FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Název:	Mobilní aplikace pro amatérskou tenisovou ligu
Student:	Jan Steuer
Vedoucí:	Ing. Tomáš Nováček
Studijní program:	Informatika
Studijní obor:	Webové a softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	Do konce letního semestru 2020/21

Pokyny pro vypracování

Cílem je vytvořit mobilní aplikaci pro operační systém Android, která zjednoduší a zpřehlední organizaci amatérské tenisové ligy AMTEL.

Proveďte následující kroky:
1) Analyzujte aktuální stav ligy a současný postup při její organizaci.
2) Na základě komunikace s účastníky a organizátory zdokumentujte a analyzujte požadavky na aplikaci.
3) Implementujte aplikaci v jazyce Kotlin pro operační systém Android.
4) Implementace bude podporovat synchronizaci mezi jednotlivými instancemi přes vzdálenou databázi.
5) Proveďte uživatelské a integrační testy aplikace.

Seznam odborné literatury

Dodá vedoucí práce.

Ing. Michal Valenta, Ph.D.
vedoucí katedry

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
děkan

V Praze dne 31. října 2019

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Bakalářská práce

Mobilní aplikace pro amatérskou tenisovou ligu

Jan Steuer

Katedra softwarového inženýrství
Vedoucí práce: Ing. Tomáš Nováček

12. května 2020

Poděkování

Velice rád bych poděkoval vedoucímu mé bakalářské práce a zadnímu odborníkovi na Apakrychle Ing. Tomášovi Nováčkovi za velmi užitečné rady, cennou zpětnou vazbu a aktivní přístup během tvorby mé bakalářské práce. Rovněž bych rád poděkoval vedoucímu soutěže *AMTEL Opava* Mgr. Jiřímu Vaňkovi za umožnění vývoje aplikace. Také děkuji všem účastníkům testování aplikace.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval(a) samostatně a že jsem uvedl(a) veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 46 odst. 6 tohoto zákona tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou, a veškeré jejich dokumentace (dále souhrnně jen „Dílo"), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla, a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu), licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 12. května 2020 . . . . . .

Odkaz na tuto práci

Steuer, Jan. *Mobilní aplikace pro amatérskou tenisovou ligu.* Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2020.

Abstrakt

Tato bakalářská práce se zabývá vývojem mobilní aplikace pro ulehčení organizace tenisové ligy. Čtenáře postupně provede jednotlivými fázemi tvorby softwaru – analýzou, návrhem, implementací a testováním. Zpočátku vysvětlí problémovou doménu a navrhe řešení aplikace. Poté zasvětí čtenáře do úvodu vývoje Android aplikací a nabídne zajímavé body implementace. V samotném konci pak zejména popíše průběh uživatelského testování. Výstupem této práce je funkční a uživatelsky přívětivá Android aplikace, která nahrazuje dosavadní způsob organizace ligy *AMTEL Opava*.

Klíčová slova mobilní aplikace, tenisová liga, organizace tenisové ligy, AMTEL Opava, Android, Kotlin

Abstract

This bachelor thesis deals with the development of a mobile application to facilitate the organization of a tennis league. The reader is gradually guided through the individual phases of software development – analysis, design, implementation and testing. Initially, the thesis puts an interpretation on the problem domain and proposes an application solution. It then familiarises the reader with the introduction to Android application development and offers interesting implementation points. At the very end, it mainly describes the progression of user testing. The output of this thesis is a functioning and user-friendly Android application, which replaces the current way of organizing the *AMTEL Opava* league.

Keywords mobile app, tennis league, organization of a tennis league, AMTEL Opava, Android, Kotlin

Obsah

Úvod **1**
Cíl práce . . . 1
Členění práce . . . 2

1 Analýza **3**
1.1 Fungování ligy . . . 3
1.1.1 Úvod . . . 3
1.1.2 Struktura soutěže . . . 3
1.1.3 Stručná pravidla . . . 4
1.2 Postup při organizaci . . . 4
1.2.1 Začátek sezóny . . . 4
1.2.2 Průběh sezóny . . . 5
1.2.3 Závěr sezóny . . . 6
1.3 Zhodnocení stávajících alternativ . . . 6
1.3.1 Metodika vyhledávání existujících aplikací . . . 6
1.3.2 Výběr hodnocených aplikací . . . 7
1.3.3 Předpokládané rysy aplikací pro organizaci ligy . . . 8
1.3.4 Hodnocení stávajících aplikací . . . 8
1.3.4.1 HuráLiga . . . 9
1.3.4.2 VAŠE LIGA . . . 11
1.3.4.3 TennisCall . . . 15
1.3.4.4 Zbylé aplikace . . . 18
1.3.4.5 Závěr . . . 19
1.4 Cíloví uživatelé . . . 20
1.4.1 Vedoucí soutěže . . . 20
1.4.2 Vedoucí týmu . . . 20
1.4.3 Host . . . 20
1.4.4 Use case diagram . . . 21
1.5 Funkční požadavky . . . 22

1.5.1 Seznam požadavků 22

2 Návrh **29**
2.1 Výběr platformy 29
2.1.1 Minimální verze 30
2.2 Implementační jazyk 30
2.3 Návrh uživatelského rozhraní 31
2.3.1 Barevná paleta a font 32
2.3.2 Základní rozložení 33
2.3.3 Rozhraní vedoucího soutěže 33
2.3.4 Rozhraní vedoucího týmu 34
2.3.5 Rozhraní pro uživatele bez přiřazené role 34
2.4 Návrh základní architektury 35
2.4.1 MVVM 35
2.4.1.1 Model 36
2.4.1.2 ViewModel 36
2.4.1.3 View 36
2.5 Vybrané technologie 36
2.6 Návrh databáze 36
2.6.1 Typ použité databáze 37
2.6.2 Cloud Firestore 37
2.6.3 Databázový model 38
2.7 Autentizace 40
2.7.1 Firebase Authentication 40

3 Implementace **43**
3.1 Úvod do Androidu 43
3.1.1 Historie 43
3.1.2 Základní komponenty 44
3.1.2.1 Activities 45
3.1.2.2 Services 46
3.1.2.3 Broadcast receivers 46
3.1.2.4 Content providers 47
3.1.3 Ostatní 47
3.2 Zachování dat během konfiguračních změn 47
3.3 Komunikace se vzdálenou databází 50
3.3.1 Základní pojmy 50
3.3.1.1 DAO 50
3.3.1.2 Entita 50
3.3.1.3 Repository pattern 50
3.3.1.4 Coroutines 51
3.3.2 Řešení 51
3.4 View Binding 53
3.5 RecyclerView a Adaptéry 54

3.6 Zajímavé body implementace 55
3.6.1 Generování utkání 55
3.6.2 Domluva utkání 57
3.6.3 Zadávání výsledků 57
3.6.4 Vyhledávání a řazení hráčů/týmů 58
3.6.5 Pořadí týmů 60
3.6.6 Tvorba baráže a ukončení sezóny 61

4 Testování **63**
4.1 Testování autorem aplikace 63
4.1.1 Testování správy paměti 64
4.1.1.1 Leak Canary 64
4.1.1.2 Android Profiler 65
4.1.2 Testování uživatelského rozhraní 65
4.1.2.1 Firebase Test Lab 65
4.1.2.2 Show Layout Bounds 67
4.2 Uživatelské testování 67
4.2.1 Distribuce mezi uživatele 67
4.2.2 Testovací scénář 67
4.2.3 Monitorování průběhu testování 69
4.2.4 Vyhodnocení uživatelského testování 70
4.2.4.1 Shrnutí 72

Závěr **75**
Naplnění cílů 75
Budoucnost aplikace 76

Literatura **77**

A Seznam použitých zkratek **83**

B Ukázka výsledné aplikace **85**

C Propozice AMTEL Opava **87**
C.1 Organizační pokyny 87
C.1.1 Systém soutěže 87
C.1.1.1 Hrací řád 87
C.1.1.2 Nadstavba 87
C.1.2 Věk 88
C.1.3 Registrace na ČTS 88
C.1.4 Vklad do soutěže 88
C.1.5 Soupiska 88
C.2 Technická ustanovení 88
C.3 Závěrečná ustanovení 90

D Obsah přiloženého média **91**

Seznam obrázků

1.1 Zápis o utkání 5
1.2 *HuráLiga* – přehled konrkétní ligy 9
1.3 *HuráLiga* – profil hráče 11
1.4 *VAŠE LIGA* – domluva zápasu 12
1.5 *VAŠE LIGA* – zadání výsledku zápasu 13
1.6 *VAŠE LIGA* – dashboard hráče 14
1.7 *VAŠE LIGA* – hráčovy statistiky 14
1.8 *TennisCall* – zaslání pozvánky 15
1.9 *TennisCall* – správa zápasu 16
1.10 *TennisCall* – hráčovy statistiky 17
1.11 *SportyHQ* – profil hráče 18
1.12 *playwaze* – tvorba vlastní ligy 19
1.13 Diagram případů užití 21

2.1 Rozdělení jednotlivých verzí Androidu 30
2.2 Ukázka wireframů aplikace 32
2.3 Vývoj s Firebase 37
2.4 Schéma databáze 39
2.5 Ukázka výsledné aplikace I 41
2.6 Ukázka výsledné aplikace II 41

3.1 Vývoj loga Android 44
3.2 Životní cyklus aktivity 45
3.3 Srovnání životního cyklu aktivity a viewModelu 48
3.4 Ukázka použití RecyclerView v aplikaci 54
3.5 Správa skupin v rozhraní vedoucího soutěže 61

4.1 Ukázka běhu programu v Android Profiler 64
4.2 Přehled testovaných zařízení v Firebase Test Lab 65
4.3 Ukázka konkrétního testu v Firebase Test Lab 66
4.4 Statistiky testované aplikace v Firebase Crashlytics 69

4.5 Přehled konkrétního pádu aplikace v Firebase Crashlytics 69
4.6 Porovnání sekce *Účet* před a po uživatelském testování 73

B.1 Ukázka výsledné aplikace III 85
B.2 Ukázka výsledné aplikace IV 86
B.3 Ukázka výsledné aplikace V 86

Seznam tabulek

1.1 Testované aplikace 7

2.1 Barevná paleta aplikace 33

4.1 Seznam testovaných zařízení 63

Seznam ukázek kódu

2.1 Ukázka *null-safety* v jazyku Kotlin 31
2.2 Ukázka Kotlinu s *Android KTX* 31
3.1 Ukázka BroadcastReceiveru v praxi 46
3.2 Ukázka komunikace ViewModelu s aktivitou 49
3.3 Ukázka metody v DAO 50
3.4 Spouštění coroutiny pomocí viewModelScope 52
3.5 Porovnání kódu s a bez View Binding 53
3.6 Ukázka fungování *Scheduling* algoritmu 56
3.7 Ukázka použití metody *doCompleteSearch2()* 59
3.8 Přepis českých znaků do anglické „transkripce“ 59
4.1 Ukázka použití Firebase Crashlytics ve zdrojovém kódu 70

Úvod

Amatérská tenisová liga Opava, zkráceně *AMTEL Opava*, je soutěž tenisových týmů odehrávající se každý rok v okrese Opava. Její historie sahá až do doby před sametovou revolucí, kdy vznikla jako ligová soutěž mezi tehdejšími podniky. V dnešní době proti sobě soupeří 54 týmů v několika výkonnostních skupinách. V průběhu roku pak jednotlivé týmy vzájemně hrají utkání a bojují o celkové umístění v tabulce.

Přestože od doby vzniku prošla soutěž[1] výrazným vývojem, je současný stav z hlediska organizace značně zastaralý. Hráči jsou informováni o aktuálním skóre prostřednictvím emailů v týdenních cyklech, chybí přehledné statistiky hráčů i týmů. Vedoucí tenisových týmů musí každý týden přeposílat výsledky utkání vedoucímu soutěže. Ten je ručně zadává do několika tabulek v tabulkovém procesoru. Chybí ucelený komunikační kanál mezi jednotlivými členy ligy.

Celkově se jedná o komplexní soutěž, která vyžaduje mnoho času pro organizaci. Veškeré tyto záležitosti má přitom na starosti pouze jeden člověk, pan Mgr. Jiří Vaněk. Bohužel tak občas soutěž doprovází organizační zmatek, na nějž si stěžují někteří účastníci ligy. S panem Vaňkem jsem se proto sešel a společně jsme se domluvili na realizaci mobilní aplikace pro Android, jež zjednoduší a zpřehlední organizaci takovéto soutěže a odstraní výše uvedené nedostatky.

Cíl práce

Cílem této bakalářské práce je vývoj mobilní aplikace pro operační systém Android, která nahradí současný způsob organizace amatérské tenisové ligy *AMTEL Opava*.

[1] V této práci volně zaměnuji slova *liga* a *soutěž* jako označení pro *Amatérskou tenisovou ligu Opava*.

Aplikace sjednotí různorodou komunikaci mezi jednotlivými vedoucími týmů i vedoucím soutěže. Také nabídne prostor pro zadávání výsledku utkání a jejich okamžitou prezentaci.

Vedoucí soutěže již nebude muset ručně vytvářet soupisky na základě přihlášených týmů. O vše se postará aplikace, jež sama vygeneruje utkání pro každou skupinu zvlášť. Závěr sezóny, tedy tvorba baráže a následné přesunutí týmů do patřičných skupin, bude také v její režii.

Každý uživatel této aplikace bude mít přístup k aktuálním výsledkům týmů napříč všemi skupinami. Také si bude moci zobrazit výsledkové listiny minulých ročníků. Aplikace rovněž povede statistické záznamy individuálních hráčů i týmů. Vedoucí týmů pak můžou tyto záznamy využít pro taktizování během přípravy utkání.

Všechny ligové údaje budou nejen na jednom místě a v přehledné podobě, ale hlavně budou transparentní a okamžitě dostupné všem uživatelům s patřičným oprávněním. Aplikace tak značně urychlí stávající organizaci a usnadní práci všem vedoucím.

Členění práce

Práce je rozdělena do samostatných kapitol podle jednotlivých fází vývoje v uvedeném pořadí – analýza, návrh, implementace a testování.

První kapitola charakterizuje fungování ligy *AMTEL Opava* a vysvětluje současný postup při její organizaci. Také provádí rozbor již existujících alternativ dostupných na trhu. Konec kapitoly je pak určen k analýze cílových uživatelů a sběru požadavků, jež by měla aplikace splňovat.

Další kapitola seznamuje čtenáře s výběrem platformy a volbou implementačního jazyku. Rovněž popisuje návrh uživatelského rozhraní. V neposlední řadě se zabývá návrhem základní architektury a výběrem vhodných technologií včetně návrhu databáze.

Předposlední kapitola začíná úvodem do Androidu, ve kterém popisuje historii a základní stavební prvky tohoto operačního systému. Poté se věnuje způsobu zachování dat během konfiguračních změn. Také se zaobírá komunikací se vzdálenou databází, použitím nové funkce View Binding a využitím třídy RecyclerView spolu s Adaptéry. Na závěr vysvětluje čtenáři zajímavé body implementace.

Poslední kapitola popisuje průběh testování a charakterizuje k tomu použité technologie. Také se zabývá uživatelským testováním aplikace a jejím vyhodnocením.

Kapitola 1

Analýza

V této kapitole popisuji fungování ligy (1.1) a současný postup při organizaci (1.2). Rovněž zde rozebírám již existující alternativy (1.3). V neposlední řadě se pak soustředím na rozbor cílových uživatelů (1.4) a sběr požadavků na aplikaci (1.5).

1.1 Fungování ligy

Za mnoho let existence soutěže došlo k upřesnění organizačních pokynů a pravidel [1] do nejmenších detailů. Neřeší se jenom věk účastníků, ale například i podmínky, za jakých se může hráč registrovaný v *Českém tenisovém svazu* ligy účastnit. Pro zjednodušení jsem tak vynechal některé podrobnosti, jež nejsou pro pochopení fungování ligy důležité. Podrobná pravidla jsou uvedena v příloze C na konci této práce.

1.1.1 Úvod

Amatérská tenisová liga Opava, zkráceně *AMTEL Opava*, je soutěž tenisových týmů hrající se každý rok v okolí Opavy. Zjednodušeně si ji lze představit jako ligu, ve které proti sobě soupeří tenisové týmy bojující o celkové prvenství ve skupině. Liga má v okrese dlouhou tradici. Vznikla jako podniková soutěž před mnoha lety ještě v období socialismu. Od té doby se neustále rozrůstá, a tak v ní nyní najdeme soupeřit 54 týmů. Každý tenisový tým má pak svého vedoucího, který za tým zodpovídá a komunikuje s vedoucím soutěže, Mgr. Jiřím Vaňkem.

1.1.2 Struktura soutěže

Liga je rozdělena do devíti, respektive desíti týdenních kol, v nichž hrají týmy naplánovaná utkání. Protože se utkání konají napříč různými obcemi v okrese, musí týmy přejíždět. Vždy tak jeden tým hostí svého oponenta na domovském

kurtu. Vedoucí domácího týmu pak zodpovídá za organizaci utkání a výsledky posílá vedoucímu soutěže.

Pod záštitou *AMTEL*u se de facto odehrávají dvě soutěže – soutěž pro hráče do 49 let včetně a soutěž pro hráče starší 49 let. Aby se předešlo velkým výkonnostním rozdílům, je soutěž pro hráče mladších 49 let včetně rozdělena do čtyř výkonnostních skupin. Hráči nad 49 let mají skupinu jednu.

1.1.3 Stručná pravidla

Ve všech skupinách se střetne každý tým s každým právě jednou. Během takového střetu jsou odehrány přesně tři zápasy – dvě dvouhry a jedna čtyřhra. Každý zápas se pak hraje podle běžných tenisových pravidel na dva vítězné sety. V průběhu roku týmy sbírají body za vítězství (i prohry) a bojují o celkové prvenství v tabulce.

Podobně jako v jiných ligách, nastává po konci všech odehraných kol baráž (v případě více skupin v soutěži). V baráži se týmy z druhých míst utkají s předposledními týmy vyšší skupiny o postup, respektive udržení. Vítěz skupiny postupuje do vyšší skupiny, naopak poslední tým v tabulce automaticky sestupuje. Další sezóna pak typicky začíná koncem května následujícího roku.

1.2 Postup při organizaci

„*Tento způsob organizace zdá se mi poněkud nešťastným*," dala by se parafrázovat slavná hláška z filmu Rozmarné léto. Zcela totiž vystihuje aktuální situaci. Organizace *Amatérské tenisové ligy Opava* funguje, je ovšem troufalé tvrdit, že současný stav je ideální. V následujících řádcích rozvádím nynější organizační postup na základě rozhovoru s vedoucím soutěže Mgr. Jiřím Vaňkem. [2]

1.2.1 Začátek sezóny

Každý vedoucí tenisového týmu musí před zahájením nové sezóny podat přihlášku, zaplatit za tým a odeslat soupisku týmu vedoucímu soutěže.

„*V dubnu každého roku vyzvu vedoucí týmů, aby mi zaslali závaznou přihlášku, zaplatili vklad a následně vyplnili a zaslali soupisku. Do soupisky lze zasahovat do konce 2. hracího týdne, cca konce května,*" upřesňuje pan Vaněk. [2]

To vše je třeba stihnout do předem určeného data. Vedoucí na to mají obvykle pět týdnů čas. V realitě to ale vypadá tak, že během těchto pěti týdnů odešle přihlášku, vklad a soupisku pouze 60 % vedoucích. [2] Proto musí vedoucí soutěže před blížícím se termínem odevzdání ručně rozesílat emaily

vedoucím týmů, kteří tak doposud neučinili. Rovněž musí odeslat emaily s jednotlivými soupiskami všem vedoucím, již mají možnost zkontrolovat správnost svojí a ostatních soupisek.

Jakmile všichni vedoucí týmů splní své povinnosti, vytvoří vedoucí soutěže harmonogram pro aktuální sezónu. Ten obsahuje rozpis všech utkání a jejich doporučený termín. Termín je přiřazen na základě hracího dne domácího týmu uvedeného v přihlášce. Veškerý proces probíhá ručně v tabulkovém procesoru.

1.2.2 Průběh sezóny

Sezóna je rozdělena do devíti, respektive desíti hracích kol, podle počtu přihlášených týmů ve skupině. Každé kolo trvá přesně týden. Během tohoto týdne jsou odehrána všechna naplánovaná utkání.

Jakmile je utkání odehráno, má vedoucí domácího týmu povinnost zaslat vedoucímu soutěže *Zápis o utkání*. Mohlo by se zdát, že takový úkon je jednoduchý, ale opak je ve skutečnosti pravdou. Vedoucí týmu si totiž prvně musí stáhnout zaslanou šablonu v tabulkovém procesoru (viz obr. 1.1), do té zapsat výsledek, uložit si ji u sebe, a vyplněnou odeslat zpět vedoucímu soutěže. To vše je třeba stihnout do konce daného kola.

AMATÉRSKÁ TENISOVÁ LIGA 2018 - ZÁPIS O UTKÁNÍ

SKUPINA	A
DATUM	24.05.2019
KOLO	1
DOMACÍ	TK Malé Hoštice
HOSTÉ	Kateřinky

D o m á c í	**H o s t é**	**SKÓRE**			**GAME**		**SET**		**BODY**	
JMÉNA	JMÉNA	1.	2.	3.	D	H	D	H	D	H
Jakub Ribka	Martin Beneš	4:6	5:7		9	13	0	2	0	1
Ondřej Roleder	Schröter Jan	3:6	6:2	4:6	13	14	1	2	0	1
Ribka - Roleder	Beneš - Schröter	4:6	5:7		9	13	0	2	0	1

VÍTĚZ UTKÁNÍ	**BODY**	**SETY**	**GAME**
Kateřinky	**0:3**	**1:6**	**31:40**

Zápis zpracoval: Jakub Ribka

Obrázek 1.1: Vyplněná šablona pro zápis o utkání [3, snímek autora]

Vedoucí soutěže pak ke konci týdne ručně zadá výsledky všech odehraných utkání do jednotlivých tabulek skupin. Následně všem hráčům rozešle emaily s doplněnými tabulkami. Tabulky s průběžným pořadím týmů ve skupinách však mezi nimi nejsou. O důvodu jejich absence hovoří pan Vaněk:

„*Tabulka s aktuálním pořadím je zbytečná. Každý tým v tabulce vidí, jak si vedou soupeři, stejně tak, jak na tom jsou průběžně jednotlivé týmy.*“ [2]

Aby došlo k jisté komunikaci mezi vedoucím soutěže a jejími účastníky, rozhodl se pan Vaněk psát čtvrtletní zpravodaje reflektující průběh sezóny. V nich vyjadřuje svůj názor nad děním v lize a motivuje týmy k lepším výkonům. Zpravodaj pak posílá účastníkům *AMTEL*u prostřednictvím emailu a nově je také umisťuje na web v online podobě. Pro představu přikládám krátkou ukázku z posledního zpravodaje:

„*Přímo tragická je situace ve sk. A, kde v zásadě není co hodnotit ani prognózovat. Z 20 utkání je odehráno TRAPNÝCH 8 a jedno je předehrané. Oněch 12 odložených utkání, tedy plných 60 % je ničím neobhájitelných!!!! Nadpolovičních 5 týmů má po pěti hracích týdnech odehráno 1 nebo 2 utkání!!!!! Položte si VŠICHNI otázku, PROČ SE DO TÉ SOUTĚŽE PŘIHLAŠUJETE, když se vám nechce hrát?*“ [4]

1.2.3 Závěr sezóny

Závěr sezóny není z hlediska organizace příliš složitý. Jakmile jsou odehrána všechna kola, hraje se baráž, která rozhodne o složení skupin pro následující rok. Baráž probíhá organizačně stejně jako kola.

1.3 Zhodnocení stávajících alternativ

V této části analýzy provádím průzkum a zhodnocení již existujících řešení. Zaměřuji se na analýzu aplikací, jež se nějakým způsobem podílejí na organizaci sportovní ligy. Ačkoliv takových aplikací existuje na trhu mnoho, velký počet z nich pouze prezentuje aktuální ligové informace. Uživatel v nich nikterak neovlivňuje obsah. Těm se tak v této kapitole vůbec nevěnuji. Hlavní přínos pro mojí práci mají aplikace vyžadující aktivní přístup uživatele.

V tomto textu hodnotím jak mobilní, tak webové aplikace. Výběr platformy není pro tuto analýzu příliš důležitý. Zaměřuji se na samotné funkce, jež by měly aplikace pro organizaci sportovní ligy obsahovat. Mnou předpokládané rysy takových aplikací uvádím v podkapitole 1.3.3.

1.3.1 Metodika vyhledávání existujících aplikací

Průzkum jsem provedl v listopadu roku 2019. Pro vyhledávání jsem použil nejrozšířenější webový vyhledávač [5] – google.com. Do vyhledávacího pole jsem zadal následující klíčová slovní spojení:

- mobile app for organising a sports league,
- sports league app,
- tennis league app,

- aplikace organizace ligy,
- amatérská liga organizace.

Na základě zobrazených výsledků jsem se buď přímo, nebo pomocí vícero odkazů, dostal k hodnoceným aplikacím. Několikrát jsem také objevil testovanou aplikaci dle doporučení v obchodě Google Play (podle mnou již dříve stažených aplikací).

1.3.2 Výběr hodnocených aplikací

Při selekci hodnocených aplikací jsem uvažoval nad projekty, jež svojí existencí umožňují nebo ulehčují organizaci sportovní ligy. Bohužel se však v praxi ukázalo, že je jejich používání z valné většiny zpoplatněno. Rovněž řádné otestování takové aplikace se ukázalo jako netriviální úkol, neboť pro vyzkoušení všech funkcionalit je třeba být aktivním členem nějakého turnaje nebo ligy. Ve výsledku jsem tak vybral aplikace, které splňovaly alespoň jeden z následujících bodů:

- Jako nezúčastněný hráč jsem byl schopen zdokumentovat fungování aplikace a popsat jej.
- Nejasnosti se daly vykomunikovat s tvůrci aplikace.
- Měl jsem k nim přístup od hrajícího účastníka.

Vybrané aplikace jsou dostupné na mobilní nebo webové platformě. Jazykové preference nehrály významnou roli – zahrnul jsem jak mezinárodní projekty převážně v anglickém jazyce, tak domácí projekty v češtině.

Název	Platforma	Jazyk[2]	Sport	Odkaz
SportyHQ	webová	EN	různé	[6]
Tournamentsoftware	webová	EN	různé	[7]
MILO OPEN	webová	CS	tenis	[8]
MULTILIGA	mobilní, webová	CS	různé	[9]
HuráLiga	webová	CS	různé	[10]
VAŠE LIGA	webová	CS	různé	[12]
TennisCall	mobilní, webová	EN	tenis	[13]
playwaze	mobilní	EN	různé	[14]

Tabulka 1.1: Seznam hodnocených aplikací

[2]Primární jazyk aplikace.

1.3.3 Předpokládané rysy aplikací pro organizaci ligy

V této podkapitole uvádím rysy aplikací, u nichž si myslím, že jsou pro zjednodušení organizace ligy signifikantní.

Uživatelské rozhraní aplikace by mělo být plně přizpůsobeno osobám, kteří ji primárně používají, tedy v tomto případě hlavně dospělým. Přehlednost a jednoduchost GUI[3] jsou však hodnoceny kladně. Propracovaná grafika a zvuky nejsou nutné. Pokud zbytečně ruší uživatele, mohou být dokonce nežádoucí. Ovládací prvky aplikace by měly mít jasnou funkci, nad kterou nemusí uživatel déle přemýšlet.

Aplikace by měla současně podporovat přihlášení a uživatelské účty. Nicméně, možnost zobrazení ligových statistik a výsledků je vítaná i pro neregistrované uživatele. Do jednotlivých lig, turnajů nebo zápasů by uživatel měl mít možnost registrovat se zvlášť. Výhodou je pak informace pro administrátora a uživatele o proběhlé platbě (pokud je účast placená).

Přihlášený uživatel by měl mít možnost zadat výsledek odehraného zápasu. Pokud některý z hráčů zadá výsledek špatně, měl by existovat pojistný mechanismus, jenž by umožnil tento omyl vyřešit. Na základě dodaných výsledků by pak aplikace měla vypočítat aktuální pořadí týmů/jednotlivců a vypracovat jejich statistiky. Ty by měly být nejlépe v přehledné formě distribuovány návštěvníkům aplikace.

U každého zápasu by mělo být uvedeno místo a čas konání. Protože výběr správného data a místa není vždy jednoduchý úkol, je vítané, aby aplikace nabídla účastníkům jednotné prostředí pro komunikaci (ve formě diskuse nebo chatu).

Mezi funkce, jež dělají z dobré aplikace perfektní, pak řadím možnost přidávat oblíbené týmy, diskutovat s ostatními uživateli aplikace, vyzvat libovolného hráče na zápas, organizovat vlastní turnaj, zobrazit harmonogram a být upozorněn na blížící se zápas. Oceňuji také originální nápady použité v praxi.

1.3.4 Hodnocení stávajících aplikací

Při hodnocení aplikací jsem vycházel z mnou předpokládaných rysů aplikací pro organizaci ligy uvedených v podkapitole dříve (1.3.3). Pro podrobnější otestování jsem zvolil tři aplikace, u nichž jsem byl schopen rozumným postupem rozpoznat způsob jejich fungování. V hodnocení jsem si dovolil naznačit své úvahy, ke kterým jsem během testování došel. Veškeré hodnocené aplikace jsem analyzoval bez jakýchkoliv utracených finančních prostředků. Zbylé alternativy sumarizuji v části 1.3.4.4. Na konci podkapitoly přikládám závěr (1.3.4.5), jenž uceleným způsobem shrnuje mnou získané poznatky.

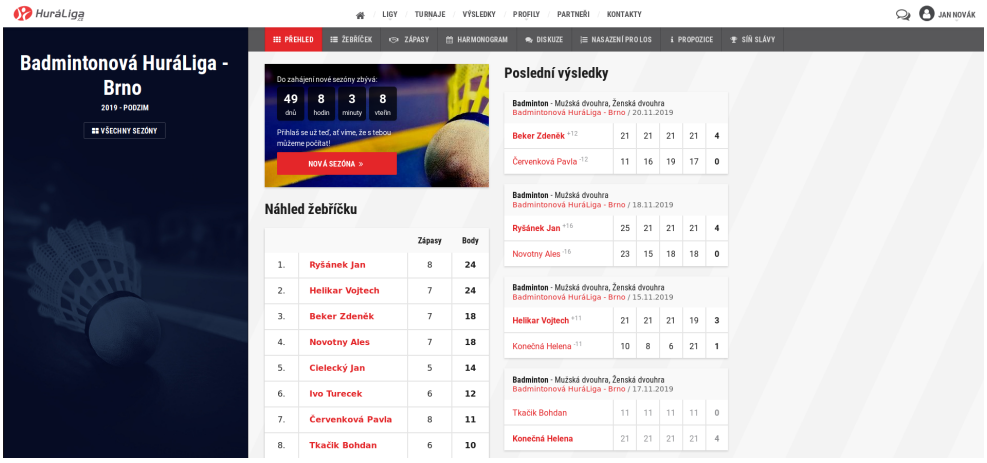
[3] Graphical User Interface – grafické uživatelské rozhraní.

1.3.4.1 HuráLiga

HuráLigu [10] nejlépe definuje příspěvek, kterým lákají autoři portálu nové příznivce na Facebooku: „*Vtáhneme tě do hry! Jsme aktivní sportovci, kteří chtějí posunout amatérský sport o krok dál. Vyvinuli jsme proto webový portál HuráLiga.cz, díky kterému další sportovci jako my poznají nové hráče z okolí a budou pravidelně sportovat. Organizujeme ligy v různých sportech po celé ČR. Navíc každý hráč sleduje svoje statistiky a herní vývoj.*“ [11] Za pravdu jim může dát k dnešnímu dni (21.04.2020) přes sedm tisíc registrovaných uživatelů. [10]

- **Struktura**

  Hráč má možnost zúčastnit se placené ligy nebo turnaje napříč mnoha sporty. Ligy probíhají v jednotlivých městech České republiky a jsou organizovány samotnými tvůrci *HuráLigy*. [10] Každá liga je rozdělena do několika čtrnáctidenních kol, během kterých účastníci vzájemně bojují o co nejlepší umístění v dané sezóně. Na rozdíl od centrálně řízených lig může turnaj pořádat kdokoliv. Přihlášení do ligy nebo turnaje je intuitivní a přehledné. Vše je řádně rozděleno dle měst, sportů a aktuálního stavu (odehráno, probíhá, v přípravě).



Obrázek 1.2: *HuráLiga* – přehled konkrétní ligy [10, snímek autora]

- **Domluva zápasu**

  V každém zápasu je soupeř automaticky přiřazen na základě losu. Losování probíhá většinou Švýcarským systémem, tedy ve zkratce tak, že se hráči na aktuálně lepším místě v tabulce přidělí nejbližší horší hráč, s nímž ještě nehrál. [15] Díky tomuto postupu je tak možné, aby se v lize utkal větší počet účastníků než je počet kol. Zároveň je zaručená

jistá vyrovnanost soupeřů. Výběr konkrétního místa a data si hráči domlouvají prostřednictvím soukromých zpráv přes vnitřní chat, který se zpřístupní po nalosování hráčů. Po domluvení termínu zápasu jej pro informaci ostatním protihráčům a organizátorovi zveřejní některý z hráčů přes ikonu kalendáře přímo v aplikaci. Z mého pohledu tak aplikace během domluvy zápasu nepostrádá žádnou důležitou funkcionalitu. Naopak musím vyzdvihnout systém losů a uživatelsky přívětivé prostředí.

- **Vyhodnocení zápasu**

  Jakmile je zápas odehrán, zpravidla vítěz utkání zapíše výsledek přímo do aplikace. Pokud náhodou nastane situace, že se jeden ze soupeřů není schopen zápasu zúčastnit, může se v aplikaci omluvit a nárokovat kontumační výhru pro oponenta. Zápas je možné odložit a odehrát do termínu losu přespříštího kola, pokud i tak k němu nedojde, ani jeden ze soupeřů nezíská bod. V případě špatně zadaného výsledku má soupeř možnost výsledek opravit nebo nahlásit pořadatelům. Hráči mohou dokonce dočasně odstoupit z ligy a přidat se zpět až budou připraveni. Vyhodnocení zápasu je jednoduché i pro nezkušené uživatele. Autoři si dali opravdu práci a vymysleli řešení, které funguje v praxi.

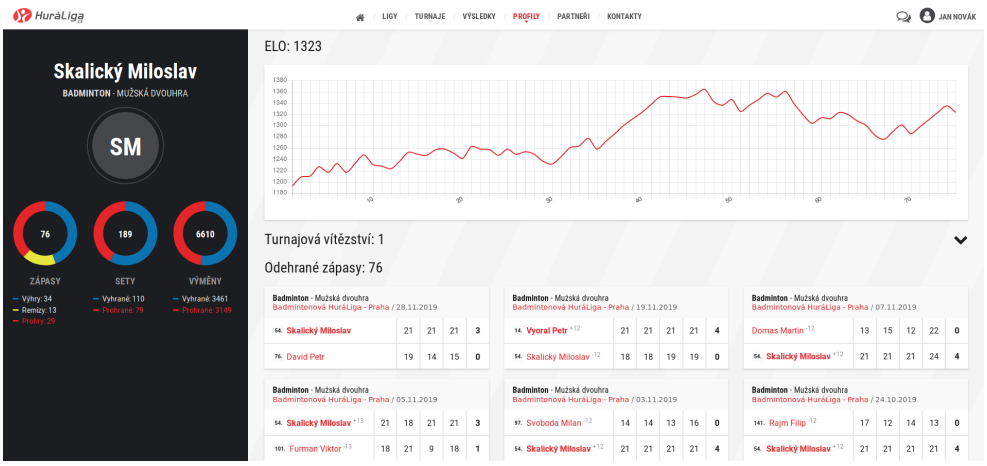
- **Dostupné výsledky**

  HuráLiga [10] nabízí přehled všech výsledků všech zápasů na jedné stránce. Výsledky je možné filtrovat podle sportu a města. Protože není takový systém příliš praktický, má každá liga a turnaj vlastní tabulku s aktuálním žebříčkem hráčů. U každého z nich jsou evidovány body, počet odehraných zápasů, sety, výměny/hry a buch (součet bodů jeho soupeřů). V žebříčku se dá též prokliknout na profil hráče, kde se nacházejí hráčovy statistiky a poslední výsledky zápasů (viz obr. 1.3). Bohužel není možné řadit výsledky podle jednotlivých kritérií. Dostupné výsledky jsou k dispozici i pro neregistrované uživatele.

- **Dostupné statistiky**

  U každého hráče jsou k dispozici statistiky a přehled posledních odehraných utkání (viz obr. 1.3). Mezi zobrazené statistiky patří graf historického vývoje hráčova ELO[4], počty výher, remíz a proher jednotlivých zápasů, setů a výměn/her. Bohužel chybí u grafu ELO přehledné označení osy `x` a možnost zjistit hodnotu ELO v konkrétním bodě. Dostupné statistiky jsou však k dispozici i pro neregistrované uživatele.

[4]ELO, též Elo – je statistické ohodnocení výkonnosti hráče či týmu na základě výsledků jeho her podle systému, jehož autorem byl Arpad Elo. Hráči s vyšším ELO mají statisticky větší šanci na výhru než hráči s nižším ELO. [16]



Obrázek 1.3: *HuráLiga*, – profil hráče [10, snímek autora]

- **Ostatní**

  Stránky dovolují diskusi přihlášených hráčů v rámci ligy nebo turnaje. Zajímavou funkcionalitou je porovnání dvou profilů (vzájemné zápasy, skóre, porovnání ELO) a žebříček nejlepších hráčů v celé republice. Tvorba vlastního turnaje obsahuje pokročilé možnosti nastavení a zajímavý LIVE mod, který prezentuje aktuální pořadí a výsledky na samostatné obrazovce, jež se každých 60 vteřin aktualizuje.

1.3.4.2 VAŠE LIGA

VAŠE LIGA [12] vznikla jako studentský nápad v roce 2007. V té době hrál Tomáš Reinbergr dlouhá léta squash. Na vysoké škole ale přišla krize. Ztratil motivaci a přestalo ho bavit soupeřit se stále stejnými protivníky, u nichž přesně věděl, kam to zahrají. Tehdy se tak v jeho hlavě zrodila první myšlenka na vznik *VAŠÍ LIGY* [12], která je dnes největší amatérskou ligou v České republice. Z počátku fungovala liga pouze na vysokých školách, ale postupně se otevřela široké veřejnosti a momentálně expanduje do zahraničí. Během dvanáctiletého procesu dvakrát změnila název a přidala další sporty do své nabídky.

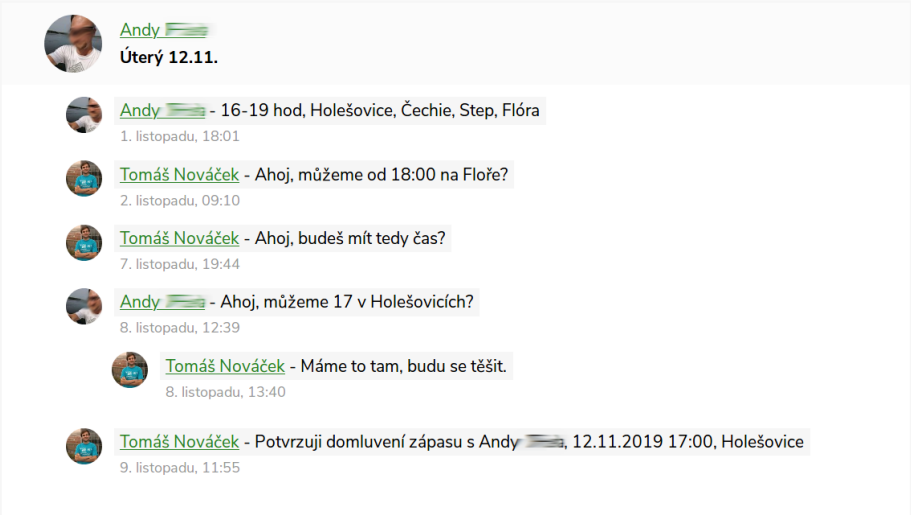
- **Struktura**

  Hráč má možnost zúčastnit se placené ligy pořádané samotnými organizátory *VAŠÍ LIGY*. [12] V úvodu si tak vybere město a sport, ve kterém bude ligu hrát. Registrace do ligy je přehledná, každá liga nabízí přehled hráčů, podrobná pravidla, žebříček, výsledky, harmonogram a další. Pro zobrazení výše zmíněných ukazatelů není třeba být přihlášen.

Ligový ročník se skládá ze tří sezón a každá sezóna ze čtyř kol. Kolo trvá přibližně jeden kalendářní měsíc. Během kola se hráč standardně střetne s dalšími čtyřmi hráči ve skupině. Na základě výsledků se pak hráči po konci kola automaticky přesunou do lepší nebo horší skupiny. Díky tomu je zaručená odpovídající úroveň soupeřů. Pořádat vlastní turnaje není možné.

- **Domluva zápasu**

  Hráči mají měsíc na to, aby odehráli veškeré zápasy ve skupině. Pro plánování termínů existuje v aplikaci tzv. *plánovač*, kam si uživatel předem navolí ideální hrací dny s popisem vyhovujícího místa a času. Účastníci skupiny pak vidí preferované termíny ostatních. Aplikace vhodně upozorňuje na konec kola a naplánování nových termínů. Navíc mezi účastníky skupiny funguje diskuse, kde se mohou společně domluvit na konkrétních podrobnostech (viz obr. 1.4). Domluvené zápasy se hráči zobrazují v přehledném harmonogramu a je možné je exportovat do Google kalendáře.

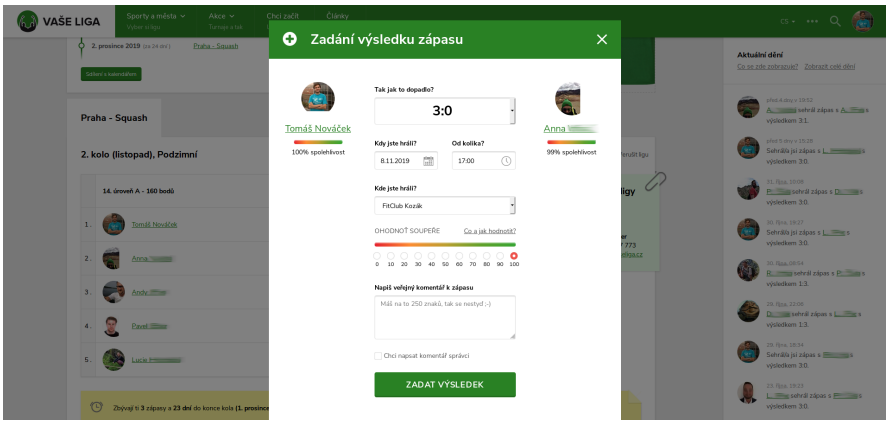


Obrázek 1.4: *VAŠE LIGA* – ukázka domluvy zápasu [12, snímek autora]

- **Vyhodnocení zápasu**

  Ve všech skupinách se hraje systémem každý s každým. Aplikace vytvoří tabulku s danými hráči, kam si pak hráči sami píší výsledky. V praxi to vypadá tak, že jakmile některý hráč odehraje svůj zápas, klikne v tabulce do příslušného pole (průsečík hráčova řádku a soupeřova sloupce), čímž se spustí dialogové okno, do kterého pak zadá výsledek (viz obr.

1.5). Postup je to intuitivní. Pokud není hráč se správnosti výsledku spokojen, může pomocí jiného dialogové okna zažádat o opravu výsledku. K již proběhlým zápasům je možné přidávat komentáře. V případě neschopnosti nastoupit do hry má hráč možnost skrečovat utkání nebo se odhlásit z ligy.



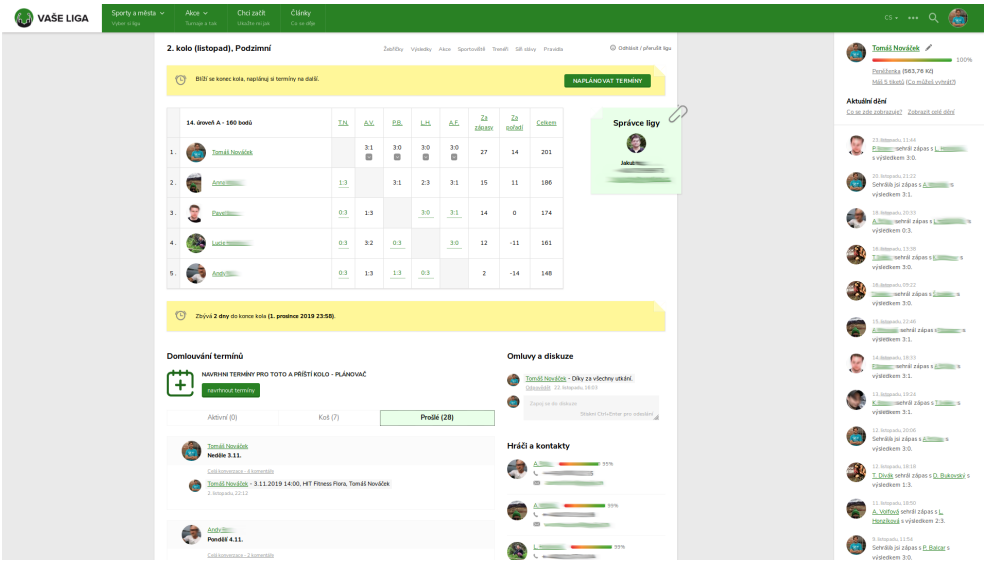
Obrázek 1.5: *VAŠE LIGA* – zadání výsledku zápasu [12, snímek autora]

- **Dostupné výsledky**

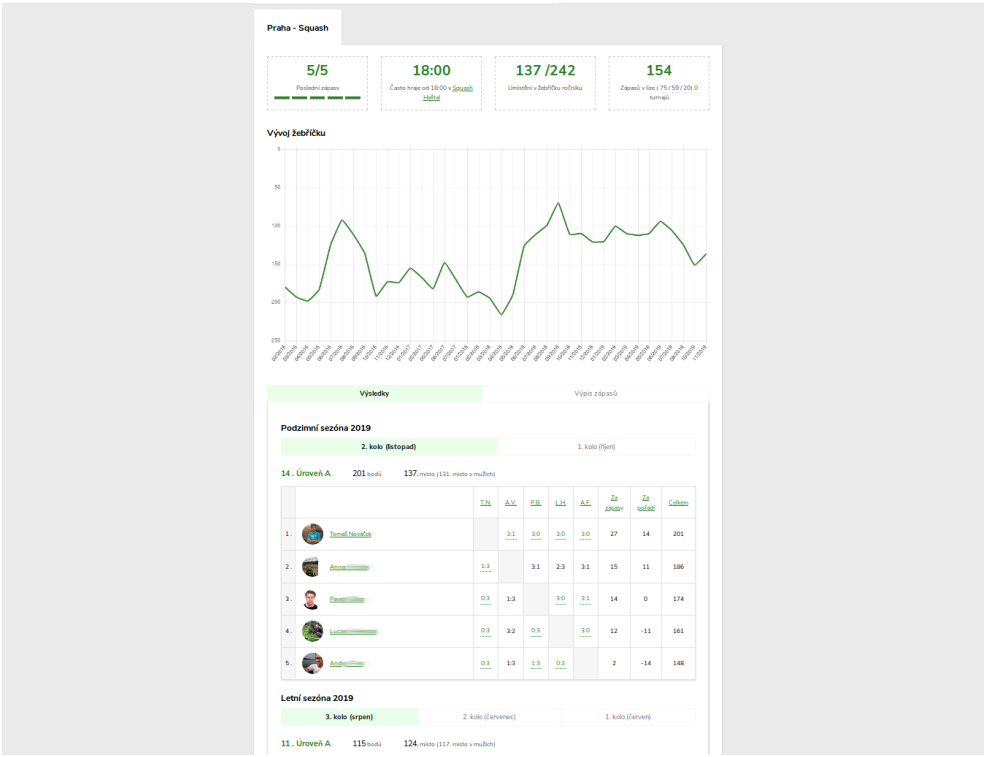
  Dostupné jsou jak výsledky jednotlivých zápasů (viz obr. 1.6), tak žebříčky hráčů napříč všemi skupina v lize. Žebříčky jdou filtrovat podle kol, sezón nebo ročníků. Každou takovou tabulku pak lze řadit podle parametrů (výhry, prohry, skreče, počet bodů za kolo...). V žebříčku všech účastníků je implementováno hledání, výběr pohlaví soutěžících a možnost vyfiltrovat pouze sledované hráče. Výsledky skupiny jsou formátovány do přehledné tabulky (viz obr. 1.6). Je možné se podívat na výsledky kterékoliv skupiny v kterékoliv lize i bez registrace.

- **Dostupné statistiky**

  Všichni účastníci *VAŠÍ LIGY* [12] mají svůj profil, na němž se nacházejí základní informace – jméno, věk, fotka, popisek a seznam lig, v nichž účastník hraje. U každé ligy jsou k dispozici hráčovy statistiky, které evidují vítězství/prohry v posledních pěti zápasech, umístění v žebříčku ročníku a jeho graf, počet zápasů v lize (včetně výher, proher a skrečí) a výpis proběhlých zápasů (viz obr. 1.7). Vše je zobrazeno v graficky přívětivé a přehledné podobě. Přístup k ním mají i neregistrovaní uživatelé.



Obrázek 1.6: *VAŠE LIGA* – dashboard hráče s výsledkovou tabulkou skupiny [12, snímek autora]



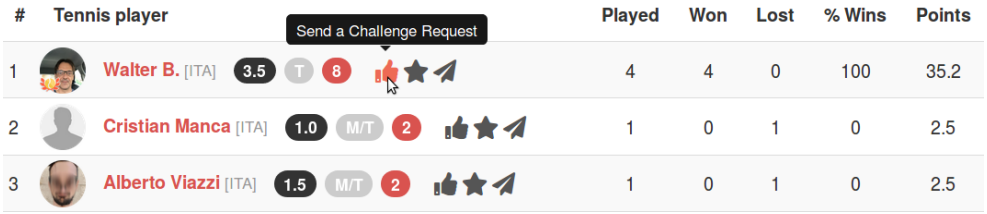
Obrázek 1.7: *VAŠE LIGA* – přehled hráčových statistik [12, snímek autora]

- **Ostatní**

  Jako příjemnou funkci považuji možnost sledovat své oblíbené hráče. Stejně tak si myslím, že tzv. *spolehlivost* je skvělý nápad, jak ohodnotit účastníky za příkladné chování. Každý účastník totiž dostává na základě svého chování zpětnou vazbu. To je tvořeno hodnocením od ostatních soupeřů dle toho, jak hráč vypisuje termíny, reaguje při domluvě zápasů, chová se na kurtu, chodí včas atd. Podle toho si pak může hráč dopředu udělat obrázek o svém protivníkovi a připravit se. Rovněž oceňuji domluvené slevy na různých sportovištích a jejich přehled přímo na stránkách.

1.3.4.3 TennisCall

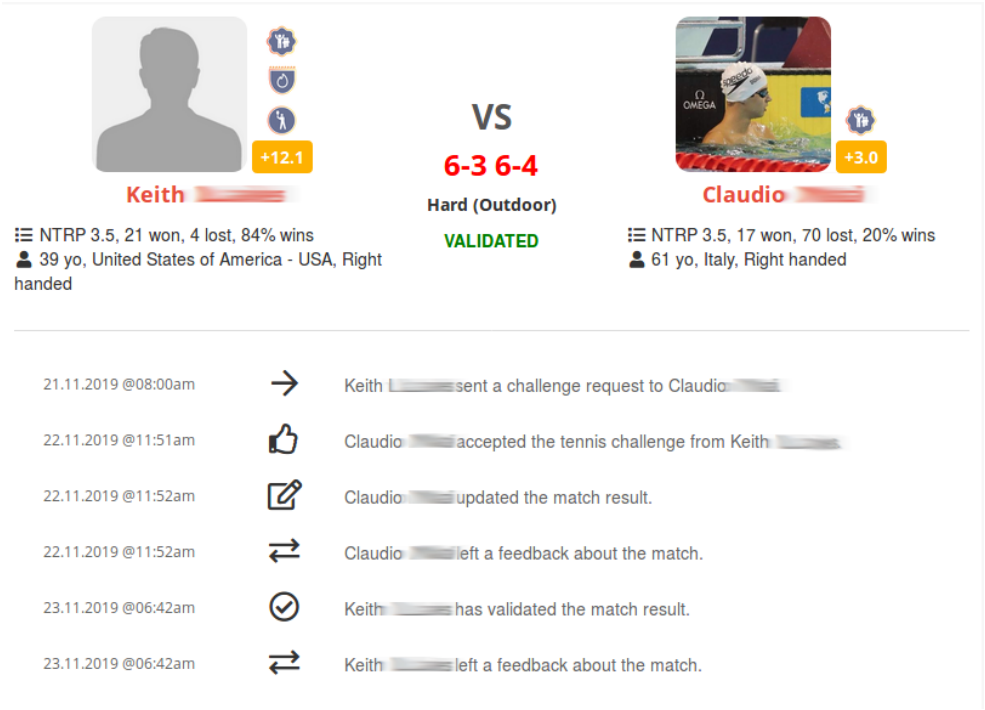
Na rozdíl od dříve zmíněných aplikací (1.3.4.1 a 1.3.4.2) je *TennisCall* [13] zahraniční projekt zaměřený čistě na tenisové prostředí. Ve skutečnosti se pak jedná spíše o sociální síť sdružující amatérské tenisty než o aplikaci zjednodušující fungování ligy. *TennisCall* [13] tak zdarma poskytuje svým uživatelům platformu pro soutěžení s lidmi ze stejného města a komunikaci s jinými hráči. Ačkoliv je projekt dostupný po celém světě, velkému zájmu se těší na Blízkém východě. Aplikace je přístupná jak na webu, tak i jako mobilní aplikace.



Obrázek 1.8: *TennisCall* – zaslání pozvánky k ligovému zápasu [13, snímek autora]

- **Struktura**

  Uživatel je implicitně přiřazen do ligy podle města, které zadal při registraci. Městské ligy se dělí podle dvouhry/čtyřhry a úrovní hráčů v nich hrajících (od začátečníky po pokročilé). Uživatel si může sám vybrat, v jaké kategorii bude hrát. Lize není přiřazen žádný harmonogram ani soupiska zápasů – hráči sami vyzývají své ligové protivníky. Pokud se uživatel nechce účastnit ligy, může dobrovolně odstoupit a utkat se v nehodnoceném přátelském zápasu s libovolným hráčem. Motivací účastníků v městské lize je posun v žebříčku a chuť si zahrát.



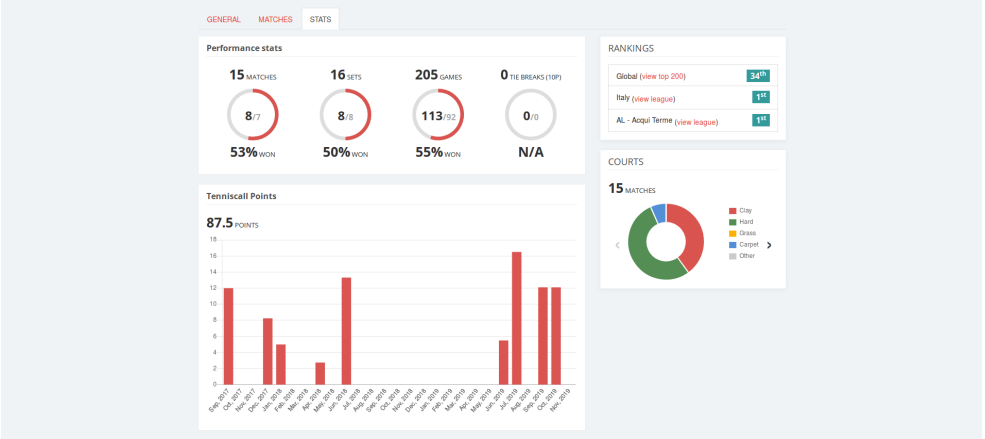
Obrázek 1.9: *TennisCall* – dialogové okno sloužící pro domluvu a zadání výsledku zápasu [13, snímek autora]

- **Domluva zápasu**

  Pokud se uživatel účastní nějaké ligy, má možnost vyzvat kteréhokoliv ze svých protivníků kliknutím na příslušné tlačítko (palec nahoru u soupeřova jména – viz obr. 1.8). Tím je soupeř vyzván a uživatel pak čeká na jeho potvrzení. Zároveň vznikne dialogové okno reprezentující daný zápas (viz obr. 1.9), ve kterém se oba účastníci mohou konkrétně dohodnout prostřednictvím chatu. Dialogové okno je velmi hezky zpracováno, oponenti mají ucelený přehled o fázi a domluvě mače. Uživatel má rovněž pravomoc napsat zprávu libovolnému uživateli aplikace. Ve zprávě se pak můžou domluvit na přátelském zápasu.

- **Vyhodnocení zápasu**

  Po odehrání mače mají oba soupeři možnost v dialogovém okně utkání (viz obr. 1.9) vložit výsledek a vyměnit si vzájemně zpětnou vazbu k zápasu. Zároveň musí protihráč potvrdit správnost skóre. Pokud dojde k neshodě, rozhodnou o stavu administrátoři stránek. Výsledek utkání se ihned promítne do hráčových statistik a ligového žebříčku.



Obrázek 1.10: *TennisCall* – přehled statistik hráče [13, snímek autora]

- **Dostupné výsledky**

  Všechny výsledky všech zápasů jsou dostupné na jednom místě. Aby byla v praxi tato stránka použitelná, má uživatel možnost vyfiltrovat výsledky dle místa konání. Mimo to může konkretizovat svůj dotaz na základě zadaných kategorií (dvouhra, čtyřhra, pohlaví, rok…). Rovněž lze vyhledat aktuální žebříček kterékoliv městské ligy. Pokud by se uživatel rád dozvěděl, jaké výsledky podává konkrétní hráč, musí jej najít v sekci *hráči*. Hráče je možné fulltextově vyhledat.

- **Dostupné statistiky**

  Statistiky (viz obr. 1.10) jsou dostupné po kliknutí na hráčův profil. Podobně jako u již hodnocených aplikací, uživatel se může dozvědět o počtu odehraných zápasů, setů a gemů. U všech je v podobě kruhového diagramu přehledně znázorněno množství výher. Navíc jsou k dispozici umístění v jednotlivých městských ligách a také umístění mezi všemi hráči. Co se mi osobně líbí, jsou dva koláčové grafy prezentující množství vnitřních/venkovních utkání a typ povrchu, na kterém se zápas hrál.

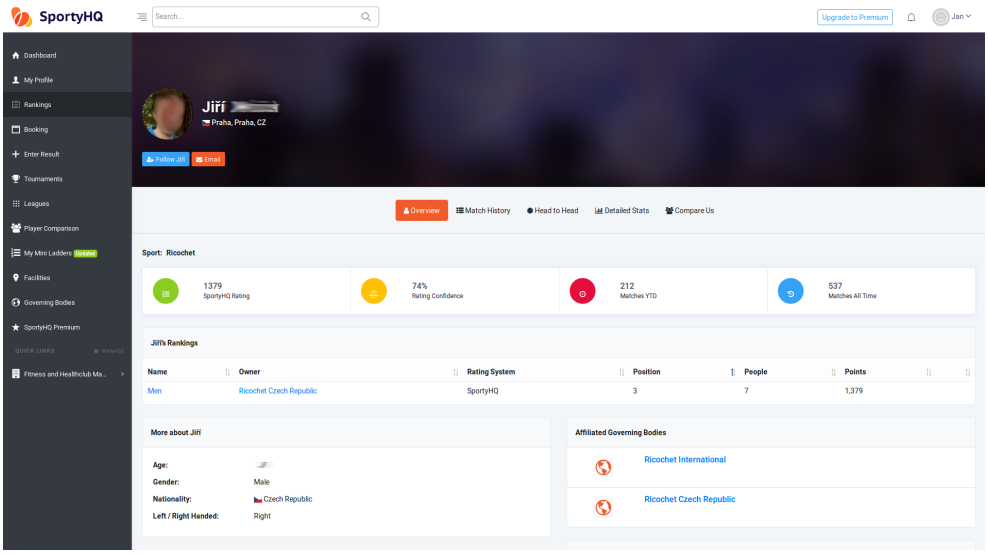
- **Ostatní**

  Za zajímavý nápad musím vyzdvihnout systém odznaků, jež účastníci svým chováním a výkony dostávají. Stejně tak oceňuji možnost vytvářet skupiny hráčů, kteří chtějí hrát spolu nebo je pojí něco společného. V neposlední řadě chválím povedený chat, jenž se funkčně podobá chatu ze známých sociálních sítí.

1.3.4.4 Zbylé aplikace

V této podkapitole se snažím stručným způsobem analyzovat zbylé aplikace uvedené v tabulce 1.1. Spíše než na podrobný popis jednotlivých funkcionalit se tak v tomto textu zaměřuji na jejich charakteristické vlastnosti. Pokud mě aplikace zaujala nějakou mimořádnou funkcí, dovolil jsem si takovou funkci explicitně zdůraznit.

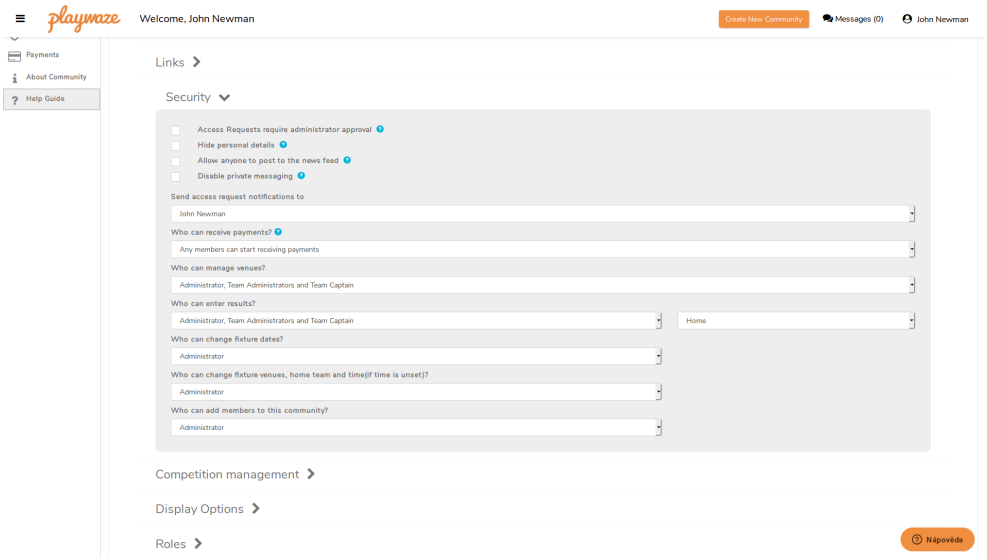
Zbylé aplikace mají vesměs podobnou strukturu. Uživatel má možnost se rozhodnout, zda-li se chce zúčastnit turnaje nebo ligy. Konkrétně v aplikacích *Tournamentsoftware* [7], *SportyHQ* [6] a *playwaze* [14] jsou ligy a turnaje organizovány samotnými uživateli, na rozdíl od aplikace *MULTILIGA* [9], kde ligy pořádají tvůrci aplikace. Od nich se pak odlišuje aplikace *MILO OPEN* [8], jenž je pokusem o převedení organizace ligy *TK Milo Olomouc* do online prostředí a slouží tak pouze úzké skupině hráčů.



Obrázek 1.11: *SportyHQ* – profil hráče ligy [6, snímek autora]

Pokud se bavíme o organizaci ligy, musím vyzdvihnout aplikace *playwaze* [14] a *SportyHQ* [6], které umožňují pokročilou konfiguraci vlastní ligy a dovedl bych si je představit používat jako správce ligy či turnaje. Bohužel je však tato funkcionalita v aplikaci *SportyHQ* [6] placená. Výhodou aplikací *Tournamentsoftware* [7], *SportyHQ* [6] a *playwaze* [14] je pak možnost organizovat týmové soutěže.

Všechny aplikace podporují profily hráčů, u kterých jsou vedeny více či méně pokročilé statistiky. Znovu musím pochválit aplikaci *SportyHQ* [6], jež přehlednou formou zobrazuje velké množství informací o daném hráči. Dále ve všech aplikacích nalezeneme přehled účastníků konkrétní ligy/turnaje, aktuální žebříček hráčů v nich soutěžících a ve většině případů také harmonogram zápasů a výsledky konkrétních mačů.



Obrázek 1.12: *playwaze* – tvorba vlastní ligy je komplexní záležitost [14, snímek autora]

Bohužel jsem nebyl schopen ani u jedné aplikace zjistit, jak probíhá domluva účastníků ohledně společného utkání. Každopádně aplikace *SportyHQ* [6], *playwaze* [14] a *MILO OPEN* [8] umožňují odesílat zprávy v rámci aplikace. Zpráva pak v případě *MILO OPEN* [8] a *SportyHQ* [6] přijde uživateli na email. Je tedy na místě se domnívat, že určitá domluva bude probíhat právě pomocí těchto zpráv. V aplikaci *MULTILIGA* [9] je pak možné vyzvat některé hráče (ty, kteří přijímají výzvy) k přátelskému zápasu.

Graficky je nejméně zdařilá aplikace *Tournamentsoftware* [7], která neodpovídá dnešní době a jejíž responzivní design není plně funkční. Oproti zbylým aplikacím jsou aplikace *MULTILIGA* [9] a *playwaze* [14] dostupné i na mobilních telefonech se systémem Android. Jejich mobilní variantu však nemůžu nikomu doporučit. Mobilní aplikace *playwaze* [14] dokonce během používání často padala a neodpovídala. Nejlépe uživatelsky přívětivá aplikace je tak z mého pohledu *SportyHQ* [6], ve které se uživatel snadno orientuje.

1.3.4.5 Závěr

Na trhu existuje mnoho aplikací, které se určitou měrou podílejí na organizaci sportovní ligy. Uživatel si tak může celkem snadno vybrat právě tu aplikaci, jež mu na základě jeho osobních preferencí nejvíce vyhovuje.

Pokud by se rád zúčastnil ligy v jeho domácím městě, jenom v České republice má několik eventualit. Zároveň má jistotu, že jsou dané ligy skvěle organizovány a tvoří je početná hráčská skupina. Za úvahu rovněž stojí možnost si zahrát ligu či turnaj spravovanou lokálními nadšenci z celého světa.

Co se týče organizace vlastní ligy nebo turnaje, zde výčet eventualit pro uživatele naštěstí neklesá. Kvalitních aplikací je na trhu dostatek, zároveň se v jistých věcech odlišují, takže si uživatel skutečně může vybrat pro něj to nejvhodnější řešení.

Ať už si tedy kdokoliv chce zahrát jeho oblíbený sport, zasoutěžit si s ostatními, poznat nové přátelé nebo vést svůj turnaj, nemusí mít strach s volbou té správné platformy. Jenom musí počítat s tím, že to často nebude zadarmo.

1.4 Cíloví uživatelé

Cíloví uživatelé aplikace se dělí do tří rozdílných kategorií:

- vedoucí soutěže,
- vedoucí týmu,
- host.

1.4.1 Vedoucí soutěže

Vedoucí soutěže definuje množství skupin a počet hrajících týmů v každé z nich. Zároveň zasílá pozvánky ostatním vedoucím týmů, prostřednictvím kterých jim uděluje přístup do organizátorské sekce aplikace. V neposlední řadě má neomezené pravomoce pro změnu libovolných údajů a právo veta během řešení sporných situací. Na rozdíl od jiných uživatelů má možnost psát zpravodaje, jež jsou pak všem dostupné na hlavní stránce.

1.4.2 Vedoucí týmu

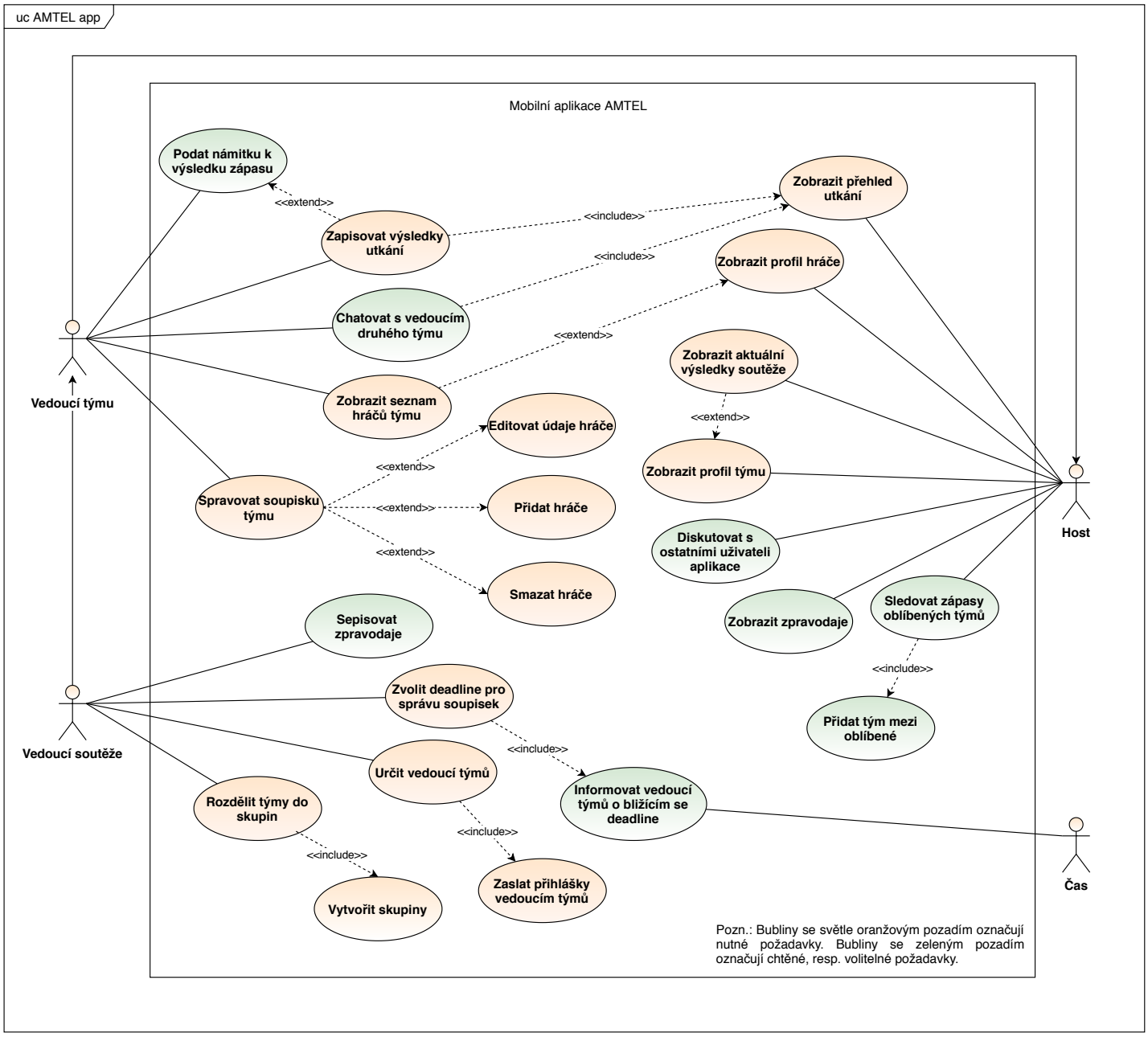
Vedoucí týmu zodpovídá za svůj tým a prostřednictvím aplikace komunikuje s ostatními vedoucími. Konkrétně spravuje soupisku týmu, domlouvá se s dalšími vedoucími týmů nad termínem utkání a zapisuje výsledky proběhlých zápasů. Aby byla zajištěna správnost vložených výsledků, má vedoucí týmu možnost podat námitku, jež je následně zaslána ke zpracování vedoucímu soutěže a vedoucímu druhého týmu. Ten má možnost výsledek opravit.

1.4.3 Host

Host na rozdíl od zbylých dvou cílových uživatelů používá aplikaci zcela bez přihlášení. Nikterak neovlivňuje vnitřní organizaci soutěže, pouze přijímá zobrazená data. V praxi tak má možnost si zobrazit aktuální výsledky skupin, prohlížet si profily týmů i hráčů, diskutovat s ostatními uživateli a číst zpravodaje.

1.4.4 Use case diagram

Pro větší přehlednost přikládám use case diagram (diagram případů užití), který dříve uvedené informace shrňuje do uceleného obrázku (viz obr. 1.13). Cílem takového diagramu je detailní specifikace požadavků na systém, jež zobrazuje účastníky a jejich oprávnění. [17, s. 25]



Obrázek 1.13: Diagram případů užití aplikace *AMTEL Opava*

1.5 Funkční požadavky

V této podkapitole prezentuji funkční požadavky na aplikaci, jež jsem sestavil na základě komunikace s účastníky ligy *AMTEL Opava*.

Cílem funkčních požadavků je jasně vymezit hranice systému, umožnit přesnější odhad pracnosti, vyjasnit si zadání se zákazníkem a zachytit omezení, která jsou na informační systém kladena. [17, s. 3]

Jednotlivé požadavky se dále dělí na funkční a nefunkční. Funkční požadavky definují, co byl měl popisovaný systém obsahovat, jak by se měl v určitých situacích chovat. Nefunkční požadavky pak vyjadřují vlastnosti a omezení služeb, které jsou systémem nabízeny. [18, s. 2]

Pro lepší přehlednost předkládám konkrétní požadavky do souvisejících tematických skupin. Každý požadavek má přiřazenou svoji prioritu:

- **Nutný** – požadavek je nezbytný i pro prvotní fázi systému; tvoří základ, bez kterého by se aplikace neobešla; má nejvyšší prioritu.
- **Chtěný** – požadavek není esenciální pro běh aplikace, přesto je jeho zanesení do aplikace chtěné; jedná se o požadavky rozšiřující stávající funkcionalitu systému; má střední prioritu.
- **Volitelný** – požadavek ne příliš důležitý pro prvotní fáze systému; uvedení takového požadavku v praxi je plánováno do budoucna; má nejnižší prioritu.

1.5.1 Seznam požadavků

F **Funkční požadavky**

a) **Aplikace jako celek**

i. **F1 – Aplikace bude podporovat přihlášení** – nutný
Uživatelé, kteří budou chtít v aplikaci provádět změny, se musí přihlásit. Konkrétně se jedná o vedoucího soutěže a vedoucí týmů. Zbylí uživatelé budou moci do aplikace přistupovat bez přihlášení.

ii. **F2 – Aplikace bude schopna zobrazovat notifikace** – volitelný
Aplikace bude zobrazovat uživateli notifikace, které vyvolala nějaká událost. V praxi se bude jednat o notifikování blížícího se deadlinu, zadání výsledku, odehrání utkání oblíbeného týmu a podobně.

iii. **F3 – Aplikace bude schopna rozesílat emaily účastníkům** – chtěný
Aplikace bude umět rozesílat automatické emaily svým účastníkům. Automatický email je odeslán, pokud se blíží deadline, došlo ke změně údajů, byl zadán výsledek zápasu, vedoucí týmu podal námitku nebo odpověděl na zprávu jinému vedoucímu.

iv. **F4 – Aplikace bude umět přistupovat ke vzdálené databázi** – nutný
Protože se budou data v aplikaci často měnit a je žádoucí k nim přistupovat z vícero zařízení, bude aplikace umět přistupovat ke vzdálené databázi.

b) **Rozhraní aplikace**

i. **F5 – Rozhraní aplikace bude rozděleno na jednotlivé rozhraní dle konkrétních rolí uživatelů** – nutný
Každý uživatel aplikace bude mít jasně přidělenou roli, jež definuje, jaké rozhraní aplikace mu bude zobrazeno. Jinými slovy, některé role budou mít odlišně nastavena práva pro zobrazení a editaci obsahu než jiné role, s čímž souvisí výběr rozhraní. Zároveň platí, že rozhraní pro vedoucího soutěže zahrnuje požadavky z rozhraní pro vedoucí týmů, jež obsahuje požadavky z rozhraní pro ostatní uživatele.

ii. **F6 – Aplikace bude obsahovat rozhraní pro vedoucího celé soutěže** – nutný
Vedoucí soutěže bude mít k dispozici specifické funkce určené pouze pro jeho roli (např. volba vedoucích týmů, editace skupin, generování utkání a jiné). Aplikace proto bude obsahovat rozhraní zahrnující tyto jedinečné funkce. Rozhraní nebude přístupné jiným rolím.

iii. **F7 – Aplikace bude obsahovat rozhraní pro vedoucí jednotlivých týmů soutěže** – nutný
Vedoucí týmů budou mít k dispozici specifické funkce určené pouze pro jejich roli (např. zápis výsledku utkání, chat s ostatními vedoucími, správa soupisky týmu a jiné). Rozhraní pro vedoucí týmů bude tyto funkce obsahovat. Rozhraní nebude přístupné pro uživatele bez přihlášení.

iv. **F8 – Aplikace bude obsahovat rozhraní pro uživatele bez přiřazené role** – nutný
Uživatelé bez přihlášení budou mít svoje vlastní rozhraní, jež znemožní neoprávněně zasahovat do vnitřní organizace ligy. Tedy nepřihlášení uživatelé si budou moci prohlížet výsledky utkání, aktuální pořadí týmů, soupisky týmů, statistiky a jiné bez možnosti jejich úprav.

c) **Rozhraní pro vedoucího soutěže**

i. **F9 – Rozhraní pro vedoucího soutěže bude ukryté před ostatními uživateli** – nutný
Jak již bylo napsáno dříve (viz požadavek F6), rozhraní pro vedoucího soutěže bude zahrnovat funkcionalitu určenou pouze pro samotného vedoucího soutěže. Z tohoto důvodu bude rozhraní ukryto před zbylými uživateli aplikace.

ii. **F10 – Bude možné vytvářet a upravovat skupiny** – nutný
Vedoucí soutěže bude mít pravomoce pro tvorbu nových skupin, jejich editaci a přiřazení týmů do jednotlivých skupin.

iii. **F11 – Bude možné spravovat vedoucí týmů** – nutný
Vedoucí soutěže bude udělovat přístup do rozhraní pro vedoucí týmů těm osobám, které vedou v lize vlastní tým. Také bude moci odebrat vedoucímu jeho roli a umožnit vést tým jinému hráči.

iv. **F12 – Bude možné stanovit konečný termín pro správu soupisek vedoucími týmů** – nutný
Každý vedoucí týmu má povinnost vytvořit soupisku vlastního týmu před zahájením nové sezóny. Vedoucí soutěže bude mít v aplikaci možnost stanovit deadline pro odevzdání finální verze soupisky.

v. **F13 – Rozhraní bude přístupné pouze po přihlášení** – nutný
Z důvodů uvedených v požadavku F9 bude rozhraní pro vedoucího soutěže přístupné pouze po přihlášení prostřednictvím zadané kombinace emailu a hesla.

vi. **F14 – Vedoucí soutěže bude mít neomezená práva pro editaci účastníků, utkání, soupisek, skupin apod.** – nutný
Vedoucí soutěže zodpovídá za regulérní a správný průběh celé ligy, z tohoto důvodu bude mít v aplikaci právo veta a neomezená práva pro editaci obsahu.

vii. **F15 – Bude možné sepsat a zveřejnit článek (zpravodaj) reflektující uplynulých několik týdnů soutěže** – chtěný
Vedoucí soutěže píše pro oživení ligy zpravodaje, ve kterých hodnotí průběh aktuální sezóny (viz podkapitola 1.2.2). Aplikace umožní zápis takovýchto zpravodajů a jejich zveřejnění uživatelům.

d) **Rozhraní pro vedoucí týmů**

i. **F16 – Rozhraní bude přístupné pouze po přihlášení** – nutný
Z důvodů uvedených v požadavku F7 bude rozhraní pro vedoucí týmů přístupné pouze po přihlášení prostřednictvím zadané kombinace emailu a hesla.

ii. **F17 – Bude možné zapisovat výsledky jednotlivých utkání, ve kterých hrál daný tým jako domácí** – nutný
Vedoucí týmů budou moci zapisovat výsledky domácích utkání přímo v aplikaci, aby se tak zjednodušil a zrychlil současný způsob organizace.

iii. **F18 – Vedoucí týmu hostů bude moci podat námitku k výsledku zápasu** – chtěný
Protože chybovat je lidské, bude moci vedoucí hostujícího týmu podat námitku k výsledku proběhlého zápasu. Námitka pak bude odeslána na email vedoucímu domácího týmu a vedoucímu soutěže.

iv. **F19 – Bude možné spravovat soupisku týmu do stanoveného termínu** – nutný
Každý vedoucí týmu má povinnost sestavit soupisku týmu před zahájením nové sezóny. Odevzdat ji musí do termínu stanoveného vedoucím soutěže. Aplikace bude umožnovat tvorbu a správu takové soupisky do zvoleného data.

v. **F20 – V okně utkání bude možné chatovat s vedoucím druhého týmu** – chtěný
Aby se vedoucí hrajících týmů dohodli na společném termínu a čase utkání, budou moci spolu komunikovat (chatovat) v okně utkání.

e) **Rozhraní pro ostatní uživatele**

i. **F21 – Bude možné zobrazit výsledky jednotlivých zápasů, tabulku s aktuálním pořadím týmů a statistiky jednotlivých hráčů** – nutný
Název požadavku je samovysvětlující.

ii. **F22 – Bude možné sledovat utkání oblíbených týmů** – volitelný
Uživatelé si budou moci přidávat své oblíbené týmy a sledovat jejich utkání. Pokud oblíbený tým odehraje libovolné utkání, přijde uživateli notifikace o výsledku utkání.

iii. **F23 – Bude možné diskutovat s ostatními uživateli** – volitelný
V aplikaci se bude nacházet prostor pro společnou a svobodnou diskusi uživatelů.

iv. **F24 – Bude možné zobrazit jednotlivé zpravodaje sepsané vedoucím soutěže** – chtěný
Jak již bylo napsáno dříve (viz požadavek F15), vedoucí soutěže píše své občasné komentáře (pojmenovány jako zpravodaje) k aktuálnímu vývoji ligy. Uživatelé si tyto zpravodaje budou moci v aplikaci zobrazit.

v. **F25 – Rozhraní pro ostatní uživatele bude pouze v omezeném režimu (omezené zobrazení osobních údajů hráčů, nemožnost organizanizačních změn apod.)** – nutný
Aby se předešlo neoprávněným změnám ve vnitřní organizaci ligy, nebudou mít nepřihlášení uživatelé možnost měnit organizační záležitosti. Rovněž z důvodů ochrany soukromých údajů budou mít uživatelé omezený režim, který nezobrazí některé citlivé údaje hráčů.

f) **Autentizace**

i. **F26 – Půjde vytvořit profil vedoucího soutěže** – nutný
Profil vedoucího soutěže bude vytvořen autorem této práce. Autor této práce zašle email vedoucímu soutěže s vygenerovaným heslem, kterým se vedoucí soutěže do aplikace přihlásí. Hned po přihlášení si jej bude moci změnit.

ii. **F27 – Půjde vytvořit profil vedoucího týmu** – nutný
Profil vedoucího týmu bude vytvořen na základě pozvánky vedoucího soutěže. Pozvánka bude obsahovat pseudonáhodně vygenerované heslo, kterým se vedoucí týmu přihlásí poprvé do aplikace. Poté si jej bude moci změnit.

iii. **F28 – Přihlášený uživatel bude identifikován pomocí emailu a hesla** – nutný
Uživatel musí pro přihlášení zadat platnou kombinaci emailu a hesla.

iv. **F29 – Profil vedoucího týmu bude přístupný pouze přes pozvánku** – nutný
Název požadavku je samovysvětlující (viz požadavek F27).

v. **F30 – Část pro ostatní uživatele bude dostupná bez nutnosti přihlášení** – nutný
Název požadavku je samovysvětlující.

g) **Organizační část**

i. **F31 – Aplikace bude podporovat tvorbu soupisek jednotlivých týmů soutěže** – nutný
Aplikace umožní vedoucím týmů vytvořit a spravovat soupisku vlastního týmu.

ii. **F32 – Aplikace vytvoří přehled utkání na základě přihlášených týmů ve skupinách** – nutný
V přehledu utkání budou vypsána jednotlivá utkání dle přihlášených týmů ve skupinách.

iii. **F33 – Aplikace bude podporovat vkládání výsledků odehraných utkání** – nutný
Pro zjednodušení a zrychlení současného postupu organizace bude aplikace umožňovat vedoucím týmů a soutěže vkládat výsledky odehraných utkání.

iv. **F34 – Aplikace vypočítá aktuální pořadí týmů a hráčů na základě dostupných výsledků** – nutný
Název požadavku je samovysvětlující.

v. **F35 – Aplikace vytvoří soupisky skupin pro následující sezónu na základě umístění týmů v předchozí sezóně (včetně odehrané baráže)** – nutný
Název požadavku je samovysvětlující.

vi. **F36 – Aplikace bude notifikovat vedoucí týmů ohledně blížících se utkání** – volitelný
Každý vedoucí týmu bude mít k dispozici harmonogram nadcházejících utkání. Jakmile se bude utkání blížit, aplikace na něj vedoucího týmu upozorní.

vii. **F37 – Pokud se bude blížit deadline tvorby soupisek, aplikace pošle automatický email s upozorněním vedoucím týmů, kteří ještě soupisku nesestavili** – chtěný
S blížícím se termínem odevzdání soupisek bude aplikace automaticky rozesílat přednastavené emaily s upozorněním, aby vedoucí týmů soupisku odevzdali. Emaily budou zasílány se vzrůstající tendencí těm vedoucím, kteří tak do té doby neučinili.

N **Nefunkční požadavky**

a) **Aplikace jako celek**

i. **N1 – Aplikace bude dostupná na zařízeních s operačním systémem Android** – nutný
Aplikace bude programována a optimalizována pro mobilní telefony a tablety běžících na operačním systému Android (výběr operačního systému se dočtete v podkapitole 2.1). Minimální verze API[5] bude 22 (viz podkapitola 2.1.1).

ii. **N2 – Aplikace bude vytvořena na základě pravidel pro hráče do 49 let včetně** – nutný
Aplikace bude naprogramována dle pravidel ligy *AMTEL Opava* pro skupiny A, B, C, D (hráči do 49 včetně) s možným rozšířením do budoucna pro skupinu 50+ a vedlejší ženskou ligu.

iii. **N3 – Aplikace bude podporovat dočasné uložení dat offline** – chtěný
Dočasným uložením dat se rozumí uložení do doby, než se mobilní zařízení opětovně připojí k internetu nebo než je překročena velikost cache paměti, do které se data dočasně ukládají. Velikost cache paměti je implicitně nastavena na 100 MB (viz podkapitola 2.6.2).

b) **Rozhraní aplikace**

i. **N4 – Rozhraní aplikace bude přehledné** – chtěný
Rozhraní aplikace je přehledné, pokud si alespoň 80 % uživatelů během uživatelského testování aplikace myslí, že rozhraní je přehledné.

[5] Application Programming Interface – sada funkcí a postupů umožňujících vytváření aplikací, které přistupují k funkcím nebo datům operačního systému, aplikace nebo jiné služby. [19]

Kapitola 2

Návrh

V této kapitole rozebírám výběr platformy (2.1) a volbu implementačního jazyku (2.2). Rovněž zde popisuji návrh uživatelského rozhraní (2.3). V neposlední řadě se v této části práce zabývám návrhem základní architektury aplikace (2.4) a výběrem vhodných technologií (2.5) včetně návrhu databáze (2.6).

2.1 Výběr platformy

Výběr správné platformy je důležitý proces během návrhu aplikace. Naštěstí můžeme v dnešní době říci, že pro vývojáře již není tak složitý, jako dříve. Období, kdy si koncový uživatel mohl vybrat mezi telefony s operačním systémem Symbian, Windows Phone, BlackBerry OS, iOS nebo Android, je od konce roku 2011 nenávratně pryč. [20] V tom čase totiž začaly trhu dominovat poslední dva zmíněné operační systémy, iOS a Android. [20] Jejich dominance se nyní nedá popřít.

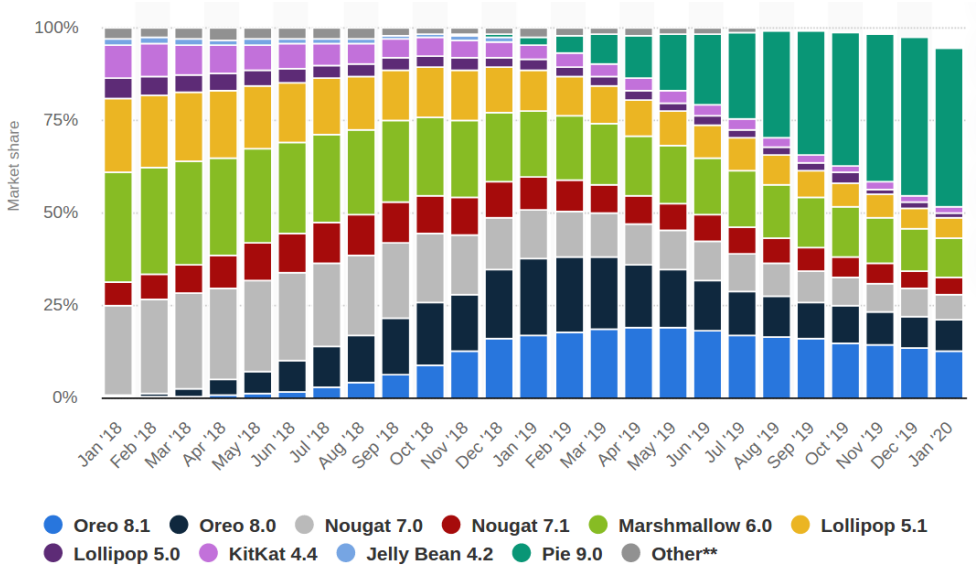
K dnešnímu datu (21.04.2020) zaujímá iOS s 27 % druhé místo a na vrcholu pomyslného stupně vítězů stojí Android s 72% tržním podílem. [21] Pro zbývající soupeře (KaiOS, Samsung, Windows a ostatní) tak připadá pouhé necelé jedno procento podílu.

V dnešní době se zároveň nabízí možnost vyvíjet aplikace pro obě platformy zároveň. Takový vývoj se označuje jako hybridní. Narozdíl od nativních aplikací, tedy aplikacích závislých na konkrétní platformě, jsou hybridní aplikace tvořeny kombinací webových a nativních technologií. Aplikace tak využívá webové technologie a běží částečně v prohlížeči, je ale zapouzdřena v nativní aplikaci. [22] Díky tomu je práce rychlejší a levnější. Nevýhodou takového přístupu je ovšem menší výkonnost, horší uživatelská zkušenost, těžší údržba a složitější ladění chyb.

Z důvodu největšího zastoupení na mobilním trhu a snazšího ladění chyb jsem se proto rozhodl pro vývoj aplikace v Androidu.

2.1.1 Minimální verze

Operační systém Android je dostupný na trhu již od roku 2008. [23] Za tu dobu vzniklo 17 verzí [23], jež postupně přidaly nové funkce nebo upravily stávající. Určitě můžeme říci, že se vývoj v Androidu během 12 let značně posunul vpřed. Při tvorbě nové aplikace se proto musí Android vývojář rozhodnout, jakou nejnižší verzi bude jeho aplikace podporovat. Zde je vhodné nalézt rovnováhu mezi nabízenými funkcemi a počtem mobilních zařízení, které danou verzi podporují. Čím nižší verze je, tím vyšší počet telefonů a tabletů (nyní i televizí a hodinek) umožňuje aplikaci nainstalovat. Nevýhodou je pak menší počet funkcí a prvků, jež má vývojář k dispozici.



Obrázek 2.1: Rozdělení jednotlivých verzí Androidu [24, snímek autora]

Z výše zmíněných důvodů jsem si vybral minimální verzi 5.1 (API 22). Tuto verzi a vyšší používá dle 2.1 přes 97 % zařízení s Androidem. Zároveň se jedná o dostatečně moderní verzi, jež nebude vývoj příliš omezovat.

2.2 Implementační jazyk

Pro tvorbu aplikací v Androidu je možné použít programovací jazyk Java nebo Kotlin, který společnost Google podpořila v roce 2017. [25] Kotlin je moderní staticky typovaný jazyk, který vyvíjí firma JetBrains. Oproti Javě, se kterou je plně kompatibilní, má několik výhod navíc. Zdrojový kód stejného programu je v Kotlinu čitelnější a kratší. Další nespornou výhodou je tzv. *null-safety*, kdy kompilátor při překladu kontroluje práci s nullovatelnými typy a v určitých

případech nám nedovolí program zkompilovat. Díky tomu je za běhu programu riziko vyhození výjimky *NullPointerException* mnohem menší.

```
var foo: FooClass = FooClass()
foo = null //error pri kompilaci
var nullableFoo: FooClass? = FooClass()
nullableFoo = null //v poradku
nullableFoo.bar() //error pri kompilaci
nullableFoo?.bar() //v poradku, metoda bar() se zavola pokud
    nullableFoo neni null
```

Ukázka kódu 2.1: Ukázka *null-safety* v jazyku Kotlin

Dva roky po prvním zmínění podpory Kotlinu společnost Google oznámila, že se Kotlin stává primárním jazykem pro vývoj v Androidu a optimalizuje pro něj nové funkce a prvky. [25] Google také vytvořil set rozšíření zvaný *Android KTX*, jenž poskytuje stručnější, příjemnější a idiomatičtější použití Kotlinu pro práci s Androidem. [26] Dle Google I/O 2019 [25] se tak již více než polovina Android vývojářů rozhodla začít programovat v Kotlinu. Díky svým moderním vlastnostem, promyšlenému designu, vysoké podpoře a oblíbenosti mezi uživateli se Kotlin jeví jako vhodný jazyk pro vývoj v Androidu.

```
//Kotlin
PreferenceManager
   .getDefaultSharedPreferences(context)
   .edit()
   .putString(SOME_TEXT, "Chewie, we are home.")
   .apply()

//Kotlin + Android KTX
PreferenceManager
   .getDefaultSharedPreferences(context)
   .edit { putString(SOME_TEXT, "Chewie, we are home.") }
```

Ukázka kódu 2.2: Ukázka Kotlinu s *Android KTX*

2.3 Návrh uživatelského rozhraní

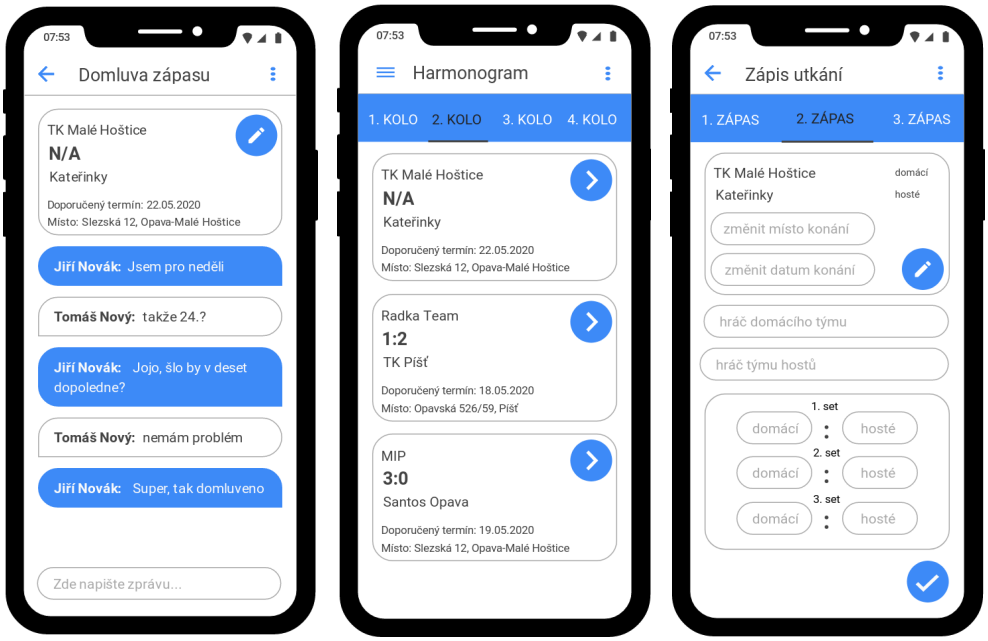
U návrhu uživatelského rozhraní pro Android je nutné brát v potaz velký rozdíl mezi zařízeními, na kterých aplikace poběží. Některé starší telefony mají menší displeje s nižším rozlišením, naopak moderní zařízení včetně tabletů disponují velkými kvalitními displeji. Pro uživatele je v každém případě žádoucí, aby se vzhled aplikace dokázal přizpůsobit parametrům displeje. S touto myšlenkou jsem se tak během návrhu rozhraní snažil pracovat.

Základem bylo vymyslet koncept tak, aby aplikace byla přehledná a intuitivní na všech dostupných mobilních zařízeních včetně tabletů. Nejlepší možností, jak toho dosáhnout, je vytvořit více layoutů – souborů definující

vzhled jednotlivých obrazovek v aplikaci – závislých na druhu displeje (malý, velký, s nízkým/vysokým rozlišením). Tento přístup je ovšem časově náročný. Kompromisem se tak pro mě stalo řešení, ve kterém jsem navrhl jedno schéma pro všechny dostupné zařízení s použitím responzivních prvků. Schéma jsem v konečné fázi vývoje otestoval na dostatečném množství zařízení prostřednictvím nástroje *Firebase Test Lab* (více v podkapitole 4.1.2.1).

Můj další krok směřoval k rozložení jednotlivých elementů na mobilním displeji tak, aby i přes omezenou šířku displeje byly údaje přehledné a srozumitelné. Tento problém jsem vyřešil rozdělením obsahu do samostatných obrazovek, mezi se může uživatel jednoduše přesouvat. Pro snazší orientaci se navíc v levé části obrazovky bude nacházet navigační menu.

Aplikaci jsem se také rozhodl rozdělit do vícero uživatelských rozhraní (viz požadavek F5) podle konkrétní role uživatele. Tím se rozumí, že vedoucí soutěže po přihlášení do aplikace uvidí jiný obsah než vedoucí týmu, respektive návštěvník bez jakékoliv přiřazené role.



Obrázek 2.2: Wireframy znázorňující domluvu zápasu, harmonogram a zápis utkání

2.3.1 Barevná paleta a font

S ohledem na používání aplikace ve venkovním prostředí jsem zvolil světlé téma, jež zajišťuje dostatečnou zřetelnost i na přímém světle. Jako primární barvu jsem vybral světle modrou, která evokuje pocit důvěryhodnosti a má uklidňující vliv na uživatele. [27] Tu doplňuje bílá a světle šedá barva na po-

zadí. Pro text budu využívat zejména tmavě šedou barvu, ale taky světle šedou a modrou pro nejméně, respektive nejvíce relevantní informace. Pro aplikaci jsem dále vybral font *Open Sans*, jenž je dostupný na webové stránce Google Fonts pod open source licencí [28] a může tak být bezpečně distribuován.

Název	Kód	Vizualizace
Primární modrá	#2FB7F4	
Velmi světle šedá	#FAFAFA	
Bílá	#FFFFFF	
Tmavě šedá	#444444	
Světle šedá	#AAAAAA	
Červená	#E8411	

Tabulka 2.1: Barevná paleta aplikace

2.3.2 Základní rozložení

K přehledné a srozumitelné navigaci v aplikaci poslouží *toolbar* – jedna z navigačních UI[6] komponent umístěná pod řádkem s notifikacemi a časem, jež v mém případě bude obsahovat titulek a ikonu navigačního menu. Po kliknutí na ikonu se otevře rozbalovací menu s dostupnými možnostmi. Pro větší přehlednost budou možnosti v menu podpořeny patřičnými ikonami a vybraná možnost bude barevně odlišena od ostatních. Výhodou takového návrhu je fakt, že uživatel je již zvyklý tento postup používat z jiných aplikací.

Části spjaté s konkrétní rolí uživatele jsem se snažil umístit tak, aby byly pro uživatele co nejvíce intuitivní. Pokud se například vedoucí týmu přihlásí do aplikace, bude automaticky přesměrován do části, ve kterém může spravovat soupisku týmu nebo editovat své osobní údaje. Konečným záměrem je, aby uživatel bez složitého hledání docílil svého úmyslu. Konkrétní příklady uvádím v další podkapitole.

2.3.3 Rozhraní vedoucího soutěže

Na úvod je vhodné zopakovat, jaké činnosti bude vedoucí soutěže v aplikaci vykonávat. Zjednodušeně řečeno se bude starat o celkovou organizaci soutěže. Tedy vytvářet skupiny, generovat utkání, udělovat přístup vedoucím týmů do administrátorské sekce (vedoucí se pak může přihlásit a spravovat vlastní tým). Bude mít možnost přesunout tým do libovolné skupiny, editovat údaje jakéhokoliv hráče, vytvářet nebo upravovat výsledky všech utkání a psát zpravodaje. Aby se tak vedoucí soutěže dokázal v aplikaci orientovat, bylo třeba rozhraní vhodně navrhnout.

[6] User Interface – uživatelské rozhraní.

Většinu organizačních záležitostí jsem se rozhodl umístit na jedno společné místo, respektive do jedné části aplikace, která se ale bude kvůli větší přehlednosti dělit na menší celky. Do ní bude vedoucí soutěže přesměrován automaticky po přihlášení. Menší celky pak umožní provést konkrétní činnost a vedoucí se mezi nimi může jednoduše přesouvat.

Ne vždy je ale vhodné umístit vše na jedno místo. Například seznam zpravodajů od vedoucího soutěže bude samostatnou obrazovkou, kterou si můžou zobrazit všichni uživatelé aplikace. Rozdíl mezi vedoucím soutěže a zbylými uživateli je ale ten, že vedoucí soutěže glosuje aktuální dění v lize. Protože je intuitivní upravovat a vytvářet příspěvky v místě stávajících, bude tvorba nových příspěvků umístěna právě tam. Viditelná ovšem zůstane pouze pro vedoucího soutěže.

To stejné platí o editaci hráčů. Seznam všech hráčů bude dostupný na samostatné obrazovce viditelné pro všechny uživatele. Vedoucí soutěže ale jako jediný bude moci upravovat a mazat hráče právě tam.

2.3.4 Rozhraní vedoucího týmu

Koncepčně bude rozhraní vedoucího týmu dost podobné rozhraní vedoucího soutěže. Vedoucí týmu může v aplikaci aktualizovat soupisku hráčů a týmové údaje (adresa kurtu, hrací dny). Mimo to bude zadávat nebo kontrolovat výsledky utkání. Do administrátorské sekce (správa týmu) proto bude přesměrován automaticky po přihlášení.

Jednotlivá utkání jsem se rozhodl rozdělit podle skupin do samostatných kol. Utkání, ve kterých se bude přihlášený vedoucí týmu nějakým způsobem angažovat (jeho tým se zúčastní daného utkání), budou odlišena jinou barvou. Po kliknutí na příslušnou kartu utkání bude vedoucí přesměrován do sekce, kde proběhne domluva s vedoucím protějšího mužstva.

Z domluvy utkání se následně půjde dostat k samotnému zadávání výsledku. Protože se jedno utkání skládá ze tří samostatných zápasů, rozhodl jsem se rozdělit každý zápas zvlášť do oddělených obrazovek.

2.3.5 Rozhraní pro uživatele bez přiřazené role

Nepřihlášený uživatel bude moci procházet profily týmů a hráčů, uvidí rozpisy utkání i aktuální pořadí týmů napříč skupinami a ročníky. Také může číst zpravodaje, vyhledávat konkrétní hráče či týmy, může si připomenout pravidla soutěže nebo se realizovat v diskusi pod utkáním. Vše bude umístěno v samostatných obrazovkách, mezi kterými bude možnost se plynule pohybovat. Například z okna zápasu povedou odkazy k týmům a hráčům, kteří v daném zápase hráli. U profilu týmů může uživatel rozkliknout konkrétního hráče nebo utkání, u profilu hráče naopak konkrétní tým či zápas. Kliknutím na hráčovu kartu se uživatel z celkového seznamu hráčů dostane na hráčův profil. U seznamu týmů to bude obdobné.

2.4 Návrh základní architektury

Volba správné architektury je neméně důležitým stavebním prvkem celé aplikace. Líbí se mi přirovnání, jež je uvedeno na stránce ITnetwork.cz: „*Programování aplikací si můžeme představit jako stavbu výškové budovy. Když ji architekt špatně navrhne, od určité výšky nám spadne. Musí mít pevnou páteřní strukturu.*“ [29]

Pokud chceme uživateli zobrazit libovolná data, pak je vhodné použít třívrstvou architekturu. Třívrstvá architektura dělí aplikaci do tří samostatných vrstev podle konkrétní funkce. Vrstvy jsou na sobě nezávislé a úprava jedné z nich má malý vliv na zbylé dvě. Tento přístup zajišťuje přehlednost, snazší testování a usnadňuje flexibilitu celého projektu. V praxi se nejčastěji používá právě jeden z těchto tří návrhových vzorů:

- MVC – Model-View-Controller,
- MVP – Model-View-Presenter,
- MVVM – Model-View-ViewModel.

Pro můj projekt jsem si zvolil poslední možnost – MVVM.

2.4.1 MVVM

MVVM se skládá z následujících vrstev:

- Model – představuje datovou a logickou vrstvu aplikace, komunikuje s ViewModelem.
- View – představuje prezenční vrstvu aplikace, komunikuje s ViewModelem.
- ViewModel – prostředník mezi vrstvami Model a View.

Architektura MVVM je obzvláště výhodná pro vývoj v Androidu díky kolekci knihoven, které jsou dostupné pod souhrnným balíčkem *Android Architecture Components*. Ta obsahuje dvě třídy, jež zjednodušují komunikaci mezi vrstvami a umožňují jejich nezávislost:

- LiveData – třída navržena pro uchování dat, která vnitřně implementuje *Observer pattern*[7] a je nezávislá na životním cyklu aktivity[8]. [30]
- ViewModel – v tomto případě třída vytvořená za účelem uchování a správy UI dat, která je nezávislá na konfiguračních změnách (podrobněji v podkapitole 3.2). [31]

[7] Návrhový vzor umožňující sledovat změny hodnot.

[8] Klíčová součást aplikace v Androidu, zjednodušeně si ji lze představit jako samostatnou obrazovku, jež uživatel v aplikaci vidí, viz podkapitola 3.1.2.1.

Protože jsem jednotlivé vrstvy popsal velmi stručně, v následujících řádcích je rozebírám podrobněji.

2.4.1.1 Model

Model vystavuje svá data na základě požadavku od ViewModelu. Ty v ideálním případě implementují *Observer pattern*, a tak je možné sledovat jejich změny a reagovat na ně. Tímto přístupem je zajištěna nezávislost na ViewModelu.

2.4.1.2 ViewModel

ViewModel funguje jako most mezi Modelem a View. View předá svůj požadavek ViewModelu, ten ho zpracuje a pošle dál Modelu. Poté sleduje data vystavená Modelem a jakmile dojde ke změně, reaguje na ni a odpoví zpět View.

2.4.1.3 View

View předává své požadavky ViewModelu a čeká, až mu odpoví. Po získané odpovědi aktualizuje zobrazená data.

2.5 Vybrané technologie

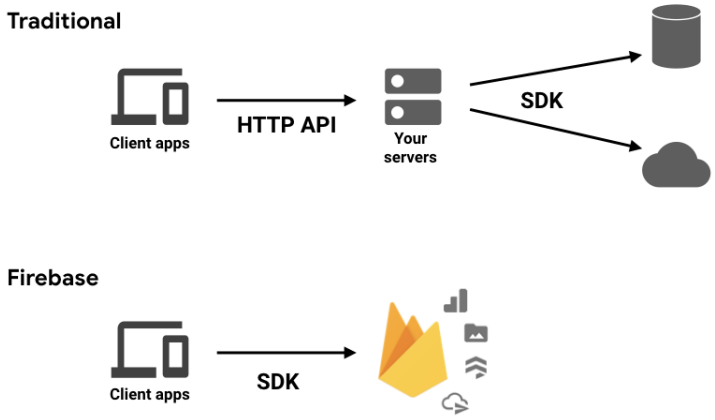
Logiku mojí aplikace jsem se rozhodl postavit na základech platformy Firebase, která nabízí vývojářům celou řadu služeb zjednodušující vývoj mobilních a webových aplikací. Na jednom místě se k dispozici nachází nástroje pro tvorbu databáze, cloudové úložiště, autentifikaci, posílání notifikací, analýzu projektu, testování a mnoho dalších. Jednotlivé služby lze používat zvlášť nebo zkombinovat.

Díky skvěle navrženému API, perfektní dokumentaci a množstvím nabízených služeb má vývojář možnost se plně soustředit na vývoj vlastní aplikace. Nemusí se tak starat o implementaci backendu na serverové části. Pro malé projekty[9] jsou navíc všechny služby zadarmo. Nicméně, je zde třeba brát na vědomí, že s rostoucí uživatelskou základnou vyvíjené aplikace si později Firebase nechá značně zaplatit. [32]

2.6 Návrh databáze

V této podkapitole postupně rozebírám typ použité databáze (2.6.1) a technologii (2.6.2), kterou jsem si pro tento účel vybral. Na konci této části rovněž prezentuji navržený databázový model (2.6.3).

[9]Projekty využívající tzv. *Spark* plán od Firebase, viz `https://bit.ly/2VBBUzF`.



Obrázek 2.3: Rozdíl mezi tradičným vývojem a vývojem s Firebase [33]

2.6.1 Typ použité databáze

Pro svůj projekt jsem se rozhodl použít dokumentovou NoSQL (zkratka nemá přesnou definici, neznamená ovšem *no to SQL*[10] ani *not only SQL* [34, s. 21]) databázi. Jak je již z názvu patrné, základ tvoří dokumenty, které se dají popsat jako strukturovaná sada dvojic klíč-hodnota. Pro každý klíč navíc platí, že je unikátní. Data jsou pak typicky uložena ve formátu JSON[11].

Svojí strukturou umožňují NoSQL databáze ukládat velké množství dat, mezi kterými můžeme velice rychle vyhledávat. Na úkor rychlého vyhledávání jsou ovšem dotazy značně omezené. Mimo to můžeme ocenit vysokou míru škálovatelnosti. Ta je obzvlášť vhodná, pokud si nejsme dopředu jistí databázovým schématem. Rozšířit či upravit databázi je díky tomu snadné i během vývoje.

2.6.2 Cloud Firestore

Cloud Firestore představuje vysoce flexibilní realtime NoSQL databázi, která běží v cloudu[12]. [35] Tvoří jeden ze základních pilířů platformy Firebase. Skvěle se doplňuje s další službou od Firebase, Firebase Authentication.

Vnitřní strukturu databáze tvoří dokumenty, jež jsou uspořádány do kolekcí. Kupříkladu můžeme mít v databázi uloženou kolekci Lidé a v nich dokumenty reprezentující konkrétní instance osob – Tomáše, Honzu nebo Aničku. Jednotlivé dokumenty mohou obsahovat další kolekce. Tímto přístupem může vývojář vytvořit komplexní datovou strukturu podle osobních požadavků.

[10] Structured Query Language – dotazovací jazyk pro práci v relačních databázích.

[11] JavaScript Object Notation – formát dat nezávislý na platformě, založen na seznamu párů název-hodnota.

[12] Cloud – data a programy na internetu, ke kterým můžeme vzdáleně přistupovat.

Ačkoliv obecně v NoSQL databázích nemáme možnost se tak dobře dotazovat jako v relačních databázích, podařilo se autorům Cloud Firestore odvést dobrou práci a vymyslet systém umožňující skládat a kombinovat vícero dotazů včetně řazení. Limity zde ovšem stále najdeme.

Cloud Firestore navíc automaticky vytváří index pro každou položku v dokumentu. Ten označuje dokument, ve kterém se daná položka nachází. Jednotlivé položky jsou pak spolu s indexy uspořádány v seřazeném seznamu. Díky tomu mohou být dotazy prováděny velmi rychle. [36]

Mezi největší pozitivum této databáze pak patří vysoká míra flexibility. Měnit a upravovat položky v databázi lze velmi snadno i během vývoje. To se zejména hodí, pokud v průběhu implementace dostaneme nový nápad a chceme jej uvést do praxe.

Za další výhodu můžeme označit synchronizaci dat v reálném čase. Změna v klientské části aplikace se okamžitě zpropaguje do cloudového úložiště. Změny je rovněž možné v reálném čase odebírat (přijímat), a tak lze aktualizovat data v aplikaci téměř okamžitě.

Jako poslední klad bych uvedl dočasnou podporu uložení dat offline. Cloud Firestore ukládá často přistupovaná data na cache v paměti zařízení. Pokud se zařízení odpojí od internetu, jsou operace CRUD[13] stále dostupné. Po připojení zařízení zpátky k internetu dojde k automatické synchronizaci se serverem. [35] Velikost cache paměti je implicitně nastavena na 100 MB. [37]

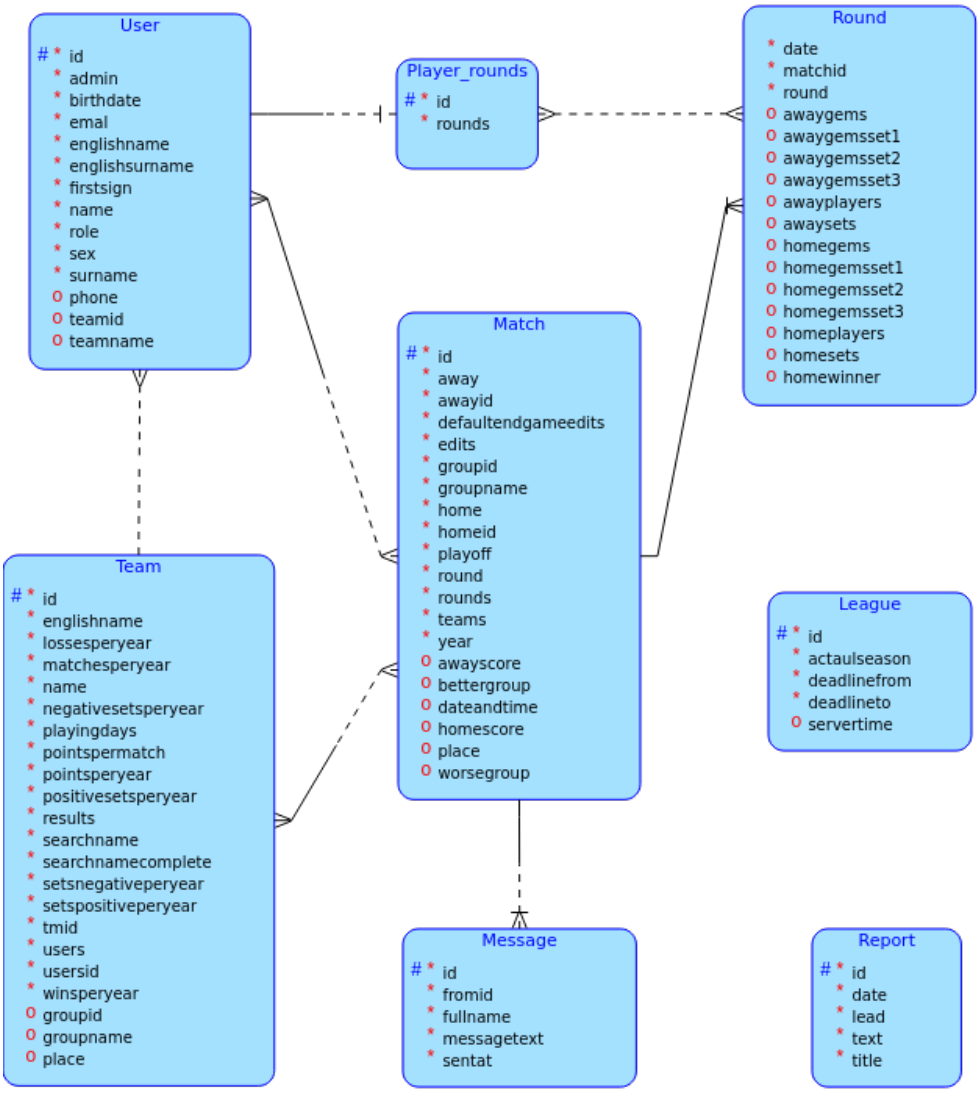
2.6.3 Databázový model

Databázi jsem rozvrhl do sedmi základních dokumentů. Jmenovitě jde o *Group* (Skupina), *League* (Liga), *Match* (Utkání), *Message* (Zpráva), *Report* (Článek), *Team* (Tým) a *User* (Uživatel). Každý dokument pak většinou odkazuje na ostatní. Fakticky:

- *Group* obsahuje seznam týmů dle jednotlivých sezón.
- *Match* odkazuje na týmy a uživatele, kteří v zápasech daného utkání hráli.
- *Message* identifikuje svého odesílatele (uživatele).
- *User* má identifikátor přiřazeného týmu.
- *Team* identifikuje seznam vlastních uživatelů.

Mimo to existuje entita *Round*, která je uložená jako hodnota v dokumentech *Match* a *PlayerRounds*. Dále dokument *Match* obsahuje kolekci zpráv. Konkrétní zpráva je již zmíněný dokument *Message*. V poslední řadě dokument *User* obsahuje kolekci o jednom dokumentu. Tím dokumentem je *PlayerRounds*. Schéma popisuje obrázek 2.4.

[13] Create, Read, Update, Delete – tvorba, čtení, editace a mazání dat v databázi.



Obrázek 2.4: Schéma databáze

Zkušené oko si může povšimnout, že některé informace jsou v databázovém modelu uvedeny vícekrát. Tento přístup je u NoSQL databází běžný. Umožňuje vytvářet složitější dotazy a snížit jejich počet. V případě Cloud Firestore navíc vývojář ušetří peníze, protože po překročení výchozího limitu se za množství CRUD operací platí. Nevýhodou je ale nutnost aktualizovat všechna redundantní data.

Zejména u *Team* lze vidět množství položek, které by bylo vhodnější umístit do samostatného dokumentu. Momentální řešení je prozatimní a rád bych jej v budoucnu zlepšil. Důvodem současného stavu jest fakt, že až v průbě-

hu vývoje jsem se dozvěděl o 1MB limitu dokumentu v Cloud Firestore. Na druhou stranu, data jsou k dispozici bez nutnosti vytvářet nový dotaz. Nicméně, z objektového hlediska to není pěkné.

2.7 Autentizace

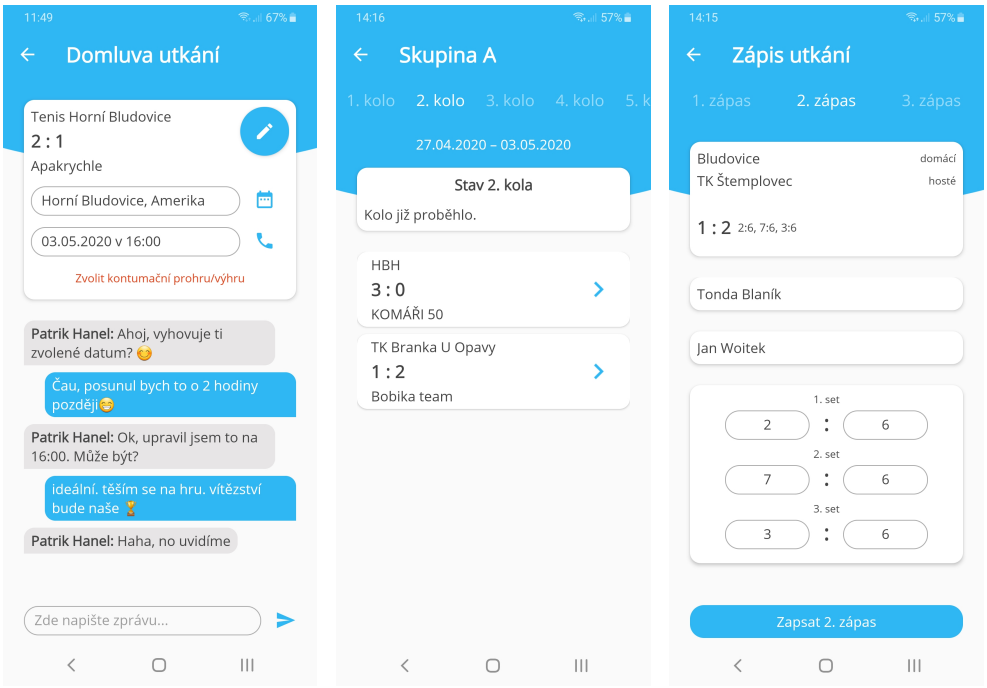
Aby se mohli vedoucí dostat do své administrátorské sekce, musí se v aplikaci prvně přihlásit. Tím bude zajištěna bezpečnost před nechtěnými zásahy do organizace ligy od běžných uživatelů. Na základě přihlášení pak bude uživatel rozlišen podle role, jaká je mu v soutěži přiřazena.

V současné podobě soutěže jsou hráči identifikování emailem. Z důvodu zachování stejného konceptu proto bude přihlášení v aplikaci probíhat pomocí kombinace emailu a hesla. K tomu jsem si vybral službu Firebase Authentication.

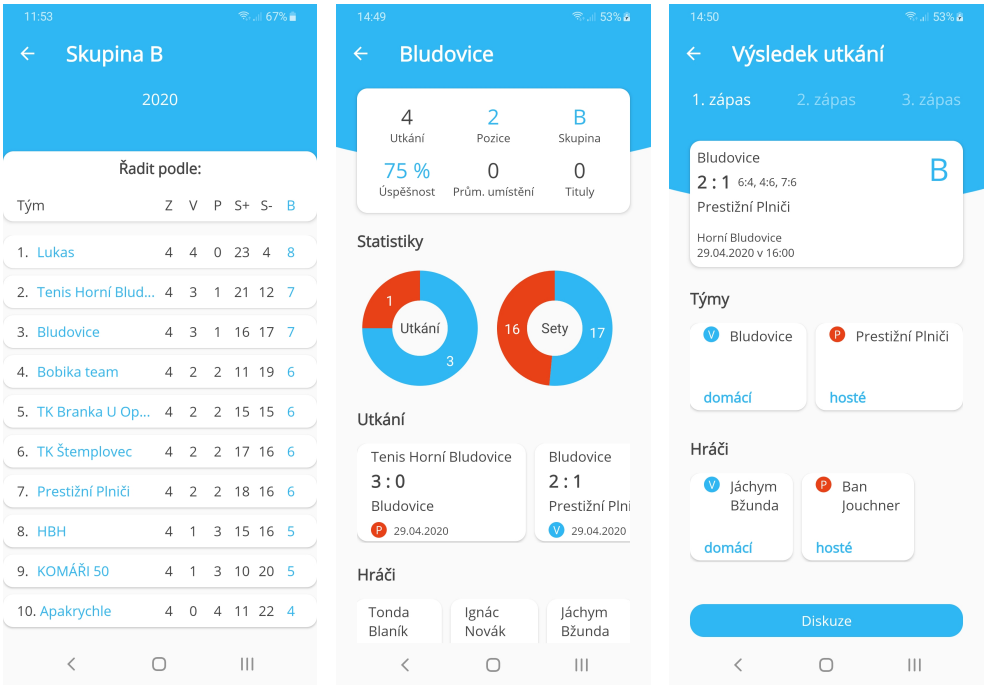
2.7.1 Firebase Authentication

Pokud je aplikace komplexnější a vyžaduje přihlášení, musí tradičně vývojáři navrhnout a sestavit vlastní server, který řeší autentizaci a autorizaci uživatelů. Takový úkol je kritickou částí úspěchu produktu. Vymyslet vše správně a zajistit bezpečný chod je často velmi obtížné. Jestli se chce vývojář této části vyhnout, má možnost použít službu Firebase Authentication.

Firebase Authentication je další z mnoha služeb poskytována od Firebase. Mimo email umožňuje přihlášení pomocí účtů třetích stran (Google, Facebook, Twitter, Github a další). Jakmile uživatel zadá své přihlašovací údaje, ověření proběhne interně na serveru Firebase. Ten potom odpoví klientovi. Vývojář se tak nemusí starat o logiku ani bezpečnost. Firebase Authentication navíc úzce spolupracuje s dostupnými databázemi od Firebase. V nich může autor projektu definovat bezpečnostní pravidla s využitím autorizačního tokenu. [38]



Obrázek 2.5: Snímky výsledné aplikace znázorňující domluvu utkání, konkrétní kolo v přehledu utkání a zápis utkání



Obrázek 2.6: Snímky výsledné aplikace znázorňující pořadí týmů ve skupině, profil týmu a výsledek utkání

Kapitola 3

Implementace

V této kapitole popisuji historii a základní stavební prvky Androidu (3.1). Dále zde řeším způsob zachování dat během konfiguračních změn (3.2) a komunikaci se vzdálenou databází (3.3). V neposlední řadě se zabývám použití View Binding (3.4) a RecyclerView s Adaptéry (3.5). Na konci této části pak také rozebírám zajímavé body implementace (3.6).

3.1 Úvod do Androidu

Dosud jsem několikrát v této práci zmínil operační systém Android, málo o něm napsal, vysvětlil jenom pár pojmů. Stále jsem však jednoznačně neuvedl, jak vznikl a z jakých komponent se skládá. V následujících řádcích tuto situaci napravuji.

3.1.1 Historie

Před několika lety si lidé pod slovem Android představovali robota podobného člověku. Dnes, po 17 letech vývoje tohoto mobilního operačního systému [39], je stav jiný. Málokomu se v první řadě nevybaví jeho telefon, tablet nebo alespoň roztomilý zelený robot.

Počátek systému spatřil světlo světla v roce 2003. Za ním stála čtveřice autorů – Andy Rubin, Rich Miner, Nick Sears a Chris White. Ti tehdy založili firmu Android Inc., kterou o dva roky později koupila společnost Google. Ta provozuje Android do dnešní doby. [39]

Dva roky po odkoupení společnosti byla pod vedením Andyho Rubina představena první verze Androidu vycházející z linuxového jádra. Spolu s ní bylo za účelem vyvinutí otevřeného standardu pro mobilní zařízení založeno konsorcium Open Handset Alliance. To se dodnes podílí na vývoji a tvoří ji skupina technologických firem v čele s Googlem. [39]

Ve stejném roce získala společnost Google i několik patentů. Veřejnost byla přesvědčena, že se blíží oznámení telefonu s novým operačním systémem.

A skutečně, v říjnu 2008 byl v USA představen první chytrý telefon s Androidem – T-Mobile G1. Spolu s uvedením telefonu byla veřejnosti uvolněna i první verze SDK[14]. [39]

Od onoho prvotního uvedení vydává Android nové verze každý rok. Ve všech dalších verzích jsou pak přidávány nové funkce. Významným milníkem se stala verze 5.0 Lollipop. Ta kompletně obměnila původní virtuální stroj Dalvik, na němž Android běžel, za nového nástupce Android Runtime (ART). Ten umožnil přechod Androidu z mobilů a tabletů na hodinky a televize. [40]

Mimo to přišla verze 5.0 s novým vizuálním stylem – Material Design. [41] Jak se můžeme na stránkách Material Design dočíst:
„*Material Design je přizpůsobitelný systém směrnic, komponent a nástrojů. Podporuje a doporučuje nejlepší postupy při návrhu uživatelského rozhraní,*“ [42]
Smyslem Material Design je tedy dát aplikacím na různých platformách jednotný vizuální styl.

Až do poslední verze (Android 10) byly všechny verze pojmenovány podle cukrovinek v abecedním pořadí. [43] V současné době se chystá verze Android 11.

Obrázek 3.1: Historie vývoje loga Android. Shora dolů: původní logo, logo z roku 2014 a nynější logo [44]

3.1.2 Základní komponenty

Každá aplikace v Androidu se skládá z několika základních stavebních bloků, tzv. komponent. Ty poskytují aplikaci funkcionalitu a v mnohých případech

[14]Software Development Kit – sada knihoven, nástrojů a dokumentace, umožňující vývoj aplikací.

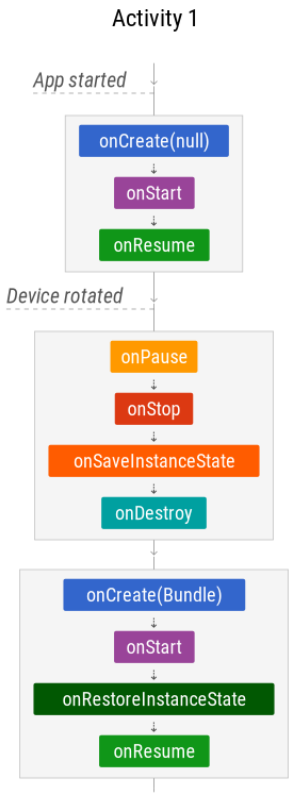
na sobě závisí. [45] Jejich použití je ale čistě na vývojáři. Pokud se pro danou komponentu v aplikaci nenajde využití, pak ji autor může jednoduše vynechat. Všechny použité komponenty je nicméně nutné uvést v souboru *AndroidManifest.xml*. Ten mimo to informuje SDK Build Tools[15], operační systém a Google Play o základních údajích vyvíjené aplikace. [46]

3.1.2.1 Activities

Aktivita představuje v Android aplikaci jednu obrazovku s konkrétním uživatelským rozhraním. Navíc přijímá od uživatele vstup (např. kliknutí na tlačítko) a reaguje na něj. Pro lepší představu – aplikace Kalendář může obsahovat několik aktivit. Jedna aktivita zobrazuje samotný kalendář, druhá umožňuje vložit novou událost a třetí představuje konkrétní událost s podrobnostmi. Ačkoliv jsou jednotlivé aktivity nezávislé na sobě, dohromady poskytují uživateli ucelený zážitek.

Každá aktivita má svůj životní cyklus, který popisuje, v jaké fázi se aktivita nachází. Aby měl uživatel z aplikace dobrý dojem, je třeba naprogramovat aktivitu s ohledem na konkrétní fáze cyklu. Uvedu příklad. Pokud uživatel pomocí domovského tlačítka odejde z aplikace pryč, a poté se po nějaké době opět vrátí, je chtěné, aby aplikace byla ve stejném stavu, jako když odcházel. Jeden z možných postupů, jak tohoto stavu docílit, popisuji podrobněji v podkapitole 3.2.

Již jsem zmínil, že aktivita obsahuje konkrétní uživatelské rozhraní. Stále jsem však neuvedl, jak tvorba takového rozhraní v Androidu probíhá.



Obrázek 3.2: Životní cyklus aktivity během otočení displeje [47]

Veškerá grafika aplikace (grafické prvky, animace, vektorové obrázky atd.) je definována v souborech XML[16]. Vývojář pomocí tohoto jazyku a předem dané syntaxe vytváří layouty. Ty vymezují rozvržení uživatelského rozhraní a jsou složeny z objektů třídy View. [48] Takové objekty se nazývají widgety a v praxi jimi jsou například tlačítka, textová pole, obrázky a podobně.

[15] SDK komponenta pro sestavení Android aplikací.
[16] Extensible Markup Language – rozšiřitelný značkovací jazyk.

3.1.2.2 Services

Services (služby) slouží k vykonávání různých úkolů na pozadí aplikace. Jsou velmi podobné aktivitám, s tím rozdílem, že neposkytují uživatelské rozhraní. [45] Pro představu, hudební přehrávač v telefonu využívá služby pro přehrávání písniček na pozadí. Jiným příkladem může být aplikace, jež potřebuje stáhnout z internetu velké množství dat, aniž by blokovala interakci uživatele v aplikaci.

Služby mohou být spuštěny z jiných komponent, zejména pak z aktivit. Operační systém udržuje jejich běh do té doby, než skončí se svým úkolem (ne vždy toto platí, v některých případech může operační systém ukončit jejich běh ještě před koncem, například během nedostatku operační paměti). [45]

Služby dále rozlišujeme podle toho, jestli vykonávají úkoly ve skrytu před zrakem uživatele, nebo upozorňují uživatele o jejich přítomnosti. Výše zmíněný hudební přehrávač je skvělým příkladem služby, která je pro uživatele viditelná. Synchronizace s databází je naopak službou, o níž uživatel často ani netuší.

```
//BroadcastReceiver, ktery reaguje na vypadek internetu
class NetworkChangeReceiver : BroadcastReceiver()
{
   override fun onReceive(context: Context, intent: Intent)
   {
      //internet neni dostupny
      if(!checkInternet(context))
      {
         //zobraz zpravu uzivateli o vypadku internetu
         Toast.makeText(context, "No internet, sorry!",
             Toast.LENGTH_SHOW).show()
      }
   }

   fun checkInternet(context: Context): Boolean
   {
      //zkontroluj spojeni s internetem, pokud je spojeni dotupne,
          vrat true, jinak vrat false
   }
}
```

Ukázka kódu 3.1: Ukázka BroadcastReceiveru v praxi

3.1.2.3 Broadcast receivers

Broadcast receiver je komponenta, která umožňuje reagovat na oznámení jiných aplikací nebo systému (ve skutečnosti může reagovat i na oznámení aplikace, ve kterém se broadcast receiver nachází). Zároveň můžeme vytvořit vlastní událost, tu předat systému, a ten již sám upozorní broadcast receivery

ostatních aplikací. Systém je navíc schopen dodat události aplikacím, které v daný moment neběží. [45]

Jeden příklad za všechny. Vyvíjíme aplikaci, u níž je kritické spojení k internetu. Pokud je spojení přerušeno, musí na to aplikace adekvátním způsobem zareagovat (např. zobrazit zprávu). V takovém případě je rozumné naimplementovat třídu využívající broadcast receiver, která vykoná patřičnou akci, pokud obdrží zprávu o změně dostupnosti sítě (viz ukázka kódu 3.1).

3.1.2.4 Content providers

Poslední základní komponentou jsou content providery. Ty slouží ke sdílení dat mezi aplikacemi v zařízení (i sdílení dat v rámci jedné aplikace). Content providery mají jistou podobnost s databázemi. Data z content providerů jsou uspořádána do tabulek a umožňují standardní metody pro práci (čtení, zápis, mazání). [45]

Například sám operační systém Android poskytuje content provider, který spravuje uživatelovy kontakty. Libovolná aplikace, jež k tomu má oprávnění, pak může kontakty získat a nakládat s nimi. [45]

3.1.3 Ostatní

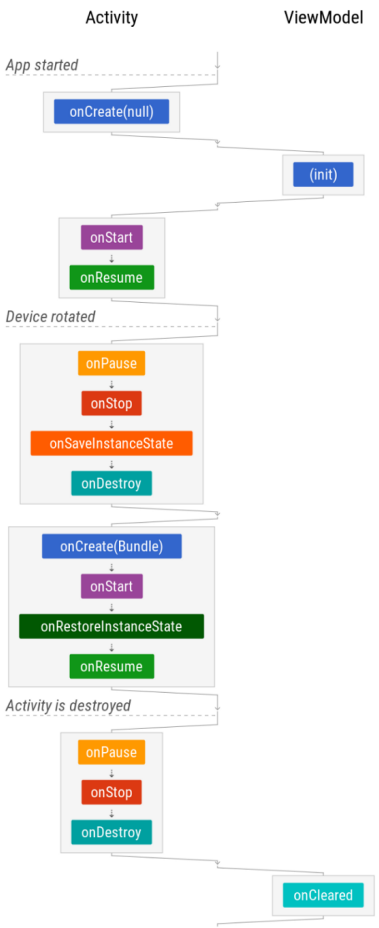
Pro vývoj Android aplikací je dnes oficiálně určeno IDE[17] Android Studio. [49] Součástí Android Studia je i nástroj Gradle, jež slouží k automatickému sestavení (build) aplikace. Mimo to v něm můžeme spouštět testy, vytvářet dokumentaci, exportovat balíčky a doplňovat knihovny, které jsou potřebné pro implementaci projektu. [50] Vše pak probíhá v souboru *build.gradle*, jenž je součástí každé Android aplikace.

3.2 Zachování dat během konfiguračních změn

Již jsem v podkapitole 3.1.2.1 napsal, že je žádoucí, aby vývojář vytvářel aplikaci s ohledem na životní cyklus jednotlivých komponent. Velmi častou chybou je nesprávné ošetření případů, které nastanou během konfiguračních změn v zařízení. Konfiguračními změnami se rozumí například otočení obrazovky nebo změna jazyku v zařízení. Zde totiž nezkušený programátor čelí na první pohled ne tak patrné výzvě. Uvažujme případ otočení obrazovky.

Uživatel spustí libovolnou aktivitu. Jak jsem už uvedl dříve, aktivita se vždy nachází v jednom z mnoha stavů životního cyklu. Každý stav má metodu, kterou může programátor implementovat. Tím popisuje, jak se aplikace ve vybrané fázi cyklu bude chovat. Metody jsou pak volány automaticky systémem (metodám, které jsou zavolány až „ve správných čas“, se říká callback metody).

[17] Integrated Development Environment – vývojové prostředí usnadňující práci vývojářům.



Obrázek 3.3: Srovnání životního cyklu aktivity a viewModelu [51]

První volaná metoda v aktivitách se nazývá *onCreate()*. Ta slouží zejména pro inicializaci widgetů v aktivitě. Následují metody *onStart()* a *onResume()*. Potíž je v tom, že během otočení obrazovky se aktivita zničí (je zavolána metoda *onDestroy()*) a vytvoří znovu (opět zavolána metoda *onCreate()*). Tím dojde v aktivitě ke ztrátě uložených dat. Pokud byla například v dané aktivitě uživatelovi zobrazena stažená data z internetu, pak o ně během znovuvytváření aktivity přijde a je třeba začít stahování nanovo.

Jedním z nejlepších způsobů, jak tomuto nechtěnému stavu zabránit, je přesunout data a logiku z aktivity do ViewModelu – třídy z balíčku Android Jetpack[18]. I to je jeden z důvodů, proč jsem se rozhodl právě pro MVVM architekturu popsané v podkapitole 2.4.1.

Třída ViewModel je navržena tak, aby data v ní umístěná přežila konfigurační změny. [31] Výhodou této třídy je navíc snadná spolupráce s aktivitami (a fragmenty[19]). Jak jde z obrázku 3.3 poznat, ViewModel se nachází pouze v jednom stavu, dokud není zničen spolu s volající aktivitou (nebo fragmentem).

Použití ViewModelu je vhodné zejména s uplatněním třídy LiveData. Jak již víme z podkapitoly 2.4.1, třída LiveData uchovává data a díky implementaci *Observer pattern* můžeme změny uložených dat sledovat. To se nám zejména hodí v aktivitě, v níž konkrétní instanci třídy ViewModel používáme.

V praxi tak aktivita pouze obsluhuje požadavky od uživatele a předává je ViewModelu ke zpracování. Ten, jakmile požadavek zpracuje, uloží změny do objektu třídy LiveData. Aktivita se o změnách pomocí observerů dozví a podle toho aktualizuje uživatelské rozhraní (viz ukázka kódu 3.2). Tento přístup navíc ulehčuje testování a zpřehledňuje zdrojový kód.

[18] Sada nástrojů doporučených pro vývoj v Androidu.
[19] Zjednodušeně řečeno samostatné okno s uživatelským rozhraním v aktivitě.

```
//Aktivita
class MyActivity : AppCompatActivity()
{
   //inicializace viewModelu pomoci Kotlin + KTX
   private val myViewModel by viewModels<MyViewModel>()
   //deklarace textoveho pole
   private lateinit var bookTitle: TextView

   override fun onCreate(savedInstanceState: Bundle?)
   {
      //inicializace textoveho pole
      bookTitle = findViewById(R.id.bookTitle)
      //volam metodu objektu myViewModel
      myViewModel.getNameOfMyFavouriteBook()
      //pozoruji zmeny v LiveData
      myViewModel.nameOfMyFavouriteBook.observe(this) { name ->
         //aktualizuji textove pole
         bookTitle.text = name
      }
   }
}

//ViewModel
class MyViewModel : ViewModel()
{
   //inicializace MutableLiveData, obsahuje data typu String (text),
       data MUZEME menit
   private val _nameOfMyFavouriteBook = MutableLiveData<String>()

   //inicializace LiveData, obsahuje data typu String (text), data
       NEMUZEME menit
   val nameOfMyFavouriteBook: LiveData<String>
      get() = _nameOfMyFavouriteBook

   //metoda, ktera priradi do MutableLiveData nazev me oblibene knihy
   fun getNameOfMyFavouriteBook()
   {
      //zde muzeme napriklad provest nejakou operaci s databazi

      //nastavuje hodnotu dat v MutableLiveData
      _nameOfMyFavouriteBook.value = "The Hitchhiker's Guide to the
          Galaxy"
   }
}
```

Ukázka kódu 3.2: Ukázka komunikace ViewModelu s aktivitou

3.3 Komunikace se vzdálenou databází

Databáze je důležitou součástí mnoha aplikací na mobilním trhu. Některé si vystačí pouze s lokální, tedy databází uloženou v paměti zařízení. Jiné, častokrát komplexnější aplikace, využívají ke svému běhu databázi vzdálenou. Protože i já jsem ve svém projektu vzdáleně přistupoval k databázi, popisuji v této podkapitole postup, jakým jsem tak činil a proč si myslím, že můj přístup byl správný.

3.3.1 Základní pojmy

V části této práce používám odborné pojmy. V této podkapitole je vysvětluji.

3.3.1.1 DAO

Z anglického Data Access Object. Jedná se o objekt nebo rozhraní, které se stará o spojení s databází. Pomocí něj jsme schopni abstrahovat a zapouzdřit veškerý přístup ke zdroji dat. [52]

```
suspend fun <T> find(collectionName: String, field: String, value:
    T?): QuerySnapshot
{
   return Firebase.firestore
      .collection(collectionName)
      .whereEqualTo(field, value)
      .get()
      .await()
}
```

Ukázka kódu 3.3: Ukázka metody v DAO

3.3.1.2 Entita

Entita je objekt, který reprezentuje data v databázi. V případě relačních databází typicky představuje jeden řádek tabulky. [53] V NoSQL databázích pak konkrétní dokument.

3.3.1.3 Repository pattern

Návrhový vzor repository slouží k oddělení perzistentních dat od zbytku aplikace. [54] Jinými slovy, repository (nejčastěji implementován jako singleton[20]) komunikuje s databází, API nebo keší a poskytuje tak jeden vstupní bod pro práci s daty. Ostatní třídy se proto nemusí starat, odkud data pochází.

[20] Návrhový vzor, který zajišťuje pouze jednu instanci dané třídy v celém programu.

3.3.1.4 Coroutines

Coroutiny, nebo také česky korutiny či koprogramy je koncept podobný vláknům umožňující asynchronní programování. Podobně jako vlákna, coroutiny jsou schopné běžet paralélně nebo spolu komunikovat. Narozdíl od vláken mají velice malou režii. Můžeme jich tedy s přehledem vytvořit několik tisíc na jednom zařízení bez starosti o ztrátu výkonu. [55] To by s vlákny moc dobře nešlo.

Tento koncept v dnešní době podporuje stále více moderních jazyků, např. JavaScript, C#, Python nebo také Kotlin. Ještě předtím než vysvětlím, proč jsou coroutiny v Androidu užitečné, odbočím k fungování vláken v Android aplikacích obecně.

Každá Android aplikace běží implicitně v hlavním vlákně (ang. main thread, někdy také nazýváno jako UI thread). To se stará o vykreslování grafiky a obsluhu uživatelových požadavků (kliknutí na tlačítko, zadání textu apod.). [56] Ačkoliv tak může být představa jednoho vlákna pro vývojáře dobře pochopitelná, v praxi má často na svědomí špatný výkon aplikací. Pokud totiž provádíme déle trvající operaci v hlavním vlákně (například komunikace s databází nebo spojení se sítí), hlavní vlákno se stará o běh oné operace a uživatelské rozhraní nemusí fungovat tak, jak by mělo. Navíc zde hrozí, že se uživateli zobrazí hláška „aplikace přestala pracovat“, a to je jistě to poslední, co by chtěl uživatel v daný moment spatřit. Z těchto důvodů je vhodné déle trvající činnosti spouštět v jiných vláknech.

Tím se konečně dostávám zase zpět ke coroutinám. Coroutiny totiž v Androidu umožňují chytře provádět asynchronní operace mimo hlavní vlákno, a přesto s hlavním vláknem snadno komunikovat. Navíc se díky své syntaxi v Kotlinu tváří jako synchronní kód, který je mnohem čitelnější.

3.3.2 Řešení

Optimálním řešením je rozvrhnout program do více nezávislých částí. Datovou vrstvu poskládat z entit, objektů DAO a několik repository. Každé repository pak nechat obsluhovat konkrétní entitní třídy[21]. Tedy přijímat požadavky z jiných tříd v aplikaci, řešit je a odpovídat na ně.

Cloud Firestore poskytuje vývojářům metody a třídy, jež samy vykonávají asynchronní operace s databází. To je v dnešní době běžné i u jiných knihoven (např. ROOM nebo retrofit). Protože nejsme dopředu schopni říct, jak dlouho daná operace potrvá, je nutné vymyslet způsob, který čeká na odpověď a přesto neblokuje hlavní vlákno. Možnosti jsou dvě.

První možnost, dříve hojně využívána, je použít vlastní callback metody. Jinými slovy vytvořit rozhraní (interface), jejichž metody jsou volány až po skončení operace a implementovány v místě, kde chceme s výsledkem operací pracovat. Takový přístup ovšem výrazným způsobem snižuje čitelnost

[21] Entitní třída odpovídá tabulce v relačních databázích, v případě Cloud Firestore kolekci.

zdrojového kódu. Mezi programátory se mu ostatně přezdívá *callback hell* – callbackové peklo.

Druhou eventualitou je využít asynchronní nebo reaktivní programování. Reprezentantem reaktivního programování může být kupříkladu knihovna RxJava, která funguje podobně jako návrhový vzor *Observer pattern*. Asynchronní programování pak představuje v Androidu Kotlin se svými coroutinami.

Protože je práce s coroutinami v Androidu preferovaná oproti knihovně RxJava, využil jsem pro svůj projekt právě je.

Budu konkrétnější. Většinu metod v mých DAO jsem implementoval jako tzv. suspend metody (viz ukázka kódu 3.3). Slovíčko suspend zde pouze říká, že běh dané metody může nějakou dobu trvat. Repository pak metody volají a pomocí dispečerů (dispatchers) informují systém, že se jedná o vstupně-výstupní operace (práce se sítí). Systém se tak může lépe rozhodnout, které vlákno použije pro danou coroutinu. [57]

```
class MyViewModel : ViewModel()
{
   private val _user = MutableLiveData<User>()
   val user: LiveData<User> = _user

   ...

   fun getUser(userId: String)
   {
      //spousteni coroutiny pomoci viewModelScope
      viewModelScope.launch {
         //najdi uzivatele s ID userID
         UserRepository.findUser(userId)?.let { user ->
            //tento radek se vykona pouze pokud neni nalezeny uzivatel
                rovny hodnote null
            _user.value = user
         }
      }
   }
}
```

Ukázka kódu 3.4: Spouštění coroutiny pomocí viewModelScope

Komunikaci s repository jsem většinou přenechal ViewModelu. Ten je pro práci s coroutinami přímo ideální díky *ViewModelScope*. Každá coroutina totiž pracuje ve vlastním rozsahu (scope), jenž říká, po jakou dobu platnosti bude coroutina existovat. [55] *ViewModelScope* tak definuje rozsah platnosti v něm spuštěných coroutin. Jakmile je ViewModel zničen, spolu s ním zaniknou i spuštěné coroutiny. [57]

ViewModel tedy v mém případě spouští coroutiny, předává žádosti repo-

sitory a čeká na odpověď. V momentě, kdy je odpověď vrácena, ji nějakým způsobem zpracuje a aktualizuje LiveData (lépe řečeno MutableLiveData). Ty poté informují aktivitu (nebo fragment) o změně. Aby se uživatel dozvědel o průběhu provádění operace, informoval jsem jej ve většině situací pomocí dialogu s ProgressBarem[22].

V případě dlouho trvajících operací, které jsem potřeboval úspěšně dokončit i po odchodu uživatele z aplikace, jsem spouštěl coroutiny ve službách (services). Ve většině situací jsem pak uživatele informoval o práci na pozadí pomocí notifikací.

Můj přístup rozdělil aplikaci do několika nezávislých celků, kde každá část má svojí zodpovědnost. Díky tomu jsem dosáhl více udržitelného a přehlednějšího kódu.

3.4 View Binding

V pozdější části tvorby mojí aplikace jsem si také vyzkoušel novou funkci v Androidu nazvanou View Binding. Ta byla poprvé přidána do Android Studia 3.6 a ulehčuje psaní kódu, jenž interaguje s objekty třídy View. Android Studio pro každý layout vygeneruje samostatnou třídu, která odkazuje na jednotlivé widgety a jejich ID. Tím je nejen zaručena *null-safety*[23] a *type-safety*[24], ale i rychlejší kompilace. [58] View Binding pak také snižuje množství kódu, které programátor musí napsat.

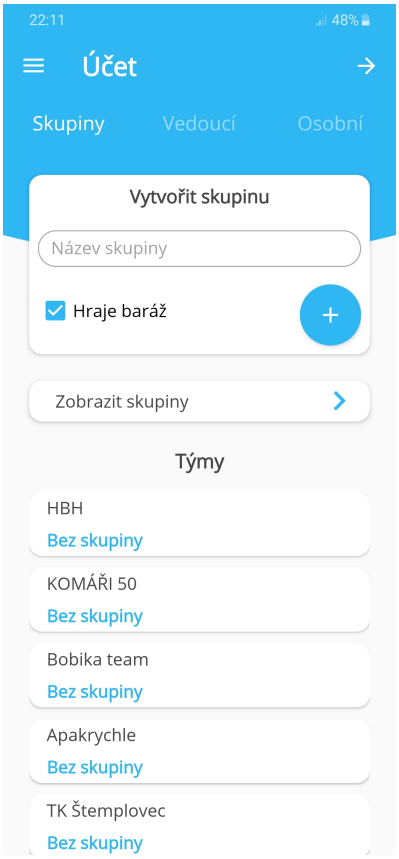
```
//kod s pouzitim View Binding
private lateinit var binding: ILikeTrainsBinding
override fun onCreate(savedInstanceState: Bundle)
{
   super.onCreate(savedInstanceState)
   binding = ILikeTrainsBinding.inflate(layoutInflater)
   setContentView(binding.root)
   binding.name.text = "Jakub"
}

//kod bez pouziti View Binding
private lateinit var name: TextView
override fun onCreate(savedInstanceState: Bundle)
{
   super.onCreate(savedInstanceState)
   setContentView(R.layout.i_like_trains)
   name = findViewById(R.id.name)
   name.text = "Jakub"
}
```

Ukázka kódu 3.5: Porovnání kódu s a bez View Binding

[22] Indikátor stavu načtení.
[23] Zabránění vyhození výjimky *NullPointerException*.
[24] Každý objekt má jasně specifikovaný typ.

3.5 RecyclerView a Adaptéry

RecyclerView je třída zodpovědná za zobrazení po sobě jdoucích grafických prvků uspořádaných do seznamu. [59] Setkáme se tak s ní skoro ve všech aplikacích. Například aplikace Gmail používá RecyclerView pro znázornění příchozích/odchozích emailů. Výhoda RecyclerView spočívá v tom, že se jednotlivé prvky načítají až těsně předtím, než jsou zobrazeny. Poté, co zmizí z obrazovky, jsou uvolněny z paměti. Díky tomu může RecyclerView obsahovat velké množství dat bez obav na plynulost.



Obrázek 3.4: Seznam týmů je implementován pomocí RecyclerView

Adaptér můžeme popsat jako třídu spojující zdroj dat s UI komponentami, které daná data zobrazují. [60] V případě RecyclerView tak adaptér definuje, jak se mají data v něm znázornit.

V mé aplikaci jsem využil RecyclerView spolu s adaptérem mnohokrát. Pokud mám být přesný, tak při zobrazení týmů, hráčů, skupin, utkání, zpravodajů a zpráv.

Adaptéry jsem rozdělil do tří kategorií. První kategorie tvoří obyčejné adaptéry, jež přijímají nějakou množinu dat na vstupu a charakterizují jejich použití v RecyclerView. Zbylé dvě kategorie jsou zajímavější.

Druhou kategorii jsem interně pojmenoval jako realtimové adaptéry. Adaptéry z této kategorie totiž v reálném čase reagují na změny v Cloud Firestore databázi a tyto změny okamžitě propagují do RecyclerView. Firebase naštěstí poskytuje open-source knihovnu FirebaseUI, která nejen tuto funkcionalitu značně ulehčuje. Konkrétně jsem tak pro modifikaci dat v reálném čase využil třídu FirestoreRecyclerAdapter.

Realtimové adaptéry pak používám k zobrazení zpráv v chatu, jenž slouží pro společnou domluvu vedoucích týmů před utkáním, k zobrazení výsledků utkání v přehledu utkání a k zobrazení aktuálního seznamu týmů, které vedoucí soutěže spravuje ve svém rozhraní.

Ačkoliv třída FirestoreRecyclerAdapter interně načte všechna dotazovaná data z Cloud Firestore najednou, můžu si tento přístup dovolit. Množství dat

totiž ve všech třech případech užití nebude větší než pár desítek (výjimkou může být zmíněný chat, nicméně dle rozhovoru s panem Vaňkem [61] neočekávám, že by měl být vedoucími tak hojně využíván).

Poslední kategorií jsou adaptéry, které načítají data postupně po stránkách. To je vhodné zejména u dat, kterých je velké množství nebo ke kterým budou uživatelé přistupovat často. K implementaci jsem proto využil třídu FirestorePagingAdapter z již zmíněné knihovny FirebaseUI. Tento druh adaptéru pak používám k zobrazení všech hráčů v lize, k zobrazení všech ligových týmů v sekci dostupné i pro nepřihlášené uživatele, k zobrazení seznamu týmových utkání v profilu týmu, k zobrazení všech zpravodajů a v poslední řadě k zobrazení zpráv v hromadné diskusi pod utkáním. Aby měl uživatel z procházení dat lepší dojem, je během načítání nových stránek zobrazen ProgressBar. Pokud uživatel dospěje na konec seznamu, zobrazí se mu Toast[25] s informací, že již vše bylo načteno.

3.6 Zajímavé body implementace

V této podkapitole jsem si dovolil vybrat problémy, které mi přišly během implementace nejzajímavější. Problém vždy nejprve definuji a poté nastíním způsob, jakým jsem jej řešil. Rád bych také dodal, že ačkoliv názvy zde vybraných metod jsou přesné, neberou v potaz parametry. Například místo *fairyTale(long: Person, stout: Person, sharpEyes: Person)* tak uvádím pouze *fairyTale()*.

3.6.1 Generování utkání

Popis problému Ve všech skupinách se střetne každý tým s každým právě jednou. Počtu týmů ve skupině odpovídá patřičný počet kol. Vedoucí soutěže se může rozhodnout počet kol navýšit, a to i při ponechání stejného množství týmů. Kvantum domácích a venkovních utkání by měl být pro jakýkoliv tým co nejvíce vyrovnaný.

Řešení problému Pro vyřešení tohoto problému jsem využil *Scheduling* algoritmus (viz názorná ukázka 3.6). Ten funguje tak, že jednotlivé týmy rozdělí na polovinu do dvou řad. Pokud je počet týmů lichý, pak je nutné mezi týmy přidat fiktivní tým. Jakmile libovolný tým hraje s fiktivním protihráčem, potom takové utkání není zařazeno do programu soutěže a tým má pro dané kolo volno.

První tým v první řadě zůstává po celou dobu výkonu algoritmu na svém místě. Ostatní týmy si v každém kole vymění pozice. Týmy z horní řady se každé kolo posunou o jednu pozici směrem doprava. Tým na konci horní řady (nejvíce vpravo) je pak umístěn na konec spodní řady. Naopak týmy

[25] Plovoucí text s krátkou informací, jenž po brzké době zmizí.

z dolní řady se každé kolo posouvají doleva. První tým (nejvíce vlevo) je potom přesunut do horní řady na druhou pozici (hned za fixní tým). Tento postup je opakován dle počtu kol.

Popsaný algoritmus funguje dobře, ovšem nebere v úvahu optimální poměr domácích a venkovních zápasů. To jsem vyřešil tak, že jsem každému týmu přiřadil unikátní číslo z množiny 1–n, kde n se rovná celkovému počtu týmů ve skupině. Pokud je součet dvou těchto hodnot ve vygenerovaném utkání kladný, potom tým z horní řady hraje jako domácí. Zmíněný postup sice neřeší pravidelné střídání venkovních a domácích utkání pro každý tým, ale zachovává optimální poměr, což je v konečném důsledku důležitější. V praxi se pak stejně jedná pouze o organizační záležitost, neboť mnoho týmů nemá svůj domovský kurt a soutěží na veřejných kurtech v okrese.

```
//kazdy tym je reprezentovan cislem

//prvni kolo
1 2 3 4
8 7 6 5

//druhe kolo
1 8 2 3
7 6 5 4

//treti kolo
1 7 8 2
6 5 4 3

...
```

Ukázka kódu 3.6: Ukázka fungování *Scheduling* algoritmu

Jak jsem již zmínil, vedoucí soutěže může počet kol navýšit. V tom případě je mezi týmy přidáno větší množství fiktivních týmů. V důsledku pak má více reálných týmů v každém kole volno.

Proces generování utkání a následné přidání do vzdálené databáze spouštím z časových důvodů ve službě *GenerateScheduleService*. Vedoucímu soutěže se během tohoto děje zobrazí notifikace.

Vedoucí soutěže má rovněž možnost utkání ve skupině pro danou sezónu přegenerovat. Během přegenerování jsou již dříve vygenerovaná utkání smazána a celý dříve popsaný postup se opakuje.

Veškerá logika tohoto problému je zapouzdřena ve třídě *RobinRoundTournament*. Ta poskytuje veřejné metody *setTeams()*, *setRounds()* a *createMatches()*. Pomocí prvních dvou je možné nastavit týmy a počet kol. Metoda *createMatches()* pak implementuje popsaný algoritmus.

3.6.2 Domluva utkání

Popis problému Vedoucí týmů libovolného utkání se před jeho zahájením musí domluvit na organizačních záležitostech.

Řešení problému Domluvou utkání se zabývá aktivita implementovaná ve třídě *MatchArrangementActivity*. Ta poskytuje nejen informace o místě a datu konání, ale i prostor pro konverzaci mezi vedoucími. Místo konání je doporučeno podle kurtu domácího týmu. Vygenerování doporučeného data konání má ale svůj kratičký prolog. Začnu malou oklikou.

Každé kolo má přiřazeno týden, během kterého jsou utkání doporučena odehrát. Číslo týdne stanovuje vedoucí soutěže. Během prvního spuštění aktivity s domluvou utkání je doporučené datum zvoleno metodou *DateUtil.findDate()*. Volba probíhá na základě několika podmínek.

V první řadě je bráno v potaz číslo týdne, ve kterém se dané utkání hraje. Pro zpřesnění se poté metoda *findDate()* „podívá“ na hrací dny obou týmů. Pokud je nalezena shoda, pak je datum utkání stanoveno na daný den. V opačném případě doporučené datum konání koresponduje s hracím dnem domácího týmu.[26]

Místo i datum konání lze změnit. Po jejich změně je vedoucímu protějšího týmu zaslán email. Rovněž lze díky intentům[27] telefonicky kontaktovat soupeře a utkání uložit do kalendáře.

Vedoucí týmů si navíc můžou přímo v aktivitě spolu psát. Pro tento účel jsem použil adaptér *ShowMessagesFirestoreAdapter*, jenž v reálném čase reaguje na změny v databázi (viz předešlá podkapitola 3.5). Adaptér také v metodě *getItemViewType()* rozlišuje, zdali je autor zprávy odesílatel nebo příjemce. Podle toho pak každé zprávě přiřadí správný layout.

3.6.3 Zadávání výsledků

Popis problému Všechna utkání se skládají ze tří samostatných zápasů. Každý zápas se hraje na dva vítězné sety. Výsledky utkání zapisují vedoucí domácích týmů. Vedoucí venkovních týmů můžou s výsledkem nesouhlasit a podat stížnost.

Řešení problému Pro větší přehlednost jsem použil ViewPager[28] a každý zápas umístil do samostatného fragmentu.

[26] V průběhu psaní těchto řádků jsem si uvědomil, že datum by mohlo být vytvořeno již během generování utkání nebo při změně hracího týdne. Tuto připomínku si tedy zapisuji do seznamu úkolů a rád bych se jí věnoval v budoucnu.

[27] Třída popisující záměr, který chceme, aby Android framework provedl.

[28] Widget umožňující rozdělit obsah do více samostatných obrazovek, mezi kterými je možné se přesouvat.

V případě zadání regulérního výsledku jsou data po potvrzení od uživatele poslána do vzdálené databáze a konkrétní zápas přidán do profilu patřičným hráčům.

V ten moment je také spuštěna služba *CountMatchScoreService* aktualizující týmové statistiky prostřednictvím třídy *MatchScoreCounter*. Ta dopředu vypočítá poměr vyhraných a prohraných utkání nejen pro aktuální sezónu, ale i napříč všemi ročníky. Také modifikuje celkový poměr získaných, respektive ztracených setů. Díky tomu jsou v profilech týmů statistické údaje ihned k dispozici.

V pozadí aplikace je pak odeslán automatický email informující vedoucího soutěže a vedoucího protějšího mužstva o konečném stavu utkání.

Zároveň je vedoucímu domácího týmu umožněna jedna oprava každého zápasu. Pokud opravu zápasu vyčerpá a bude chtít výsledek změnit potřetí, musí požádat vedoucího soutěže o zápis výsledku z jeho strany.

Rovněž může domácí vedoucí zadat kontumačního výherce/poraženého. V tom případě se výsledek utkání vyplní automaticky a je ihned uložen do databáze. Odchozí email pak o této činnosti informuje příslušné osoby.

Podat stížnost může pouze vedoucí venkovního týmu. K tomu mu slouží speciální tlačítko, po jehož kliknutí je správnost výsledku zkontrolována a následně emailem zaslána vedoucímu soutěže k posouzení.

Veškerá výše popsaná funkcionalita je volána z tříd *MatchInputResultFragment* a *MatchInputResultFragmentVM*. Pro rozesílání emailů na pozadí slouží třída *EmailSender*.

3.6.4 Vyhledávání a řazení hráčů/týmů

Popis problému Z důvodu velkého množství účastníků ligy a vysokého počtu přihlášených týmů je nepraktické ručně vyhledávat konkrétního hráče, respektive tým. Pro lepší přehlednost je rovněž vhodné umožnit uživatelům řadit hráče, resp. týmy podle zadaných parametrů.

Řešení problému Při řešení problému s vyhledáváním jsem se potýkal s omezením ze strany databáze Cloud Firestore. Ta totiž sama neumožňuje fulltextové vyhledávání, a tak doporučuje použití externí služby Elasticsearch dostupné na platformě Google Cloud. Bohužel je služba Elasticsearch placená (od 16 USD měsíčně [62]), proto jsem se rozhodl naimplementovat vlastní vyhledávač s použitím veřejných metod z Cloud Firestore API.

Označení „vlastní vyhledávač“ je možná příliš troufalý termín. Ve skutečnosti jsem pouze vytvořil metodu *retrieveByPrefix()* v rozhraní *DAO*, která je schopna z databáze vrátit dokumenty začínající libovolným textem. Ten je předaný jako argument zmíněné metody.

V dalším kroku jsem se pokusil uživatelům usnadnit vyhledávání bez diakritiky a rozdílu malých/velkých písmen. Do databáze tedy ukládám samostatnou položku představující již takto upravený název.

Mnou popsaný postup se dá akceptovat u vyhledávání hráčů. Nicméně v případě týmů je stále lehce nepraktický. Z analýzy názvů týmů hrajících v *AMTELu* jsem totiž došel k (ne moc překvapivému) zjištění, že velký počet z nich začíná jednou ze zkratek „TK“, „TJ“, „TC“, „SK“ nebo „SC“. Aby je mohl uživatel během hledání ignorovat, ukládám názvy týmů v databázi i bez jmenovaných akronymů. Shrnuto podtrženo, každý tým v Cloud Firestore obsahuje dvě položky, jednu s kompletním názvem týmu, druhou pak bez zkratky.

Během vyhledávání zkoumám uživatelem postupně zapsané znaky. Ty analyzuje metoda *doCompleteSearch2()*. Po proběhlé analýze pak *doCompleteSearch2()* informuje metodu *retrieveTeamsByPrefix()*, jež se na základě výsledku rozhodne, podle které položky týmy v databázi vyhledat.

```
companion object
{
   val ACRONYMS = listOf("TK", "TJ", "TC", "SK", "SC")
}

val acronyms = ACRONYMS.map { it.toLowerCase(getDefault()) }
    .toMutableSet()

var longestAcronym = acronyms.maxBy { it.length } ?.length
   private set

...

//pokud je text na 100 % akronym, pak funkce vrati false (neni
    potreba provadet hledani nad kompletnim nazvem tymu), v opacnem
    pripade vrati true
private fun doCompleteSearch2(text: String): Boolean
{
   if(text.length <= longestAcronym ?: 0) return true
   acronyms.forEach {
        if(text.startsWith("$it ", true)) return true
   }
   return false
}
```

Ukázka kódu 3.7: Ukázka použití metody *doCompleteSearch2()*

Pro řazení používám metodu *orderBy()* z Cloud Firestore API. Jelikož si tato metoda neumí poradit s českými znaky, ukládám do databáze i české názvy přepsané do anglické pseudo transkripce (viz ukázka 3.8).

Vyhledané hráče/týmy se stejným jménem je možné mezi sebou dále řadit. Rád bych omezení stejného jména obešel, bohužel se však jedná o záměr autorů Cloud Firestore. Jako možné řešení se nicméně nabízí řadit výsledky v klientské části aplikace. V budoucnu se tím budu ještě zabývat.

```
private fun changeToEnglishTranscription(text: String): String
{
   return text.toLowerCase(Locale.getDefault())
      .replace("č", "czz")
      .replace("ch", "hzz")
      .replace("ď", "dzz")
      .replace("ř", "rzz")
      .replace("š", "szz")
      .replace("ť", "tzz")
      .replace("ž", "zzz")
}
```

Ukázka kódu 3.8: Přepis českých znaků do anglické „transkripce“

3.6.5 Pořadí týmů

Popis problému Týmy jsou v každé sezóně implicitně řazeny podle bodů. Za vítězství jsou přiděleny dva body, za porážku jeden a za kontumační prohru nula. Kritéria pro určování pořadí ve skupině při bodové shodě jsou následující:

- V případě dvou týmů se bere v úvahu vzájemné utkání.
- Ve třech a více týmech je pořadí určené na základě malé tabulky těchto týmů.

Malá tabulka je tvořena vzájemnými utkání týmů v ní hrajících. Prvně se porovnávají vzájemné sety, při shodě gemy. Pokud dva týmy mají stejný počet vítězných setů i gemů, jsou seřazeny podle vzájemného utkání.

Řešení problému Správně vyřešit tento problém byl jeden z nejkomplexnějších úkolů během implementace mého projektu. Veškerou logiku jsem umístil do samostatné třídy *RankingSolver*. Ta zahrnuje několik metod, jejichž význam teď vysvětlím.

Při řazení podle bodů je jako první volána metoda *sortByPoints()*. Ta pouze obaluje další dvě metody, *setPointsAndTeams()* a *setOverallRanking()*. Metoda *setPointsAndTeams()* rozdělí jednotlivé týmy do mapy podle počtu získaných bodů v konkrétní sezóně (sezóna je jedním z parametrů konstruktoru třídy *RankingSolver*). Metoda *setOverallRanking()* je o něco složitější.

Ta prvně seřadí týmy z mapy dle kvanta bodů (sestupně) a poté sleduje, kolik týmů se umístilo stejně. Pokud existuje jenom jeden tým s daným počtem bodů, přidá jej do výsledného seznamu. V opačném případě zavolá metodu *diffTeamsWithSamePoints()*, jejíž úkol je rozřadit týmy se stejným množstvím bodů.

Metoda *diffTeamsWithSamePoints()* zpočátku stáhne z databáze společná utkání. Poté zkontroluje, zdali počet týmů se stejným množstvím bodů odpovídá dvěma. Pokud ano, pak s pomocí metody *compareTwoTeams()* porovná společné utkání obou týmů a určí vítěze. V opačném případě přenechá svojí zodpovědnost metodě *setResults()*, která vypočítá pro každý tým počet získaných setů a gemů. Ta pak týmy i seřadí od nejlepší po nejhorší.

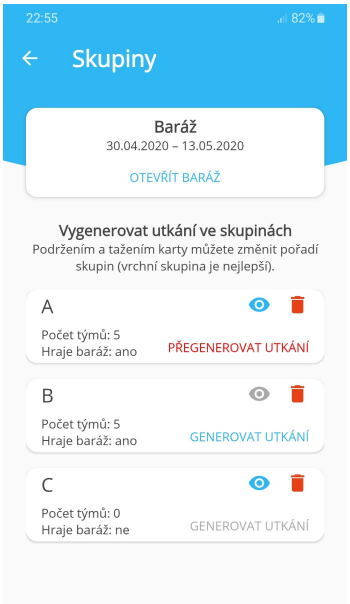
Po ní je zavolána metoda *checkResults()*. Ta reviduje, zdali dva a více týmů v malé tabulce nezískaly stejné množství setů i gemů. Pokud ano, zavolá rekurzivně metodu *diffTeamsWithSamePoints()* a výše popsaný proces se opakuje. Výsledky pak metoda *checkResults()* spojí do jednoho seznamu.

Takový seznam je zároveň i seřazený a deklaruje konečné pořadí týmů se stejným počtem bodů. Dříve jmenovaná metoda *setOverallRanking()* týmy z tohoto seznamu začlení mezi již vyřešené týmy.

Řazení týmů je možné i dle jiných kritérií (výhry, prohry, získané/ztracené sety, počet utkání). Zde již byla naštěstí implementace snadná. Každá instance týmu tyto údaje obsahuje, proto stačilo řadit týmy podle nich.

3.6.6 Tvorba baráže a ukončení sezóny

Popis problému Po konci základní části je vygenerována baráž. V baráži se týmy z druhých míst utkají s předposledními týmy vyšší skupiny o postup, respektive udržení. Vítěz skupiny postupuje do vyšší skupiny, naopak poslední tým v tabulce automaticky sestupuje. Jakmile je baráž odehrána, končí sezóna.



Obrázek 3.5: Vedoucí soutěže může jednoduše změnit pořadí skupin

Řešení problému O ukončení sezóny včetně baráže se stará třída *SeasonFinisher* spuštěná ve službě *SeasonFinisherService*. Třída *SeasonFinisher* v první řadě otevře na dva týdny baráž. Abych byl přesnější, vytvoří skupinu s názvem „Baráž“ a doplňujícími parametry (název „Baráž“ je tedy zarezervovaný, nelze vytvořit takto pojmenovanou skupinu). Poté s pomocí vlastní metody *updateTeamRanks()* aktualizuje všem týmům konečné umístění pro danou sezónu. Konečné umístění slouží jako nutný údaj v týmových statistikách pro zjištění průměrného umístění a počtu získaných titulů napříč všemi ročníky.

Následně je zavolána metoda *transferTeams()*, která tvoří gró celého problému. Ta prostřednictvím metod *transferBottomTeams()*

a *transferTopTeams()* přesune nejhorší a nejlepší týmy na další rok do patřičných skupin. Jak ovšem metody poznají, která skupina je lepší a naopak?

Dovolím si rovnou odpovědět. Každá skupina má přiděleno číslo podle pořadí, v jakém byla vytvořena. To zároveň rozlišuje lepší a horší skupiny. Čím nižší číslo, tím lepší skupina. Vedoucí soutěže ovšem toto pořadí může jednoduše změnit. Stačí, když v sekci *Skupiny* (viz obrázek 3.5) podrží kartu libovolné skupiny a tažením ji přesune na jiné místo. Při odchodu je pak pořadí skupin v databázi aktualizováno dle nového uspořádání.

Zpět ale k ukončení sezóny. Po přesunutí nejlepších/nejhorších týmů přichází na scénu vygenerování utkání v baráži. To se odehrává v metodě *createPlayoffMatches()*.

Metoda *createPlayoffMatches()* projde seřazené skupiny a zkontroluje, zdali má smysl vygenerovat utkání baráže. Smyslem se myslí fakt, kdy krajní skupiny (nejlepší a nejhorší) mají alespoň tři týmy a zbylé skupiny minimálně čtyři. Jestliže je vše v pořádku, vygeneruje utkání tak, jak je popsáno v podkapitole Domluva utkání (3.6.1). Utkání v baráži navíc evidují lepší/horší skupinu, takže třída *MatchScoreCounter* (viz podkapitola 3.6.3) dokáže po odehrání přerozdělit týmy do skupin podle výsledku.

V úplném závěru třída *SeasonFinisher* inkrementuje ročník. Nová sezóna je pak připravena.

Kapitola 4

Testování

V první části této kapitoly (4.1) popisuji způsob, jakým jsem testoval aplikaci v průběhu implementace. V další části (4.2) se pak zaměřuji na uživatelské testování včetně návrhu testovacího scénáře a vyhodnocení zpětné vazby.

4.1 Testování autorem aplikace

V průběhu implementace jsem testoval funkčnost aplikace primárně na telefonu Samsung Galaxy A40 (viz tabulka 4.1). Po každé naprogramované části jsem manuálně zkontroloval, zdali vše funguje tak, jak má. Budu více přesný. V případě spojení se vzdálenou databází jsem ověřoval, jestli byla data do databáze opravdu nahrána. K tomu mi pomohlo i rozhraní v Cloud Firestore, které zvýrazňuje nově vložené, či alespoň upravené nebo smazané dokumenty. Díky tomu jsem tak byl schopen dobře mapovat rozsah změn.

Název	Velikost (palce)	Rozlišení (pixely)	Verze
Samsung Galaxy A40	5,9	2340 x 1080	Android 9.0
Asus Google Nexus 7	7,0	1920 x 1200	Android 6.0
Honor 4C	5,0	1280 x 720	Android 6.0

Tabulka 4.1: Seznam testovaných zařízení

U částí, jež se zabývají složitější implementační logikou, jsem porovnával, zdali se výstup těchto částí shoduje s očekávaných výsledkem. Kupříkladu u generování utkání jsem sledoval, jestli byla všechna utkání vygenerována tak, aby byly dodrženy základní podmínky (každý s každým, optimální poměr domácích/venkovních utkání). V části implementující pořadí týmů jsem zase prvně vložil výsledky utkání tak, aby více týmů mělo stejný počet bodů a já mohl ověřit, že určení pořadí funguje správně i pro malou tabulku. Navíc jsem vše kontroloval výpisem v logu[29], do kterého jsem si zaznamenával jednotlivé

[29]Soubor, který uchovává záznamy o činnosti běhu programu.

dílčí úkoly, jejich úspěchy a neúspěchy. V mnoha případech jsem si také prvně vytvořil zjednodušený testovací soubor, v němž jsem si vyzkoušel implementaci nanečisto.

V neposlední řadě jsem pak procházel napříč všemi obrazovkami v aplikaci a sledoval korektnost interakce individuálních částí. Během dvouměsíčního intenzivního vývoje aplikace jsem tak objevil poměrně velké množství chyb. Ty jsem se poté vždy snažil co nejrychleji opravit. Rovněž jsem během daného období mnohokrát simuloval běh ligy – od zaslání pozvánek vedoucím týmů, přes zadávání výsledků až po vygenerování baráže, ukončení sezóny a zahájení nového ročníku. Abych zároveň nebyl vázán jenom na jedno testovací zařízení, použil jsem pro testování i tablet Asus Google Nexus 7 a telefon Honor 4C (viz tabulka 4.1). Také jsem vyzkoušel fungování aplikace s žádným nebo pomalým připojením k internetu (2G síť).

Obrázek 4.1: Ukázka běhu programu v Android Profiler

4.1.1 Testování správy paměti

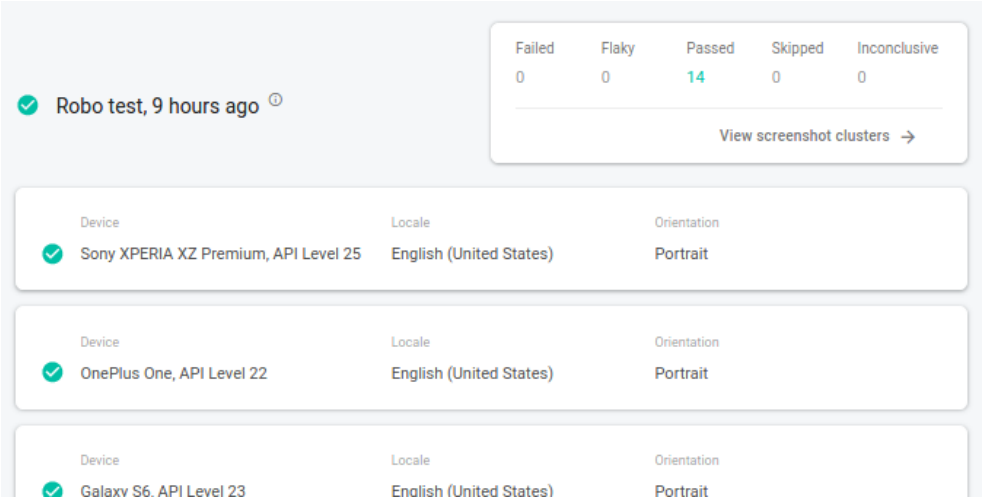
4.1.1.1 Leak Canary

Pro ověření správnosti zacházení aplikace s pamětí jsem využil volně dostupnou knihovnu LeakCanary, jež slouží pro detekci úniku paměti (ang. memory leak). [63] Únik paměti označuje situaci, kdy počítačový program není schopen uvolnit alokovanou (přidělenou) operační paměť, kterou již dále nevyužívá a obejde se i bez ní. Většinou se tak stane u objektů, které nejsou nadále potřebné, ale ostatní objekty na ně stále odkazují. Pokud aplikace obsahuje velké množství úniků paměti, pak může dokonce spadnout. Neboli jak řekl Benjamin Franklin: „Malý únik dokáže potopit i velkou loď.“ [64]

Knihovnu LeakCanary je velice jednoduché umístit do programu. Stačí ji přidat mezi závislosti do souboru *build.gradle* (viz podkapitola 3.1.3) a poté spustit aplikaci. S prvním spuštěním se zároveň nainstaluje samostatná aplikace LeakCanary, která upozorňuje na možné problémy s paměti pomocí notifikací. Ta také eviduje potřebné informace pro zjištění příčiny úniku.

4.1.1.2 Android Profiler

Mimo to jsem analyzoval běh aplikace s použitím Android Profiler, jenž je součásti vývojového prostředí Android Studio verze 3.0 a vyšší. Android Profiler poskytuje nástroje, které pomáhají vývojářům porozumět fungování jejich aplikace. V reálném čase zobrazuje využití procesoru, množství přidělené operační paměti, komunikaci se sítí a spotřebu baterie.



Obrázek 4.2: Snímek znázorňující část výstupu konkrétního testu na fyzických zařízeních v Firebase Test Lab

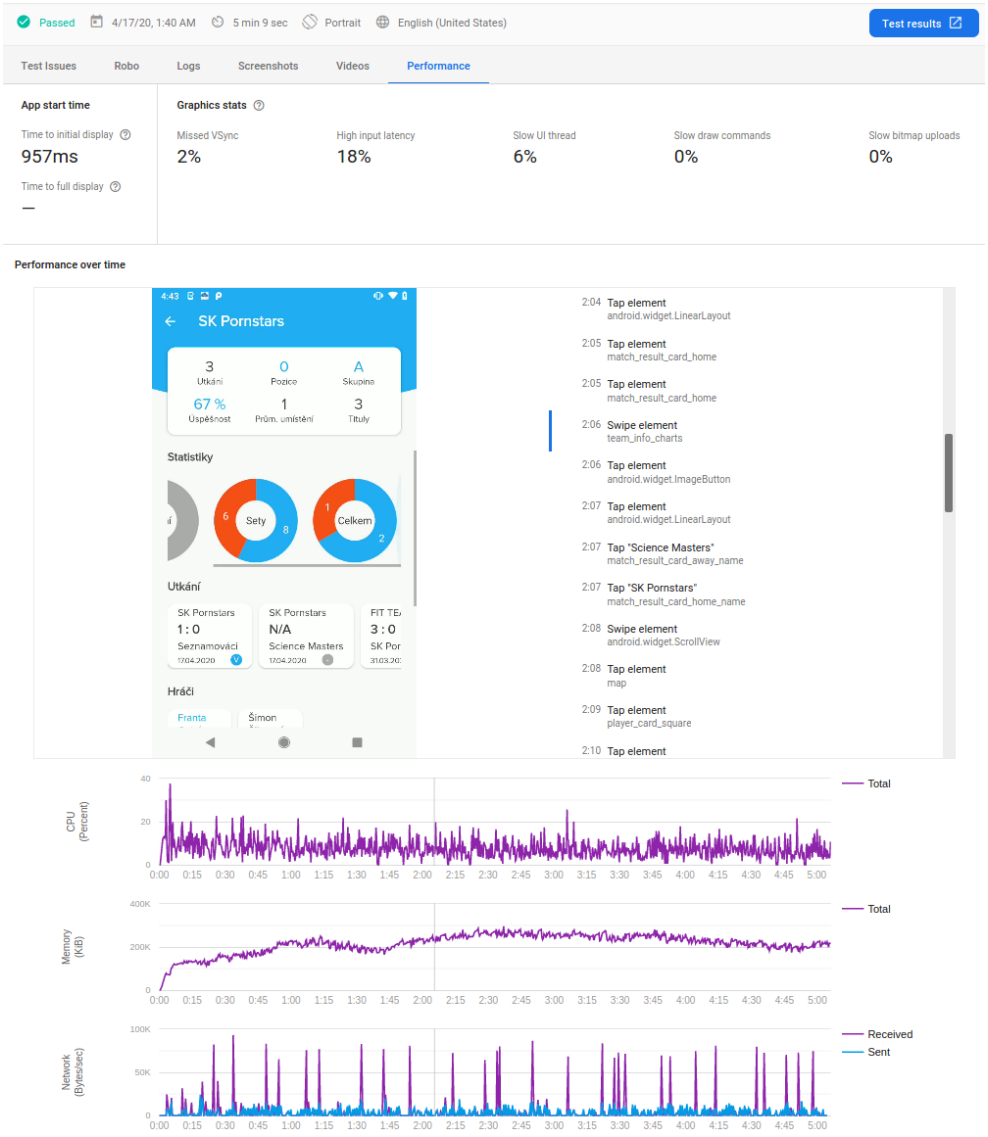
4.1.2 Testování uživatelského rozhraní

4.1.2.1 Firebase Test Lab

Abych ověřil správnost zobrazení a funkčnost uživatelského rozhraní na vícero zařízeních s různými verzemi Androidu, využil jsem testovací infrastrukturu Firebase Test Lab. Ta poskytuje desítky reálných zařízení uložených v datacentru společnosti Google, na kterých může vývojář otestovat vlastní aplikaci. Navíc Firebase Test Lab nabízí automatické testování s pomocí testovacího nástroje Robo. Ten zpočátku analyzuje strukturu aplikace a poté simuluje chování reálných uživatelů. [65] Tedy v praxi Robo kliká na tlačítka, pohybuje se napříč obrazovkami, píše text do textových polí a podobně.

Výstupem takového testu je přehled znázorňující průchod Roba napříč aplikací včetně snímků jednotlivých obrazovek a videa. Video je zvlášť užitečné, neboť spolu s ním informuje vývojáře o využití jednotlivých prostředků v zařízení (viz obrázek 4.3).

Pomocí Firebase Test Lab jsem také ověřil, že interakce mezi jednotlivými komponenty aplikace je plně funkční. S využitím dalšího nástroje – Firebase Crashlytics (viz podkapitola 4.2.3) – jsem následně odladil běh aplikace a vyřešil chyby spojené s velkým množstvím dostupných verzí Androidu na trhu.



Obrázek 4.3: Ukázka konkrétního testu v Firebase Test Lab

4.1.2.2 Show Layout Bounds

Pro ujištění, že všechny grafické prvky v mé aplikaci jsou korektně rozloženy, jsem použil vývojářskou možnost *Show Layout Bounds*. Ta je dostupná v nastavení v sekci pro vývojáře. Účel *Show Layout Bounds* tkví v tom, že vykresluje hranice všech zobrazených elementů v aplikaci, a tak poskytuje užitečnou zpětnou vazbu o správnosti rozmístění. Vývojář tak například může upravit layout, ve kterém se některé grafické prvky vzájemně překrývají.

4.2 Uživatelské testování

Po vzájemné emailové a telefonní komunikaci jsem se s panem Vaňkem domluvil na možnosti otestování aplikace vybranými vedoucími ligy *AMTEL Opava*. Připravil jsem tedy aplikaci pro distribuci mezi reálné uživatele a vymyslel testovací scénář. V průběhu testování jsem pak sledoval, zdali aplikace funguje na telefonech účastníků správně. Na konci testování jsem získal od účastníků zpětnou vazbu a tu vyhodnotil.

4.2.1 Distribuce mezi uživatele

Pro rozšíření aplikace mezi testující uživatele jsem použil platformu Firebase App Distribution. Ta poskytuje místo pro uložení jednotlivých verzí aplikace na serveru a umožňuje zaslat testerům odkaz ke stažení. Pokaždé, když je nahrána nová verze, dostanou účastníci testování email s aktuálním odkazem. Firebase App Distribution se tak díky tomu hodí i pro iterativní způsob testování. Vývojáře rovněž informuje o tom, kdo si již aplikaci nainstaloval nebo alespoň otevřel příchozí odkaz ke stažení. Jako další výhodu této platformy pak můžu uvést spolupráci s jiným nástrojem z rodiny Firebase – Firebase Crashlytics (více v kapitole 4.2.3).

4.2.2 Testovací scénář

Pro účastníky testování jsem vymyslel testovací scénář, kterého se měli během zkoušení aplikace držet. Testovací scénář se skládal z dvanácti, respektive třinácti úkolů, jež simulovaly celkový průběh ligy v deseti dnech. Účastníci reprezentovali vedoucí týmů (většina jimi skutečně je doopravdy) a já představoval vedoucího soutěže.

Vše začalo tak, že jsem v aplikaci přidělil každému účastníkovi roli vedoucího týmu. Tím jim bylo zároveň odesláno vygenerované heslo nutné pro prvotní přihlášení.

Vedoucí týmu se musel po získání hesla přihlásit a vyplnit své osobní údaje. Poté měl za úkol do stanoveného termínu vytvořit vlastní tým včetně soupisky s několika hráči. Některé hráče ze soupisky si měl rovněž vyzkoušet upravit nebo smazat.

Po pár dnech byla možnost editace soupisky uzavřena a vygenerována utkání. Zde by se ještě hodilo říci, že zaregistrované týmy jsem rozdělil do dvou skupin po pěti. V každé skupině se tak podle plánu mělo odehrát deset utkání.

Vedoucí týmů poté dostali za úkol domluvit se s ostatními vedoucími ohledně přípravy utkání a zapsat výsledky utkání, ve kterých hráli jako domácí tým. Také měli otestovat podání stížnosti k výsledku venkovního zápasu.

V konečné fázi si vedoucí zkusili zobrazit profily jejich týmů a profil některého z týmových hráčů. Také se měli pokusit o navštívení přehledu utkání cizího týmu a odhlásit se.

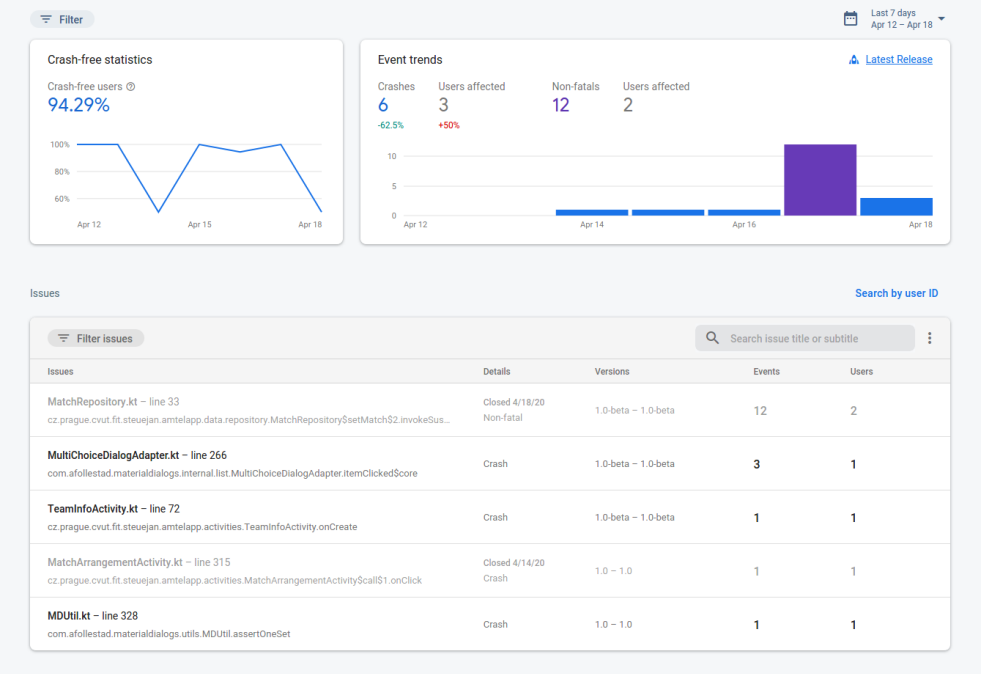
Poslední testovací den byla vygenerována baráž. Poté byla sezóna ukončena.

Pro lepší představu přikládám na závěr ucelený seznam úkolů k otestování aplikace, který byl zaslán účastníkům testování:

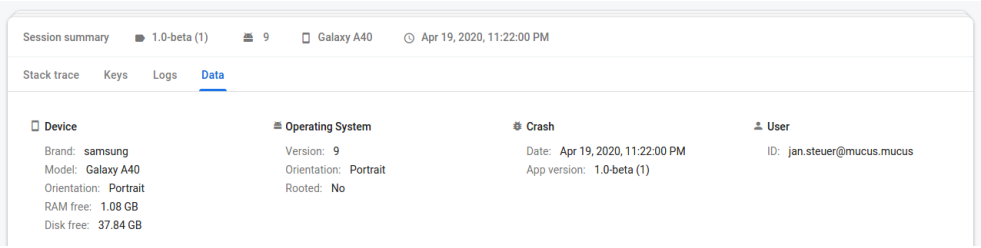
1. Přihlaste se a doplňte osobní údaje.
2. Vytvořte tým.
3. Přidejte do týmu minimálně dva hráče.
4. Přidejte do týmu jednoho hráče navíc a pak ho smažte.
5. Upravte údaje libovolného hráče.
6. Vyzkoušejte domluvu utkání s jiným vedoucím týmu.
7. Vyplňte výsledky utkání, ve kterých hrajete jako domácí tým.
8. Nahlaste jeden výsledek zápasu (ten, ve kterém jste jako hostující tým).
9. Zobrazte si profil Vašeho týmu.
10. Zobrazte si profil některého z hráčů ve Vašem týmu.
11. Zobrazte si výsledek utkání cizího týmu.
12. Odhlaste se.
13. Libovolně procházejte aplikací a zkoušejte, co Vás napadne.

4.2.3 Monitorování průběhu testování

Po celou dobu průběhu testování jsem monitoroval, zdali aplikace na některém z testovaných zařízení nespadla. K tomu jsem využil již dříve řečený nástroj – Firebase Crashlytics. Ten reportuje pády všech zařízeních, na kterých aplikace běží. Mimo to předává autorovi aplikace základní informace o typu zařízení, stack trace[30] a záznam z logu. Díky těmto informacím je vývojář schopen dobře odhalit příčinu pádu a poskytnout řešení.



Obrázek 4.4: Výpis pádů v Firebase Crashlytics během testování aplikace na fyzických zařízeních (snímek pochází z doby před uživatelským testováním)



Obrázek 4.5: Přehled konkrétního pádu aplikace v Firebase Crashlytics

[30] Seznam volaných metod v průběhu pádu programu.

Použití Firebase Crashlytics je v praxi velmi jednoduché. Stačí přidat požadovanou závislost do souboru *build.gradle* a o zbytek se automaticky postará Firebase Crashlytics SDK. K dispozici jsou rovněž metody pro snazší nalezení příčiny pádu programu (viz ukázka kódu 4.1).

```
object TestingUtil
{
   //ziskani instance predstavujici spojeni se serverem Crashlytics
   private val crash = FirebaseCrashlytics.getInstance()

   //zapise na server ID prihlaseneho uzivatele
    fun setCurrentUser(userId: String) { crash.setUserId(userId) }

   //zapise na server vlastni log
    fun log(message: String) { crash.log(message) }

   //zapise na server vlastni vyjimku, ktera ale neni fatalni
    fun throwNonFatal(ex: Exception) { crash.recordException(ex) }
}
```

Ukázka kódu 4.1: Ukázka použití Firebase Crashlytics ve zdrojovém kódu

4.2.4 Vyhodnocení uživatelského testování

V této podkapitole sumarizuji zhodnocení aplikace *AMTEL Opava* očima účastníků testování. Uživatelského testování se zúčastnil vzorek devíti lidí. Sedm respondentů bylo s fungováním ligy obeznámeno již dopředu, tj. minimálně jednou odehráli skutečné utkání v lize a byli seznámeni s formátem soutěže. Zbývající dva respondenty jsem zasvětil do fungování ligy já osobně.

Účastníci měli necelý týden na zaslání zpětné vazby prostřednictvím webového dotazníku Google Forms. Otázky byly zpravidla rozděleny do tematických celků dle jednotlivých úkolů v testovacím scénáři. Pro lepší představu uvádím jednu typovovou otázku:

Podařilo se mi změnit heslo bez problému.

- Souhlasím.
- Spíše souhlasím.
- Spíše nesouhlasím.
- Nesouhlasím.
- Heslo jsem si nezměnil.

Většina zbylých otázek byla položena v podobném duchu. Pokud respondent se zadanou otázkou (spíše) nesouhlasil, mohl uvést svůj důvod v doplňujícím komentáři. Odpovědi byly anonymní.

Přihlášení a osobní Téměř všichni respondenti souhlasili s tím, že se jim podařilo přihlásit se k svému účtu bez problému. Dva respondenti souhlasili s malými výhradami. Heslo si zkusily změnit dvě třetiny účastníků. Se změnou hesla nikdo z nich neměl potíž. Nastavit či změnit osobní údaje (jméno a příjmení, datum narození, telefonní číslo, email) přišlo snadné všem dotázaným s výjimkou jedné osoby, ta v této otázce spíše souhlasila. S odhlášením pak neměl problém nikdo. Co se týče připomínek nebo vylepšení, padla zde možnost doplnění fotek a přihlášení přes Facebook.

Tvorba týmu a soupisky Všem respondentům se podařilo vytvořit vlastní tým bez problémů. Jeden dotázaný byl ovšem zmaten položkou „jméno“ při tvorbě týmu, raději by uvítal například označení „název týmu“. Název týmu pak nejde po vytvoření změnit. S touto funkcí souhlasilo sedm dotázaných, zbylí dva by rádi změnili název i zpětně. Všichni účastníci se ale shodli na tom, že jim nedělalo problém přidat nové hráče do soupisky ani libovolného člena týmu smazat. S úpravou údajů hráče v soupisce byla ale zaznamenána i první negativní odpověď. Jeden respondent si stěžoval, že nemůže změnit pohlaví hráče. Všichni dotázaní si byli dobře vědomi toho, do kdy můžou soupisku editovat. Žádná nová funkce jim nechyběla.

Domluva utkání Všem účastníkům, kteří si zkusili změnit místo nebo datum utkání, nedělala změna problém. Co se týče přehlednosti, šest dotázaných odpovědělo kladně a dva měli menší výhrady. Jeden si domluvu utkání vůbec nevyzkoušel. Nikdo by nepřidal žádnou novou funkci.

Zápis utkání Na otázku, zdali bylo všem účastníkům jasné, jestli můžou výsledek utkání zapsat, nebo pouze nahlásit, odpovědělo kladně bez výhrad pouze pět respondentů. Dva dotázení souhlasili s menšími výhradami, jeden spíše nesouhlasil a jeden úplně nesouhlasil. Se zadáním výsledku ani jeho nahlášením ovšem neměl nikdo žádný problém. Také všichni bez výjimky souhlasili s aktuální formou zápisu výsledku – výsledek každého zápasu lze zadat pouze dvakrát (samotné zadání a poté jedna oprava). Dostal jsem rovněž jeden námět pro vylepšení stávajícího řešení, a to jest možnost vytvořit krátký report z utkání.

Přehled utkání a výsledek utkání Dva účastníci našli utkání, ve kterých hrál jejich tým, s malými potížemi, ostatní pak bez problémů. Na otázku: „*Líbilo se mi, že po kliknutí na utkání, ve kterých hrál můj tým, jsem byl přesměrován do sekce Domluva utkání*,“ odpovědělo kladně osm dotázaných.

Jeden nesouhlasil, ovšem zároveň z něj vyplynulo, že nemá s ligou žádný vztah. Naopak všem bylo zřejmé, kdo ve kterém utkání vyhrál a jaký byl konečný stav. Samotný výsledek utkání (tři samostatné zápasy) byl srozumitelný bez výhrad sedmi dotázaným. Dva si nebyli s interpretací jistí, ne ovšem natolik, aby nesouhlasili.

Pořadí týmů Naprosto všichni účastníci nalezli umístění svého týmu bez potíží. Tabulka s aktuálním pořadím týmů byla také pro všechny pochopitelná.

Profil/statistiky týmu Dva respondenti měli s nalezením profilu jejich týmu menší nesnáze. Zbylí respondenti naopak nalezli tým bez problémů. Profil týmu pak přišel přehledný téměř všem. Týmovým statistikám rozuměli až na jednoho respondenta všichni. Ten také jako jediný vznesl připomínku. Dle něj jsou statistiky týmů příliš zaměřené na sety a rád by je doplnil i o body.

Profil/statistiky hráče Nalézt profil hráče z vlastního týmu se podařilo bez výhrad všem. Všem také přijde profil hráče přehledný. S interpretací hráčových statistik neměl nikdo větší problém. Všichni respondenti by nechali vše tak, jak je.

Aplikace jako celek Osm z devíti respondentů si myslí, že je aplikace přehledná. Jeden dotázaný uvádí aplikaci jako přehlednou s mírnými nedostatky. Všichni souhlasili, že by aplikaci rádi používali.

Rovněž jsem obdržel dva náměty pro zlepšení. Jeden účastník by uvítal odeslání automatického emailu s žádostí o doplnění výsledku utkání, pokud uběhlo 48 hodin od odehrání utkání a vedoucí domácího týmu výsledek stále nezapsal. Druhý účastník by si přál po spuštění aplikace zobrazit aktuální tabulku nebo poslední odehraná utkání, namísto nynějšího přesměrování do administrátorské sekce.

4.2.4.1 Shrnutí

Největší problém, který se během testování objevil, je chyba při odesílání emailů na pozadí. O odesílání emailů se stará JavaMail API, která interně používá Gmail autentizaci. Implementace je umístěna v klientské části aplikace. Pokud je email odeslán z nové IP adresy, Google vyhodnotí přihlášení jako bezpečnostní riziko a pokus zablokuje. Tímto problémem se tedy budu v budoucnu zabývat jako první, nicméně už nyní předběžně pracuji s myšlenkou na jeho odstranění (tvorba vlastního serveru či použití Firebase Cloud Functions a umístění logiky odesílání emailů tam). Rovněž mám v plánu se v pozdější fázi vývoje zabývat podněty pro vylepšení aplikace, které jsem získal od účastníků testování.

Některé připomínky jsem ovšem vyřešil již nyní. Například jsem dle obdržené zpětné vazby zaměnil v přehledu všech hráčů datum narození za ročník.

Také jsem během tvorby týmu přejmenoval položku ve formuláři z „jména“ na „název týmu“ (obr. 4.6).

Rovněž jsem upravil dialog, který je uživatelovi zobrazen před uložením nového týmu. Nyní tak jsou uživatelem zadané údaje zrekapitulovány před odesláním do databáze. Nově je také během smazání hráče zdůrazněno hráčovo jméno.

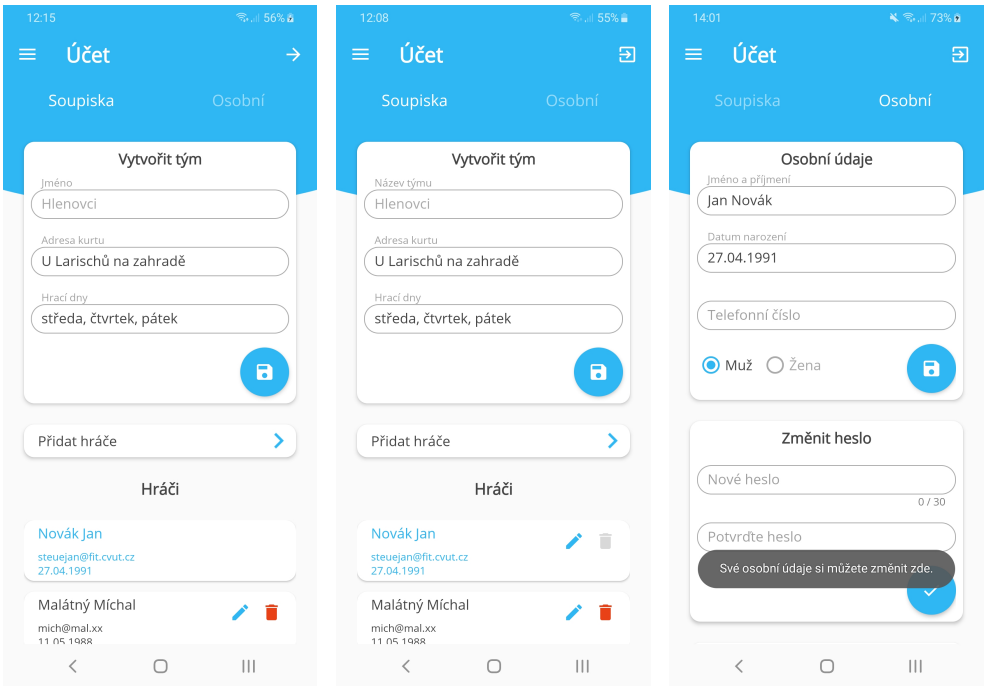
I u změny hesla došlo k úpravě. Po úspěšné změně hesla nebylo dříve vymazáno textové pole s jeho potvrzením. To jsem teď opravil.

Malé vylepšení proběhlo také v editaci hráčů. Pokud je nyní upravován běžný hráč, není zobrazeno textové pole s telefonním číslem. V případě, že chce vedoucí týmu upravit sám sebe, je automaticky přesměrován do části s osobními údaji (obr. 4.6, uprostřed a vpravo).

V zápisu utkání nebylo respondentům častokrát jasné, zdali mají výsledek utkání zadat nebo jej pouze nahlásit. Situaci jsem napravil tak, že se vedoucímu po spuštění dané aktivity zobrazí krátká zpráva s instrukcemi.

V poslední řadě jsem rovněž nahradil ikonku pro odhlášení za více intuitivní (obr. 4.6).

I přes výše zmíněné nedostatky byla ale aplikace přijata uživateli velmi kladně. Aplikaci hodnotilo 88,9 % respondentů jako přehlednou i bez mírných nedostatků a nenašel se žádný dotázaný, který by nesouhlasil s osobním používáním této aplikace.



Obrázek 4.6: Snímek vlevo zachycuje stav před uživatelským testováním. Zbylé dva snímky znázorňují drobné úpravy po testování.

Závěr

V této práci jsem se zabýval vývojem mobilní aplikace pro zjednodušení organizace amatérské tenisové ligy.

Na začátku jsem analyzoval fungování ligy *AMTEL Opava* a popsal současný postup při její organizaci. Pro lepší představu jsem také provedl rešerši již existujících aplikací usnadňující vedení libovolné soutěže. V neposlední řadě jsem analyzoval cílové uživatele a posbíral požadavky, které byly ze strany účastníků ligy pro aplikaci žádoucí.

Po analýze jsem navrhl uživatelské rozhraní aplikace a databázový model. Také jsem popsal výběr použité architektury včetně volby vhodných technologií.

Aplikaci jsem poté naimplementoval a čtenáře srozuměl s lehkým úvodem do vývoje aplikací v Androidu. Rovněž jsem nastínil řešení několika problémů, se kterými jsem se během tvorby aplikace potýkal. Na konci jsem též vysvětlil zajímavé body implementační části.

V konečné fázi jsem aplikaci otestoval s použitím automatických testů pomocí Firebase Test Lab a nasimuloval celkový průběh soutěže s reálnými uživateli. Od nich jsem také získal zpětnou vazbu a tu vyhodnotil.

Naplnění cílů

Hlavním cílem této práce bylo vytvořit aplikaci, která zjednoduší a zpřehlední organizaci amatérské tenisové ligy *AMTEL Opava*. Tento cíl byl splněn díky přesunutí všech organizačních záležitostí na jedno místo – do mnou naimplementované aplikace.

Vedoucí soutěže má v aplikaci k dispozici přehled nad kompletní správou uživatelů, utkání, skupin, týmů i baráže. Aplikace navíc usnadňuje eventuální rozšíření ligy o další skupiny včetně možnosti jejich přizpůsobení. Vedoucí soutěže může rovněž v aplikaci vytvářet zpravodaje, které jsou dostupné všem uživatelům.

Aplikace zjednodušuje organizaci i pro vedoucí týmů. Ti se můžou přímo v aplikaci domluvit s ostatními vedoucími nad nadcházejícími utkáními. Díky zadávání výsledků utkání v aplikaci odpadla nutnost přeposílat zápisy vedoucímu soutěže přes email. Zápis výsledku je pak také snazší než stažení šablony v tabulkovém procesoru a její kompletace.

Každý uživatel má v aplikaci přístup k výsledkovým listinám týmů napříč skupinami a ročníky. Rovněž aplikace eviduje statistiky hráčů i týmů a prezentuje jejich výkony v přehledné formě. Jedná se tedy o velké zpřehlednění oproti současnému stavu organizace. Tento fakt podporuje i získaná zpětná vazba z uživatelského testování.

Budoucnost aplikace

Aplikace byla naprogramována za účelem skutečného nahrazení současného způsobu organizace ligy *AMTEL Opava*. Uživatelské testování již nyní ukázalo, že je možné v blízké budoucnosti uvažovat o nasazení aplikace do reálného provozu. Nicméně musím také dodat, že se zde stále nachází prostor pro opravy chyb a zkvalitnění stávající funkcionality.

Za nejzásadnější chybu uvádím problém s odesíláním emailů na pozadí z klientské části aplikace. Tímto problémem se tedy určitě budu zabývat jako první. Také jsem během uživatelského testování obdržel řadu užitečných připomínek a vylepšení, které by bylo vhodné do aplikace naimplementovat – jmenovitě se například jedná o přehled nadcházejících utkání, upozornění vedoucího na zadání výsledku proběhlého utkání, přihlášení přes Facebook, větší důraz na body v týmových statistikách a jiné.

Přesto si myslím, že současná verze aplikace je již nyní kvalitním počinem, která i přes problémy s emaily dokáže nahradit stávající formu organizace. Rovněž přidala užitečné funkce navíc a je připravena pro možné rozšíření do budoucna. Námětů je mnoho. S odevzdáním bakalářské práce tedy má činnost na vývoji aplikace rozhodně nekončí.

Literatura

[1] AMTEL Opava [dokument ke stažení]. *Propozice AMTEL 2019*. 2019. [cit. 2020-04-21]. Dostupné z: https://www.amtelopava.cz/wp-content/uploads/2019/06/AMTEL-2019-PROPOZICE-2.doc

[2] Rozhovor s Jiřím VAŇKEM, vedoucím soutěže AMTEL Opava. Opava 20.10.2019

[3] *Soupiska týmů na tenisovou soutěž AMTEL 2019 a MAKE CUP 2019*. AMTEL Opava, 2019. Interní dokument ke stažení.

[4] VANĚK, Jiří. *Zpravodaj Č. 7* [online]. 26.06.2019 [cit. 2020-04-21]. Dostupné z: https://www.amtelopava.cz/2019/06/zpravodaj-c-7

[5] CHRIS, Alex. *Top 10 Search Engines In The World* [online]. [cit. 2020-04-21]. Dostupné z: https://www.reliablesoft.net/top-10-search-engines-in-the-world

[6] *SportyHQ: Web-based sports competition and membership platform* [online]. © 2020 [cit. 2020-04-21]. Dostupné z: https://www.sportyhq.com/

[7] *Tournamentsoftware: Your sport statistics, one place* [online]. [cit. 2020-04-21]. Dostupné z: https://www.tournamentsoftware.com/

[8] *TKMiloOlomouc* [online]. © 2016 [cit. 2020-04-21]. Dostupné z: http://milo.gotenis.cz/milo/Default.aspx

[9] *MULTILIGA* [online]. © 2015–2020 [cit. 2020-04-21]. Dostupné z: https://www.multiliga.cz/Brno

[10] *HuráLiga* [online]. © 2020 [cit. 2020-04-21]. Dostupné z: https://www.huraliga.cz/

[11] HuráLiga.cz. *Facebook* [online]. © 2020 [cit. 2020-05-06]. Dostupné z: https://www.facebook.com/huraliga.cz

[12] *VAŠE LIGA* [online]. [cit. 2020-04-21]. Dostupné z: https://www.vaseliga.cz/

[13] *TennisCall: Social Tennis, Play Tennis, Tennis App* [online]. 2019 [cit. 2020-04-21]. Dostupné z: https://play.google.com/store/apps/details?id=io.tenniscall&hl=cs

[14] *Playwaze* [online]. 2020 [cit. 2020-04-21]. Dostupné z: https://play.google.com/store/apps/details?id=com.playwaze&hl=cs

[15] Pravidla pro turnaje hrané švýcarským systémem. *Šachový svaz České republiky* [online]. 2002, 09.09.2002 [cit. 2020-05-10]. Dostupné z: https://www.chess.cz/sachovy-svaz-cr/legislativa/pravidla-pro-turnaje-hrane-svycarskym-systemem/

[16] VESELÝ, Jiří; KALENDOVSKÝ, Jan; FORMÁNEK, Bedrich. *Malá encyklopedie šachu*. 1. vyd. Praha: Olympia, 1989. 430 s. , heslo koeficient Elo

[17] MLEJNEK, Jiří. *Analýza a sběr požadavků* [přednáška]. Praha: ČVUT v Praze. [cit. 2020-04-21] Dostupné po přihlášení z: https://moodle-vyuka.cvut.cz/pluginfile.php/225503/mod_resource/content/4/03.prednaska.pdf

[18] FiMuny [dokument ke stažení]. *PB007 – SWING výpisky z přednášek*. [cit. 2020-04-21]. Dostupné z: http://fi.muny.cz/data/PB007/PB007-pb007-zapisky.docx

[19] API. In: *Lexico* [online]. © 2020 [cit. 2020-04-21]. Dostupné z: https://www.lexico.com/en/definition/api

[20] Data is Beautiful. Most Popular Mobile OS 1999 - 2019 [video] *YouTube* [online]. 2019. [cit. 2020-04-22]. Dostupné z: https://youtu.be/MMyMB4zm9so

[21] Mobile Operating System Market Share Worldwide. *Statcounter* [online]. 2020 [cit. 2020-04-22]. Dostupné z: https://gs.statcounter.com/os-market-share/mobile/worldwide

[22] Hybridní aplikace. *Ackee* [online]. 2018, 23.10.2018 [cit. 2020-04-22]. Dostupné z: https://www.ackee.cz/blog/glossary/hybridni-aplikace/

[23] Android version history. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2020-04-22]. Dostupné z: https://en.wikipedia.org/wiki/Android_version_history

[24] Mobile Android operating system market share by version worldwide from January 2018 to January 2020. *Statista* [online]. 2020 [cit. 2020-04-23]. Dostupné z: https://www.statista.com/statistics/921152/mobile-android-version-share-worldwide/

[25] HAASE, Chet. *Google I/O 2019: Empowering developers to build the best experiences on Android + Play* [online]. 07.05.2019 [cit. 2020-04-23]. Dostupné z: https://android-developers.googleblog.com/2019/05/google-io-2019-empowering-developers-to-build-experiences-on-Android-Play.html

[26] Android KTX. *Android Developers* [online]. 2020, 03.04.2020 [cit. 2020-04-23]. Dostupné z: https://developer.android.com/kotlin/ktx

[27] Psychologie barev. *Jinudy: grafický design* [online]. © 2012 [cit. 2020-04-23]. Dostupné z: http://www.jinudy.cz/clanky/psychologie-barev/

[28] Open Sans. *Google Fonts* [online]. [cit. 2020-04-23]. Dostupné z: https://fonts.google.com/specimen/Open+Sans

[29] ČÁPKA, David. Monolitická a dvouvrstvá architektura. *ITnetwork* [online]. © 2020 [cit. 2020-04-23]. Dostupné z: https://www.itnetwork.cz/navrh/architektury-a-dependency-injection/monoliticka-a-douvrstva-architektura

[30] LiveData Overview. *Android Developers* [online]. 2020, 22.01.2020 [cit. 2020-04-27]. Dostupné z: https://developer.android.com/topic/libraries/architecture/livedata

[31] ViewModel Overview. *Android Developers* [online]. 2020, 22.01.2020 [cit. 2020-04-27]. Dostupné z: https://developer.android.com/topic/libraries/architecture/viewmodel

[32] Pricing plans. *Firebase* [online]. [cit. 2020-04-24]. Dostupné z: https://firebase.google.com/pricing

[33] You can make your backend vanish! [fotografie] In: *Medium* [online]. 25.09.2018 [cit. 2020-04-24]. Dostupné z: https://miro.medium.com/max/1400/0*DylbZPWyXT7S0Fn5

[34] SVOBODA, Martin. *Introduction* [přednáška]. Praha: ČVUT v Praze a Univerzita Karlova. [cit. 2020-04-25] Dostupné z: https://www.ksi.mff.cuni.cz/~svoboda/courses/191-MIE-PDB/lectures/MIEPDB16-Lecture-01-Introduction.pdf

[35] Cloud Firestore. *Firebase documentation* [online]. 2020, 20.04.2020 [cit. 2020-05-04]. Dostupné z: https://firebase.google.com/docs/firestore

[36] How do queries work in Cloud Firestore? – Get to Know Cloud Firestore #2 [video] *YouTube* [online]. 30.05.2018. [cit. 2020-05-04]. Dostupné z: https://youtu.be/Ofux_4c94FI

[37] KERPELMAN, Todd. "Why is my Cloud Firestore query slow?". *Medium* [online]. 2019, 01.08.2019 [cit. 2020-04-23]. Dostupné z: https://medium.com/firebase-developers/why-is-my-cloud-firestore-query-slow-e081fb8e55dd

[38] Understand Firebase Security Rules for Cloud Storage. *Firebase documentation* [online]. 2020, 08.04.2020 [cit. 2020-05-04]. Dostupné z: https://firebase.google.com/docs/storage/security/index

[39] Android (operating system). In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2020-04-27]. Dostupné z: https://en.wikipedia.org/wiki/Android_(operating_system)

[40] Android Lollipop. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2020-04-27]. Dostupné z: https://en.wikipedia.org/wiki/Android_Lollipop

[41] Android 5.0 Lollipop. *Android* [online]. © 2014 [cit. 2020-04-27]. Dostupné z: https://www.android.com/intl/en_uk/history/#/lollipop

[42] Material Design. *Material* [online]. [cit. 2020-04-27]. Dostupné z: https://material.io/design

[43] SAMAT, Sameer. GOOGLE. A pop of color and more: updates to Android's brand. GOOGLE. *The Keyword* [online]. 2019, 22.08.2019 [cit. 2020-04-27]. Dostupné z: https://www.blog.google/products/android/evolving-android-brand/

[44] From top to bottom: Google's original Android brand, the 2014 refresh, and the 2019 version. [fotografie] In: *AndroidCentral* [online]. 22.08.2019 [cit. 2020-04-27]. Dostupné z: https://www.androidcentral.com/sites/androidcentral.com/files/styles/xlarge/public/article_images/2019/08/android-logo-three-generations.jpg?itok=bX-mJpU0

[45] Application Fundamentals. *Android Developers* [online]. 2019, 27.12.2019 [cit. 2020-04-27]. Dostupné z: https://developer.android.com/guide/components/fundamentals#Components

[46] App Manifest Overview. *Android Developers* [online]. 2020, 17.04.2020 [cit. 2020-04-27]. Dostupné z: https://developer.android.com/guide/topics/manifest/manifest-intro

[47] Scenario 3: Rotation and other configuration changes [fotografie] In: *Medium* [online]. 05.12.2017 [cit. 2020-04-27]. Dostupné z: https://miro.medium.com/max/964/1*DCo7awxJ3KhnW88h365vhA.png

[48] Layouts. *Android Developers* [online]. 2020, 17.04.2020 [cit. 2020-04-27]. Dostupné z: https://developer.android.com/guide/topics/ui/declaring-layout

[49] Build high-quality apps for every Android device with Android Studio. *Android Developers* [online]. 2020, 16.04.2020 [cit. 2020-04-27]. Dostupné z: https://developer.android.com/distribute/best-practices/develop/build-with-android-studio

[50] Configure your build. *Android Developers* [online]. 2020, 17.04.2020 [cit. 2020-04-27]. Dostupné z: https://developer.android.com/studio/build

[51] ViewModel scoping [fotografie] In: *Medium* [online]. 18.01.2019 [cit. 2020-04-27]. Dostupné z: https://miro.medium.com/max/1400/1*InXHWv6E6bLpOAXbTRZ9Zg.png

[52] Core J2EE Patterns – Data Access Object. *Oracle* [online]. © 2019 [cit. 2020-04-28]. Dostupné z: https://www.oracle.com/technetwork/java/dataaccessobject-138824.html

[53] PAVLÍČEK, Josef. *ORM – Objektově relační mapování* [přednáška]. Praha: ČVUT v Praze. [cit. 2020-04-28] Dostupné z: https://courses.fit.cvut.cz/BI-TJV/lectures/files/bi-tjv-6.pdf

[54] Android Repository pattern. *StackOverflow* [online]. 2018, 29.05.2018 [cit. 2020-04-28]. Dostupné z: https://stackoverflow.com/questions/50573183/android-repository-pattern

[55] Coroutines Basics. *Kotlin* [online]. [cit. 2020-04-28]. Dostupné z: https://kotlinlang.org/docs/reference/coroutines/basics.html

[56] Painless threading. *Android Developers Blog* [online]. 2009, 06.05.2009 [cit. 2020-04-28]. Dostupné z: https://android-developers.googleblog.com/2009/05/painless-threading.html

[57] Use Kotlin coroutines with Architecture components. *Android Developers* [online]. 2019, 27.12.2019 [cit. 2020-04-28]. Dostupné z: https://developer.android.com/topic/libraries/architecture/coroutines

[58] View Binding. *Android Developers* [online]. 2020, 01.04.2020 [cit. 2020-04-28]. Dostupné z: https://developer.android.com/topic/libraries/view-binding

[59] Create a List with RecyclerView Part of Android Jetpack. *Android Developers* [online]. 2020, 25.04.2020 [cit. 2020-04-28]. Dostupné z: https://developer.android.com/guide/topics/ui/layout/recyclerview

[60] Adapter. *Android Developers* [online]. 2019, 27.12.2019 [cit. 2020-04-28]. Dostupné z: https://developer.android.com/reference/android/widget/Adapter

[61] Telefonický rozhovor s Jiřím VAŇKEM, vedoucím soutěže AMTEL Opava. Horní Bludovice 24.04.2020

[62] Elasticsearch Service pricing. *Elastic* [online]. © 2020 [cit. 2020-05-01]. Dostupné z: https://www.elastic.co/elasticsearch/service/pricing

[63] LeakCanary. *GitHub* [online]. © 2020 [cit. 2020-05-03]. Dostupné z: https://github.com/square/leakcanary

[64] Quote by Benjamin Franklin. *Goodreads* [online]. © 2020 [cit. 2020-05-03]. Dostupné z: https://www.goodreads.com/quotes/663983-a-small-leak-will-sink-a-great-ship

[65] Get started with Robo tests. *Firebase documentation* [online]. 2020, 17.04.2020 [cit. 2020-05-04]. Dostupné z: https://firebase.google.com/docs/test-lab/android/robo-ux-test

Příloha **A**

Seznam použitých zkratek

API Application Programming Interface

CRUD Create, Read, Update, Delete

DAO Data Access Object

GUI Graphical User Interface

IDE Integrated Development Environment

JSON JavaScript Object Notation

SDK Software Development Kit

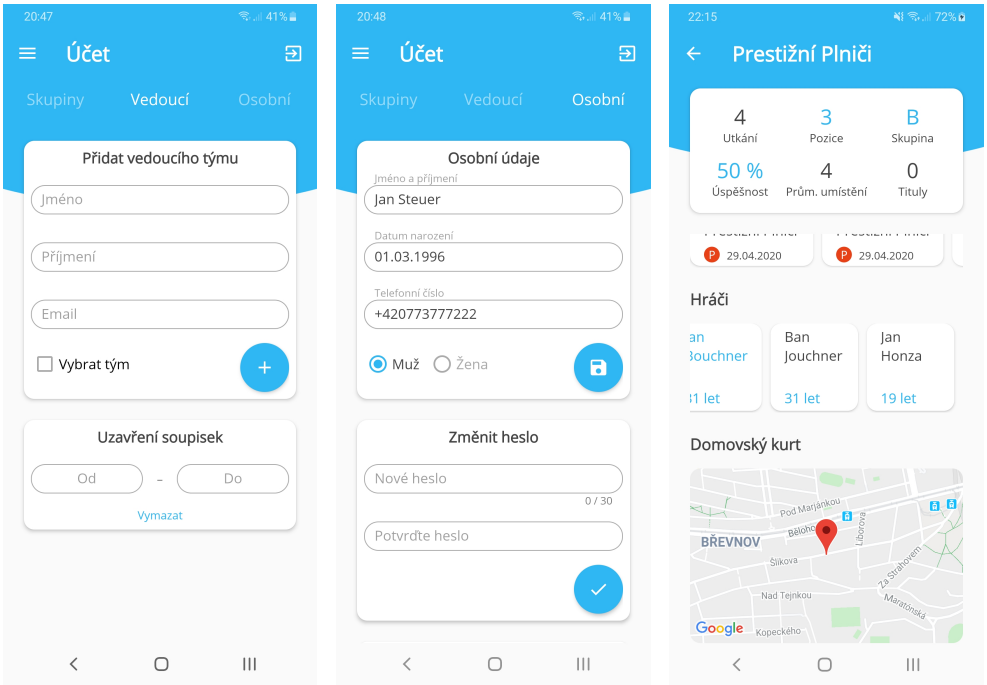
SQL Structured Query Language

UI User Interface

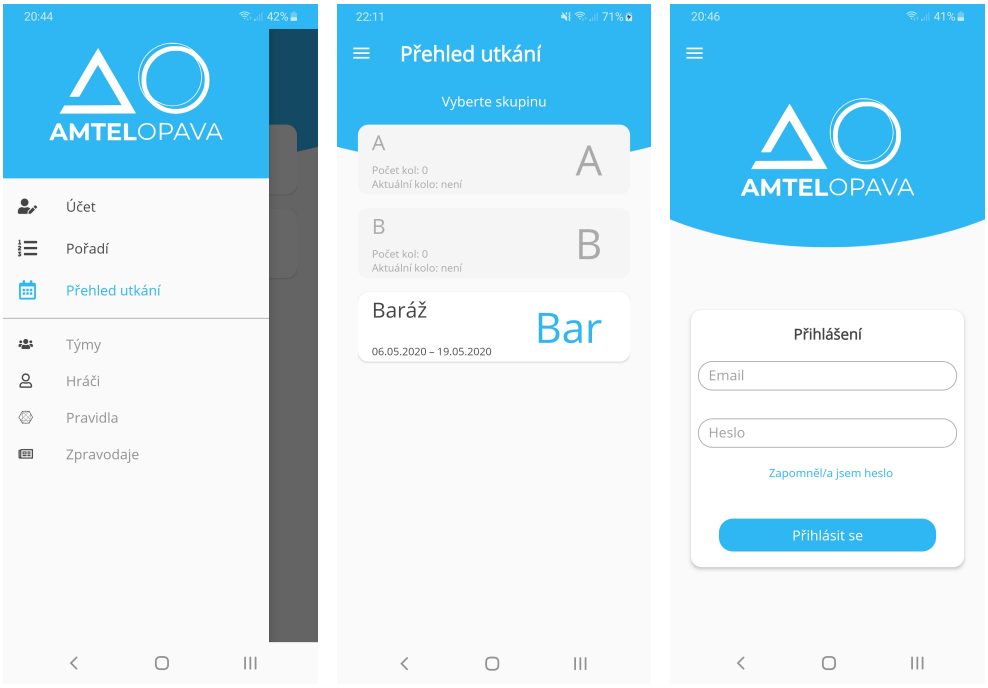
XML Extensible Markup Language

Příloha B

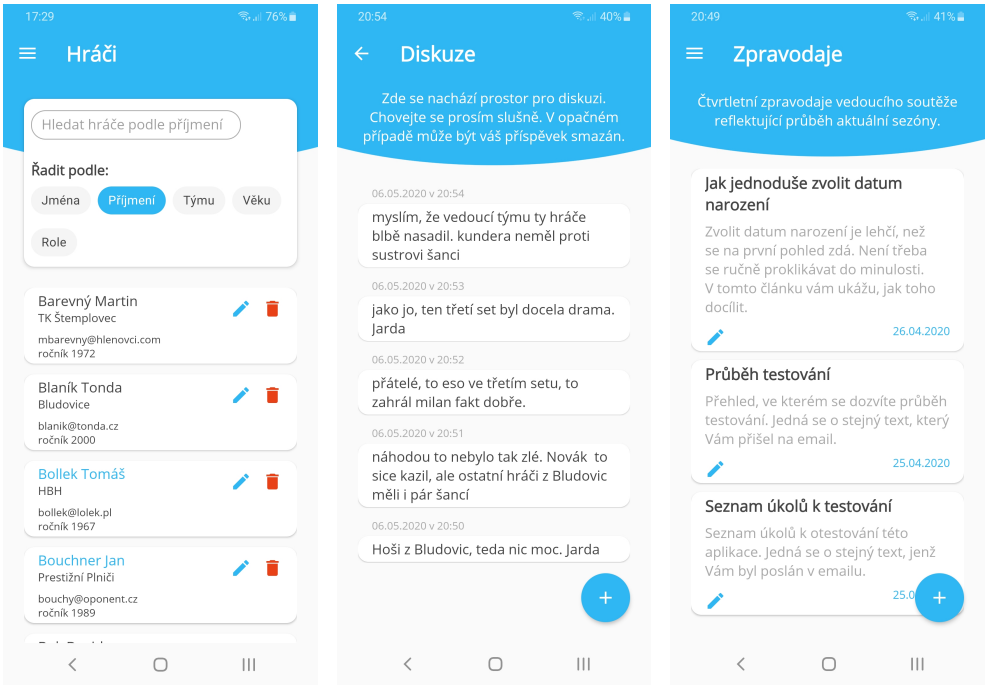
Ukázka výsledné aplikace



Obrázek B.1: Snímky výsledné aplikace znázorňující přidání nového vedoucího týmu v rozhraní pro vedoucího soutěže, sekce s osobními údaji pro přihlášené uživatele a mapa s adresou kurtu dostupná v profilu týmu



Obrázek B.2: Snímky výsledné aplikace znázorňující navigační menu, přehled utkání během baráže a přihlášení



Obrázek B.3: Snímky výsledné aplikace znázorňující seznam všech hráčů, diskuzi pod utkáním a zpravodaje

Příloha C

Propozice AMTEL Opava

Tato příloha doslovně přepisuje poslední dostupné propozice ligy *AMTEL Opava* z roku 2019. Autorem textu je Mgr. Jiří Vaněk, vedoucí soutěže. [1]

C.1 Organizační pokyny

C.1.1 Systém soutěže

V ročníku 2019 se budou hrát dvě soutěže:

1. Hráči do 49 let včetně, plus starším hráčům účast povolena
2. Hráči 50+
   a) Tuto soutěž bude hrát 38 týmů rozdělených do 4 výkonnostních lig. Skupiny A, B budou 9členné, sk. C, D vytvoří 10 týmů.
   b) Skupinu 50+ bude hrát 8 týmů.

C.1.1.1 Hrací řád

Skupiny A, B, C, D – jednokolově každý s každým. Skupina 50+ každý s každým, poté o pořadí – 1–4. resp. 5.–8. místo.

Týmy ve sk. A, B, C které skončí po dlouhodobé části na posledním, tj. 9., resp. 10. místě sestupují přímo do sk. B, C, D.

C.1.1.2 Nadstavba

Osmý tým sk. A, B, resp. 9. tým sk. C po dlouhodobé části – baráž s 2. týmem ze sk. B, resp. C, resp. D.

Celkoví vítězové sk. B, resp. C, resp. D postupují přímo do vyšší soutěže.

C.1.2 Věk

Hranice pro vstup do soutěže:

- sk. A, B, C, D – roč. narození 2001,
- sk. 50+ – roč. narození 1969.

C.1.3 Registrace na ČTS

Hráče v minulosti registrované na ČTS lze uvést na soupisce týmu, pokud poslední soutěžní utkání odehrál nejpozději v roce 2009. Do tohoto omezení nespadají hráči nejnižší soutěže, meziokresního přeboru, kde se registrace toleruje.

U soutěže sk. A se toleruje registrace u soutěží dospělých do úrovně OBLASTNÍ PŘEBOR.

Od roč. narození 1979 se již registrace NEZKOUMÁ. Za dodržování těchto pravidel zodpovídá vedoucí družstva.

C.1.4 Vklad do soutěže

Vklad do soutěže činí 1 200 Kč a je splatný do 12. května 2019 na č.ú. 164383****/3030[31].

Jako variabilní symbol použijte své POŘADOVÉ ČÍSLO v příloze AMTEL 2019 – VKLAD. Při nezaplacení vkladu v daném termínu nebude tým do soutěže zařazen.

C.1.5 Soupiska

Vedoucí týmu ji zašlou nejpozději do 12.05.2019. V průběhu soutěže již nelze připisovat další hráče. Vedoucím týmů budou zaslány soupisky všech soupeřů ve skupině. Změny lze hlásit formou DOPLNĚNÉ SOUPISKY pouze do 02.06.2019.

C.2 Technická ustanovení

PREAMBULE. Hrací řád je dán Rozpisem utkání a určením HRACÍHO DNE jednotlivých domácích týmů, který je pro všechna utkání ZÁVAZNÝ. Hodina začátku utkání v pracovní dny je stanovena pro období květen a září 2019 na 16 hodin, pro červen a srpen na 17 hodin. Začátky utkání během víkendu budou vždy upřesněny vedoucím domácího týmu (samozřejmě platí, pokud se vedoucí nedohodnou jinak).

[31] Cenzurováno.

1. Vedoucí týmu, který má jakýkoli problém s dodržením Hracího řádu vyvolá PÍSEMNĚ přes email jednání s vedoucím soupeře o dohodě na změně termínu utkání NEJPOZDĚJI 24h před začátkem utkání. V kopii to bude POVINNĚ zasláno vedoucímu soutěže. Nedodržení této lhůty má za následek KONTUMACI utkání ve prospěch soupeře. Termín jednoho utkání lze z iniciativy jednoho ze soupeřů zrušit pouze jednou (samozřejmě platí, pokud se vedoucí nedohodnou jinak).

2. V každém utkání se odehrají 3 zápasy – dva singly + 1 debl.

    a) POUZE pro 50+. V každém utkání se odehrají 3 zápasy – dva debly + 1 singl. Minimální počet hráčů, který v utkání musí nastoupit jsou TŘI hráči – přičemž 1 hráč může odehrát NEJVÝŠE 2 zápasy, tj. singl + debl nebo 2 debly (samozřejmě platí, pokud se vedoucí nedohodnou jinak).

3. Vedoucí týmů se dohodnou na dvojicích pro singly. Pokud nedojde k dohodě, rozhodne o soupeřích los. Není-li dohodnuto jinak, hraje se debl, jako závěrečný zápas.

    a) POUZE pro sk. E a 50+. DOMÁCÍ tým má právo rozhodnout o složení singlových resp. deblových soupeřů. POUZE pro 50+ – není-li dohodnuto jinak, hraje se singl, jako závěrečný zápas.

4. Zápasy se hrají dle pravidel tenisu, na dva vítězné sety. Za stavu 6:6 se v každém setu hraje tie break.

    a) POUZE pro 50+. Za stavu 1:1 na sety se místo 3. setu hraje SUPER TIE BREAK, do 10 bodů (samozřejmě platí, pokud se vedoucí týmů nedohodnou jinak).

5. Domácí tým dodá na první dva zápasy 8 nových míčů – Wilson US Open nebo Dunlop FORT TP. Informace k objednání míčů najdete v příloze AMTEL 2019 – MÍČE.

6. Vedoucí domácího týmu je povinen NEPRODLENĚ po každém odehraném utkání odeslat emailem vedoucímu soutěže Zápis o utkání.

7. Vedoucí soutěže „pověsí“ následně přehled všech výsledků v jednotlivých kolech na naše webové stránky – www.skrtopava.cz.

8. Bodování. Vítězství – 2 body, porážka – 1 bod, kontumace – 0 bodů.

9. KRITÉRIA PRO URČOVÁNÍ POŘADÍ VE SKUPINĚ při bodové shodě:

    - VZÁJEMNÉ UTKÁNÍ dvou týmů,
    - v případě bodové rovnosti 3 a více týmů týmů, MALÁ TABULKA těchto týmů.

C.3 Závěrečná ustanovení

Každý tým má přidělené číslo, které vyčte v příloze AMTEL 2019 – KONTAKTY v kolonce P.č. Systém vám umožní v případě potřeby rovněž předehrávat svá utkání. K tomu by měly sloužit náhradní termíny především prázdniny, tedy 27.–34. týden a 38. týden. Zde se iniciativě samozřejmě meze nekladou a KAŽDÁ PŘEDEHRÁVKA je vítána, což platí především pro „podzimní“ termíny (např. víme dopředu o služebních cestách, o dovolených, o lékařských zákrocích, atd.). Tímto vyzývám všechny týmy ke vzájemné vstřícnosti. Soutěž jistě nehrajeme proto, abychom vyhrávali kontumačně. Povinností KAŽDÉHO TÝMU je odehrát do 14.07.2019 NEJMÉNĚ 5 utkání ve sk. A, B, a nejméně 6 utkání ve sk. C, D a 50+. Nedodržení této povinnosti má za následek vyloučení týmu ze soutěže. !!!!!! LETOS TOTO PRAVIDLO NEKOMPROMISNĚ DODRŽÍM!!!!!!! A nebudu pátrat z čího zavinění bylo to které utkání odloženo!!! Na odehrání těchto 5, resp. 6 utkání máme všichni 8 (slovy OSM) týdnů, takže není ŽÁDNÁ OMLUVA či lépe VÝMLUVA pro nesplnění tohoto požadavku!!!! U zářijových kol upozorňuji na fakt, že nejpozdějším termínem pro jejich odehrání bude PÁTEK 27.09.2019.

Rovněž pátek 27.09.2019 je dnem uzávěrky pro odeslání Zápisů o utkání, abych je mohl o víkendu zpracovat a dohodnout dvojice pro BARÁŽ. Proto doporučuji 8.–9. kolo v případě možnosti předehrát.

Příloha D

Obsah přiloženého média

- `readme.txt` ..... stručný popis obsahu média
- `apk` ..... adresář se spustitelnou formou implementace
- `misc`
  - `amtel` ..... současný způsob organizace *AMTEL Opava*
  - `testing` ..... materiály k uživatelskému testování
  - `wireframes` ..... wireframy aplikace
- `src`
  - `impl` ..... zdrojové kódy implementace
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF