STŘEDOŠKOLSKÁ ODBORNÁ ČINNOST

Obor č. 14: Pedagogika, psychologie, sociologie a problematika volného času

Tréninkový deník běžce na lyžích

Martin Lank
Liberecký kraj

Liberec 2017

STŘEDOŠKOLSKÁ ODBORNÁ ČINNOST

Obor č. 14: Pedagogika, psychologie, sociologie a problematika volného času

Tréninkový deník běžce na lyžích

A cross-country skier's training logbook

Autor **Martin Lank**

Škola **Střední průmyslová škola strojní a elektrotechnická a Vyšší odborná škola, Liberec 1, Masarykova 3**

Kraj **Liberecký**

Konzultant **Ing. Jiří Pliska**

Liberec 2017

Prohlášení

Prohlašuji, že jsem svou práci SOČ vypracoval/a samostatně a použil/a jsem pouze prameny a literaturu uvedené v seznamu bibliografických záznamů.

Prohlašuji, že tištěná verze a elektronická verze soutěžní práce SOČ jsou shodné.

Nemám závažný důvod proti zpřístupňování této práce v souladu se zákonem č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) v platném znění.

V Liberci dne 28. 3. 2017 ……

Martin Lank

Poděkování

Chtěl bych poděkovat hlavně rodině za podporu a panu Ing. Pliskovi za konzultace. Také děkuji kamarádovi Davidovi Zakouřilovi za vytvoření ikony aplikace.

Anotace

Práce se zabývá vývojem nativní Android aplikace určené běžcům na lyžích. Ta má za úkol ulehčit evidenci tréninkového deníku a přispět tak ke zvýšení jeho relevantnosti. Práce vychází z mých několikaletých zkušeností a poznatků z pozice závodníka a snaží se eliminovat nepříjemnosti spojené s evidencí tréninkového deníku.

Aplikace je navržena s ohledem na strukturu oficiálního Excel dokumentu pro evidenci elektronického tréninkového deníku Českého svazu lyžařů. Výhoda zadávání v aplikaci je, že se tréninky evidují nezávisle na sobě, kdežto v Excelu se všechny tréninky téhož dne evidují kombinovaně. Tréninky je možné upravovat, hromadně zobrazit a filtrovat dle několika kritérií. Nezbytnou funkcí je také možnost exportu do již zmíněného oficiálního dokumentu od svazu.

Samotná aplikace je postavena na návrhovém vzoru MVP, využívá reaktivního programování a ctí Material design guidelines od Googlu.

Klíčová slova

tréninkový deník; běh na lyžích; evidence; Android; Java

Annotation

This work deals with the development of a native Android app designed for cross-country skiers. This app's job is to make the recording of a training logbook easier and thereby help to increase its relevancy. The work has been based on my own personal experience and on my findings from a racer's point of view, with the aim of eliminating all of the hassle connected with logging.

The app is designed with the consideration of the structure of the official Excel file for logging electronic logbook, managed by the Czech Ski Association. The advantage of logging in this app is the fact that the sessions are logged independently of the others, whereas in Excel all the sessions done in a day are logged into a single row. Training sessions are editable and viewable in a filterable list by a few criteria. An essential function is the possibility to export data into the aforementioned Excel file.

The app itself is built on MVP design pattern, it uses reactive programming and follows Google's Material design guidelines.

Keywords

training logbook; cross-country skiing; evidence; Android; Java

Obsah

Seznam použitých zkratek a výrazů

Anotace	slouží k přiřazení třídě, metodě nebo poli dodatečné informace/příznaku, na jejímž základě se může ovlivnit chování programu
API	Application Programming Interface
AS	Android Studio
BE	běh
Bindování	z angl. binding - svázání, propojení View s definicí v XML layoutu
BK	Butter Knife
Build	proces sestavení aplikace do Androidem spustitelného souboru APK
DAO	Data Access Object
DB	databáze
DZ	den zátěže
FAB	Floating Action Button
GPS	Global Positioning Systém, systém k určování polohy
HTTPS	Hypertext Transfer Protocol Secure
HZ	hodin zátěže
IDE	Integrated Development Environment, česky vývojové prostředí
IM	imitace
JSON	JavaScript Object Notation je formát pro přenos dat
JZ	jednotka zátěže
KL	kolečkové lyže
Layout	rozvržení UI
Logování	zaznamenávání dat v průběhu programu k analýze
LY-K	lyže klasicky
LY-V	lyže volně
Mapování	převod dat mezi databází a POJO
Minifikace	proces, při němž dochází k odstraňování nevyužitého kódu
MVP	Model View Presenter, návrhový vzor
Obfuskace	úprava kódu do podoby nečitelné člověkem
OFL	Open Font Licence
ORM	Object-relational Mapping
OS	obecná síla
OTD	oficiální TD
OTP	One Time Password, jednorázové heslo
POJO	Plain Old Java Object, tj. prostý objekt bez dědičnosti, implementace rozhraní a anotací (nezahrnuje anotace polí)
ProGuard	nástroj pro obfuskaci a minifikaci aplikace
Relfexe	zpětné zjišťování struktury objektu za běhu programu
RP	reaktivní programování
Rx	Reactive Extension
SCM	Sportovní centrum mládeže
SLČR	Svaz lyžařů České republiky

Sport-tester	zařízení pro záznam dat během tréninku (např. TF, časy, vzdálenost)
SQL	Structured Query Language
SS	speciální síla
Stetho	debuggovací nástroj
TD	tréninkový deník
TF	tepová frekvence
toolbar	panel nástrojů
UI	User Interface
UML	Unified Modeling Language
URL	Uniform Resource Locator
UUID	Universally Unique Identifier
UX	User Experience
View	základní třída pro všechny UI prvky nebo každý prvek, který od ní dědí
XML	Extensible Markup Language, typ datového souboru, jsou v něm definovány layouty

Úvod

Snad každý sportovec na vyšší, než rekreační úrovni, se někdy setkal s takzvaným tréninkovým deníkem (TD). V něm se sčítají tréninkové dávky, nemoci a zranění, což jsou cenná data pro pozdější analýzu. Nicméně, ať už si sportovec deník vede z vlastní iniciativy nebo pod nátlakem trenéra, jde o činnost velice nezáživnou a mnohdy i značně zdlouhavou. Kvůli tomu mají sportovci často sklony k odkládání této činnosti a dopisují TD zpětně. V lepším případě jeden či dva dny, v tom horším týden nebo dokonce několik měsíců. Deník pak ztrácí vypovídající hodnotu, což je samozřejmě špatně.

V dnešní době jsou chytrá mobilní zařízení nedílnou součástí našich životů. Máme je neustále při sobě, neboť v mnoha ohledech již zcela dokáží nahradit osobní počítače a jsem si jist, že tento trend díky rozvoji vysokorychlostního internetu v mobilním telefonu se bude i nadále zvyšovat. Není tedy divu, že i při cestách z tréninků či závodů mají sportovci telefon stále u sebe. A to je ideální chvíle pro zapsání TD.

Proto jsem si vybral toto téma, abych konkrétně běžcům na lyžích v co největší míře zpříjemnil a usnadnil evidenci TD v podobě mobilní aplikace, kterou mohou využít třeba právě na cestách. A jelikož se zajímám o vývoj hlavně na platformě Android, je aplikace navržena právě pro ni.

1 Tréninkový deník

Jak jsem již v úvodu naznačil, TD je místo, kam si sportovec zaznamenává veškerou aktivitu. V této části se budu věnovat elektronickému tréninkovému deníku určenému běžcům na lyžích. Tento TD je v podobě Excel dokumentu a je veřejně dostupný na webu českého svazu lyžařů (SLČR[1]). Dále pro něj budu užívat zkratku OTD (Oficiální TD).

1.1 Význam

TD je užitečný zejména pro vrcholové sportovce a lidi, kteří chtějí mít o svých aktivitách přehled. Správná evidence je klíčová pro pozdější analýzu a zpětnou vazbu. Je například možné odhalit, že sportovec bývá opakovaně nemocný každý podzim. Zřejmě je tedy náchylnější na chřipky a měl by zvážit možnosti, jak jí předejít. Nebo je možné vyzkoumat, že se opakovaně dostává do závodní formy až v polovině zimní sezóny. To může znamenat například špatně zvolené tréninkové dávky a pro trenéra to je signál, že se něco dělalo špatně už na letních soustředěních a je potřeba to změnit. TD je vhodný i pro prosté zkoumání a porovnávání odtrénovaných hodin. Proč se nedostavují výsledky i přes svědomitou přípravu? Možná proto, že v porovnání s ostatními sportovec stále trénuje o deset hodin měsíčně méně. Ročně to je sto dvacet hodin, což již má na sportovcovu výkonnost značný vliv.

Kromě toho se evidence TD pro některé nadějné sportovce stává i povinností, a to ve chvíli, kdy jsou na základě úspěchů a výkonnosti přijati do dotačních programů (například Sportovní Centrum Mládeže – SCM). Správná evidence totiž bývá jednou z podmínek členství.

1.2 Analýza tréninkového deníku od SLČR

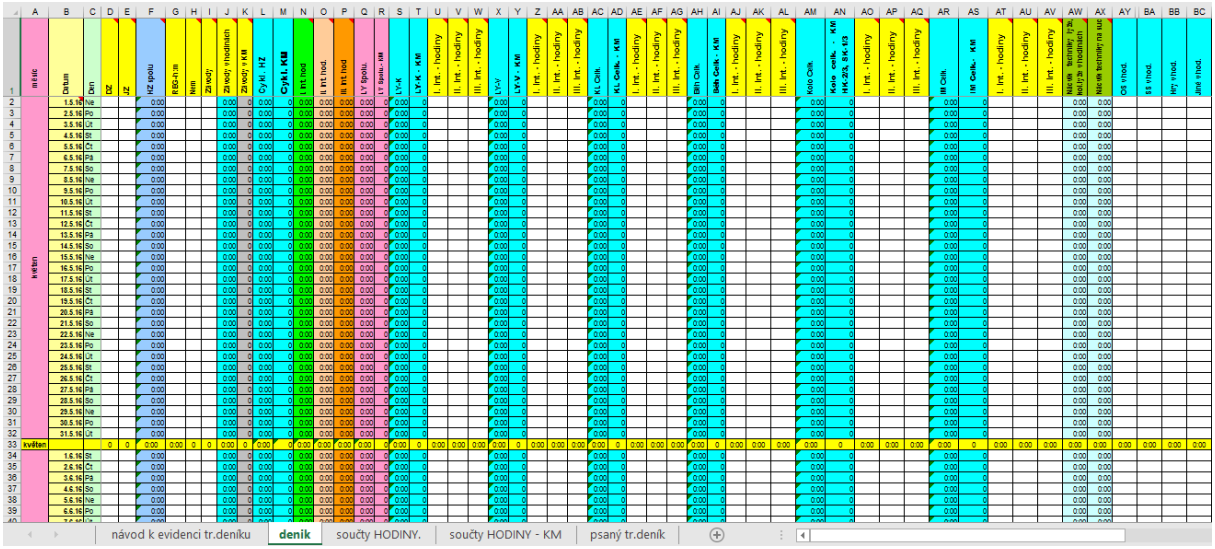
Pokud chceme z TD získat kvalitní zpětnou vazbu, je zapotřebí TD pravidelně a poctivě evidovat. Jaké údaje se zaznamenávají a s jakými nepříjemnostmi se při zadávání sportovec setkává, popíši podrobněji v několika následujících podkapitolách.

[1] Běh na lyžích - Dokumenty. Svaz lyžařů ČR [online]. 2017 [cit. 2017-02-19]. Dostupné z: http://www.czech-ski.com/beh-na-lyzich/metodika

1.2.1 Části TD

TD je rozdělen na pět částí/listů. První část slouží jako návod k části druhé. Je v ní vysvětleno, co do kterých kolonek patří a obsahuje také několik příkladů. Tato část je vhodná zvláště pro osoby začínající s evidencí.

Druhá část je jednou ze stěžejních částí celého TD. Zadávají se zde veškeré číselné údaje, a to bez jakýchkoli popisků, komentářů a podobně. Proto je vyplňování této části ze všech nejsložitější.



Obrázek 1 – Ukázka TD
Zdroj: vlastní

Podívejme se na obrázek 1. Jak si můžeme všimnout, každý řádek v tabulce odpovídá jednomu dni v roce. V běhu na lyžích sezóna začíná 1. května a končí následující rok 30. dubna. Dále je deník po svislé ose rozdělen na měsíce[2]. Vždy na konci měsíce je celý řádek vyhrazený součtům hodnot za celý měsíc. Sloupce lze pomyslně rozdělit na základní část (B-E), kde se vyplňují údaje týkající se dne jako celku, a část hlavní (F a dále), kde se zapisují konkrétní časy k aktivitám, ze kterých se trénink skládá. Nyní si postupně probereme všechny sloupce obou částí.

[2] Dříve místo měsíců byly ještě tzv. cykly, přičemž každý měl 28 dní. Od toho se ale opustilo a zůstaly jen měsíce.

	A	B	C	D	E	F	G	H	I	J	K
1	měsíc	Datum	Den	DZ	JZ	HZ spolu	REG-h:m	Nem	Závody	Závody v hodinách	Závody v KM
2		1.5.16	Ne			0:00				0:00	0
3		2.5.16	Po			0:00				0:00	0
4		3.5.16	Út			0:00				0:00	0
5		4.5.16	St			0:00				0:00	0
6		5.5.16	Čt			0:00				0:00	0
7		6.5.16	Pá			0:00				0:00	0
8		7.5.16	So			0:00				0:00	0
9		8.5.16	Ne			0:00				0:00	0
10		9.5.16	Po			0:00				0:00	0
11		10.5.16	Út			0:00				0:00	0
12		11.5.16	St			0:00				0:00	0

Obrázek 2 – Detail základní části
Zdroj: vlastní

Jak můžeme vidět na obrázku 2, po datu a názvu dne v týdnu se nachází sloupec D – DZ (den zátěže) a E – JZ (jednotka zátěže). JZ se počítá jako každá aktivita trvající alespoň 30 minut, přičemž abychom mohli počítat více aktivit, musí mezi nimi být odpočinek minimálně jednu hodinu. DZ se vyplní tehdy, pokud počet JZ v daný den je vyšší než nula. HZ spolu (hodin zátěže spolu) je celkový počet odtrénovaných hodin v daný den. Pokud jsme daný den měli nějakou cílenou regeneraci jako například masáž či saunu (tzn. ne pouze volný den), zaznamenává se celkový čas do sloupce „REG-h:m“. Do dalšího sloupce se zapíše jednička, pokud jsme v daný den nemocní nebo omezení z tréninku vlivem zranění. Počet závodů se pak pochopitelně píše do sloupce závodů. Obecně se v TD vyplňují jen bíle zbarvené buňky, ale existují výjimky, jako například poslední dvě položky základní části: přesný čas závodu a jeho kilometry. Nyní se podíváme na část hlavní. Aby to bylo více srozumitelnější, začnu nyní odzadu.

L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	AA	AB
Cykl. HZ	Cykl. KM	I. Int. hod	II. Int. hod.	III. Int. hod	LY Spolu.	LY Spolu.- KM	LY-K	LY-K - KM	I. Int. - hodiny	II. Int. - hodiny	III. Int. - hodiny	LY-V	LY-V - KM	I. Int. - hodiny	II. Int. - hodiny	III. Int. - hodiny
0:00	0	0:00	0:00	0:00	0:00	0	0:00	0				0:00	0			
0:00	0	0:00	0:00	0:00	0:00	0	0:00	0				0:00	0			
0:00	0	0:00	0:00	0:00	0:00	0	0:00	0				0:00	0			
0:00	0	0:00	0:00	0:00	0:00	0	0:00	0				0:00	0			
0:00	0	0:00	0:00	0:00	0:00	0	0:00	0				0:00	0			
0:00	0	0:00	0:00	0:00	0:00	0	0:00	0				0:00	0			
0:00	0	0:00	0:00	0:00	0:00	0	0:00	0				0:00	0			
0:00	0	0:00	0:00	0:00	0:00	0	0:00	0				0:00	0			
0:00	0	0:00	0:00	0:00	0:00	0	0:00	0				0:00	0			
0:00	0	0:00	0:00	0:00	0:00	0	0:00	0				0:00	0			
0:00	0	0:00	0:00	0:00	0:00	0	0:00	0				0:00	0			

Obrázek 3 – Detail hlavní části I.
Zdroj: vlastní

Žluté buňky se vždy vážou na cyklickou aktivitu. Zjednodušeně řečeno to jsou ty, které jsou pro běžce na lyžích specifické nebo je to jejich častý tréninkový prostředek (např. kolo). Zde na obrázku 3 se nacházejí dva zástupci - lyže volně (LY-V) a lyže klasicky (LY-K). Dále existují ještě kolečkové lyže (KL), běh (BE), kolo a imitace (IM). Každá z těch tří žlutých buněk reprezentuje určitou intenzitu zátěže. Ty jsou tři a jsou dány procentuálním rozsahem z maximální tepové frekvence. Zátež do 75 % se řadí do první, mezi 75 % až 85 % do druhé a nad 85 % do třetí. V krajních případech se intenzita určuje i dle laktátu v krvi. To, do jaké intenzity sportovec čas zapíše, musí vědět on sám dle své maximální tepové frekvence (nebo změřeného laktátu). Typicky ale například rozklus či výklus patří do první intenzity, rovnoměrný běh/jízda do druhé a sprinty, závody do třetí. Vždy to ale není tak jednoznačné. Sloupce vždy nalevo od první inteznity tvoří další výjimku co se týče zadávání. Zapisuje se celkový počet kilometrů dané aktivity. Všechny ostatní sloupce, které jsou dál nalevo od LY-K, jsou automaticky obnovované. Fialové sčítají celkový počet kilometrů za den na lyžích (skate i klasika), oranžová až zelená sčítají časy jednotlivých intenzit a poslední dva sloupce sčítají celkové časy a kilometry cyklických aktivit.

AR	AS	AT	AU	AV	AW	AX	AY	BA	BB	BC
IM Celk.	IM Celk.- KM	I. Int. - hodiny	II. Int. - hodiny	III. Int. - hodiny	Nácvik techniky v lyže, kol.lyže hodinách	Nácvik techniky na suchu	OS v hod.	SS v hod.	Hry v hod.	Jiné v hod.
0:00	0				0:00	0:00				
0:00	0				0:00	0:00				
0:00	0				0:00	0:00				
0:00	0				0:00	0:00				
0:00	0				0:00	0:00				
0:00	0				0:00	0:00				
0:00	0				0:00	0:00				
0:00	0				0:00	0:00				
0:00	0				0:00	0:00				
0:00	0				0:00	0:00				
0:00	0				0:00	0:00				

Obrázek 4 – Detail hlavní části II.
Zdroj: vlastní

Na druhém detailním obrázku (č. 4) kromě hlavní části TD cyklické aktivity IM vidíme ještě šest dalších. Tyto aktivity se v intenzitách neevidují. Ke všem se zapisuje pouze čas. Konkrétně se jedná o techniku na lyžích, nácvik techniky na suchu, obecnou sílu (OS), speciální sílu (SS), hry a jiné, kam se píše vše jinak nezařaditelné.

Třetí list TD je list časových součtů a plánů (obrázek 5). Trenér by měl stanovit počty hodin, které by se měly v daných měsících odtrénovat. Záleží jen na trenérovi, jak detailně plán rozepíše. Může stanovit celkové hodiny roční, nebo jednotlivých aktivit či intenzit. Stejně tak vidíme měsíční součty všech intenzit a všech aktivit. Pro názornost poslouží opět přiložený obrázek.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	AA
1	PLÁN A HODNOCENÍ RTC 2016 - 2017									Jméno:										rok nar:				klub:			
2		květen		červen		červenec		srpen		září		říjen		listopad		prosinec		leden		únor		březen		duben		celkem	
3		P	S	P	S	P	S	P	S	P	S	P	S	P	S	P	S	P	S	P	S	P	S	P	S	P	S
4	DZ		0		0		0		0		0		0		0		0		0		0		0		0	0	0
5	JZ		0		0		0		0		0		0		0		0		0		0		0		0	0	0
6	HZ spolu	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00
7	REG	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00
8	NEM		0		0		0		0		0		0		0		0		0		0		0		0	0	0
9	Závody		0		0		0		0		0		0		0		0		0		0		0		0	0	0
10	Záv.hod.	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00
11	Cykl.HZ	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00
12	I.	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00
13	II.	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00
14	III.	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00
15	LY spolu.	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00
16	LY-K	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00
17	I.	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00
18	II.	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00
19	III.	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00
20	LY-V	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00
21	I.	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00
22	II.	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00
23	III.	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00
24	KL	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00

Obrázek 5 – Ukázka součtů v TD
Zdroj: vlastní

Velice podobná je také čtvrtá část. Liší se jen v tom, že místo součtů časů jsou tam součty kilometrů. Jelikož tyto součtové části TD nejsou pro tuto práci důležité, více se jim věnovat nebudu. Velmi důležitá je ale část psaná, tedy pátá, a také poslední. Znázorňuje ji obrázek 6.

	A	B	C	D	E	F	G	H
1	2016-2017			trénink	popis tréninku	místo	TF - R/V	poznámky
2	neděle	1.5	R:					
3			D:					
4			O:					
5			V:					
6	pondělí	2.5	R:					
7			D:					
8			O:					
9			V:					
10	úterý	3.5	R:					
11			D:					
12			O:					
13			V:					
14	středa	4.5	R:					
15			D:					
16			O:					
17			V:					

Obrázek 6 – Ukázka psaného TD
Zdroj: vlastní

Psaná část je důležitá proto, že zde sportovec eviduje v psané formě to, co doopravdy dělal. V druhé části se totiž evidovala pouze čísla, což je z druhého pohledu pouze změť čísel, ze které nepoznáme, co jsme doopravdy dělali. Jde tu o to slovně i s časy, popř. kilometry popsat trénink. Z obrázku je patrné, že na každý den připadají čtyři řádky. To proto, že denně lze reálně odtrénovat pouze čtyři tréninky, respektive fáze. Fáze jsou značeny počátečními písmeny denní doby: R – ráno, D – dopoledne, O – odpoledne, V – večer. Ke každé fázi můžeme zaznamenat nějakou aktivitu/trénink. V kolonce popis tréninku by mělo být pár slovy stručné shrnutí tréninku. Místo tréninku a poznámky myslím nepotřebují komentář. Důležitý je ale sloupec tepových frekvencí (TF). Tepy se nevážou na fázi tréninku, ale na vstávání a usínání. Měří se ranní tepy ihned po probuzení a večerní po ulehnutí a následném uklidnění organismu na lůžku. V praxi se tyto údaje evidují jen na soustředěních, málokdo to dělá každý den. Ale dá se z toho poznat přetrénovanost, či zda se nezačíná v těle rozvíjet onemocnění, kterému můžeme vhodnou odezvou předejít.

1.2.2 Příklad evidence reálného tréninku

Již víme, jak trénink evidovat a nyní si na reálných příkladech ukážeme, jak vypadá evidence v praxi:

Trénink č. 1

- Doprava na kole na trénink – 20 min, 8 km (II. intenzita, pospíchal jsem)
- Rozklusání – 10 min, 1,5 km
- Protažení, rozcvičení (atletická abeceda) – 15 min
- Kombinovaný běh se sílou: běh 6x800 m rovnoměrně (4 min kolo), mezi koly 5 min posilování – ob kolo s expandéry (ruce; SS) a vlastní vahou (břicho, záda; OS)
- Na závěr 10 min vyklusání, 1,5 km
- Protažení 10 min
- Cesta domů 25 min, 8 km (teď už I. intenzita)

Trénink č. 2

- Rozklusání 20 min, 3 km
- Zapracování, abeceda, protažení 12 min
- Výběhy schodů 12 min (frekvenčně, III. intenzita, 2 min silově nohy)
- Klusání 6 min, 800 m
- Znovu výběhy 12 min
- Běh 10 min, 1,5 km (rovnoměrný běh, II. intenzita)
- Posilování v posilovně 40 min
- Protažení 7 min

Tak, jak trénink vidíte zapsaný, by správně měla vypadat psaná část TD dané fáze. Tu sportovec musí samozřejmě napsat sám, takto zapsaný tréninkový plán žádný trenér nedává. Všimněte si upřesněných intenzit v případech, kde to není zcela jasné. V TD by se první trénink do řádku dne v časové části rozepsal následovně (slovo intenzita je za římskými čísly pro zkrácení vynecháno):

- DZ a JZ – 1
- BE – 20 min v I. 24 min v II., 6,2 km
- kolo – 25 min v I., 20 min v II., 16 km
- OS – 15 min
- SS – 15 min
- jiné – 25 min

Celkem: 2 h 24 min

Zápis druhého tréninku by vypadal podobně:

- DZ a JZ – 1
- BE – 36 min v I. 24 min v III., 5,3 km
- OS – 44 min
- jiné – 19 min

Celkem: 1 h 59 min

Pokud by tréninky byly ve stejný den (dopolední a odpolední fáze), počet JZ by se změnil na dvě a odpovídající hodnoty časů by se sečetly.

1.2.3 Problémy s evidencí TD

V této části se pokusím popsat filozofii evidence, to jak sportovec při zadávání smýšlí a jak samotné vyplňování probíhá prakticky. V návaznosti poté vysvětlím problémy, které sportovce při vyplňování trápí.

Nejproblematičtější je evidence tréninků. Zaměřím se tedy na ni a využiji k tomu dva uvedené příklady. Po otevření TD sportovec povětšinou vybere list pro záznam časů. To protože se mu do té psané nechce. Já osobně jsem prvně psal psanou část tehdy, když jsem věděl, že v daný den bylo více tréninků nebo byl trénink hodně složitý. To bývá typicky na soustředěních. Jinak jsem vždy začínal časovou částí. Pak tedy následuje vyplnění DZ a JZ, případně pouze zvýšení JZ, pokud již máme nějaký trénink zaznamenaný. Jestliže jsme byli v tréninku nějakým způsobem omezeni vlivem zranění, musíme zapsat jedničku i do nemoci. V poznámce pak musíme uvést, o co se jednalo. Nyní začíná ta nejbolestivější část. Sportovec si v hlavě postupně začíná rekapitulovat trénink. Nejdříve jel na trénink na kole, ale protože nestíhal, tak zapsal těch 20 minut do druhé intenzity. Podle hodinek zjistil, že cesta byla dlouhá 8 km, a tak to tam doplnil také. Pak přijel na trénink a šel se rozklusat. Najel tedy na kolonku běhu a zapsal 10 minut do I. intenzity a zapsal 1,5 km. Stejně tak zapsal pak rozcvičení. Nyní hlavní část tréninku, která je nejsložitější. Podle „sport testeru“ zjistí, že jedno kolo běžel zhruba 4 minuty. Vynásobí si to tedy šesti a doplní do běhu II. intenzity. Stejně tak to udělá s pětiminutovým posilováním ale s rozdílem, že půlka času jde do OS a půlka do SS. Ne vždy to jde ale takto lehce udělat, poněvadž jsou často opakované tréninky, ve kterých se kola běhají s různou intenzitou. Například běh na 65 %, 85 %, 100 %, 85 %, 100 %, 85 % a nakonec 100 %. Časy v nich se na delších tratích (> 1 km) mohou lišit i několik minut. Následující částí tréninku je

vyklusání, jehož čas musí sportovec k hodnotě, která už v kolonce běhu I. intenzity je. V tomto případě tam bylo 10 minut. Sečíst deset plus deset není problém, ale opět – existují mnohem složitější tréninky. A nyní si uvědomme, že během soustředění jsou až čtyři tréninky za den. Rázem se pak už nesčítají takto jednoduchá celá čísla. Pokud je v kolonce například z ranního a dopoledního tréninku hodnota „1:23“ (h:mm), v Excelu máme dvě možnosti, jak přičíst další hodnoty. Buď si v hlavě říci, že je to „60+23“, tedy 83 min., k tomuto číslu onu hodnotu přičíst (např. 10 min) a zapsat do buňky „0:93“. Excel si to přebere a zformátuje to na „1:33“. Druhý způsob je upravit přímo zformátované číslo a přičíst ho ke dvaceti třem. Oba způsoby jsem během evidence kombinoval, jelikož se každý hodí někdy jindy. V příkladovém tréninku zbývá už jen znovu přičíst deset minut protahování k již zapsaným patnácti a doplnit cestu domů. Tím máme hotovu časovou část. Dříve, než se přejdeme na psanou část, rád bych se ještě zmínil o možnostech upravování.

Představme si případ, kdy v půlce zadávání tréninku si nejsme jisti, zda jsme zapsali to kolečko navíc „za trest“. V ten moment bychom se měli zastavit a zadané hodnoty překontrolovat. Pokud v onen den již něco zadaného bylo, je téměř nemožné to překontrolovat (nikdo si nepamatuje původní hodnoty). Pokud je to první trénink dne, lze to provést opětovným procházením tréninku v hlavě a porovnávání zadaných hodnot (tj. spousta sčítání zpaměti). Ve výsledku je jednodušší začít znovu jako v prvním případě. Tak či onak, je to obrovské zdržení už tak zdlouhavého zadávání. Proto se to v praxi většinou přechází, což je samozřejmě špatně. Výjimkou nejsou ani případy, kdy si vůbec neuvědomíme, že jsme na něco zapomněli

V psané části je nejprve potřeba nalézt správné kolonky, tj. datum a fáze. Následuje znovu promítání celého tréninku jako v časové části. Zapisujeme chronologicky a vpisujeme co nejvíce informací. Příklad jsem již uvedl ve vzorových trénincích. Do kolonky poznámky uvedeme naše pocity z tréninku, ať už se týkají počasí, výkonnosti nebo něčeho úplně jiného.

Problémy s evidencí tréninků jsou myslím zřejmé. Pojďme si je shrnout:

1) nutnost přičítat časy a kilometry k již zadaným hodnotám

2) dvojí vyplňování stejné informace v psané a časové části (časy s intenzitami, kilometry)

3) žádná nebo minimální zpětná kontrola zadaných dat

Důsledkem výše zmíněných bodů je zadávání velmi zdlouhavé a mentálně vyčerpávající. Tím vzniká určitá chybovost zadaných dat.

1.2.4 Možnosti ulehčení evidence v aplikaci

Na základě předchozí analýzy jsem vymyslel, jak celou evidenci značně zjednodušit. V první řadě je nutné, aby se deník evidoval po trénincích, nikoliv po dnech. To nám umožní i samotný trénink dále rozdělit na dílčí části. Jinými slovy, nebude se evidovat po „časech intenzit aktivit“, ale po jednotlivých krocích/částech tréninku. Díky tomu bude nejen jednodušší a příjemnější zadávání, ale i primitivní kontrola, úprava a také třeba pozdější detailnější analýza. Dvojího vpisování stejných informací se rovněž vyhneme, a to tak, že ke každé přidávané části doplníme krátký popis (ten, co bychom normálně psali do psané části).

Přesnější evidenci může pomoci i tzv. „sport-tester“. Jedná se o sportovní hodinky s funkcemi pro měření tepové frekvence a vzdálenosti podle GPS. Sportovec si pak může trénink rozdělit podle částí na jednotlivá kola a odpadnou mu starosti se zapamatováním.

2 Vývoj aplikace

Praktickou částí této práce je aplikace. V této kapitole se podíváme na její vývoj krok za krokem. Řekneme si něco o specifikách vývoje na mobilní zařízení obecně, dále si stanovíme požadavky na aplikaci, popíšeme si její strukturu a použité technologie.

2.1 Specifika vývoje pro mobilní zařízení

Vývoj pro mobilní zařízení se od vývoje na stolní počítače v několika ohledech liší. Jedná se především o omezení, se kterými musí programátoři počítat. Pokud by tato omezení nerespektovali, vznikaly by nestabilní, pomalé aplikace s nepřívětivým uživatelským rozhraním (User interface – UI) a přehnanou spotřebou baterie. Každý z nás na takovou aplikaci určitě někdy narazil. Jistě mi dáte za pravdu, že je velmi nepříjemné takovou aplikaci užívat a v telefonu vám nevydržela déle než pár dní. Pojďme se tedy podívat, o která omezení se jedná:

- omezený výpočetní výkon
- omezená paměť
- omezená velikost obrazovky
- omezená výdrž baterie
- omezená dostupnost internetového připojení

Zmíněná omezení se týkají všech mobilních zařízení bez ohledu na platformu či značku.

2.2 Představení platformy Android

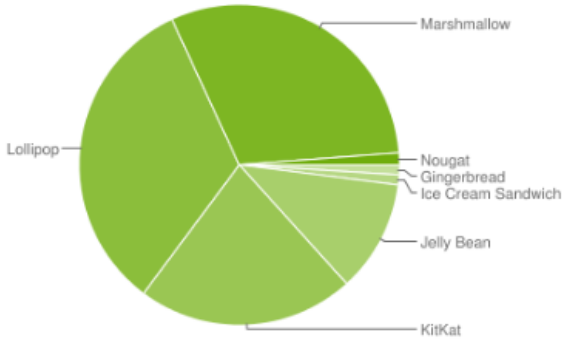
Operační systém Android je vyvíjen americkou společností Google od roku 2007. Je založen na jádře Linuxu. Primární programovací jazyk je Java (který jsem si také vybral), ale do podvědomí vývojářů se postupně dostává i jazyk Kotlin.

2.2.1 Verze

V současné době existuje dvacet pět verzí Androidu. Jednotlivé verze se označují jménem a tzv. API (Application Programming Interface) číslem. S jeho pomocí vývojáři určují, jaké verze jejich aplikace podporuje. Se zpětnou kompatibilitou Google pomáhá knihovnami - tzv. Support Libraries, díky nimž je možno využívat nejnovějších funkcí i na starších API. Podíl jednotlivých verzí můžeme vidět na přiloženém obrázku.

Version	Codename	API	Distribution
2.3.3 - 2.3.7	Gingerbread	10	1.0%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	1.0%
4.1.x	Jelly Bean	16	4.0%
4.2.x		17	5.7%
4.3		18	1.6%
4.4	KitKat	19	21.9%
5.0	Lollipop	21	9.8%
5.1		22	23.1%
6.0	Marshmallow	23	30.7%
7.0	Nougat	24	0.9%
7.1		25	0.3%

Data collected during a 7-day period ending on February 6, 2017.
Any versions with less than 0.1% distribution are not shown.



Obrázek 7 – Podíl jednotlivých verzí Androidu
Zdroj: Dashboards: Platform Versions. *Android Developers* [online]. Mountain View: Google, 2017 [cit. 2017-03-11]. Dostupné z: https://developer.android.com/about/dashboards/index.html#Platform

Z tabulky na obrázku 7 je patrné, že nejrozšířenější verze je nyní 6.0 Marshmallow, která byla představena v roce 2015. Já jsem se rozhodl podporovat API 16+, které ještě nedávno mělo zastoupení kolem 8 %.

Android je otevřená platforma, proto mimo oficiální verze existuje mnoho modifikovaných. Zejména výrobci mobilních telefonů rádi systém upravují a přidávají do

něj nejen bloatware[3], ale i přímo zasahují do jeho zdrojových kódů. To způsobuje, že i na stejných verzích API se aplikace na různých zařízení chovají jinak. Proto je nutné aplikace testovat na skutečných zařízení.

2.2.2 Vývojové prostředí

Od roku 2013 má Android své oficiální vývojové prostředí (Integrated development enviroment – IDE) - Android Studio[4] (AS). Do té doby se využíval Eclipse[5] s velkým množstvím pluginů[6], což nebylo praktické. AS je založené na IntelliJ IDEA, je v něm zabudovaný emulátor a obsahuje potřebné komponenty pro vývoj.

2.3 Funkční požadavky

Na základě analýzy tréninkového deníku a vlastního uvážení jsem stanovil funkční požadavky aplikace pro tuto práci.

F1 Zadávání tréninků

Uživatel bude moct podrobně zaznamenat trénink tak, aby obsahoval všechny potřebné části, viz OTD. Každý trénink bude povinně obsahovat informaci o místě konání, datu, stručný popis tréninku a fázi. Volitelně bude poznámka. Trénink se bude zaznamenávat po nezávislých částech. Každá část bude mít dle typu aktivity povinná a nepovinná pole. Části půjdou duplikovat, přesouvat a mazat. Tato funkce musí být nezávislá na internetovém připojení.

F2 Denní přehled

Aplikace bude umožňovat zobrazení denního přehledu tréninků. Ty půjdou mazat a upravovat. Uživatel bude mít možnost volby vybrat jiný den a musí mít vědět, jaký den je zrovna zobrazený.

[3] Bloatware – předinstalovaný software od výrobce; obecně nepopulární mezi uživateli, jelikož často nelze jednoduše odstranit, případně vůbec
[4] Download Android Studio and SDK Tools | Android Studio [online]. 2017 [cit. 2017-03-11]. Dostupné z: https://developer.android.com/studio/index.html
[5] Eclipse – The Eclipse Foundation open source community website [online]. 2017 Ontario, Canada [cit. 2017-03-11]. Dostupné z: https://eclipse.org/
[6] Plugin – modul rozšiřující funkcionalitu programu

F3 Seznam aktivit

Uživatel musí mít možnost zobrazit aktivity v seznamu. Ten musí umožňovat filtrování dle fází a tréninkového prostředku, a také možnost seřazení dle data tréninku.

F4 Export tréninků

Aplikace musí umožňovat export tréninků do OTD. Data budou exportována do časové a psané části TD tak, jako se evidovalo ručně.

2.4 Nefunkční požadavky

V návaznosti na specifika kladená na vývoj mobilních aplikací, vývoj na Androidu, obecných předpokladů pro aplikace a mých vlastních nárocích jsem stanovil nefunkční požadavky.

NF1 Lokalizace

Aplikace bude lokalizována v češtině, ale bude připravena na nenáročné dodání dalších lokalizací.

NF2 Design

Návrh designu musí ctít současné zvyklosti Google Gudelines.

NF3 Databáze

Aplikace musí tréninky ukládat do interní lokální SQLite databáze.

NF4 Technologie

Aplikace musí být verzována pomocí systému Git, musí být napsaná s využitím reaktivních principů a implementovat návrhový vzor MVP. Měla by dodržovat zásady Best Practice[7].

NF5 Obecné

Je nezbytné, aby aplikace byla stabilní, nedocházelo k nepředvídatelným pádům a při

[7] Best practice – nejlepší či osvědčená praxe; postupy, pomocí nichž několik nezávislých vývojářů dosáhlo dobrých výsledků, a doporučují tyto postupy ostatním

rotaci displeje nedocházelo ke ztrátě dat. Plynulost a svižnost je další nutný požadavek.

2.5 Použité technologie

V této kapitole se pokusím shrnout ty nejpodstatnější technologie, které jsem při vývoji této práce využil.

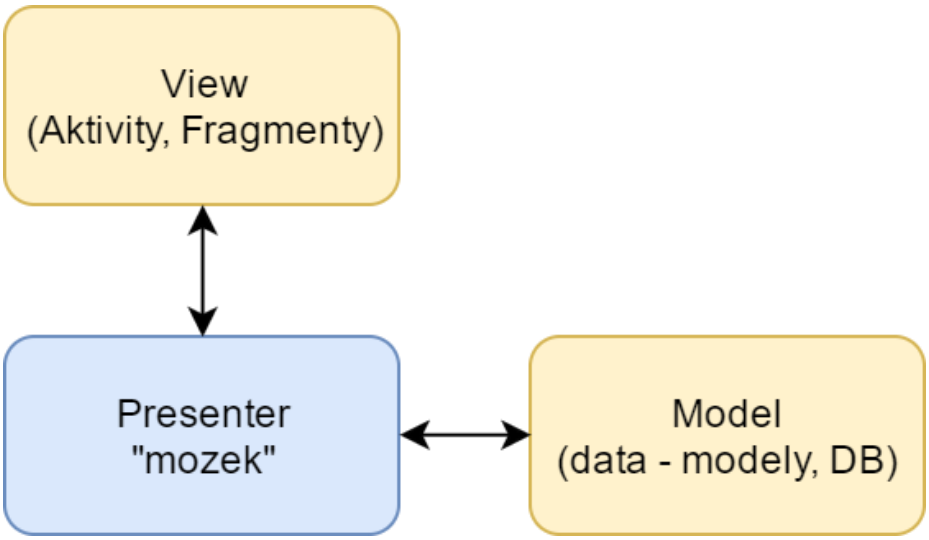
2.5.1 Git

Git je nástroj sloužící k spravování verzí souborů. Zachycuje průběh vývoje, umožňuje zpětné procházení, zpřehledňuje vývoj a zjednodušuje práci v týmu.

Git můžeme mít pouze lokální, ale nejčastější použití je se synchronizací se serverem. V mém případě jsem synchronizaci využíval čistě z bezpečnostního důvodu, jako zálohu. Jeden z nejznámějších serverů je GitHub, který ale bohužel neumožňuje privátní repositáře. Například BitBucket to umožňuje, nicméně rozhodl jsem se pro Google Cloud, jelikož lze AS spárovat právě s Google účtem a práce je tak snadnější.

2.5.2 MVP, Nucleus

Android vývojářům nevnucuje žádnou architekturu jako je tomu například u iOSu, a vývojáři tak mají úplnou volnost. Já jsem se rozhodl pro architekturu Model-View-Presenter (MVP), která rozděluje aplikaci na tři vrstvy, viz obrázek 8. Nucleus je knihovna, která tento návrhový model reaktivně implementuje, a stará se o odpojování a připojování View při rotaci zařízení.



Obrázek 8 – Schéma MVP architektury na Androidu
Zdroj: vlastní

Application flow[8] je tedy následující: View (v případě androidu je to Aktivita nebo Fragmenty, ve kterých jsou definovány UI prvky) presenteru přes interface poskytuje Observable události (viz. 2.5.4), na které naslouchá. Presenter v závislosti na zachycené události rozhoduje, co se bude dít dál. Je to mozek celé aplikace, kde je veškerá logika. Presenter pracuje s daty, které mu poskytuje modelová vrstva. Ta se stará o veškerou logiku ukládání a poskytování dat. Často proto bývá nazývána tzv. bussines logic vrstvou. Po získání dat presenter znovu notifikuje UI, což opět probíhá přes interface. Z View je možné zavolat veřejnou metodu na presenteru, ale presenter by nikdy neměl za běhu zjišťovat aktuální stav UI prvků.

2.5.3 Dagger 2 (Dependency Injection)

Vkládání závislosti se s narůstajícím kódem stává dost nepraktické a nepřehledné. Konstruktory jsou dlouhé, špatně čitelné a upravovatelné. Dagger 2 je knihovna od Googlu, která vkládání závislostí přetváří do pouhých anotací.

2.5.4 ReactiveX – RxJava, RxAndroid

Reaktivní programování (RP) je poměrně nový způsob vývoje, se kterým přišel Microsoft u svého jazyka C#. Stal se velmi oblíbeným, a tak začala vznikat tzv. reaktivní rozšíření (ReactiveX, Rx), které umožnují využívat benefitů RP i na jiných jazycích. Na androidu je

[8] Application flow – český aplikační tok; popisuje, jak aplikace pracuje.

RP poměrně čerstvou novinkou, která ihned sklidila úspěch a rázem se stala jedním z nejprobíranějších témat.

Základními prvky RP jsou Observables (= pozorovatelný) a Observers (= pozorovatelé). Observable je objekt, který emituje jednu nebo více položek, na který se Observer zaregistruje. Pozorovatelů může být několik, a jakmile je emitována nová položka, všechny pozorovatelé se notifikují. Mezi emitovanými položkami a pozorovateli mohou být tzv. operátory, které emitované položky modifikují.

```
// „čistý“ objekt
MyObject object;

// příklad Observable objektu, který emituje položky typu MyObject
Observable<MyObject> objectObservable;
```

Díky Rx je možné psát čitelnější, přehlednější a stabilnější kód. Perfektně zvládá error-handling (tj. zpracovávání chyb). Konkrétně na androidu ve velké míře zjednodušuje přepínání mezi vlákny, protože spousta věcí se musí dělat asynchronně.

Jelikož se jedná o poměrně velkou knihovnu, učící křivka Rx je poměrně strmá. Spolehlivě zde platí přísloví „dobrý sluha, zlý pán“. Neznalost návazností a propojenosti vede k neočekávanému chování aplikace, které se s ohledem na fakt, že se část děje asynchronně, velmi špatně ladí, respektive debugguje[9].

2.5.5 Lambda výrazy – Retrolambda

Při použití operátorů z Rx (a nejen jich) vzniká mnoho anonymních tříd, což bez tzv. lambda výrazů velmi prodlužuje kód. Lambda výraz umožňuje celou anonymní třídu zkrátit do jednoho řádku. Podívejme se na příklad, ve kterém nastavujeme callback[10] na kliknutí tlačítka, po jehož kliknutí chceme zavolat metodu, která zobrazí obrázek.

```
// bez lambda výrazu
button.setOnClickListener(new View.OnClickListener() {
    @Override
```

[9] Debuggování – ladění; proces sloužící k nalezení chyb v programu
[10] Callback – „zpětné volání“, tj. kód, který se spustí po provedení nějaké akce

```
    public void onClick(View view) {
        showImage();
    }
});

// s lambda výrazem
button.setOnClickListener(v -> showImage());
```

Můžeme si všimnout, že lambda výraz se skládá z proměnné, šipky a metody. Proměnná před šipkou reprezentuje proměnnou, která by v anonymní třídě byla předána jako parametr. Za šipkou pak následuje samotné volání metody. Další příklad ukazuje použití lambda výrazu, pokud metoda přímo přebírá parametr před šipkou (například pro zobrazení obrázku podle kliknutého tlačítka).

```
button.setOnClickListener(this::showImage);
```

Zde už je zápis zjednodušený na úplné maximum. Ukazatel this udává, v jakém kontextu je metoda volána. Pokud by metoda byla v jiné třídě, místo this by byl název dané třídy. Pokud bychom místo metody chtěli volat přímo kód, můžeme ho zapsat do složených závorek místo volání metody z prvního příkladu.

Některé jazyky lambda výrazy podporují nativně, některé ne. Java je podporuje od verze 8. Protože se na android vyvíjí zatím pouze ve verzi 6, knihovna Retrolambda nám zajistí zpětnou kompatibilitu.

2.5.6 SQLite, SQLBrite, SQLBriteDao

Android má nativní podporu databází SQLite3. Knihovna SQLBrite tuto nativní databázi obaluje a poskytuje k ní reaktivní přístup. A protože práce s kurzory je poněkud nízko úrovňová a mapování dat na objekty vyžaduje mnoho boilerplate[11] kódu, využil jsem ještě nástavbu SQLBriteDao. Pro každou tabulku pak existuje DAO třída (Data Access Object), ve které jsou definované metody pro práci s databází. Krom dotazů na vytvoření a upgrade databáze jsou to typicky metody, které vracejí namapované objekty. Mapování

[11] Boilerplate kód – opakující se stejný či velmi podobný kód na několika místech, který však nemůže být vynechán, zredukován či zjednodušen

neprobíhá na bázi ORM. V metodách jsou klasické SQLite příkazy, po jejíž vykonání od SQLBrite dostaneme kurzor. Kurzor je následně namapován pomocí předem vygenerovaných tříd anotačním preprocesorem na základě anotací v POJO[12] třídách. V těch si jednotlivá pole, která mají být namapována, označíme anotací s parametrem specifikující název sloupce v kurzoru (tabulce). Ve výsledku docílíme toho, že máme metody, kterým předáme objekt a ten je následně uložen do databáze a metodu, která nám objekt zase vrátí. Výsledek se tedy podobá ORM, ale bez reflexe a s větší kontrolou nad věcí.

2.5.7 Stetho

Stetho je knihovna od fy. Facebook, která umožňuje debuggování layoutů, interaktivní procházení SQLite databází či logování[13] a inspekci síťových volání. Knihovna využívá nástrojů pro vývojáře prohlížeče Google Chrome.

2.5.8 Retrofit, OkHttp

Se síťovou komunikací je opět spojeno mnoho boilerplate kódu, pokud bychom využívali standardní Java knihovny. Další skutečností je, že všechen kód pro komunikaci s internetem musí běžet mimo hlavní vlákno. A vzhledem k tomu, že v projektu používám RP, rozhodl jsem se použít knihovnu Retrofit, která umí přetvářet požadavky v Observables. Její hlavní výhodou je jednoduchost. Veškeré HTTP požadavky (endpointy) jsou definované v rozhraní. Pomocí anotací se definuje URL (Uniform Resource Locator)[14], typ požadavku (GET, POST,…), hlavičky, tělo požadavku aj. Také podporuje automatické převádění objektu na JSON (JavaScript Object Notation) a zpět. Zde se to děje pomocí reflexe. Jak už jsem řekl, odpověď lze získat jako Observable objekt. Lze ale získat i čistý objekt (viz příklad v kapitole 2.5.4). Objektem může být naše vlastní třída, nebo některá ze tříd Retrofitu, jež mohou poskytovat informace jako hlavičky, Cookies[15], stavové kódy či netknuté tělo odpovědi.

[12] POJO – Plain Old Java Object, tj. prostý objekt bez dědičnosti, implementace rozhraní a anotací
[13] Logování je zaznamenávání dat v průběhu programu k analýze.
[14] URL jednoznačně definuje doménovou adresu serveru
[15] Cookies jsou data, které server posílá klientovi, který je ukládá a znovu odesílá zpět při každém požadavku, čímž se identifikuje.

Retrofit pro své fungování potřebuje tzv. http klienta, který zprostředkovává veškerou komunikaci. K tomu se využívá knihovna OkHttp, která je zcela nezávislá na Retrofitu a lze ji použít samostatně. Umožňuje například přidávání stejných hlaviček ke každému požadavku, pokus o automatické znovu přihlášení – např. obnovování tzv. API klíčů při odpovědi „401 Unauthorized (neoprávněný)“ a následné opakování požadavku, či jednoduché nastavení vlastních SSL certifikátů, Proxy, časových limitů aj.

2.5.9 ButterKnife, RxBindings

Čím větší layout, tím více Views[16] musíme v aktivitě či fragmentu definovat. Tj. znovu se opakuje stejný kód několikrát, Views musíme přetypovávat[17]. Abych si práci ulehčil, využil jsem knihovnu ButterKnife (BK), která mi pouhou anotací spojí View objekt s její definicí v XML layoutu, kde je definované.

```
// bez použití BK
TextView txtDate =
((TextView)rootView.findViewById(R.id.add_training_text_view_date));
TextInputLayout inputPlace =
((TextInputLayout)rootView.findViewById(R.id.add_training_input_place));
TextInputLayout inputDescription =
((TextInputLayout)rootView.findViewById(R.id.add_training_input_description));
// s použitím BK
@BindView(R.id.add_training_text_view_date)
TextView txtDate;
@BindView(R.id.add_training_input_place)
TextInputLayout inputPlace;
@BindView(R.id.add_training_input_description)
TextInputLayout inputDescription;
```

Krom bindování[18] Views BK umožňuje anotací nastavovat callbacky na klik, dotyk, změnu textu a mnoho jiných událostí. To jsem ale využil zřídka kdy, jelikož jsem ve většině případů potřeboval na callbacky naslouchat v presenteru. Dalo by se to sice vyřešit voláním metody v presenteru, ale to není reaktivní přístup. Mnohem lepší je vytvořit Observables, které emitují událostí, které nás zajímají (viz předešlý příklad),

[16] Views – jakékoli UI prvky; všechny dědí ze základní třídy *View* nebo *ViewGroup*
[17] Přetypování je konverze jednoho datového typu na druhý.
[18] Bindování – z angl. binding, svázání, propojení View s definicí v XML layoutu

a z presenteru na ně naslouchat. Díky knihovně RxBindings toto nemusím implementovat sám.

2.5.10 IcePick

Při otočení obrazovky všechny aktivity zanikají a fragmenty také, pokud nejsou explicitně nastaveny jinak. To vnáší mnoho komplikací, například že se všechny globální proměnné nastaví na původní hodnoty. Proto se hodnoty proměnných, které chceme zachovat, ukládají do tzv. Bundle objektu. Ten je systémem předán jako parametr metodě onCreate, ze kterého můžeme hodnoty získat a obnovit je. Je to opět spousta kódu navíc. Knihovna IcePick tento kód generuje za nás, stačí pole, které chceme zachovat, označit anotací.

2.6 Struktura projektu

Každý projekt má po určitou úroveň pevně danou adresářovou strukturu, kterou musí dodržet, aby IDE vědělo, jaké soubory má indexovat. Od určitého adresáře to je IDE ale jedno, a veškeré strukturování modelových tříd je na vývojáři. V této kapitole nebudu až na dvě výjimky popisovat onu pevně danou strukturu, ale především vlastní strukturu mých tříd. Tu zobrazuje příloha A. Kořenový adresář leží zhruba v polovině celého stromu adresářové struktury projektu, kde se nacházejí ty nejdůležitější aplikace – zdrojové kódy, resources[19] a assets[20]. Zdrojové kódy se nachází ve složce java, ve které se nachází další složky dle jména balíčku[21]. Poslední složka tohoto stromu je kořenová složka našich zdrojových kódu. Pro přehlednost je popis struktury těchto zdrojových kódů uveden přímo v ukázce.

[19] Resources (česky suroviny) je termín, který v androidu označuje dodatečné soubory potřebné pro vývoj aplikace. Jsou to například XML layouty, obrázky (drawables), definice textů, barev nebo definice menu. Tyto suroviny se nachází ve složce *res* a lze k nim přistupovat v kódu pomocí před vygenerované třídy *R*. Při kompilaci jsou beze změny (vyjma při použití ProGuardu – záleží na nastavení) zabaleny do výsledného APK souboru.

[20] Assets jsou velmi podobné Resources. Mají složku *assets,* a též se sem ukládají dodatečné soubory. Hlavní rozdíl ale je, že k nim lze přistupovat až za běhu programu. Ukládají se sem vlastní písma, testovací XML či JSON soubory, loga, různé zvukové nahrávky aj. Soubory jsou beze změny zabaleny do APK souboru.

[21] Jméno balíčku je unikátní identifikátor jmenného prostoru (namespace) aplikace. Typická podoba je dle názvu domény vývojáře nebo organizace (tj. cz.domena.jmenoaplikace), jelikož ta právě zaručuje unikátnost při publikaci.

2.7 Návrh modelů a databáze

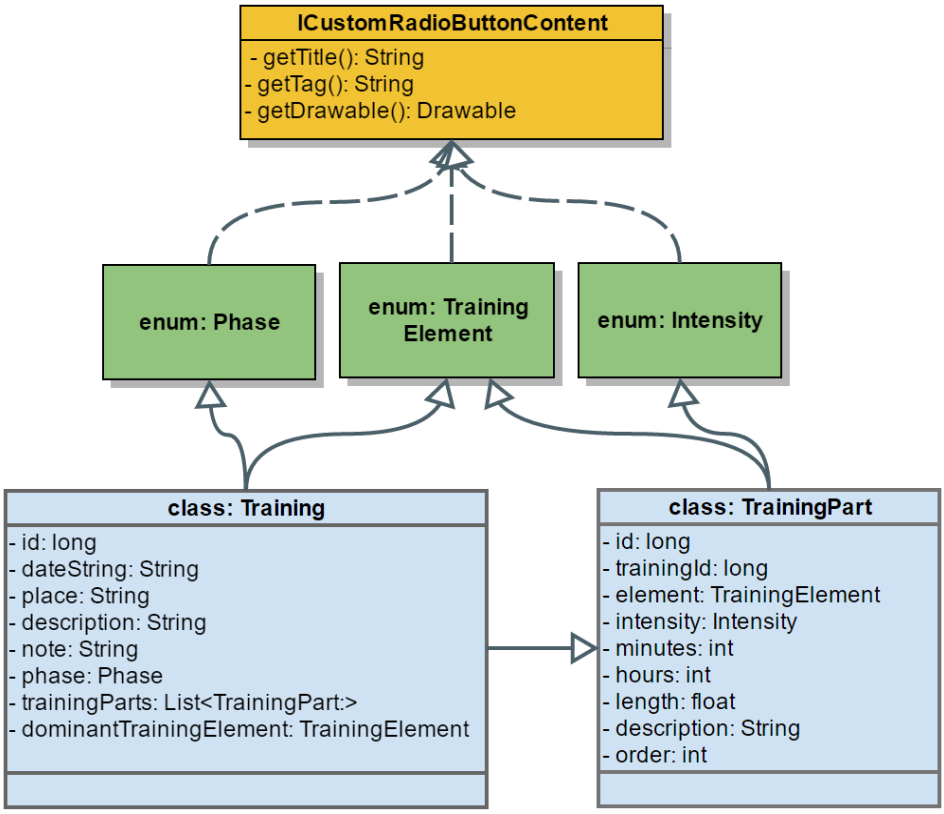
V této kapitole budu popisovat modely, které vznikly na základě analýzy TD. Mimo to samozřejmě vzniklo i mnoho pomocných modelů, které však s TD nesouvisí a jejich použití je značně specifické. V druhé části si ukážeme schéma databáze vycházejících z modelů.

2.7.1 Modelové třídy

Abychom získali přehled, vytvořil jsem zjednodušený UML (Unified Modeling Language) diagram (viz obrázek 9) který popisuje základní modely této aplikace.

Hlavním modelem je třída *Training*, která drží kompletně všechny informace o tréninku. Kromě těch základních jako jsou datum, místo apod. drží fázi, pro kterou vznikl enum[22] *Phase, a* list tzv. *TrainingParts* (tréninkových částí), kterých může v tréninku být libovolný počet a mohou se i opakovat. Tyto tréninkové části tvoří stěžejní část TD. Oba modely *Training a TrainingParts* v sobě mají pole typu *TrainingElement* (tréninkový prostředek). Ve třídě *Training* je pod názvem *dominantTrainingElement*, která je pro nás v tuto chvíli spíše dodatečnou informací, a značí nejdůležitější tréninkový prostředek v daném tréninku. V modelu *TrainingPart* však hraje zásadní roli.

[22] Enum je tzv. výčtový datový typ. Umožnuje přesně definovat hodnoty, kterých může nabývat. Jinak se chová jako statická třída, tj. můžeme v ní mít metody. Výhodou oproti statickým třídám ale je, že enum jsou singletony, tzn. v celé JVM (Java Virtual Machine) je zaručena jedna instance tohoto objektu.



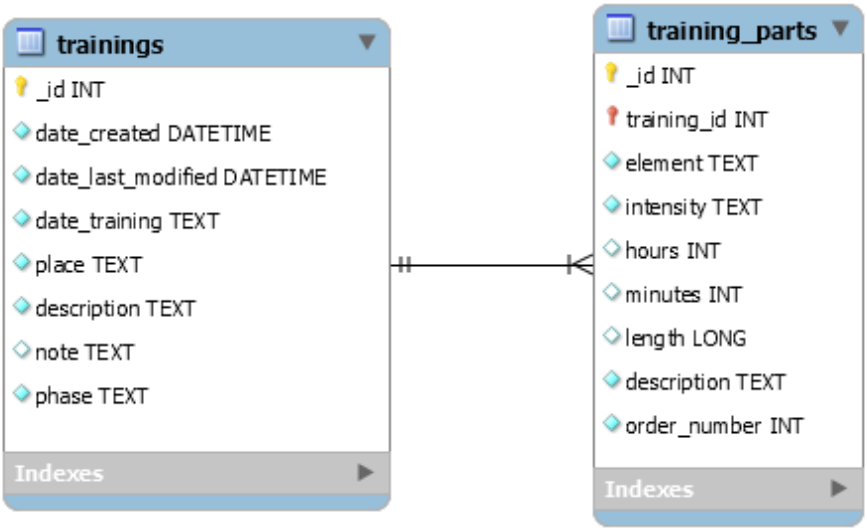
Obrázek 9 – UML diagram základních modelových tříd
Zdroj: vlastní

Dalším významným modelem je enum *Intensity*, reprezentující intenzitu dané tréninkové části. Pokud je *null*[23], je brána jako necyklická aktivita. Všechny modely typu enum implementují interface ICustomRadioButtonContent, jelikož ze všech jsou genericky vytvářeny instance vlastních RadioButtonů[24]. Ten vždy potřebuje ikonu, (tj. metoda *getDrawable*()), tzv. tag (jedinečný krátký identifikátor podobný iniciálám) a titulek.

2.7.2 Schéma databáze

Vzhledem ke skutečnosti, že je využívána SQLite databáze, je návrh tabulek databáze poměrně jednoznačný. Obecně má totiž každý model svou vlastní tabulku (a v tomto případě i Dao třídu). Oproti plnohodnotnému MySQL je SQLite značně omezena v datových typech. Proto je skoro vše typu TEXT nebo INT. Jak schéma vypadá, ukazuje obrázek 10. Dao třídy poskytují vrstvu mezi databází a objektem, jak bylo již řečeno v kapitole 2.5.6.

[23] Pokud pole či proměnná má hodnotu null, znamená to, že prvek nebyl nastaven.
[24] RadioButton je UI komponenta která je většinou součástí skupiny těchto prvků. A v této skupině jde vždy vybrat právě jeden prvek.



Obrázek 10 – Schéma databáze
Zdroj: vlastní

2.7.3 JUnit testy

Abych si byl jist, že Dao třídy vrací korektní data, pro každou byly napsány unit testy. Ty prověřují všechny metody dané Dao třídy. Každý test probíhá ve třech krocích. V prvním jsou data do databáze nejdříve zapsána, v druhém zpět získána a porovnána se vstupními daty. Příklad takové metody můžete vidět níže.

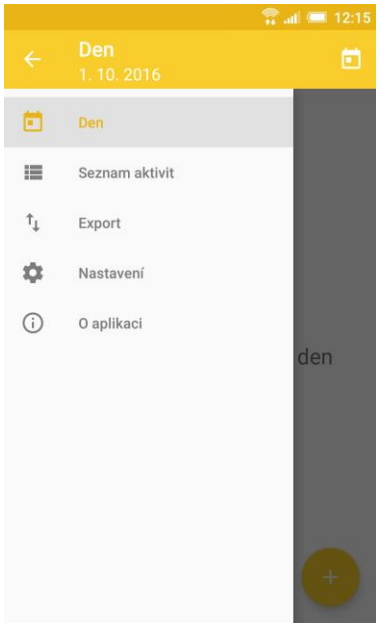
```
@Test
public void testGetTrainingsByDate() throws Exception {
    List<Training> trList = insertTrainings (); // uložení tréninků

    List<Training> list = trainingDao.getTrainingsByDateObservable(new
Date(SAMPLE_VALUE_DATE_1)).toBlocking().first(); // získání tréninku z databáze

    // zpětné ověření dat
    assertNotNull(list);
    assertTrue(list.size() == trList.size());

    for (int i = 0; i < list.size(); i++) {
        compareTrainings(list.get(i), trList.get(i));
    }
}
```

2.8 Grafika aplikace

Veškerý graficky návrh, tj. layouty obrazovek, dialogů a celkového vzhledu aplikace je čistě mou prací. Nevyužíval jsem žádného nástroje, pouze papíru a tužky. Náčrt byl pak rovnou přenesen do kódu. Hlavní ikonu aplikace pro mě vytvořil kamarád David Zakouřil, za což mu tímto děkuji. Ostatní ikony jsou buďto prací Googlu a jsou vývojářům dostupné pod licencí Apache[25], nebo prací komunity pod licencí OFL (Open Font License)[26]. Obě licence mě opravňují k opakovanému použití ikon.

2.9 Obrazovky

V této kapitole se zaměřím na konkrétní obrazovky, které popíši. Obecně každá obrazovka má svůj fragment, presenter a interface, přes které presenter volá metody na fragmetnu. Všechny fragmenty jsou obměňovány v jedné aktivitě, která poskytuje hlavní navigaci aplikace, viz obrázek 11. U každé obrazovky uvedu výslednou podobu a případně příklad nějakého zajímavého kódu.



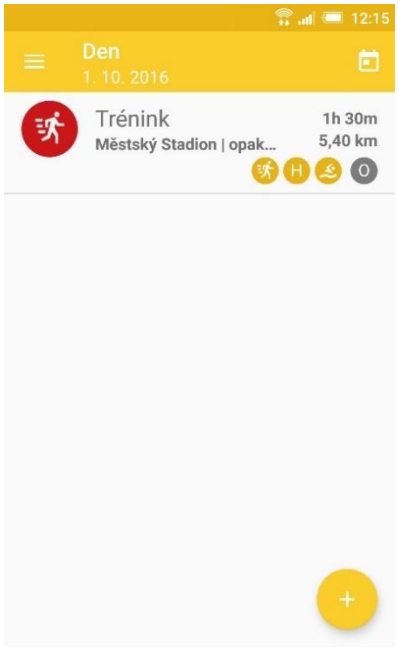
Obrázek 11 – Hlavní navigace aplikace
Zdroj: vlastní

[25]google/material-design-icons · GitHub [online]. 2017 [cit. 2017-03-12]. Dostupné z: https://github.com/google/material-design-icons/blob/master/LICENSE
[26]Templarian/MaterialDesign · GitHub [online]. 2017 [cit. 2017-03-12]. Dostupné z: https://github.com/Templarian/MaterialDesign/blob/master/license.txt

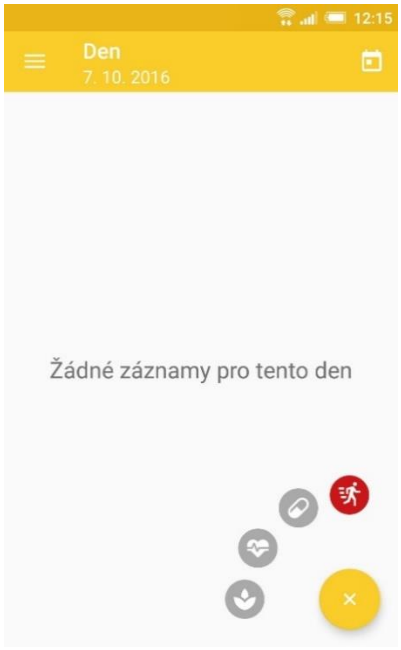
2.9.1 Den

Obrazovka Den splňuje funkční požadavek F2. Zachycují ji obrázky 12 a 13. Výběr dne probíhá pomocí akce v toolbaru[27], po jejímž kliknutí se zobrazí dialog pro výběr dne. Zobrazení denních aktivit je v seznamu, přičemž každá položka obsahuje informaci o typu aktivity, místě, popisu a fázi. Dále ikona vyobrazuje hlavní tréninkový prostředek a všechny ostatní využité tréninkové prostředky. Pokud v daný den není přidaná ještě žádná položka, je zobrazen text informující o této skutečnosti. Pokud chceme danou část upravit, učiníme tak přetažením doprava. Přetažením doleva část smažeme. Pro otevření obrazovky pro přidání tréninku slouží FAB (Floating Action Button)[28]. Po jeho kliknutí se nejprve zobrazí menu se čtyřmi ikonami. Díky němu je jednak možné otevřít obrazovku pro přidávání tréninků, ale také je aplikace tímto způsobem připravená na rozšíření o obrazovky pro přidání tepů, nemocí a regenerací. Neimplementované obrazovky mají příslušnou ikonu zšedivělou. Obrazovka poskytuje průvodce prvním spuštěním, který představuje její základní prvky.

[27] Toolbar (panel nástrojů) je UI komponenta androidu. V toolbaru se zobrazuje titulek či podtitulek obrazovky a takzvané akce. Ty jsou pak značené ikonkou nebo nápisem.
[28] Floating Action Button je UI komponenta androidu. Jedná se o tlačítko umístěné v dolním rohu obrazovky. Floating („vznášející se“) je proto, že je nad veškerým obsahem, a při posouvaní obrazu je stále na stejném místě. Z toho důvodu se používá k vyvolání akce pro přidání obsahu.



Obrázek 12 – Obrazovka Den s přidanou položkou
Zdroj: vlastní



Obrázek 13 – Obrazovka Den s otevřeným menu
Zdroj: vlastní

Pro ukázku jsem se rozhodl použít kód, který reaguje na změnu data uživatelem. Jedná se o příklad využívající RP, přičemž Observable emituje položky typu *Date*. Po emitování (což vyvolá dialog pro změnu data), se nejprve nastaví vnitřní proměnná *date,* uživateli se zobrazí progress[29] a odešle se zpráva po Rx sběrnici (její význam zde však není podstatný). Poté se chain[30] přepne na asynchronní vlákno a začíná interakce s databází. Z ní je obdržen list tréninků a poté je každý doplněn o list tréninkových částí a dominantní element. Poté je chain přepnut na hlavní vlákno a výsledek chainu uložen v cache[31] pomocí funkce *deliverLatestCache*. Tato funkce je součástí Nucleus knihovny. Výsledek je nutné rozdělit na data a View, což je reference na interface daného fragmentu. Data jsou předána metodě *onTrainingLoaded* a v případě jakékoli chyby se zavolá metoda *onErrorHappened*.

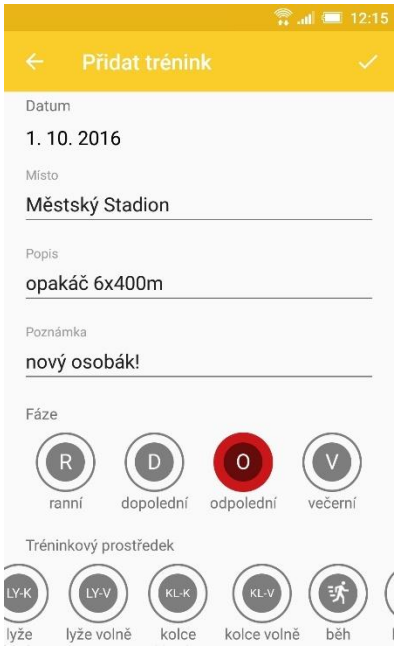
[29] Progress je anglické označení pro View informující o načítání obsahu.
[30] Chain znamená česky řetěz. V případě Rx je to však označení pro sled událostí pomocí operátorů.
[31] Cache je dočasná paměť. V případě Rx se cache využívá k tomu, aby po zaregistrování se na Observable objekt nemusel celý chain probíhat znovu, ale ihned se použije poslední výsledek.

```
view.getDateObservable()
        .doOnNext(date -> {
            this.date = date;
            view.showProgress(true);
            view.showDate(date);
            rxBus.send(new DateEvent(date));
        })
        .observeOn(Schedulers.io())
        .flatMap(date -> trainingDao.getTrainingsByDateObservable(date).take(1))
        .concatMap(trainings -> Observable.from(trainings)
                .concatMap(training ->
trainingPartDao.bindTrainingWithDominantTrainingElement(training).take(1))
                .concatMap(training ->
trainingPartDao.getTrainingPartsByTrainingId(training.getId()).take(1)
                        .doOnNext(training::setTrainingParts)
                        .map(trainingParts -> training))
                .toList())
        .observeOn(AndroidSchedulers.mainThread())
        .compose(deliverLatestCache())
        .subscribe(split(this::onTrainingLoaded, this::onErrorHappened),
Throwable::printStackTrace);
```

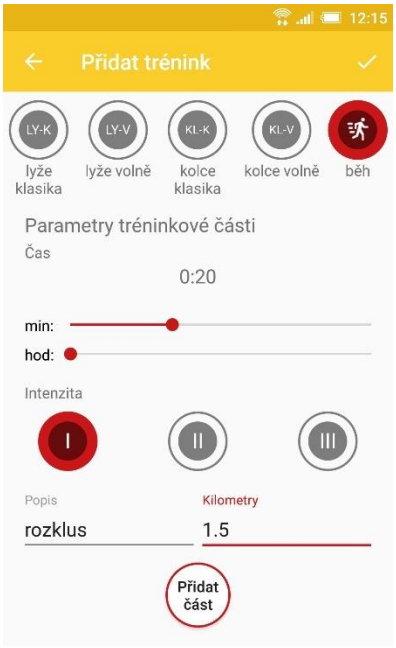
2.9.2 Přidávání tréninku

Obrazovka pro přidávání tréninků je nejdůležitější z celé aplikace. Kromě toho, že musela splňovat požadavky F1, bylo nutné ji navrhnout co nejlépe, jelikož bude využívána nejvíce. Klíčová zde byla rychlost zadávání, přehlednost a „intuitivnost“. Na základě těchto faktorů vznikla obrazovka, jejíž podobu si po částech popíšeme.

V první části se vyplňují formality jako datum tréninku, jeho místo, fáze a případná poznámka k němu, viz obrázek 14. Výběr data probíhá stejně jako na obrazovce Den, tedy přes dialog. Dny, ve kterých jsou již přidané tréninky se všemi fázemi, vybrat nelze. Fáze se vybírá pomocí čtveřice RadioButtonů, přičemž se zobrazují pouze ty volné (v daný den ještě nepřidané). Ostatní položky jsou klasická textová pole. Kromě poznámky jsou všechna povinná, a dokud nejsou vyplněna, je pod příslušným polem zobrazeno červené upozornění.



Obrázek 14 – Obrazovka Přidat trénink – formality
Zdroj: vlastní

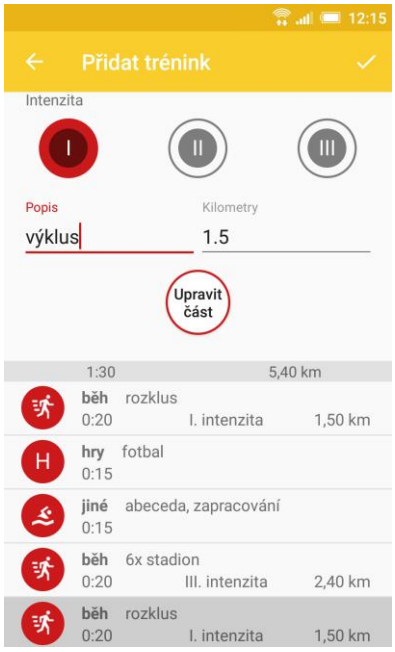


Obrázek 15 – Obrazovka Přidat trénink – přidání tréninkové části
Zdroj: vlastní

Druhá část je vidět na obrázku 15. Je to ta nejvýznamnější část celé aplikace. Přidávají se v ní jednotlivé tréninkové části. Ty jsou na výběr v horizontálně posouvatelném seznamu a po jejím kliknutí se s animací zobrazí formulář pro vyplnění dalších informací. Ten není zobrazený fixně, jelikož různé části vyžadují zaznamenání různých informací[32]. Zaznamenání času aktivity probíhá skrz dva posuvníky (tzv. SeekBary). Prvním se volí minuty, druhým hodiny. Výsledný čas je nad nimi okamžitě zobrazován ve formátu „h:mm“. Intenzita se stejně jako fáze volí pomocí RadioButtonu. Popis a vzdálenost se vyplňují v klasickém textovém poli. Pokud má tréninková část vyplněna všechna povinná pole, je umožněno její přidání. Každá přidaná část je ihned promítnuta v seznamu částí, k čemuž slouží poslední sekce této obrazovky.

[32] Snažil jsem se stoprocentně držet OTD, to znamená, že povinné informace jsou vynucovány, a jiné zase nejsou nabízeny vůbec. Postupně však zjišťuji, že to nebyl úplně správný krok, a myslím si, že do budoucna povolím uživateli zadat, jaké informace chce. V tomhle směru je totiž OTD značně omezený, a detailnější záznamy v aplikaci umožní detailnější analýzy.

Seznam částí je též velmi důležitý, protože uživatel může snadno kontrolovat, kolik a co už zadal. Podívejme se na něj na obrázku 16.



Obrázek 16 – Obrazovka Přidat trénink – seznam částí
Zdroj: vlastní

Záhlaví seznamu tvoří součty časů a vzdáleností dosud přidaných části. Každá část je pak zobrazena se všemi přidruženými informacemi. Po kliknutí na danou část je umožněna její editace. Po dvojitém kliknutí se část duplikuje, což je vhodné pro přidávání podobných částí, jako je například klus a výklus, kde se většinou liší pouze popisek. Také u opakovaných tréninků je to velmi užitečná funkce. Pokud se část přidrží déle, lze ji libovolně přesouvat. Není tak nutné části mazat, což se provede přetažením na stranu, a znovu přidávat.

Pokud jsme se zadaným tréninkem spokojeni, kliknutím na akci v toolbaru se uloží. Rád bych ještě zmínil, že pokud uživatel klepne na tlačítko zpět (ať už omylem nebo úmyslně), zobrazí se mu dialog s potvrzením. Považuji to za více než užitečně.

Uživatel je při prvním spuštění s celou obrazovkou včetně všech jejích funkcí seznámen interaktivním průvodcem. S jeho pomocí je postupně přidaný ukázkový trénink. I toto považuji za velmi užitečné, jelikož by uživatel na některé funkce v seznamu částí nemusel nikdy přijít. Z toho důvodu průvodce nejde přeskočit, a musí se dokončit celý. Ukázka průvodce je na obrázcích 17 a 18.

Obrázek 17 – Obrazovka Přidat trénink – ukázka průvodce I.
Zdroj: vlastní

Obrázek 18 – Obrazovka Přidat trénink – ukázka průvodce II.
Zdroj: vlastní

Není divu, že i presenter této obrazovky je nejsložitější z celé aplikace. Pro ukázku jsem si zvolil opět Rx chain, který elegantně validuje celou obrazovku.

```
Observable.combineLatest(
        view.getPlaceObservable(),
        view.getDescriptionObservable(),
        view.getNoteObservable(),
        view.getPhaseObservable(),
        (place, description, note, phase) -> {

            boolean valid = true;

            if (TextUtils.isEmpty(place)) {
                valid = false;
                view.showPlaceError(true);
            } else {
                view.showPlaceError(false);
            }


            if (TextUtils.isEmpty(description)) {
                valid = false;
                view.showDescriptionError(true);
            } else {
                view.showDescriptionError(false);
            }
```

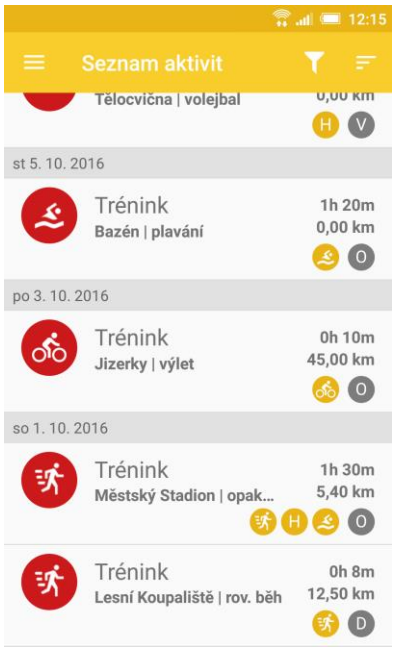
```
            if (phase == null) {
                valid = false;
            }

            if (partList.size() == 0) { // partList je list tréninkových částí
                valid = false;
            }

            return valid;
        }
).subscribe(view::setAddTrainingButtonEnabled, Throwable::printStackTrace);
```

Je zde ukázka operátoru *combineLatest*, který tuto úlohu zvládá na jedničku. Parametry tohoto operátoru jsou observables emitující aktuální hodnoty místa, popisu, poznámky a fáze. Operátor dělá to, že do tzv. kombinační funkce předává vždy poslední hodnoty, a to vždy, když se nějaká změní. Kombinační funkce slouží ke zpracování těchto hodnot, v našem případě k validaci. Ta je pak již jednoduchá. Na začátku je nastavena proměnná typu boolean[33] s počáteční hodnotou true. Poté už jen kontrolujeme, zda daná hodnota splňuje naše požadavky. Pokud ne, je tato proměnná nastavena na false a je zobrazena chyba. Pokud je hodnota validní, chybu skryjeme. Každá kombinační funkce musí vracet nějakou hodnotu, se kterou pak chain dále pracuje. Zde je vrácen výsledek, zda je formulář validní či nikoliv. Tento chain již nijak nepokračuje, je již „subsrcibenut“ (naslouchán na změny, zaregistrován...). To zajistí, že po každé změně hodnot se provede validace a podle výsledku validace se povolí či zakáže tlačítko pro přidání tréninku.

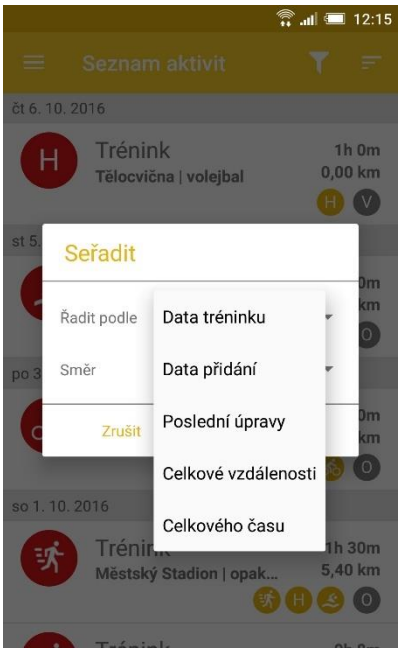
2.9.3 Seznam aktivit

Obrazovka Seznam aktivit splňuje požadavky F3 a ještě tyto požadavky doplňuje. Aktivity jsou zobrazeny pomocí stejných layoutů jako v případě obrazovky Den. Znázorňuje to obrázek 19.

[33] Dvoustavová proměnná, která může nabývat hodnot true a false (pravda a nepravda).



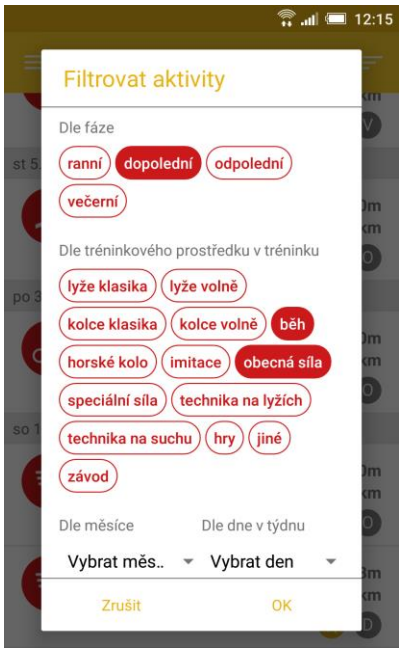
Obrázek 19 – Obrazovka Seznam aktivit
Zdroj: vlastní



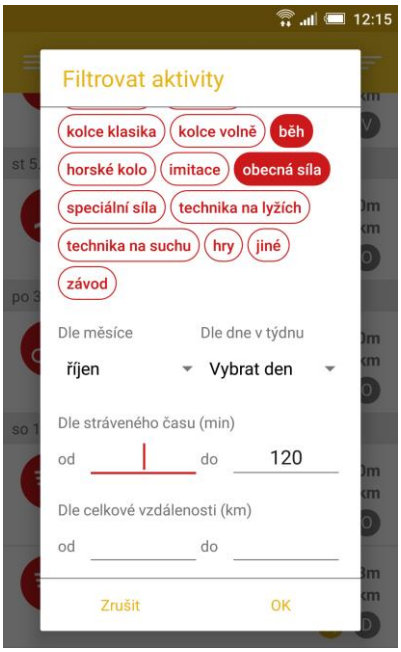
Obrázek 20 – Obrazovka Seznam aktivit – dialog umožňující řazení
Zdroj: vlastní

Dle požadavků jsou aktivity sdruženy po dnech se záhlavím zobrazující datum a den v týdnu. Pro řazení aktivit slouží dialog, který se vyvolá akcí v toolbaru, viz obrázek 20. Lze v něm nastavit směr řazení (vzestupně a sestupně), a také vybrat z pěti možností, podle kterých bude řazení provedeno.

Požadavek filtrace je též řešen pomocí dialogu, viz obrázky 21 a 22. Kromě třídění podle fáze a tréninkových prostředků je možné třídit podle měsíce, konkrétního dne v týdnu a podle rozsahu celkového času a vzdálenosti aktivity Filtry jsou mezi sebou kombinovány logickou funkcí AND, stejně tak tréninkové prostředky mezi sebou. Jednotlivé fáze jsou z databáze vybírány logickou funkcí OR.



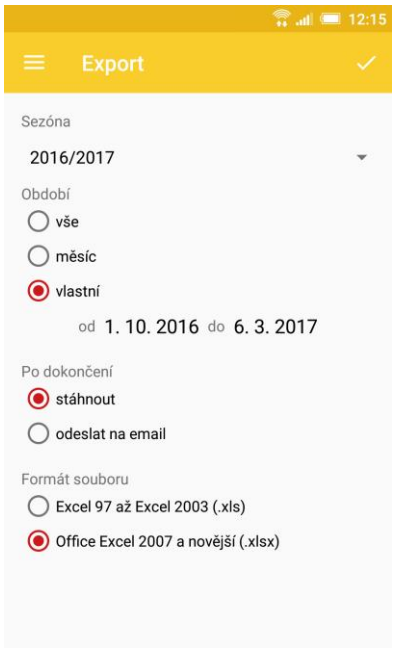
Obrázek 21 – Seznam aktivit – dialog pro filtrování I
Zdroj: vlastní



Obrázek 22 – Seznam aktivit – dialog pro filtrování II
Zdroj: vlastní

2.9.4 Export

Obrazovka Export odpovídá požadavkům F4 a zároveň ještě přidává funkcionalitu pro bližší výběr rozsahu exportu. Výslednou podobu můžeme vidět na obrázku 23.



Obrázek 23 – Obrazovka Export
Zdroj: vlastní

Výběr rozsahu exportovaných dat začíná volbou sezóny. Ty jsou automaticky rozpoznané z přidaných tréninků. Dále můžeme volit období v dané sezóně. Můžeme zvolit všechna data, nebo určitý měsíc či dokonce konkrétní termín. U volby měsíce se zobrazují jen ty, ve kterých je alespoň jeden přidaný trénink. Též výběr vlastního období je vymezen rozsahem, v němž se nacházejí tréninky.

Dále můžeme zvolit, co se má stát po dokončení exportu. Na výběr je stažení souboru do zařízení a odeslání na email. To je vhodné pro přímé odeslání deníku trenérovi. Pro stažení je vyžadováno oprávnění pro zápis do externího úložiště, o které je v případě API > 23 požádáno[34]. Odeslání emailu je podmíněno nastavením odesilatelova emailu v nastavení.

Poslední volbou, kterou obrazovka disponuje, je výběr formátu souboru. Podporované jsou v tuto chvíli formáty Excel 97 až Excel 2003 (tj. přípona .xls) a formát Excel 2007 a novější (.xlsx). V případě, že je zvoleno stažení souboru, uživatel je při výběru formátu upozorněn, není-li daný formát v jeho zařízení otevíratelný. Samozřejmostí je validace celého formuláře před spuštěním exportu, který provedeme akcí v toolbaru.

2.9.4.1 Zvážení možností implementace logiky exportování

V zásadě byly dvě možnosti. Provádět vše lokálně na zařízení, nebo využít vzdálené služby. Zvažoval jsem obě možnosti, a nakonec zvolil druhou. Rozhodl jsem se tak na základě několika skutečností. Knihovny pro práci s Excelem v Javě sice existují, nicméně buď podporují pouze formát XLS či pouze formát XLSX, nebo jsou značně naddimenzované pro použití na mobilních zařízeních (tj. velikost > 20 MB). Volbě webové aplikace nahrávají také možnosti rozšíření aplikace.

V návaznosti na zvolené provedení exportu se implementace rozdělila na dvě části. V aplikaci se data webové službě pouze připraví, načež je pak proveden samotný zápis do dokumentu OTD[35].

[34] Google od API 23 zavedl tzv. Runtime Permissions. To znamená, při instalaci aplikace se neudělují žádná oprávnění, ale udělují (a zpětně i odebírají) za běhu aplikace. Ta pak o oprávnění žádá až tehdy, kdy jej vyžaduje. Toto opatření si klade za cíl zvýšit bezpečnost aplikací a obezřetnost uživatelů.
[35] Z dokumentu byly odstraněny některé listy za účelem zrychlení exportu (až o 50 %). Jsou to listy, které se využívaly minimálně nebo jsou pro exportovaná data zcela nezávislé (například listy návodů k vyplňování atd.) Dále byl doplněn o automatické dopočítávání dnů a jejich názvů v závislosti na prvním vyplněném dnu sezóny.

2.9.4.2 Implementace v aplikaci

Jak bylo řečeno, úkol aplikace je pouze data připravit. Rozhodl jsem se data transformovat do podoby, jaká náleží OTD. Vytvořil jsem si modely pro denní počet JZ, celkový denní čas a vzdálenost tréninkového prostředku a model pro psanou část. Využil jsem vlastností SQLBriteDao, a vytvořil si třídu ExportDao. Byť jsou Dao třídy zamýšlené tak, aby měly každá vlastní tabulku. V tomto případě však bylo porušení vhodné. Díky tomu jsem mohl modely nechat automaticky namapovat z SQL dotazů. Níže je uveden příklad metody, která obstarává data z databáze a mapuje je do listu.

```
public Observable<List<DailyTrainingCount>> getJZs(DateRange range) {
    return query(SELECT("COUNT(" + Training.COL_DATE_TRAINING + ") AS " +
DailyTrainingCount.COL_JZ, Training.COL_DATE_TRAINING + " AS " +
DailyElementTotals.COL_DATE)
            .FROM(Training.TABLE_NAME)
            .INNER_JOIN("(" +
                    SELECT(TrainingPart.COL_TRAINING_ID, sumHoursAndMinutes())
                            .FROM(TrainingPart.TABLE_NAME)
                            .GROUP_BY(TrainingPart.COL_TRAINING_ID)
                            .HAVING(COL_TOTAL_TIME + " >= 30")
                            .asCompileableStatement().sql +
                    ") ")
            .ON(TrainingPart.COL_TRAINING_ID + " = " + Training.TABLE_NAME + "." +
BaseColumns._ID)
            .GROUP_BY(Training.COL_DATE_TRAINING)
            .HAVING(generateWhereRangeClause(range)))
            .run()
            .mapToList(DailyTrainingCountMapper.MAPPER);
}
```

Po obstarání všech exportovaných dat je odeslán http požadavek, jehož tělo tvoří JSON s exportovanými daty.

2.9.4.3 Implementace webové aplikace

Webová aplikace je postavena na českém PHP frameworku Nette[36], se kterým mám již zkušenosti a práce tak byla rychlejší. Nette využívá stejný návrhový vzor, jako jsem použil

[36] Rychlý a pohodlný vývoj webových aplikací v PHP | Nette Framework [online]. 2017 [cit. 2017-03-12]. Dostupné z: https://nette.org/cs/

v mobilní aplikaci, tedy MVP. Podrobněji se o Nette však zmiňovat nebudu, jelikož to sahá přes rámec této práce. I při vývoji této aplikace byl využit Git.

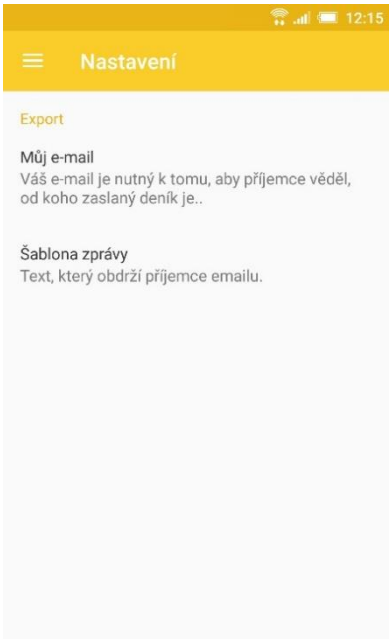
V Nette byl vytvořen Api modul výhradně pro komunikaci s mobilní aplikací. Ten bylo kvůli absenci uživatelských účtů potřeba zabezpečit. Předpokladem je použití zabezpečeného protokolu HTTPS, ale to nestačí. Zabezpečení proti zneužití API je řešeno pomocí jednorázových hesel (OTP – One Time Passwords) generovaných podle normy RFC 6238[37] a API klíčů. Princip je následující (zjednodušeně): V aplikaci je vygenerován OTP token, ke kterému se připojí vygenerované UUID (Universally Unique Identifier)[38] charakterizující nainstalovanou instanci aplikace, a je odeslán http požadavek žádající o registraci zařízení. Webová služba pak porovnává, zda přijaté OTP je validní v určitém časovém okně a nebylo již v tomto okně použito. Pakliže ověření proběhne v pořádku, je vygenerován 128 bitový klíč, na jehož základě se s každým http požadavkem na API aplikace autentizuje. Bez něj jsou všechny požadavky odmítnuty.

Pro práci s Excel dokumenty jsem využil knihovnu PHPExcel[39]. Pracuje se s ní velmi pohodlně. Na zápis dat do dokumentu OTD vnikla pomocná třída *DenikHelper*. Z presenteru jsou jí v konstruktoru předána data, která jsou poté zapsána. Pro ukládání souboru vznikla další pomocná třída *FileHelper*. Ta se kromě ukládání stará o přidružení souboru danému zařízení („uživateli“).

2.9.5 Nastavení

Tato obrazovka je oproti funkčním požadavkům přidaná navíc. Její přítomností se pouze zlepšuje UX (User Experience). Disponuje dvěma položkami. První slouží pro nastavení uživatelova emailu, druhá zase pro nastavení šablony pro tělo odesílaného emailu. Uživatel tedy při každém exportu nemusí tyto údaje vyplňovat znovu. Rád bych zde zmínil, že se nejedná o standardní řešení nabízené Android API, nýbrž mou vlastní implementaci. Díky tomu mám větší kontrolu nad vzhledem i daty, které uživatel zadává. Snímek obrazovky můžeme vidět na obrázku 24.

[37] TOTP: Time-Based One-Time Password Algorithm. IETF Tools [online]. David M'Raihi, Salah Machani, Mingliang Pei, Johan Rydell, 2011 [cit. 2017-03-11]. Dostupné z: https://tools.ietf.org/html/rfc6238
[38] UUID je univerzální identifikátor složený ze 128 bitového čísla.
[39] GitHub - PHPOffice/PHPExcel [online]. 2017 [cit. 2017-03-12]. Dostupné z: https://github.com/PHPOffice/PHPExcel



Obrázek 24 – Obrazovka Nastavení
Zdroj: vlastní

2.9.6 O aplikaci

Stejně jako obrazovka Nastavení i tato je zde navíc – viz obrázek 25. Poskytuje uživateli základní informace o aplikace, jejím autorovi a vybízí ho ke zpětné vazbě prostřednictvím tlačítka. Po kliknutí na toto tlačítko se otevře emailový klient s předvyplněnými informacemi, jako jsou verze aplikace, Android API, model zařízení a podobně.

Obrázek 25 – Obrazovka O aplikaci
Zdroj: vlastní

3 Publikace

Po dokončení aplikace je třeba ji nějakým způsobem distribuovat uživatelům. Tento proces lze vývojáři označit za velmi důležitý, jelikož se jejich práce odhalí očím veřejnosti. Proto je i této fázi potřeba věnovat náležitou pozornost.

3.1 Příprava na publikaci

Před publikováním aplikace je nejprve potřeba ji na to připravit. Pokud bychom totiž aplikaci publikovali ve tvaru neuzpůsobeném produkci, byla by velmi zranitelná a bez ochrany před reverzním inženýrstvím[40]. Žádný vývojář nechce, aby mu kdokoli, téměř bez námahy, zjistil obchodní tajemství či prolomil zabezpečení. Proto existuje sada nástrojů, které tato rizika minimalizují nebo přinejmenším potencionálním útočníkům velmi ztěžují práci. Společně s výše popsanými úpravami dochází také ke zmenšování velikosti výsledného souboru aplikace, což má vliv i na její rychlost.

3.1.1 ProGurard

ProGuard je silný nástroj na řešení všech výše popsaných problémů. Pracuje ve třech krocích. Prvním krokem je odstranění všech nepoužitých tříd, metod, polí a jiného nepotřebného kódu. V případě release buildu (viz Build types) jde například o metody pro logování nebo zajišťující speciální funkcionality pro debuggování. Druhým krokem je tzv. obfuskace. Obecně je to taková úprava zdrojových kódů, po které nebudou člověkem snadno čitelné a rozluštitelné. Provádí se nahrazováním jmen proměnných, konstant, metod a tříd nicneříkajícími znaky. To může při využití reflexe vést k neočekávanému chování programu, proto se vytváří speciální soubor s pravidly pro vynechání obfuskace. Posledním krokem je optimalizace. Tj. např. zjednodušovaní algebraických operací (partial evaluation), optimalizace cyklů a další techniky.

Daní za použití ProGuardu je čas sestavování, který se může u velkých projektů prodloužit až na několik minut. Proto se používá zejména u release buildů, kde dlouhý čas není omezujícím faktorem.

[40] Reverzní inženýrství je proces, při kterém se zkoumá produkt za účelem zjištění jeho funkčních principů.

3.1.2 Build types

Díky nástroji Gradle, který slouží k sestavování aplikace, můžeme definovat různé typy buildů (sestavení). Nejčastěji se konfigurují tzv. release a debug buildy. Z kódu aplikace máme vždy přístup k informaci o tom, o jaký typ se jedná, a lze tak ovlivnit chování aplikace. Můžeme například povolit logování nebo přidat nějaké pomocné vývojářské funkce.

U jednotlivých typů můžeme nastavit několik parametrů. Jedním z těchto parametrů je povolení ProGuardu zmíněného v minulé podkapitole. To je asi nejčastější případ. Dalšími parametry můžeme například změnit jméno balíčku, jméno verze nebo přiřadit konfiguraci podepisování.

3.1.3 Podepsání aplikace

Každý sestavený soubor musí být podepsaný certifikátem. Debug buildy jsou ve výchozím nastavení AS podepisovány automaticky vygenerovaným certifikátem. Ten má od vygenerování platnost jeden rok a je poté využívaný pro všechny debug buildy téže aplikace. Můžeme si ale nastavit vlastní, což je podmínkou u release buildů. Vlastní certifikát pro debug se hodí v případě, že vyvíjíme střídavě z více stanic najednou. Protože telefon neumožní update aplikace s jiným certifikátem (taková byla alespoň má zkušenost při vývoji rok zpátky, od té doby používám jen vlastní debug certifikáty).

K čemu vlastně toto podepisování slouží? Podpisem aplikace certifikátem se ztotožní její autor. Toho se využívá pro ověření integrity aplikace při její instalaci. Android také umožňuje sdílení určitého prostoru napříč aplikacemi, které jsou podepsány stejným certifikátem. Lze ho vygenerovat přímo v AS nebo využitím utility Keytool.

3.2 Možnosti distribuce

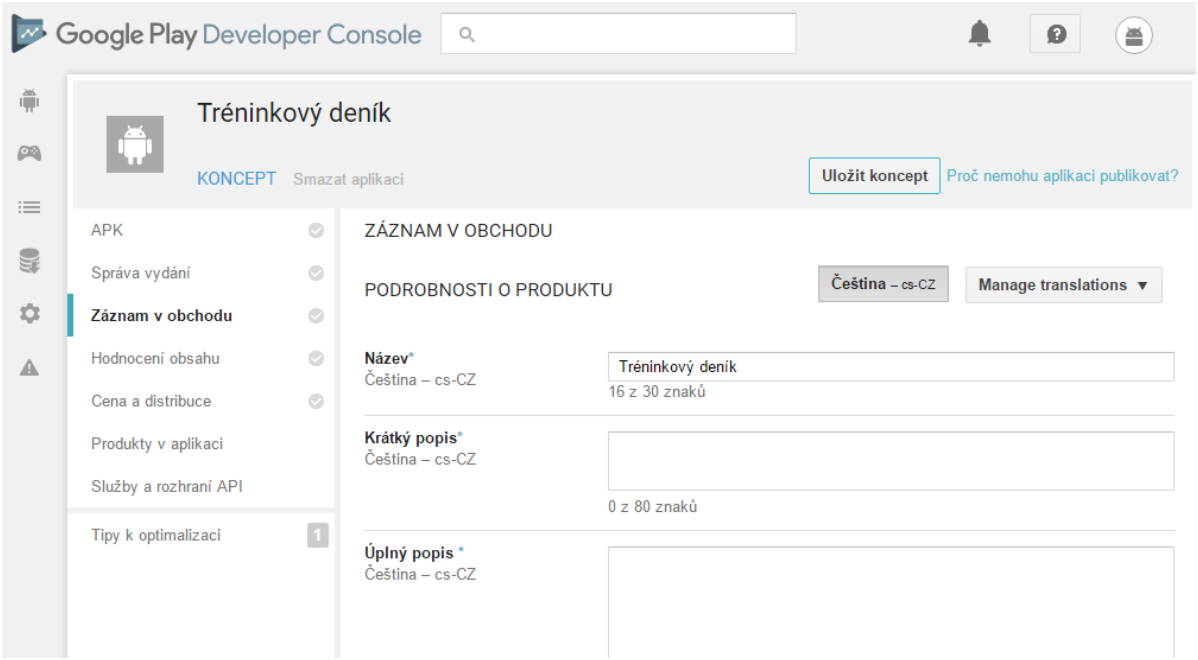
Publikovat aplikace lze několika způsoby. Například pomocí emailu nebo umístěním instalačního souboru na webové stránky. Nejlepším způsobem je ovšem využít takzvaných online distribučních služeb. Jedna z takových služeb je Google Play, která je zároveň jedinou oficiální distribuční službou na Android. Díky tomu je zde určitá záruka kvality služby a mnoho možností pro vývojáře.

3.3 Zveřejnění na Google Play

Jak už bylo řečeno, Google Play je jediná oficiální distribuční služba. Pojďme si ukázat, co bylo třeba podniknout pro úspěšné zveřejnění aplikace.

3.3.1 První kroky

Vůbec první věc, která je třeba udělat, je stát se vývojářem. Toho dosáhneme tak, že se na stránce play.google.com/apps/publish/signup/ přihlásíme pod účtem, pod kterým chceme publikovat aplikace, a zaplatíme $25. Následně jsme vpuštěni do vydavatelského prostředí, kde můžeme vytvořit naší první aplikaci.



Obrázek 26 – Vyplnění informací o aplikaci v obchodu Google Play
Zdroj: vlastní

Po vytvoření aplikace můžeme začít vyplňovat informace o aplikaci. Pro představu poslouží obrázek 26. Počet nastavení, které se musí, či mohou vyplnit, je opravdu spousta. Od snímků obrazovky přes hodnocení aplikace z hlediska přístupnosti (nevhodné pro děti…) až k nastavení dostupnosti v jednotlivých zemích. V další kapitole bych se rád zmínil o různých distribučních kanálech, ve kterých lze nezávisle publikovat.

3.3.2 Distribuční kanály

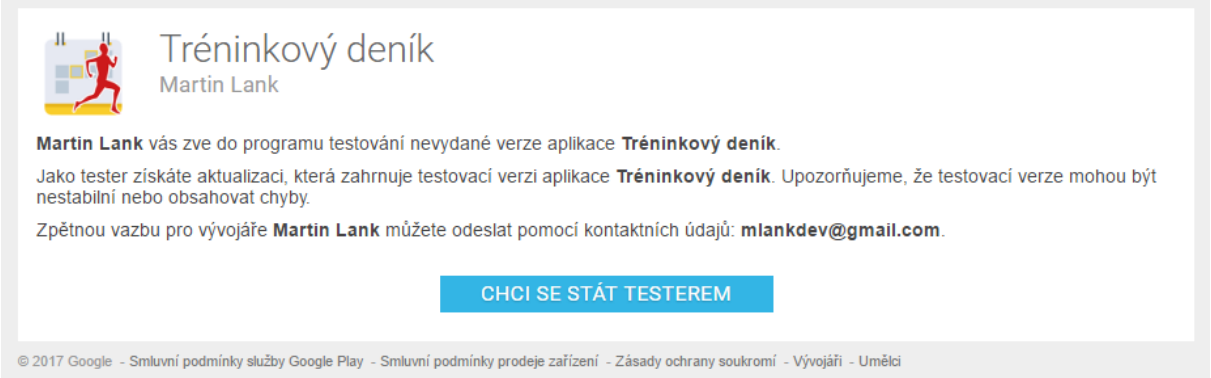
Google Play umožňuje distribuovat více verzí aplikace současně za pomocí takzvaných kanálů. Tyto kanály slouží pro stejnojmenné verze aplikací, tj. alfa, beta a produkční.

Kanály alfa a beta slouží pro testování daných verzí aplikace a oba umožňují jak otevřené, tak uzavřené testování. Všechny tři kanály mohou být aktivní současně.

3.3.3 Dokončení publikace

Pokud jsme ve stavu, kdy máme všechna povinná pole vyplněná, můžeme dokončit publikaci. Pro publikaci mé práce jsem si prozatím zvolil uzavřený beta kanál, protože chci aplikaci nejdříve otestovat na menším vzorku lidí, které znám. V nastavení jsem si pro ně vytvořil seznamy a můžu tak jednoduše volit testovací okruh lidí. Poté, co máme vybraný kanál a zvolen okruh lidí, můžeme aplikaci publikovat. Tento proces trvá několik hodin. Google během ní aplikace analyzuje, zda neobsahují škodlivé kódy nebo jinak neporušují smluvní podmínky.

Pokud výsledek analýzy naší aplikace vyšel kladný, je vygenerován odkaz pro testery. Ten je třeba odeslat lidem, které jsme zadali do seznamu testerů v nastavení, jelikož musí potvrdit roli testera. Až poté jim je aplikace zpřístupněna ke stažení. Situaci nastiňuje obrázek 27.



Obrázek 27 – Potvrzení účasti v testovacím programu
Zdroj: vlastní

Závěr

Cílem práce bylo vytvořit aplikaci, která běžcům na lyžích ulehčí a zpříjemní zadávání tréninkového deníku. Jedním z požadavků bylo, aby se data dala exportovat do oficiálního tréninkového deníku od SLČR, který je ve formě Excel dokumentu. Na základě rozsáhlé analýzy tohoto dokumentu jsem vymyslel řešení, které jsem použil v aplikaci. Určitou výhodou také bylo, že jsem deník neviděl poprvé, ale sám jsem si ho jako závodník několik let vedl.

Aplikace z pohledu technického řešení splnila svůj cíl. Je postavena na MVP architektuře, využívá moderních technologií (jako jsou například reaktivní rozšíření) a po celou dobu vývoje byl využíván verzovací systém Git. Z mého pohledu je kód napsán velmi čistě a čitelně. Nejen díky tomu je aplikace stabilní a s předvídatelným chováním.

Pokud se na aplikaci podíváme z pohledu uživatele, nebo chcete-li závodníka, i v tomto směru aplikace splnila své funkční požadavky. Aplikace disponuje pěti obrazovkami. Po startu aplikace se zobrazí denní přehled aktivit. Kromě výběru dne lze také z menu otevřít obrazovku pro přidávání (zároveň i editaci) tréninků. Díky jejímu rozvržení je možné trénink evidovat tak podrobně, jak jen závodník chce. Pro hromadné zobrazení zadaných tréninků slouží obrazovka Seznam aktivit. Položky v něm jsou členěny do dnů a lze je filtrovat a řadit dle mnoha kritérií. Byť statistické údaje zatím chybí, výsledkem je poměrně silný nástroj, který později umožní podrobný rozbor. Pro export dat slouží stejnojmenná obrazovka, která umožňuje volbu rozsahu dat, formátu souboru a volbu mezi odesláním na email a stažením. Samotný export je řešen pomocí webové aplikace postavené na PHP frameworku Nette. Zabezpečení API je řešeno pomocí jednorázových hesel a unikátních identifikátorů instancí aplikace. Poslední obrazovkou je obrazovka O aplikaci. Je velmi jednoduchá, ale důležitá. Kromě informací lze skrze ni získat zpětnou vazbu od uživatelů.

Ačkoli práce splnila stanovené cíle, lze ji dále v mnohém rozvíjet. Webová služba by se společně s aplikací daly rozšířit o uživatelské účty. Díky nim by data mohla být zálohována a synchronizována napříč zařízeními. Nástavbou na to by mohla být registrace sportovních (běžeckých) oddílů, což by dalo trenérům možnost analyzovat tréninkový deník svěřence online, nezávisle na něm. Též aplikace by mohla být vylepšena. Je

připravena na obrazovky pro přidávání tepů, nemocí a regenerací. Implementovat by se dala také funkce, která by umožnila sdílení tréninku v rámci tréninkové skupiny, jelikož v ní závodníci obvykle trénují společně, a tedy absolvují stejné tréninky. Stejně tak by bylo zajímavé porovnávat a analyzovat deníky napříč oddíly. V neposlední řadě by se práce díky její škálovatelnosti dala rozšířit i mezi další sporty, které mají zase svá specifika.

Na hodnocení práce z hlediska užitečnosti, a zda skutečně pomohla zvýšit relevantnost tréninkových deníků, je příliš brzy. Já však pevně věřím, že tomu tak bude a jsem přesvědčen, že díky výše popsaným rozšířením by mohla časem zcela nahradit tréninkový deník v Excelu a posunout tak celkový systém evidence na vyšší úroveň.

Práce na tomto díle mě velmi bavila a jednoznačně jsem se díky ní v tomto oboru zdokonalil. Časová náročnost byla poměrně velká. Nebojím se mluvit o průměrně třech až čtyřech hodinách denně a jsem tak velmi rád, že se mi podařilo optimálně odhadnout rozsah práce a dílo dokončit.

V současné době je aplikace publikována na Google Play pouze v rámci uzavřeného beta testování. Přejete-li se do něj zapojit a aplikaci si tak vyzkoušet, zašlete na adresu denik@mlank.cz Google email (Gmail), pro který chcete aplikaci zpřístupnit.

Seznam obrázků

Použitá literatura

VÁVRŮ, Jiří a Miroslav UJBÁNYAI. *Programujeme pro Android*. 2., rozš. vyd. Praha: Grada, 2013. Průvodce (Grada). ISBN 978-80-247-4863-4.

NURKIEWICZ, Tomasz a Ben CHRISTENSE. *Reactive Programming with RxJava: Creating Asynchronous, Event-Based Applications*. Sebastopol: O'Reilly Media, 2016. ISBN 978-1-4919-3159-2.

Android Developers [online]. Mountain View: Google, 2017 [cit. 2017-03-11]. Dostupné z: https://developer.android.com/

Architecting Android..The clean way? *Fernando Cejas* [online]. Berlin: Fernando Cejas, 2014 [cit. 2017-03-11]. Dostupné z: https://fernandocejas.com/2014/09/03/architecting-android-the-clean-way/

A. Adresářová struktura projektu

```
+---assets[20]
+---java
|   \---cz
|       \---mlank
|           \---denik – kořenový adresář zdrojových kódů; konec pevné struktury
|               +---db – adresář pro DAO třídy
|               |   \---base – adresáře base jsou obecně určený pro abstraktní
                          třídy, ze kterých další objekty z nadřazeného adresáře
|               +---di – adresář pro interface a třídy definující DI
|               +---domain – veškeré modelové třídy
|               |   +---event – modely definující RxBus události
|               |   +---model – obecné modely
|               |   +---request – modely definující tělo HTTP požadavku
|               |   +---response – modely definující tělo HTTP odpovědi
|               |   \---rest – složka pro interface definující HTTP URL
|               +---interactor – třídy zajišťující obstarávání dat
|               +---mvp – kořenová složka pro implementaci MVP Nucleusu
|               |   +---presenter – složka pro presentery
|               |   |   \---base
|               |   \---view – složka s rozhraními, které příslušný presenter může
                          volat na přidružené aktivitě či fragmentu
|               |       \---base
|               +---ui – složka pro veškeré Views
|               |   +---activity – pro aktivity
|               |   |   \---base
|               |   +---adapter – pro adaptéry
|               |   +---dialog – dialogy
|               |   |   \---base
|               |   +---fragment – pro fragmenty
|               |   |   \---base
|               |   \---view – ostatní vlastní Views
|               \---utils – ostatní třídy s užitečnými metodami
|                   \---OTPUtils
\---res[19]
    +---color – definice barev
    +---drawable – nezařazené bitmapy
    +---drawable-hdpi – bitmapy pro displeje s vysokou hustotou pixelů
    +---drawable-mdpi – bitmapy pro displeje se střední hustotou pixelů
    +---drawable-v21 – bitmapy pro API verze 21 a výš
    +---drawable-xhdpi – bitmapy pro displeje s velmi vysokou hustotou pixelů
    +---drawable-xxhdpi
    +---drawable-xxxhdpi
    +---layout – definice layoutů
    +---layout-land – definice layoutů širokoúhlý režim (landscape)
```

```
+---layout-v21 – layouty pro API verze 21 a výš
+---menu – definice menu
+---mipmap-hdpi – složka pro ikonu aplikace
+---mipmap-mdpi
+---mipmap-xhdpi
+---mipmap-xxhdpi
+---mipmap-xxxhdpi
+---values – složka pro definice textů, polí, stylů a dalších
\---values-land
```
