VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

APLIKACE PRO VYHLEDÁVÁNÍ PARTNERŮ DO POSILOVNY (ANDROID)

APPLICATION FOR SEARCHING FOR FITNESS BUDDIES (ANDROID)

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE FILIP KALOUS
AUTHOR

VEDOUCÍ PRÁCE prof. Ing. ADAM HEROUT, Ph.D.
SUPERVISOR

BRNO 2016

Abstrakt

Tato práce se zabývá tvorbou mobilní aplikace. Jejím účelem bude pomoci vyhledat partnery k cvičení nehledě na sportovní aktivitu, i když je cílena hlavně na posilování ve fitness centrech. V práci je rozebrána tvorba samotné aplikace pro operační systém Android a implementace serverové části tvořená pomocí frameworku Ruby on Rails. V praxi by aplikace měla umožnit uživatelům kdykoliv a kdekoliv najít vhodného partnera k cvičení.

Abstract

This thesis is concerned with the development of a application for smartphones. It's purpose will be to help find sports partners, no matter the sport activity, although it is aimed mainly on weight lifting activities in fitness centres. The development of the application for Android operation system and implementation of the server part made with the help of Ruby on Rails framework is analyzed in the thesis. In practice the application should enable its users to find a suitable sports partner anywhere and anytime.

Klíčová slova

Mobilní aplikace, Android, Ruby on Rails, API, MVC, Sportovní aktivity, Material Design

Keywords

Mobile application, Android, Ruby on Rails, API, MVC, Sports activities, Material Design

Citace

KALOUS, Filip. *Aplikace pro vyhledávání partnerů do posilovny (Android)*. Brno, 2016. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Herout Adam.

Aplikace pro vyhledávání partnerů do posilovny (Android)

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana prof. Ing. Adama Herouta, Ph.D. Návrh a implementaci serverové části jsem vypracoval společně s kolegou Dominikem Plškem, který měl stejné zadání bakalářské práce, ale pro mobilní platformu iOS. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

. . . . . .
Filip Kalous
16. května 2016

Poděkování

Děkuji prof. Ing. Adamu Heroutovi, Ph.D za odborné vedení a pomoc při tvorbě této práce. Rád bych poděkoval kolegovi Dominikovi Plškovi za výbornou spolupráci při tvorbě serverové části. Dále bych chtěl poděkovat všem, kteří mi pomáhali s testováním aplikace a za jejich zpětnou vazbu.

Obsah

Kapitola 1

Úvod

Mobilní aplikace se úspěšně rozšiřují v mnoha odvětvích. Existují velmi úspěšné služby, které pracují s geografickou polohou uživatele, například v oblasti ubytování nebo dopravy. Tato práce se zaměřuje, jak těchto informací využít ve fitness světě. Velké množství lidí má problém pustit se do cvičení, protože nemají s kým a představa cvičit sami se jim nezamlouvá. Toto platí pro jakoukoliv sportovní aktivitu. Přitom zájemců o fitness a posiloven je v okolí mnoho. A právě pomoct nalézt partnera k cvičení je to, co chce dokázat mobilní aplikace Gymber.

Chytrý telefon vlastní podle stránky statista.com 2 miliardy lidí. Z tohoto obrovského počtu funguje 80 % na operačním systému Android. To je důvod, proč aplikace, která byla zamýšlena původně jen pro platformu iOS, bude i na Androidu. Aby aplikace mohla naplnit svůj hlavní cíl, je potřeba získat mnoho uživatelů. Lidí, kteří chtějí cvičit. Lidí, kteří budou vytvářet nabídky k cvičení. Cílem práce je vytvořit aplikaci umožňující najít partnera k cvičení nezávisle na sportovní aktivitě. Gymber umožní sdílet nabídky uživatelů, reagovat na ně, ale i vyhledávat vhodná místa pro zvolenou aktivitu. Samostatnou skupinou jsou trenéři, kteří budou moci aplikaci využít jako nový zdroj poptávky.

Tato práce bude strukturou dodržovat zadání. Kapitola po úvodu projde technologie a postupy, které byly použity při vývoji aplikace. Představí samotný systém Android a jeho zvyklosti při tvorbě uživatelského rozhraní. Dále architekturu Model-View-Controller, která byla využita při tvorbě back-endu a komunikaci s aplikací. V další části bude rozbor návrhu aplikace a back-endu, spolu s řešením komunikace mezi těmito části pomocí API[1]. Součástí této kapitoly bude i studium cílové skupiny. Následující kapitola představí implementaci obou částí a před závěrem popíši testování aplikace a zhodnocení dosažených výsledků. Aby další text dával smysl, příští kapitola představí samotnou mobilní aplikaci Gymber.

[1] Application Programming Interface – rozhraní využívané k získávání informací ze serveru

Kapitola 2

Mobilní aplikace Gymber

Cílovou skupinu jsem nastínil již v úvodu, teď popíši, jak aplikace má fungovat a co za pozitiva má jejím uživatelům přinést. Měla by představovat službu, v jistém slova smyslu i sociální síť pro sportovní nadšence. Základní funkční požadavek aplikace je registrace většího počtu uživatelů. I z tohoto důvodu je aplikace vyvíjena pro dvě nejpoužívanější mobilní platformy – Android a iOS (vyvíjí kolega Dominik Plšek jako bakalářskou práci).

Hlavní podstatou je přinést přehled sportovních aktivit, z kterých si bude možné vybírat podle přání uživatele. Co nejrychleji a nejjednodušeji přijmout některou nabídku bez složité domluvy s dalším zúčastněným. Dalším cílem, který by chtěla služba dosáhnout, je snížit ceny profesionálních lekcí cvičení. Momentální situace většinou zavazuje trenéry k určitému fitness centru a zaniká tak tržní a finanční konkurence. Ceny lekcí jsou pro obyčejné lidi celkem vysoké. Tím, že v aplikaci můžeme tvořit jak neplacené, tak i placené nabídky, vznikne pro trenéry konkurenční prostředí. Ideální následkem je snížení cen trenérských lekcí a nárůst poptávky.

V důsledku možnosti vytvoření nabídky každým uživatelem bude aplikace obsahovat hodnoticí systém. Je totiž nežádoucí mít přehlcenou databázi nabídkami od nekvalitních trenérů se špatnou pověstí. Následkem toho by aplikace mohla přijít o uživatele.

Nápad na aplikace náleží mému kolegovi Dominiku Plškovi, který vyvíjí stejnou aplikaci pro platformu iOS od společnosti Apple. Obě výsledné aplikace budou pracovat nad stejnou serverovou částí. Komunikace aplikací se serverem bude probíhat pomocí námi standardizovaného API popsané službou Apiary [4].

Absence jakékoliv podobné aplikace ve známých on-line obchodech (př. Google Play, Apple Store) je velkou výhodou pro budoucnost našeho vývoje. Na druhou stranu zatím neexistuje žádná zpětná vazba uživatelů na tento typ aplikací. Jedině oni odkryjí možné chyby vzniklé při vytváření programu Gymber.

Kapitola 3

Získávání a studium informací

Tato kapitola popisuje technologie a přístupy, které jsou použity pro tvorbu aplikace Gymber. Samozřejmostí je využití Android SDK[1]. K implementaci serverové části byl využit framework Ruby on Rails. Informace jsem převážně získával z oficiálních zdrojů na webu Android Developers [3] a Ruby on Rails Guides [6]. Pokusím se vysvětlit, z jakých možností daných technologií jsem si mohl vybírat. V první řadě popíši jmenované technologie a pojmy s nimi související.

3.1 Operační systém Android a jeho zvyklosti při tvorbě uživatelského rozhraní

Společnost Android Inc. byla založena v Kalifornii v říjnu 2003 Andym Rubinem, Richem Minerem, Nickem Searsem a Chrisem Whitem. V roce 2005 jí odkoupil Google. Prvním úkolem po odkoupení bylo vytvořit mobilní platformu založenou na linuxovém jádře. Celý systém je otevřený software (open source), který vyvíjí konsorcium Open Handset Alliance tvořený výrobci mobilních telefonů a společností Google. V roce 2008 v USA byl představen první telefon s tímto systémem, T-Mobile G1 (HTC Dream) a uvolněno SDK 1.0. Nyní, jak již bylo zmíněno, je Android nejpopulárnější mobilní platforma na světě s 80% podílem na trhu.

3.1.1 Základní prvky Androidu

V této podsekci popíši základní stavební kameny systému Android.

Zdroje Přeloženo z anglického *Resources*. Pomocí zdrojů se definuje statický kontext, který se následně využívá v kódu, hlavně pří tvorbě uživatelského rozhraní. Patří sem řetězce, barvy, animace nebo i návrhy vlastních grafických prvků.

Layout Grafický návrh je v Androidu popsán pomocí značkovacího jazyka XML. Každá obrazovka může obsahovat jenom jediný prvek. Aby výsledné návrhy mohly být složitější než jen jednoprvkové, využívá se kontejnerů. Tyto komponenty Android nazývá jako *layout*. Dva nejzákladnější jsou – *LinearLayout, RelativeLayout*.

Relativní kontejner se využívá pří komplexnějších rozloženích obrazovky, protože ke složení výsledného rozložení není potřeba takového výkonu jako v případě lineárního. Z dalších

[1] Software Development Kit

kontejneru zmíním například GridLayout, v kterém se prvky vkládají do mřížky definované velikosti. Také je možné použít seznamy *ListView, GridView*, které následně využívají adaptérů k získání položek seznamu. Pro zobrazení dokumentu napsaném v HTML se používá *WebView*. Vytvořená rozložení se následně připojují k aktivitám nebo fragmentům.

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

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/label_example_btn"
        android:layout_centerHorizontal="true"
        android:layout_centerVertical="true"
        android:text="Example button"/>

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/example_button"
        android:layout_below="@id/label_example_btn"
        android:text="@string/login"
        android:textAllCaps="false"/>

</RelativeLayout>
```

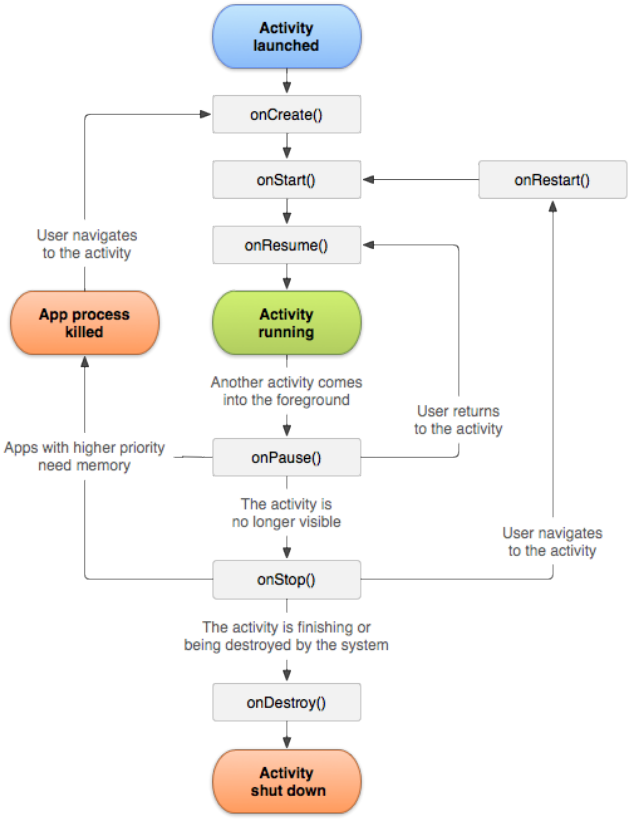
Listing 3.1: Kód ukazuje jednoduché rozložení obsahující 3 prvky. Jedním z prvků je kontejner *RelativeLayout*, který zastřešuje elementy *TextView* a *Button*. Kontejner se stará o uspořádání prvků. Každému prvku je možné přiřadit atribut id, pomocí něhož se na prvek odkazuje.

V kódu 3.1 je vidět jak vypadá jednoduchý layout. Aby se prvky správně poskládaly, využívá se parametru *android:id*, pomocí něhož se na prvky odkazuje. Tohoto odkazu se následně využívá při implementaci funkčnosti, kdy díky *id* získáme referenci na požadovaný prvek. K odchytávání událostí z akčních prvků, jako jsou tlačítka nebo položky seznamů, jsou připravena rozhraní nazývaná *Listeners*.

Aktivity Aktivita je komponenta zobrazující v okně uživatelské rozhraní, se kterým může uživatel interagovat. Většinou aktivita vyplňuje celou obrazovku. Aktivita se dá představit jako kontrolér v modelu MVC[2]. Typické je, že aplikace jako první spouští tzv. hlavní (main) aktivitu. Každá aktivita může spustit další a spolu s tím jí předat i data. V tomto případě se data předávají v podobě záměrů. Při startu jiné aktivity je ta aktuální uložena systémem na zásobník. Při stisknutí tlačítka zpět nebo ukončení aktivity, se poslední uložená nahraje opět do popředí. Na obrázku 3.1 je zobrazen životní cyklus aktivity.[3]

Fragmenty V Androidu byly fragmenty představeny v API 11 (Android 3.0). Přidávají další vrstvu mezi aktivitu a rozhraní. Hlavním důvodem vzniku fragmentů byla rostoucí poptávka po tabletech. Odpovědí společnosti Google na ní byl právě vznik fragmentů.

[2]Model-View-Controller model
[3]Zdroj http://developer.android.com/guide/components/activities.html



Obrázek 3.1: Obrázek popisuje životní cyklus aktivity. Celý cyklus začíná spuštěním aktivity a zavoláním metody *onCreate()*, kterou je povinné implementovat. Po vykonaní metody *onResume* již aktivita běží a typicky s ní uživatel může interagovat. Při spuštění jiné aktivity je volána metoda *onPause()*, při navrácení k původní aktivitě *onResume()*. Aktivita není ukončena do té doby, než se spustí metoda *onDestroy()*.

Fragmenty se vkládají do aktivit, a ty jich můžou obsahovat libovolné množství. Například při výběru položky ze seznamu můžeme na tabletu zobrazit její detail hned vedle, a nemusí se spouštět speciální aktivita. Od té doby jsou fragmenty využívané v mnohem větší míře. V této aplikaci například k funkci navigačního menu. Fragmenty se specifikují buď již v návrhu rozložení nebo programově. Architektura je velmi podobná aktivitám, avšak jsou vázány k své rodičovské aktivitě a nepodporují takovou funkcionalitu.

AndroidManifest *AndroidManifest* je soubor ve formátu XML a má ho každá aplikace fungující pod platformou Android. Tento soubor poskytuje systému informace nutné pro spuštění aplikace a bez něj by aplikace nebyla spustitelná. Jsou v něm vypsané všechny aktivity, které aplikace obsahuje. Také definuje povolení, která chce aplikace od uživatele získat. Například přístup k internetu, geografické poloze atd. Důležitým parametrem souboru, a také povinným, je element *application*, který definuje aktivitu spouštějící se jako první, popisek a ikonu aplikace atd. Druhým povinným elementem je *manifest*. V něm je definován balíček, který obsahuje aplikaci a již popisovaná povolení.

Název	Verze	API	Rok
Froyo	2.2	8	2010
Gingerbread	2.3	10	2010
Honeycomb	3.x	11 - 13	2011
Ice Cream Sandwich	4.0	15	2011
Jelly Bean	4.1.x	16	2012
	4.2.x	17	2012
	4.3	18	2012
KitKat	4.4 - 4.4.x	19 - 20	2013
Lollipop	5.0 - 5.1	21 - 22	2014
Marshmallow	6.0.x	23	2015
N	Developer Preview 2		2016

Tabulka 3.1: Historie verzí systému Android. Z tabulky je vidět, že Google se snaží každý rok představit novou verzi systému Android. V době psaní práce byla nejaktuálnější verze *Marshmallow*, přičemž v červenci 2016 Google uvede novou verzi s názvem začínajícím na písmeno N. V tabulce neuvádím verze, které již nejsou používané.

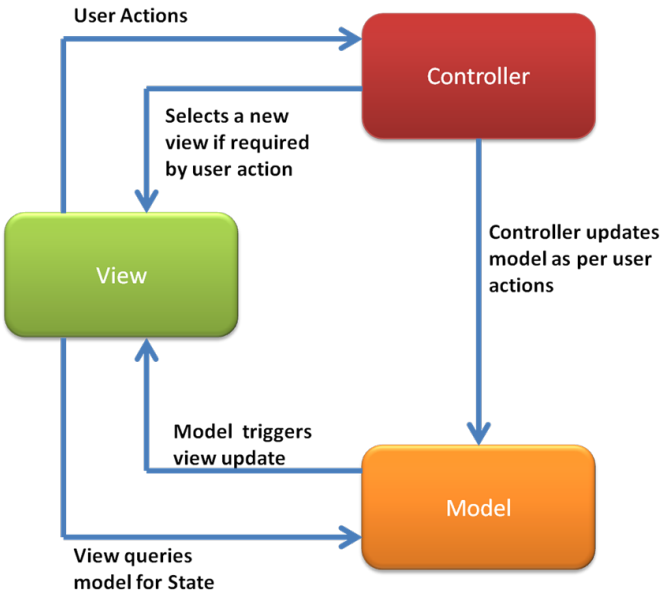
3.1.2 Zvyky při tvorbě uživatelského rozhraní

Zde bych rád popsal některé dobré zvyky při tvorbě uživatelského rozhraní pro systém Android, které vycházejí především od Googlu a také mých zkušeností při tvorbě této práce.

Je velmi dobré si určit pro jaká zařízení chci aplikaci tvořit. Zda jen pro telefony nebo i tablety, a také jestli bude možné změnit orientaci obrazu. Dále bych uvedl používání *Action Baru*, což je lišta zobrazující se na samém vrchu aplikace. Pomocí ní může mít uživatel přístupné důležité akce. Také se pomocí ní dostane k navigačnímu menu. Určitě je dobré uživateli zobrazovat důležité informace pomocí vyskakovacích zpráv (*Toast nebo SnackBar*), například při smazání nějaké položky v seznamu. Díky tomu bude uživatel vědět, zda se operace úspěšně provedla či nikoli. Jeden z dalších dobrých zvyků spočívá v přepisování výchozích konfigurací komponent. Například vezmu-li obyčejné tlačítko (třída *Button*), podědím jeho konfiguraci a přepíši jeho metody nebo doplním vlastní. Také mohu upravit výchozí vzhled tlačítka. Naposled bych uvedl používání Material Designu, který představil Google v roce 2014. Spousta aplikací byla podle tohoto designu předělána a uživatelé si na něj ve velkém zvykli. Pokud není nutné se držet nějakého předepsaného designu od zadavatele apod., je velmi dobré zvážit použití Materialu. O samotném Material Designu se více rozepíší v následující kapitole, která bude popisovat návrh aplikace 4.2.1.

3.1.3 Verze systému a SDK

Aktuálně telefony postavené na systému Android fungují až na osmi verzích systému (tabulka 3.1 a obrázek 4.1. Tato fragmentace nesvědčí ani vývojářům, ani uživatelům. Při vývoji se nastavuje verze, kterou aplikace bude minimálně podporovat. Tento fakt znesnadňuje vývoj moderních aplikací. Výsledný produkt z grafického hlediska může vypadat jinak a vývojář musí testovat aplikaci na velkém množství telefonů. Řešení konfliktů spočívá ve



Obrázek 3.2: Model-View-Controller model. Obrázek zobrazuje propojení mezi jednotlivými částmi modelu a akce nad nimi prováděné. View má k dispozici odkaz na Model, aby mohl zobrazovat jeho data. Controller obsahuje odkaz na Model, aby mohl upravovat jeho data. Uživatel své akce provádí nad View.

zvýšení podporované verze, avšak tímto krokem přichází o uživatele. V jiném případě je možné rozvětvit kód podle verze systému, na kterém je aplikace spouštěna. Ani jedno z těchto řešení není ideální a přináší problémy při návrhu a funkcionalitě aplikace pro různé verze.

Android SDK je softwarový vývojářský balíček poskytující vývojáři verzi systému (API), nástroje pro emulaci mobilního zařízení, dokumentaci příslušné verze a další pomocné nástroje. Mezi velmi důležité součásti patří *Android Support Library*. Tato knihovna umožní používat novější prvky i v starších verzích. Například fragmenty (verze API 11) je možné využít až k API 4.

3.2 Model-View-Controller architektura a Ruby on Rails

Model-View-Controller Tento model byl poprvé využit v jazyce Smalltalk a poprvé ho popsal Trygve Reenskaug v roce 1979. Model-View-Controller (dále jako MVC) je architektura, která je rozdělena, jak již zní z názvu, na tři části. Datový model s business logikou (Model), uživatelské rozhraní (View) a aplikační logiku (Controller). Propojení mezi jednotlivými částmi je znázorněno na obrázku 3.2.

Model v aplikaci slouží jako datová struktura. Model ale neobsahuje jen data. Jeho další součástí je kontrola nebo validace dat (e-mail musí být vložen v přesně definovaném formátu), dále také business logika (metoda vracející věk z data narození). Model také vůbec netuší odkud mu data chodí, ani jak se zobrazují.

View zobrazuje data na výstup podle šablony. Představuje kód, který generuje uživatelské rozhraní. Data přijímá od Controlleru. V případě webu je to ta část, která generuje HTML. Generovat však může například i JSON nebo XML.

Controller je nejhůře představitelná část. Stará se o přijímaní požadavků z uživatel-

ského rozhraní (např. kliknutí na tlačítko), vykonání potřebných změn na Modelu nebo změny přímo View. Následně vybere potřebnou šablonu (View), podle které se nový obsah vygeneruje uživateli.

Tok událostí:

- Uživatel provede nějakou akci v uživatelském rozhraní.
- Tento požadavek zachytí router, který rozhodne jaký Controller a jakou akci chceme použít.
- Controller rozhodne, co je potřeba provést a po provedení změn pošle data na View.
- View vloží data do šablony a zobrazí ji uživatelovi.

Ruby on Rails Ruby on Rails (dále jako Rails) je framework sloužící k vývoji webových aplikací psaný v jazyce Ruby. Tento framework představil poprvé v roce 2005 David Heinemeier Hansson jako open-source. Momentálně do něj přispívá tisíce vývojářů. Rails jsou založeny na již popsaném modelu MVC. Rails se snaží vést vývojáře podle několika základních principů:

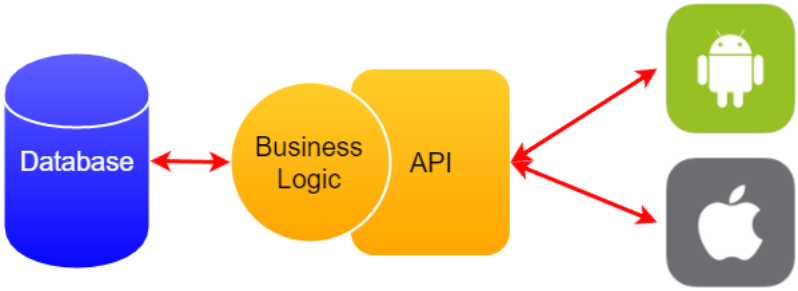
- Neopakujte se – dobrý kód je znovupoužitelný a nemusí se psát znova.
- Konvence má přednost před konfigurací – Rails předpokládají, co a jak chci udělat a nemusím specifikovat každou drobnost.
- REST je nejlepší architektura pro webové aplikace (popis zde 4.3.2).

Rails jsou framework, který je poskládán ze spousty malých částí. Pro příklad uvedu alespoň ty, které představují jednotlivé části z MVC modelu.

- Active Record – základní prvek pro modely, dokáže provádět základní i pokročilé operace nad databází, a také definovat relační vztahy mezi modely.
- Active View – má na starosti pohledy v aplikaci. Řídí renderování šablon, ve výchozím nastavení v HTML či XML.
- Active Controller – Zpracovává příchozí požadavky, získá z nich parametry a provede příslušné akce.
- dále pak například – Action Mailer (podpora práce s e-maily), Active Support (soubor podpůrných funkcí.

K Rails je možné přidávat spoustu balíčků, například od jiných vývojářů. Tyto balíčky se v Rails aplikaci jmenují gemy a přidávají se pomocí souboru Gemfile. Při spuštění aplikace je zkontrolováno, zda jsou všechny balíčky nainstalované a dostupné. V opačném případě se doinstalují pomocí nástroje *Bundler*.

V Rails aplikaci se od začátku počítá s využitím databáze. Ve výchozím nastavení se používá SQLite3. Aplikaci je možné spustit ve třech možných prostředích, a to Development, Test a Production. Přičemž pro každé prostředí je možné nastavit jinou konfiguraci databáze i použitých gemů. Databáze se upravuje pomocí migrací. Pro každou změnu na modelech je potřeba vytvořit migraci, která po spuštění příkazu *rake db:migrate* aktualizuje strukturu databáze. Práci s databází obstarávají modely s velkou mírou abstrakce. Není tedy potřeba psát dotazy přímo v SQL, ale využívat vestavěných metod.



Obrázek 3.3: Na obrázku je představen princip API. Pomocí tohoto rozhraní je možné komunikovat se serverem a jeho databází nezávislé na platformě (v případě Gymberu se systémy Android a iOS).

Velkou přidanou hodnotou v Rails je používání generátoru. Tímto způsobem je možné vygenerovat hlavní součásti aplikace podle šablon, které Rails obsahují. Je možné generovat samotné části, jako třeba Model (příklad 3.2) nebo je možné využít *scaffolding* („lešení“). V tomto případě Railsy vytvoří celou strukturu. Model, pohledy, kontrolér, vytvoří migraci do databáze, připraví testovací soubory a upraví soubor sloužící k směrování požadavků URL.

```
$ rails generate model User username:string e-mail:string
```

Listing 3.2: Ukázka generování modelu *User* obsahující textové atributy *username* a *e-mail*.

3.3 Application Programming Interface

Z důvodu, že s kolegou tvoříme stejnou aplikaci, ale pro jiné platformy, je nutné mít nějaké rozhraní, pomocí něhož budeme posílat a získávat data z/do databáze. K tomuto účelu jsme využili Application Programming Interface (dále jen API), které bych v této sekci krátce shrnul.

V našem případě se jedná o webové API na straně serveru. Princip API je znázorněn na obrázku 3.3. Webové API pracuje s koncovými body (endpoints). K nim se přistupuje pomocí HTTP[4] požadavku (POST, GET, DELETE, atd.), který obsahuje URL[5] stránky, o kterou má klient zájem. Díky aplikačnímu rozhraní je možné ke stejným datům přistupovat z neomezeně platforem, a je zajištěno, že dostanou všichni data ve stejném formátu.

Důležitou složkou je mít hotové API velmi dobře zdokumentováno. Z této dokumentace pak vývojář čerpá k zjištění informací, které akce má provádět a v jakém formátu má očekávat data. V případě Gymberu je dokumentace popsána pomocí služby Apiary [4], která je založena na open-source popisovacím jazyku API Blueprint (ukázka v kódu 3.3).

JavaScript Object Notation JavaScript Object Notation (JavaScriptový objektový zápis, dále jen JSON) je datový formát určený k přenosu dat. JSON může obsahovat jak pole, tak celé objekty. Data mohou být ve formátu řetězce, čísla nebo nabývat hodnot *true*, *false* a *null*. V kódu 3.3 je vidět jak vypadá zpráva, která má tělo tvořené formátem JSON. Zpráva obsahuje objekt, který obklopuje pole *users*. V tomto poli jsou pak jednotlivé objekty obsahující parametry představující informace o uživateli.

[4]Hypertext Transport Protocol
[5]Uniform Resource Locator

```
### List All Users [GET]

+ Response 200 (application/json)
  {
    "users": [
      {
        "id": 1,
        "first_name": "Petr",
        "last_name": "Bolek",
        "image": {
          "image": {
            "url": null
          }
        },
        "is_trainer": false
      }
    ]
  }
```

Listing 3.3: Ukázka jazyku Blueprint a datového formátu JSON. Jazyk s otevřeným kódem Blueprint využívá služba Apiary pro popis API. V této ukázce je definované chování akce *List All Users* odpovídající HTTP metodě GET. Odpověď na tuto akci má návratový kód 200 a tělo ve formátu JSON, které obsahuje pole objektů *users*.

Kapitola 4

Návrh mobilní aplikace a serverové části

Tato kapitola rozebírá návrh řešení mobilní aplikace, serverové části a API. Nezabývá se ještě implementací částí, ale jak jednotlivé části budou fungovat a kooperovat spolu. Návrh aplikace jsme částečně řešili spolu s kolegou Dominikem Plškem. Ve výsledku by měly obě aplikace mít stejnou funkcionalitu. I grafika aplikací bude stejná, s tím rozdílem, že aplikace budou dodržovat zvyky a pokyny vlastní platformy.

Výsledná aplikace by měla být jednoduchá, neměla by zatěžovat uživatele zbytečnými informacemi ani se je snažit bezdůvodně získat. V dnešních aplikacích je velice časté hned s prvním spuštěním požadovat po uživateli funkčně nepotřebné informace. Touto cestou Gymber nepůjde.

4.1 Cílová skupina uživatelů

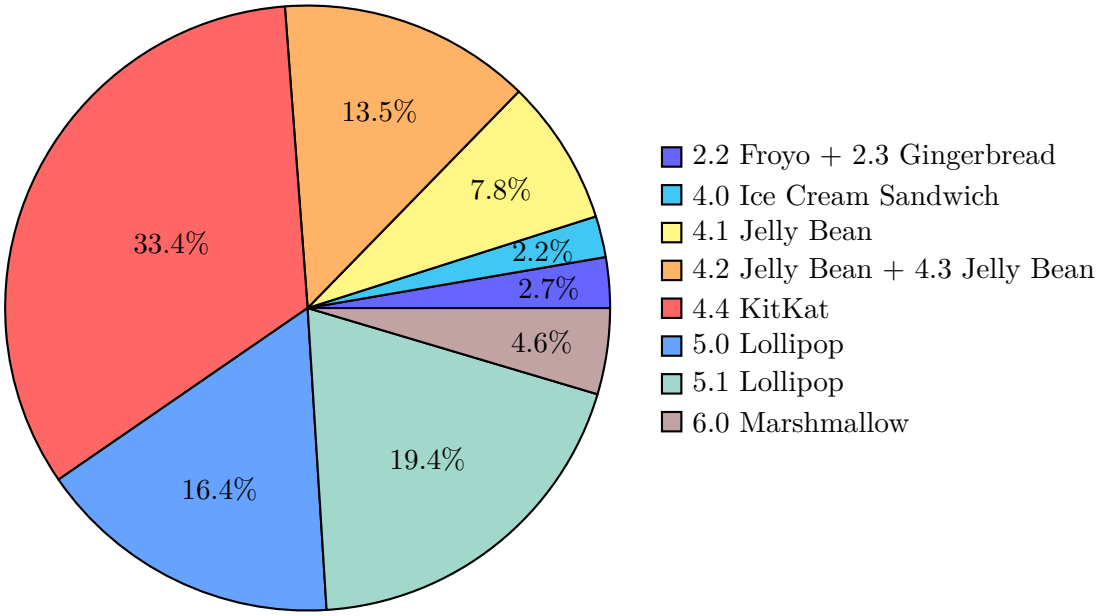
Než se dostanu k návrhu aplikace, rád bych ještě upřesnil cílovou skupinu aplikace Gymber. Pro koho by aplikace měla být, bylo lehce nastíněno v úvodu. Nyní bych rád rozdělil cílové uživatele na skupiny:

1. Ti, kteří nechtějí cvičit sami
2. Ti, kteří nevědí jak cvičit
3. Amatérští trenéři
4. Profesionální trenéři
5. Ti, kteří chtějí získat informace o fitcentrech, uživatelích

Dá se předpokládat, že mezi první dvě skupiny bude patřit majoritní podíl uživatelů. Poslední skupina byla zahrnuta spíše pro úplnost. Není úmysl cílit aplikaci na tuto část uživatelů. Trenérům by aplikace měla přinést větší poptávku po placených lekcích.

4.2 Layout aplikace s Material Design

Ještě předtím, než bylo možné začít s návrhem aplikace, bylo nutné si ujasnit pro jakou minimální verzi API Androidu bude tvořena. Jak dokazuje obrázek 4.1, nevyplatí se již



Obrázek 4.1: Procentuální rozložení aktuálně používaných verzí Android API

navrhovat aplikaci pro API menší než 2.2. Nižší verze už nikdo nepoužívá. Při tvorbě moderní aplikace by to bylo značně omezující. V starších verzích chybí spousta grafických prvků a bylo by nutné větvit zobrazování a práci s aplikací podle verze systému. Já osobně jsem se rozhodl podporovat minimální verzi Android 4.1 Jelly Bean (verze API 16). Podpora od této verze pokryje 95% uživatelů a omezení oproti novějším verzím je minimální.

Jedno ze specifik telefonů s Androidem je, že každý druhý telefon má jiné parametry obrazovky. Tím myslím jak velikost obrazovky, tak hlavně rozlišení plus zobrazovací technologii displeje. U aplikace, která se bude snažit dostat na hodně zařízení, je to problém. Na každém zařízení by měla v ideálním případě vypadat úplně stejně. S tím bylo nutné k návrhu přistoupit.

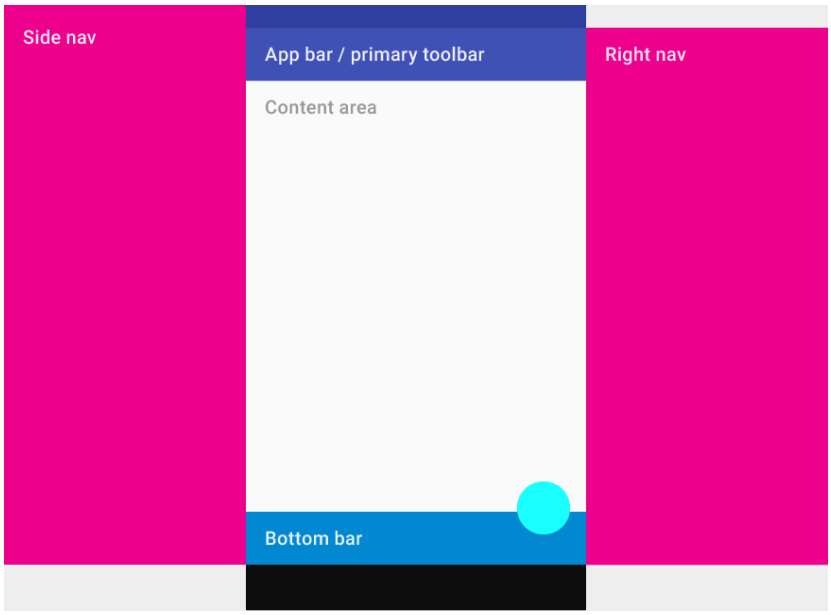
4.2.1 Material Design

Při tvorbě designu aplikace jsem se rozhodl použít prvky Material Designu od společnosti Google. Material Design představil Google až ve verzi systému Android 5.0 Lollipop. Ale díky pomocným knihovnám k Androidu a knihovnám od jiných vývojářů můžu používat tyto prvky i v mnou zvolené nejnižší verzi systému.

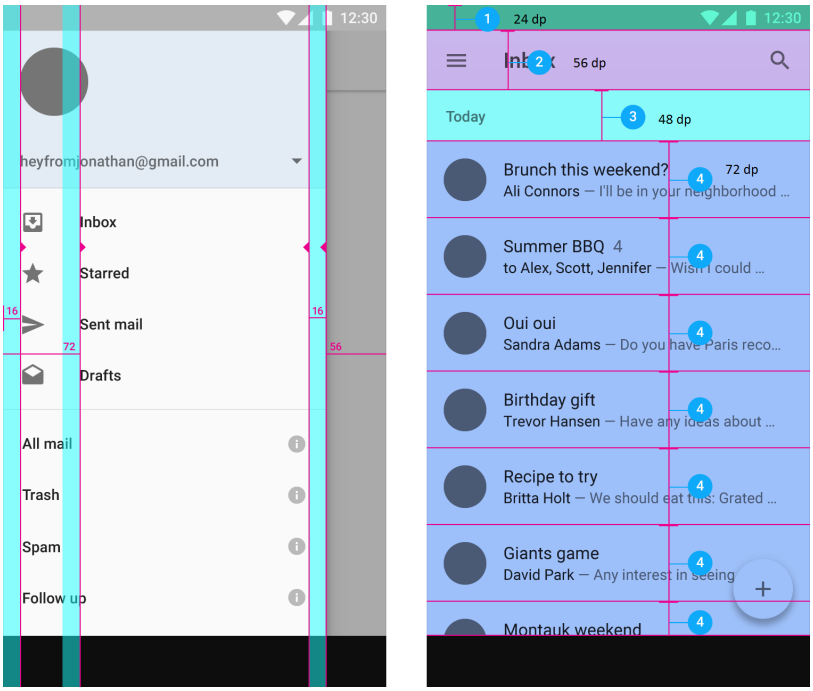
Material Design je vizuální jazyk, který se snaží skloubit klasické principy s inovací a novými technologickými možnostmi. Snaží se představit jednotný styl napříč platformami a velikostí zařízení. Material Design představuje design, který pracuje se všemi 3 osami. Využívá jednoduché geometrické tvary a jejich stíny, které se mění na základě uživatelovi interakce. Všechny prvky musejí mít tloušťku 1dp[1]. Interakce uživatele musí ovlivnit jen nejvrchnější *material*. Všechny předpisy jsou k naleznutí zde[5]. Tento design popisuje, jak by měl projekt vypadat jako celek, dále také vzhled samostatných prvků (tlačítka, seznamy, toolbary, menu atd.), ale i barvy, fonty, ikony, texty v aplikaci.

Na obrázku 4.2 je vidět základní struktura rozložení prvků pro Material Design. Obrázek 4.3 popisuje, jak podle Material Designu má vypadat seznam položek a navigační menu.

[1] density-independent pixel – tento pixel má vždy stejnou velikost nezávisle na hustotě pixelů na obrazovce



Obrázek 4.2: Na obrázku je možné vidět, jak Material Design popisuje strukturu zobrazení na mobilních telefonech. Největší sekci viditelné části zabírá prostor, kde je zobrazen obsah (*Content Area*). K horní straně je připnut aplikační panel, po levé a pravé se zobrazují navigační menu (na pravé straně být nemusí). Tyto menu vždy po vysunutí překrývají obsah ve viditelné části. Spodní panel je volitelný.



Obrázek 4.3: Levý obrázek: Ukazuje definici odsazení prvků v bočním navigačním menu. Celé menu musí mít pravé i levé odsazení 16dp. Pokud jsou k položkám přiřazeny ikony, položky jsou přesně 72dp od levého okraje. **Pravý obrázek:** Na tomto obrázku je vidět struktura aplikace se seznamem a velikost jednotlivých prvků. Všechny položky mají přesně dané velikosti. Aplikační panel (toolbar) má definovanou výšku 56dp a položky seznamu 72dp. Rozcestník označený číslem 3 není povinný. Nejvrchnější panel (číslo 1) s velikostí 24dp obstarává systém.

4.2.2 Hlavní obrazovky aplikace

Jak jsem se zmínil, aplikace bude navrhovaná podle předpisů Material Designu. Také se ale musí držet společného designu s aplikací pro iOS. Jako hlavní barvu Gymberu jsme s kolegou vybrali světle oranžovou. Podle nás tato barva ideálně sedí ke sportovnímu odvětví. Proto barva bude představovat i akční barvu. Bude tedy spojená s akčními prvky v aplikaci, jako jsou tlačítka, odkazy, aj. Doplňující barvy budou tmavě červená a bílá.

Níže v této sekci popíši rozložení základních obrazovek aplikace a jejich vzájemné propojení.

Splash Activity – úvod aplikace Tato obrazovka bude sloužit k načtení informací před zapnutím aplikace. Po celou dobu běhu této aktivity bude na ploše svítit logo aplikace. Během toho se zjistí, zda je uživatelovo přihlášení validní či nikoliv. V kladném případě bude uživatel rovnou přesměrován na hlavní obrazovku, v opačném případě na přihlašovací obrazovku. Dále se zkontroluje připojení k internetu a inicializuje se API třetích stran (prozatím jen Facebook).

Main Activity – hlavní obrazovka Po úspěšném přihlášení bude uživatel poslán na nejdůležitější aktivitu aplikace. Zde bude zobrazen seznam všech nabídek uživatelů (feed). Tato výchozí obrazovka by měla fungovat jako rozcestník pro zbytek aplikace. Pomocí tlačítka (Floating Action Button, dále jen FAB[2]) aplikace spustí aktivitu pro vytvoření nové nabídky. Samozřejmostí je kliknutí na položku feedu a zobrazení si jejího detailu.

Hlavní navigační prvek bude v aplikaci představovat navigační menu, dostupné z akčního menu (toolbar) umístěného v horní části obrazovky nebo pomocí jeho vysunutí z levé části (vychází ze struktury na obrázku 4.2). Toto menu je generováno jako součást *Main Activity* a vzniklé události v něm bude obstarávat ona. *Main Activity* i navigační menu je možné vidět na obrázku 4.4.

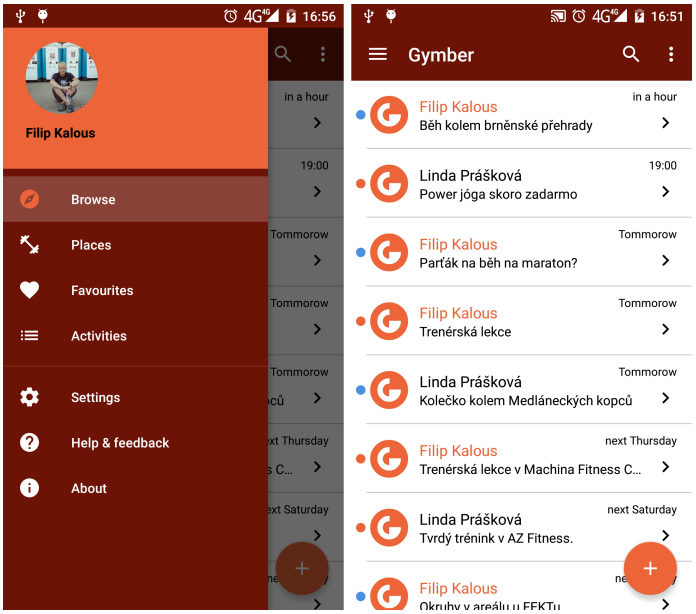
Login Activity – přihlášení Pokud ještě uživatel není přihlášen v aplikaci, neodkladně bude přesměrován ze *Splash Activity* na přihlašovací obrazovku. Uživatel dostane na výběr ze dvou možností – přihlášení se přes svůj Facebookový účet nebo projít přes registrační formulář. Poté již získá přístup do aplikace.

Create Activity – vytváření nabídky Jednoduchá obrazovka obsahující formulář pro vytvoření nabídky. Po odeslání nabídky proběhne kontrola, zda jsou vložena všechna potřebná data. Pokud nejsou, aktivita na to upozorní a nechá uživatele opravit vložené informace. Po odeslání se správnými daty je uživatel informován o úspěšném či neúspěšném uložení nabídky na straně serveru. Tímto krokem se následně aktualizuje seznam nabídek.

Aktivita sloužící k vytváření nové nabídky je ukázána a popsána na obrázku 4.5.

Detail Activity – detail nabídky Na tuto obrazovku bude možné se dostat dvěma způsoby. První je z hlavní obrazovky. V detailu se zobrazí všechny dostupné informace o nabídce. Z detailu bude možné dostat se na profil uživatele, který nabídku vytvořil. Pokud bude nabídka spojena s lokací uloženou v databází, bude možné si zobrazit detail této lokace. Ve spodní části se zobrazí tlačítko pro přijmutí dané nabídky.

[2] Více zde – https://www.google.com/design/spec/components/buttons-floating-action-button.html



Obrázek 4.4: Ukázka navigačního menu a hlavní obrazovky Gymberu zobrazující seznam nabídek

Druhý způsob je z přehledu nabídek vytvořených uživatelem. Opět se zobrazí dostupné informace o nabídce. Jediný rozdíl bude ve funkci tlačítka. Uživatel bude schopný nabídku smazat, pokud bude stále aktuální.

Location Activity K této aktivitě se uživatel dostane dvěma způsoby. Buď z detailu nabídky nebo ze seznamu lokací v aplikaci. V aktivitě se zobrazí informace o lokaci, jako otevírací doba, adresa, webové stránky. Dále bude na obrazovce FAB tlačítko, které aktivuje aplikaci umožňující telefonát na dané místo. Na velké části obrazovky se zobrazí úvodní fotka místa.

Profile Activity *Profile Activity* bude v aplikaci sloužit k zobrazení informací o vlastním profilu, jednoduchých statistik a samozřejmě editace osobních informací. K této aktivitě bude přístup jen pomocí navigačního menu, a to po klepnutí na plochu s fotkou v horní části obrazovky.

Person Activity *Person Activity* bude zobrazovat profily ostatních uživatelů. V ní uživatel uvidí sdílené informace, počty sledujících a sledovaných, vytvořených a přijatých aktivit. Navíc bude možné zobrazit uživatelovy poslední aktivity a také zařadit jeho osobu mezi mnou sledované uživatele.

4.2.3 Registrace uživatele

Aby vůbec bylo možné služby aplikace využívat, je potřeba zaregistrovat uživatele do systému. Nabízelo se více způsobů registrace, které jsme mohli uživateli nabídnout. Vybrány byly tyto dvě:

- Registrace pomocí e-mailové adresy
- Přihlášení pomocí API třetích stran (nyní jen Facebook)

Pokud se uživatel rozhodne pro registraci pomocí e-mailové adresy, bude nutné nejdříve zkontrolovat, zda je e-mail volný. Jestliže je e-mail volný, uživatel zadá základní osobní informace a je založen účet. Pokud již je účet obsazen, je na něj zaslán e-mail s aktivačním kódem a přesměrován na poslední obrazovku registrace. Jednotlivé kroky registrace uživatele jsou znázorněny a popsány na obrázku 4.6.

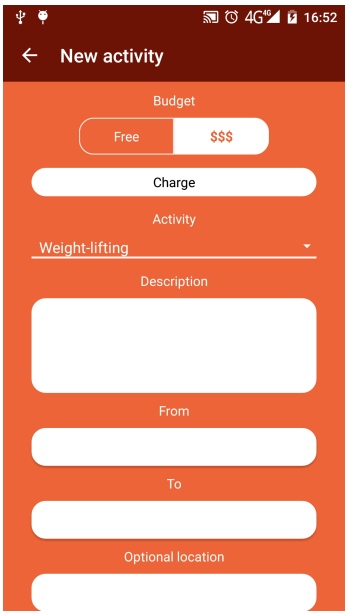
Pro druhý popis přihlášení pomocí API třetí strany popíši příklad s API Facebooku. Při rozhodnutí uživatele pro tento způsob se uživateli zobrazí dialog pro přihlášení a autorizování aplikace Gymber přes Facebook. Po úspěšné autorizaci aplikace získá základní údaje o uživateli, které použije k vytvoření vlastního záznamu uživatele v databázi.

Hlavní rozdíl mezi těmito dvěma způsoby přihlášení představují získané informace. Při registraci přes Facebook není možné, bez dalších povolení, získat e-mailovou adresu uživatele. Aby ale i nadále bylo možné kontrolovat unikátnost uživatele v databázi, je nutné získat identifikační číslo uživatele z Facebooku. Toto číslo pak může nahradit e-mail při kontrole unikátnosti.

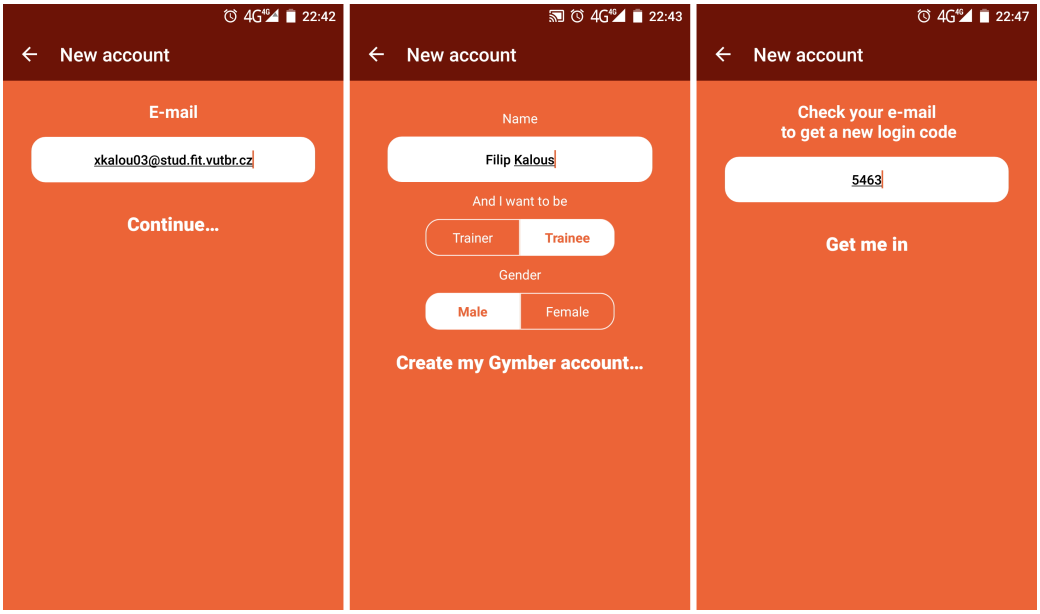
4.2.4 Seznam nabídek (feed)

Nejdůležitější část služby Gymber. Seznam tvořený jednotlivými nabídkami sportovních aktivit od uživatelů. Možností, jak ho navrhnout, se nabízela spousta, např. položky zabírající až třetinu obrazovky zobrazující všechny informace nebo položky bez fotky nebo minimalistické řádky, kterých se vejde na obrazovku i deset.

Jak je vidět z obrázku 4.7, vybrán byl design minimalistický. Jedním z důvodů bylo i doporučení Material Design pro tvorbu seznamů. To neznamená, že i když je položka malá, tak nezobrazí všechny důležité informace.



Obrázek 4.5: Na obrázku je možné vidět aktivitu sloužící k vytvoření nabídky. Obrazovka obsahuje přepínač pro výběr placené/neplacené nabídky. Závisle na tomto přepínači se objevuje/skrývá vstup pro cenu nabídky. Dalším prvkem je rozbalovací menu, v němž uživatel vybírá sportovní aktivitu. Dále již následuje popis, časový rozsah a lokace nabídky. Pod hranou telefonu se nachází tlačítko „Post“ pro odeslání nabídky.



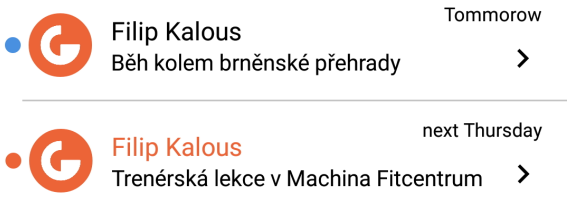
Obrázek 4.6: Tři obrázky postupně znázorňující registraci uživatele pomocí e-mailové adresy. **Levý obrázek:** Obrazovka sloužící k zadání e-mailu, kontrole, zda je volný a přesměrování na další obrazovku. **Prostřední obrázek:** V této části registrace aplikace získá základní data o uživateli. Obsahuje vstup pro jméno a dva přepínače. První pro upřesnění, zda chce být uživatel trenér, druhý pro vybrání pohlaví uživatele. **Pravý obrázek:** Po úspěšném odeslání dat o uživateli, je zaslán uživateli e-mail s aktivačním kódem. Na této obrazovce tento konkrétní kód zadá do jediného vstupu. Pokud byl kód správný, je přesměrován do hlavní části aplikace.

Řazení a filtrování Řazení nabídek nebylo jednoduché si ujasnit. Bylo třeba vzít v potaz, že každý uživatel má jiné preference. Proto nebylo možné nastavit výchozí řazení a uživateli neumožnit změnu. Seznam je primárně seřazen podle počátečního času nabídek. Řazení bude možné změnit výběrem z několika možností – podle sportovní aktivity, ceny, atd. Možností navíc by také mohlo být určit si řazení pomocí výběru v nastavení.

Hlavní aktivita taktéž bude obsahovat funkci vyhledávání, přesněji filtrování v nabídkách. Tlačítko s touto funkcí bude přímo v aplikačním panelu (toolbar). Filtrovat nabídky by mělo být možné podle vícero parametrů a preferencí. Minimálně podle sportovní aktivity, jména, pohlaví.

4.2.5 Seznam lokací

Jednou z důležitých částí aplikace je seznam fitcenter a sportovních zázemí. K tomuto účelu se perfektně hodí mřížkové rozložení. Toho jde docílit mnoha způsoby. Rozhodl jsem se pro *RecyclerView*, který je více flexibilní a přináší více možností uprav oproti *ListView*. Ve výchozím nastavení se zobrazí dvě lokace na jednom řádku. Pokud bude lokace něčím specifická, zabere celý řádek.



Obrázek 4.7: Obrázek znázorňující položky seznamu. Každý řádek zobrazí malou fotku tvůrce nabídky a jeho jméno, den nebo čas konání (záleží v jakém časovém horizontu se aktivita uskuteční) a její popis. Malé kolečko úplně vlevo symbolizuje cenu nabídky. Modré značí neplacenou, oranžové naopak nabídku za poplatek. Jméno nabízejícího nabývá také dvou barev. Oranžový text značí, že uživatel je trenér. Další informace je možné se dozvědět po kliknutí na položku.

4.3 Back-end a komunikace s ním

4.3.1 Databáze

Databázi jsme navrhovali spolu s kolegou. Databáze bude obsahovat 11 tabulek, přičemž se počítá s její expanzí. Entity-Relationship diagram, který byl navržen při tvorbě databáze je ukázán v příloze B.

Tabulky a jejich krátký popis (k vybraným tabulkám je podrobnější popis v následujících odstavcích):

- Users – obsahující data uživatelů aplikace.
- Relationships – popisující vztahy mezi uživateli.
- Offers – obsahuje nabídky uživatelů.
- OfferTypes – obsahuje možné sportovní aktivity, které jdou přidělit k nabídkám.
- Places – obsahuje místa (fitcentra, tělocvičny, apod.).
- PlaceImages – obsahuje URL adresu k obrázku a cizí klíč k příslušnému místu
- Addresses – zde se nacházejí adresy míst spojeny cizím klíčem k místu.
- Countries – neměnící se tabulka se všemi zeměmi na světě.
- States – obsahuje části země, v případě ČR kraje. Napojeno cizím klíčem na příslušný stát.
- AdminUsers – obsahuje uživatele s většími právy, více popíši dále.

Tabulka Users Tabulka bude obsahovat jen nezbytně nutné informace o uživatelovi. Velká část sloupců je nepovinná (e-mail, telefonní číslo, město bydliště, atd.) a uživatel bude moci údaje doplňovat podle vlastního uvážení. Jediné nutné údaje k registraci budou křestní jméno, příjmení a pohlaví uživatele.

Tabulka Relationships Tato tabulka ukládá informace o vzájemném sledování se uživatelů navzájem. Každý záznam obsahuje identifikační číslo (dále jen id) sledujícího a sledovaného.

Tabulka Offers Všechny nabídky aktivit vytvořené v aplikaci budou zde. Nabídka bude vždy pomocí id navázána na jejího tvůrce. Dále v ní budou uložená nezbytná data. Id odkazující na typ aktivity, popisek, místo či lokace konání (aktivity) a hlavně počáteční a koncový čas. Místo, kde se aktivita bude odehrávat může být definováno dvěma způsoby. Pokud uživatel vybere místo z uložených v databázi, záznam bude obsahovat jeho id. Když uživatel zapíše místo konání přímo, bez vybraní z možností, v záznamu se objeví název místo id.

Tabulka AdminUsers Tato tabulka bude sloužit k uložení přístupových údajů k webovému rozhraní. V něm budou správci přidávat, upravovat a doplňovat informace k místům. V tabulce budou sloupce jen nutné pro přihlášení – id, e-mail a zašifrované heslo. Registrace uživatele bude probíhat pomocí formuláře na webu Gymberu.

4.3.2 Popis Gymber API

Representational state transfer Representational state transfer (zkráceně REST) je architektura rozhraní, navržená pro distribuované prostředí. REST implementuje čtyři základní metody, známé pod označením CRUD[3]. K těmto metodám jsou implementovány odpovídajícími metody HTTP protokolu, kterými jsou GET, POST, PUT a DELETE. Tato architektura se postupem času společně s formátem JSON stává standardem při vývoji API a bude využita i v případě Gymber API.

Pro popis Gymber API využíváme služby Apiary. Tato služba umožňuje popsání aplikačního rozhraní a rovnou i jeho testování. Testování probíhá přes privátní adresu. Jako výborný program k testování a zjišťování funkcionality API posloužil Postman[4]. Tato služba pomáhala při psaní celého API a back-endu.

4.3.3 Zabezpečení komunikace

Komunikaci mezi aplikací a back-endem nemůže odstartovat kdokoliv. Je potřeba kontrolovat, zda uživatel posílající dotaz disponuje dostatečnými právy. Gymber k tomu využívá dvou klíčů.

Každá instalace aplikace má v sobě zabudován autorizační token. Ten poslouží jako klíč k registraci uživatele. Na serveru proběhne kontrola tokenu, pokud se shoduje a uživatel není zaregistrován, vytvoří se účet. Odpověď na registraci obsahuje nový, náhodně vygenerovaný API klíč, ten si aplikace uloží a využívá ho při každém dotazu na server. Při absenci tohoto klíče není možná autorizace a dotaz není uskutečněn. Níže jsou uvedeny příklady klíčů.

- Autorizační klíč – 8715a0786e870f02eeb7e957c167d37848310fd476a2371cede9b1f12e 7fd354ff6da5911cf77c201564c822c66a6affce4e72fd442103d708e2264fa702fd3b
- API klíč – mgtDmmaagVYPFu–RN-KKg

[3] vytvoření (create), čtení (read), editace (update), smazání (delete)
[4] Více na https://www.getpostman.com/

Kapitola 5

Implementace

V této kapitole projdu implementaci zajímavých částí aplikace, zakomponované knihovny a důvody k jejich použití. Dále bude popsána tvorba back-endu a jak aplikace posílá/získává data na/z databáze.

5.1 Aplikace pro Android

5.1.1 Datové modely

V této části se zaměřím na vlastní datové struktury používané v aplikaci a k čemu slouží.

FullActivityItem() Model použitý pro zobrazení dat na hlavní obrazovce v seznamu nabídek. Model obsahuje všechny nutné atributy pro zobrazení dat v položce seznamu 4.7. Navíc obsahuje id nabídky, uživatele a lokace. Ty jsou dále důležitá při zobrazení detailu nabídky.

DetailOfferModel() Detailní model se tvoří po rozklepnutí nabídky, rozparsováním dat přijatých z databáze. Obsahuje jak základní data o nabídce, tak získaná data o lokaci a uživateli. K tomu využívá dvou dalších modelů popsaných dále (UserModel, PlaceModel). V modelu jsou implementovány dvě metody k usnadnění získávání informací, jmenovitě *getLocationName()* a *isPlaceSet()*. Jak bylo zmíněno v návrhu, uživatel může využít dvou způsobů sdělení místa konání. Tyto metody upřesní, který zvolil.

UserModel() Datový model kopírující schéma tabulky *Users* v databázi. Dále obsahuje metodu pro získání celého jména. Model je využit u zobrazení detailu nabídky, tvoření seznamu sledovaných uživatelů a samozřejmě při zobrazení profilu uživatele.

PlaceModel() I tento model sedí k odpovídající tabulce v databázi. Opět je využit při zobrazení detailu nabídky. Dále se využívá při tvorbě seznamu dostupných míst. Pro snazší práci disponuje dvěma modely:

- *OpeningHoursModel* – využívá se pro uložení získané otevírací doby.
- *AddressModel* – obsahuje adresu.

5.1.2 ApplicationController

Protože v aplikaci používám knihovnu Volley (širší popis v kapitole 5.1.5) a *SharedPreferences*, rozhodl jsem se vytvořit třídu *Application Controller*. Tato třída rozšiřuje třídu *Application*. Je založena na návrhovém vzoru jedináček (Singleton), tudíž se vytvoří její jediná instance při startu aplikace, která běží až do úplného ukončení aplikace. Obsahem této třídy je fronta kumulující požadavky přes Volley, potom instance *SharedPreferences* a také obsahuje instanci *JSONParseru*. Co se týká uživatele, třída obsahuje API klíč a autorizační token.

5.1.3 SharedPreferences

Pod pojmem *SharedPreferences* se skrývá objekt pro ukládání klíč-hodnota dvojic. *SharedPreferences* jsou sdílená pro celou aplikaci narozdíl od *Preference*, které patří přímo k aktivitě. V kódu 5.1 je příklad jak vytvořit *SharedPreferences* s klíčem „sharedPref“ a následně je ukázán i přístup k nim. Při návrhu jsem se inspiroval z postupů na webu Yakiva Mospana [8].

```
1  private static SharedPreferences sharedPreferences;
2  private static SharedPreferences.Editor sharedEditor;
3
4  sharedPreferences = getApplicationContext().getSharedPreferences("sharedPref
       ", MODE_PRIVATE);
5  sharedEditor = sharedPreferences.edit();
6
7  public static SharedPreferences.Editor getSharedEditor() {
8    return sharedEditor;
9  }
10
11 ApplicationController.getSharedEditor().putInt("user_id", userId).commit();
```

Listing 5.1: Kód zobrazující vytvoření *SharedPreference* s klíčem „sharedPref“. První dva řádky deklarují statické proměnné vytvářených preferencí a jejich editoru. Řádek 4 představuje získání těchto sdílených nastavení v módu „MODE_PRIVATE“. Tento mód znamená, že soubor, v němž jsou hodnoty ukládány, bude přístupný jen z aplikace Gymber. Data jsou uložená ve formátu XML. Řádek 7-9 ukazuje metodu pro získání editoru, pomocí kterého je možné upravovat *SharedPreferences*. Na řádku 11 je ukázka vložení hodnoty id uživatele pod klíčem „user_id“.

Aplikace obsahuje dva druhy *SharedPreferences* – jedny pro uložení informací o uživateli, jako identifikační číslo, jméno apod., druhé pro uložení nastavení z aktivity *SettingActivity*. První jmenované mají klíč „sharedPref“, druhé „settingsPref“. Každá z *sharedPreferences* je vytvořena v *ApplicationController* s odpovídajícím editorem. Podle klíče jsou pojmenované i metody pro přístup k těmto preferencím, a to *getSharedPreferences* a *getSettingsPreferences*. Při potřebě upravovat nebo vkládat data je potřeba se dostat k editorům. K tomu slouží metody *getSharedEditor* a *getSettingsEditor*.

5.1.4 Aktivity aplikace

V této sekci popíší jednotlivé aktivity aplikace i jejich podčásti. Vzhled základních obrazovek je ukázán v příloze C.

JSONParser Ještě než se pustím do popisu samotných aktivit, lehce bych se zaměřil na vlastní pomocnou třídu *JSONParser*. Protože v aplikaci se ve více případech potřebují

podobná data, např. o uživateli, nebylo by moudré duplikovat kód. Parsování JSON dat by prodlužovalo a znepřehledňovalo kód v aktivitách. Těmto problémům jsem předešel vytvořením této pomocné třídy, která zprostředkovává všechnu práci s informacemi ve formátu JSON.

MainActivity *MainActivity* je hlavní aktivitou aplikace. Když přeskočím aktivity spouštěné z důvodu přihlášení, tato je výchozí aktivitou pro celou aplikaci. Zde se inicializuje navigační menu *navigationView*, toolbar a napojení na Facebook API.

Menu funguje tím způsobem, že *MainActivity* přepíná fragmenty příslušných částí aplikace. Přepínání fragmentů je popsáno v kódu 5.2.

```
1 FragmentTransaction trans   = fragmentManager.beginTransaction();
2 Fragment fragment     = new MainFragment();
3 trans.replace(R.id.layout_frame, fragment);
4 trans.addToBackStack(null);
5 trans.commit();
```

Listing 5.2: Přepnutí fragmentu pomocí FragmentManageru. Pro přepnutí se využívá transakce, která tuto výměnu zprostředkuje. Do proměnné *fragment* je uložen nový fragment. Metoda *replace()* vymění v layoutu specifikovaném pomocí id *R.id.layout_frame* aktuální uživatelské rozhraní z nově vytvořeného fragmentu. Funkce *addToBackStack()* přidá starý fragment na zásobník. Tím pádem je možné se k němu dostat při stisknutí tlačítka zpět. Metoda *commit()* transakci potvrdí a fragmenty se prohodí.

I když *MainActivity* přepíná několik fragmentů, rád bych se zaměřil nejdříve na *MainFragment*. Tento fragment nahraje aktivita již při svém startu a zobrazuje seznam nabídek (feed). Ten je načítán pomocí knihovny Volley. Množství zobrazovaných dat vyžaduje asynchronní načítání v pozadí, protože tato operace může trvat i několik sekund. Při jejím provedení na hlavním vlákně by se uživatelské rozhraní zaseklo, než by byla dokončena. Získaná data jsou zobrazována pomocí položek v *ListView*. Tento list využívá vlastní upravený *adapter* sloužící k propojení s daty. Pokud list neobdrží žádná data, zobrazí na obrazovce text „No Content“.

ViewHolder *ViewHolder* je pomocná třída pamatující si reference na prvky. V adaptéru se pro nahraní dat do prvků pracuje s metodou *getView*. Pokud by tato metoda pokaždé načítala prvky z rozložení obrazovky, které chci změnit, zabralo by to zbytečně moc času. Princip využívání *ViewHolderu* tedy spočívá v tom, že při prvním tvoření seznamu se uloží reference na měněné prvky. Pro další zobrazování metoda již využívá těchto uložených referencí.

CreateActivity Aktivita zprostředkovávající formulář pro vytvoření nové nabídky. Uživatelské rozhraní obsahuje prvky pokrývající všechny povinné parametry nabídky. Pokud uživatel zadá cenu nabídky, aktivita odchytne tuto událost a automaticky připojí používanou měnu v systému. Pro výběr data konání je použit výborný *Date Time Picker* zhotovený v souladu s požadavky Material Design. Pro správnou funkcionalitu již stačí jen přepsat metody *onTimeSet* a *onDateSet*. Po vybrání počátečního času nabídky je uložen a použit pro zobrazení dialogu k výběru koncového času s přičtením jedné hodiny. Pakliže uživatel chce změnit čas, při opětovném zobrazení již dialog obsahuje předtím nastavený čas.

Po potvrzení nabídky jsou nejdříve data zkontrolována pomocí metody *postNewActivity*. Jestliže uživatel zapomněl vyplnit některé pole, je na to upozorněn nápovědou v příslušném vstupu a nabídka se neodešle. V opačném případě jsou data předána metodě

prepareOfferForPost v *JSONParseru*. Po zpracování nabídky je na ploše zobrazen *Toast* informující, zda se nabídka uložila či nikoli. Toast je prvek poskytující jednoduchou zpětnou vazbu na provedenou operaci. Při jeho zobrazení se na spodní části obrazovky objeví malé okénko s definovanou zprávou. V tomto kontextu má zpráva znění – „Offer was created.“ nebo „Offer wasn't created.“.

DetailActivity *DetailActivity* zastřešuje dva fragmenty, a to *DetailFragment* a *AcceptFragment*. První z jmenovaných slouží k zobrazení detailních informací o nabídce. Druhý slouží k přijmutí nabídky.

Aktivita vždy primárně zobrazí fragment s informacemi (obsahuje informace o nabídce, odkaz na uživatele, lokaci). Tento fragment obsahuje tlačítko, které je svou funkcí závislé na aktivitě, která *DetailActivity* spustila. Tuto informaci zjistí aktivita, respektive fragment, ze záměru (Intent). V kódu 5.3 je popsáno, jak bude tlačítko reagovat.

Po spuštění *AcceptFragment* má možnost uživatel předat nabízejícímu zprávu a případně upřesnit počáteční čas. Po potvrzení nabídky se pošle JSON zpráva na server. Obsahuje id uživatele, který přijal nabídku. Zpráva a upřesněný čas se zasílá jen v případě změny vstupů.

```
1  int myDetail = getActivity().getIntent().getExtras().getBoolean("my_detail");
2  Button acceptOrDeleteButton = (Button) view.findViewById(R.id.accept_button);
3  .
4  .
5  @Override
6  public void onClick(View v) {
7    switch (v.getId()) {
8
9      case R.id.accept_button:
10
11       if(myDetail == 0)
12         DetailActivity.changeToAccept(userName, timeFromAct, offerId);
13       else if(myDetail == 1) {
14         final SimpleDialog mDialog = new SimpleDialog(getActivity(), R.style.
             Material_App_Dialog_Simple);
15
16         mDialog.message("Are you sure, you want to cancel joining this activity?");
17         mDialog.positiveAction("YES")
18           .negativeAction("NO")
19           .title("Delete offer")
20           .cancelable(true)
21           .canceledOnTouchOutside(true)
22           .show();
23
24         mDialog.negativeActionClickListener(new View.OnClickListener() {
25           @Override
26           public void onClick(View v) {
27             mDialog.dismiss();
28           }
29         });
30
31         mDialog.positiveActionClickListener(new View.OnClickListener() {
32           @Override
33           public void onClick(View v) {
34             cancelOrDeleteOffer(false);
35             mDialog.dismiss();
36           }
37         });
38       }
39       else {
40         .
41         .
42         .
43       }
44       break;
45   }
46 }
```

Listing 5.3: Kód ukazuje chování tlačítka Accept/Cancel/Delete na základě proměnné *myDetail*. Pokud aktivitu spustila *MainActivity*, tak proměnná *myDetail* bude rovna hodnotě 0 a zavolá se metoda pro zobrazení fragmentu *AcceptFragment*. Jestliže *myDetail* nabývá hodnoty 1, vyskočí dialog s potvrzením o zrušení připojení k aktivitě. Poslední případ nastane, pokud je proměnná rovna 2. V tom případě je zobrazen dialog o vymazání nabídky z databáze. Kód pro druhý a třetí případ je prakticky stejný, až na jinou hlášku a zavolání metody *cancelOrDeleteOffer()* s jinou hodnotou parametru, proto jsem se rozhodl v kódu ukázat jen část pro zrušení připojení k aktivitě.

SplashActivity V tomto a dalších odstavcích popíši, co obnáší zapnutí aplikace a přihlášení uživatele. Již v návrhu byl zmíněn účel této aktivity. Zaprvé zkontroluje dostupnost připojení k internetu (implementace podle kódu 5.4). Při nepřítomnosti sítě je uživatel upozorněn a aktivita vyčká, dokud není (opět) připojen. Tato akce je potvrzena klepnutím do obrazovky.

V dalších krocích zkontroluje, zda je již uživatel přihlášen. Tuto informaci zjistí díky záznamu v *SharedPreferences* s defaultním nastavením nepřihlášen. Pakliže je již přihlášen, aktivita vytvoří záměr a spustí *MainActivity*. Nepřihlášený uživatel bude stejným způsobem přesměrován do *LoginActivity*.

```
1  private boolean isInternetAvailable() {
2    ConnectivityManager cm = (ConnectivityManager) getApplicationContext().getSystemService(Context.
         CONNECTIVITY_SERVICE);
3
4    NetworkInfo activeNetwork = cm.getActiveNetworkInfo();
5
6    return activeNetwork != null && activeNetwork.isConnectedOrConnecting();
7  }
```

Listing 5.4: Kontrola dostupnosti připojení k internetu

LoginActivity Jak již bylo řečeno, *LoginActivity* je spuštěna v případě, že uživatel není ještě přihlášen. K tomuto stavu se může dostat dvěma způsoby:

- První instalace aplikace nebo reinstalace
- Vymazání paměti přidělené k aplikaci (cache)

LoginActivity zpracovává všechnu logiku ohledně přihlášení pomocí Facebooku. Registraci a opětovné přihlášení do aplikace zajišťuje aktivita popsaná v následující sekci.

```
1  FacebookSdk.sdkInitialize(getApplicationContext());
2  callbackManager = CallbackManager.Factory.create();
3
4  loginFBButton.registerCallback(callbackManager, new FacebookCallback<LoginResult>() {
5    @Override
6    public void onSuccess(LoginResult loginResult) {
7
8      getInfoFromFB();
9  }
10
11 private void getInfoFromFB() {
12   GraphRequest request = GraphRequest.newMeRequest(
13       AccessToken.getCurrentAccessToken(),
14       new GraphRequest.GraphJSONObjectCallback() {
15         @Override
16         public void onCompleted(JSONObject object, GraphResponse response) {
17
18           JSONObject userObject = response.getJSONObject();
19           try {
20
21             long id = userObject.getLong("id");
22             String firstName = userObject.getString("first_name");
23             String lastName = userObject.getString("last_name");
24             String gender = userObject.getString("gender");
25
26             registerUser(id, firstName, lastName, gender, AccessToken.getCurrentAccessToken().
                   getToken());
27
28           } catch (Exception e) {e.printStackTrace();}
29         }
30       });
31   Bundle parameters = new Bundle();
32   parameters.putString("fields", "id,first_name,last_name,gender");
33   request.setParameters(parameters);
34   request.executeAsync();
35 }
```

Listing 5.5: Autorizace uživatele a získání dat přes Facebook API

Autorizaci a získání dat popíši podle kódu 5.5. První řádek inicializuje Facebook SDK. Bez tohoto řádku by se žádné dotazy na Facebook neprovedly. Potom následuje reakce

na uživatelovo klepnutí na tlačítko „Přihlásit se přes Facebook“. Při správné autorizaci se zavolá metoda *onSuccess(LoginResult loginResult)*. Pokud se nezdaří, zavolá se metoda *onError(FacebookException error)*. Při zrušení autorizace uživatelem se volá metoda *onCancel()*. Tyto dvě metody nejsou zahrnuty do níže zobrazeného kódu, nejsou v tomto kontextu důležité a zbytečně by prodlužovaly vložený kód.

Správný průběh zařídí spuštění metody *getInfoFromFB()*. Ta obsahuje *GraphRequest* (žádost zprostředkována z Graph API od Facebooku), pomocí níž se získají základní informace o uživateli. Tyto údaje jsou potřeba pro vytvoření účtu v databázi Gymberu. Data, která chci získat, definuji na řádku 32, pomocí *Bundle*, což je třída pro předávání dat mezi aktivitami. Získaná data jsou uložena ve formátu JSON. S těmi se provede registrace uživatele na serveru. Server v odpovědi zašle všechny informace o uživateli, hlavně id a autorizační klíč. Tyto důležité informace pro další práci jsou uloženy do *SharedPreferences*.

RegistrationActivity Aktivita spouštěná při registraci nebo znovu přihlášení uživatele. Mění tři na sobě závislé fragmenty – *RegisterFragmentEmail, RegisterFragmentInfo, RegisterFragmentCode*. Tyto fragmenty popíši tak, jak na sebe navazují. Způsob kontroly registrace uživatele a aktivace jeho účtu na back-endu je popsán později.

RegisterFragmentEmail Princip registrace a přihlašování uživatele byl popsán v předešlé kapitole (odkaz 4.2.3). Z toho vychází, že uživatel v aplikaci nemá přihlašovací jméno a heslo. Tím pádem, postup registrace i přihlášení funguje úplně stejně. Tento fragment má tedy za úkol získat e-mail uživatele. Následně zkontroluje přítomnost e-mailu v databázi. Pokud je již e-mail použit, přeskočí se fragment *RegisterFragmentInfo*.

RegisterFragmentInfo Tento fragment obsahuje třípoložkový formulář. Od uživatele je získáno celé jméno, zapojení uživatele (trenér/sportující) a pohlaví. Tyto údaje se vzápětí pošlou do databáze, aby mohl být zaregistrován nový uživatel. Z odpovědi se poté uloží stejné informace jako při registraci přes Facebook do *SharedPreferences*.

RegisterFragmentCode Poslední fragment registrace má za úkol zkontrolovat aktivační kód zadaný uživatelem s jeho protějškem v databázi. Při úspěšné kontrole je uživatel přihlášen do aktivace. S tímto krokem se ukončí celá aktivita *RegistrationActivity*.

Jak jsem se zmínil dříve, *MainActivity* přepíná mezi fragmenty pomocí navigačního menu. Předtím jsem popsal jen *MainFragment*, nyní projdu i ostatní fragmenty.

PlacesFragment *PlacesFragment* v aplikaci zobrazuje seznam dostupných lokací uložených v databázi. Seznam funguje na stejném principu jako v případě výpisu nabídek v *MainFragment*. Rozdíl je v použitém listu. Zde je použit *RecyclerView listPLaces*. Pro uložení referencí na prvky v rozložení je opět využito *ViewHolderu*. Kromě vlastního adaptéru *PlacesAdapter* rozšiřující *RecyclerView.Adapter*, využívá *listPlaces* ještě manažer rozvržení *GridLayoutManager*. Ten je použit kvůli zobrazovaní položek do mřížky. Navíc umožňuje skrze metodu *getSpanSize(int position)* určit, kdy se položka má roztáhnout přes oba sloupce.

RecyclerView oproti *ListView* nemá přímou podporu pří klepnutí na některou položku pomocí *OnItemClickListener*. Po hledání řešení jsem se nakonec rozhodl využít třídu *ItemC-*

lickSupport dostupný z webu Little Robots[1]. Po klepnutí na položku je spuštěna *LocationActivity* zobrazující všechny dostupné informace o lokaci.

LocationActivity – detail lokace Aktivita zobrazuje všechny dostupné informace o lokalitě. Největší část obrazovky tvoří fotografie lokace. Pod ní se zobrazí adresa a otevírací doba. Do budoucna se počítá s recenzemi lokace od uživatelů. Tyto recenze budou zobrazovány pomocí *ListView* s vlastním vzhledem a adaptérem. Aktivita umožňuje pomocí tlačítka spustit aktivitu pro volání, kam vloží telefonní číslo získané z databáze. Aby tuto funkci systém povolil, musí soubor *AndroidManifest.xml* definovat oprávnění pro aplikaci *android.permission.CALL_PHONE* a aktivita musí být vytvořena se záměrem *Intent.ACTION_DIAL*. Do tohoto záměru se vloží telefonní číslo pomocí metody *Uri.parse()* ve formátu například „tel:789456123“.

ActivitiesFragment a FavouritesFragment *ActivitiesFragment* je přehled vytvořených a přijatých nabídek uživatelem. Fragment je rozdělen na dvě části pomocí tzv. „tabů“ na „My activities“ a „Joined activities“. Mezi těmito taby jde při běhu aplikace volně přepínat. Seznamy jsou tvořeny pomocí *ListView* a mají vlastní adaptéry s jinými *ViewHoldery*. Při klepnutí na položku v tabu „My activities“ je uživatel přesměrován na *DetailActivity*. Zde může nabídku smazat. V tabu „Joined Activities“ se po klepnutí na položku seznamu zobrazí také *DetailActivity*, ovšem s možností zrušit připojení k nabídce.

FavouritesFragment představuje seznam uživatelů, které přihlášený uživatel sleduje. Data pro list získá z databáze z tabulky *Relationships*, které rozparsuje do *UserModel*. Ta pak předá adaptéru pro *ListView*. Fragment obsahuje *OnItemClickListener*, který po klepnutí na položku spustí *PersonActivity*.

Následující odstavce se zaměří na aktivity týkající se uživatelů.

PersonActivity *PersonActivity* zobrazuje detail jiného uživatele. Aktivitě dominuje fotka uživatele a pro přehlednost obsahuje také dva taby – „Summary (přehled)“ a „Info“. Pro zobrazení a přepínání tabů je využit *TabLayout + ViewPager*. K *ViewPageru* je přidělen vlastní adaptér, který se stará o přepínání mezi taby. Každý z nich je představován vlastním fragmentem. Při implementaci tohoto adaptéru je nutné přepsat dvě metody, *getItem(int position)* – dle pozice vytvoří a vrátí fragment a *getCount()* – vrací počet tabů.

Druhý tab, jak již název napovídá, obsahuje jen dostupné informace (např. pohlaví a město bydliště, atd.). Tab „Summary“ již obsahuje zajímavější informace – kolik nabídek vytvořil a přijal, počet jím sledovaných a jeho sledujících uživatelů. Také je možné zobrazit poslední aktivity, které vytvořil. Pod fotku je k nalezení tlačítko s tvarem srdce, které po klepnutí zapíše/zruší sledování uživatele do databáze.

ProfileActivity Tato aktivita zprostředkovává uživateli jeho profil a úpravy nad ním. Aktivita přepíná mezi dvěma fragmenty – s přehledem (*ProfileSummaryFragment*) a editací údajů (*ProfileEditFragment*). Přehled je velmi podobný jako tab „Summary“ v *PersonActivity*, proto se v následujícím odstavci zaměřím na tab s editací.

Aby každý fragment nemusel načítat ta samá data z databáze dvakrát, načtou se již v *ProfileActivity* a do fragmentů jsou předávána pomocí rozhraní *EventBus* (viz. 5.1.5).

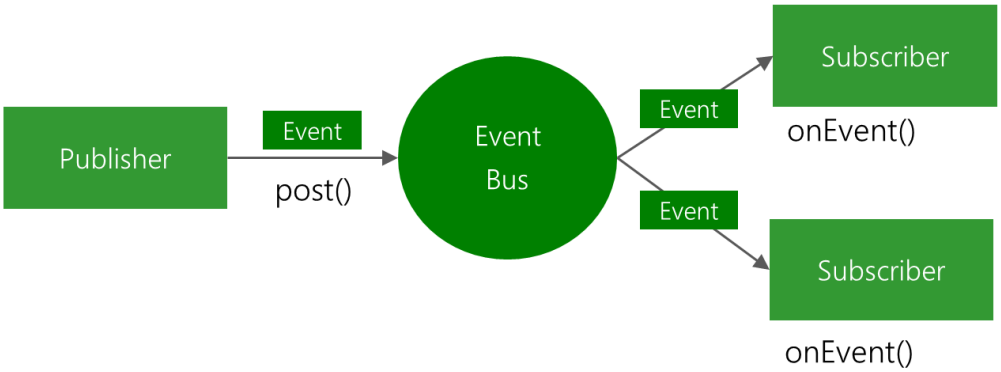
[1] http://www.littlerobots.nl/blog/Handle-Android-RecyclerView-Clicks/

Formulář pro úpravu údajů si z těchto dat načte hodnoty a připraví je jako nápovědy. Před odesláním dat proběhne kontrola, zda bylo něco změněno, jinak se nic neodešle. Pokud uživatel klepne na tlačítko zpět, vyskočí dialog s dotazem na uložení změn. Protože fragment nedokáže ovládat tyto události, musí kontrolu i případné uložení hodnot obstarat aktivita. Ta ale ovšem neví, které údaje byly změněny. Tento problém jsem vyřešil definováním popisku (tagu) při vytváření fragmentu pro editaci. V dialogu, při zvolení uložení hodnot, najdu fragment ve *FragmentManager* právě pomocí definovaného popisku a zavolám metodu *saveEditedValues* nad ním.

SettingsActivity – nastavení *SettingsActivity* umožňuje uživateli změnit některé vlastnosti aplikace. Tato obrazovka je přístupná z navigačního menu nebo aplikačního panelu (toolbaru). Obsahuje vstup pro změnu jazyka aplikace. Jednou z hlavních možností nastavení je změna výchozího řazení nabídek v seznamu na hlavní stránce. Pokud uživatel využije vyhledávání v seznamu nabídek, aplikace si uchovává historii dotazů. Proto nastavení obsahuje tlačítko, které tuto historii smaže. Počet nastavení bude v budoucnu stoprocentně růst s tím, jak se budou do aplikace přidávat funkce.

5.1.5 Použité knihovny

Volley *Volley* je knihovna pro rychlou a snadnou komunikaci se sítí, vyvinutá společností Google. V aplikaci ji využívám jako náhradu za *AsyncTask*, z kterého principiálně vychází. Vytvořená byla z toho důvodu, že Android SDK neobsahoval žádnou třídu pro práci se sítí, aniž by byl rušen uživatel zamrznutím uživatelského rozhraní apod. V dřívějších verzích API se používali pro síťovou komunikaci třídy *HttpUrlConnection* nebo *HttpClient*. Tyto třídy avšak obsahovaly bugy a od API 22 je *HttpClient* považován za zastaralý a nepodporovaný. *Volley* pracuje ve třech vrstvách – žádost, cache a síťová žádost. Pokud se po vygenerování žádosti najde záznam v cache je ihned vrácena odpověď, jinak se zasílá žádost na definovanou URL adresu. V hlavním vlákně je zpracovávána jen žádost a odpověď. Tím pádem uživatel může během komunikace po sítí pracovat dál, aniž by byl rušen. Žádosti se vkládají do fronty a automaticky se vykonávají. *Volley* navíc dovoluje prioritizovat žádosti. Více o



Obrázek 5.1: Princip knihovny EventBus. Aktivita (Publisher) umístí na sběrnici událost. Ta se přenese na sběrnici a čeká, kdo si ji vyzvedne. Aby si tuto událost mohla jiná aktivita vyzvednout, je nutné se ke sběrnici registrovat (*eventBus.register(this)*). Knihovna navíc umožňuje přenášet tzv. „sticky“ události, které zůstanou na sběrnici i po vyzvednutí. Tyto události jsou označeny parametrem *sticky*. Při ukládání položky na sběrnici nezáleží na typu vkládaného objektu.

knihovně je možné najít na webu Volley[2].

Facebook SDK *Facebook SDK*, momentálně ve verzi 4.11.0, umožňuje integrovat služby Facebooku do aplikace. Po přihlášení se aplikace propojí s uživatelským účtem. Gymber služeb Facebooku využívá jako alternativní volbu registrace do aplikace a pro získání základních uživatelových dat pomocí Graph API. Více o Facebook SDK a Graph API je k nalezení na webu Facebooku pro vývojáře[3].

EventBus *EventBus* představuje jednoduchou sběrnici, která přenáší data mezi komponentami aplikace. V Androidu jdou mezi komponentami (aktivitami) přenášet data jen pomocí záměrů (Intent). V případě, že je potřeba přenášet například celý model, je tato operace velmi časově náročná. *EventBus* je mnohem rychlejší a navíc velmi malá knihovna (50 kB), takže aplikaci nezatíží. Princip knihovny je ukázán a vysvětlen na obrázku 5.1. Při práci s touto knihovnou jsem také využíval článku Andrease Schradeho [7]. Více o této knihovně zde[4].

Material Date Time Picker Material Design představuje i formulář pro výběr data (času). V Androidu se prvek, který by tento formulář představoval, nenachází. Tato knihovna představuje přesně tyto prvky, a navíc podporuje Android již od verze 4.0. Po nastavení knihovny jako závislosti k aplikaci již pro použití stačí přidat tento prvek do uživatelského rozhraní. Následně nastavit aktivitu (fragment) aby implementovala rozhraní *OnTimeSetListener* nebo *OnDateSetListener*, a podle použitého prvku implementovat metody *OnTimeSet()* nebo *OnDateSet()*. Více o této grafické knihovně je možné nalézt na GitHubu[5].

MaterialLibrary *MaterialLibrary* je open-source knihovna umožňující používat prvky pro MaterialDesign ve verzích nižších než Android Lollipop (5.0). Původním autorem je Rey Pham. Z této knihovny jsem v aplikaci využil následující prvky – *ProgressView* a *Spinner*. Více o knihovně na oficiální GitHub stránce[6].

CircleImageView Prvek uživatelského rozhraní pro zobrazení obrázku v kruhu. V aplikaci je použit pro zobrazení profilových fotek, k čemuž se hodí nejvíce. Více o knihovně *CircleImageView* na GitHubu[7].

5.2 Serverová část

V této sekci projdu implementaci back-endu pomocí Ruby on Rails.

5.2.1 Datové modely

Jako počátek práce na back-endu se dá označit vytvoření modelů. Model se skládá z několika částí, jak je vidět z kódu 5.6. Definuje vztahy s ostatními modely (belongs_to,

[2] http://developer.android.com/training/volley/index.html
[3] https://developers.facebook.com/docs/android/
[4] https://github.com/greenrobot/EventBus
[5] https://github.com/wdullaer/MaterialDateTimePicker
[6] https://github.com/rey5137/material
[7] https://github.com/hdodenhof/CircleImageView

has_many, has_one), dále validaci ukládaných dat a veřejné i privátní metody. Samotné atributy modelu jsou pak popsány v schématu databáze.

Back-end aplikace Gymber tvoří jedenáct modelů (uvedené v kategoriích, jak spolu souvisí):

1. *AdminUser*
2. *Offer, OfferType*
3. *User, Relationship*
4. *Place, PlaceImage, PlaceType, Address, Country, State*

```
class Offer < ActiveRecord::Base
  belongs_to :user
  belongs_to :place
  belongs_to :offer_type
  enum state: [:pending, :accepted, :canceled]

  validates :user_id, :time_from, :time_to, presence: true
  validates :budget, numericality: { greater_than_or_equal_to: 0.0 }, presence: true
  validates_length_of :description, :maximum => 140, :allow_blank => true
end
```

Listing 5.6: Ukázka konkrétního modelu – Offer. Z kódu je vidět, že model dědí třídu *ActiveRecord::Base*. Řádky obsahující metodu *belongs_to* vytvářejí vztah na jiné modely, zde konkrétně na *user*, *place* a *offer_type*. Pokud je použit tento vztah, v tabulce *offers* budou vytvořeny cizí klíče na záznamy v příslušících tabulkách. Metody začínající na *validates* vymezují požadavky na vkládaná data. Například *description* nesmí být delší jak 140 znaků.

Model AdminUser je použit pro registraci uživatele, který upravuje místa (lokace) v databázi pomocí formulářů na webu. S aplikací jako takovou de facto nesouvisí.

Nejobsáhlejší model aplikace je User. Obsahuje metody na vytváření autorizačních tokenů, aktivačních klíčů, autentizaci při přihlašování a tvorbu nových vztahů mezi sledovanými uživateli. K šifrování těchto důležitých tokenů a klíčů je využit gem *BCrypt*. Ten, jak již je vidět z názvu, používá šifrovací funkci Bcrypt k odvození 60 znakového klíče. Do databáze se následně ukládají zašifrované hodnoty. Samotný autorizační token je tvořen jako náhodný 22 znakový *base64* kód. Ostatní modely jsou jednodušší a obsahují jen vazby a validační metody.

5.2.2 Databáze

Tabulky v databázi se tvoří pomocí migrací a zakládají se na modelu. Migrace pro tvorbu tabulky obsahuje jméno, typ sloupce a integritní omezení. Cizí klíče se vytvoří automaticky podle vazeb na modelu. V případě M:N relace je i vygenerována rozpisová tabulka obsahující klíče z obou tabulek. Takto vznikly tabulky – *offers_users a places_users*. Ukázka 5.7 ukazuje zápis dvou migrací.

```
class CreateOffers < ActiveRecord::Migration
  def change
    create_table :offers do |t|
      t.decimal :budget, precision: 5, scale: 2, default: 0
      t.text :description
      t.datetime :time_from
      t.datetime :time_to
      t.string :location
      t.timestamps null: false
    end
  end
end

class AddActivationToUser < ActiveRecord::Migration
  def change
    add_column :users, :activation_digest, :string
    add_column :users, :activated, :boolean, default: false
```

```
    add_column :users, :activated_at, :datetime
  end
end
```

Listing 5.7: Migrace – první migrace vytváří tabulku *offers* složenou ze sedmi sloupců, z čehož jeden je id (identifikační číslo). Druhá migrace popisuje přidání sloupců do tabulky *users*

Tabulky *countries* a *states* jsou předvyplněny daty pomocí metody *seed*. Na serveru jsou uloženy dva *csv* soubory, z kterých se načítají názvy zemí a v případě České republiky i kraje. Toto proběhne při vytvoření nebo obnovení celé databáze.

5.2.3 Jednotlivé prvky serverové části

Controllers Kontroléry se starají o zpracování požadavků. Požadavky se třídí podle metod HTTP, což bylo popsané v návrhu. Zde se o směrování starají tzv. *routes*. Podle URL adresy a metody je poznáno, který kontrolér a jeho akce se má použít pro zpracování dotazu. Například pro URL – *https://gymber.herokuapp.com/api/v1/offers* – a metodu GET se použije *OffersController* s akcí *index*. Jak jsem znímil, back-end je rozdělen na dvě části. Jedna se stará o přidávání míst adminy. Druhá zpracovává požadavky od aplikace. Nejdříve projdu první, menší část.

Admin sekce Tato sekce je celá zapouzdřena v složce (namespace) */controllers/admin*. V ní je definován *BaseController*, z kterého ostatní kontroléry dědí. *BaseController* dědí základní třídu *ApplicationController* a kontroluje jenom, zda je uživatel přihlášen. *AdminUsersController* pracuje nad modelem *AdminUser* a zajišťuje registraci uživatele. K přihlášení uživatele je použit *SessionsController*. Po validaci přihlašovacích údajů je uživateli vytvořeno sezení (session), které se udržuje až do odhlášení. Toto sezení je udržováno a spravováno v modulu *Admin::SessionsHelper* nacházející se ve složce */helpers/admin*. Tento pomocník obsahuje metody k vytvoření i zrušení sezení, získání aktuálního uživatele a jeho kontrolu přihlášení.

K samotnému vkládání lokací a obrázků k nim náležícím jsou připraveny dva kontroléry – *PlacesController* (nad modelem *Places*), *PlaceImagesController* (nad modelem *PlaceImage*). První jmenovaný zajišťuje uložení záznamu o lokaci do databáze, jeho případné editaci či smazání. Druhý kontrolér poskytuje stejné akce pro práci s obrázky míst. Rails navíc umožňují omezit přijímané parametry, což se perfektně hodí ke kontrole dat. Jak je vidět na ukázce 5.8, pokud *PlaceImagesController* obdrží zprávu s parametry jinými než uvedenými, vygeneruje chybu a obrázek neuloží.

```
def place_image_params
      params.require(:place_image).permit(:place_id, :image, :is_cover)
end
```

Listing 5.8: Definice metody omezující parametry, které se mohou vyskytnout ve zprávě. Tato konkrétní platí nad modelem *Place Image* v kontroléru *PlaceImagesController*.

API sekce Tato sekce se nachází ve složce (namespace) *controllers/api/v1*. I tato část má vlastní *BaseController*, z kterého ostatní vycházejí. Velký rozdíl je v přihlašování uživatelů. U admin sekce je k tomu využito sezení. Pokud by se stejným přístupem pracovalo i zde, kam bude posílat požadavky mnoho uživatelů, nemuselo by to mít dobré výsledky. Server by si musel udržovat velké množství otevřených sezení, zvlášť když je aplikace vytvořena tak, aby uživatel nebyl přihlašováním zdržován. Toto by mohlo vézt ke zpomalení zasílání odpovědí a ke prodloužení době zpracování požadavků.

Autentizaci uživatele zajišťuje právě *BaseController*. V případě, že proběhne kontrola *X-Auth-Token* i *X-Api-Key* bez problému, je požadavek poslán dál k jednotlivým kontrolérům.

Registraci uživatele zajišťují dva kontroléry – *UsersController* a *UserServicesController*. Kontrolér se volí podle zvolené metody registrace uživatelem v aplikaci. Buď pomocí e-mailu (UsersController) nebo pomocí API třetí strany (UserServicesController). Při registraci uživatele pomocí třetí strany, se ale také využívá *UsersController*, aby nebyl zbytečně duplikován kód. Rozdíl je v kontrolovaných informacích (unikátnost e-mailu nebo id uživatele u třetí strany). *UsersController* je dále využíván při editaci informací uživatelem, zasílání aktivačního e-mailu a jeho následnou kontrolu. Zobrazení odkazu z aktivačního e-mailu obstarává *AccountActivationsController*. V něm zobrazí kód, který uživatel zadá v aplikaci a následně je mu umožněn přístup.

Další kontroléry umožňují již jen základní věci. *OffersController* pracuje s daty uloženými v tabulce *Offers* a nad tím samým modelem. Umožňuje vytvoření nabídky, její získání, smazání a výpis všech nabídek. *PlacesController* zasílá jen seznam všech lokací nebo jednu konkrétní. K získání nabídek na základě id uživatele je vytvořen *UserOffersController*. Poslední kontrolér *RelationshipsController* provádí akce se vztahy mezi uživateli (sledující, sledovaný), a to výpis všech vztahů uživatele, vytvoření vztahu nebo smazání konkrétního vztahu.

Serializers Back-end pracuje s velkým množstvím dat, ať směrem z nebo do aplikace. I když jsou všechna data v JSON formátu, tedy v textové podobě, je rozumné omezit zasílané množství. V některých případech není vhodné zasílat všechny informace ze záznamu z databáze. Například chci získat jen základní informace o uživateli, a ne jeho narozeniny, bydliště apod. K tomuto účelu jsou uzpůsobeny serializéry.

Při zasílání dat, Railsy automaticky hledají podle názvu modelu, zda je k němu vytvořen serializér. Pokud ano, použije ho. Ovšem k jednomu modelu je možné jich vytvářet více. V případě, že u akce v kontroléru se má použít jiný serializér než výchozí, je nutné to specifikovat tak, jak je v ukázce 5.9. Zde je vidět, že při akci *show* chci získat detailní informace o konrétní nabídce. Proto použiji serializér, který pošle většinu dat ze záznamu v databázi. V akci *index*, která posílá záznamy všech nabídek, se využívá výchozí *OfferSerializer* obsahující méně parametrů.

```
def show
  @offer = Offer.find(params[:id])
  render json: @offer, serializer: OfferDetailSerializer, status: :ok if stale?(@offer)
end
```

Listing 5.9: Vyžádání konkrétního serializéru pomocí parametru *serializer*. Tomuto parametru je předán název serializéru, který je nutné použít.

Kapitola 6

Testování a zhodnocení

Po ukončení prací na aplikaci a jejich funkcí je nutné ji patřičně otestovat. Nejvíce jsem se zaměřil na testování uživatelského rozhraní, jako nejdůležitější části aplikace. Dále proběhlo testování k nalezení chyb a pádů. Důležitým faktorem u testování byla také funkčnost bez sekání a zpomalení aplikace.

6.1 Testování uživatelského rozhraní

Zda je uživatelské rozhraní funkční a „user-friendly“, jsem se rozhodl testovat pomocí uživatelů, kteří budou plnit úkoly v aplikaci. Cílem bude zjistit, zda je rozhraní dostatečně jednoduché a intuitivní pro uživatele. Testovací úlohy jsou nastavené tak, abych bylo možné zjistit, zda není problém nalézt základní funkce aplikace. Pro představu uvádím testovací úkony:

- Přihlášení (registrace) uživatele
- Přijetí nabídky
- Vytvoření vlastní nabídky
- Přidání uživatele mezi sledované
- Změna jazyka
- Nalezení mnou přijatých nebo vytvořených nabídek
- Smazání mnou vytvořené nabídky
- Zobrazení otevírací doby lokace
- Editace vlastního profilu

Úkol: Přijetí nabídky Tento úkol by měl představovat nejběžnější činnost v aplikaci. Uživatel po spuštění aplikace uvidí seznam všech nabídek. I začátečník by měl rychle pochopit smysl seznamu a jak se dostat k detailu nabídky. Po vyhledání a vybrání ideální aktivity se klepnutím dostane k detailu aktivity. Po načtení této obrazovky by měl uživatel stisknout velké tlačítko s nápisem „Accept“. Následně je uživateli zobrazena obrazovka s nabídkou zaslání uživateli, který nabídku vytvořil, krátkou zprávu a posunout lehce čas. Test je ukončen klepnutím na tlačítko „Send“.

Úkol: Vytvoření vlastní nabídky Test opět začíná z hlavní obrazovky aplikace. Uživateli by nemělo činit problémy pochopit význam akčního tlačítka plus v pravé dolní části obrazovky. Tímto tlačítkem se dostane k obrazovce pro vytvoření nové nabídky. Uživatel bude mít za úkol vytvořit nabídku podle jeho preferencí. Test končí úspěšným odesláním nabídky.

Úkol: Smazání mnou vytvořené nabídky Před začátkem testu se uživatel bude nacházet na hlavní obrazovce. Uživatel bude mít za úkol se pomocí navigačního menu dostat k němu vytvořené nabídce z minulého testu a smazat ji. Cílem je zjistit jednoduchost přístupu k vlastním nabídkám a jejich smazání.

Vyhodnocení

Nejčastějšími problémy spočívaly ve vyhledání profilu uživatele. Uživatelé byly lehce zmateni z přístupu k profilu přes detail nabídky. Dalším problémem se ukázal být přepínač placené/neplacené nabídky, kdy uživatelé nevěděli, která volba je právě zvolena. Jiné problémy se u vytváření nabídky už nevyskytly. Uživatelé také neměli problém s nalezením obrazovek nastavení, vlastních aktivit a sledovaných uživatelů. Je tedy vidět, že uživatelé umějí bez problémů pracovat s navigačním menu. Uživatelé také rychle pochopili jak se dostat k vlastnímu profilu a změnu informací o sobě. Velmi kladně byl hodnocen přehled jednotlivých lokací.

Na základě zpětné vazby byly všechny akční prvky označeny akční barvou, která symbolizuje, že s těmito prvky je možné provést nějakou akci. Tím se například uživatel rychleji zorientoval v detailu nabídky a přišlo mu logické kliknout na jméno uživatele nebo lokalitu. Problém s přepínačem jsem se pokusil vyřešit zvýrazněním písma ve zvolené části. Vybraná část přepínače má nyní tučný styl písma. S některými uživateli jsem debatoval o vzhledu aplikace a umístění prvků již během implementace. Na otázku, jaké mají dojmy z používání a smyslu aplikace, jsem dostával nejčastěji pozitivní odezvu.

Testování probíhalo jak s uživateli, kteří se pohybují v prostředí techniky a moderních technologií, tak i s průměrnými uživateli chytrých telefonů a uživateli, kteří jím ani nedisponují. Mezi testované subjekty byli zařazeni i pravidelně cvičící lidé.

6.2 Výkonnost aplikace

Výslednou aplikaci je nutné optimalizovat pro velkou škálu chytrých telefonů se systémem Android. Zvláště na starších zařízeních by se mohlo objevovat nepříjemné sekání aplikace. I z tohoto důvodu byla aplikace implementována, aby se ve vlákně s uživatelským rozhraním nevykonávali složité, časově náročné operace. Největší potíží je rychlost připojení k internetu, na které je aplikace velmi závislá. Testování výkonnosti, pádů aplikace nebo případných chyb budou testovat betatesteři v rámci Google Play Store.

Kapitola 7

Závěr

Cílem práce bylo vytvořit moderní aplikaci, která umožní uživateli najít partnera ke sportování, především posilování ve fitcentrech. Na základě návrhu byl tento cíl naplněn. Aplikace navíc přináší přehled sportovních areálů (posiloven, hřišť, atd.). Aplikace má moderní design, jednoduché a intuitivní ovládání. Vzhled aplikace a uspořádání prvků bylo vylepšováno na základě připomínek prvních testovacích uživatelů. Aplikace získává data ze serveru s databází pomocí vlastního privátního aplikačního rozhraní nezávislého na platformě, na které běží aplikace.

Služba Gymber je v době vydání této práce ojedinělou. Teprve zpětná vazba od uživatelů ukáže, zda je skutečně připravena pro plné využití. Největším přínosem aplikace je umožnění navázat kontakt s lidmi, kteří pravidelně cvičí a chtějí předat své vědomosti dále. Při práci na tomto projektu jsem zdokonalil své schopnosti v jazyce Java a Ruby, při navrhování uživatelského rozhraní a získal větší podvědomí o tvorbě aplikací pro systém Android.

7.1 Výhled do budoucna

K získání většího počtu nezbytných uživatelů bude aplikace vhodně propagována. Za tímto účelem bude spuštěna úvodní webová stránka Gymberu, která je již ve vývoji. Jejím účelem bude v rychlosti vysvětlit, co služba přináší a odkazovat návštěvníky přímo do obchodů s aplikacemi jednotlivých platforem. Dalším propagačním kanálem pak zajisté budou sociální sítě a případně blogy trenérů nebo dalších uživatelů. Další testování aplikace bude probíhat také ve spolupráci s fitness studiem 3D Fitness, díky němuž by se o aplikaci mohla dozvědět širší veřejnost.

Než bude aplikace vydána na Google Play Store je potřeba přidat některé chytré funkce, které uživateli ulehčí či zjednoduší práci s aplikací. Mezi tyto funkce bych zařadil získání statistik o uživateli zobrazovaných v profilu, plánování nabídek opakujících se v určitý čas nebo rozšíření možností pro individuální nastavení. Momentálním velkým trendem je sdílení veškerého obsahu na sociální sítě. Proto se jako logický krok jeví zlepšení propojení s Facebookem pro okamžité sdílení nabídek, vlastních statistik atd. Velkým hnacím motorem vylepšování aplikace budou samotní uživatelé a jejich zpětná vazba. Hodnocení a reference v obchodě Google Play Store budou ukazateli spokojenosti a také doufám, že přísunem nových nápadů a vylepšení aplikace.

Literatura

[1] Allen, G.: *Android 4: průvodce programováním mobilních aplikací*. Brno: Computer Press, 2013, ISBN 978-80-251-3782-6.

[2] Sam Ruby, D. T. a. D. H. H.: *Ruby on Rails : průvodce agilním vývojem webových aplikací*. Brno: Computer Press, 2011, ISBN 978-80-251-3647-8.

[3] WWW stránky: Android Developers.
URL http://developer.android.com

[4] WWW stránky: Apiary.
URL https://apiary.io

[5] WWW stránky: Material Design.
URL https://www.google.com/design/spec

[6] WWW stránky: Ruby on Rails Guides.
URL http://guides.rubyonrails.org

[7] WWW stránky (Andreas Schrade): How to use the greenrobot EventBus library.
URL http://www.andreas-schrade.de/2015/11/28/android-how-to-use-the-greenrobot-eventbus/

[8] WWW stránky (Yakiv Mospan): Best practices for SharedPreferences.
URL http://blog.yakivmospan.com/best-practices-for-sharedpreferences/

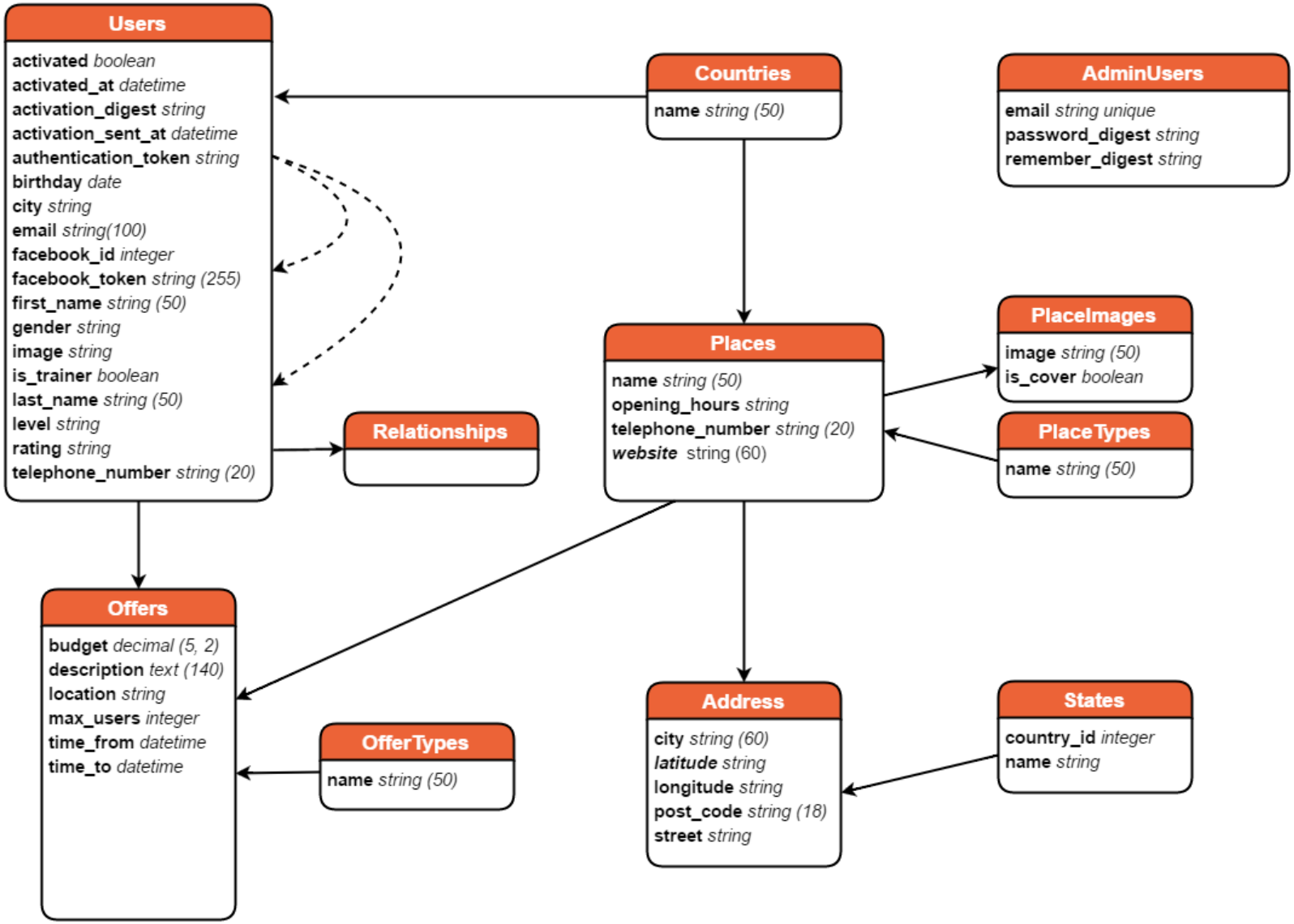
Příloha A

Obsah CD

- BP_elektronicka_verze.pdf – Elektronická forma této bakalářské práce
- BP_verze_pro_tisk.pdf – Verze pro tisk (změna barvy odkazů)
- Gymber.apk – Instalační soubor aplikace typu APK
- gymber_video.mp4 – Demonstrační video představující aplikaci
- android_project – Složka obsahující zdrojové kódy aplikace ve formě Android Studio projektu
- backend_project – Složka obsahující zdrojové kódy serverové části
- latex – Složka obsahující zdrojové kódy technické zprávy pro LaTeX

Příloha B

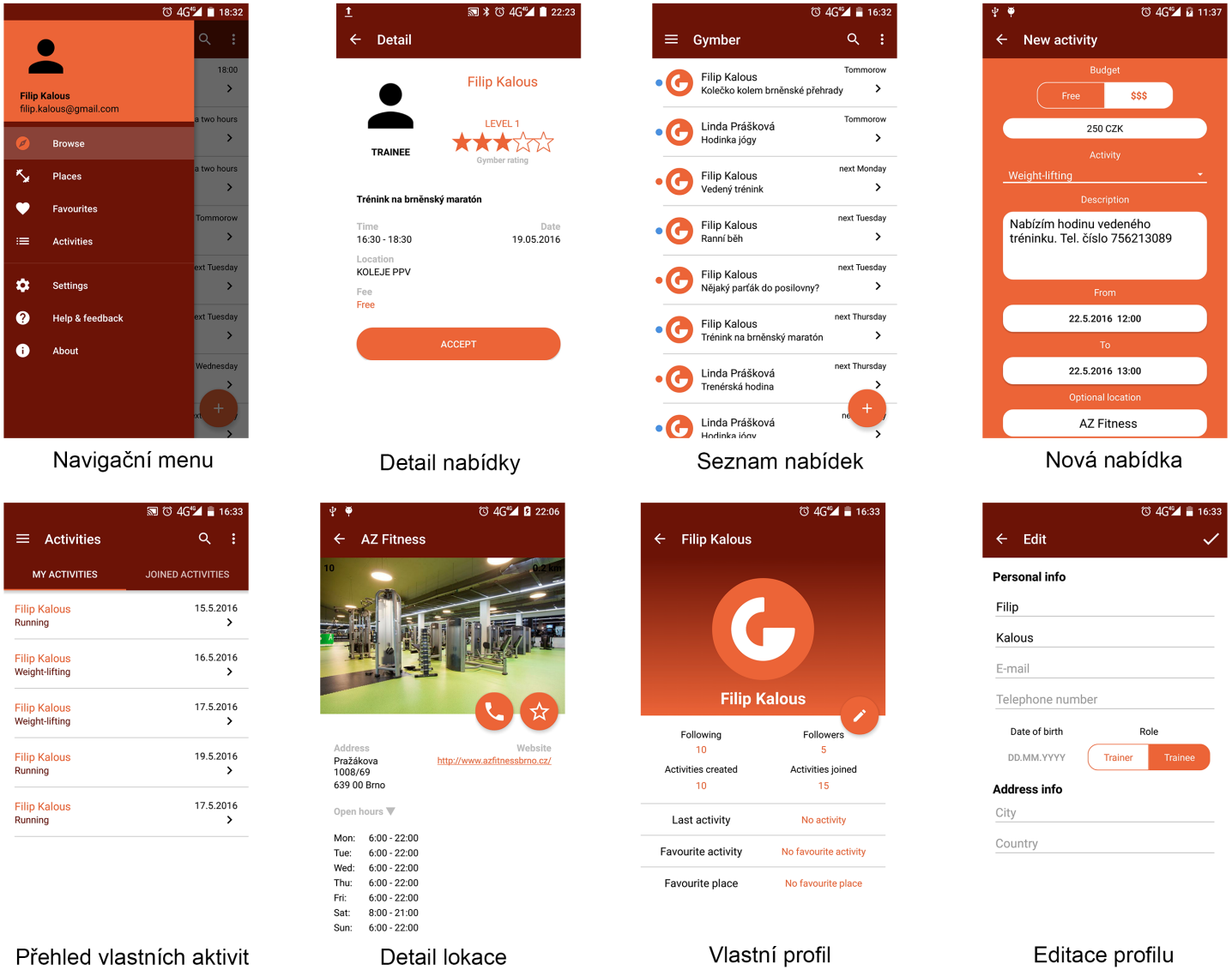
ER Diagram



Obrázek B.1: Entity-Relationship Diagram databáze Gymber

Příloha C

Obrazovky aplikace



Obrázek C.1: Ukázka hlavních obrazovek aplikace