VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

MOBILNÍ APLIKACE PRO SPRÁVU A REZERVACE SPORTOVNÍCH LEKCÍ

MOBILE APP FOR MANAGEMENT AND RESERVATION OF SPORTS LESSONS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — **Bc. TOMÁŠ HYNEK**
AUTHOR

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

BRNO 2019

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav počítačové grafiky a multimédií (UPGM) Akademický rok 2018/2019

Zadání diplomové práce

Student: **Hynek Tomáš, Bc.**
Program: Informační technologie Obor: Počítačová grafika a multimédia
Název: **Mobilní aplikace pro správu a rezervace sportovních lekcí**
Mobile App for Management and Reservation of Sports Lessons
Kategorie: Uživatelská rozhraní

Zadání:

1. Prostudujte problematiku návrhu a tvorby mobilních aplikací pro Android.
2. Zmapujte problematiku organizace individuálních a skupinových sportovních lekcí. Využijte rozhovory s více trenéry a účastníky lekcí. Formulujte závěry vzhledem k řešené aplikaci.
3. Navrhujte a prototypujte prvky uživatelského rozhraní řešené aplikace. Průběžně je testujte na vhodném počtu a skladbě testovacích subjektů a iterativně vylepšujte.
4. Navrhněte a implementujte komunikační infrastrukturu - případné serverové řešení, zasílání zpráv mezi mobilními aplikacemi apod.
5. Implementujte řešenou aplikaci v minimální použitelné podobě.
6. Testujte vyvíjené řešení na uživatelích, formulujte a daty podložte své závěry, iterativně aplikaci vylepšujte.
7. Zhodnoťte dosažené výsledky a navrhněte možnosti pokračování projektu.

Literatura:

- Steve Krug: Don't Make Me Think, Revisited: A Common Sense Approach to Web Usability, ISBN-13: 978-0321965516
- Steve Krug: Rocket Surgery Made Easy: The Do-It-Yourself Guide to Finding and Fixing Usability, ISBN-13: 978-0321657299
- Susan M. Weinschenk: 100 věcí, které by měl každý designér vědět o lidech, Computer Press, Brno 2012
- Android Developers: https://developer.android.com/index.html

Při obhajobě semestrální části projektu je požadováno:

- Body 1 až 3, značné rozpracování bodů 4 až 6.

Podrobné závazné pokyny pro vypracování práce viz http://www.fit.vutbr.cz/info/szz/

Vedoucí práce: **Herout Adam, prof. Ing., Ph.D.**
Vedoucí ústavu: Černocký Jan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2018
Datum odevzdání: 22. května 2019
Datum schválení: 6. listopadu 2018

Abstrakt

Cílem této práce je vytvořit mobilní aplikaci pro operační systém Android, která umožní urychlit a obstarat kompletní proces a správu rezervací sportovních lekcí. V aplikaci vystupují dvě role uživatele. Prvním typem uživatele je trenér, který může nabízet svoje sportovní lekce. Pokud si jeho lekce někdo rezervuje, aplikace mu to patřičně oznámí. Všechny své rezervace lekcí může spravovat a zobrazovat pomocí kalendáře. Naopak uživatel nebo-li sportovec, který provádí rezervace, může lekce vyhledávat dle jejich názvu nebo vzdálenosti místa konání. Aplikace s názvem Fittyy je vytvořena na základě pravidel Material Designu. Dále jsou v aplikaci využity moderní technologie Android Jetpacku, které umožňují ukládání lokálních dat, implementaci architektury MVVM nebo zpracování dat na pozadí. Pro komunikaci mezi trenérem a sportovcem byl využit CMS systém firmy Dactyl Group s.r.o.

Abstract

The goal of this thesis is to create a mobile application for Android that will offer management for reservations of training lessons. There are two user roles in the application. The first one is coach who can offer his lessons to other users. Users then can book this lesson right from the application. Coach can also manage all of his lessons and see his reservations in calendar. The second type of user is an athlete who can search for training lessons by name or place distance and then he can book them. The name of the application is Fittyy and it complies with Material Design rules. It uses advanced technologies like Android Jetpack to store local data, implement MVVM model or process server requests in the background. Communication between coach and athlete was implemented using CMS system made by Dactyl Group s.r.o.

Klíčová slova

rezervační systém, sportovní lekce, mobilní aplikace, Android, Kotlin, MVVM, Sketch, Google, Dagger, Retrofit, Android Jetpack, Zeplin, Material Theming, Material Design

Keywords

reservation system, sport lessons, mobile application, Android, Kotlin, MVVM, Sketch, Google, Dagger, Retrofit, Android Jetpack, Zeplin, Material Theming, Material Design

Citace

HYNEK, Tomáš. *Mobilní aplikace pro správu a rezervace sportovních lekcí.* Brno, 2019. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce prof. Ing. Adam Herout, Ph.D.

Mobilní aplikace pro správu a rezervace sportovních lekcí

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením pana prof. Ing. Adama Herouta, Ph.D. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Tomáš Hynek
20. května 2019

Poděkování

Děkuji vedoucímu mé diplomové práce panu prof. Ing. Adamu Heroutovi, Ph.D. za skvělé vedení a odbornou pomoc při vytváření této práce v rámci pravidelných konzultací. Dále bych chtěl poděkovat Ing. Milanu Doubkovi a firmě Dactyl Group s.r.o. za ochotu a poskytnutí jejich systému CMS.

Obsah

Kapitola 1

Úvod

Čím dál více lidí zjišťuje, že sportovat a stravovat se zdravě je jedna z věcí, která je pro zdraví člověka velmi důležitá. Bez ohledu na to, jaký sport člověk dělá, je s tím téměř vždy spojená rezervace lekcí nebo sportovišť. Většina sportovních působišť pro tyto účely využívá rezervační systém, který jim pomůže jednoduše a pohodlně spravovat rezervace lekcí. Jenže existuje hromada lidí, kteří se živí jako trenéři na volné noze. Pro jejich účely je aktuálně těžké používat jakýkoliv rezervační systém, protože jsou všechny orientované na sportoviště.

Dalším trendem je na všechny věci spojené s každodenním životem používat mobilní telefony a s nimi spojené mobilní aplikace. Pokud si spojíme tyto dva trendy s problémem trenérů spravovat svoje rezervace, vznikne téma, kterým se právě zabírá tato práce.

V rámci této práce byla vyvinuta aplikace pro operační systém Android nazývající se Fittyy. Aplikace se zaměřuje na urychlení a umožnění kompletního procesu a správy rezervace sportovních lekcí. V aplikaci je možné vystupovat jako trenér, který může vytvářet a následně poskytovat sportovní lekce na jakémkoliv sportovním působišti. Pokud si jeho lekci sportovec zarezervuje, trenér musí jeho rezervaci potvrdit nebo je mu potvrzena automaticky serverem. Záleží jak je lekce nastavena. Pro jednodušší správu si může trenér zobrazit svoje rezervace v kalendáři. Zde se pro každou rezervaci zobrazuje informace o času a místě konání lekce. Pokud by kalendář v aplikaci trenérovi nevyhovoval, může si nastavit synchronizaci s Kalendářem Google.

Naopak sportovec, který rezervace provádí, může být buď stálým klientem trenéra anebo člověk, který chce teprve začít sportovat. K tomu potřebuje většinou vyhledat správného trenéra, který danou lekci pro daný sport poskytuje. V tomto případě může sportovec vyhledávat jednotlivé lekce dle jejich názvu nebo si jednoduše zobrazit seznam nejbližších lekcí v okolí. Sportovec má možnost zobrazit detail lekce obsahující podrobnější informace. Zároveň je to jediné místo, odkud si lekci může rezervovat. Více informací o specifikaci problémů trenérů a analýze uživatelů je popsáno v kapitole 2. Zde se také dozvíte, proč byla tato aplikace inspirovaná aplikací firmy Airbnb.

Proto aby tato aplikace mohla být vytvořena, bylo zapotřebí nastudovat nejnovější technologie, které se využívají při návrhu uživatelského prostředí a vyvíjení implementačních částí aplikací operačního systému Android. Tomuto tématu se věnuje kapitola 3.

Velmi důležitou částí této práce, nacházející se v kapitole 4, je návrh aplikace jak z pohledu uživatelského prostředí, tak z pohledu implementace programové části. Nachází se zde popis procesu vytvoření návrhu a jak byly zakomponovány výhody aplikace Airbnb do tohoto procesu. Co se týče návrhu implementační části aplikace, je zde popsáno jak mají fungovat jednotlivé procesy a funkce aplikace, jak bude vypadat architektura aplikace

při použití modelu MVVM a schéma lokální databáze. Poslední část této kapitoly se věnuje návrhem serveru.

Na základě návrhu v předchozí kapitole byla aplikace a potřebný serverový backend implementovány. V kapitole 5 lze nalézt popis, jak byla implementována architektura MVVM, jak lze použít Google Play Services jako je Google Places a Google Maps, jakým způsobem komunikuje aplikace se serverem, jak lze implementovat návrh uživatelského prostředí nebo do jaké míry bylo potřeba doimplementovat systém CMS od firmy Dactyl Group s.r.o.

Aby tato aplikace byla užitečná a použitelná, je potřeba ji řádně otestovat. Tomuto tématu se věnuje kapitola 6. Je zde popsáno jak lze testovat aplikaci již od počátku návrhu až po konec projektu a následnou publikaci. Poslední kapitola 7 je věnována zhodnocením této práce.

Kapitola 2

Specifikace a analýza existujících řešení

Hlavním účelem této aplikace, nesoucí název **Fittyy**, je nabídnout trenérům efektivní správu sportovních lekcí a zároveň zjednodušit proces rezervace lekce ze strany klienta.

2.1 Specifikace problému a jeho řešení

Nápad na tuto aplikaci vznikl již několik let zpátky, kdy jsem často navštěvoval posilovnu a zjistil, že většina trenérů využívá papírový diář a telefon k tom, aby se domluvili s klientem na jejich sportovní lekci. Přišlo mi to docela hloupé, protože domluva s klientem na správném času a dni může být velmi zdlouhavá. Tak mě napadlo, že by bylo super mít pro tyto účely chytrého pomocníka. Z toho nápadu vznikla tato aplikace. Předtím, než jsem se ale do této práci pustil, bylo potřeba prověřit pár věcí.

2.1.1 Komunikace trenéra s klientem a jejich problémy

Prvním průzkumem byl krátký dotazník týkající se trenérů a jejich způsobu komunikace s klienty. Dotazníku se účastnili čtyři trenéři i přesto, že bylo osloveno kolem třiceti trenérů. Nicméně čtyři z nich, kteří poskytují právě sportovní lekce, s radostí odpověděli, že preferují komunikaci telefonicky, emailem nebo osobně. Tři z nich využívají i sociální sítě. Další otázka byla jak si evidují rezervace klientů. Zde všichni odpověděli, že využívají papírový nebo elektronický kalendář.

V dotazníku byly uvedeny i otázky na to s jakými problémy se trenéři často potýkají při nabízení svých služeb. Téměř většina odpověděla, že má častý problém vůbec s komunikací a domluvou rezervace nebo s rušením rezervací lekcí ze strany klienta. Někdo i odpověděl, že má problém s placením nebo dochvilností klienta. Nicméně hlavní problémy zde tkví v procesu objednání lekce a výběr správného časového úseku, který by vyhovoval obou stranám.

Poslední otázka se vztahovala na to, co by trenér chtěl mít v mobilní aplikaci. Všichni čtyři trenéři odpověděli, že chtějí, aby klient měl možnost provést rezervaci v aplikaci a následně mít možnost si zobrazit rezervační kalendář. Tři trenéři požadovali, aby si mohli nastavit své lekce v určitých časových úsecích. Po dvou hlasech získala odpověď týkající se možnosti komunikace s klientem přímo v aplikaci nebo možnost prezentace vlastního profilu s místem, kde trenér působí.

Z těchto odpovědí je možné vidět, že problémy v komunikaci mezi klientem a trenérem jsou značné. Většina by v aplikaci chtěla, aby si klient sám rezervoval ve volný časových úsecích trenéra jeho lekci. Naproti tomu trenér bude mít možnost jednoduše spravovat své lekce.

Chvíli potom byla provedena podrobnější studie s dvěma trenéry. Při studii s prvním z trenérů bylo zjištěno, že dříve využíval Google kalendář, který byl veřejný pro všechny jeho klienty. Klienti se mohli přihlásit pod jedním emailem a heslem, kde pak mohli individuálně přidávat rezervace lekcí dle libosti. Bohužel se tento systém příliš neosvědčil, protože po delší době používání, se klienti začali navzájem nevědomky přepisovat v případě, kdy bylo přihlášených více uživatelů přes jeden sdílený email. Trenér tak našel jinou službu nazývající se SuperSaaS (více v kapitole 2.3). Druhý trenér zmínil, že také využívá Google kalendář. Jenom si všechny rezervace spravuje sám ve svém neveřejném kalendáři. Což mu v tomto případě přidává více práce a nutnost stále komunikovat s klienty a řešit termíny lekcí. Ovšem sám řekl, že by uvítal mobilní aplikaci, která by mu ušetřila práci při vytváření a správě rezervací.

2.1.2 Průzkum rezervace sportovní lekce

Následně se nabízela otázka zda neudělat spíše webovou stránku než nativní mobilní aplikaci. Odpověď na tuto otázku byla zodpovězena díky druhému dotazníku, který nebyl nijak cílově zaměřen na určitý okruh uživatelů ale spíše cílil na potencionální uživatele aplikace (o uživatelích více v 2.2). Zde byla položena pouze jedna jednoduchá otázka, která se ptala na to, zda by člověk využil webovou nebo mobilní aplikaci k tomu, aby si objednal sportovní lekci. Na tuto otázku odpovědělo celkem šedesát dva lidí, kdy přesně 56,5% odpovědí (35 hlasů) připadlo mobilní aplikaci. Díky tomu se táto práce zabývá vývojem mobilní aplikace.

2.1.3 Řešení problémů

Díky analýze informací získaných od trenérů a potencionálních klientů lze říci, že by měla z pohledu trenéra aplikace nabízet jednoduchou efektivní správu rezervací a správu nabízených služeb nebo sportovních lekcí. Dále by měla trenérům umožnit prezentovat svoje lekce na svém profilu, který by mohl sloužit i jako portfólio služeb. Díky tomu by potencionální klient měl možnost zjistit kdo a co nabízí na jakém sportovním působišti.

Z pohledu klienta bude pak aplikace nabízet možnost vyhledání sportovních lekcí na základě sportovní kategorie, místa působiště nebo trenéra. Následně klient může rezervovat lekci a čekat zda ji trenér schválí nebo zamítne. V tomto případě by tedy rezervace lekcí byla ze strany klientů bez přítomnosti nebo potřeby trenéra. Klient může vyhledat požadovanou lekci a pokud by ji chtěl rezervovat, zobrazí se mu její dostupnost pro požadovaný den a čas. Po provedení rezervace bude vyžadováno její schválení, pokud trenér nebude mít nastavené u lekce automatické schvalování rezervace.

Celkově by měla pro všechny uživatele aplikace umožňovat zobrazení kalendáře rezervací lekcí, možnost rušení rezervací lekcí či zobrazení detailu lekce nebo detailu rezervace lekce.

2.2 Studium uživatelů

V aplikaci vystupují dvě hlavní role, které v rámci cílové skupiny uživatelů liší. První role zahrnuje lidi, kteří jsou označováni jako trenéři a věnují se vedením sportovních lekcí. Popis cílové skupiny zahrnující trenéry je následující:

- lidé v rozmezí věku 20 až 60 let,
- jejich zájmy i práce se týkají sportovních aktivit,
- řeší podobné problémy při práci,
- většinou žijí nebo se snaží žít zdravým způsobem života,
- měly by být zkušení a vzdělaní v daném sportovním odvětví, kterému se věnují,
- v některých případech musí vlastnit platný certifikát, který je opravňuje k trénování v daném sportovním odvětví,
- většina se ráda prezentuje na sociálních sítí.

Zde bych ještě poznamenal, že i když je aplikace primárně cílená na trenéry, v roli trenéra může být teoreticky i samotné místo, tedy např. posilovna, která tuto aplikaci využije jako svůj rezervační systém. Ale na toto se v této práci nebere ohled a aplikace je zaměřena hlavně na osoby, které mají problém se správou rezervací. V případě posiloven jsem se většinou setkal s tím, že mají vlastní na zakázku vyrobený systém pro rezervaci lekcí.

Druhá role zahrnuje mnohem větší část skupiny lidí. V této práci budou zmiňování jako sportovci nebo klienti trenérů. Hlavním kritériem, pro začlenění člověka do této cílové skupiny, je potřeba objednat si sportovní lekci u trenéra. Podrobněji tuto skupinu lidí lze popsat jako:

- lidé v rozmezí věku 15 až 80 let,
- lidé, kteří chtějí zhubnout, mít dobrou kondici nebo sportovat pro zábavu,
- řeší podobné nepříjemnosti při objednávání lekcí,
- žijí nebo by chtěli žít zdravým způsobem života,
- můžou mít ale většinou mají málo zkušeností s danou sportovní aktivitou,
- sledují známé sportovce nebo trenéry na sociálních sítích pro inspiraci a motivaci.

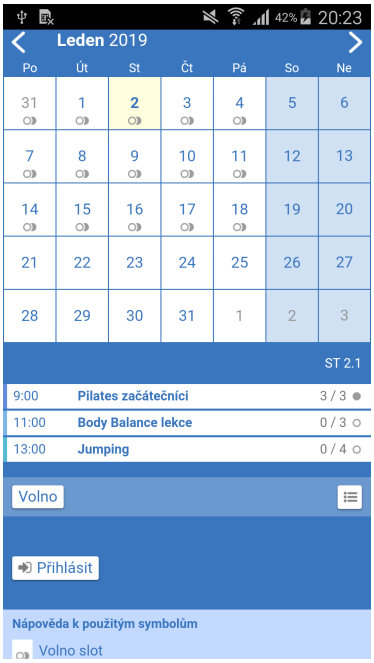
2.3 Analýza existujících řešení

Následující odstavce se zaobírají analýzou existujících řešení. Již na začátku bych poznamenal, že se mi podařilo najít pouze dvě podobné existující služby, která měly i mobilní aplikaci. Přesto věřím, že existuje vícero rezervačních systému pro sportovní lekce, jen se mi je bohužel nepodařilo najít nebo jsou těžko dohledatelné.

SuperSaaS

Jako první službu bych rozebral rezervační systém nazývající se SuperSaas[1]. Tuto službu jsem již dříve zmínil v kapitole 2.1), protože jeden z trenérů ji začal nedávno využívat. Rezervační systém funguje pouze přes webové rozhraní, kde nabízí přizpůsobitelnost dle jeho využití. Je to všeobecný rezervační systém, který si předplatitel nastaví dle svého přání.

[1] https://www.supersaas.cz/



Obrázek 2.1: Ukázka rezervačního systému SuperSaaS ve webovém prohlížeči na mobilním telefonu.

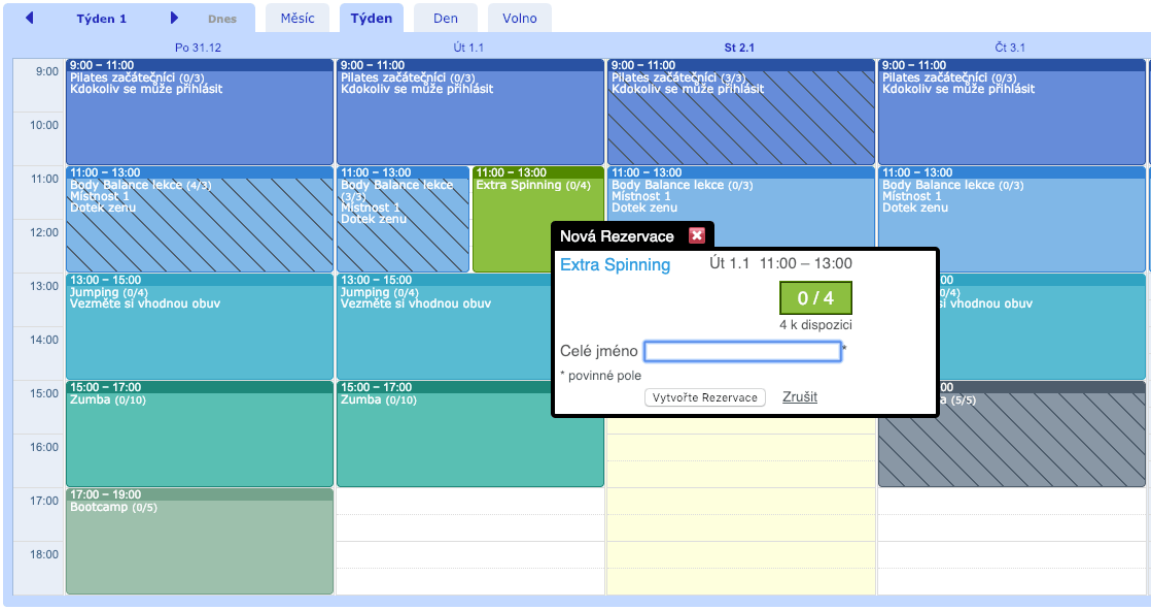
Výhodou tohoto systému je přizpůsobitelnost, zasílání emailových nebo textových zpráv, upozornění, integrované platby, integrace do vlastního webu nebo Facebooku, synchronizace s Google kalendářem, správa uživatelů, bezpečnost a spolehlivost systému. Naopak jako nevýhodu vidím, že systém není možné jednoduše a intuitivně ovládat nebo spravovat přes mobilní zařízení, grafika a uživatelské prostředí systému je mírně zastaralé a hůře ovladatelné. Celkově je systém použitelní spíše pro fitness centra, kde mají stolní počítače, přes které lze snadno a rychle ovládat systém. Pro jednotlivé trenéry a jejich lekce je tento proces obtížnější, i přesto, že webový systém je optimalizovaný pro mobilní zařízení. Na následujících obrázcích 2.1 a 2.2 je možné vidět ukázku rezervačního systému pro fitness klub.

Co se týče ceny, tak firma nabízí i verzi zdarma, která je ovšem omezena na maximálně padesát budoucích rezervací a padesát registrovaných uživatelů. Dále se jednotlivé placené varianty, začínající od 150 Kč za měsíc, liší maximálním počtem budoucích rezervací a počtem proběhlých rezervací uložených v historii.

Smarter Lessons

Další služba Smarter Lessons[2] má zase pouze rezervační systém fungující ve webovém rozhraní prohlížeče. Už na první pohled je uživatelské prostředí rezervačního systému velmi zastaralá. Těžko hledat výhody této služby, protože se pravděpodobně její vývoj zastavil někdy kolem roku 2015. Nicméně rezervační systém zajišťoval vícero kalendářů v jednom, proces plateb, posílání emailů, slevy, analýzy a statistiky.

[2] https://www.smarterlessons.com/business/index.html



Obrázek 2.2: Ukázka rezervačního systému SuperSaaS ve webovém prohlížeči počítače.

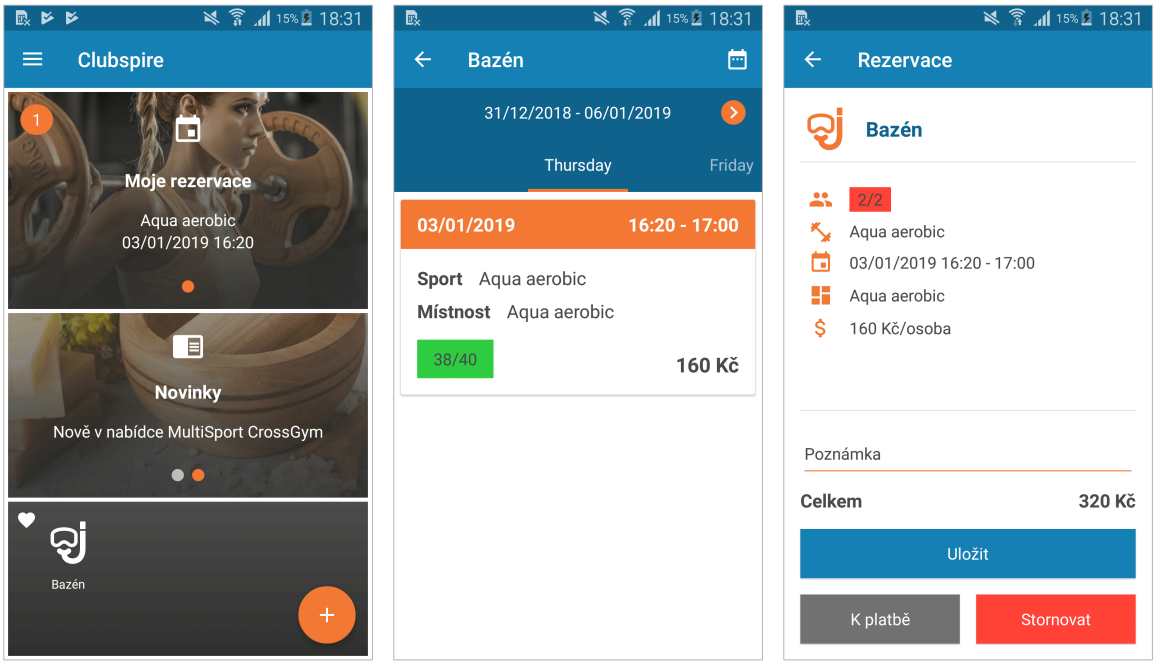
Square

Firma Square[3] má systém, který pokrývá velmi rozsáhlou potřebu všech menších nebo středně velikých podniků. Jejich systém nabízí rozsáhlý komplexní nabídku služeb začínající od správy prodávaných věcí až po rozsáhlou správu rezervací. Zde zmíním pouze informace, které se týkají této práce. Square nabízí rezervační systém, kde je možné spravovat kalendář rezervací, provádět bezhotovostní transakce nebo spravovat klienty. Také umožňuje podporu práce v týmů. K tomuto systému firma nabízí vlastní platební a obslužný terminály, které se dají snadno nainstalovat na recepci fitness centra. Výhodou této služby je moderní aplikace pro mobilní telefony nebo tablety. Ovšem znovu jako v předešlém případě je tato služba orientována na podniky a firmy, které mají recepce.

Clubspire

Nejblíže podobné vyvíjené aplikaci je služba Clubspire[4]. Tento komplexní rezervační systém je přímo zaměřen na správu veškeré aktivity ve fitness a dalších sportovních či relaxačních center. Tento systém nabízí rezervační systém, pernamentky, deposit, členství, stornopoplatky, emaily, poukázky, řízení vstupů, on-line rezervace nebo pokladní systém. Tento systém používá v České republice přes 200 sportovních míst. Minimální cena používání je 490 Kč za měsíc. Stejně jako služba Square, je tento systém znovu zaměřen na dané sportovní místa a ne na samostatné trenéry. Velkou výhodou je Clubsire mobilní aplikace[5], která je individuálně zveřejněna v mobilních obchodech pro jednotlivé zákazníky této služby. Jeden ze zákazníků s mobilní aplikací je Fit&Co [6].

[3] https://squareup.com/us/en/software/appointments
[4] https://www.clubspire.cz/
[5] https://play.google.com/store/apps/details?id=com.clubspire.android
[6] https://play.google.com/store/apps/details?id=com.clubspire.android.fitco



Obrázek 2.3: Obrazovky z ukázkové aplikace pro Android systému ClubSpire, **vlevo:** nadcházející rezervace a novinky na úvodní obrazovce, **uprostřed:** nabídka lekcí pro danou aktivitu, **vpravo:** rezervace lekce s možností platby.

Na následujících obrázcích 2.3 je možné vidět, že je aplikace vytvořena v Material Designu, která je doporučována firmou Google[7] (více viz. 3.1).

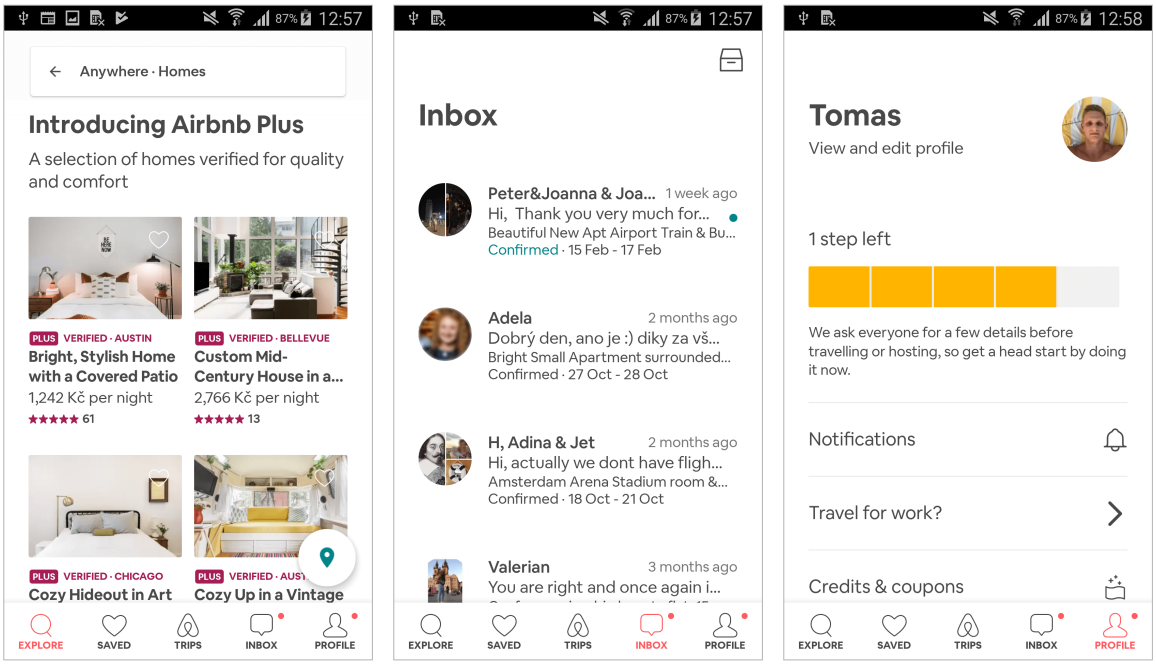
Airbnb

Jako poslední existující řešení jsem analyzoval službu Airbnb[8] a její aplikaci, která umožňuje nabízet svoje prostory pro bydlení k pronájmu na určitý počet dní. Airbnb z tohoto pohledu přímo nesouvisí s rezervačním systém pro sportovní lekce, nicméně jsem si tuto službu vybral hned pro několik ohromujících faktů. Službu Airbnb využilo již přes 400 miliónů uživatelů od roku 2008, kdy rezervační systém oficiálně odstartoval. Průměrně 2 milióny uživatelů využije denně službu k tomu, aby našli nocleh. Služba také nabízí přes 15 tisíc různorodých zážitků ve více jak tisíci městech. Všechny tyto a jiné informace jsou dostupné zde [3].

Výhodou této služby je, že proces nabízení bydlení a rezervace pronájmu je velmi jednoduchý. Jelikož jsem osobně několikrát službu využil, vím, že vše je možné udělat v pár krocích rovnou i se zaplacením a to všechno přes mobilní aplikaci[9]. Služba se dostala velmi veliké oblíbenosti z důvodu levnějších cen, než klasické hotely nebo apartmány.

Mobilní aplikace Airbnb pro systém Android má přes neuvěřitelných 50 miliónů stažení na Google Play[10] s průměrným hodnocením 4,5 z 5 hvězdiček od 492 tisíc uživatelů.

[7] https://www.google.com/intl/cs_cz/about/
[8] https://www.airbnb.cz/
[9] https://play.google.com/store/apps/details?id=com.airbnb.android
[10] Google Play je oficiální distribuční síť aplikací a her od Google pro operační systém Android.



Obrázek 2.4: Ukázka obrazovek z aplikace Airbnb pro platformu Android, **vlevo:** vyhledávání ubytování dle místa, termínu, kapacity osob a typu ubytování, **uprostřed:** seznam konverzací sloužící pro komunikaci s poskytovateli ubytování, **vpravo:** profil uživatele slouží pro navigaci do ostatních částí aplikace.

Z pohledu využitých technologií při vývoji aplikace pro platformu Android můžu říct, že aplikace je velmi specifická. Původně vývojáři využívali React Native[11], který se jim ovšem v poslední době neosvědčil [6]. Proto se rozhodli vyvíjet vlastní technologie, které budou více na míru jejich řešení. Aktuálně využívají vlastní technologii Server-Driven Rendering, kdy většina obsahu a grafických komponent je vygenerována na serveru a poslána v JSON formátu do aplikace. Dokonce si vytvořili vlastní vizuální jazyk DLS (Design Language System) [4], který jím ulehčuje práci při návrhu uživatelského prostředí a zlepšuje jejich přehlednost.

Na řadu přichází objasnění důvodu, proč jsem si tuto aplikaci vybral pro analýzu do této práce. Hlavní důvod je ten, že aplikaci používá každý den milióny uživatelů, kteří jsou dle hodnocení s aplikací velmi spokojeni. A protože v aplikaci jsou funkce jako nabízení nebo rezervace ubytování, potvrzování, rušení a zamítání rezervací, prezentace profilu uživatele a vyhledávání ubytování, je tímto velmi podobná aplikaci Fittyy, i když její zaměření není na rezervace sportovních lekcí. Jak vypadá aplikace pro platformu Android je možné vidět na následujících obrázcích 2.4, 2.5 a 2.6. Zde bych poukázal na moderní design, který si Airbnb udělalo kompletně na míru. Za zmínku stojí převážně vlastní ikonky, vlastní komponenty [5] a správně vybraný kontrast barev, který utváří aplikaci velmi čistou a svěží. Z důsledku toho jsem se rozhodl velmi inspirovat designem aplikace Airbnb, protože přeci jen ví lépe, jak by měla aplikace vypadat, co přesně a na jakých obrazovkách uživatel chce nebo potřebuje.

[11] React Native je framework od Facebooku umožňující vyvíjet jednu aplikaci, která může fungovat na platformě Android i iOS [10].

Vyhledávání jednotlivých ubytování (obrázek 2.4) je možné různě filtrovat tak, aby uživatel měl co nejpřesnější výsledky. Pomocí filtru lze vybrat kde se přesně má ubytování nacházet, které dny chce uživatel přespat, pro kolik lidí by ubytování mělo být, v jakém cenovém rozsahu, jaký typ ubytování (sdílený pokoj ve sdíleném bytě, soukromý pokoj ve sdíleném bytě nebo soukromý byt, apartmán či dům), zda ubytování podporuje Instant Booking[12], zda je vhodný byt pro rodiny nebo pro práci, nebo si uživatel může přesně vyfiltrovat co by měl byt obsahovat (vybavení bytu, počet pokojů nebo koupelen, atp.).

V aplikaci je také chat (obrázek 2.4), přes který může komunikovat klient s majitelem bytu a domluvit se na předání klíčů, kdy přesně klient dorazí a kdy bude odcházet, nebo je zde možné řešit problémy nebo dotazy související s ubytováním. Zařazení chatu do aplikace dává smysl, protože majitel bytu není nucen dávat svoje telefonní číslo klientovi a můžou se spolu spojit jednoduše skrz aplikaci. Dokonce v chatu nelze posílat telefonní čísla, emaily a podobné kontaktní údaje. Firma Airbnb vždy tyto údaje cenzuruje. Nicméně jsem se rozhodl aktuálně chat do aplikace Fittyy nezařadit a to hlavně z technických a časových důvodů, protože trenéři dle dotazníku z analýzy v kapitole 2.1.1 dali přednost jiným funkcím resp. více jich požadovalo jiné funkce aplikace.

Vlastní profil uživatele (obrázek 2.4) je trochu chaotický, protože jsou zde všechny ostatní věci, které již nebylo kam dát. Uživatel má možnost odsud vidět svůj profil jak jej vidí poskytovatelé ubytování, editovat profil, zobrazit si notifikace, pozvat přátelé, nabídnout vlastní ubytování k pronájmu, spravovat nastavení aplikace nebo zobrazit nápovědu. Zde jsem prozradil, že pokud je uživatel v roli nabízejícího nebo hledajícího, používá se v obou případech stejná aplikace a stejný účet. Dle zdroje [2] jsem zjistil, že se při nabízení ubytování změní struktura spodního menu, kde přibudou tři důležité položky: kalendář, seznam nabízených ubytování a statistiky. Což je pro inspiraci k navrhnutí aplikace Fitty z pohledu trenéra docela důležité, protože jak jsem jíž dříve poznamenal v 2.1, trenéři chtějí mít v aplikaci kalendář s rezervacemi. K tomu logicky chce mít každý přehled o jeho nabízených službách.

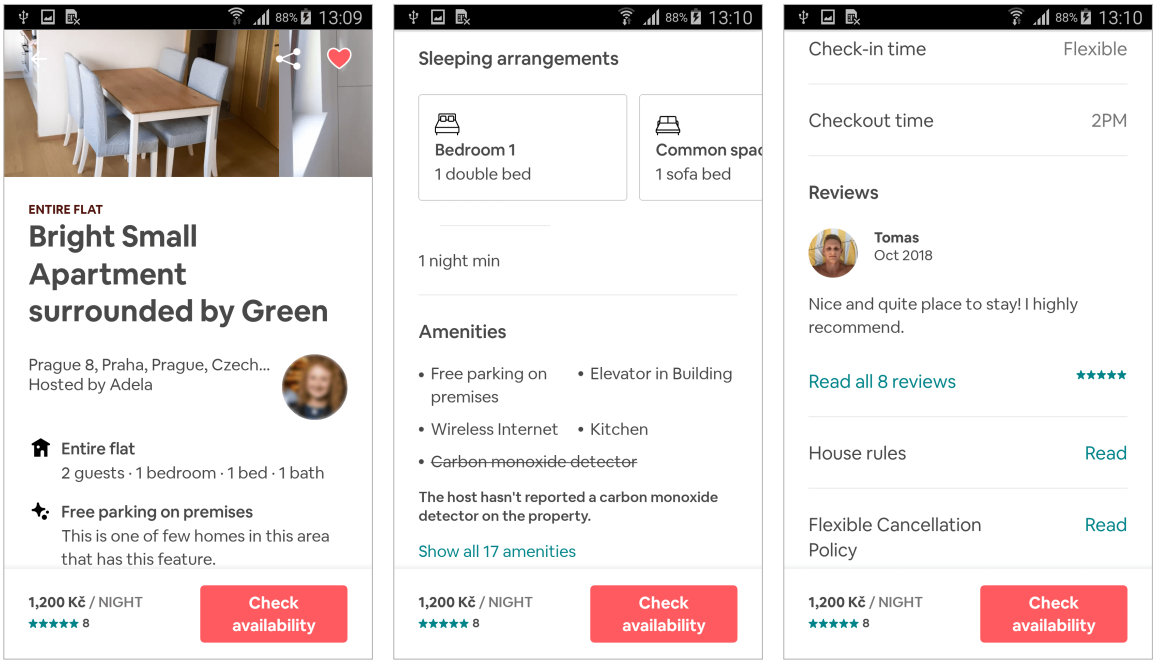
Uživatel po vyhledání ubytování má možnost zobrazit detailní profil místa (obrázek 2.5). Zde se nachází podrobné informace o ubytování jako je např. informace o tom, kde se nachází, kolik na noc stojí, jakou kapacitu osob má, kdo je poskytovatelem, jaké vybavení obsahuje, jaké jsou pravidla a požadavky na bydlení nebo hodnocení od klientů. Občas je tam až tolik informací, že se v tom člověk ztratí. Nicméně všechny informace, které se týkají ubytování, tam být zmíněny musejí. Jinak by klient napsal zápornou recenzi o tom, že poskytovatel ohledně ubytování něco zatajil.

Vzhledem k tomu, že tu bohužel nemůžu ukázat proces rezervace ubytování, protože bych musel nějaké ubytování reálně rezervovat a zaplatit, uvádím zde alespoň detail proběhlé rezervace 2.6, která je ve schváleném stavu. Rezervace může mít čtyři stavy:

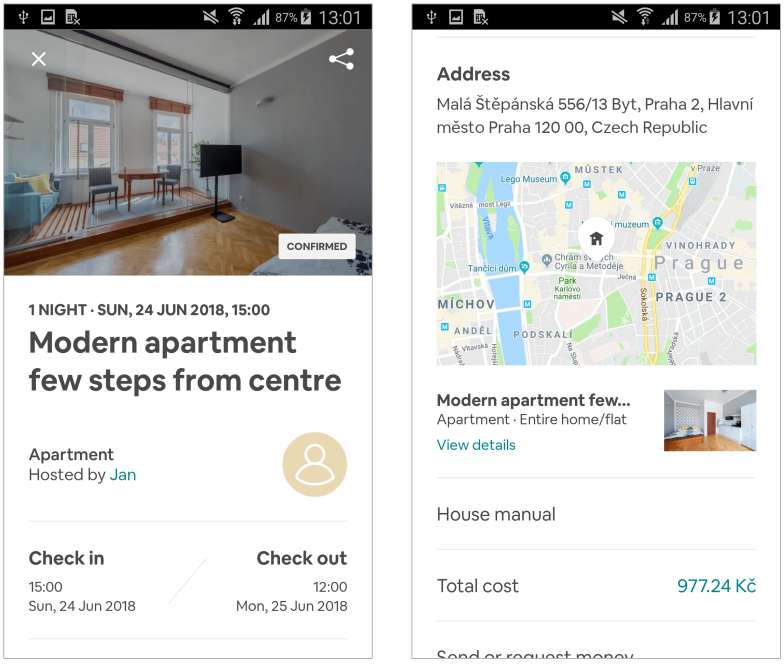
- nevyřízená – první stav po vytvořené rezervaci,
- schválená – poskytovatel schválil rezervaci ubytování,
- zamítnutá – poskytovatel zamítl rezervaci ubytování,
- zrušená – klient nebo poskytovatel zrušil rezervaci.

Stav zamítnutá rezervace nastane po stavu nevyřízená, když poskytovatel zamítne rezervaci. Jak klient tak poskytovatel může vyvolat stav zrušená, a to když zruší rezervaci, která byla

[12] Instant Booking představuje automatické schválení rezervace bytu bez potřeby majitele/poskytovatele.
[13] Check-in je výraz značící čas, kdy se musí zákazník dostavit k ubytování
[14] Check-out vyznačuje čas, kdy nejpozději musí zákazník opustit ubytování.



Obrázek 2.5: Zobrazení profilu ubytování v aplikaci Airbnb pro platformu Android, **vlevo:** prezentace fotek ubytování, informace o majiteli a max. obsazenosti, **uprostřed:** základní informace o apartmánu a o minimálním počtu nocí, **vpravo:** informace o check-in[13]a check-out[14]čase, zobrazení recenzí a domovních pravidel.



Obrázek 2.6: Detail schválené rezervace ubytování v aplikaci Airbnb pro platformu Android, **vlevo:** obrazovka s informacemi o délce a datu ubytování, check-in a check-out čase, **vpravo:** informace o poloze apartmánu a zaplacené částce za ubytování.

ve stavu schválená. Tímto modelem a procesem rezervace se také inspiruji, protože mi přijde srozumitelný a dává potřebnou informaci o stavu jak klientovi, tak poskytovateli ubytování.

Nakonec bych chtěl zde ještě říct nějaké poznatky, které jsem zjistil při používání aplikace. Občas se mi stávalo, že na obrazovce bylo příliš informací nebo byly obrázky a informace příliš veliké, a tak jsem musel zbytečně hodně posouvat informacemi. Další věc, co se mi stávala, byla ztráta orientace v aplikaci. Například obrazovka detailu rezervace 2.6 a detail profilu ubytování 2.5 při stejném obsahu jsou velmi totožné, a tak jsem se přestal orientovat, kde vlastně jsem. Nebo se mi stávalo, že jsem chtěl zobrazit informace o ubytování ale intuitivně jsem nevěděl jak. Ovšem po delším používání všechny zmíněné problémy zmizely. Dle mého názoru je aplikace velmi povedená, protože zobrazit co nejvíce informací na obrazovce mobilního zařízení není vůbec jednoduché.

Kapitola 3

Návrh a vývoj mobilních aplikací pro Android

V následujících kapitolách představím nejdůležitější informace týkající studia použitých technologií při vývoji aplikace Fittyy.

3.1 Mobilní platforma Android

3.1.1 Základní informace

Operační systém Android je open-source platforma na bázi Linuxu určená hlavně pro mobilní zařízení [14]. Její vývoj, pod hlavičkou konsorcia firem z Open Handset Alliance, vede americká firma Google. Díky Googlu se tento operační systém rozšířil do celého světa. Za velkým počtem technologií, které se využívají při vývoji aplikace pro platformu Android, stojí právě Google.

Největší výhodou a zároveň nevýhodou této toho operačního systému je jeho otevřenost, ať už ze strany výrobců nebo uživatelů. Nejčastěji právě výrobci upravují konfigurace, widgety nebo i firmwary. Důsledkem těchto operací je problematičtější vývoj aplikací, protože se na každém telefonu pak chovají jinak. Z vlastní zkušenosti mohu říci, že největší problém je u čínských výrobců.

Další značně velkou nevýhodou je, že pokud vyjde nová verze nebo aktualizace operačního systému, výrobci by měli sami provést aktualizaci svých telefonů. Kvůli tomu telefony starší jak dva roky většinou již nezískávají nové aktualizace systému. Na rozdíl od konkurenčního systému iOS, který má vlastní zařízení, vlastní operační systém a podporuje roky staré zařízení. Proto se při vývoji pro platformu Android musí počítat s hodně staršími verzemi a značně to omezuje vývoj aplikace resp. použití novějších technologií.

3.1.2 Verze Androidu

Každý rok v květnu se pořádá konference vývojářu Google I/O[1], která představuje novou verzi Androidu a s tím spojené nové technologie. V roce 2018 byla představena poslední verze s názvem Android Pie. Aktuálně je dostupný pouze na nejnovějších pár zařízeních [1]. Dokonce i v oficiálním přehledu použití není zahrnut [12]. Každopádně při počátečním vývoji Android aplikací musí být specifikována minimální podpora verze. Pro tuto práci jsem zvolil podporovat minimálně verzi Android Lollipop 5.0 a to z toho důvodu, že tato verze

[1] https://events.google.com/io/

přinesla podporu Material Designu (více viz. 3.2.1 a s tím spojené ulehčení implementace nového designu obrazovek. Díky tomu si aplikaci Fitty bude moci aktuálně nainstalovat kolem 85 % uživatelů. Ovšem toto číslo se každým měsícem mění a předpokládám, že na konci této práce se bude hodnota pohybovat okolo 90 %.

3.2 Návrh mobilních aplikací pro Android

Při návrhu mobilních aplikací je doporučeno dle studie Jakoba Nielsena [18] využití iterativního designu. Tato metoda je velmi efektivní pro otestování designu za účelem dosáhnutí co největší použitelnosti aplikace. Iterativní návrh je založen na minimálně dvou až třech iteracích, které zahrnují jeden až dva nové návrhy designu. Pro každou verzi se provádí základní vyhodnocení použitelnosti. Tímto se dosáhne mnohem lepšího výsledného designu aplikace. K tomuto procesu byl použit nástroj od firmy InVision[2].

3.2.1 Material Design

Material Design je vizuální jazyk, který představuje obecné zásady dobrého designu s inovativními technologiemi [13]. Tenty jazyk byl představen roku 2014 firmou Google za účelem sjednotit a vylepšit uživatelské rozhraní a zkušenosti s používáním Android aplikací. Dnes se tímto směrem řídí většina designérů i při navrhování aplikací pro iOS. Proto jsem se tohoto designu držel i já při návrhu aplikace Fittyy.

Základ Material Designu nebo-li Material Foundation je složen z následujících sekcí:

- základní informace o Material Designu
- **prostředí** – zabývá se vlastnostmi, které se odrážejí ve využití povrchů, hloubky a stínů,
- **uspořádání** – definuje uspořádání komponent, hustotu pixelů na jednotlivých obrazovkách, responzivní design nebo mezery mezi prvky.
- **navigace** – definuje jak a pomocí čeho lze navigovat mezi jednotlivými obrazovkami,
- **barvy** – stanovuje v jakém kontrastu a jaké barvy lze využít pro obrazovky,
- **typografie** – doporučuje jaký font a styl písma použít v určitém případě,
- **ikonografie** – udává velikost, rozvržení a vlastnosti ikonek
- **tvary** – informuje o možných tvarech pro tlačítka, obrázky, karty, apod.,
- **pohyb** – zahrnuje přechody a animace v aplikaci,
- **interakce** – zabývá se způsobem jakým by se uživatel měl dozvědět o své interakci s aplikací,
- **komunikace** – kompletně zahrnuje způsob prezentace informací, chyb nebo různých stavů uživateli.

[2] https://www.invisionapp.com/

Všechny tyto informace slouží jako doporučení, kterým by se měli designéři řídit při návrhu uživatelského prostředí aplikace. Obecně se vizuální jazyk používá pro Android, iOS ale i web.

Proto aby každá aplikace nevypadala úplně stejně, byla vytvořena dokumentace nazývající se Material Guidelines. Ta pomáhá designérům nastylovat a vytvořit vlastní komponenty tak, aby stále splňovali pravidla Material Design. Pro každou komponentu systému Android je zde přesně definováno: jak by měla vypadat, co by měla obsahovat, jak by se měla chovat, jaké styly písma nebo které tvary a rozvržení prvků v komponentě použít.

Od doby, kdy byl Material Design představen, bylo díky němu vytvořeno nespočet návrhů a aplikací. Za ty roky ale Google zjistil, že navrhnout přesný design, který by seděl na daný produkt je docela zdlouhavý proces a ještě složitější je implementace těchto komponent. Proto roku 2018 Google přidal do Material Guidelines tzv. Material Theming[3]. Díky němu je možné aplikaci navrhnout a implementovat přesně do stylu, který by odrážel danou službu nebo značku, kterou má aplikace reprezentovat. Pár příkladů využití Material Theming lze nalázet v Material Studies[4].

Pokud tedy designér dodrží pravidla Material Degsignu a Guidelines, může si být jistý, že návrh uživatelského prostředí aplikace bude splňovat správnou uživatelskou zkušenost nebo-li User Experience (UX) a zároveň bude reálné jej implementovat programátorem. Nevýhodou Material Deisgn je fakt, že zpětná kompatibilita se staršími verzemi Androidu není příliš dobrá. Většina komponent je k dispozici, ale například všechny základní stíny, které jsou v dokumentaci popsány, nelze ve verze nižší jak Android 5.0 Lollipop zobrazit a je nutné je speciálně dodělat. Takovýchto drobných omezeních je více a celkově to použití komponent ztěžuje.

3.2.2 Tvorba návrhu mobilních aplikací

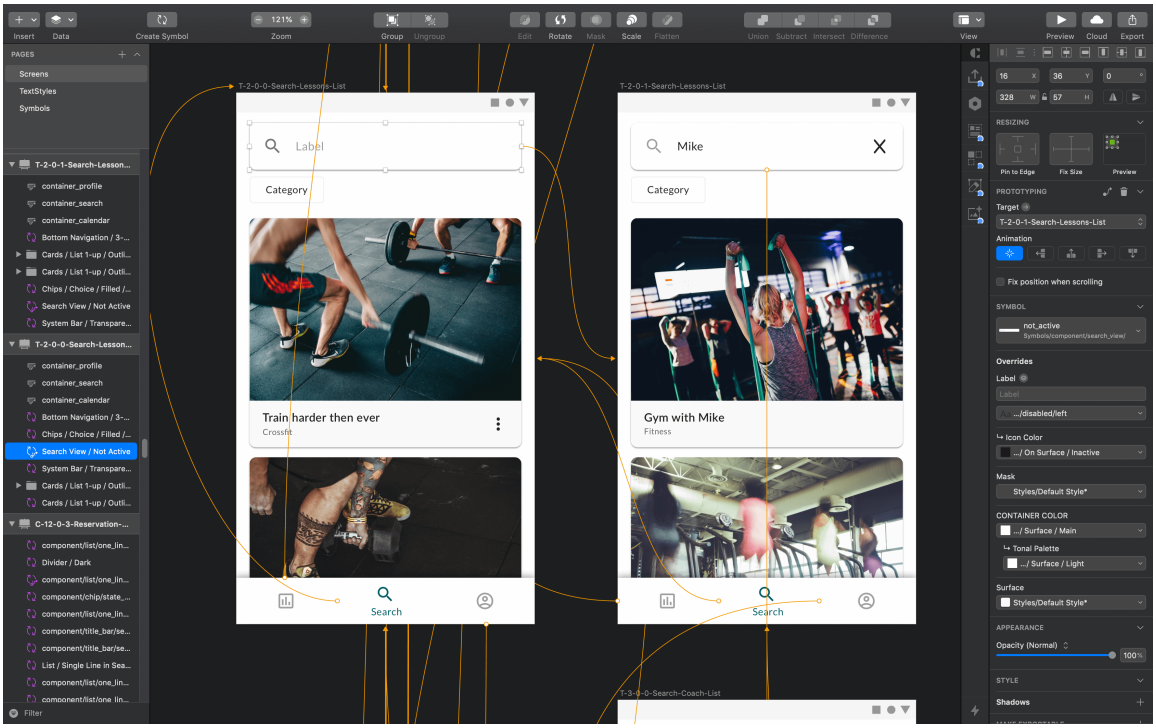
Při tvorbě návrhu mobilních aplikací musí designér brát ohled na pravidla Material Designu a možnosti, které platforma Android umožňuje implementovat. To není vůbec jednoduché, protože mnoho designéru navrhují věci, které programátor nemůže implementovat nebo v lepším případě mu to zabere hodně času. Proto musí každý designér najít tu správnou hranici, která udává možnosti realizace. Nicméně pokud se bude držet pravidel Material Design, neměl by být v ničem problém (až na vyjímky 3.3.3).

Proces tvorby návrhu mobilní aplikace začíná při prvních náčrtech tzv. mockupech aplikace. Vytvoření mockupu spočívá v nakreslení obrysů jednotlivých komponent bez jakéhokoliv vyznačujícího obsahu. Slouží tedy spíše k první demonstraci, testování a hodnocení návrhu.

Sketch

Dalším krokem je již tvorba návrhu mobilní aplikace ve speciálním nástroji. Existuje mnoho nástrojů na vytváření uživatelského prostředí, ovšem zde zmíním pouze ten, který je aktuálně nejrozšířenější a který jsem sám použil. Tento nástroj resp. aplikace se nazývá Sketch[5]. Sketch vznikl roku 2008 překvapivě v Nizozemí. Aplikace je hlavně zaměřena na vytváření návrhů mobilních aplikací a webových stránek. Velkou výhodou je možnost vytváření jednotlivých komponent nebo-li tzv. symbolů, které lze libovolně v návrhu používat. Uživatel tedy nemusí kopírovat jednotlivé komponenty ale pouze vybere ze seznamu symbolů.

[3] https://material.io/design/material-theming/overview.html#material-theming
[4] https://material.io/design/material-studies/about-our-material-studies.html#
[5] https://www.sketchapp.com/



Obrázek 3.1: Ukázka tvorby grafického návrhu Fittyy v aplikaci Sketch.

Další výhodou je možnost vytvoření stylů písma a barev. Z mé vlastní zkušenosti, kdy jsem používál konkurenční aplikace jako je Adobe Photoshop nebo Adobe Illustrator, je aplikace Sketch plynulejší a méně náročná na hardware počítače. Tvorba designu je poměrně zdlouhavá činnost a tedy práce v rychlejším a plynulejším prostředí je hodně znát.

Následně uvedu pár hlavních funkcí aplikace, které mě přijdou důležité a užitečné. Uživatel aplikace má možnost kompletně vytvořit a navrhnout grafiku vektorově a následně ji exportovat do pdf nebo do jiné služby. Velkou výhodou je možnost vytváření vlastních symbolů resp. komponent, které lze používat na více místech. Následně pokud proběhne změna komponenty v symbolech, změna se projeví na všech místech, kde byla komponenta použita. Uživatel si může vytvořit vlastní barevnou paletu ze které můžou vycházet komponenty. Při změně barvy v paletě se změna projeví plošně na všech místech, kde byla použita. Podobně může uživatel vytvořit styly písma, které pak kdekoliv v návrhu může využít. Do aplikace může uživatel libovolně zkopírovat vektorový objekt, v mém případě to byla často ikona. Použití již samotné aplikace Sketch ulehčuje každodenně práci každému uživatel. Jenže uživatel má navíc možnost si do aplikace doinstalovat doplňky. Dovolím si zde pár doplňků a jejich částí, které jsem používal, zmínit:

- **CrafManager Data** – umožňuje duplikovat obsah anebo náhodně naplnit komponenty informacemi jako např. jména, ulice, města, data, články, atd.,
- **CraftManager Sync** – nahrává jednotlivé návrhy do služby InVision,
- **Unsplasher** – dle nastaveného filtru nebo textového řetězce stáhne volně dostupný obrázek z webové databáze obrázků služby Unsplash[6],

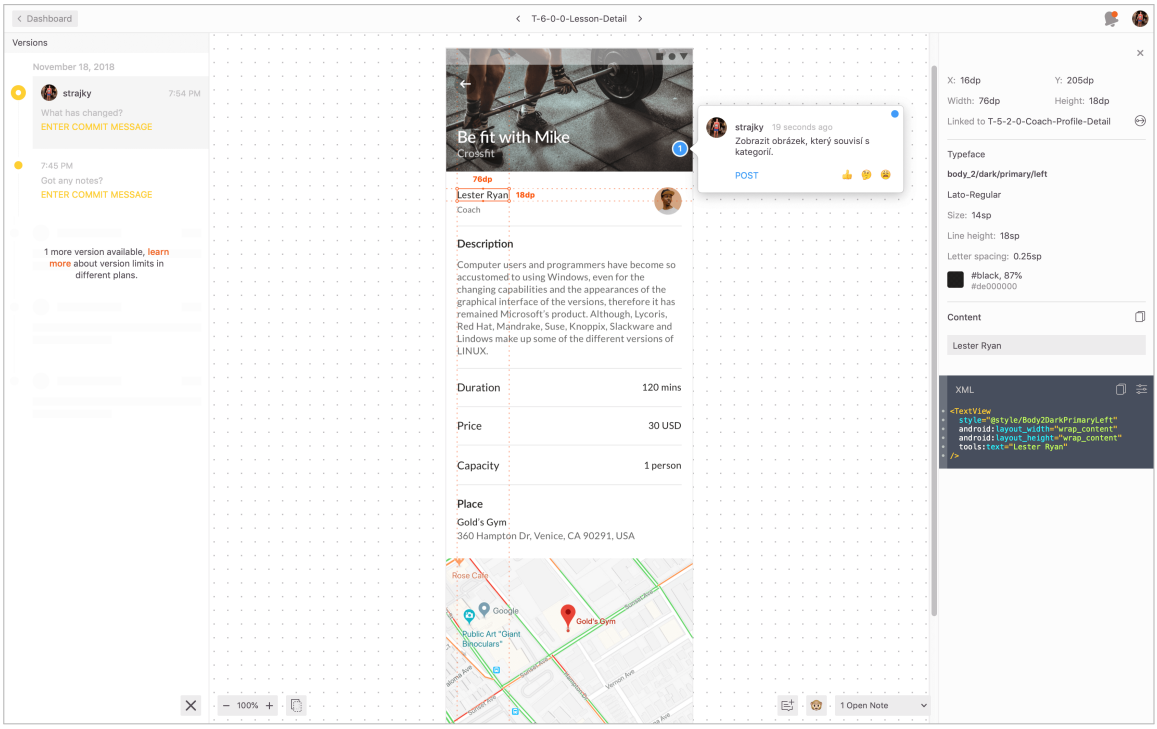
[6] https://unsplash.com/

- **Zeplin** - doplněk pro nahrávání grafických návrhů do služby Zeplin.

Zeplin

Poslední krok, který musí designér udělat, je odprezentovat resp. předložit správnou cestou svůj grafický návrh aplikace klientovi a programátorovi. Zde budu znovu čerpat z vlastní zkušenosti. Pro tento krok jsem si zvolil aplikaci Zeplin[7]. Tato služba umožňuje vytvořit jakousi komunikační bránu mezi grafickým designérem, programátorem a zákazníkem, pro kterého je projekt určen.

Jakmile designér dokončí svůj návrh v aplikaci Sketch, nahraje ho pomocí doplňku do Zeplinu. Do Zeplinu je možné přidávat uživatele, kteří mohou komentovat návrhy. Největší výhodou od ostatních služeb jako je např. InVision, je možnost exportování textový stylů, barev a komponent ze Sketche do Zeplinu. Ve Sketch je také možné nastavit jaká komponenta ukazuje na jakou obrazovku, tedy po nahrání do Zeplinu, vznikne interaktivní obsah vypadající jako reálná aplikace. Více informací o konverzi grafického návrhu na kód použitelný v aplikaci je v kapitole 5.2.



Obrázek 3.2: Ukázka grafického návrhu Fittyy v aplikaci Zeplin, který se využívá pro konverzi návrhu mezi designérem, klientem a programátorem.

3.3 Vývoj mobilních aplikací pro Android

Pro vývoj mobilních aplikací na Android je potřeba znát alespoň základy objektově orientovaného jazyka Java. Dále je potřeba mít vhodný program, ve kterém lze programovat. Pro tyto účely Google ve spolupráci s JetBrains[8] vytvořili Android Studio. Tento program

[7] https://zeplin.io/
[8] https://www.jetbrains.com/

plně splňuje požadavky pro vývoj. V Android Studiu lze nalézt překladač kódu, interpret, vizuální editor, editor kódu, analyzér balíčků aplikace, emulátor zařízení, Software Development Kit (SDK) obsahující kompletní balík vývojářských a odladujících nástrojů. Následně je potřeba aplikaci implementovat a zkompilovat. Výstupem je soubor Android Package Kit (APK), který lze nainstalovat na libovolný telefon se systémem Android.

3.3.1 Kotlin

Kotlin[9] je JVM[10] založený staticky typovaný jazyk vyvinutý firmou JetBrains, která stojí za vytvoření vývojového prostředí Android Studia. Hlavní výhody Kotlinu o proti Javě jsou [16]:

- **méně napsaného kódu**,
- **bezpečnost** – Kotlin se zabývá tzv. prázdnými null hodnotami již v čase kompilace programu, což znamená nutnost explicitně specifikovat zda objekt muže být nulový a před jeho použitím zkontrolovat jeho neplatnost,
- **funkční programování** – využívá konceptů z funkčního programování jako jsou např. lambda výrazy, které řeší některé problémy mnohem jednodušším způsobem,
- **funkce rozšíření** – umožňuje rozšířit libovolnou třídu o nové funkce, i když nemáme přístup ke zdrojovému kódu,
- **vysoce interoperabilní** – umožňuje nadále využívat většinu knihoven a kód napsaný v jazyce Java, dokonce je možné vytvářet smíšené projekty se soubory Kotlinu i Javy.

Z mé zkušenosti se mi nejvíce osvědčil Kotlin právě při redukci kódu a z hlediska bezpečnosti nulových proměnných. Zde bych uvedl jeden zdroj [7], který uvádí, že při použití Kotlinu na místo Javy, je možné ušetřit o 30 % více řádků kódu a o 10 % snížit počet metod.

```
Button loginButton = (Button) findViewById(R.id.login_button);
loginButton.setOnClickListener(new OnClickListener() {
    @Override
    public void onClick(View view) {
      handleButtonClick(view);
    }
});
```

Výpis 1: Ukázkový kód Javy, kde se inicializuje tlačítko z obrazovky a následně se pro něj nastaví naslouchání kliknutí. Po kliknutí na tlačítko se volá metoda, která obslouží tuto interakci.

[9] https://kotlinlang.org/docs/reference/android-overview.html

[10] Java Virtual Machine je sada počítačových programů a datových struktur, které využívá virtuálního stroje ke spuštění programů napsaných v jazyce Java.

```
login_button.setOnClickListeners { view -> handleButtonClick(view); }
```

Výpis 2: Přehlednější a kratší kód napsaný v Kotlině. Kód dělá přesně to stejné jako ukázka kódu ve výpisu 1. Je zde použitá lambda konstrukce a zjednodušené nastavení naslouchání kliknutí.

```
public class User {
	private long id;
	private String name;

	public long getId() {
		return id;
	}

	public void setId(long id) {
		this.id = id;
	}

	public String getName() {
		return name;
	}

	public void setName(String name) {
		this.name = name;
	}
}
```

Výpis 3: Vytvoření datové třídy v Javě. Třída reprezentuje uživatele, který má jméno a identifikační číslo.

```
data class User(var id: Long, var name: String)
```

Výpis 4: Pomocí Kotlinu lze redukovat přebytečný kód. O proti Javě ve výpisu 3 je definice třídy uživatele možná pouze na jednom řádku. Možnosti k získání a nastavení atributů jsou již zahrnuty automaticky ale jsou volány jiným způsobem, který je ukázán ve výpisu 5 na posledním řádku.

3.3.2 Dagger 2 a vkládání závislostí

Při větších projektech jako je tenhle, je zapotřebí, aby kód bylo možné dále škálovat a byl co nejvíce přehledný. Proto se při vývoji aplikací využívá tzv. dependecy injection nebo-li vkládání závislosti. Hlavní účel této technologie je umožnit jednomu modulu použít jiný modul(modul může například představovat třídu). Dependecy injection tuto možnost ulehčuje a velmi zpřehledňuje. Každý modul, ke kterému je možné přistoupit přes konstruktor je

```
// Díky tomuhle nebude možné zkompilovat kód, proměnná nemůže být null
var notNullUser: User = null

// Proměnná user může být null
var user: User? = null

// Znovu kvůli této části nebude možné zkompilovat kód, protože
// user může být null
user.print()

// Použití bezpečného operátora - pokud za user dáme otazník, metoda
// print() se provede pouze pokud user nebude null
user?.print()

// Smart cast - pokud zkontrolujeme proměnnou před null, dále
// nepotřebujeme používat volání pomocí bezpečného operátora
if (user != null) user.print()

// Pomocí vykřičníků můžeme ignorovat skutečnost, že user může být null
user!!.print()

// Elvis operátor je užitečný pokud user je null ale přesto chceme
// prověst v tomto případě operaci přiřazení
val name = user?.name ?: "Neznámé jméno"
```

Výpis 5: Ukázka použití bezpečnosti Kotlinu proti null hodnotám.

jednou jako jedináček[11] vytvořen v jednom hlavním injektoru, odkud se dále jeho instance předává jako atribut konstruktoru modulu, který ho potřebuje. Celkově se tedy všechny moduly vytváří v jednom souboru a jsou sdíleny mezi sebou dle potřeby.

Na Androidu je pro tuto technologii přístupný framework nazývající se Dagger 2[12], který jsem použil v aplikaci Fitty. Tento framework je určený převážně pro nižší nevýkonné zařízení, protože využívá tzv. předkompilátor, který vytváří všechny třídy, které jsou potřeba k další práci [15]. Díky tomuto je Dagger 2 velmi účinný o proti ostatním technologiím. Většinou je využívaný pro databázové třídy a třídy spojené s obsluhou komunikace se serverem. To jsou hlavní důvody, proč jsem tuto technologii použil. Z mé zkušenosti můžu říci, že ušetří práci při vytváření jedináčků, zlepšuje výkon aplikací a přehlednost kódu aplikace. Jedinou nevýhodou je prvotní implementace a zavedení této technologie do projektu, která zabere kolem půl dne.

3.3.3 Android Jetpack

Další z velmi důležitých věcí, se kterou by měl přijít do styku každý programátor systému Android, je Android Jetpack vyvinutý firmou Google. Android Jetpack je kolekce

[11] Jedináček je návrhový vzor tvořen třídou, která se stará o to, aby její instance existovala jen jednou [17].
[12] https://google.github.io/dagger/

softwarových komponent, která ulehčuje vývoj aplikací pro platformu Android [11]. Tyto komponenty jsou vytvořeny tak, aby splňovaly a napomáhaly k nejlepším osvědčeným vývojovým postupům, snižovali množství nadbytečného kódu a zjednodušovali složité úkoly. Android Jetpack je rozdělen do čtyřech částí, které obsahují jednotlivé knihovny sloužící pro určité účely. V následujících podkapitolách jsou jednotlivě popsány pouze ty, které byly použity v aplikaci Fittyy.

Foundation

Pří vývoji aplikací pro Android je důležité dbát na zpětnou kompatibilitu verzí systému, která nám umožňuje zvýšit potencionální počet uživatelů. Z toho důvodu se využívá knihovna AppCompat[13], díky které můžeme používat většinu komponent systému Android (příkladem můžou být dialogové okna, vrchní nebo spodní lišty pro ovládání aplikace apod.).

Další použitou knihovnou Foundation je Multidex[14]. Tuto knihovnu je zapotřebí využít pokud projekt přesáhne 65 536 metod, což při velkém projektu je velmi pravděpodobné. Není potřeba myslet a počítat tyto metody, protože samotné Android Studio při kompilace řekne, že tento limit byl překročen. Nebudu se více o Multidex více rozepisovat, protože to není pro tuto práci důležité. Pouze je potřeba mít na paměti, že v případě přesáhnutí limitu je nutné importovat tuto knihovnu.

Architecture

Nejzajímavější a nejvíce obsáhlejší částí je balíček komponent Architecture. Obecně komponenty tohoto balíčku napomáhají navrhnout robustní, testovatelnou a udržovatelnou aplikaci.

Velmi důležitou součástí projektu je jeho architektura. Nejrozšířenějšími architekturami je Model View Controller (dále MVC), Model View Presenter (dále MVP) a Model-View-ViewModel (dále MVVM). Protože jsem chtěl vycházet hlavně z nových programátorských trendů a řídit se specifikací od Google, tak jsem vybral architekturu MVVM. Důvodem je právě jejich doporučení, a taky komponenta, která je obsažena v Architecture nazývající se **ViewModel**[15]. Co se týče MVVV, tak je to návrhový vzor jehož hlavním cílem je dosáhnout oddělení jednotlivých vrstev, které mají rozdílné role. MVVM lze dělit na tři vrstvy: Základ Material Designu nebo-li Material Foundation je složen z následujících sekcí:

- View zobrazuje uživatelské rozhraní a informuje ostatní vrstvy o akcích uživatele,
- ViewModel se stará o přenos a zobrazení informací ve View,
- Model získává informace z datových zdrojů (z databáze, ze serveru, atd.) a posílá je do ViewModel.

Hlavním rozdílem mezi MVVM a zmíněnými modely MVP a MVC je ten, že se musí dodržovat striktní pravidlo, které říká, že ViewModel nesmí držet reference na View [8]. ViewModel má poskytovat pouze informace do View. Díky tomuhle pravidlu je možné aby jeden View měl více View-Model nebo naopak více View mohlo mít jeden View-Model. Výhodou těchto vztahů je možnost sdílení informací mezi jednotlivými obrazovkami, což

[13] https://developer.android.com/topic/libraries/support-library/packages#v7-appcompat
[14] https://developer.android.com/studio/build/multidex
[15] https://developer.android.com/topic/libraries/architecture/viewmodel

do té doby bylo docela složité implementovat. Více informací týkajíc se implementace je možné najít v kapitole 5.1.

Další komponenta **Lifecycles**[16] provádí akce v reakci na změnu stavu životního cyklu aktivity[17] nebo fragmentu.[18]. Znamená to tedy, že tato komponenta zajišťuje informaci o tom, zda je aktivita nebo fragment pozastaven, zavřen, zobrazen, vytvořen, apod.

Věc, která je úzce svázaná s modelem architektury MVVM je taková, že ViewModel je zodpovědný za odhalování událostí nebo informací, které může View sledovat. Tato část se právě týká další komponenty **LiveData**. LiveData je komponenta, která je zodpovědná za držení stavu a hodnoty resp. se jedná o třídu, která má určitý typ a na základě toho hodnotu. Následně je možná naslouchat na LiveData pomocí tzv. Observer (pozorovatele), který bude informován o jakékoliv úpravě zabalených dat. Hlavním důvodem využití je předejít chybě týkající se stavu, kdy aplikace neodpovídá [8].

V architektuře MVVM je možné najít tzv. **Repository**, která představuje třídu obsluhující operace nad daty jak v databázi, tak nad daty ze serveru. Pro možnost implementace databáze je v balíčku Architecture zahrnutá komponenta **Room**[19]. Room je objektově-mapující knihovna založená na technologii SQLite[20] poskytující ukládání lokálních dat. Umí pracovat jednoduše s LiveData objekty. V době kompilace umožňuje validaci SQL dotazů. Dále umožňuje implementaci práce s tabulkami relační databáze, sledování změn dat nebo spojování či vytváření složitých dotazů. Hlavním cílem komponenty je umožnění vytvoření mezipaměti mezi daty v zařízení a daty uloženými na serveru. Díky tomu může uživatel zobrazit konzistentní daty i přesto, že není připojen k internetu.

Poslední použitou knihovnou z Architecture je **DataBinding**, pomocí které je možné připojit komponenty v uživatelském rozhraní přímo s datovými zdroji. Což při implementaci znamená, že data se nebudou vkládat programově až po inicializaci uživatelského rozhraní, ale již při její inicializaci. Velkou výhodou je, že jakákoliv změna příslušné hodnoty, která je napojena na rozhraní, je promítnuta okamžitě sama na obrazovku aplikace bez potřebné další implementace.

```
<TextView
    android:text="@{viewmodel.userName}" />
```

Výpis 6: Ukázka kódu v XML a použití knihovny Databinding pro napojení proměnné jména uživatele přímo na implementaci uživatelského rozhraní za pomocí ViewModel.

Behavior

Behavior řeší komponenty týkající se integrace služeb systému Android jako jsou například notifikace, oprávnění, sdílení, stahování nebo asistenti.

Hlavní využívanou knihovnou je **Permissions** potřebná pro zpřístupnění mnoho funkcí zařízení aplikaci. Pokud aplikace potřebuje číst telefonní seznam, polohu zařízení, textové

[16] https://developer.android.com/topic/libraries/architecture/lifecycle
[17] Aktivita reprezentuje všechny interakce uživatele a stará se o zobrazení v aplikaci.
[18] Fragment reprezentuje část zobrazení a interakcí v aktivitě.
[19] https://developer.android.com/topic/libraries/architecture/room
[20] SQLite je relační databázový systém založený na strukturovaným dotazovacím jazyce SQL (Structured Query Language) využívaný pro práci s daty v relačních databázích.

zprávy, přístup k internetu, atd., je zapotřebí, aby si aplikace vyžádala přístup od uživatele. Pokud práva na použití služby zamítne, aplikace nemá možnost číst informace z této služby.

User Interface

Balíček User Interface (UI) zahrnuje komponenty, které se využívají při zobrazení obrazovek aplikace. Jedna z nejpoužívanějších komponent je **Layouts**. Velmi důležitá komponent starající se o strukturu uživatelského rozhraní aplikace. Další důležitou komponentou je **Fragment** používaný pro reprezentaci části zobrazení a interakcí v aktivitě. Posledním komponentou, kterou zmíním, je **Animation & transitons** zahrnující všechny možné animace a přechody mezi layouty uživatelského prostředí aplikace.

3.3.4 Lokalizační a mapové služby

Google umožnil vývojářům použít zdarma nebo za poplatek jejich služby, které lze najít pod označením **Google Play Services**[21]. Z nejzajímavějších služeb, které tento balíček obsahuje, je Google Maps, Google Places, Google Analytics, Google Sign In a Google Mobile Ads. Osobně jsem využil právě mapy a místa od Google, které dále popíšu.

Google Maps

Jak už názve napovídá, Google Maps[22] je služba od Google, která umožňuje využít jejich velmi kvalitní mapy v aplikaci. Celá tato služba je zahrnuta v jedné knihovně a po jejím importu a nastavení do projektu aplikace, můžeme na jakoukoliv obrazovku implementovat zobrazení mapy. Knihovna automaticky řeší spojení a komunikaci s jejich API[23], stahování map, zobrazení nebo interakci s mapou. Google Maps dále umožňují přidávat různé tvary, body, značky do mapy, které mohou obsahovat dodatečné informace o daném místě. V roce 2018 se Google rozhodl tuto službu zpoplatnit, ale až od určitých požadavků na tuto službu. Pro tento projekt by měla bohatě stačit verze zdarma bez poplatků.

Google Places

Google Places[24] je služba od Google, která umožňuje vyhledávat místa dle názvu nebo adresy. Tato funkce je využita při vytváření lekce, kdy trenér musí zadat, kde bude lekce uskutečněna. Google Places je možné použít buď na vyhledání míst, které reprezentují přímou adresu, město nebo stát nebo na vyhledání míst, kterou jsou svázané s určitou firmou, restaurací, podnikem, posilovnou, obchodem a podobně.

3.3.5 Komunikace se serverem

Pro fungování aplikace Fittyy je důležité mít internetové připojení, protože je zapotřebí komunikovat s API serveru, které předává a poskytuje informace o lekcí, trenérech nebo

[21] https://developers.google.com/android/guides/setup

[22] https://developers.google.com/maps/documentation/android-sdk/intro

[23] API je zkratka pro Application Programming Interface označující rozhraní, které určuje jakým způsobem jsou funkce knihovny volány ze zdrojového kódu programu.

[24] https://developers.google.com/places/android-sdk/intro

nových rezervací. Pro tento účel je využita knihovna Retrofit[25], která implementuje rozhraní mezi aplikací a API serveru za pomocí HTTP[26] protokolu.

Pro komunikace s API se využívá dvou základních dotazů GET a POST. Metoda GET se využívá když aplikace žádá v dotazu nějaké data a očekává, že je od serveru dostane. Naopak metoda POST posílá data serveru v předem domluveném formátu a očekává odpověď o tom, zda je server patřičně přijal a uložil. Více o implementaci komunikace se serverem je napsáno v kapitole 5.1.7.

3.3.6 Firebase Cloud Messaging

Firebase Cloud Messaging[27] (dále jako FCM) je řešení od firmy Google pro zasílání zpráv mezi různými platformami. Pomocí služby FCM je možné upozornit zařízení, že aplikace má nová data k zobrazení. V případě tohoto projektu je tato funkce užitečná pro upozornění uživatele o tom, že má novou rezervaci lekce nebo se stav rezervace změnil.

3.3.7 RxJava

RxJava je technologie používání pro reaktivní programování na Androidu. Umožňuje vytváření asynchronní datové streamy, které jsou emitovány jako tzv. *Observables*, tedy pozorovatelné data. Dále pak poskytuje tzv. *Observers*, což jsou pozorovatelé, kteří na tyto data naslouchají a čekají se nimi bude nebo kdy se objeví [9]. Nejčastější využití je spojené s voláním endpointů na API server, kde se čeká na odpověď serveru, ve které se nachází data.

[25] https://square.github.io/retrofit/

[26] Hypertext Transfer Protocol je internetový protokol určený pro komunikace se servery po internetovém připojení.

[27] https://firebase.google.com/docs/cloud-messaging/

[27] Endpoint je rozhraní serveru, které poskytuje určitou funkcionalitu a data.

Kapitola 4

Návrh aplikace a serveru

V následující kapitole jsou obsaženy informace týkající se kompletního návrhu aplikace. Aby mohli být všechny služby v aplikaci funkční, je potřeba i serverového řešení, které bude navrhnuto a popsáno taky v této kapitole.

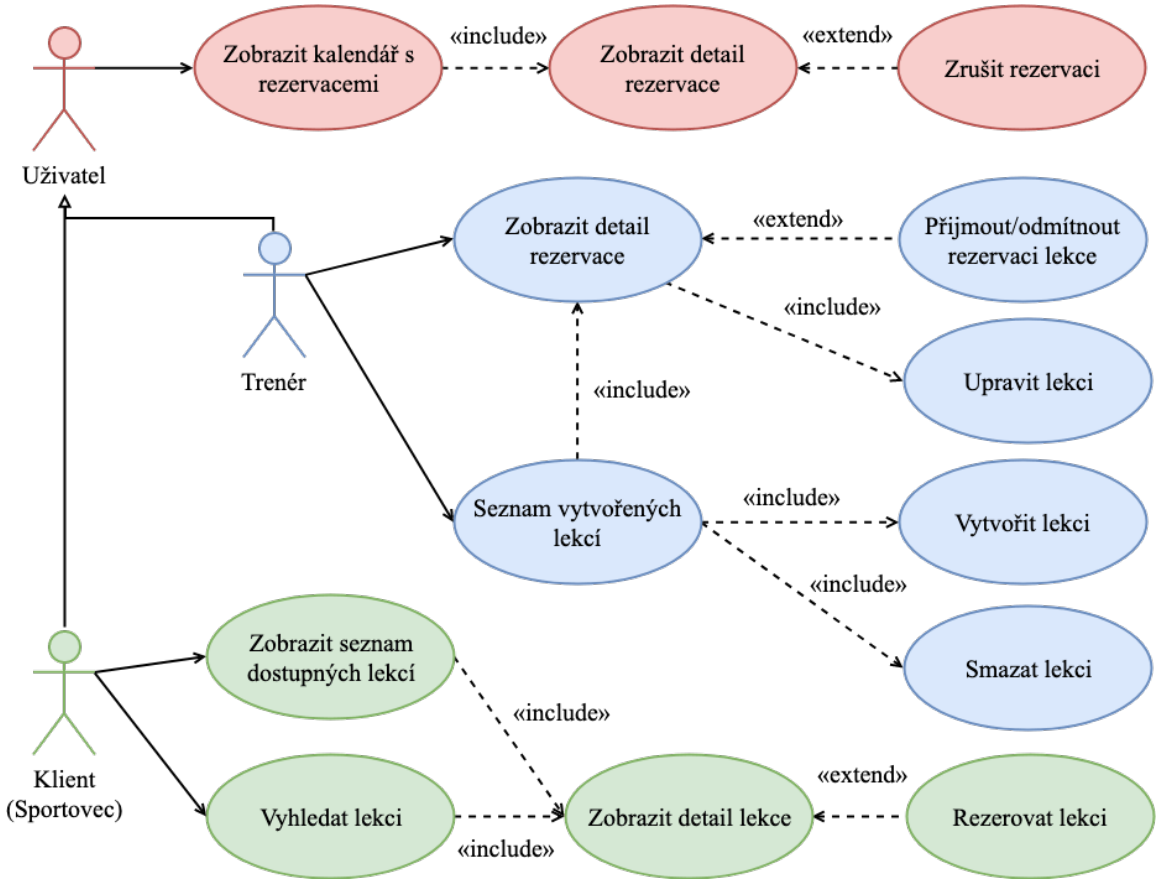
Pro začátek bych zde napsal pár základních informací o službě, kterou by aplikace měla spolu se serverem nabízet. Rezervační systém pro sportovní lekce Fittyy je vhodný a zaměřený primárně na sportovní lekce. Tento systém by měl umožňovat rychlou a jednoduchou správu nebo rezervaci sportovní lekce. V aplikaci vystupují dvě role. Jedna z nich je trenér, který nabízí sportovní lekce a přijímá jejich rezervace. Naopak druhá vystupující role je sportovec nebo-li klient, který si lekce u trenéra objednává. Podrobnější návrh služby je rozepsán v následujících kapitolách.

Dále chci zde podotknout, že na začátku této práce byla vypracována analýza pomocí dotazníku, který se ptal na to, zda by člověk použil spíše webovou nebo mobilní aplikaci proto, aby si mohl objednat sportovní lekce. Celkem jsem posbíral 63 odpovědí od lidí v rozmezí věku 18 až 58 let. Díky tomuto dotazníku bylo zjištěno, že tato práce se bude zaobírat návrhem a vývojem mobilní aplikace (pro mobilní aplikaci hlasovalo 56,5 % potenciálních uživatelů, tedy 35 lidí).

4.1 Analýza a specifikace aplikace

Vůbec prvním krokem při návrhu aplikace byla identifikace problémů, které by aplikace měla vyřešit svými službami. K tomu jsem využil diagram případu užití, který je vhodný pro odhalení vnějšího pohledu na modelovaný systém, díky čemuž pomáhá i k objevení hranic systému, v tomto případě mobilní aplikace.

Diagram případu užití na obrázku 4.1 vycházel z analýzy a průzkumu problému trenéru 2.1.1. Jak se ale ukázalo v průběhu testování, trenéři měli ještě jiné požadavky a priority, které zde ale nebudu uvádět, neboť se jim věnuji až v rámci kapitoly týkající se testování 6. Jak tedy můžete vidět na obrázku 4.1, vytvořil jsem diagram, kde jsem zanesl pouze klíčové a důležité případy užití, které řeší problémy a nedostatky trenérů či sportovců. Všichni uživatelé bez rozdílu role mohou zobrazovat vlastní kalendář obsahující rezervace a odtud si zobrazit detail rezervace. Pokud je rezervace schválená, uživatel ji může z jejího detailu zrušit. Uživatel v roli trenéra má možnost zobrazit seznam vlastních lekcí, vytvořit nebo upravit lekci, případně lekci i smazat. Správa rezervací je možná přes detail rezervace, kde má trenér možnost přijmout nebo zamítnout rezervaci. Uživatel, který vystupuje jako sportovec (klient) může lekce vyhledávat podle klíčových slov nebo si jen jednoduše



Obrázek 4.1: Diagram případů užití mobilní aplikace Fittyy

zobrazit seznam dostupných lekcí. Pokud se mu detailní informace o lekci zalíbí, muže si ji zarezervovat.

4.2 Uživatelské rozhraní

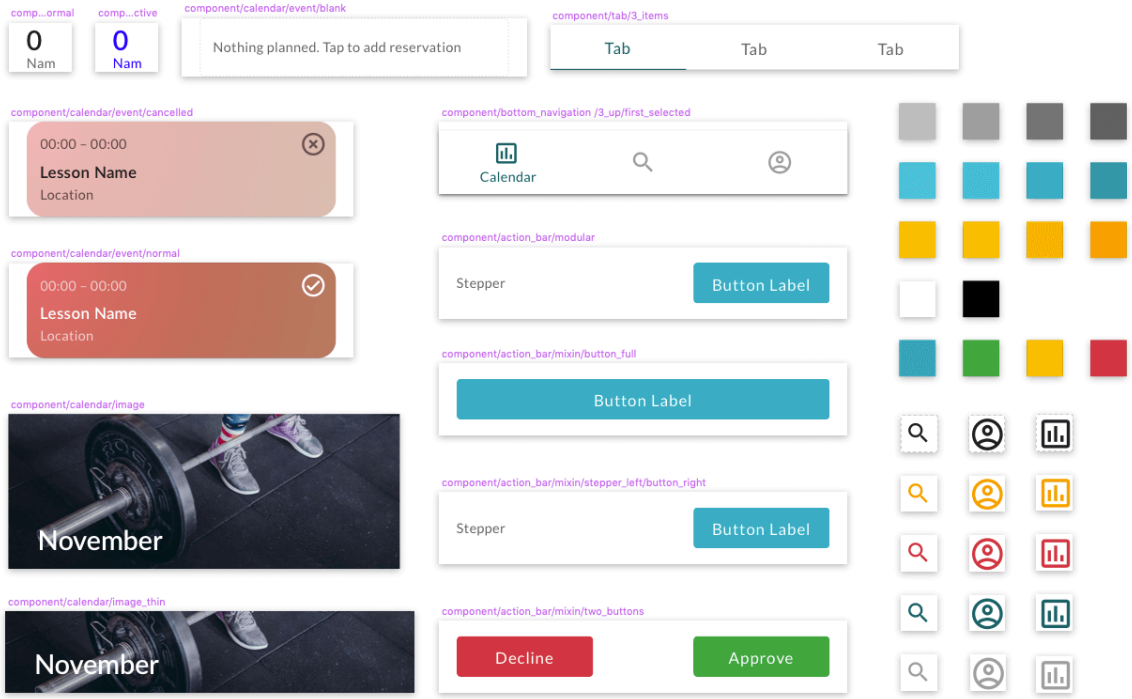
Návrh uživatelského rozhraní je velmi důležitou součástí vývoje mobilní aplikace. Jak již bylo zmíněno v kapitole 3.2, při návrhu mobilní aplikace se nejčastěji využívá designového systému Material Design. Dále jsem v této kapitole uváděl, že se velmi inspiruji aplikací Airbnb.

V následujících podkapitolách podrobněji rozeberu, jak jsem aplikaci Fittyy navrhoval, na jaké problémy jsem narazil a jakým způsobem jsem je vyřešil.

4.2.1 Návrh uživatelského prostředí

Pro návrh uživatelského prostředí jsem využil nejznámější placenou aplikaci Sketch (popsaná dříve v 3.2.1), kterou používají profesionální grafičtí návrháři. Díky ní jsem mohl vytvořit grafické komponenty aplikace, které jsem pak mohl libovolně používat na kterékoliv obrazovce. Pokud jsem tuto komponentu upravil v knihovně komponent, změna se mi promítla do celého návrhu. To mi později šetřilo velice času, protože hodně prvků v návrhu jsem upravoval. Na obrázku 4.2 je zobrazena ukázka designových komponent, které jsem opakovaně využívali při vytváření uživatelského prostředí v programu Sketch. Na levé

straně obrázku jsou komponenty, které se využívají v návrhu kalendáře rezervací. V prostřední části obrázku je komponenta navigačního menu a různé druhy tlačítek používaných skrz celou aplikaci. Úplně na pravé straně je ukázka zadefinovaných barev a barevných stavů ikonek.

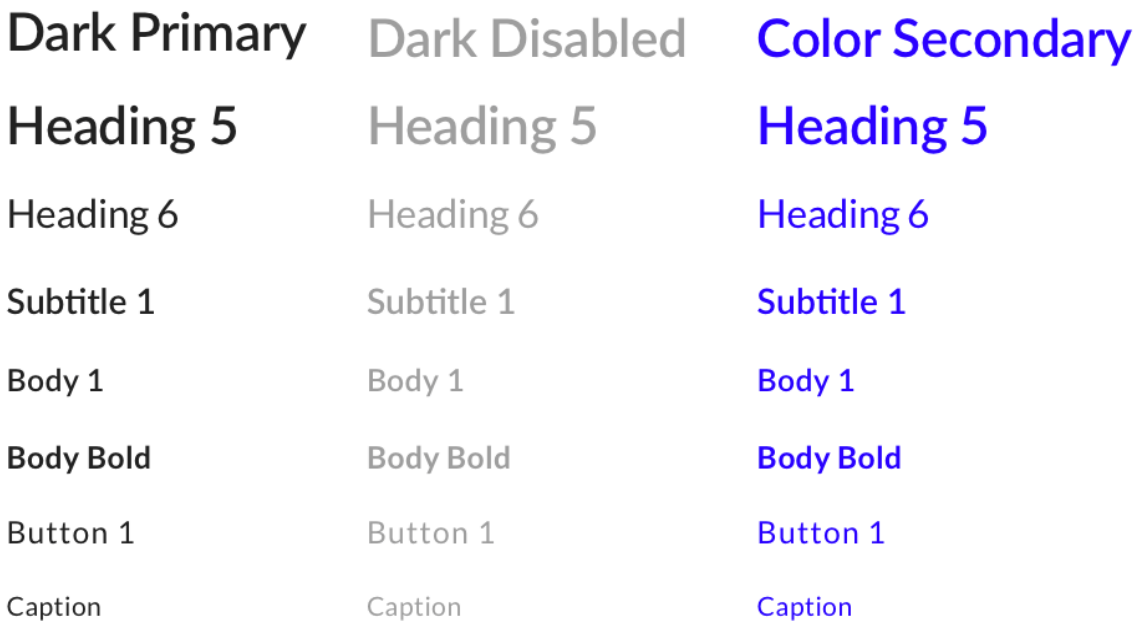


Obrázek 4.2: Ukázka vytvořených komponent, které se opakovaně používaly při návrhu uživatelského prostředí v programu Sketch.

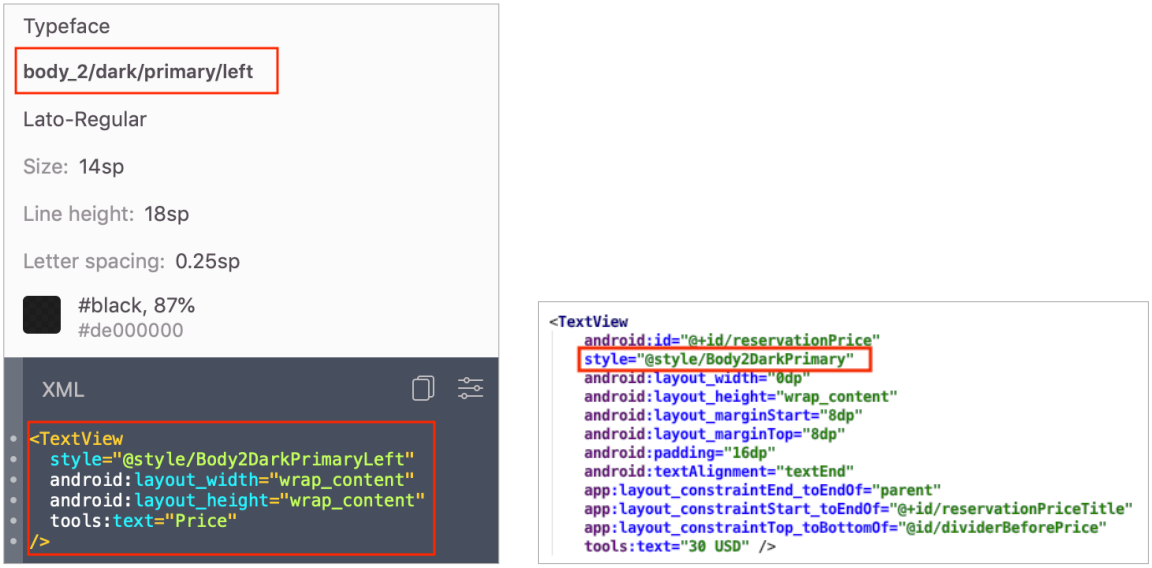
Na dalším obrázku 4.3 lze vidět tři odstíny textových stylů. Při vytváření textových stylů jsem znovu vycházel z pravidel Material Themingu, a taky z knížky, která píše o chování uživatelů [20]. Zde Susan Weinschenk popisuje, že nemáme od uživatelů očekávat jejich pozornost na informace, které jim zobrazujeme. Dále zde zmiňuje, že pro lepší získání pozornosti lidí na věci, které jsou důležité, je dobré používat různé barvy a velikosti. Tuto informaci jsem použil i při vytváření barev aplikace. Textové styly jsem navrhl tak, že každý z nich obsahuje informaci o typu fontu, velikosti písma, řádkování, barvy písma a vlastním názvu, který vystihuje jeho vlastnosti. Důvod použití resp. pojmenování stylů a definice tohoto systému má další souvislost s programem Zeplin, o kterém jsem již psal v podkapitole 3.2.2. Díky tomuto programu jsem schopen jednodušeji přenést grafický návrh do programové podoby. A právě textové styly se v tomto programu také zobrazují, tudíž mně u každého použitého textového pole stačí zjistit název stylu např. *Headline1DarkPrimary* a v programové části aplikace přidat pouze jeden řádek s tímto hodnotou obsahující tento název. Pokud bych nepoužil textové styly, musel bych pokaždé složitě vypisovat všechny jednotlivé vlastnosti textového pole, což by zabralo mnohem více času. Dále mi tento systém jednoduše umožnil možnost měnit globálně vlastnosti textových polí aniž bych to musel měnit na každé obrazovce. Později se mi tento krok velmi vyplatil, protože jsem v průběhu vývoje měnil font písma. Použití tohoto systému můžete vidět na obrázku 4.4.

Nicméně Zeplin má mnoho dalších funkcí. Já jsem v tomto programu pro tuto práci nejvíce využil možnost zobrazit si přehledně grafický návrh doplněný o všechny parametry

potřebné pro umístění komponenty na obrazovce a možnost interaktivního návrhu. Díky tomu jsem mohl procházet obrazovkami v aplikaci, aniž bych měl cokoliv naprogramované. Což se mi nejčastěji hodilo, pokud jsem chtěl vědět kam dané tlačítko na obrazovce vede.



Obrázek 4.3: Ukázka textových stylů vytvořených dle pravidel Material Themingu.



Obrázek 4.4: Ukázka použití textového stylu při konverzi z návrhu do programové podoby, **vlevo:** informace zobrazující se v programu Zeplin, **vpravo:** část kódu při implementaci obrazovky detailu rezervace, kde byl použit textový styl z návrhu.

Postup návrhu uživatelského prostředí

Poté co jsem navrhl základní komponenty, bylo potřeba více promyslet, co všechno bude aplikace obsahovat a na základě toho namalovat prvotní náčrty. V tomto mi velice pomohl diagram případů užití, který zachycoval ty nejdůležitější funkce aplikace. Zde jsem narazil na první problém, kdy bylo potřeba nějak rozumně rozlišovat návrh obrazovek z pohledu trenéra nebo z pohledu sportovce. Proto jsem každou obrazovku začal označovat *T-X-Y-Z-Název-Obrazovky* nebo *C-X-Y-Z-Název-Obrazovky*, kde jednotlivé písmena znamenají:

- T — obrazovka z pohledu sportovce,
- C — obrazovka z pohledu trenéra,
- X — hlavní číslo identifikující obrazovku,
- Y — vedlejší identifikační číslo obrazovky,
- Z — číslo identifikující stav obrazovky.

Jako příklad uvádím označení obrazovky *T-2-0-1-Search-Lesson-List*, která zobrazuje seznam lekcí z pohledu sportovce.

V dalším kroku jsem se zaměřil primárně na procesy vytváření lekce, rezervace lekce a vyhledávání lekcí či trenérů, které jsou pro aplikaci stěžejní. Nakonec jsem doplnil méně důležité obrazovky, které souvisí např. se zobrazením rezervací, nastavením aplikace, úpravou profilu, zobrazením profilu, apod. Jednotlivé grafické návrhy obrazovek jsou zobrazeny v kapitole 4.3, kde jsou podrobněji popsány i s procesem, který se může na obrazovce odehrávat.

4.3 Návrh a specifikace procesů v aplikaci

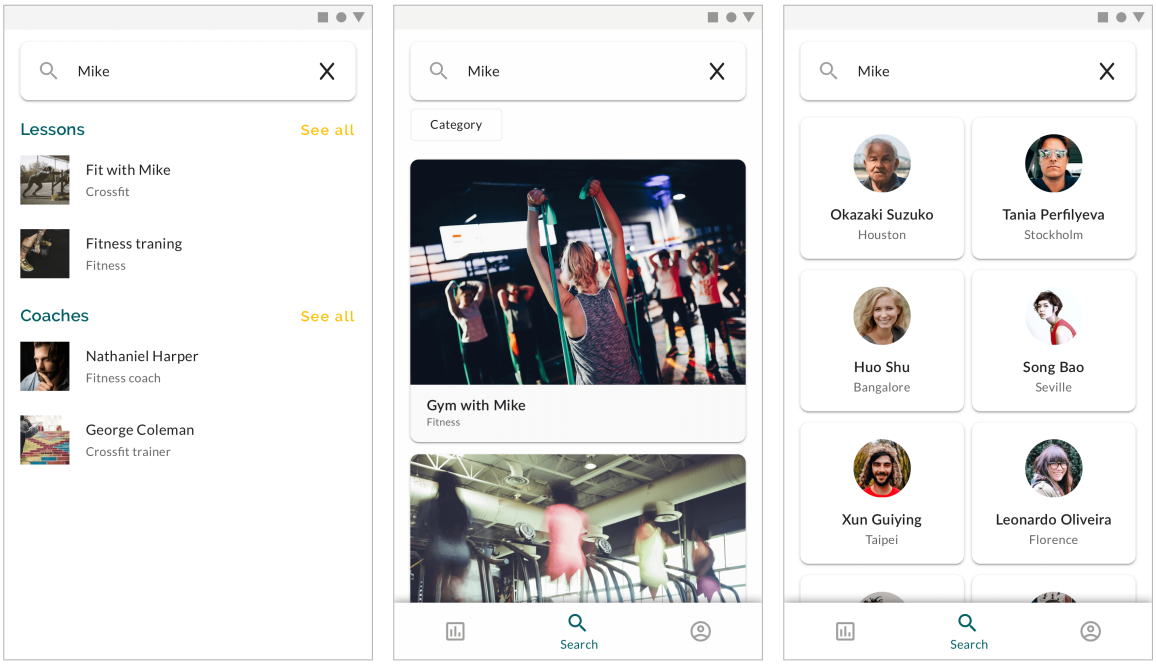
Jednou z nejdůležitějších částí této práce bylo správně navrhnout a specifikovat jednotlivé procesy, ve kterých interaguje uživatel s aplikací. V rámci návrhu uživatelského prostředí jsem vytvořil pomocí nástroje Sketch celkem 62 obrazovek, které byly navrhnuty na základě průzkumu mezi trenéry, který jsem popsal v podkapitole 2.1.1. V následujících podkapitolách budou popsány a ukázány ty nejdůležitější. Ještě zde chci podotknout, že návrhy jsou vytvořeny v anglickém jazyce, ale samotná aplikace je lokalizována i do českého jazyka.

4.3.1 Registrace uživatele

Proto aby mohl uživatel využívat aplikaci, je zapotřebí se registrovat nebo přihlásit. Registrace uživatele obnáší volbu jeho role, pod kterou bude v rámci aplikace prezentován. Pokud uživatel zvolí roli trenéra, bude moci nabízet sportovní lekce. Naopak volba role sportovce obnáší možnost vyhledávání lekcí a vytváření rezervací. Dále je v registraci požadovaný email a heslo pro možné přihlášení se do aplikace. Po vyplnění těchto údajů musí doplnit uživatel své jméno a příjmení a následně se teprve může registrovat.

4.3.2 Přihlášení uživatele

Přihlášení do aplikace obnáší správné vyplnění emailu a hesla. Jakmile se tyto údaje ověří, uživatel může začít využívat aplikaci. Bez úspěšného přihlášení nebo registrace není možné zobrazit žádné informace týkající se služeb trenérů a dat v aplikaci. Pokud uživatele zadá nevalidní email nebo krátké heslo, tlačítko pro přihlášení není možné stlačit.



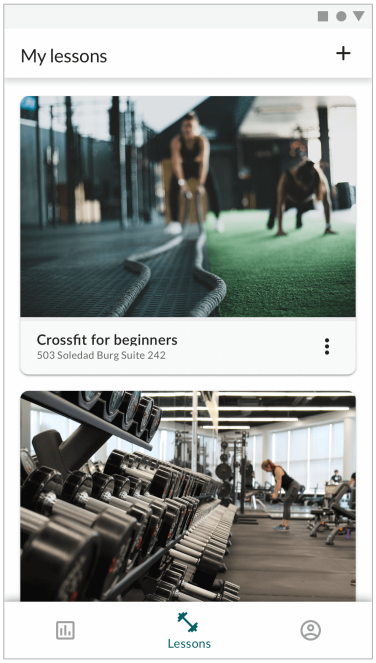
Obrázek 4.5: Grafický návrh pro vyhledávání lekcí a trenérů, **vlevo:** smíšené vyhledávání lekcí a trenérů, **uprostřed:** podrobnější vyhledávání lekcí dle výrazu „Mike“, na které se lze dostat přes tlačítko *See all*, **vpravo:** podrobnější vyhledávání trenérů dle výrazu „Mike“, na které se lze dostat přes tlačítko *See all*.

4.3.3 Vyhledávání lekcí a trenérů

Velmi důležitou obrazovkou v aplikaci je vyhledávání a objevování nových lekcí. Navrhl jsem to způsobem, kdy po kliku ve spodním menu může pouze sportovec zobrazit poslední aktuální nové lekce. Pokud by chtěl sportovec vyhledávat, musí kliknout na vyhledávací pole, které otevře obrazovku vyhledávání. Na obrazovce 4.5 vlevo je společné vyhledávání lekcí a trenérů dle zadaného výrazu obsaženého v textovém poli, které se nachází v horní části obrazovky. Pokud tedy sportovec zadá patřičný výraz pro vyhledávání, zobrazí se mu dvě lekce a dva trenéři (obrázek 4.5 vlevo). Odtud se může sportovec prokliknout na detail lekce nebo profil trenéra. V případě kdy v seznamu nenajde hledanou lekci nebo trenéra, má možnost zobrazit více výsledků v novém okně (obrázek 4.5 vpravo a uprostřed). Seznam lekcí je tvořen pouze základními informacemi o lekci, jako je název a umístění lekce. Pokud uživatel hledá trenéra, může v seznamu trenérů naleznout jejich jméno a místo, kde působí.

4.3.4 Seznam lekcí trenéra

V předchozí podkapitole 4.3.3 je popsáno vyhledávání lekcí, které ale trenér nemá možnost vykonávat. Místo této obrazovky má trenér zobrazen seznam svých lekcí (obrázek 4.6). Proto se liší i popisek a ikonka ve spodním menu. Odtud si může jednoduše spravovat svoje lekce, zobrazit detail lekcí nebo vytvářet nové lekce pomocí tlačítka plus v pravém horním rohu. U každé lekce může trenér pomocí tlačítka reprezentovaného třemi tečkami, nevratně smazat lekci.



Obrázek 4.6: Grafický návrh zobrazující seznam lekcí, které může zobrazit pouze majitel v roli trenéra.

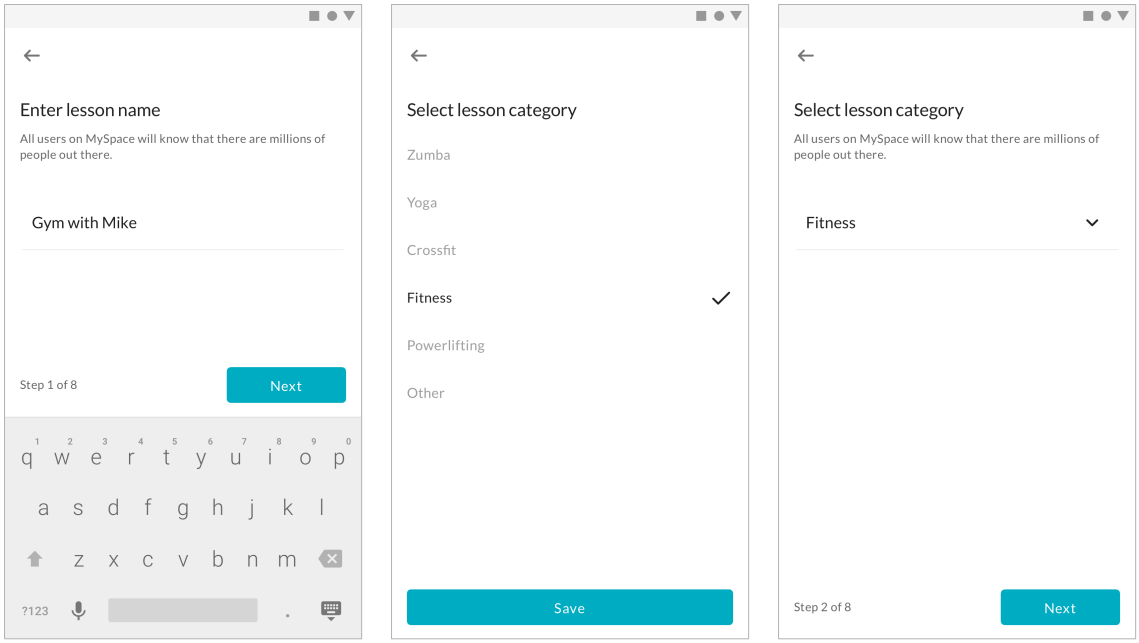
4.3.5 Vytvoření a úprava lekce

Vytváření lekcí je možné provést buď ze seznamu lekcí trenéra nebo z profilu trenéra. Kompletní proces je rozdělen do jednotlivých kroků, které na konci vedou k úspěšnému vytvoření lekce. Bez všech kroků, které budou zde zmíněny, není možné lekci vytvořit. Každá obrazovka, která představuje jeden krok v procesu, je tvořena titulkem, krátkou vysvětlivkou, sekcí pro vyplnění patřičných informací, číslem aktuálního kroku, počtem celkových kroků a tlačítkem, které slouží pro přesun do dalšího kroku. Trenér může procházet proces i zpětně a vracet se tam, kde potřebuje upravit informaci související s lekcí.

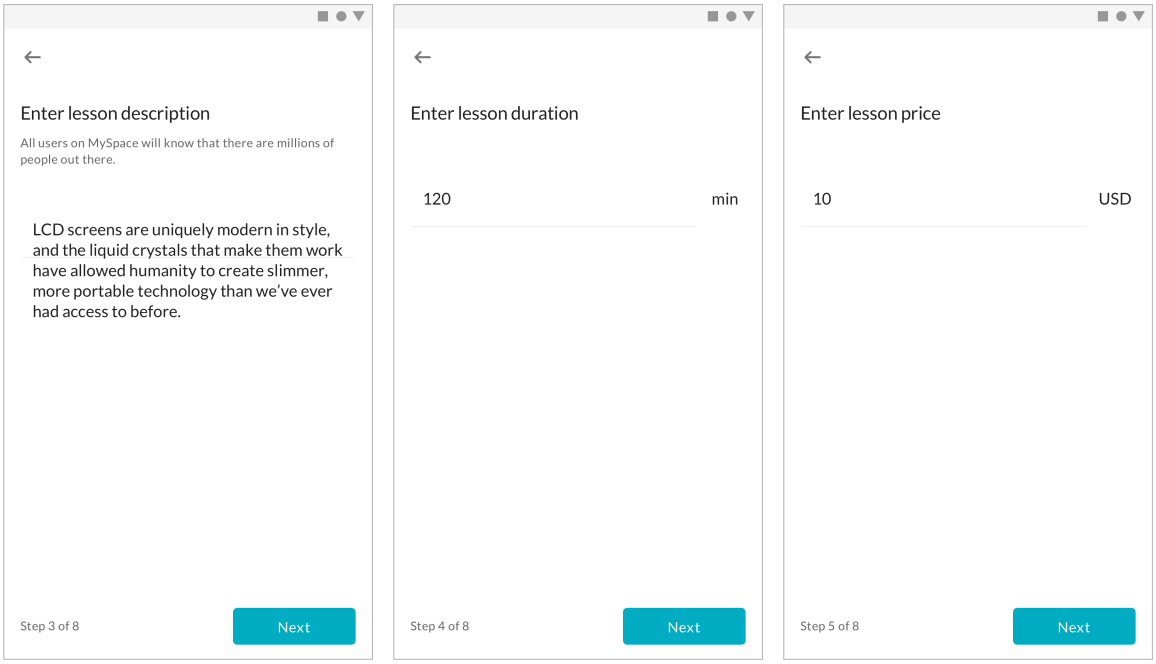
První obrazovka (obrázek 4.7 vlevo) žádá vložení názvu lekce, který by měl být výstižný a podle kterého se lekce může vyhledat. Dokud trenér nezadá nějaký název, nebude moci postoupit do dalšího kroku.

Ve druhém kroku musí trenér vybrat kategorii, do které lekce spadá. Po kliku na textové pole (obrázek 4.7 vpravo) se zobrazí nová aktivita, která obsahuje seznam všech možných kategorií (obrázek 4.7 uprostřed). Po výběru kategorie se trenér tlačítkem uložit vrátí zpět do kroku dvě s vyplněnou kategorií.

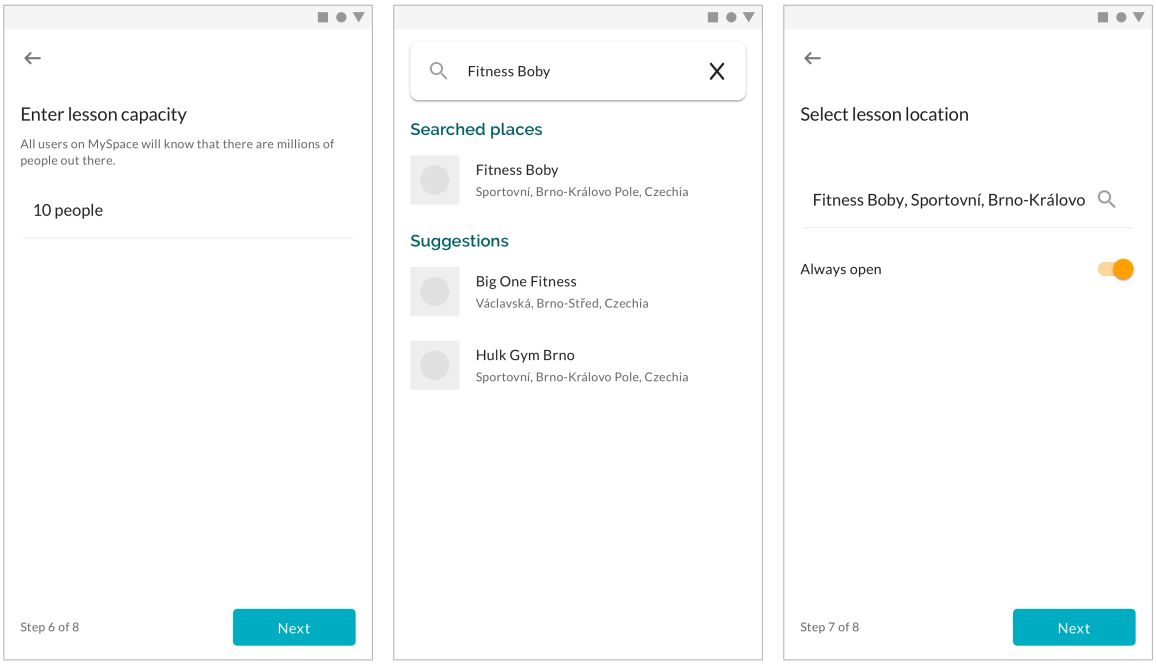
V následujícím třetím kroku musí trenér stručně popsat, co lekce obnáší a proč by si sportovec měl lekci rezervovat (obrázek 4.8 uprostřed). Následuje krok čtyři, kde trenér zadává délku trvání lekce (obrázek 4.8 vlevo). Délka lekce se zadává v minutách a je důležitá hlavně při rezervaci, kdy se tvoří jednotlivé časové bloky, kdy je lekce dostupná, a taky jako informační časový údaj, kdy bude lekce končit. Pátým krokem je výběr ceny, kterou bude po uskutečnění lekce muset klient zaplatit. Zde v návrhu je cena v amerických dolarech, ale v aplikaci jsem nakonec použil pouze českou korunu. Všechny tři zmíněné kroky jsou opět doplněny tlačítky, které trenéra směřují kupředu pouze tehdy, pokud má všechny informace vyplněné.



Obrázek 4.7: Grafický návrh prvních tří kroků vytváření lekce, **vlevo:** zadání názvu lekce, který se využívá při vyhledávání lekcí a je důležitý pro upoutání potenciálních klientů, **uprostřed:** výběr kategorie sportovní lekce, který se provádí v nové aktivitě aplikace, **vpravo:** druhý krok obnáší výběr kategorie lekce.



Obrázek 4.8: Grafický návrh obrazovek pro vyplnění popisu, délky trvání a ceny lekce, **vlevo:** vyplnění stručného popisu, který může sportovce zaujmout, **uprostřed:** délka trvání používající se při vytváření dostupnosti lekce a pro časovou informaci ukončení lekce, **vpravo:** cena, kterou klient musí zaplatit za lekci.

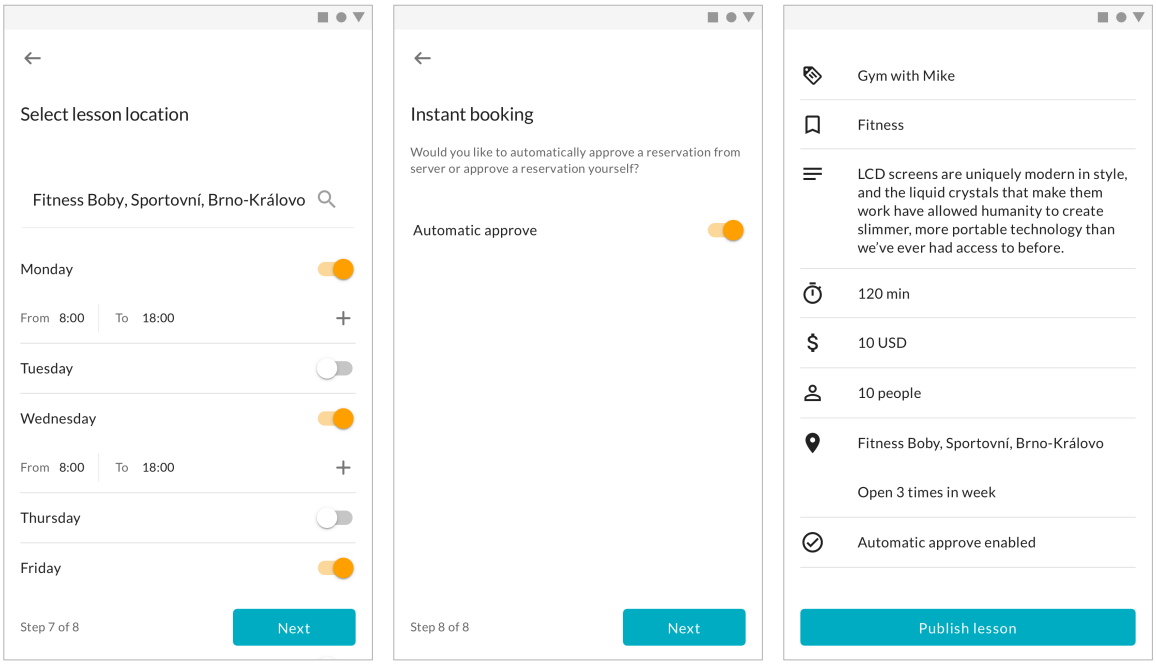


Obrázek 4.9: Grafický návrh pro zadání maximální kapacity, vyhledání a zvolení místa konání lekce, **vlevo:** maximální kapacita lekce udává, kolik sportovců si ji může rezervovat na stejný čas, **uprostřed:** vyhledávání místa konání pomocí zadaného řetězce, **vpravo:** nastavení místa konání lekce a časové dostupnosti pomocí oranžového přepínače.

V šestém kroku se po trenérovi požaduje kapacita lekce (obrázek 4.9 vlevo). Tato číselná hodnota představuje maximální kapacitu obsazenosti lekce pro daný časový úsek. To znamená, že pokud je např. kapacita rovna jedné, tak na lekci v určité datum a čas se může rezervovat pouze jeden sportovec.

Sedmý krok požaduje, aby trenér vybral místo konání lekce (obrázek 4.9 vpravo). Kliknutím na textové pole, kde se má nacházet název umístění, se trenér dostane do nové aktivity (obrázek 4.9 uprostřed), která umožňuje textové vyhledávání ve službě Google Places. Díky tomu trenér nalezne přesnou adresu a název místa (typicky fitness centra, posilovny, sportovního působiště), které se použijí v detailu lekce nebo jako doplňující informace při vyhledávání lekcí sportovcem. Zároveň je na této obrazovce po trenérovi požadováno, aby nastavil časovou dostupnost lekce. Ve výchozím nastavení je lekce dostupná kdykoliv, což je reprezentováno oranžovým přepínačem přepnutým doprava. Pokud ale trenér má jiný požadavek, musí kliknout na přepínač, který se přepne do nesepnutého stavu a má šedou barvu. Tím si trenér odemyká možnost nastavit časový úsek dostupnosti pro každý den zvlášť. Pokud má být lekce pro daný den dostupná, musí být přepínač v sepnutém stavu, kdy je obarvený oranžovou barvou (obrázek 4.10 vlevo). Následně je možné pro daný den vybrat časový úsek, který reprezentuje interval dostupnosti lekce. Trenér si tak může naplánovat dostupnost lekce na celý týden. Dostupnost lekce se každý týden periodicky opakuje dle tohoto nastavení.

Při návrhu jsem chtěl trenérovi dát co nejvíce možností tak, aby vlastní lekce byly nastavené přesně podle jeho představ. Proto jsem do dalšího kroku osm přidal obrazovku s nastavením automatického schvalování (obrazovka 4.10 uprostřed). Znovu lze nastavení provést přepínačem, který vychází ze specifikace Material Designu. Díky tomu si trenér



Obrázek 4.10: Grafický návrh závěru vytváření lekce trenérem, **vlevo:** nastavení dostupnosti lekce pro každý den v týdnu, **uprostřed:** nastavení automatického schvalování rezervace lekce, **vpravo:** shrnutí všech nastavených informací lekce a tlačítko pro publikování.

může vybrat, zda nová rezervace lekce sportovcem bude automaticky schválena, nebo ji trenér musí schválit či zamítnout dodatečně sám.

Obrazovka 4.10 vpravo je poslední fází procesu vytváření lekce. Obsahuje shrnutí všech informací, které trenér dosud vyplnil. Pokud zde najde nějakou chybu nebo nesrovnalost, může se kliknutím na danou položku dostat zpět do požadovaného kroku, kde informaci upraví. Následně se již zpět do shrnutí dostane tlačítkem na uložení, tak aby nemusel znovu procházet všemi kroky. Jakmile trenér překontroluje všechny informace, zbývá jen kliknout na tlačítko publikovat lekci, které odesílá požadavek na server. Pokud je vytvoření úspěšné, zobrazí se mu detail lekce. Pokud nastane nějaký problém, zobrazí se chybová hláška.

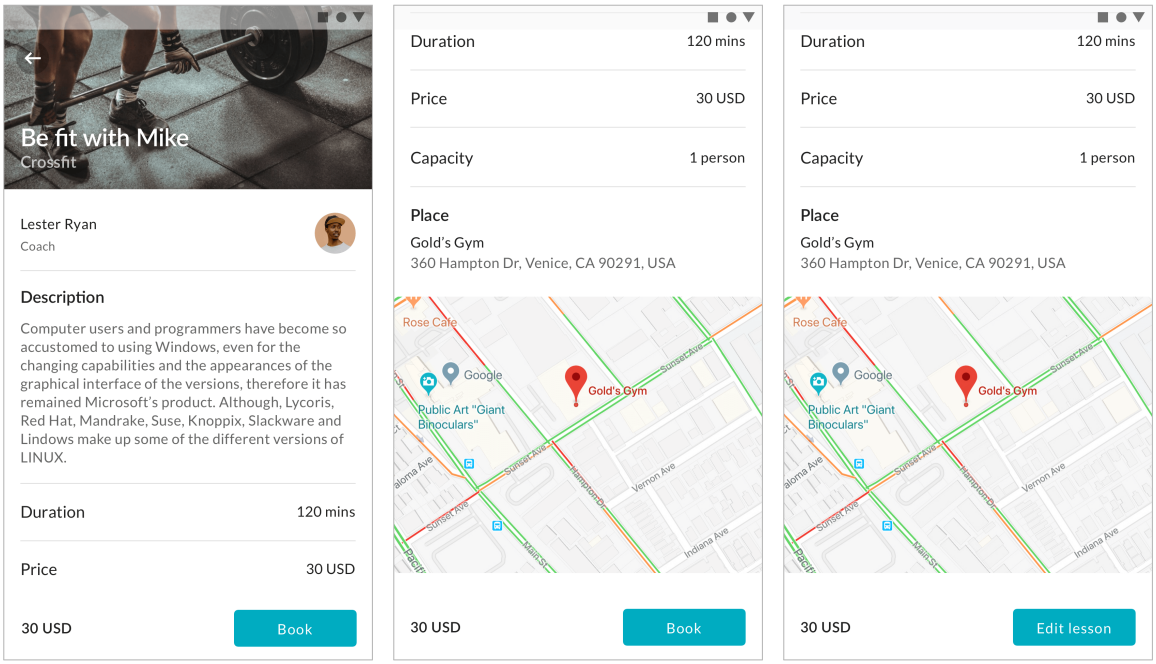
Takto funguje proces vytváření lekce, který je možná delší, ale na druhou stranu umožňuje trenérovi co nejvíce nastavit vlastnosti lekce a pokrýt tak jeho potřeby.

Pokud trenér chce lekci upravit, proces je velmi podobný. Pouze v prvním kroku zobrazí rovnou souhrn informací lekce, odkud se může dostat do nastavení jednotlivých údajů o lekci.

4.3.6 Detail lekce

Pokud sportovec lekci vyhledá, může si pro ni zobrazit její detail s podrobnějšími informacemi. Tato obrazovka představuje místo, které má reprezentovat lekci trenéra a nalákat uživatele k tomu, aby si ji rezervoval. Proto jsem se při návrhu zaměřil na co nejlepší přehlednou prezentaci informací. Vycházel jsem přitom z aplikace Airbnb popsané v kapitole 2.3. Detail lekce na obrázku 4.11 představuje zobrazení následujících informací:

- název lekce,
- kategorie reprezentující zaměření lekce,



Obrázek 4.11: Grafický návrh detailu lekce, **vlevo:** horní část detailu lekce z pohledu sportovce zobrazující základní informace o lekci, **uprostřed:** spodní část detailu lekce z pohledu sportovce obsahující dodatečné informace o ceně a místě konání lekce, **vpravo:** detail lekce z pohledu vlastníka tedy trenéra, který má možnost lekci pouze upravit.

- jméno a avatar trenéra spojený s prokliknutím na jeho profil,
- popis sportovní lekce,
- délka trvání lekce,
- cena lekce,
- maximální kapacita lidí, kterou lekce může mít,
- mapa a přesná adresa, kde se lekce uskuteční.

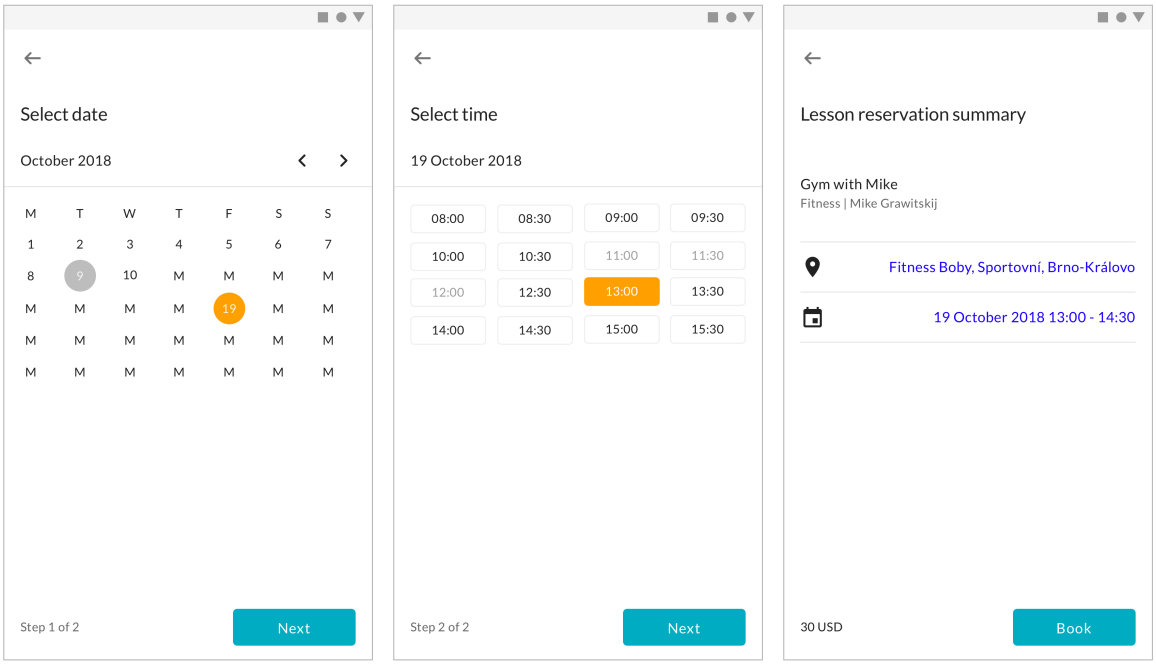
Pokud informace o lekci sportovce zaujme, může si ji pomocí tlačítka v pravém spodním rohu rezervovat. Následně mu je zobrazen proces rezervace lekce, který je popsán v podkapitole 4.3.7.

Trenéři na rozdíl od uživatelů v roli sportovce, můžou zobrazit pouze svoje lekce, kde místo rezervačního tlačítka mají akci (obrázek 4.11 vpravo), která je vezme do procesu úpravy lekce popsaný v podkapitole 4.3.5.

4.3.7 Rezervace lekce

Proto aby uživatel v roli sportovce mohl jít na sportovní lekci trenéra, musí si předně lekci rezervovat. V rámci této aplikace je to jedna z nejdůležitějších částí, kterou poskytuje. Snažil jsem se tento proces udělat co nejkratší. Výsledkem je rezervace lekce ve třech krocích:

1. První krok obnáší **výběr dostupného data** (obrázek 4.12 vlevo), ve které chce sportovec lekcí mít. Pokud je lekce nedostupná pro daný den, je toto datum zašedlé



Obrázek 4.12: Grafický návrh procesu rezervace lekce sportovcem, **vlevo:** prvním krokem rezervace je výběr data, ve který je lekce dostupná, **uprostřed:** druhý krok zahrnuje výběr času, kdy je lekce dostupná, **vpravo:** poslední krok zobrazuje přehled informací, které sportovec vybral a obsahuje tlačítko pro závaznou rezervaci lekce.

a není možné ho vybrat. Naopak dostupné dny jsou černou barvou. Jakmile sportovec vybere datum, musí kliknout na tlačítko dále. Na obrazovce je v jeden čas zobrazen pouze jeden měsíc pro daný rok. Pomocí šipek lze docílit změny měsíc vpřed nebo vzad. Nelze ovšem jít zpět do měsíce, který je již v minulosti.

2. Dalším krokem musí sportovec provést **výběr přesného času začátku lekce** (obrázek 4.12 uprostřed). Časy, které jsou zašedlé, jsou již nedostupné a trenér je v tuto dobu zaneprázdněný. Pokud se některé časy vůbec nezobrazují v seznamu, znamená to, že lekce není v tento časový blok vypsaná. Hodiny, které jsou zvýrazněny černou barvou, jsou dostupné a sportovec je může vybrat. Následně je znovu potřeba kliknout na tlačítko dále.

3. Poslední obrazovka (obrázek 4.12 vpravo) zobrazuje **shrnutí informací**, které sportovec v předchozích krocích vybral. Slouží pouze pro kontrolu. Nakonec tlačítkem rezervovat v pravém spodním rohu sportovec závazně objedná lekci. Následně se zobrazí detail rezervace popsaný v podkapitole 4.3.8.

4.3.8 Detail rezervace lekce

Pokud uživatel rezervuje lekci, může si kdykoliv zobrazit její detail, který je dostupný z kalendáře rezervací popsaný v 4.3.9. Grafický návrh detailu rezervace lekce (obrázek 4.13 a 4.14) zahrnuje pro sportovce i trenéry:

- název lekce,

- kategorie reprezentující zaměření lekce,
- informace o trenérovi nebo informace o sportovci, záleží z jaké role se na návrh díváme,
- adresu konání lekce,
- tlačítko vedoucí na detail lekce,
- pokud je rezervace schválená, tak tlačítko zrušení rezervace,
- datum a čas konání lekce,
- stav rezervace.

Stavy rezervací

Rozdílem v návrhu pro sportovce a trenéry je v akcích, které na obrazovkách můžou konat. Tyto akce jsou odlišné na základě aktuálního stavu rezervace. Stavy, které rezervace může mít, vycházejí z aplikace Airbnb. Zde je jejich popis a možnosti, jak se do nich dostat:

1. **Nevyřízená rezervace (Pending).** Pokud je lekce od trenéra nastavena tak, že není její rezervace automaticky potvrzována ze serveru a musí ji trenér potvrdit sám, dostává se její stav ihned po vytvoření rezervace do stavu „Nevyřízená“. To pro sportovce znamená, že musí čekat, než ji trenér schválí a nic víc s touto rezervací nemůže udělat (obrázek 4.13 vlevo). Trenér má naopak možnost rezervaci z tohoto stavu změnit do stavu „Schválená“ nebo „Zamítnutá“ pomocí tlačítek ve spodní části obrazovky (obrázek 4.14 vlevo). Pokud ji trenér do doby konání neschválí, rezervace lekce se dostává do stavu „Zamítnutá“.

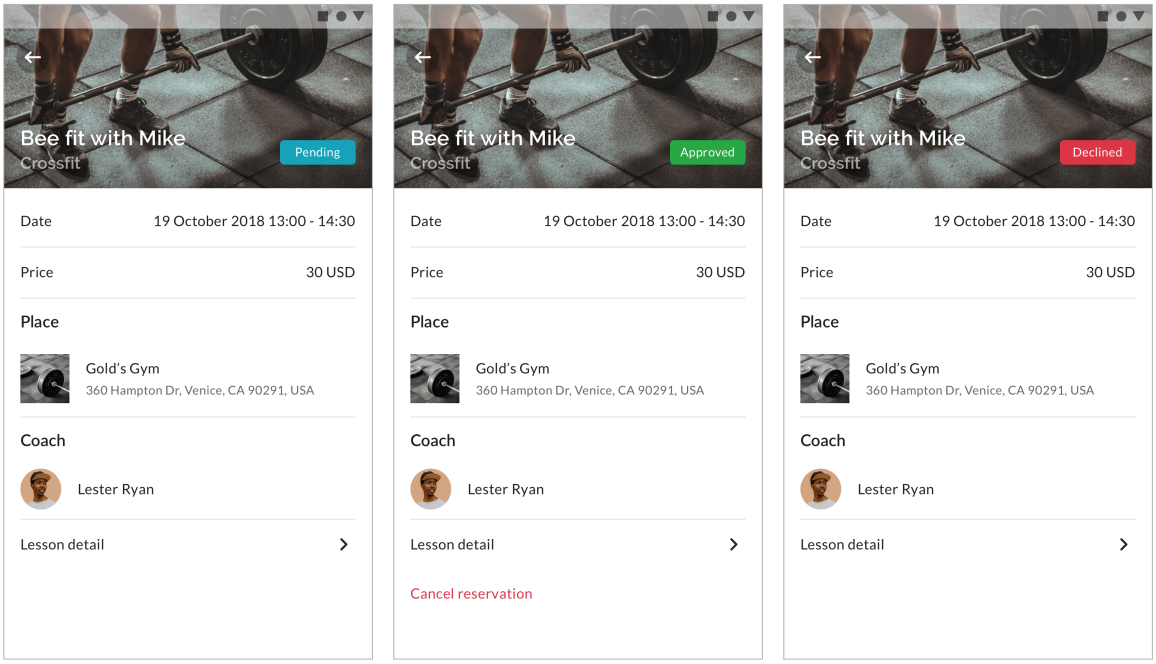
2. **Schválená rezervace (Approved).** Do tohoto stavu se rezervace může dostat ihned po vytvoření rezervace sportovcem, pokud má trenér u lekce nastavené automatické schvalování serverem. Druhý způsob, jak se dostat do tohoto stavu, je možný schválením rezervace ze stavu „Nevyřízená“. Pokud je rezervace schválená, znamená to pro obě role, že lekce se bude konat a sportovec se v daný den a čas má dostavit na patřičné místo (obrázek 4.13 uprostřed a 4.14 uprostřed). Pokud se ale z nějakých nenadálých okolností rezervovaný čas jednomu z nich nehodí, oba mají možnost rezervaci zrušit a dostat ji tak do stavu „Zrušená“.

3. **Zamítnutá rezervace (Declined).** Pokud je rezervace v nevyřízeném stavu a trenér ji nechce přijmout, může ji odmítnout a dostat ji tak do zamítnutého stavu (obrázek 4.13 vpravo a 4.14 vpravo). Následně je pak na sportovci, zda se pokusí rezervaci opakovat pro jiné datum a čas.

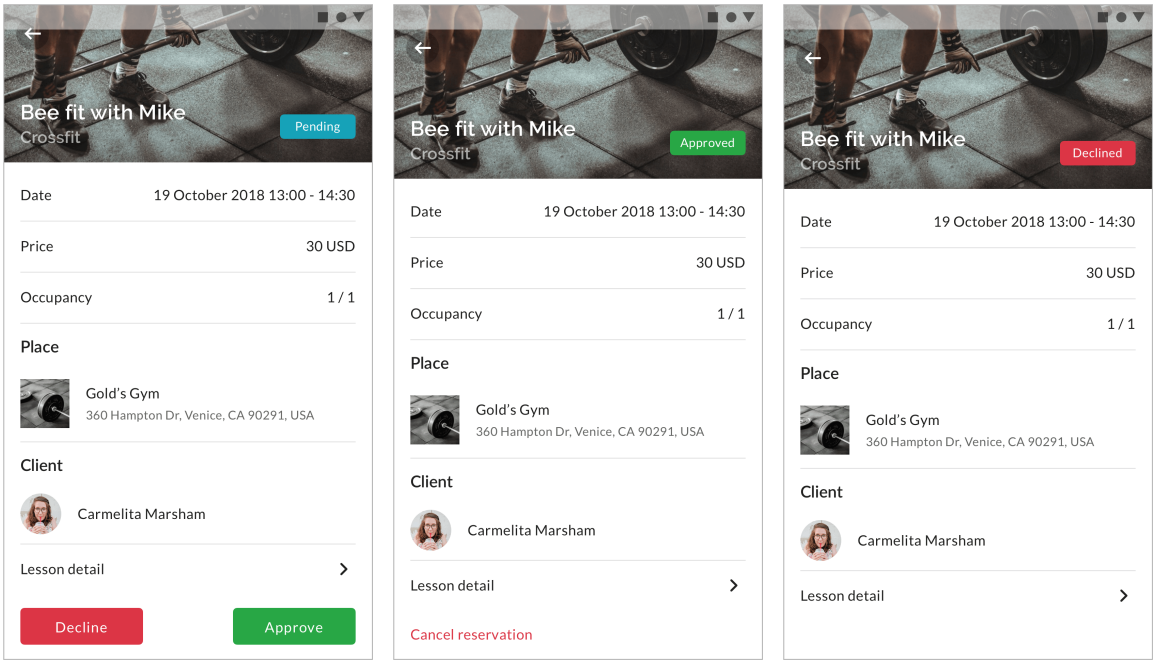
4. **Zrušená rezervace (Canceled).** Zrušená rezervace znamená, že se lekce konat nebude. Do tohoto stavu se rezervace lekce může dostat pouze tehdy, pokud byla předtím schválená a trenér nebo sportovec až potom zjistili, že se nemůžou na lekci dostavit.

4.3.9 Kalendář rezervací

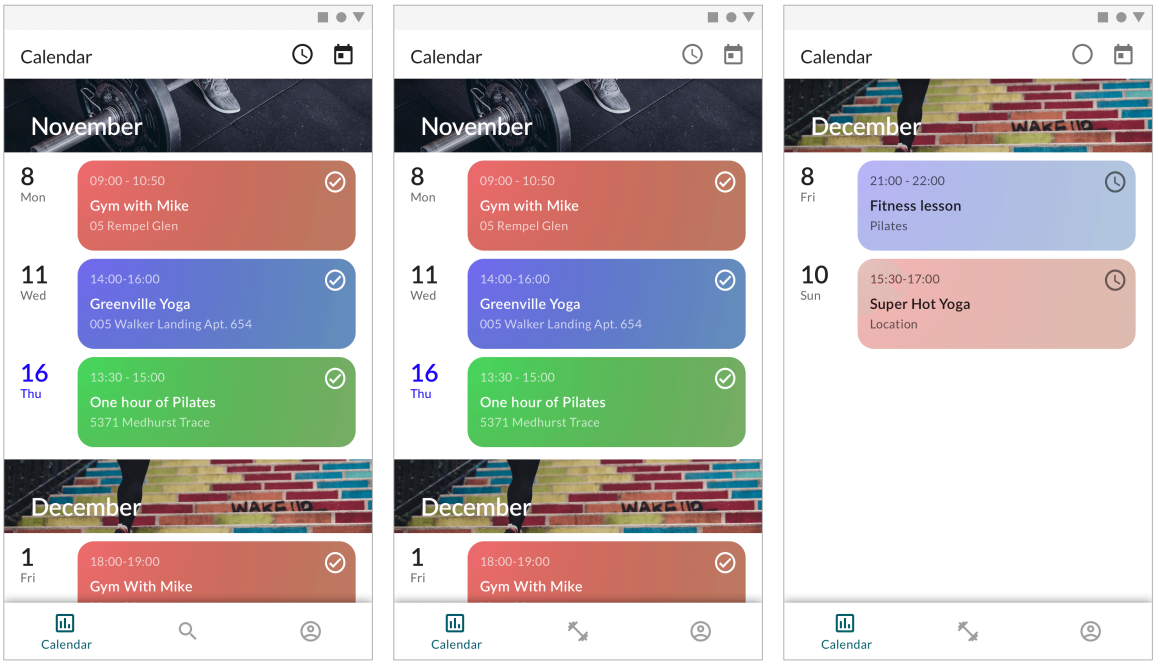
Jak trenér, tak sportovec by měl mít přehled o svých rezervacích. Proto jsem navrhl obrazovku, která obsahuje kalendář rezervací 4.15. Inspiroval jsem se při návrhu velmi známou a používanou aplikací nazývající se Kalendář Google. Seznam rezervací v kalendáři je členěn



Obrázek 4.13: Grafický návrh detailu rezervace pro jednotlivé stavy z pohledu sportovce, **vlevo:** nevyřízená rezervace čekající na schválení trenéra, **uprostřed:** schválená rezervace, která zavazuje sportovce se v daný čas a den dostavit na lekci, **vpravo:** detail zrušené rezervace lekce, která se konat nebude.



Obrázek 4.14: Grafický návrh detailu rezervace pro jednotlivé stavy z pohledu trenéra, **vlevo:** nevyřízená rezervace, kterou může trenér schválit nebo zamítnout, **uprostřed:** schválená rezervace, která zavazuje trenéra se v daný čas a den dostavit na lekci, **vpravo:** detail zrušené rezervace lekce, která se konat nebude.



Obrázek 4.15: Grafický návrh kalendáře rezervací, **vlevo:** kalendář rezervací z pohledu sportovce, **uprostřed:** kalendář rezervací z pohledu trenéra, **vpravo:** filtrování neschválených rezervací z pohledu trenéra.

do jednotlivých měsíčních bloků, které jsou odděleny fotografií a názvem měsíce. Díky tomu se uživatel může přehledněji orientovat. Každý záznam v seznamu představuje datum, název dne, začátek a konec lekce, název a umístění lekce, stav rezervace reprezentovaný ikonkou a nakonec kategorie lekce reprezentována pozadím záznamu.

Rozdíl mezi levým a prostředním návrhem 4.15 je pouze v tom, že levý návrh obsahuje jiné spodní menu a je z pohledu sportovce. Naopak prostřední návrh na obrázku 4.15 je z pohledu role trenéra. Pokud uživatel má v seznamu více rezervací a může se v něm posouvat nahoru a dolů, v pravém horním rohu lišty je ikonka, která při přesunu na rezervace, které jsou v minulosti nebo budoucnu, umožní rychlý návrat na pozici dnešního dne.

Pokud uživatel otevře kalendář rezervací, je mu vždy zobrazena pozice, která představuje aktuální den s rezervacemi nebo nejbližší budoucí den s rezervacemi. Dále může uživatel, pomocí druhé ikonky hodin v horním pravém rohu lišty (pravá obrazovka 4.15), vyfiltrovat pouze takové rezervace, které jsou v nevyřízeném stavu.

4.3.10 Profil uživatele

Protože jsem navrhl vyhledávání trenérů, bylo potřeba vytvořit grafický návrh detailu profilu, který se bude po kliku na trenéra zobrazovat. Nicméně jsem tento detail nakreslil i pro sportovce, který si tak může zobrazovat informace, které vyplnil při registraci a případně některé kontaktní informace doplnit.

Pokud to ale vezmu popořadě, prvně se uživateli ukáže profil (obrázek 4.16 a 4.17 vlevo), kde je možné vidět počet kroků, které jsou potřeba k tomu, aby měl uživatel vyplněné všechny informace o jeho osobě. Dále zde má trenér možnost začít proces vytvoření lekce a pozvat ostatní trenéry do aplikace. Naopak sportovec zde může přejít přímo do vyhledávání lekcí a trenérů nebo může pozvat ostatní sportovce k využívání aplikace. Všichni

uživatelé zde naleznout nastavení, které vede na obrazovku obsahující informace o verzi aplikace, zásady ochrany osobních údajů a možnost se z aplikace odhlásit. Pokud se podíváme zpět na profil (obrázek 4.16 a 4.17 vlevo), v horní části návrhu se nachází jméno uživatele, profilový obrázek a velká neviditelná klikatelná plocha, které uživatele vezme na detail profilu.

Detail profilu (obrázek 4.16 a 4.17 uprostřed) zahrnuje veškeré informace o uživateli. Trenérský detail profilu se zobrazuje sportovcům, kteří chtějí vědět např. kontaktní informace na trenéra. Naopak detailní profil sportovce se zase může zobrazit trenérovi, který na něj přejde z detailu rezervace a může sloužit také jako zdroj pro kontaktní údaje.

Co se týče detailu profilu sportovce (obrázek 4.16 uprostřed), uživatel zde vidí všechny vyplněné informace a pomocí tlačítka v horní částí může tyto údaje upravit (obrázek 4.16 vpravo). Všichni uživatelé v jakékoliv roli mohou vyplnit svoje jméno, email, telefonní číslo a krátký popis o sobě.

Naopak trenér na svém detailu profilu vidí dva taby[1] (obrázek 4.17 uprostřed). Přes první se lze dostat na zobrazení osobních informací. Druhý zobrazí seznam lekcí, které trenér nabízí. Tento seznam je zejména důležitý při zobrazení detailu profilu sportovcem. Pokud se trenér rozhodne o úpravu profilu (obrázek 4.17 vpravo), má zde navíc nastavení sociálních sítí, kde se hodně sportovních trenérů prezentuje buď svými výkony, nebo tím co denně dělají.

Ještě zde chci zmínit, že email, který je možné editovat, je pouze kontaktní. Není to tedy ten email, pod kterým se uživatel přihlašuje. Je to z toho důvodu, že si uživatel sám vybere, jaký email se bude zobrazovat na detailu profilu. Díky tomu může být přihlašovací a kontaktní email jiný. Samozřejmě ale uživatel může pro oba případy zvolit email stejný.

4.3.11 Vytvoření vlastní události

Proces vytvoření vlastní události trenéra byl přidán až v rámci uživatelskému testování popsaného v kapitole 6. Proces obnáší vytvoření vlastní události, která zablokuje dostupnost trenéra v daný den a čas. To zapříčiní, že nebudou lekce trenéra v tento čas dostupné. Situace, kdy bude chtít trenér vytvořit tuto událost, může nastat, když nečekaně nemůže poskytovat v danou dobu svoji vypsanou lekci.

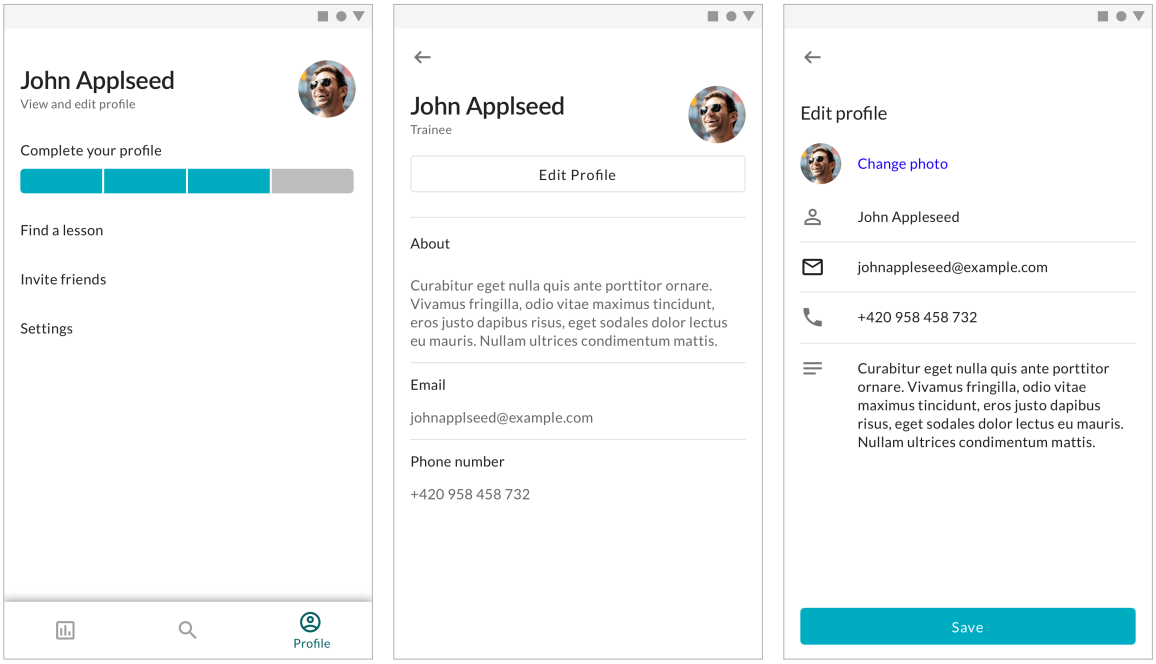
4.4 Představení grafického návrhu trenérům

Po dokončení grafického návrhu aplikace jsem z programu Sketch vyexportoval jednotlivé obrazovky a následně je nahrál do služby Invision. Zde jsem vytvořil veřejný odkaz, který umožňuje návštěvníkovi proklikat se jednotlivými obrazovky tak, jakoby používal aplikaci[2]. Následně jsem oslovil dva trenéry, kteří se na návrh podívali a proklikali si ho. Z jejich zpětné vazby jsem se dozvěděl, že se jim aplikace velmi líbí. Trenéři si ale všimli i dvou nedostatků:

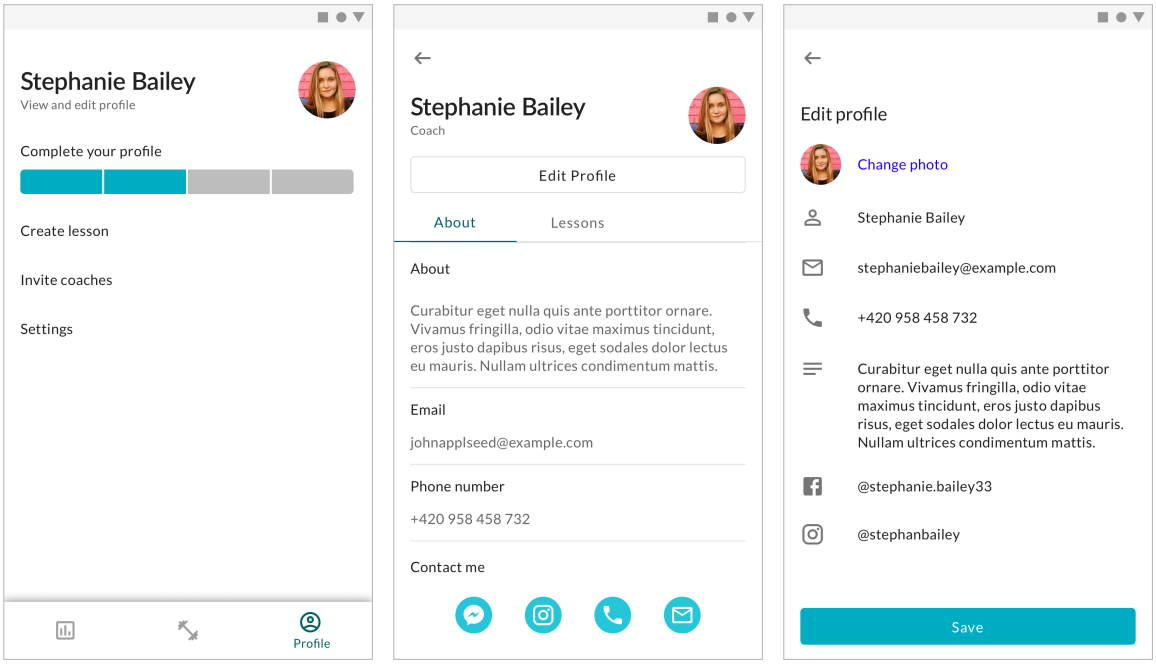
- trenér by měl mít možnost vyhledávat lekce stejně jako sportovec, aby věděl, jak jeho lekce najít a jak vyhledávání funguje a na základě těchto zkušeností vytvářet lepší popisky k lekcím,
- v seznamu lekcí ve vyhledávání by mohla být i cena lekce, která je pro zákazníka důležitá.

[1] Tab umožňuje organizovat obsah do více obrazovek viz. https://material.io/design/components/tabs.html.

[2] Odkaz na interaktivní design https://invis.io/HAOKXIVWTDV.



Obrázek 4.16: Grafický návrh profilu, detailu profilu a úprava profilu sportovce, **vlevo:** přehled akcí, informace o přihlášeném sportovci a počet kroků do vyplnění kompletního detailu profilu, **uprostřed:** detail profilu sportovce, **vpravo:** úprava detailního profilu sportovce.



Obrázek 4.17: Grafický návrh profilu, detailu profilu a úpravy profilu trenéra, **vlevo:** přehled akcí, informace o přihlášeném trenérovi a počet kroků do vyplnění kompletního detailu profilu, **uprostřed:** detail profilu trenéra zahrnující osobní informace a seznam jeho nabízených lekcí, na které se lze dostat přes tab lekce, **vpravo:** úprava detailního profilu trenéra.

Oba zmíněné nedostatky jsem bral v potaz a změny, které byly potřeba udělat, jsem promítl až při implementaci aplikace.

4.5 Architektura MVVM

Model-View-ViewModel je nejnovější architektura podporovaná firmou Google (více informací v kapitole 3.3.3). Návrh použití architektury je možné vidět na obrázku 4.18.

View

Uživatel po celou dobu používání aplikace interaguje s View, což představuje kód, který souvisí s tím, co uživatel vidí. Typicky se jedná o vytvoření nové aktivity, animace, rozmístění komponent v layoutu, zobrazení informací, umožnění interakce s obsahem aktivity, atd. View naslouchá na LiveData, které pocházejí z ViewModel a zároveň může volat funkce z ViewModelu. Pokud se ViewModel dozví o jakékoliv změně dat, na které View naslouchá, provolá to zpětným voláním, díky čemuž View může změny zobrazit uživateli. Proto aby se některý kód ve View stále neopakoval, je možné vytvořit abstraktní třídy. V tomto případě budou použity abstraktní třídy *BaseView* a *AuthenticatedView* (více o jejich implementaci v kapitole 5.1).

ViewModel

ViewModel se stará o jakýsi komunikační most, přes který tečou všechny data do View a zároveň umožňuje volání endpointů serveru a vytváření dotazů do databáze (více o databázi v 4.6) pomocí dalšího rozhraní Repository, nebo zpracování s jiným modelem. Dále se ViewModel stará o zpracování dat na pozadí a vykonává různé zpracovávající operace, které by jinak mohli ve View blokovat interakci s uživatelem. Pro dlouhodobé operace na pozadí, typicky volání na API serveru, se využívá knihovna RxJava, která slouží pro reaktivního programování a je velmi vhodná pro tyto účely (více v podkapitole 3.3.7).

Model a Repository

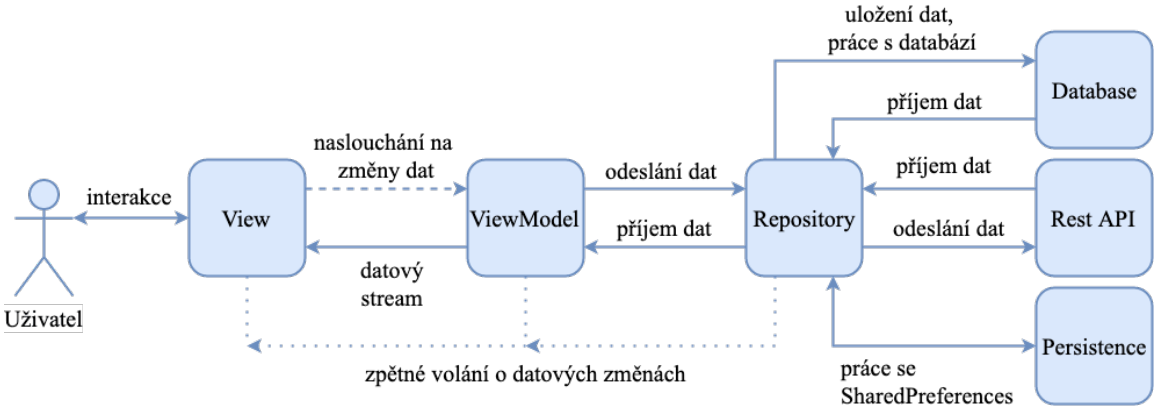
Repository představuje jeden z možných modelů, které se používají. V aplikaci budou další modely, které se například starají o zpracování Firebase tokenu, práci s Google API, apod. Nicméně Repository je hlavní stavební kámen pro práci s databází a se serverem. Pro každou entitu databáze existuje jeden Repository, který se stará o všechny funkce, volání a zpracování týkající se této entity. Všechno tedy spojené s danou entitou teče přes jeho Repository. Znovu, jako u View, zde bude jedna abstraktní třída *BaseRepository*, která bude obsahovat metody, které se využívají u každé entity databáze.

Vrstva Database je tvořena třídy, které přestavují entity a nastavení databáze. Pro získávání dat z databáze je možné využít knihovnu RxJava nebo objektu LiveData. Obojí je velmi užitečné a záleží na potřebě použití.

Dále Repository komunikuje s třídou REST API, která přestavuje implementaci volání dotazů na server pro získání či aktualizace dat nebo pro patřičné zpracování, které vyvolá nějakou akci nebo odpověď.

Nakonec Repository využívá Persistance vrstvu pro obsluhu sdílených dat v rámci aktivit nebo instance celé aplikace, které není potřeba ukládat do databáze.

Velkou výhodou pro takto navrženou architekturu je hlavně škálovatelnost. Další velkou výhodou na rozdíl od architektury Model-View-Presenter (dále MVP), je použití LiveData.



Obrázek 4.18: Diagram komunikace a spolupráce jednotlivých vrstev architektury MVVM.

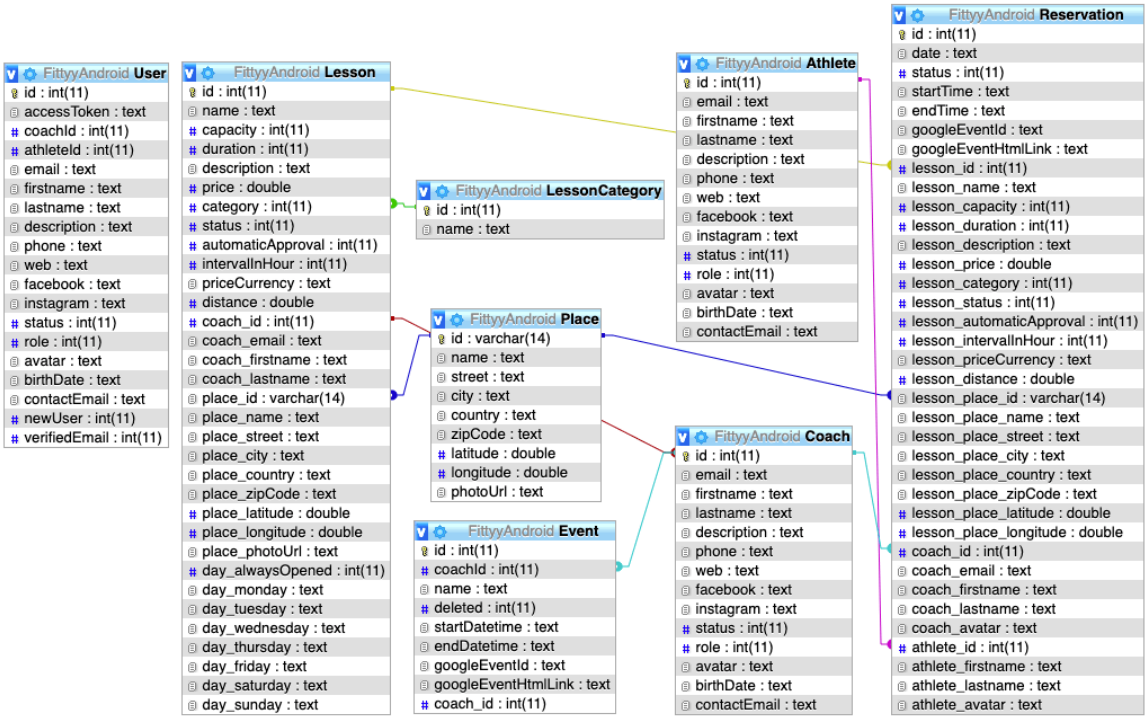
U MVP bylo potřeba vždy zavolat metodu a čekat na odpověď. Jakmile byla odpověď k dispozici, následovalo nutné navrácení dat zpět přes všechny vrstvy až do View. To u MVVM není potřeba díky LiveData. Stačí pouze zavolat získání dat a nastavit naslouchání na LiveData. Až výsledek bude k dispozici, předají se data pomocí zpětného volání sama. Poslední výhodou o proti MVP je, že při použití architektury MVVM využijeme sdíleného ViewModelu pro vícero fragmentů. Díky tomu nemusíte předávat mezi fragmenty data, protože budou přes ViewModel sdílena.

4.6 Lokální databáze

Vzhledem k tomu, že uživatel nemusí mít stále dostupné připojení k internetu, bude v aplikaci využita lokální databáze, která bude důležité informace ukládat. Díky tomu budou data dostupné i v offline režimu. Na Androidu je více možností, jak databázi vytvořit. Ve Fittyy bude využita SQLite databáze od Google nazývající se Room. Room je knihovna umožňující práci s databází jako s objekty. Její výhodou je možnost získat data z databáze zapouzdřené v Livedata.

Co se týče návrhu databáze, tak jsem již v této fázi mohl dle grafického návrhu zanalyzovat, jaké entity a atributy bude obsahovat. Pro návrh databáze jsem vytvořil entitně vztahový diagram (dále ERD), který je na obrázku 4.19. ERD je vhodný pro abstraktní a konceptuální znázornění databázových dat. Jednotlivé vztahy mezi entitami jsou nakresleny barevnými čárami. Jak jsem ale později zjistil, musel jsem pojmout práci s knihovnou Room trochu z jiného pohledu. V rámci vývoje jsem nevytvořil žádné reálné vztahy mezi entitami. Důvod, který mě k tomu vedl, nastíním v následujícím odstavci.

Představme si situaci, kdy ze serveru chodí např. objekt rezervace. Její data musí obsahovat údaje, kdy se daná rezervace vykoná, o jakou lekci jde, jaký trenér lekci vlastní a jaký sportovec rezervaci vytvořil. To nám dává dohromady tři subentity. Proto, aby server vždy neposílal úplně všechny data, rezervace bude obsahovat pouze informace, které jsou potřeba pro zobrazení na dané obrazovce, neboť data potřebná pro vykonání akce související s rezervací. Následně po obdržení dat mobilní aplikací se vše uloží do databáze. Pokud bych ale jednotlivé subentity provázal k entitě rezervace přes cizí klíče, po uložení do databáze by se informace subentit přepsala, čímž by se uložily jen ty informace, které byly přijaty a o ostatní by aplikace přišla. Toto je důvod, proč jsem nakonec neimplementoval vztahy mezi entitami. Knihovna Room má pro tyto účely speciální konstrukci



Obrázek 4.19: Diagram návrhu databáze mobilní aplikace Fittyy.

`Embedded(prefix="lesson_") var lesson: Lesson? = null`, která v tomto případě vytvoří kopii atributů subentity lekce, kde jsou jejich názvy doplněné o prefix. To lze vidět v diagramu na obrázku 4.19.

Popis entit

`User` představuje entitu uživatele, pro kterou v databázi existuje vždy pouze jediný záznam, protože reprezentuje data o přihlášeném uživateli. Díky tomu v rámci celé aplikace lze např. zjistit identifikační číslo (dále ID) nebo jméno přihlášeného uživatele. To se např. hodí pro ověření, zda zobrazovaná lekce patří přihlášenému trenérovi. Entita `Coach` představuje trenéra a `Athlete` sportovce. Oba mají pro jednoduchost stejné atributy. Spíše se vyskytují jako subentity např. v rezervaci. `Lesson` reprezentuje detailní informace o lekci. Obsahuje všechny údaje, které trenér při vytváření vyplnil a referenci na trenéra a jeho informace. `LessonCategory` představuje kategorii lekce, která obsahuje atribut název a identifikačního číslo. `Place` je entita místa vycházející z obsahu objektu, který vrací Google Places API. Vyskytuje se jako subentita v lekci, kde reprezentuje místo konání. Rezervace jsou reprezentovány entitou `Reservation` obsahující nejdůležitější informace potřebné pro sportovce, který se musí dostat do místa konání v daný den a čas. Dále rezervace obsahuje trenéra, kterému lekce patří, sportovce jenž rezervaci vytvořil a nakonec informace o lekci, na kterou je rezervace vytvořená. Celkově by tyto entity databáze měly pokrýt potřebné informace zobrazené na obrázcích, které byly vytvořeny v rámci grafického návrhu aplikace.

4.7 Návrh serveru

Pro implementaci serveru a REST API bude použit webový systém Dactyl CMS[3] vyvinutý společností Dactyl Group. Jedná se odladěný systém, který slouží jako základ pro webové stránky, e-shop, backend mobilních aplikací a případně i pro vytvoření webových aplikací na míru. Jednotlivé zmíněné způsoby použití jsou rozděleny v CMS do modulů. V rámci této práce budou využity moduly:

- **console** — cli[4] využívaná pro opakované události spouštěné na serveru (např. vždy o půlnoci se spustí nějaký skript) nebo pro migraci databáze na novější verzi,
- **api** — REST API potřebné pro komunikaci mobilní aplikace se serverem,
- **common** — bude obsahovat třídy reprezentující databázové entity a případně implementaci funkcí potřebných ve více modulech.

Dále jsou v základu Dactyl CMS tři rozšíření, které se budou pro mobilní aplikaci využívat:

- **core** — rozšíření Yii2[5] frameworku, definice vlastních tříd a rozhraních použitých v celém projektu,
- **dc-user** — definice uživatelů a jejich rolí,
- **dc-language** — překlad řetězců v aplikaci a chybových kódů využitelné např. pro překlad kategorií lekcí.

Dactyl CMS mi tak poskytne základní funkcionalitu, na které lze stavět kvalitní spolehlivé REST API. Dále jsem navrhl seznam komunikačních bodů (endpointů), které jsou potřebné pro komunikaci mobilní aplikace se serverem. Přes tyto komunikační body může aplikace posílat HTTP požadavky s určitými daty, nebo získávat HTTP odpovědi obsahující data. Seznam navržených endpointů:

- `POST /user/register` — registrace uživatele,
- `POST /user/login` — přihlášení uživatele,
- `POST /user/logout` — odhlášení uživatele,
- `POST /user/update` — aktualizace informací přihlášeného uživatele,
- `POST /user/device` — aktualizace informací o zařízení přihlášeného uživatele,
- `GET /user/status` — získání informací o přihlášeném uživateli,
- `POST /lesson` — vytvoření nové lekce,
- `GET /lesson/id` — získání informací o lekci,
- `POST /lesson/id` — aktualizace lekce,

[3] Content Management System je systém pro správu obsahu.
[4] Command Line Interface je příkazový řádek umožňující uživateli zadávání příkazů.
[5] Yii2 je PHP framework, oficiálně podporovaný firmou JetBrains, umožňující vytvořit bezpečný, rychlý a efektivní CMS systém [21].

- `DELETE /lesson/id` — smazání lekce,
- `POST /lesson/id/availability` — získání dostupnosti lekce,
- `POST /lesson/list` — seznam lekcí možný filtrovat dle ID trenéra, vzdálenosti od aktuální polohy uživatele nebo názvu lekce,
- `POST /coach/list` — seznam trenérů možný filtrovat dle jména trenéra,
- `GET /coach/id` — získání informací o trenérovi,
- `GET /player/id` — získání informací o hráčovi,
- `GET /misc/sync` — synchronizace kategorií lekce a měny.

GET se využívá pro získání dat ze serveru. *POST* je vhodný když na server data posíláme. V některých případech data po serveru žádám, ale v návrhu je použit *POST*. Je to kvůli tomu, že v těle žádosti se posílá objekt obsahující data pro filtrování výsledků. Všechny navržené endpointy (kromě registrace a přihlášení) jsou dostupné pouze tehdy, pokud mobilní aplikace zašle v hlavičce přístupový token, který získá po úspěšném přihlášení uživatele. Token bude šifrovaný a jedinečný v celém systému, tak aby kvůli bezpečnosti identifikoval pouze daného uživatele na daném zařízení.

V rámci návrhu jsem vytvořil další entitně vztahový diagram, který je v příloze A. Na základě tohoto diagramu je možné zjistit jaké vztahy mají jednotlivé entity a jaké atributy budou mít. Z velké části to vychází z návrhu databáze aplikace popsaný v kapitole 4.6.

Kapitola 5

Implementace

Druhou nejdůležitější částí této práce je implementace celého rezervačního systému v mobilní aplikaci a na serveru. Proto se tato kapitola zabývá popisem jednotlivých částí, které byly potřeba udělat pro vytvořené finálního systému. Jsou zde popsány způsoby technologií, služeb, knihoven a vědomostí, které byly získány v rámci studia návrhu a vývoje mobilních aplikací na Android z kapitoly 3. Dále jsem využil poznatky a informace z procesu návrhu aplikace a serveru popsané v kapitole 4.

5.1 Architektura aplikace

Kompletní mobilní aplikaci jsem vyvíjel na základě architektury Model-View-ViewModel, která je oficiálně podporovaná a doporučená firmou Google. Jednotlivé vrstvy ale bylo potřeba nejdříve propojit. K tomu jsem využil knihovny Dagger, která se používá k implementaci vzájemných závislostí mezi třídami resp. vrstvami MVVM. Hlavním článkem je `AppComponent`, kam je potřeba zapsat všechny moduly, které chceme přes Dagger poskytovat. Toto rozhraní je zinicializováno ihned při startu mobilní aplikace. Dále je potřeba vytvořit moduly, které obsahují jednotlivé metody poskytující instance jednotlivých tříd resp. jedináčků. Jako příklad jedináčků uvádím `provideRoomDatabase` a `provideUserRepository` ve výpisu 7. Takový typ metody pak poskytuje danou instanci v rámci celé aplikace. Není tedy potřeba kdekoliv, kde danou třídu potřebujeme, vytvářet novou instanci. To se například hodí při neustálém využívání instance třídy databáze. Následně pokud chceme využít některou z instancí, je potřeba provést tzv. injekci pomocí `@Inject`, která nám umožní využít metody poskytující dané instance.

Pokud chceme Dagger využít pro poskytování ViewModel ve View, je potřeba pro každý ViewModel vytvořit modul, který tento ViewModel poskytuje (`MainModule` ve výpisu 7). Následně je pak potřeba zadefinovat, že daná aktivita/fragment chce využít daný modul, který poskytuje ViewModel (`bindMainActivity` ve výpisu 7).

Pro lepší pochopení uvádím ukázku kódu, který znázorňuje provázanost využití modulů obsahující metody, které poskytují jedinečnou instanci databáze a repozitáře uživatele v rámci celé aplikace. Dále je zde zobrazena definice `MainViewModel`, který se využívá u aktivity `MainAcitivty`. Tato aktivita se stará o navigaci mezi fragmenty obrazovky kalendáře rezervací, vyhledávání lekcí, seznam lekcí trenéra a profilu uživatele. ViewModel obsahuje pouze implementaci metody starající se o synchronizaci kategorie lekcí ze serveru. Ukázka neobsahuje kompletní kód a zároveň její obsah byl sjednocen z vícero zdrojových souborů.

```
/* Sekce pocházející z AppModule, která je definovaná v AppComponent */
@Provides
@Singleton
internal fun provideRoomDatabase(): FittyyDatabase {
  return Room.databaseBuilder(application,
                              FittyyDatabase::class.java,
                              DB_NAME)
      .allowMainThreadQueries()
      .addMigrations(migrateFrom1to2())
      .build()
}
@Provides
@Singleton
internal fun provideUserRepository(api: FittyyRestApi,
      db: FittyyDatabase, persist: PersistenceInterface,
      deviceTracker: DeviceTracker, retrofit: Retrofit, moshi: Moshi,
      googleCalendarInterface: GoogleCalendarInterface): UserRepository {
  return UserRepository(api, db, persist, deviceTracker,
          googleCalendarInterface, retrofit, moshi)
}

/* Speciální modul poskytující ViewModel */
@Module
abstract class MainModule {
  @Binds
  @IntoMap
  @ViewModelKey(MainViewModel::class)
  abstract fun bindsMainViewModel(viewModel: MainViewModel): ViewModel
}

/* Definice která aktivita používá jaký modul resp. ViewModel */
@ContributesAndroidInjector(modules = [MainModule::class])
abstract fun bindMainActivity(): MainActivity

/* Konstruktor pro ViewModel musí obsahovat @Inject pro získání modelů */
class MainViewModel @Inject constructor(userRepository: UserRepository,
      db: FittyyDatabase) : AuthenticatedViewModel(userRepository, db)
```

Výpis 7: Ukázka z implementace závislostí pomocí knihovny Dagger umožňující použití instance databáze a repozitáře uživatele v `MainViewModel`.

5.1.1 View

View představuje v aplikaci aktivitu nebo fragment reprezentující uživatelské rozhraní, se kterým uživatel interaguje. Jedná se o inicializaci obrazovky, nastavení interakce tlačítek, naslouchání na LiveData apod. Protože se na všech obrazovkách opakuje inicializace View-

Modelu nebo různých základních nastavení, vytvořil jsem abstraktní třídu `BaseActivity` a `BaseFragment`, ze kterých obrazovky dědí. Pro obrazovky, kde musí být uživatel přihlášen, jsem vytvořil ještě abstraktní třídy `AuthenticatedActivity` a `AuthenticatedFragment`, které v případě obdržení chyby autorizace ze serveru provádí určité kroky, které vedou k ukončení a zpětnému navrácení do obrazovky s přihlášením. Díky abstraktní implementaci můžu změnit část chování obrazovek hromadně a to na jednom místě. Zde je seznam nejzajímavějších metod, které jsou obsaženy ve zmíněné abstraktní třídě *AuthenticatedActivity*:

- `initView` — abstraktní metoda volaná v `onCreate`, nutno ji vždy implementovat v podtřídě, je vhodná pro nastavení interakce tlačítek a ostatních komponent,
- `bindViewModel` — abstraktní metoda volaná v `onCreate`, musí být vždy implementována, slouží pro účely nastavení naslouchání na LiveData,
- `showProgressDialog` — implementace zobrazení dialogu, který obsahuje progressbar, funkce se volá při každém volání na server,
- `showErrorSnackbar` — funkce pro zobrazení chyby nejčastěji ze serveru,
- `createViewModel` — zařídí, že na každé obrazovce se automaticky vytvoří reference na ViewModel,
- `onUnauthorizedError` — tato funkce se volá, pokud ze serveru přijde chybová hláška, když uživatel není autorizován, jedná se o abstraktní metodu nutnou implementovat na každé obrazovce.

Zjednodušeně řečeno můžu např. na kterékoliv obrazovce, která je podtřídou zmíněné třídy, zobrazit progress dialog nebo chybovou hlášku pouhým zavoláním jedné funkce, aniž bych musel na každé obrazovce implementovat, jak se mají tyto informace zobrazit.

Jednou z důležitých částí aplikace, kterou zde musím zmínit, je způsob, jakým se naslouchá na LiveData. Proto zde uvádím ukázku kódu ve výpise 8 pocházející z abstraktní třídy `AuthenticatedActivity`. Naslouchá se na LiveData, kde `errorState` reprezentuje hodnotu třídy `ErrorIdentification` a `progressState` představuje hodnotu `true` nebo `false` typu `Boolean`. V prvním případě jde o zobrazení chybové zprávy tzv. snackbaru vysouvající se ho ze spodní části obrazovky. V případě druhém se zobrazí v dialogu pouze kulatý progressbar značící delší operaci na pozadí (typicky práci se serverem).

Navigace mezi fragmenty

Architektura MVVM se velmi často používá způsobem, kdy je vytvořena jedna hlavní aktivita s jedním view modelem. Odtud se pak vytváří vícero fragmentů využívající stejný ViewModel, tak aby mohli navzájem sdílet data. Pro tyto účely Google vytvořil Navigation Component, jenž zjednodušuje navigaci mezi jednotlivými fragmenty. Avšak tato knihovna byla v době vývoje ještě nestabilní. Proto jsem vytvořil vlastní princip navigace mezi fragmenty, který jsem využil při uvítací obrazovce, procesu vytváření lekce a při rezervaci lekce. Pro každý případ jsem vytvořil třídu jedináčka jenž je do View dopravena pomocí Daggeru.

Navigace je reprezentována vždy daty, na které lze naslouchat v aktivitě, a dle potřeby reagovat na nový stav. Nový stav nastane, když v nějakém fragmentu zavolám přes instanci navigace zobrazení nové obrazovky reprezentující fragment. Díky tomu, že aktivita naslouchá na tyto stavy, zjistí jaká obrazovka se má zobrazit a provede danou operaci zobrazení.

```
private fun bindBaseStates() {
  viewModel.errorState.observe(this, Observer { error ->
    if (error !is ErrorIdentification.None) {
      showErrorSnackbar(error.message)
      viewModel.errorState.postValue(ErrorIdentification.None())
    }
  })
  viewModel.progressState.observe(this, Observer { visibility ->
    showProgressDialog(visibility)
  })
}
```

Výpis 8: Implementace naslouchání na LiveData, které slouží pro zobrazování chybových hlášek a dialogu s progressbarem na obrazovce.

V rámci procesu vytváření lekce mi to navíc dalo možnost přenášet datový atribut, který udával, zda se nová obrazovka zobrazuje z obrazovky shrnutí nebo ne. Na základě toho jsem se v nové obrazovce mohl rozhodnout, jestli zobrazit tlačítko s akcí dále nebo uložit.

5.1.2 ViewModel

ViewModel slouží pro zpracování, získávání a posílání dat. Definují se zde funkce, které volají primárně metody pouze z repozitářů nazvaných Repository. Dále je to jediné místo, kde se správně deklarují LiveData. Stejně jako u View jsem zde implementoval abstraktní třídy nazývající se `BaseViewModel` a `AuthenticatedViewModel`, ze kterých ostatní ViewModel vycházejí (příklad na konci ukázky 7). V těchto abstraktních třídách se např. vytvářejí LiveData pro chybový stav a zobrazení progress dialogu 9. Zde bych ještě zmínil, že existují *LiveData* a *MutableLiveData*. První typ nelze za běhu programu nijak přepsat a měnit. Zatímco slovo „Mutable“ v Kotlinu značí, že data jsou modifikovatelná. Nemodifikovatelné LiveData jsou typické při získávání z databáze. Díky čemu se jakákoliv pozdější změna v databázi dané entity provolá do View, pokud se na LiveData zde naslouchá.

```
val progressState = MutableLiveData<Boolean>().apply {
  postValue(false)
}
```

Výpis 9: Ukázka definice MutableLiveData ve view modelu `AuthenticatedViewModel`.

Nejčastější implementací ve view modelu je práce s databází nebo s REST API. V obou případech se práce musí provádět na pozadí. Pro tyto účely jsem využil knihovny RxJava, která umožňuje naslouchat a čekat na odpověď ze serveru nebo databáze. Na ukázce 10 je využita třída *Single*, která umožňuje naslouchat změnám stavu volání na server. Naslouchání je prováděno asynchronně na pozadí, ale poté obsluhováno na popředí v hlavním vlákně aplikace. Tyto dvě nastavené věci zde nejsou vidět, protože jsou definovány už v rámci funkce pocházející z `ReservationRepository` (více na ukázce 11). V rámci view modelu se

pouze řeší, zda požadavek na data ze serveru nevrátí nějakou chybu a pokud ano, chyba se zobrazí přes MutableLiveData `errorState`.

```
fun syncReservationDetail(reservationId: Long) {
  reservationRepository.syncReservationDetail(reservationId)
      .doOnSuccess { resource ->
        if (resource.status is ErrorStatus) {
          errorState.postValue(resource.errorIdentification)
        }
      }
      .subscribe()
  }
```

Výpis 10: Ukázka implementace funkce view modelu, kde se využívá knihovny RxJava pro synchronizaci detailních informací rezervace ze serveru.

5.1.3 Model

V následující podkapitole popisuji použití různorodých modelů, které se využívají pro komunikaci se serverem nebo pro zpracování a ukládání dat. Bez těchto modelů by aplikace nebyla funkční.

Repository

Na ukázce 11 je využita třída *Single*, která umožňuje naslouchat na změny průběhu volání dotazu na server. Naslouchání je prováděno asynchronně na pozadí, ale poté obsluhováno na popředí v hlavním vlákně aplikace. Pokud požadavek na data uspěje a server nevrátí žádnou chybu, data jsou aktualizována v lokální databázi aplikace.

Ukázka 11 také znázorňuje, že třída *Single* musí obalovat návratový typ, který očekáváme např. ze serveru. Tento způsob umožňuje Kotlin a označuje se jako použití parametrů obecného typu. Díky tomu může být *Single* využit pro jakýkoliv návratový typ. V případě ukázky 11 se očekává odpověď `ReservationResponseData`, která obsahuje objekt rezervace a je zapouzdřená v *Resource*.

Třída *Resource* je převzata přímo od vývojářů firmy Google. Mírně jsem ji modifikoval na potřeby Fittyy aplikace, kde ji využívám převážně pro dva účely:

1. **Reprezentace stavu odpovědi ze serveru.** Komunikace se serverem funguje na principu potvrzování úspěšného zpracování požadavku. Stav, zda se zpracování povedlo, je reprezentován kódem, který přijde v hlavičce. Na základě toho Repository, která je podtřídou `BaseRepository`, zpracuje odpověď ze serveru a transformuje ji pomocí funkce `processRequestResponse` na vnitřní reprezentaci *Resource*. Následně se jednoduše např. ve ViewModelu může zjistit, zda odpověď značí stav úspěchu nebo chyby.

2. **Stav *NotStartedStatus*.** Při zapnutí aplikace je potřeba zjistit, zda je uživatel přihlášený nebo ne. K tomu využívám kombinování dvou LiveDat a transformaci na jedny LiveData obsahující uživatele obalený právě třídou Resource (ukázka 12). Je to z toho

```
fun syncReservationDetail(reservationId: Long)
  : Single<Resource<ReservationResponseData>> =
    api.syncReservation(reservationId.toString())
        .subscribeOn(Schedulers.io())
        .observeOn(AndroidSchedulers.mainThread())
        .processRequestResponse()
        .doOnSuccess { resource ->
          if (resource.status is SuccessStatus
                && resource.data != null) {
            db.reservationDao().replace(resource.data.reservation)
          }
        }
```

Výpis 11: Implementace asynchronního volání na server pro získání informací týkající rezervace určené parametrem `reservationId`. Využívá se zde RxJavy a třídy Single, která obaluje reprezentaci odpovědi.

důvodu, že se kombinují LiveData uloženého uživatele v databázi a přístupového tokenu na server. Pokud by uživatel v databázi existoval, ale přístupový token ještě ne, transformace provede vytvoření Resource se stavem *NotStartedStatus*. Pokud se následně stav uživatele nebo tokenu změní, tato funkce to zachytí a promítne změny dále. V případě kdy jsou obě podmínky `it.second != null` (tzn. existuje uživatel) a `it.first.defined` (tzn. existuje token) na ukázce 12 splněny, `loggedUser` je reprezentován stavem *SuccessStatus* obsahující data přihlášeného uživatele.

Celkem má třída `Resource` následující stavy:

- **NotStartedStatus** — stav, kdy data nejsou ještě k dispozici,
- **SuccessStatus** — úspěšný stav doplněný většinou i o data,
- **ErrorStatus** — chybový stav reprezentovaný třídou *ErrorIdentification*,
- **LoadingStatus** — získávání data je v procesu.

Pro nejčastější chyby jsem vytvořil jejich reprezentaci v *ErrorIdentification*. Každá chyba je doplněna o zprávu, která se má v případě problému objevit na obrazovce. Díky tomu uživatel ví, v čem je problém a případně mně může tuto chybu nahlásit.

Důvodem vytvoření repozitáře je práce s databází a celkově celé logiky zpracování každé entity databáze. Proto jsem repozitář vytvořil pro každou entitu databáze. Každá třída repozitáře je podtřídou `BaseRepository`, kde jsem implementoval základní funkce pro práci s danou entitou, které komunikují s *Dao* rozhraním určité entity. Pro představu jsem zde vytvořil pomocí RxJavy asynchronní funkce, které pracují s rozhraním databáze, jako je např. vložení seznamu záznamů, smazání záznamu, aktualizace nebo přepsání daného záznamu, atd.

Pokud se vrátím zpět k implementaci určité třídy Repository pro danou entitu, vždy jsem využíval buď knihovnu RxJava pro získání dat z databáze na pozadí, nebo získání dat reprezentovaných třídou LiveData.

```
val loggedUser: LiveData<Resource<User>> =
    Transformations.map(
      combineLatestLiveData(persist.getAccessTokenLiveData(),
                            db.userDao().getUserLiveData())) {
        return@map when {
          it.first.defined && it.second != null -> {
            Resource.success(it.second)
          } else -> {
            Resource.notStarted(User())
          }
        }
      }
```

Výpis 12: Kombinace LiveData přihlášeného uživatele a přístupového tokenu, které se transformují pouze na LiveData uživatele, který bude zapouzdřen v *Resource* reprezentující, zda je uživatel vážně přihlášen nebo ne.

Přístup pomocí RxJava jsem využil, pokud jsem data chtěl získat z databáze pouze na poprvé a na další jakékoliv změny jsem reagovat nechtěl. Tudíž se při této operaci naslouchá pouze do konce zpracování prvního výsledku z databáze. Naopak LiveData jsem využil pokaždé, když jsem chtěl ve View mít neustále aktuální data. To se například hodilo v detailu rezervace lekce. Prvně jsem z lokální databáze získal LiveData obsahující informace o rezervaci a naslouchal na ně ve View, kde se na každé provolání LiveData provedla určitá funkce pro naplnění informací do grafických komponent. Mezi tím jsem zavolal na server získání nových informací týkajících se této rezervace. Jakmile byl požadavek úspěšný, stačilo pouze nové data uložit do lokální databáze. Díky naslouchání na LiveData ve View se změny po uložení promítly i na obrazovku.

Database

Vytvoření a správu databáze zajišťuje knihovna Room z Android Jetpacku. S daty z databáze se pracuje jako s objekty a nejčastěji jsou obaleny LiveDaty. Každá entita má vlastní třídu reprezentující informace, které obsahuje a vlastní *DAO*[1] rozhraní. Toto rozhraní slouží pro implementaci funkcí, které zajišťují přímou práci s relační tabulkou. Zde jsem znovu využil dědičnosti a vytvořil rozhraní `BaseDao`, které obsahuje základní metody umožňující vložení, smazání nebo aktualizace záznamu.

Na ukázce 13 je možné vidět implementaci rozhraní `UserDao` využívané pro entitu přihlášeného uživatele. První dvě funkce vrací přihlášeného uživatele. Rozdílné jsou pouze ve způsobu, jakým to provedou. V prvním případě jde o *Single* z RxJavy, který data vrátí pouze jednou a pokud žádné data neexistující, skončí proces s chybou. Naopak druhá metoda vrací LiveData, na která lze naslouchat ve View, díky čemu je provolána každá změna v tabulce *User*. Pokud záznam v databázi neexistuje, LiveData nejsou nikdy provolána. Třetí funkce ve výpisu 13 provede smazání všech záznamů v tabulce uživatelů.

[1] Data Access Object je objekt představující abstraktní rozhraní pro určitou entitu databáze.

```
@Dao
interface UserDao : BaseDao<User> {

  @Query("SELECT * FROM User LIMIT 1")
  fun getUserSingle(): Single<User?>

  @Query("SELECT * FROM User LIMIT 1")
  fun getUserLiveData(): LiveData<User?>

  @Query("DELETE FROM User")
    fun deleteAll()
  }
}
```

Výpis 13: Ukázka vytvořeného *Dao* rozhraní databázové entity *User*, které se využívá pro práci s tabulkou uživatelů.

Persistance

Pro práci se serverem je potřeba přístupového tokenu, který ukládám v tzv. *SharedPreferences*, což jsou sdílená data přístupná pouze pro aplikaci. Dále zde ukládám Firebase token, který se využívá pro Firebase Messaging (více v 5.1.4) nebo ukládání aktuální zeměpisné šířky a výšky uživatele, která se využívá při vyhledávání lekcí. Pro správu sdílených dat jsem vytvořil Persistance třídu, která umožňuje jejich rychlé získání, aktualizaci nebo smazání.

Google Calendar

Jeden z modelů, které jsem přidal až v rámci testování popsaném v kapitole 6, je *GoogleCalendarModel* využívaný pro komunikaci s Google Calendar API. Model pro komunikaci využívá knihovnu, která byla přidána za účelem synchronizace schválených rezervací uživatele s jeho Kalendářem Google. Synchronizace byla možná buď přístupem synchronizace s lokálním kalendářem a vytvoření synchronizačních adaptérů, nebo pomocí zmíněné využité knihovny, která zjednodušuje komunikaci s jejich API.

Při použití knihvony pro komunikaci s API bylo potřeba v prvním kroku, aby aplikace vyžádala propojení uživatelova Google účtu s aplikací Fittyy. Jakmile uživatel zvolí účet, aplikace si vyžádá přístupový token do Google Calendar API. Ve druhém kroku musí uživatel vybrat Google kalendář, se kterým chce rezervace synchronizovat. Jakmile jej vybere, služba je nastavena.

Synchronizace rezervací se provádí vždy, když uživatel zobrazí kalendář rezervací. V tento moment se získají rezervace ze serveru, aktualizují se v lokální databázi a následně se vytvářejí události v Google kalendáři. Pokud vytvoření proběhne úspěšně, uloží se identifikační klíč vytvořené Google události k dané rezervaci a tato informace se zaktualizuje na serveru. Díky tomu aplikace ví, které schválené události jsou již zesynchronizovány. Pokud se uživatel rozhodne synchronizaci vypnout, může to udělat jednoduše v nastavení aplikace kliknutím na přepínač, který se přesune do vypnutého stavu.

5.1.4 Firebase Messaging

Během uživatelského testování popsaného v kapitole 6 vyšlo najevo, že trenéři postrádají notifikační upozornění o nových rezervacích nebo jejich změnách. Proto jsem do aplikace a na server přidal *Firebase Messaging* zajišťující tuto funkcionalitu. Službu je potřeba nastavit na webové stránce služby Firebase[2], odkud se stáhne konfigurační soubor a přístupové kódy pro komunikaci s Firebase API.

Ze strany aplikace je potřeba přidat knihovnu, konfigurační soubor a jednu servisní třídu, která zpracovává notifikace. Dále je potřeba ihned po registraci nebo přihlášení uživatele odesílat na Fittyy server přístupový Firebase token, který identifikuje dané zařízení. Díky tomu Firebase server ví, kam notifikační zprávu odeslat.

Co se týče serveru, stačilo přidat přístupový server klíč a jednu třídu, která je vytvořena přímo pro potřeby frameworku Yii2. Ukládání jednotlivých Firebase tokenů k uživateli již Dactyl CMS umožňoval. Výhodou implementace tohoto systému je, že pokud se uživatel přihlásí zároveň na různé zařízení, všude mu přijdou notifikace, protože se uloží všechny Firebase klíče.

Následně stačilo přidat posílání notifikace při vytvoření nové rezervace. Po úspěšném vytvoření rezervace se odesílá notifikace trenérovi vlastnící lekci. Zpráva musí obsahovat Firebase token/y, které má server uložené v databázi u uživatele. Odesílání notifikací je implementováno tehdy, pokud sportovec nebo trenér změní stav rezervace.

5.1.5 Google Places

Při vytváření nové lekce musí trenér zvolit lokaci, kde se bude lekce odehrávat. Proto aby mohl trenér vybrat z velkého množství sportovišť nebo fitness center, přidal jsem do aplikace knihovnu Google Places, která poskytuje vyhledávání míst dle názvu nebo adresy. Vyhledávání je vytvořeno způsobem, který uživateli doplňuje vyplňovaný dotaz. To umožňuje třída *PlacesClient* a její metoda `findAutocompletePredictions`, která asynchronně vyhledává zadaný řetězec na serveru Googlu a vrací seznam míst obsahující id, název, adresu a zeměpisné souřadnice. Pokud trenér dané místo vybere, zmíněné informace se uloží k vytvářené lekci.

5.1.6 Google Maps

V návrhu detailu lekce jsem na konec stránky vložil mapu, která zobrazuje umístění, kde se lekce bude konat. Proto abych mohl tento návrh realizovat, přidal jsem do aplikace další knihovnu Google Maps. Prvně jsem musel přidat do souboru *AndroidManifest* API klíč. Druhý krok obnášel přidat do layoutu aktivity fragment, který bude vyplněn mapou. Nakonec jsem implementoval v aktivitě rozhraní mapy `onMapReady`, které se provolá, jakmile bude mapa připravena. V implementaci rozhraní stačilo zavolat funkci `addMarker` pro přidání markeru na mapu a `moveCamera`, která zajistí přesun pohledu na adresu lekce dle zadaných atributů zeměpisné šířky a výšky.

5.1.7 Komunikace se serverem

Komunikace se serverem je založená na protokolu HTTP, kdy se využívá tzv. komunikačních endpointů, které jsou popsány v návrhu 4.7. Pro komunikaci s těmito endpointy se musí provést HTTP požadavek obsahující *GET*, *POST* nebo *DELETE*, podle toho zda

[2]Nastavení Firebase na adrese `https://console.firebase.google.com/u/1/`.

data chceme číst, posílat nebo mazat. Pro tyto účely jsem využil knihovny Retrofit, která zjednodušuje implementaci těchto volání. Dále umožňuje používat knihovnu Moshi, která všechny příchozí data transformuje z formátu JSON na objekty a všechny odchozí datové objekty zase na JSON. Což velmi ulehčuje implementaci a všechno co stačí po nastavení knihovny udělat, je vidět ve výpisu 14. Zde jsem pro jednoduchost ponechal pouze tři typy dotazů. Každý z nich musí být označen typem dotazu a ihned potom URL identifikující patřičný koncový bod. URL zde není uvedeno celé, protože její prefix nastavuje již při vytváření instance jedináčka knihovny Retrofit.

První dotaz `lessonUpdate` v ukázce 14 slouží pro aktualizaci informací o lekci. Očekává zde, že v URL jako parametr bude ID lekce. Dále se v těle dotazu posílají všechny aktualizovaná data lekce. Aplikace následně očekává ze serveru potvrzení o uložení společně i se všemi daty týkajícími se lekce. Očekávaná data jsou vždy zapouzdřena do `BaseResponse`, což je obecná třída využívaná proto, aby knihovna Moshi mohla napasovat JSON strukturu na následující parametry:

- `status` — číselný kód udávající jak server zpracoval požadavek (200 značí úspěch, 400 validační chybu, 401 chybu autorizace atd.),
- `data` — data generického typu, který je definován místo písmena `T`,
- `error` — pokud nastane chyba, obsahuje doplňující zprávu o chybě.

V případě `lessonUpdate` jsou `data` typu `LessonResponseData`, která obsahují objekt třídy `Lesson`. Díky tomu, že je všechno obalené v Single, může se funkce provádět asynchronně. Jakmile přijde odpověď, jsem schopen provést patřičné operace již v hlavním vlákně na popředí a uživateli např. zobrazit, že aktualizace lekce proběhla úspěšně. Téměř při každém volání funkce tohoto typu se uživateli aplikace zobrazí progress dialog, který značí probíhající komunikaci se serverem.

```
@POST("lesson/{id}")
fun lessonUpdate(@Path(value = "id", encoded = true) lessonId: String,
                 @Body lesson: Lesson)
    : Single<BaseResponse<LessonResponseData>>

@GET("lesson/{id}")
fun syncLesson(@Path(value = "id", encoded = true) lessonId: String)
    : Single<BaseResponse<LessonResponseData>>

@DELETE("lesson/{id}")
fun lessonDelete(@Path(value = "id", encoded = true) lessonId: String)
    : Single<BaseResponse<Array<String>>>

class BaseResponse<T>(val status: Int, val data: T?, val error: String?)
class LessonResponseData(@Json(name = "Lesson") var lesson: Lesson)
```

Výpis 14: Ukázka implementace komunikace se serverem, která znázorňuje aktualizaci, synchronizaci a mazání lekce. Dále jsou zde uvedeny dvě datové třídy, které se využívají při zpracování zmíněných operací.

Další funkcí ve výpisu 14 je `syncLesson`, která zajišťuje synchronizaci dat určité lekce, která je identifikována parametrem `lessonId`. Zde se data pouze získávají a žádné se neposílají, proto se využívá požadavku *GET*. Způsob vrácení dat ze serveru je jinak úplně stejný jako u aktualizace lekce.

Třetí funkcí na ukázce 14 je smazání lekce zavoláním `lessonDelete`, která vyžaduje ID lekce v parametru `lessonId`. Ze severu se neočekává žádný datový výsledek, pouze zda `status` bude obsahovat úspěšné smazání, které se značí číselným kódem 200.

Nakonec zde zmíním, že všechny funkce, které se využívají pro práci se serverem, jsou implementovány v rozhraním `FittyyRestApi`. Veškerá práce s tímto rozhraní je pouze pomocí repozitářů Repository a v jednom výjimečným případě pomocí třídy `RestImpl`, která má za úkol aktualizovat Firebase token na Fittyy serveru.

5.2 Implementace uživatelského prostředí

Grafické rozhraní jednotlivých obrazovek na Androidu se vytváří pomocí tzv. XML layoutů. Do nich lze přidávat jednotlivé komponenty, které jsou importovány z knihovny Material Designu. Každá obrazovka má určitý rodičovský layout, který udává jakým způsobem budou komponenty umísťovány. Nejčastěji jsem využíval `ConstraintLayout`, který umožňuje připínat jednotlivé strany komponenty k jiným komponentám nebo k hranám obrazovky. Tento druh layoutu velmi napomáhá i k responzivnímu designu. Při definici výšky a šířky komponenty lze zadat, aby se její parametry natáhly až do souřadnic, kde je komponenta přichycena resp. kde má končit.

Předtím, než jsem začal vyvíjet jednotlivé obrazovky, nadefinoval jsem si všechny designové styly do souboru `style.xml` a barvy vycházející ze specifikace Material Designu do `colors.xml`. Designové styly obsahují převážně textové styly, které se využívají u textových polí. Definované barvy reprezentují všechny možnosti, které lze v aplikaci použít. Tento způsob se později ukázal jako velmi výhodný, protože jsem během vývoje měnil font i barvy. Stačilo změnit vlastnosti textu a barev na jednom místě a následně se změna projevila do celé aplikace. Celkem jsem vytvořil jedenáct úrovní textových stylů, kde každá úroveň je vytvořena pro světlé i tmavé pozadí. Dále je ještě každý styl rozdělen na následující tři stavy:

- `Primary` — základní odstín barvy pro použití,
- `Secondary` — méně důležitá barva typicky používána u podnadpisů,
- `Disabled` — barva oznamující zakázaný stav používaný např. při tlačítku, na které nelze kliknout.

Ukázka definice textového stylu a barvy textu je ve výpisu 15, kde `Headline5` značí úroveň textového stylu, `Dark` vyznačuje použití tmavých barev pro světlé pozadí a `Primary` odstín tmavé barvy. Dále jsou ve výpisu 15 definovány tři nejpoužívanější odstíny (stavy) tmavé barvy písma.

Po celou dobu vývoje grafického prostředí aplikace jsem využíval svoje návrhy v Zeplinu. Díky tomu jsem rychleji konvertoval grafické komponenty do programové podoby a to zejména i díky navrženým textovým stylům nebo i stylům pro tlačítka. Styl každé komponenty lze jednoduše zobrazit na pravé straně programu Zeplin.

Dále jsem využil souboru `dimens`, který obsahuje číselné hodnoty využívané v XML layoutech. Tento systém definice jsem použil např. pro zakulacení rohů tlačítek a karet. Díky tomu můžu měnit globálně zakulacení prvků změnou jedné hodnoty proměnné.

```
<color name="textDarkPrimary">#DE000000</color>
<color name="textDarkSecondary">#99000000</color>
<color name="textDarkDisabled">#61000000</color>

<style name="Headline5DarkPrimary">
  <item name="android:textSize">24sp</item>
  <item name="android:textColor">@color/textDarkPrimary</item>
  <item name="android:fontFamily">@font/lato_regular</item>
</style>
```

Výpis 15: Ukázka definice textového stylu a barvy textu pro nadpis šesté úrovně.

Co se týče lokalizace aplikace, využil jsem možnosti vytvoření více souborů `strings` pro různý jazyk. Každý překlad je reprezentován jedinečným klíčem, který se může použít jak v grafickém layoutu, tak v kódu aktivity.

5.3 Změny vůči návrhu aplikace

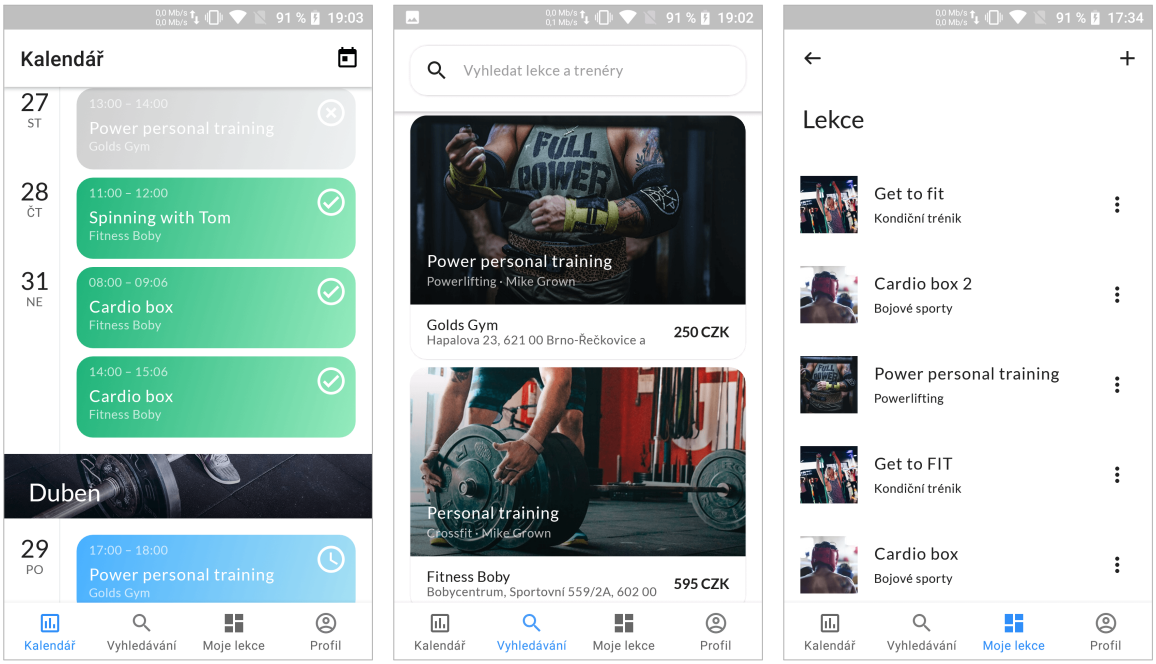
V rámci této kapitoly bych chtěl představit změny, které jsem udělal vůči návrhům popsaných v kapitole 4. Zejména se jedná o grafický návrh a procesy aplikace.

Cílem prvotní implementace aplikace bylo zprovoznit základní funkcionalitu, která je potřebná pro trenéra a sportovce. Proto jsem jako první v aplikaci implementoval přihlášení a registraci uživatele, vytvoření lekce trenéra, rezervaci lekce sportovcem, možnost vyhledávat lekce, zobrazit a upravit si vlastní profil. Následně jsem spustil uživatelské testování s trenéry, ze kterého vyšlo najevo pár poznatků, které jsou popsány blíže v kapitole 6. V rámci testování vyplynulo, že jednou z věcí, která trenérům nechyběla, byla prezentace vlastního profilu dostupného přes vyhledávání, detail lekce nebo rezervace. Důvodem je i ta skutečnost, že vyhledávání trenérů a jejich prezentace má dle mého názoru smysl až v době, kdy aplikace bude mít více uživatelů. Proto jsem se rozhodl nezahrnout implementaci vyhledávání trenérů a možnost zobrazení jejich profilů sportovcem a radši jsem se zaměřil na nedostatky, které trenéři při testování objevili.

V první verzi aplikace jsem také obměnil barvy a zakulacení rohů komponent v celé aplikace. Tato změna je vidět na obrázku 5.1 uprostřed, kde karta vyhledávání a prvek seznamu má více zakulacené rohy než bylo v návrhu. Dále jsem zvolil pro každý stav rezervace barvu, kterou je možné vidět pouze v kalendáři rezervací (obrázek 5.1 vlevo). Díky tomu se může uživatel v seznamu orientovat rychleji.

Další obrázek 5.1 vpravo je vyfocen z pohledu trenéra. Zde jsem se zaměřil na zmenšení jednotlivých lekcí, tak aby jich trenér viděl co nejvíce a zároveň mohl pomocí tlačítka s třemi tečkami vyvolat akční menu, které umožňuje lekci smazat. Lze si také povšimnout, že trenér ve spodním menu má možnost přejít na vyhledávání lekcí ostatních trenérů. Tento nedostatek vyplynul již při představení návrhu trenérům (kapitola 4.4), kteří si tuto funkcionalitu vyžádali.

Vyhledávání jsem o proti návrhu (obrázek 4.5) také změnil. Protože již nebylo potřeba kombinovat výsledky vyhledávání trenérů a lekcí, tak jsem vyhledávání přesunul pouze na jedinou obrazovku. Po příchodu na obrazovku vyhledávání (obrázek 5.1 uprostřed) se uživateli zobrazí seznam lekcí, který jsou nejblíže k jeho aktuální poloze. To mi umož-



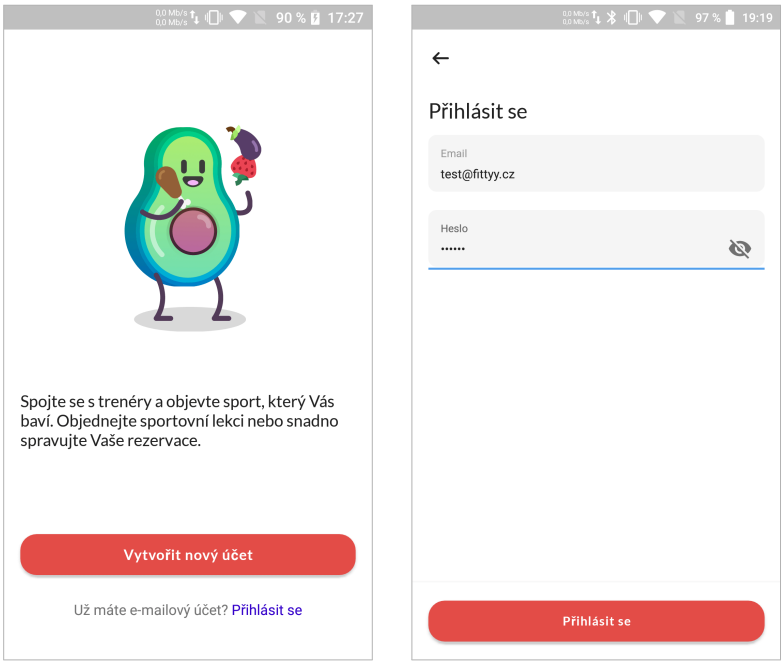
Obrázek 5.1: Obrázky z výsledné aplikace Fittyy, **vlevo:** kalendář rezervací byl zpřehledněn a byly změněny barvy rezervací na základě jejich stavu, **uprostřed:** zobrazení nejbližších lekcí, které umožňuje vyhledání na základě zadání výrazu do vyhledávacího pole ve vrchní části obrazovky, **vpravo:** seznam lekcí trenéra jsem zpřehlednil a zmenšil výšky jednotlivých prvků v seznamu.

ňuje endpoint `lesson/list`, kam je potřeba zaslat parametr `sort` s hodnotou `closest` a do hlavičky přidat aktuální zeměpisnou výšku a šířku uživatele. Pokud tedy uživatel udělil práva aplikaci tuto informaci zjišťovat. Pokud chce uživatel vyhledávat, stačí pouze kliknout do horní vyhledávací karty, která mu otevře klávesnici pro psaní. Následně může zapsat hledaný výraz, který musí potvrdit. Díky tomu se vyhledávání lekcí zkrátilo ze čtyř kroků na dva kroky.

V grafickém návrhu jsem vynechal proces přihlášení a registrace. Tyto dva procesy se v aplikaci vykonávají z úvodní obrazovky, která je znázorněna na obrázku 5.2 vlevo. Ve vrchní části je implementována animace avokáda, které si žongluje s ovocem. Sekvenci obrázků jsme stáhl z volného zdroje ve JSON formátu. Následně jsem využil knihovny Lottie od firmy Airbnb, která poskytuje komponentu `LottieAnimationView`, která slouží pro zprovoznění animace. Uživatel z úvodní obrazovky může přejít rovnou do přihlášení (obrázek 5.2 vpravo), kde jsou vyžadovány přihlašovací údaje.

Pokud se uživatel chce registrovat, klikne na tlačítko pro vytvoření nového účtu. Registrace uživatele (obrázky na 5.3) je provedena ve třech krocích:

1. **Výběr uživatelské role** (trenér nebo sportovec).
2. **Vyplnění přihlašovacích údajů.** Pokud uživatel zadá nevalidní email nebo krátké heslo, tlačítko pro pokračování není klikatelné a uživatel se nemůže dostat do dalšího kroku.
3. **Vyplnění jména a příjmení.** Pokud je vše zvalidováno aplikací a uživatel souhlasí se zpracováním osobních údajů, může odeslat požadavek registrace na server. Pokud



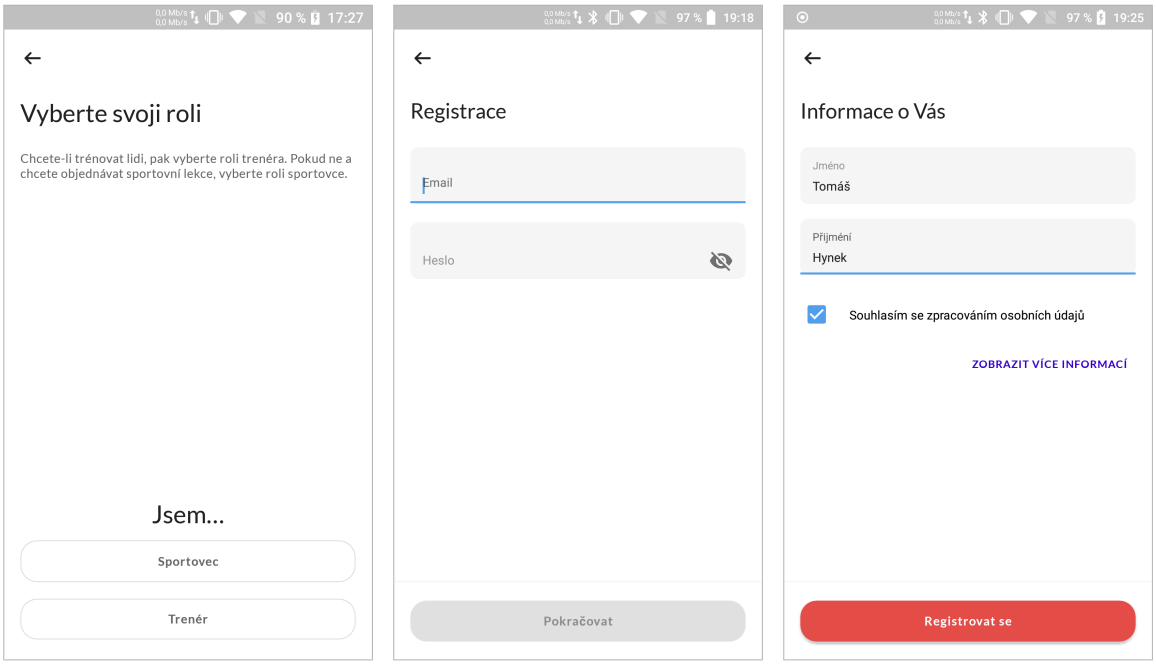
Obrázek 5.2: Obrázky z výsledné aplikace Fittyy, **vlevo:** uvítací rozcestník s animací avokáda, odkud se může uživatel přihlásit nebo registrovat, **vpravo:** obrazovka pro přihlášení uživatele.

je vyhotoven požadavek úspěšně, zobrazí se mu hlavní aktivita s kalendářem rezervací. Pokud ne, zobrazí se mu chyba, která nastala.

Pokud je uživatel při zapnutí aplikace již přihlášen, úvodní aktivita přejde rovnou do hlavní obrazovky, kde se uživatelovi zobrazí jeho kalendář s rezervacemi.

Uživatel má samozřejmě i možnost odhlášení a to z obrazovky nastavení. Odhlášení je doprovázeno potvrzovacím dialogem, zda uživatel chce opravdu tento úkon udělat (obrázek 5.4 vlevo). Pokud uživatel odhlášení potvrdí, aplikace odesílá požadavek na server. Server smaže z databáze přístupový token a následně vrací zprávu obsahující úspěšné odhlášení. Aplikace na to reaguje smazáním všech dat z databáze a přístupového tokenu. Následně je uživateli zobrazena uvítací obrazovka. Co se týče dialogů, ty jsem v aplikaci vyřešil pomocí knihovny Material-Dialogs od tvůrce Aidan Follestad. Umožňuje jednoduše nastavit a zobrazit dialog tak, aby odpovídal stylu aplikace. Způsob použití je možné vidět ve výpisu 16, kde kód nastavuje titulek a zprávu dialogu, text pozitivního i negativního tlačítka a nakonec akce, které se mají při stisku provést. Po vytvoření je ihned zobrazen dialog, který je možné vidět na obrázku 5.4 vlevo.

Další obrázek 5.4 uprostřed znázorňuje způsob volby kategorie lekce při jejím vytváření. Jak je vidět, kategorie jsou v českém jazyce. Jenže aplikace umožňuje i angličtinu. V případě kategorií to ale nebylo tak jednoduché, protože ty jsou spravovány na serveru, proto aby se mohli upravovat i bez toho, aniž by musela být vždy vydána nová verze aplikace. Proto při každém otevření aplikace, pokud je uživatel přihlášen, se kategorie synchronizují a to způsobem, kdy na server v hlavičce je vždy odesílán jazyk, který aplikace aktuálně využívá. Využívaný jazyk jsem přidal do překladů `strings`, kde pro českou lokalizaci proměnná `system_language` obsahuje hodnotu `cs`. Pro ostatní lokalizace telefonu bude vždy anglický překlad a tím pádem hodnota `en`.



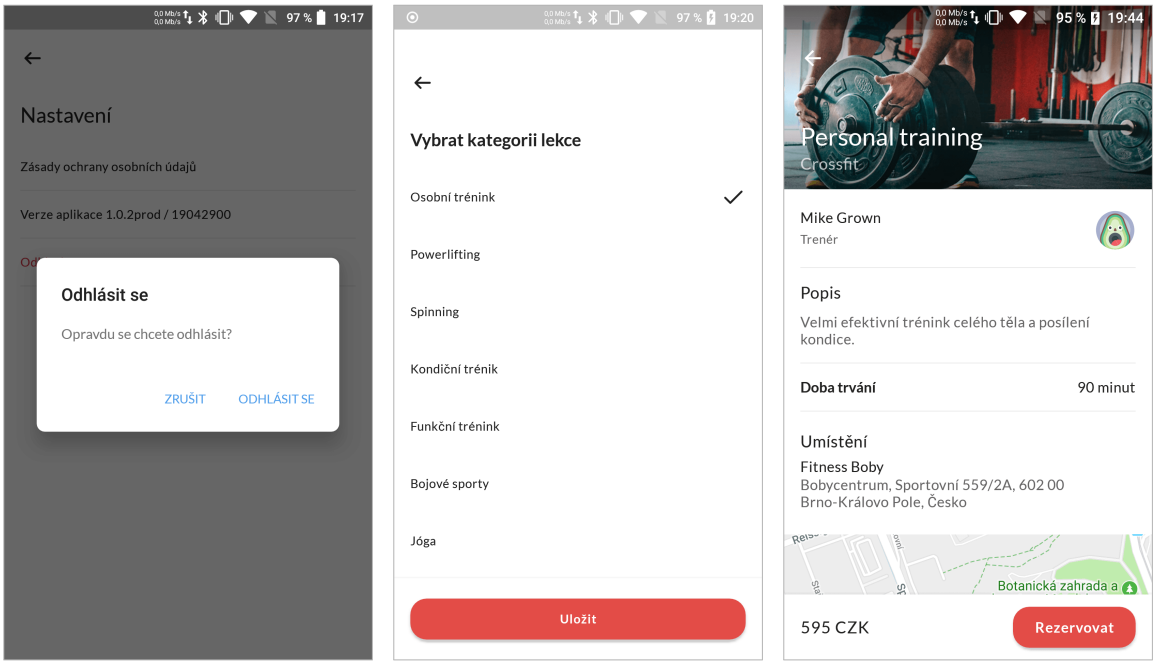
Obrázek 5.3: Obrázky z výsledné aplikace Fittyy, **vlevo:** prvně musí uživatel vybrat roli, ve které bude vystupovat v aplikaci, **uprostřed:** při registraci si uživatel volí jeho přihlašovací údaje, **vpravo:** nakonec musí uživatel zadat své jméno a příjmení.

```
private fun showLogoutDialog() {
  MaterialDialog(this).show {
    title(R.string.settings_logout)
    message(R.string.settings_logout_dialog_message)
    positiveButton(R.string.global_logout)
    negativeButton(R.string.global_cancel)
    positiveButton { viewModel.logout() }
    negativeButton { dismiss() }
    cancelOnTouchOutside(false)
    cancelable(false)
  }
}
```

Výpis 16: Ukázka použití knihovny Material-Dialogs pro vytvoření a zobrazení dialogu.

Menší úpravu jsem udělal i na detailu lekce (obrázek 5.4 vpravo). Odstranil jsem cenu lekce ze seznamu informací a ponechal ji pouze na spodním menu, tak aby byla po celou posouvání se na obrazovce vidět. Není potřeba aby tento údaj byl v detailu lekce dvakrát. Na obrázku 5.4 vpravo je také možné znovu vidět, jak vypadá výsledný styl tlačítek, který původně nebyl tak zakulacený a měl jinou barvu (návrh se nachází v kapitole 4).

U rezervace lekce jsem pozměnil způsob výběru data. V návrhu byl kalendář (obrázek 4.12), kterým se dalo listovat. Ve vytvořené aplikaci (obrázek 5.5 vlevo) jsem vytvořil pouze seznam dní pro daný měsíc s navigačními šipkami, pomocí kterých lze přecházet



Obrázek 5.4: Obrázky z výsledné aplikace Fittyy, **vlevo:** dialog pro odhlášení uživatele, **uprostřed:** výběr kategorie lekce při jejím vytváření, **vpravo:** detail lekce z pohledu role sportovce, který odtud může začít proces rezervace.

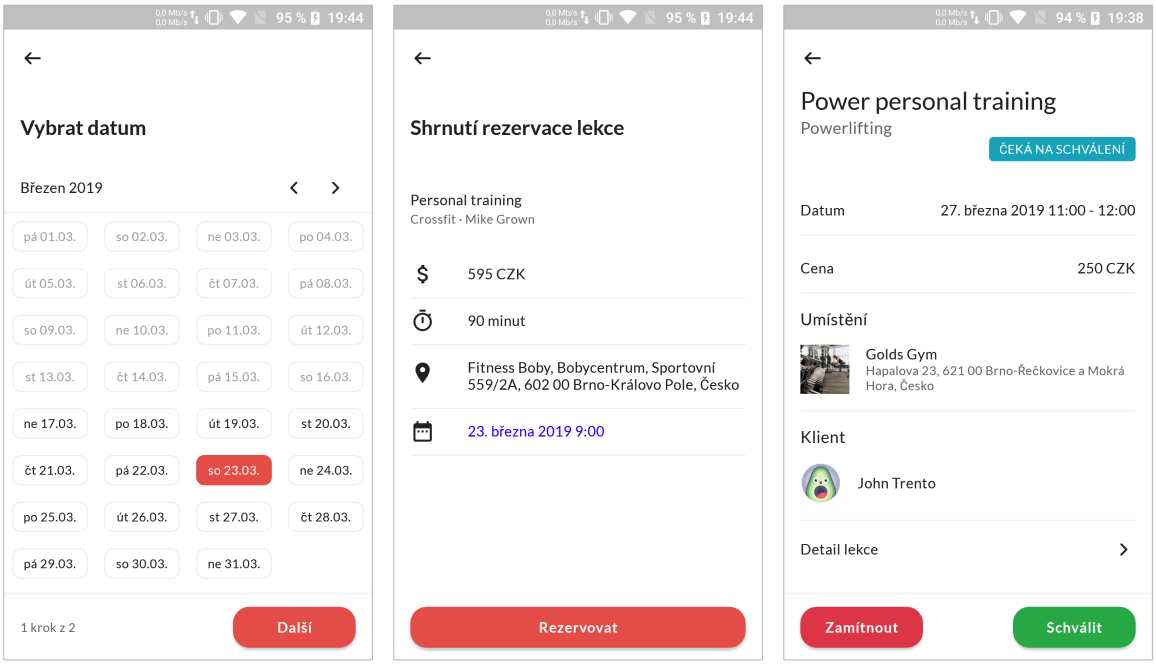
mezi jinými měsíci. Pokud sportovec zobrazuje aktuální měsíc, dny v minulosti jsou již nedostupné (zašedlé). Pokud má trenér některé dny již rezervované nebo lekce v tyto dny není k dispozici, jsou tyto data zašedlá a nemožná k výběru.

Shrnutí rezervace (obrázek 5.5 uprostřed) obsahuje navíc délku trvání a cenu lekce. Ve spodní části jsem pouze nechal tlačítko závazné rezervace lekce. Pokud server rezervaci potvrdí, sportovci je zobrazen detail rezervace. Pokud je lekce nastavena tak, že se automaticky neschvaluje serverem, musí trenér přejít do detailu rezervace, kde ji musí schválit, jinak nebude platná. K tomu se mu zobrazí detail rezervace doplněný o spodní menu, které je možné vidět na obrázku 5.5 vpravo. O proti návrhu (obrázek 4.14) jsem ještě odstranil z vrchní části obrazovky fotografii lekce. Pokud bych ji tam nechal, detail lekce a detail rezervace vypadaly velmi podobně a hodně splývaly. V rámci změn jsem se dále pro jednoduchost zaměřil pouze na osobní lekce, proto jsem z detailu rezervace odstranil informaci o kapacitě lekce. Trenéři si na tuhle změnu nestěžovali, proto jsem obrazovku takto ponechal.

5.4 Implementace serveru

Jak jsem již psal v kapitole návrhu 4.7, server je založený na systému Dactyl CMS. Aktuální verze systému, která funguje jako serverová aplikace pro Fittyy, má následující specifikace:

- kód je psaný v jazyce PHP 7.2,
- nadstavba nad frameworkem Yii 2.0,
- databáze pomocí MariaDB Server,



Obrázek 5.5: Obrázky z výsledné aplikace Fittyy, **vlevo:** sportovec při rezervaci lekce musí vybrat datum konání, **uprostřed:** shrnutí informací před potvrzením závazné rezervace lekce, **vpravo:** detail rezervace lekce z pohledu trenéra, který musí rezervaci buď schválit nebo zamítnout.

- balíčky v composeru a npm.

Proto aby mohli být implementovány jednotlivé endpointy, které jsou popsány v kapitole 4.7, má každá entita, ke které se endpoint vztahuje, svůj vlastní kontrolér. Každý kontrolér je podtřídou `ApiController` vycházející z knihovny Dactyl Core. Api kontrolér implementuje metodu pro nalezení modelu dané entity, která se využije pro změny, mazání nebo vytvoření dat a případně uložení změn do databáze serveru. Každý model entity je podtřídou `ActiveRecord`, která také vychází z knihovny Dactyl Core, kde jsou implementovány základní operace s entitou.

Kvůli zabezpečení se musí aplikace při používaní serveru vždy autorizovat pomocí přístupového kódu, který je potřeba odeslat v hlavičce HTTP požadavku. Tento přístupový kód je možné získat pouze tehdy, pokud se uživatel úspěšné přihlásí nebo zaregistruje na server pomocí mobilní aplikace Fittyy. Následně je uživatel autentizován a aplikace obdrží přístupový kód, který používá pro komunikaci se serverem po celou dobu, co je uživatel přihlášen. Jakmile se uživatel odhlásí, server tento kód odstraní z databáze a dále je již nevalidní. Pokud by aplikace chtěla tento kód použít znovu, server vrací chybu autorizace s číselným kódem 401.

Vyhledávání lekce

Jednou z hlavních funkcí aplikace a serveru je vyhledávání lekcí. To funguje na základě filtru, který se na server z aplikace posílá. Endpoint, který se pro tyto účely využívá je `/lesson/list` a může obsahovat následující volitelné parametry filtru:

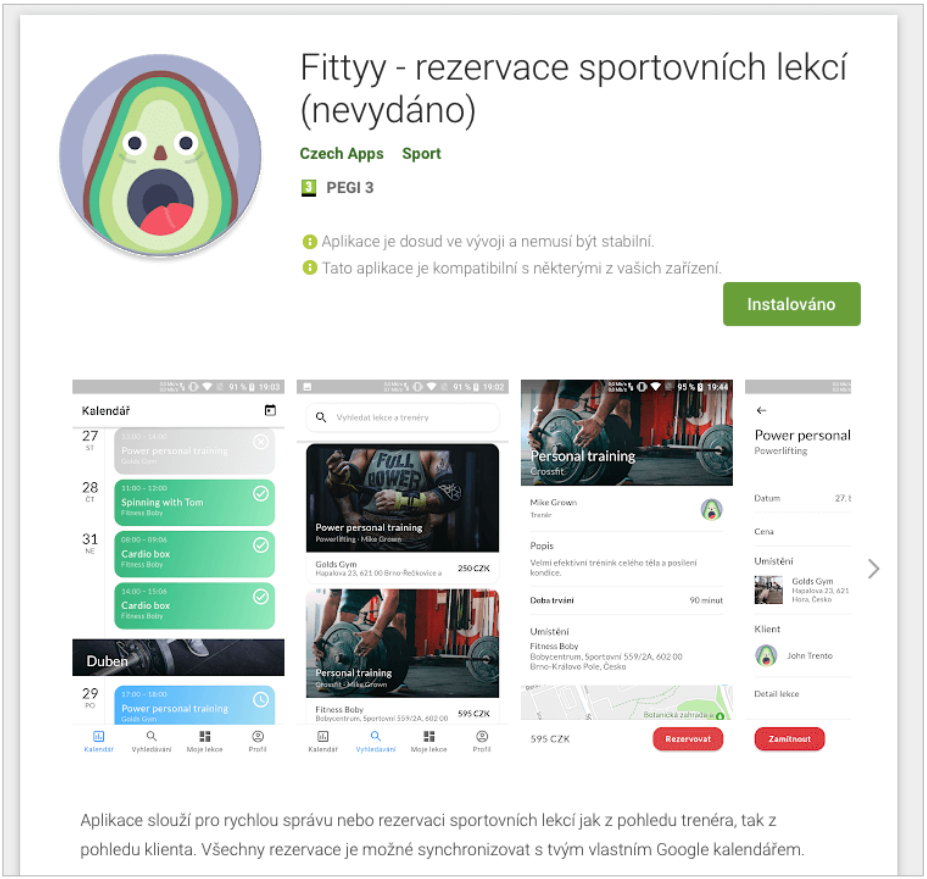
- `name` — textový řetězec, který slouží pro vyhledání lekce na základě jejího jména,

- `id_user` — možnost vyfiltrování lekcí jen daného trenéra,
- `sort` — seřazení lekcí dle vzdálenosti od telefonu uživatele, pokud obsahuje hodnotu `closest`, lekce budou seřazeny od nejbližších po nejvzdálenější místa konání.

Pro lepší představu je ve výpisu 17 zkrácená ukázka kódu vyhledávání lekce. Funkce `search` obsahuje jeden parametr obsahující hodnoty filtru, které aplikace na server zaslala. Prvním krokem v této funkci bylo potřeba získat referenci na objekt `query` z knihovny Dactyl Core. Následně vytvořit instanci třídy `ActiveDataProvider` s nastavením parametru `query`, který se využije pro jednodušší nastavení SQL dotazu. V dalším kroku se vypočítá vzdálenost lekcí od polohy uživatele. Následně se zjišťuje, zda filtr obsahuje parametr seřazení s hodnotou `closest`. Pokud ano, seřadí se lekce od nejbližší vzdálenosti po nejvzdálenější. Poté se k lekcím připojují další relační data, které jsou identifikována cizím klíčem u lekce. Nakonec se ještě provede filtrování výsledků dle identifikačního čísla trenéra a vyhledávacího řetězce, pokud tedy server takovéto parametry z aplikace přijal.

```php
<?php
public function search($params) {
  // objekt ActiveQuery reprezentující SQL query
  $query = self::find();
  // objekt, přes který se získávají data
  $dataProvider = new ActiveDataProvider([
    'query' => $query
  ]);

  // výpočet vzdálenosti lekce od uživatele
  $this->injectDistanceQuery($dataProvider, 'place');

  // seřazení dle vzdálesnosti, pokud to vyžaduje filter
  if ($this->sort == 'closest')
    $dataProvider->sort->defaultOrder = ['distance' => SORT_ASC];

  // spojení entity Lesson s ostatními entitami a subentitami
  $query->with('user');
  $query->joinWith('place');
  $query->with('openingHours');

  // vykonání filtru dle názvu lekce a id trenéra
  $query->andFilterWhere(['lesson.id_user' => $this->id_user]);
  $query->andFilterWhere(['LIKE', 'lesson.name', $this->name]);

  return $dataProvider;
}
```

Výpis 17: Ukázka implementace vyhledávání lekce na serveru.



Obrázek 5.6: Profil aplikace Fittyy na Google Play.

5.5 Vydání aplikace

Aplikaci jsem se rozhodl vydat ještě ve chvíli, když nebyla úplně kompletní, a to z toho důvodu, aby trenéři mohli aplikaci testovat a případně se v dalším vývoji zaměřit na funkce, které jim budou chybět. Pro vydání jsem se rozhodl použít veřejný Beta kanál na Google Play, který oznamuje, že je aplikace ještě ve vývoji. Aktuálně mám soubor `build.gradle` nastavený tak, že aplikaci je možné zbuildit a vydat pod dvěma variantami:

- **devRelease** — aplikace funguje se serverem, který je pro potřeby vývoje a testování,
- **prodRelease** — aplikace funguje s produkčním serverem, který se využívá pro verzi na Google Play.

K vydání aplikace na Google Play byl zapotřebí vývojářský účet, který jsem již vlastnil. Stačilo tedy vymyslet správné popisy v českém i anglickém jazyce a vytvořit fotografie z aplikace, které se nahrají do Google Play Console. Výsledek profilu aplikace Fittyy na Google Play[3] je možné vidět na obrázku 5.6.

[3]Odkaz pro stažení aplikace https://play.google.com/store/apps/details?id=cz.czechapps.fittyy

Kapitola 6

Testování aplikace

Jakmile jsem dokončil důležitý milník implementace této práce, začal jsem s uživatelským testováním aplikace. Po vydání aplikace na Google Play jsem odeslal odkaz trenérům, se kterými jsem již komunikoval při začátku této práce. Výsledky a nedostatky testování jsou rozepsány v následujících podkapitolách.

6.1 První iterace testování

V rámci této iterace se dva trenéři poprvé seznamovali s reálnou aplikací, kterou dříve viděli pouze jako interaktivní grafický návrh. Jejich poznatky a nedostatky jsou popsány níže.

Trenér 1

Tento trenér má týdně 20 až 25 osobních lekcí a k tomu pár skupinových. Používá k tomu rezervační systém, který vzhledově podobný jako Google kalendář. Vždy si musí zablokovat víkend a některé dny, kdy má např. skupinové lekce. Na skupinové lekce nemá možnost rezervace. Pomocí tohoto systému nabízí klientům časové bloky, ve kterých je volný. Časovým blokem se myslí, že je trenér k dispozici např. od 6:00 do 11:00 a od 13:00 do 18:00 hodin. Pokud ale potřebuje např. k zubaři, tak se do daného volného bloku musí zapsat jako klient. Tím zabere dané volné hodiny a nikdo se tam nemůže již zapsat.

Co se týče testování, tak trenér zmiňoval zejména nepochopení procesu vytváření lekce a časových bloků. Měl problém s tím, že nebyl schopen pochopit, jak časové bloky pro lekci nastavit. Až po mém vysvětlení jsem zjistil, že jednoduše přehlédl nastavení dostupnosti lekce při vytváření.

Trenér 2

Trenér 2 má údajně v některých týdnech až 60 osobních lekcí v různých posilovnách. Pro svoje účely používá pouze Google kalendář, kde si musí všechny rezervace zapisovat sám.

Hned pro začátek mě trenér upozornil na to, že při vytváření lekce nefunguje vyhledávání přes Google Places, které slouží pro výběr místa konání lekce. Tuto chybu jsem ihned opravil, a to způsobem, kdy jsem musel přidat platební kartu do Firebasu, aby mě v případě překročení limitu, co je pro vyhledávání zdarma, mohl Google účtovat poplatky.

Při dalším testováním mi trenér sdělil následující nedostatky nebo problémy:

- nebylo mu jasné, jak funguje nastavení časových bloků u lekce,

- uvítal by, aby klient měl možnost zakoupit si balíček osobních lekcí, který by mohl využít v aplikaci,
- rezervaci každé lekce chce mít vždy v celou hodinu, tak aby mu to ulehčilo plánování lekcí a neměl by tam žádné časové mezery,
- chtěl by, aby aplikace Fittyy měla možnost synchronizace s jeho Google kalendářem,
- chybí mu možnost vytváření vlastních rezervací.

Závěr prvního testování

Na základě poznatků trenérů jsem se rozhodl před další iterací testování dodělat následující funkce nebo úpravy:

- zpřehlednit nastavení časových bloků lekce,
- přidat vytvoření vlastní rezervace,
- přidat synchronizaci rezervací s Google kalendářem.

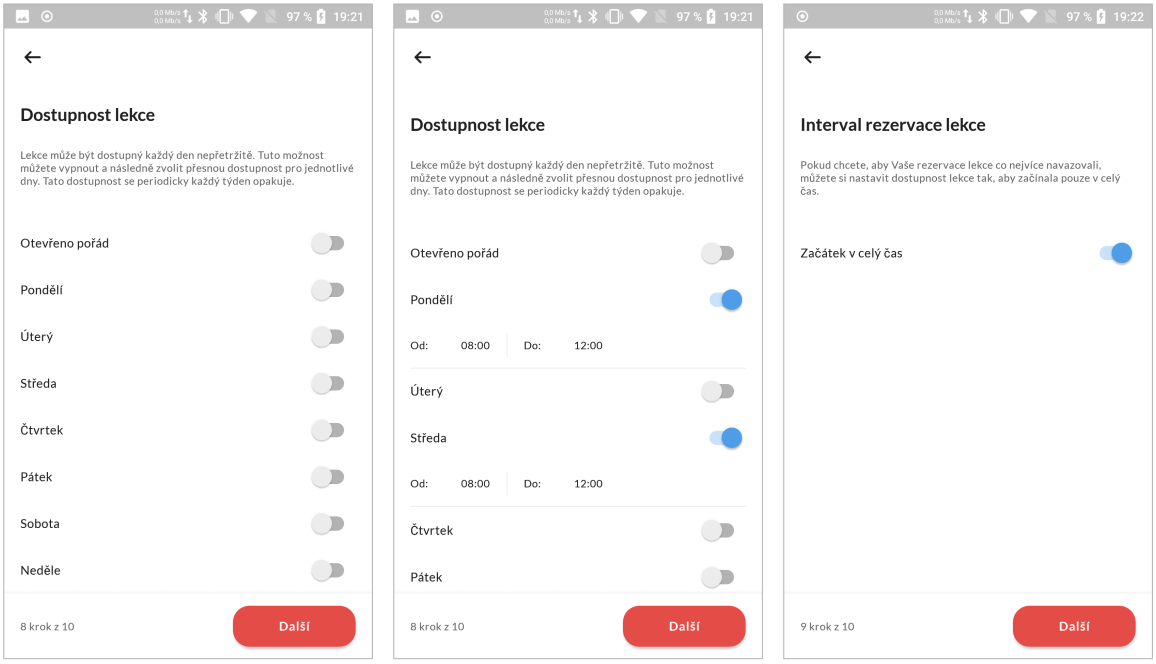
6.2 Výsledek první iterace testování

Dle prvního testování měli trenéři problém s pochopením nastavení dostupnosti lekce. Důvod byl ten, že jsem nastavení časových bloků spojil do jednoho kroku společně s výběrem umístění konání lekce. Takže jsem tento proces resp. krok navrhl špatně. Dle knížky o designu [19] by měl uživatel vždy vědět co je třeba udělat. Z tohoto důvodu jsem nastavení dostupnosti lekce rozdělil do dalšího kroku. Jak je možné vidět na obrázku 6.1 vlevo a uprostřed, trenér musí v kroku osm zvolit buď dostupnost lekce na celý týden bez omezení, nebo vybrat určité dny doplněné o přesný čas. Pokud trenér nezvolí žádný z přepínačů, aplikace jej nepustí do dalšího kroku. Textový popis umístěný pod titulkem by měl trenérovi napomoct se správným rozhodnutím a výběrem dostupnosti.

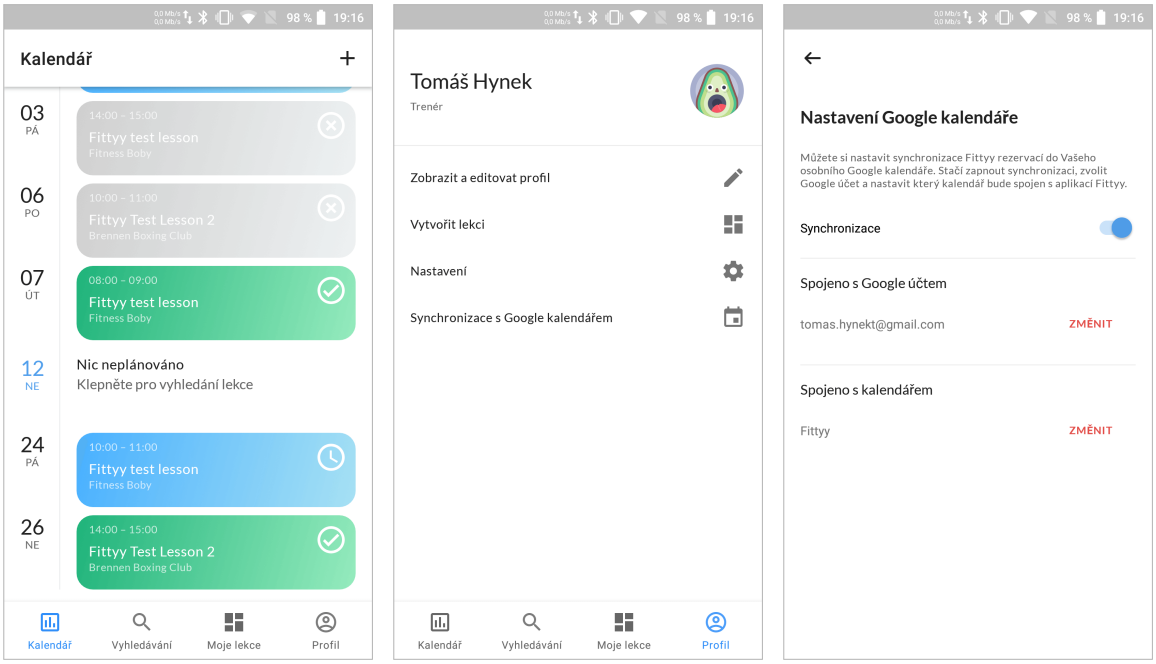
Dále měl jeden trenér požadavek na to, aby mohli být jeho lekce pouze v celé časy. Proto jsem přidal ještě jeden krok do procesu vytváření lekce, který umožňuje zapnout nebo vypnout tuto funkci. Pokud je tato funkce u lekce zapnutá, sportovec uvidí v nabídce při rezervaci pouze časy, které v řádech minut obsahují pouze nuly. To by mělo napomoct trenérovi lépe organizovat jednotlivé lekce do navazujících bloků.

V rámci úprav této fáze jsem se zaměřil na **kalendář rezervací**. Přišlo mi, že jednotlivé prvky rezervací v seznamu jsou moc veliké, a tím pádem uživatel zobrazuje méně rezervací, než by chtěl. Trenéři totiž mají několik lekcí za den, proto jsme výšku prvku seznamu snížil (obrázek 6.2 vlevo).

Další z požadavků trenéra byla **synchronizace s osobním Google kalendářem**. Jak jsem tuto funkcionalitu implementoval jsem již popsal v podkapitole 5.1.3. Do aktivity nastavení synchronizace kalendáře se lze dostat z obrazovky profilu (obrázek 6.2 uprostřed) pomocí posledního tlačítka v seznamu akcí. Následně se zobrazí uživatelovi obrazovka, kterou je možné vidět na obrázku 6.2 vpravo. Pokud chce uživatel zapnout zmíněnou funkcionalitu, musí uživatel přepnout spínač do sepnutého stavu, propojit Google účet s aplikací Fittyy a nakonec vybrat kalendář, se kterým chce rezervace synchronizovat. Následně se schválené rezervace, které ještě v Google kalendáři nejsou, synchronizují vždy při přechodu do aktivity obsahující seznam rezervací uživatele.



Obrázek 6.1: Nově přidané kroky procesu vytváření lekce, **vlevo:** nově přidaný krok, kde trenér musí zvolit dostupnost lekce, která předtím byla sjednocená s výběrem místa konání, **uprostřed:** nově přidaný krok, kde trenér má dostupnost lekce nastavenou na pondělí a středu, **vpravo:** nastavení intervalu, který způsobí, že rezervace lekce může začínat pouze v celých časech.



Obrázek 6.2: Úprava kalendáře rezervací a nová obrazovka pro nastavení synchronizace s Google kalendářem, **vlevo:** výška prvku v seznamu byla snížena tak, aby se zobrazilo co nejvíce rezervací, **uprostřed:** mezi akce v profilu byla přidána možnost zobrazení nastavení synchronizace s Google kalendářem, **vpravo:** nastavení synchronizace s Google kalendářem.

Posledním z nedostatků byla absence funkce **vytváření vlastních rezervací** nebo **časová blokace dostupnosti trenéra**. Proto jsem do aplikace přidal pouze pro trenéry možnost vytvářet si vlastní události. To lze udělat z kalendáře rezervací pomocí tlačítka s ikonkou plusu v pravém horním rohu (obrazovka 6.3 vpravo). Po kliku na tlačítko se ukáže aktivita zobrazená na obrázku 6.3 vlevo, kde trenérovi stačí vyplnit název události a časový úsek, po který nebude dostupný. Po potvrzení vytvoření se zobrazí detail události (obrázek 6.3 uprostřed), který zobrazuje přehled informací. Odtud je možné pouze smazat událost, což zapříčiní, že trenér bude v této době znovu dostupný. Vytvořené události se se zobrazují společně v kalendáři rezervací. Příklad vytvořené události, kdy je trenér nedostupný a jde k zubaři, je vidět na obrázku 6.3 vpravo. Samozřejmě se může vytvoření události provést i v případě, kdy se klient k trenérovi objedná osobně. K tomu, aby trenér mohl využívat tuto funkcionalitu, bylo zapotřebí na server přidat model pro události a endpoint pro vytváření, získávání a mazání událostí. Celkově na serveru přibyly tyto endpointy:

- `POST /event` — vytvoření nové události,
- `GET /event/id` — získání informací o události,
- `POST /event/id` — aktualizace události,
- `DELETE /event/id` — smazání události,
- `POST /event/list` — získání seznamu událostí, kam je potřeba zasílat ve filtru ID trenéra, kterému události patří.

Závěrem této iterace testování a nového vývoje jsem na Google Play vydal novou verzi aplikace. Následně jsem znovu oslovil trenéry, aby aplikaci vyzkoušeli.

6.3 Druhá iterace testování

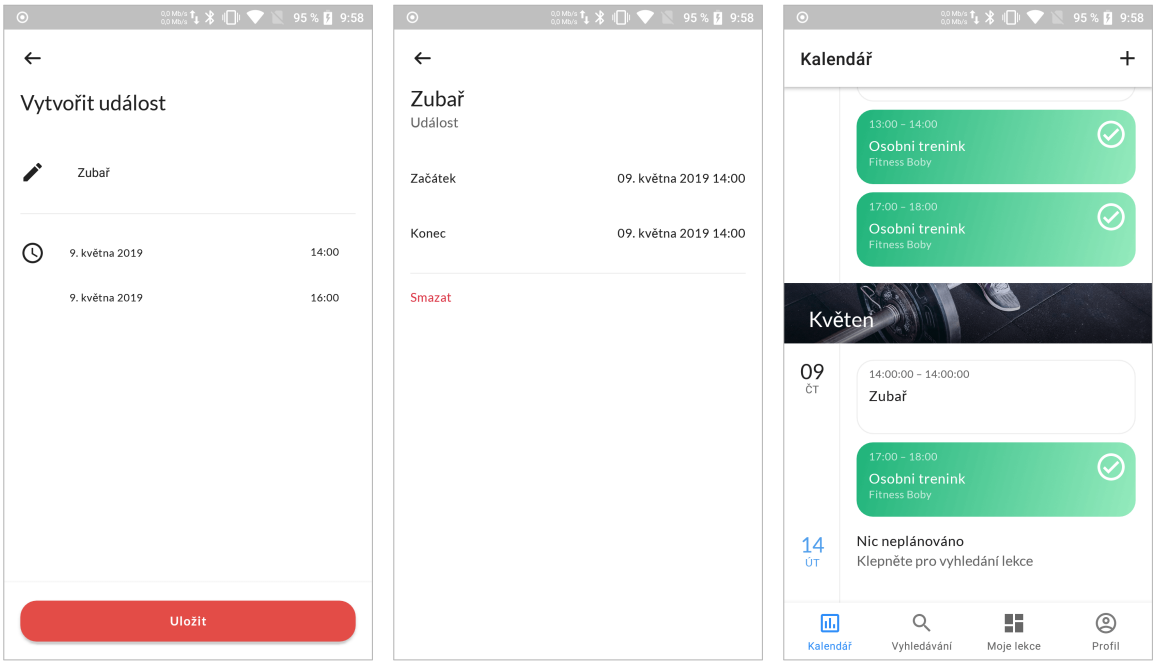
Po změnách implementovaných z předchozí iterace, trenéři vyzkoušeli novou verzi aplikace. Jejich jednotlivé náměty na vylepšení jsou uvedené níže.

Trenér 1

Trenér bohužel neměl během této iterace testování příliš času. Přesto se ale zběžně na aplikaci podíval a zmínil, že se mu líbí možnost rychlé blokace časové dostupnosti trenéra. Další nedostatky neobjevil a říkal, že aplikace je v tomto stavu použitelná.

Trenér 2

Lépe s časem na tom byl trenér dva, který aplikaci více prověřil. Přidané funkce se mu velice líbily a neměl k nim žádné výhrady. Ovšem měl zajímavou poznámku týkající se notifikací na nové rezervace. V aplikaci totiž nebyly prozatím implementovány tzv. push notifikace, které by upozornili trenéra na to, že má novou rezervaci lekce nebo, že jeho klient změnil stav rezervace.



Obrázek 6.3: Trenér má nově možnost si vytvořit vlastní událost, která vyblokuje jeho časovou dostupnost, **vlevo:** blokace dostupnosti trenéra vytvořením vlastní události, **uprostřed:** detail vlastní události, **vpravo:** vlastní události se zobrazují společně s rezervacemi lekcí.

Trenér 3

V rámci této iterace se mi povedlo získat dalšího trenéra, který mobilní aplikaci otestoval. Popisoval aplikaci jako velmi pěknou a jednoduchou co se týče ovládání a použitelnosti.

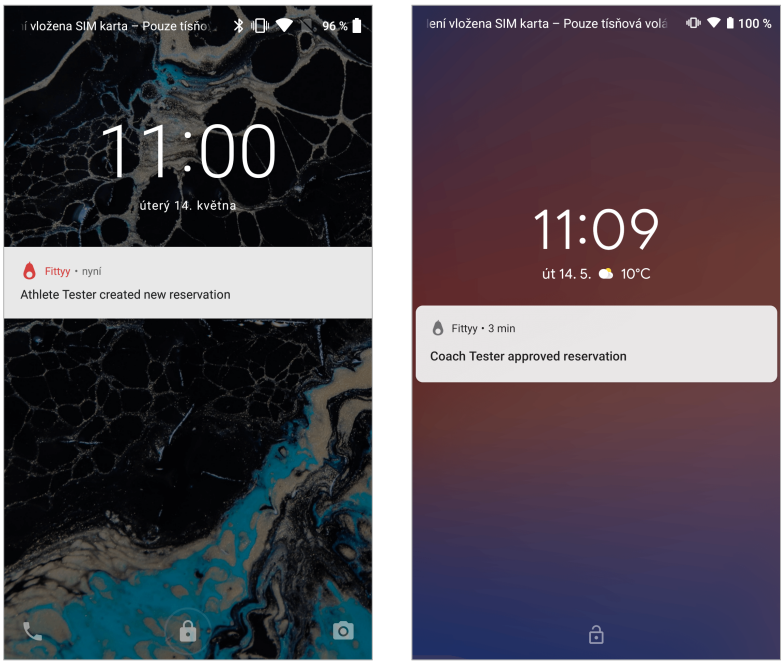
Jelikož ale trenér má pravidelné lekce se stejnými klienty, a to pouze v odpoledních hodinách dva dny v týdnu, připadá mu dostačující používání Google kalendáře. Pravděpodobným důvod je ten, že si trenér rezervace lekcí vytvoří periodicky na delší dobu a následně nemusí již nic měnit. Měl ale zajímavý poznatek. Údajně by se mu líbilo, kdyby aplikace Fittyy uměla prezentovat trenéra. Každý trenér by touto cestou mohl prezentovat svoje portfólio služeb, kde by jej klienti mohli hodnotit. Na základě toho by si pak každý mohl najít svého vhodného trenéra, který působí v jeho blízkosti či dané posilovně.

Závěr druhého testování

V rámci tohoto testování jsem zjistil, že by bylo určitě potřeba, aby každý uživatel věděl, co se děje z jeho rezervacemi, bez toho aniž by musel otevřít aplikaci. Co se týče návrhu trenéra tři, který mluvil o profilech trenérů, tak funkcionalitu, kterou zmiňoval, jsem již zahrnul dříve do grafického návrhu. Jak jsem ale psal již dříve, v aktuálním stavu, kdy v aplikaci jsou pouze tři trenéři, by nedávala smysl.

6.4 Výsledek druhé iterace testování

Ze závěru druhého testování jsem se rozhodl přidat Firebase Messaging, který umožňuje zasílání notifikačních zpráv do aplikací. Postup přidání a implementaci jsem již popsal



Obrázek 6.4: Ukázka notifikačních zpráv, které aplikace Fittyy umožňuje, **vlevo:** notifikace týkající se nové rezervace, kterou přijal trenér od sportovce se jménem Athlete Tester, **vpravo:** sportovec přijal novou notifikaci o potvrzení jeho rezervace na lekce trenéra se jménem Coach Tester.

v kapitole 5.1.4. Po přidání této funkcionality do aplikace a na server je možné zasílat následující notifikace:

- **vytvoření rezervace lekce** — trenér obdrží notifikaci obsahující jméno sportovce a hlášku oznamující o nové rezervaci (obrázek 6.4 vlevo),
- **potvrzení nebo zamítnutí rezervace** — sportovec obdrží notifikaci, že trenér s daným jménem schválil nebo zamítl jeho rezervaci lekce (obrázek 6.4 vpravo),
- **zrušení rezervace** — trenér zruší rezervaci a sportovec obdrží upozornění nebo ji zruší sportovec a trenér obdrží upozornění.

Kapitola 7

Závěr

Výsledkem této práce bylo vytvořit mobilní aplikaci pro operační systém Android, která umožní urychlit a obstarat kompletní proces a správu rezervací sportovních lekcí nejen pro trenéry, ale i pro potenciální klienty. Pro úspěšné zpracování této práce jsem musel podniknout velké množství kroků, které vedly až k tomuto závěru.

Na začátku této práce jsem musel specifikovat problémy, se kterými se trenéři nejčastěji potýkali při správě svých rezervací. Na základě těchto problémů jsem navrhl řešení, které by aplikace měla poskytnout.

K návrhu a vývoji mobilní aplikace pro Android bylo potřeba nastudovat nespočet nejnovější technologií. Převážně se to týkalo knihovny Android Jetpacku, samotného Androidu, programovacího jazyka Kotlin, knihoven od Googlu poskytující doplňující služby nebo asynchronní zpracování pomocí reaktivního programování. Díky těmto technologiím jsem mohl v aplikaci vytvořit architekturu MVVM, která je oficiálně doporučovaná firmou Google.

V dalším kroku jsem vytvořil kompletní návrh uživatelského prostředí, programové struktury aplikace, výběru technologií a návrhu serveru využívající Dactyl CMS. Pro návrh uživatelského prostředí jsem studoval existující řešení, které by mohlo být podobné aplikaci Fittyy. Nejvíce jsem se inspiroval službou Airbnb, která má velmi úspěšně fungující aplikaci a obsahuje podobný obsah informací. Pro vytvoření grafického návrhu aplikace jsem využil aplikaci Sketch, kterou používají profesionální grafici. Za hlavní úspěch procesu návrhu bych označil tu skutečnost, že až na textové styly a barvy, se vyvinutá aplikace Fittyy příliš neliší od návrhu. Pouze jsem neimplementoval některé navržené funkcionality a přidal jiné na základě testování. Velkou výhodou bylo využití programu Zeplin, do kterého jsem grafický návrh převedl. Mohl jsem tak efektivněji konvertovat grafický návrh do programové podoby.

Za stěžejní část této práce považuji implementaci mobilní aplikace. V základu jsem se řídil tím, co jsem vymyslel při návrhu v předchozím kroku. Převážně jsem se zaměřil na správnou implementaci architektury MVVM, což se mi podle mého názoru povedlo. Díky použité knihovně Dagger a zmíněné architektury je aplikace velmi snadno rozšířitelná pro budoucí účely a funkce. Implementace aplikace velmi úzce souvisela s využitím grafických návrhu. Zde jsem použil knihovnu pro Material Design. To mělo za následek mnohem jednoduší převod jednotlivých navržených grafických komponent do programové podoby. Následně jsem provedl implementaci serveru, kde bylo zapotřebí vytvořit jednotlivé modely databáze a implementovat navržené endpointy, přes které aplikace se serverem komunikuje. Zde jsem si uvědomil, že bude asi zbytečné implementovat určité věci, které jsem v návrhu udělal. Proto jsem v aplikaci provedl pár změn a již nevyvinul např. vyhledávání trenérů

a zobrazení jejich profilů. Díky tomu jsem mohl aplikaci vydat na Google Play a věnovat se jiným vylepšením.

Poslední krok zahrnoval testování aplikace, kde jsem chtěl zjistit, zda aplikace splňuje očekávání trenérů. Aplikaci jsem jim tedy poslal a provedl s nimi testování ve dvou iteracích, které velmi pomohly k tomu, aby aplikace byla co nejvíce užitečná pro jejich potřeby. Za nejdůležitější změny v aplikaci, které jsem udělal v rámci testování, bych označil zpřehlednění nastavení dostupnosti trenéra u lekce, vytvoření vlastní události za účelem zablokování časového úseku dostupnosti trenéra a nakonec přidání notifikací týkající se nových rezervací nebo změn jejich stavu. Dle mého názoru bych testování označil za velmi přínosné, protože mi pomohlo k naplnění cíle této práce.

V rámci této práce jsem si vyzkoušel procesy, které v reálném životě a firmě vykonává hned několik rolí. Jako analytik jsem musel zanalyzovat problémy trenéru a specifikovat co přesně by měla aplikace obsahovat. Následně jsem nastudoval potřebné technologie a postupy, které mě vedly k tomu, abych jako grafik navrhl aplikaci. Na základě návrhu jsem v roli vývojáře vytvořil aplikaci s fungujícím serverem. Nakonec jsem si vyzkoušel pozici testera, kdy jsem již při vývoji pravidelně testoval a vylepšoval jednotlivé části aplikace. Jakmile byla aplikace stabilní, nahrál jsem ji na Google Play[1]. Konečnou fázi bylo testování s trenéry, kde jsem zjistil důležité nedostatky, které jsem do aplikace přidal.

Co se týká dalšího vývoje, určitě mám v plánu do aplikace přidat možnost prezentace profilů trenéra a jejich vyhledávání. Chtěl by také zanalyzovat, zda by uživatelé ocenili, kdyby se v aplikaci vyskytl seznam článků a novinek, které by se týkaly sportu nebo fitness dění. Dále bych se rád zaměřil na propagaci aplikace a další testování s trenéry, které by se ještě více zaměřilo na jejich potřeby.

[1] https://play.google.com/store/apps/details?id=cz.czechapps.fittyy

Literatura

[1] AA, T.: *Android 9 Pie update tracker: When will your phone get it?* [Online; navštíveno 3.1.2019].
URL https://www.androidauthority.com/android-9-0-update-880718/

[2] Airbnb: *Airbnb Host Design.* [Online; navštíveno 3.1.2019].
URL https://blog.atairbnb.com/new-mobile-app/

[3] Airbnb: *Airbnb Press Room.* [Online; navštíveno 3.1.2019].
URL https://press.airbnb.com/fast-facts/

[4] Airbnb: *Building a Visual Language.* [Online; navštíveno 3.1.2019].
URL https://airbnb.design/building-a-visual-language/

[5] Airbnb: *DLS Components.* [Online; navštíveno 3.1.2019].
URL https://airbnb.design/wp-content/uploads/2016/04/Organized-airbnb-dls-components-sample.png

[6] Airbnb: *What's Next for Mobile at Airbnb.* [Online; navštíveno 3.1.2019].
URL https://medium.com/airbnb-engineering/whats-next-for-mobile-at-airbnb-5e71618576ab

[7] Alt, A.: *Lessons from converting an app to 100% Kotlin.* [Online; navštíveno 6.1.2019].
URL https://medium.com/keepsafe-engineering/lessons-from-converting-an-app-to-100-kotlin-68984a05dcb6

[8] By Yun Cheng, A. O. D.: *Advanced Android App Architectures.* Razeware LLC, 2018.

[9] Dupress, K. M.: *RxJava for Android App Development.* O'Reilly Media, Inc., 2015, ISBN 978-1-491-93933-8.

[10] Facebook: *React Native.* [Online; navštíveno 3.1.2019].
URL https://facebook.github.io/react-native/

[11] Google: *Android Jetpack.* [Online; navštíveno 11.1.2019].
URL https://developer.android.com/jetpack/

[12] Google: *Distribution dashboard.* [Online; navštíveno 3.1.2019].
URL https://developer.android.com/about/dashboards/

[13] Google: *Material Design.* [Online; navštíveno 28.12.2018].
URL https://material.io/design/introduction/

[14] Lacko, L.: *Vývoj aplikací pro Android*. Albatros Media a.s., 2015, ISBN 978-80-251-4347-6.

[15] Leiva, A.: *Dependency injection on Android: Dagger*. [Online; navštíveno 11.1.2019]. URL https://antonioleiva.com/dependency-injection-android-dagger-part-1/

[16] Leiva, A.: *Kotlin for Android Developers*. Leanpub, 2017.

[17] Mew, K.: *Android Design Patterns and Best Practice*. Packt Publishing, 2016, ISBN 978-17-864-6721-8.

[18] Nielsen, J.: *Iterative User Interface Design*. 1993, [Online; navštíveno 3.1.2019]. URL https://www.nngroup.com/articles/iterative-design

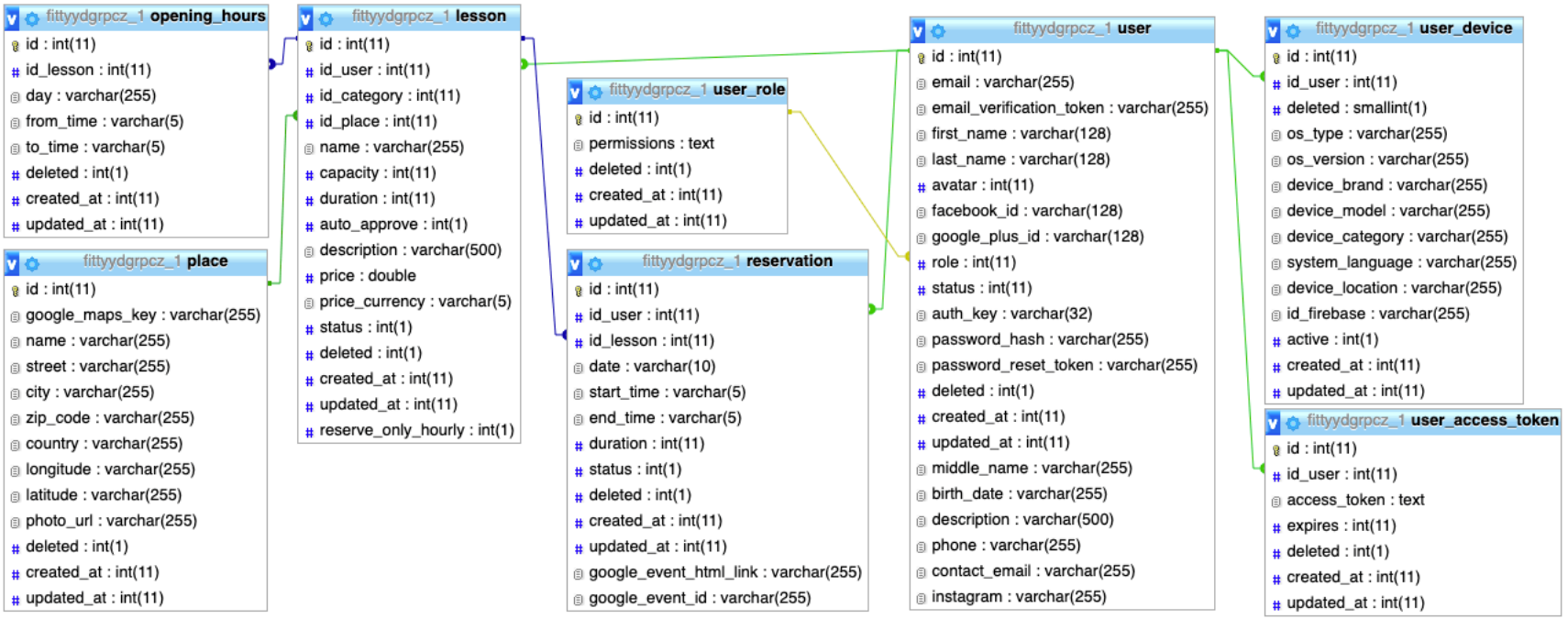
[19] Norman, D. A.: *Design pro každý den*. Dokoran, 2010, ISBN 978-80-7363-314-1.

[20] Weinschenk, S.: *100 Things Every Designer Needs to Know About People*. New Riders, Berkeley, CA 94710, 2011, ISBN 978-0-321-76753-0.

[21] Yii: *Yii PHP framework*. [Online; navštíveno 7.5.2019]. URL https://www.yiiframework.com/

Příloha A

ER diagram databáze serveru



Obrázek A.1: Návrh databáze serveru pomocí ER diagramu.

Příloha B

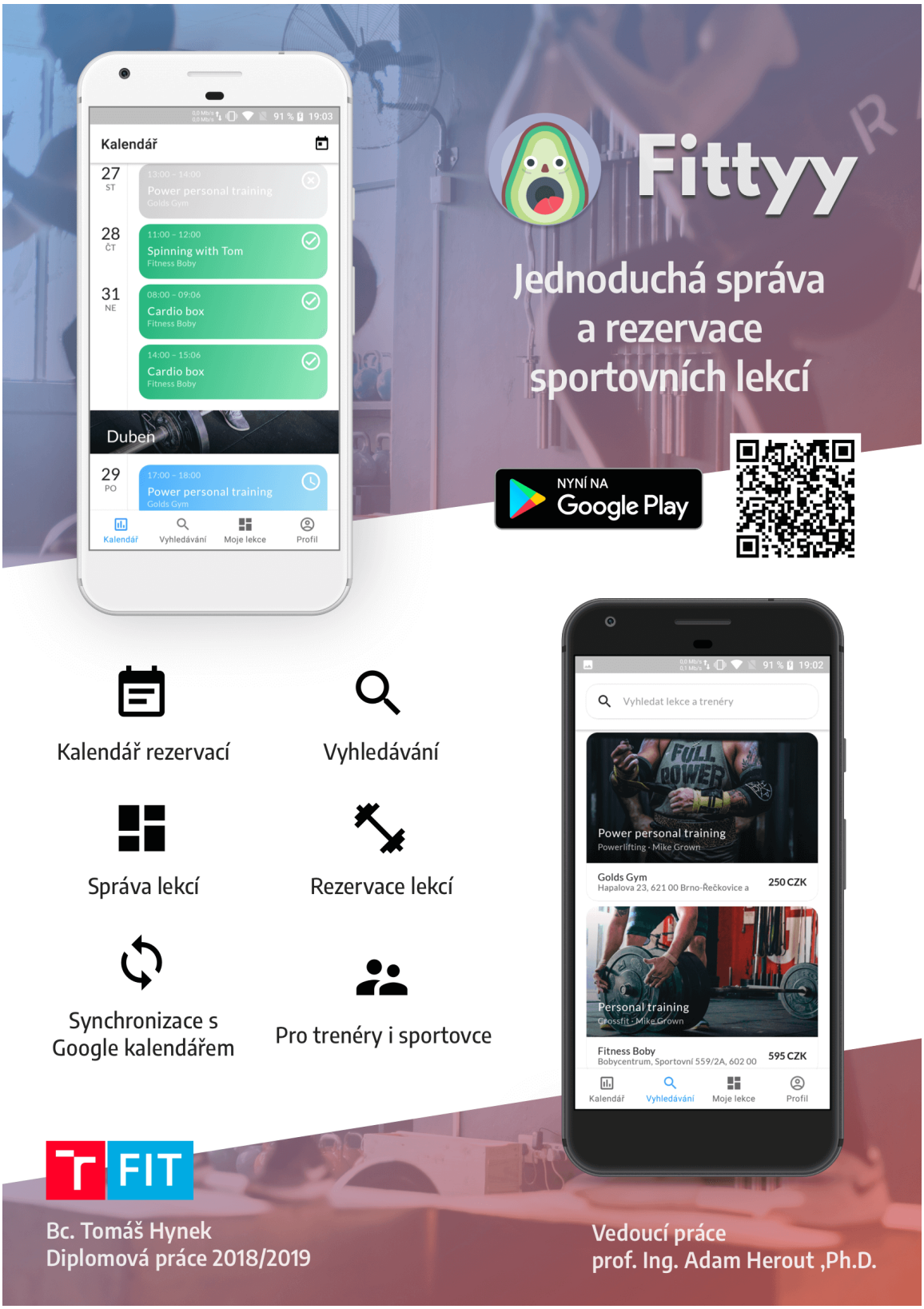
Obsah CD

Přiložené CD obsahuje následující složky a soubory:

- Fittyy_1.0.2_19051600_devRelease.apk - instalační soubor mobilní aplikace komunikující s testovacím serverem,
- FittyyAndroid.zip - komprimovaná složka obsahující zdrojové kódy aplikace,
- FittyyServer.zip - komprimovaná složka obsahující zdrojové kódy serveru,
- Latex.zip - komprimovaná složka obsahující zdrojové kódy pro LaTeX,
- xhynek09_dp.pdf - elektronická verze diplomové práce,
- xhynek09_dp_tisk.pdf - tisková verze diplomové práce,
- poster_A1.png - prezentační plakát mobilní aplikace,
- poster_A1.pdf - prezentační plakát mobilní aplikace,
- promo_video.mp4 - propagační video,
- README.txt - popis instalace aplikace,
- LICENSE.txt - licence aplikace.

Příloha C

Plakát



Obrázek C.1: Prezentační plakát mobilní aplikace Fittyy.