwoDScroll.java - implementace kontejneru skrolujícího oběma směry
    - res
      - * drawable - Grafika aplikace
      - * layout - Grafické rozvržení aplikace
      - * menu
      - * values - Texty a hodnoty použité v aplikaci
    - AndroidManifest.xml - Manifest s informacemi o aplikaci
- build.gradle - skript pro sestavení aplikace
- proguard.rules - pravidla pro kompilaci aplikace

6.8 Struktura kódu webového rozhraní

Webové rozhraní obsahuje vstupní soubor *index.html*, dále skripty *app.js* a *dao.js* s funkcemi aplikace, složku s šablonami rozhraní, knihovny a styly aplikace. V neposlední řadě pak sestavovací skripty a soubor *firebase.json* pro identifikace aplikace.

- templates - složka s šablonami Handlebars
- compile.bat - skript pro zkompilování šablon
- deploy.bat - skript pro nahrání na server
- styly.css - styly UI

- index.html - vstupní soubor aplikace
- app.js - hlavní skript aplikace
- compiled.js - zkompilované šablony
- dao.js - skript pro komunikaci s Firebase a správou dat
- handlebars.runtime-v2.0.0.js - knihovna Handlebars
- helpers.js - vlastní funkce pro vytváření šablon
- firebase.json - nastavení aplikace

Kapitola 7

Testování

Každý kus softwaru by měl být řádně otestován. Testovat by se měla funkčnost všech částí systému, kompatibilita s požadovanými platformami/zařízeními a k otestování správného návrhu aplikace zejména ze strany uživatelského rozhraní a zážitku z používání by měli být využiti nezávislý testeři. Všem těmto částem se věnuje tato kapitola.

Veškerý vývoj probíhal na notebooku s Intel Core i5-4300, 8GB RAM, SSD diskem a 64-bit Windows 8.1. Byl použit jako hlavní zařízení pro testování webového rozhraní, pro testování mobilní aplikace byl primárně využit mobilní telefon Google Nexus 5.

7.1 Automatické testování rozhraní mobilního klienta - Espresso

Pro Android existují nástroje pro testování *UI* s názvem *Espresso*. Tento nástroj je součástí *Android Support Library* a je přímo podporován *Android Studiem*.

Pro základní případy užití aplikace byly vytvořeny příslušné testy. *Espresso* nabízí automatizování většiny běžných interakcí s uživatelským rozhraním. Před provedením každé akce automaticky počká až doběhnou všechny akce, ať už se jedná o animace prostředí, otevírání nových oken či pracích běžících ve vedlejším vlákně pomocí *AsyncTask*.

Ukázkový test[1], který otestuje přidání nové skupiny, zkontroluje její existenci, smaže jí a zkontroluje smazání:

```
1  public void testPridaniASmazaniSkupiny() {
2    // kliknuti na kontextove menu
3    openActionBarOverflowOrOptionsMenu(getInstrumentation().getTargetContext());
4
5    // vybrani polozky Pridat skupinu
6    onView(withText("Pridat skupinu")).perform(click());
7
8    // vepsani zkusebniho textu skupiny
9    onView(withId(R.id.et_name)).perform(typeText("TestSkupina"));
10
11   // zavreni softwarove klavesnice
12   closeSoftKeyboard();
13
14   // kliknuti na tlaciko Pridat
15   onView(withText("PRIDAT")).perform(click());
16
```

[1] Test předpokládá, že uživatel je v aplikaci přihlášen.

```
17    // kontrola, ze v~datech existuje zkusebni skupina
18    onData(AllOf.allOf(is(instanceOf(SkupinyAdapter.SkupinaInfo.class)),
19      skupinaWithContent("TestSkupina"))).check(matches(isDisplayed()));
20
21    // vybrani zkusebni skupiny
22    onData(AllOf.allOf(instanceOf(SkupinyAdapter.SkupinaInfo.class),
23      skupinaWithContent("TestSkupina"))).perform(click());
24
25    // kliknuti na kontextove menu
26    openActionBarOverflowOrOptionsMenu(getInstrumentation().getTargetContext());
27
28    // vybrani polozky Upravit skupinu
29    onView(withText("Upravit skupinu")).perform(click());
30
31    // kliknuti na tlaciko Smazat
32    onView(withText("SMAZAT")).perform(click());
33
34    // kontrola, ze v datech neexistuje zkusebni skupina
35    onData(AllOf.allOf(is(instanceOf(SkupinyAdapter.SkupinaInfo.class)),
36      skupinaWithContent("TestSkupina"))).check(doesNotExist());
37  }
```

Pokud všechny akce proběhnou, pak test dopadl úspěšně.

7.2 Uživatelské testování mobilního klienta

Vývojové nástroje pro Android obsahují emulátor zařízení. Android je v základu tvořen pro jinou architekturu a tak emulátor musí emulovat spoustu funkcí, což způsobovalo i na celkem výkoném vývojovém notebooku pomalost emulátoru. Druhou možností bylo použití *Genymotion*[2], což je další emulátor Android s vlastním virtuálním strojem, jehož rychlost už působí plynule. I tak ale tento emulátor nepodporoval všechny funkce reálného zařízení a nepůsobil pohodlným dojmem. Proto pro vývoj byly zvoleny dostupné reálné zařízení:

- **Google Nexus 5** s Androidem 5.0.1 - 5"displej s FullHD rozlišením 1080x1920 pixelů, čtyřjádrovým procesorem 2,26GHz a 2GB RAM patří do kategorie výkoných zařízení a tak byl zvolen jako hlavní testovací prostředek.

- **Samsung Galaxy SII** s neoficiálním Androidem 4.4.3 - starší a pomalejší zařízení z roku 2011 s dvoujádrovým procesorem 1.2GHz, 1GB RAM a displejem 4,3"s rozlišením 480 x 800 pixelů.

- **Google Nexus 7** s Androidem 5.0.2 - tablet s 7"displejem s rozlišením HD 720x1280 pixelů, čtyřjádrovým procesorem 1.3GHz a 1GB RAM.

Na všech zařízeních proběhly výše zmíněné testy, byly na nich i prováděny subjektivní testy použitelnosti. I přes nižší výkon Galaxy SII, který by se v dnešní době řadil do lowendů na něm aplikace a práce s ní probíhala plynule.

[2]Genymotion - alternativní emulátor Androidu - https://www.genymotion.com/

7.2.1 Testeři

Aplikace ve verzi kdy už obsahovala funkce správy skupin a konkrétní skupiny byla distribuovaná třem testerům. Ti aplikaci testovali a běžně používali k vedení docházky vlastního družstva v našem týmu.

1. **Samsung Galaxy S3 mini** s Androidem 4.2
2. **Sony Xperia Z3 Compac**t s Androidem 4.3
3. **Samsung Galaxy S4** s Androidem 4.4.2

Testeři aplikaci běžně používali a zasílali připomínky k uživatelskému zážitku. Byl zpozorován jev, kdy po změně účasti člena je rozhraní chvíli nereaktivní. Na optimalizaci zobrazení bude zapracováno v dalším vývoji aplikace.

7.2.2 Datová náročnost

Ne všude je k dispozici *WiFi* či rychlá *3G* síť pro síťovou komunikaci. Jedním z požadavků byla nenáročnost a rychlost aplikace. Komunikace s *Firebase* skrz jejich knihovny využívá *WebSockets*, skrz které jsou zasílány jen opravdu nejnutnější informace a dílčí změny v datech. Pomocí vestavěných funkcí byly monitorovány síťové přenosy. Testovacím vzorkem byla skupina 23 členů o 47 událostech. V této skupině byly měněny účasti a přidávány nové události. Se servery *Firebase* za tuto dobu byly vyměněny jednotky maximálně desítky kB, které i pomalejší *2G* síť přenese v okamžiku. Paradoxně tak nejnáročnější částí na síťovou komunikaci je přihlášení k účtu *Google+*, které zabere okolo 100kB. I tak se ale jedná o nízký datový objem, navíc přihlášení uživatel provede pouze jednou. Aplikace je tedy datově nenáročná.

7.3 Testování webového rozhraní

Webové rozhraní bylo testováno hlavně z hlediska kompatibility napříč webovými prohlížeči. Testování probíhalo na nejběžnějších *evergreen*[3] prohlížečích.

Testovaná prostředí a prohlížeče:

- Windows 8.1 - Internet Explorer, Chrome, Firefox.
- Windows 7 - Internet Explorer, Chrome.
- Linux Mint 17.0 ve VirtualBoxu - Chromium ve VirtualBoxu.
- Nexus 5 a Android 5.0.1 - vestavěný prohlížeč, Chrome.
- Galaxy Note 2 Android 4.4.2 - vestavěný prohlížeč, Dolphin browser.

[3] Evergreen prohlížeč - takový prohlížeč, který se sám stará o svojí aktualizace. Uživatel tak vždy používá nejaktuálnější verzi daného prohlížeče.

Testování probíhalo na verzích prohlížečů aktuálních ke dni 18.12.2014. Otestované byly standardní případy užití:

- Přihlášení a odhlášení uživatele.
- Přidání, úprava a smazání skupiny.
- Přidání, úprava a smazání člena skupiny
- Přidání, úprava a smazání události skupiny.
- Změna účasti člena na události.
- Přidání komentáře.
- Zobrazení skupin.
- Zobrazení skupiny.

Výše uvedené užití proběhly na všech testovaných prostředích korektně což potvrzuje, že při vývoji javaskriptové části nebylo použito žádných nestandardních funkcí či *API*. Už z podstaty javaskriptové aplikace náš systém při vypnutém javaskriptu nefungoval.

Úspěšnost testování na mobilních prohlížečích přidává možnost použití aplikace i na zatím nepodporovaných platformách. Pojem *nepodporovaná platforma* tak v tomto případě znamená, že pro danou platformu není zatím nativní aplikace, ale systém může být používán skrz webové rozhraní.

Kapitola 8

Závěr

Cílem této práce bylo zanalyzovat současné možnosti řešení a případně navrhnout vlastní řešení vedení docházky v klubu. Navrhnuté řešení v podobně mobilního klienta pro OS Android, přístupu přes webové rozhraní a cloudové databáze se podařilo vypracovat do plně funkčního stádia. Některé funkční požadavky s nízkou prioritou kvůli časovému plánu byly odloženy na případný další vývoj aplikace. Sem patří požadavek zobrazování grafu účastí a optimalizace uživatelského rozhraní mobilní aplikace pro různé velikosti displeje zařízení.

Do případné budoucí práce na systému patří optimalizace *zobrazení skupiny* na mobilním zařízení, kde v části *Testování* bylo pozorováno zpomalení zobrazení. Dále se ve webové části dá zapracovat na uživatelském rozhraní, tak aby bylo responzivní - použitelné pro různé velikosti displeje. V budoucím vývoji aplikace by se určitě mělo zapracovat na komplexnějších možnostech sdílení skupin.

Celkově se zadání podařilo úspěšně vypracovat a už teď systém využívá většinová část trenérů našeho klubu. Trenéři uvítali jednoduchost a efektivnost zadávání docházky. Zároveň tato docházka je k prezentaci u veřejnosti dostatečně přehledná. Tímto práce na systému nekončí, plánuje se její budoucí vývoj, zejména dotažení systému do stavu, kdy bude být moc distribuován i ostatním subjektům, tak aby se využití rozšířilo i mimo náš klub.

V průběhu tvorby této diplomové práce jsem se seznámil s celým průběhem vývoje komplexního systému, který se skládá z odlišných části. Kombinace různých atraktivních technologií mě bavila a zkušenosti nabité při vytváření této práce pro mě budou přínosem v dalším uplatnění.

Literatura

[1] Clark, M. *Android Two-Dimensional ScrollView* [online]. Dostupné z: <http://goo.gl/TZGnEb>.

[2] Dalisay, M. *Android Table Scroll with Fixed Header and Column* [online]. Dostupné z: <http://goo.gl/Hi8QwT>.

[3] D.Smith, M. *Float Label Form Interaction* [online]. Dostupné z: <https://dribbble.com/shots/1254439-GIF-Mobile-Form-Interaction>.

[4] Fowler, M. *Event Sourcing* [online]. Dostupné z: <http://martinfowler.com/eaaDev/EventSourcing.html>.

[5] Google. *Developer Android* [online]. Dostupné z: <https://developer.android.com/>.

[6] Inc., G. *Platform Versions* [online]. Dostupné z: <https://developer.android.com/about/dashboards/index.html>.

[7] Post, M. *Android annotation performance unravelled* [online]. Dostupné z: <http://goo.gl/hIiyOe>.

[8] XCaffeinated. *Large Image Scrolling Using Low Level Touch Events* [online]. Dostupné z: <http://goo.gl/B8Dh4K>.

[9] ZACH MCCORMICK, V. U. V. U. D. C. S. *Data synchronization patterns in mobile application design.*

Příloha A

Seznam použitých zkratek

API (Application Programming Interface) - rozhraní pro programování aplikací

CDN (Content Delivery Network) - distribuovaný systém serverů pro dodání dat

CSS (Cascade StyleSheet) - jazyk pro úpravu vzhledu prvků webové stránky

CSV (Comma Separated Values) - soubor s daty, kde jsou jednotlivé položky odděleny čárkou

DAO (Data Access Object) - objekt pro práci s daty

DOM (Document object model) - objektový model dokumentu

GMS (Google Mobile Services) - služby Google pro mobilní zařízení

HD (High Definition) - vysoké rozlišení

HTML (HyperText Markup Language) - značkovací jazyk pro hypertext

IDE (Integrated Development Enviroment) - integrované vývojové prostředí

JS (JavaScript) - interpretovaný skriptovací jazyk využívaný hlavně při tvorbě webových stránek

JSON (JavaScript Object Notation) - formát zápisu dat

ORM (Object-relational mapping) - mapování databázových entit na objekty

OS (Operation system) - Operační systém

RAM (Random Access Memory) - paměť s libovolným přístupem

REST (Representational State Transfer)

SDK (Software Development Kit) - sada nástrojů pro vývoj software

SSD (Solid State Drive) - pevný disk bez pohyblivých částí

UI (User interface) - uživatelské rozhraní

UML (Unified Modeling Language) - univerzální modelovací jazyk

XML (eXtensible Markup Language) - rozšiřitelný značkovací jazyk

Příloha B

Uživatelská a instalační příručka

B.1 Příručka mobilní aplikace

Po spuštění aplikace je rovnou nabídnuta obrazovka obsahující tlačítko pro přihlášení.

Přihlášení

Kliknutím na tlačítko *přihlášení* je uživatel přesměrován na tradiční přihlášení do Google+ účtu. Následuje potvrzovací dialog přidělující oprávnění aplikaci (obrázek B.1 vlevo).

První kroky

Po první přihlášení se uživateli naskytne na prázdný seznam skupin (obrázek B.1 uprostřed). Po kliknutí na odkaz *PŘIDAT* se zobrazí dialogové okno s formulářem pro vytvoření nové skupiny (obrazek B.1 vpravo).

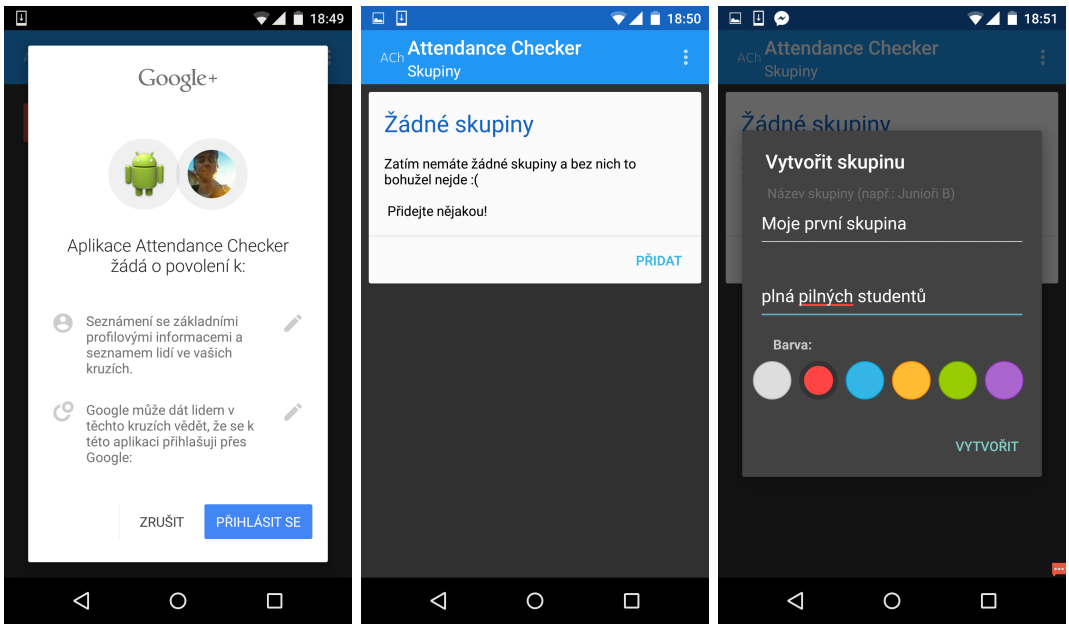
Po vytvoření skupiny se tato zobrazuje v seznamu skupiny (obrázek B.2 vlevo). Po zobrazení skupiny tato zeje prázdnotou. Uživatel je vyzván pro přidání dat. Po přidání členů a událostí (B.3 obrázek uprostřed) se uživateli zobrazí tabulka s daty skupiny. Skupina s nějakými vyplněnými daty je vyobrazena na obrázku B.2 vpravo.

Správa účastí

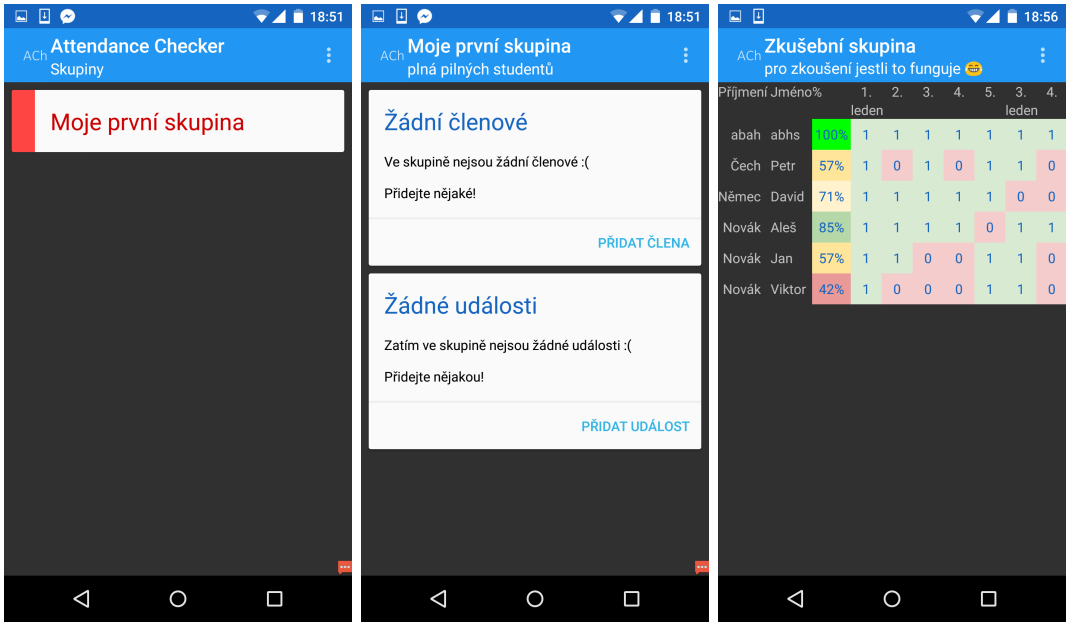
Hlavní funkcí je zadávání účasti, které probíhá kliknutím na příslušnou buňku. Poté se zobrazí dialogové okno s výběrem možností účasti (obrázek B.3 vlevo).

B.2 Příručka webového rozhraní

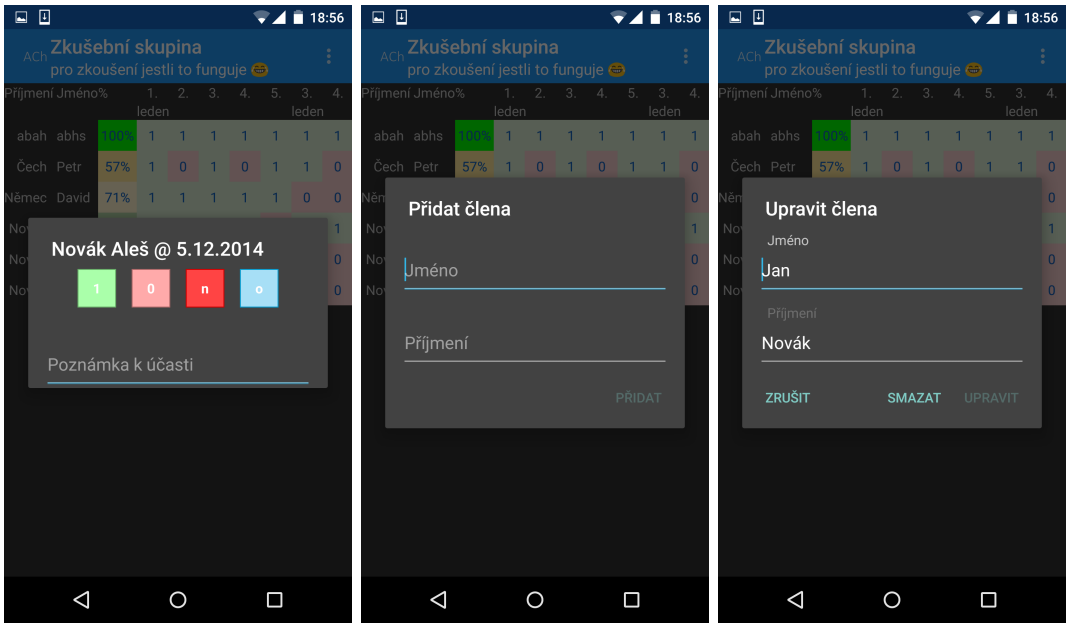
Použití webového rozhraní je totožné s mobilním klientem. Webové rozhraní má sloužit hlavně pro čtení obsahuje a tak je zobrazena pouze ukázka reálné tabulky (obrázek B.4) účastí (s rozmazanými jmény), která ilustruje přehlednost a čtivost dat.



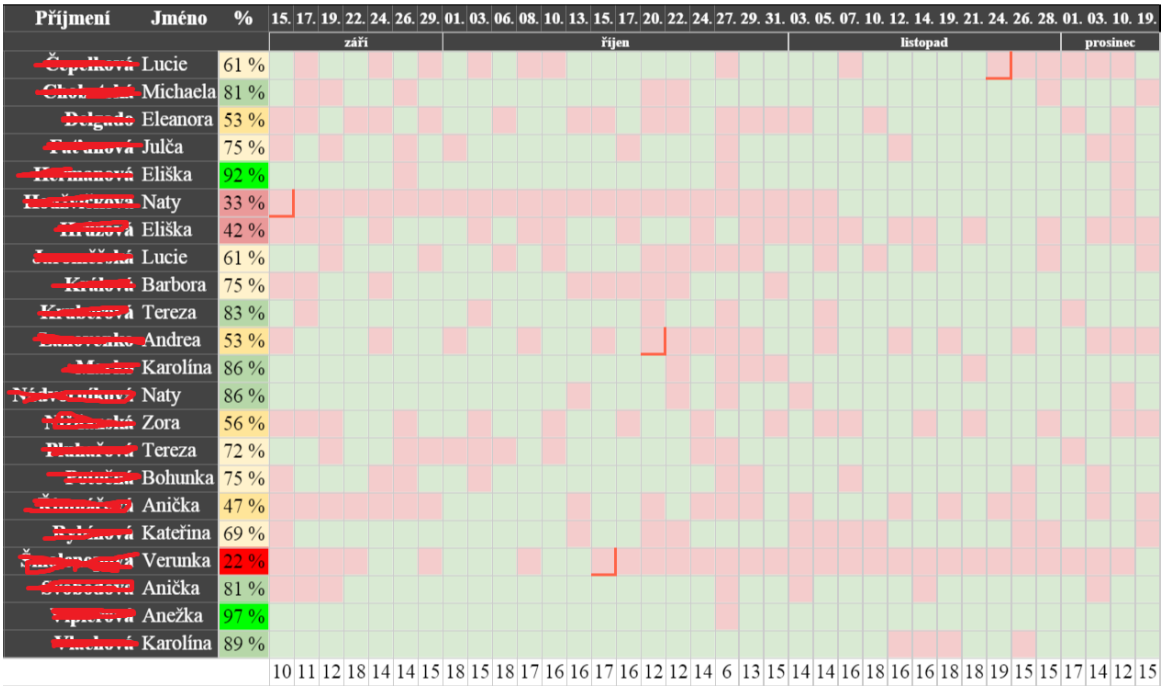
Obrázek B.1: Ukázky rozhraní aplikace - přihlášení, prázdná seznam skupin, přidání skupiny



Obrázek B.2: Ukázky rozhraní aplikace - seznam skupin, prázdná data skupiny, zobrazení dat



Obrázek B.3: Ukázky rozhraní aplikace - zadání účasti, přidání člena, úprava člena



Obrázek B.4: Ukázka webového rozhraní

B.3 Instalační příručka admina

Vlastní správa a úprava navrženého systému se dělí na části:

- **Nastavení účtu *Firebase*** - Na stránkách Firebase je potřeba založit si účet. Po přihlášení v nástrojích správce založit novou Firebase databázi a poznamenat si její URL.
- **Nastavení Android Studia a kompilace aplikace** - Mobilní aplikaci lze zkompilovat po úpravách i jen za pomocí *Android SDK Platform Tools*. Pohodlnějším způsobem je ale instalace Android Studia. Studio nainstaluje i aktuální *SDK* a se všemi kroky kompilace aplikace poradí. V SDK Manažeru je třeba doinstalovat balík *Android Support Library*. Pro zprovoznění aplikace je jí třeba podepsat, k čemuž poslouží tento postup. Pro instalaci aplikace se musí připojit koncové zařízení k počítači a v Android Studiu spustit aplikaci. Tato část bude zjednodušena nahráním aplikace na obchod GooglePlay.
- **Úprava webového rozhraní a nasazení na server** - webové rozhraní může být upravovováno jakýmkoliv dostupným nástrojem. Při změně šablon je třeba je poté zkompilovat skriptem *compile.bat* či *compile.sh*. Nahrání se spustí příkazem *firebase deploy*. Pro oba příkazy je třeba nejdříve nainstalovat platformu *node.js* a balíky *handlebars* a *firebaseconsoletools*. Při prvním spuštění nahrání aplikace je třeba se v příkazové řádce přihlásit pod údaji svého *Firebase* účtu.

Příloha C

Obsah přiloženého CD

Obsah přiloženého CD popisuje hlavní složky a soubory obsažené. Vzhledem k množství jednotlivých souborů jsou tyto popsány hromadně.

- introvic-thesis.pdf - diplomová práce ve formátu *pdf*
- tree.txt - tento obsah
- thesis/ - zdrojový kód textu diplomové práce ve formátu LaTex s přidruženými soubory.
- mobilni-klient/ - složka s mobilním klientem
  - /app
  - /app/build.gradle - hlavní sestavovací skript
  - /app/proguard-rules.pro
  - /app/src/androidTest/ - testy aplikace
  - /app/src/main/AndroidManifest.xml - definování aplikace
  - /app/src/main/java/ - zdrojové kódy aplikace
  - /app/src/main/res/ - zdroje použité v aplikaci
- webove-rozhrani/ - složka s webovým rozhraním
  - /webove-rozhrani/app.js - hlavní skript aplikace
  - /webove-rozhrani/compile.bat - skript pro zkompilování šablon
  - /webove-rozhrani/compiled.js - zkompilované aktuální šablony
  - /webove-rozhrani/dao.js - skript pro práci s DB
  - /webove-rozhrani/deploy.bat - skript pro nasazení na hostin
  - /webove-rozhrani/firebase.json
  - /webove-rozhrani/handlebars.runtime-v2.0.0.js - knihovna Handlebars
  - /webove-rozhrani/helpers.js - pomocné funkce šablon
  - /webove-rozhrani/index.html - vstupní *HTML* soubor
  - /webove-rozhrani/styly.css - CSS styly aplikace
  - /webove-rozhrani/templates/ - složka s šablonami