ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Název:	Tournament Manager - balíček pro hokej
Student:	Ondřej Košut
Vedoucí:	Ing. Zdeněk Rybola
Studijní program:	Informatika
Studijní obor:	Softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	Do konce letního semestru 2016/17

Pokyny pro vypracování

Vytvořte balíček pro podporu hokeje a příbuzných sportů (např. hokejbal) v rámci aplikace Tournament Manager pro OS Android. Balíček by měl umožnit organizaci soutěží a turnajů, evidenci dílčích zápasů, výsledků a statistik hráčů - vše s ohledem na specifika daného sportu.

Cíle práce:
- Seznamte se s principy jádra aplikace Tournament Manager.
- Analyzujte požadavky na organizaci turnajů a evidenci výsledků v hokeji a příbuzných sportech.
- Navrhněte architekturu a řešení balíčku pro daný sport.
- Implementujte balíček pro daný sport dle provedeného návrhu a řádně jej otestujte.
- Zdokumentujte implementované řešení.
- Vytvořte uživatelskou příručku pro použití balíčku pro daný sport.

Seznam odborné literatury

Dodá vedoucí práce.

Ing. Michal Valenta, Ph.D.
vedoucí katedry

prof. Ing. Pavel Tvrdík, CSc.
děkan

V Praze dne 16. listopadu 2015

České vysoké učení technické v Praze
Fakulta informačních technologií
Katedra Softwarového Inženýrství

Bakalářská práce

Tournament Manager - balíček pro hokej

Ondřej Košut

Vedoucí práce: Ing. Zdeněk Rybola

15. května 2016

Poděkování

Děkuji svému vedoucímu za cenné připomínky a především děkuji svojí rodině za podporu v průběhu psaní této práce.

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 15. května 2016 .....

Odkaz na tuto práci

Košut, Ondřej. *Tournament Manager - balíček pro hokej*. Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2016.

Abstrakt

Tato bakalářská práce se zabývá návrhem a implementací samostatného balíčku pro aplikaci Tournament Manager pro OS Android, určenou pro správu turnajů ve sportech. Aplikace jako celek byla vyvíjena v týmu pěti studentů, ve kterém se každý soustředil na jinou její část. Tato práce se zabývá částí, jíž je vytvoření balíčku konkrétně pro hokej a jemu příbuzné sporty tak, aby byl snadno využitelný pro menší přátelské turnaje. Balíček komunikuje s jádrem aplikace a využívá jeho knihovnu. Výstupem práce je tedy funkční balíček a jeho dokumentace včetně příručky pro uživatele. Pro implementaci byl využit jazyk Java pro Android.

Klíčová slova Tournament Manager, hokej, Android, softwarové inženýrství, smartphone, Java, SQLite

Abstract

This bachelor's thesis describes the design and implementation of a standalone package for the application Tournament Manager for OS Android, which is supposed to manage tournaments in sports. The application as a whole was developed by a team consisting of five students, where each student focused on a certain part. The part, which is the subject of this thesis, is the creation of a package for managing tournaments in hockey and similar sports in a way, to be useful for smaller friendly tournaments. The package communicates with the application core and uses its library. The result of this thesis is a working package, its documentation and a user guide. The package is implemented in Java for Android.

Keywords Tournament Manager, hockey, Android, software engineering, smartphone, Java, SQLite

Obsah

Seznam obrázků

Seznam tabulek

Úvod

Existuje velké množství lidí, kteří více či méně pravidelně provozují nějaký sport. Lidé si jdou večer nebo o víkendu zahrát s přáteli a často mohou chtít, ať už ze zvědavosti nebo kvůli soutěživosti, nějakým způsobem evidovat svoje statistiky. Chtěli by vědět, jak dopadli, jestli se zlepšují, jak si stojí v poměru se svými přáteli a podobně. Dlouhodobě si nic takového nelze zapamatovat, a tedy musí přijít na řadu evidence.

Evidenci lze jistě vyřešit papírovou formou, ovšem ta se po delším čase stává nepřehlednou a náročnou na udržování. Pokud chce člověk vyřešit evidenci s využitím informačních technologií, přímo se nabízí řešení pomocí chytrých telefonů. Chytrý mobilní telefon má dnes již podstatná část uživatelů mobilních telefonů, a zároveň je vždy po ruce.

Vedoucí této práce se řadí právě do zmíněné skupiny lidí, kteří chodí sportovat a chtějí si ukládat statistiky, a rozhodl se tedy, že iniciuje vytvoření aplikace, která umožní přehledně a jednoduše si evidovat statistiky zápasů se svými přáteli. Sestavil tedy tým pěti studentů – dvou studentů bakalářského a tří studentů magisterského studia – pro vytvoření takové aplikace, která by odpovídala jeho požadavkům, přičemž se každý student ve své závěrečné práci soustřeďuje na jinou část této aplikace. Tato práce je zaměřená na balíček pro hokej a jemu příbuzné sporty (jednotlivé skupiny sportů jsou řešeny samostatně stahovatelnými balíčky). Další práce se zabývají následujícími tématy: balíček pro squash a příbuzné sporty [8], návrh a realizace jádra aplikace a knihovny [6], serverová část aplikace [5] a realizace aplikace pro OS Windows 10 Mobile [7].

Tato práce obsahuje několik kapitol, které postupně popisují vývoj balíčku. První kapitola je zaměřená na analýzu, tedy hlavně specifikaci požadavků na aplikaci a balíček. Následuje kapitola zabývající se návrhem architektury jak celé aplikace, tak samotného balíčku, přičemž na začátku této kapitoly je krátký popis použité technologie Android. Poslední kapitola nakonec obsahuje informace o realizaci a o testování. Zde jsou uvedeny ukázky kódu a popis automatického i uživatelského testování.

Cíl práce

Hlavním cílem práce je vytvořit balíček pro podporu hokeje a příbuzných sportů v rámci aplikace Tournament manager (dále jen „aplikace“) pro OS Android. Tento balíček má využívat knihovnu, kterou aplikace poskytuje. Mezi hlavní funkcionality, které musí balíček podporovat je organizace soutěží, turnajů, evidence dílčích zápasů, výsledků těchto zápasů a statistik jednotlivých hráčů v zápasech. Vše s ohledem na specifika hokeje.

Cíle práce byly s ohledem na hlavní cíl stanoveny takto:

- Seznámit se s principy jádra aplikace – jádro poskytuje knihovnu, kterou lze využít k vytvoření balíčků, a proto je nutné se s jejím rozhraním seznámit, aby ho bylo možné správně využít.
- Analyzovat požadavky na organizaci turnajů a evidenci výsledků v hokeji a příbuzných sportech – jelikož je aplikace primárně určená ke sledování statistik jednotlivých hráčů, je nutné především navrhnout, jaké statistiky jsou pro uživatele v tomto sportu relevantní. Dále je nutné zjistit, zda jsou nějaká specifika hokeje, která jsou v rozporu s hlavní strukturou aplikace.
- Navrhnout architekturu a řešení balíčku pro daný sport – v této architektuře musí být možné naplnit stanovené požadavky a zároveň musí pracovat se specifiky operačního systému Android.
- Implementovat balíček pro daný sport dle provedeného návrhu a řádně jej otestovat.
- Zdokumentovat implementované řešení – bude možné využít automaticky generovanou dokumentaci.
- Vytvořit uživatelskou příručku pro použití balíčku pro daný sport – tato příručka má uživateli umožnit rychlou orientaci v aplikaci.

Kapitola 1

Analýza

K vývoji aplikace byl využit tzv. vodopádový model softwarového vývoje (angl. *Waterfall Software Development Life Cycle*). Tento postup obsahuje několik po sobě jdoucích fází, jimiž jsou: analýza požadavků, návrh, vývoj, testování a údržba. [13] Při vývoji jsou použity všechny tyto fáze kromě údržby, jelikož žádná zatím nebyla vykonána. Analýza, která je obsahem této kapitoly, je tedy právě první fází vývoje. Další fáze lze nalézt v následujících kapitolách.

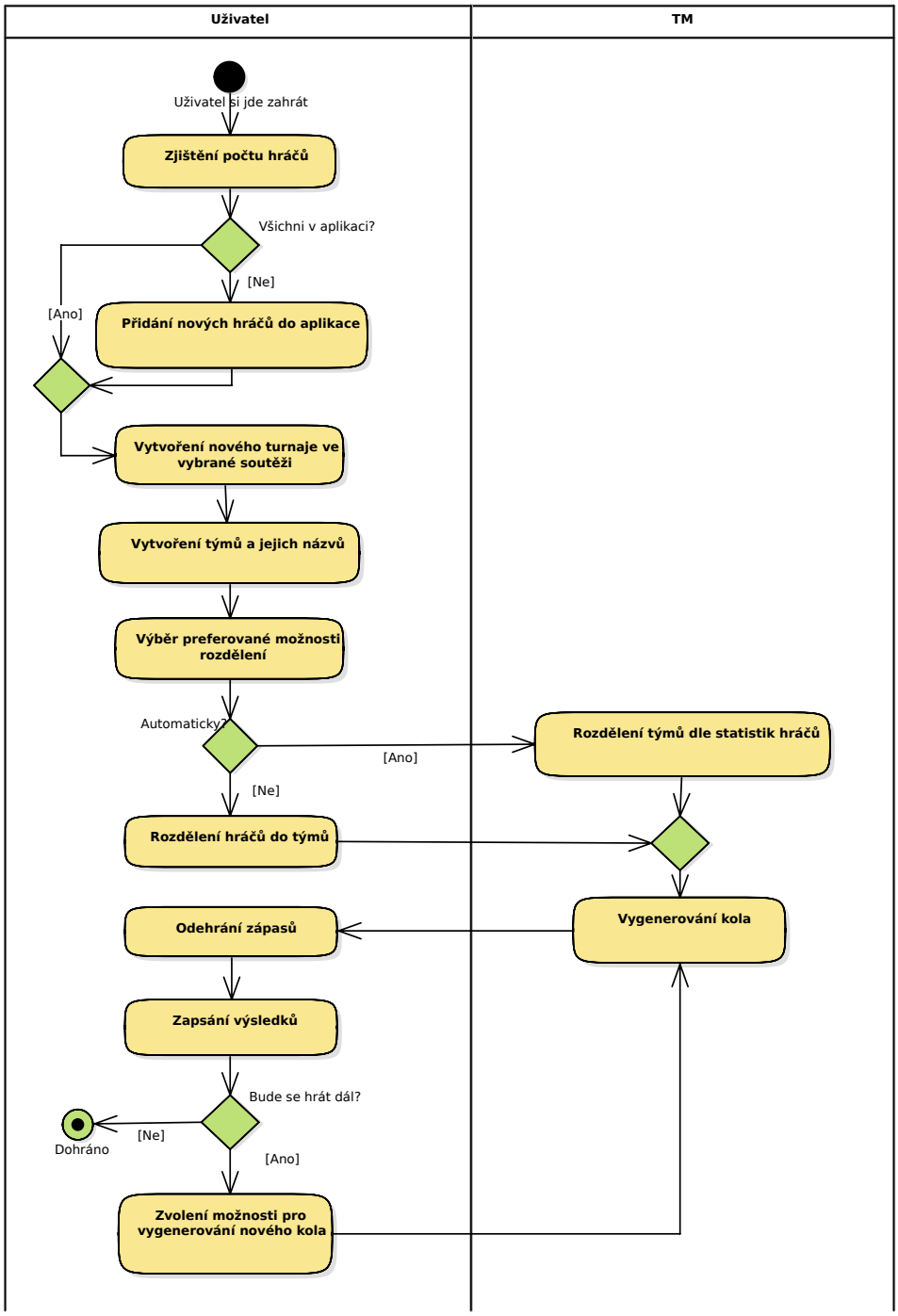
V této kapitole jsou tedy popsána některá již existující řešení dané problematiky (hlavně dostupné aplikace na OS Android), dále je rozvedena specifikace požadavků na aplikaci, na ukládání a zobrazování statistik. To vše s ohledem na daný sport – hokej. Kromě požadavků jsou specifikovány také případy užití, doménový model aplikace a některé procesy.

1.1 Procesy

Aplikace je primárně určená k usnadnění organizace menších amatérských turnajů, při které má pomoci sledovat statistiky v rámci více, či méně pravidelných herních akcí. Cílem tedy není vytvoření profesionální aplikace na zaznamenávání detailních statistik velkých turnajů, ale spíše nabídnout aplikaci využitelnou menšími skupinami nadšenců a přátel.

Pro tvorbu balíčku byl zmapován jeden základní proces, který lze vidět na obrázku 1.1, kde je vyobrazený s využitím diagramu aktivit (notace UML – dle [1]). Zobrazuje o celkový průběh využití aplikace při organizaci turnaje. Jak již bylo zmíněno, nejedná se ale o organizaci profesionálního turnaje, ale spíše o jednodenní sportovní akci.

Proces probíhá tímto způsobem: když si uživatel jde zahrát, nejdříve zjistí počet hráčů, kteří se budou účastnit turnaje. Pokud některé hráče ještě nemá vytvořené v aplikaci, učiní tak. Poté v aplikaci založí nový turnaj a přidá do něj všechny hráče (v případě, že ještě nebyli zaregistrovaní do soutěže, přidá je tam).



Obrázek 1.1: Proces odehrání turnaje

Po vytvoření turnaje a přidání hráčů musí uživatel vytvořit účastníky turnaje – v tomto případě týmy a u nich vytvořit soupisky hráčů. Dále uživatel vytvoří zápasy, což může udělat ručně nebo vygenerováním celého kola zápasů, ve kterých hrají účastníci způsobem každý s každým právě jednou. Poté již zbývá pouze odehrát zápasy a vyplnit jejich statistiky.

Pokud uživatel vše svědomitě vyplnil, může si následně prohlédnout výsledky turnaje – který z týmů má nejvíce bodů, který nastřílel nejvíce gólů atp., nebo si může prohlédnout agregované statistiky jednotlivých hráčů v celém turnaji.

1.2 Specifikace požadavků

Požadavky na aplikaci byly konzultovány s vedoucím práce, jakožto zároveň jejím zadavatelem a hlavním cílovým uživatelem. Ve spolupráci s ostatními členy vývojového týmu byly poté s vedoucím specifikovány obecné požadavky. Požadavky byly rozděleny na funkční a nefunkční. V následujících seznamech uvádím pouze jejich stručný výčet, protože jejich rozsah by byl ve srovnání s touto prací nepoměrně dlouhý a bezesporu by zhoršoval její přehlednost.

1.2.1 Funkční požadavky

Jedná se o požadavky na funkcionality systému, nebo-li co má systém „dělat“. Neudávají tedy například požadavky na operační systém atp.

- Systém se bude skládat z mobilní aplikace sloužící pro hlavní práci se systémem a daty a z webové aplikace umožňující synchronizaci dat mezi více mobilními klienty a prezentaci dat dalším osobám.
- Mobilní aplikace má umožnit zadávat a prohlížet veškeré informace o turnajích a soutěžích.
- Systém má evidovat soutěže. Soutěží se rozumí nejvyšší úroveň klasifikace opakujících se sportovních událostí.
- Systém má evidovat jednotlivé turnaje. Turnaj představuje konkrétní sportovní událost, během které se sledují výsledky jednotlivých zápasů, vyhodnocuje se úspěšnost a určuje se vítěz a další pořadí.
- Systém má umožňovat evidenci účastníků jednotlivých turnajů. Za účastníka může být považován tým složený z jednotlivců nebo přímo jednotlivci.
- Systém má umožňovat evidenci složení jednotlivých týmů.
- Systém má umožňovat evidenci jednotlivých zápasů v rámci turnaje. Tyto zápasy budou probíhat vždy právě mezi dvěma účastníky turnaje, ve kterém se zápas koná.

- Systém má umožňovat přípravu rozlosování jednotlivých zápasů v rámci turnaje. Zápas bude možné vložit jednotlivě nebo vygenerováním celého kola zápasů (kolem se myslí množina zápasů, kdy hraje každý účastník s každým právě jednou).
- Systém má evidovat jednotlivé hráče a jejich globální statistiky.
- Systém má umožnit přiřazovat registrované hráče do soutěží.
- Systém má umožnit přiřazovat hráče registrované v soutěži do jednotlivých turnajů v dané soutěži.
- Systém má evidovat osobní statistiky hráčů v rámci zápasu.
- Systém má evidovat agregované statistiky hráčů v rámci turnaje a soutěže.
- Webová aplikace má umožňovat zobrazení výsledků soutěže.
- Webová aplikace má umožňovat sdílení dat soutěže mezi uživateli.
- Mobilní aplikace bude řešena modulárně, nebo-li pro jednotlivé skupiny sportů budou vytvořeny balíčky, které bude možné instalovat samostatně a pro spuštění aplikace nebude nutné mít všechny balíčky.

Výše zmíněné jsou obecné požadavky na aplikaci. Tyto požadavky je ovšem v některých částech nutné blíže specifikovat a upravit, aby vyhovovaly evidenci hokeje, rovněž je vhodné identifikovat takové požadavky, které jsou pro balíček jako takový buď irelevantní, anebo alespoň méně důležité. V následujících odstavcích jsou postupně diskutovány.

Požadavky týkající se webové části aplikace jsou pro balíček spíše nedůležité. Pro komunikaci s webovou částí má balíček využívat rozhraní poskytnuté knihovnou a na návrh architektury a implementaci tak má tento požadavek minimální dopady.

Jedním z požadavků, který je nutno upravit, je evidence účastníků turnaje. Aplikace má obecně umožňovat, aby se turnaje mohly účastnit jak týmy, tak jednotlivci. Ovšem v hokeji (ani příbuzných sportech), jakožto týmovém sportu, není nutné se zabývat jednotlivci, a proto bude v balíčku možné vést jako účastníky turnaje pouze týmy. Dále bylo rozhodnuto, že týmy nebudou mít minimální a maximální počet hráčů, dle pravidel hokeje, aby bylo možné aplikaci využít i při „kamarádských“ turnajích, kdy se sejde menší množství účastníků. Soupisky týmů bude možné ručně nastavit.

Dále je nutné pro potřeby hokeje specifikovat, jaké statistiky je nutné ukládat. Samozřejmě je nutné u každého zápasu mít možnost nastavit výsledek zápasu. To lze udělat nastavením počtu gólů domácího a hostujícího týmu. Kromě toho je ale u zápasu potřeba evidovat, zda skončil v normálním čase,

v prodloužení, nebo až po nájezdech. To za účelem dopočítání bodů získaných za zápas. V turnaji bude totiž možné nastavit, kolik bodů získá každý tým za výhru, prohru nebo remízu a tyto body se mohou odlišně nastavit pro prodloužení a nájezdy.

Dále bude možné u každého hráče v konkrétním zápase zadat počet gólů, přihrávek, +/- bodů a zákroků. Tyto statistiky budou dostačovat pro dopočítání agregovaných statistik v rámci turnaje a soutěže. Agregované statistiky se budou zobrazovat v seznamu hráčů v turnaji a v soutěži a budou obsahovat: góly, asistence, body, zákroky, +/- body, týmové body, počet zápasů, počet výher, proher a remíz, průměrný počet gólů, bodů, týmových bodů a +/- bodů za zápas.

Balíček se bude dát využít i pro příbuzné sporty, jako je hokejbal a florbal. To z důvodu, že všechny tyto statistiky jsou relevantní i pro příbuzné sporty, a také díky tomu, že soupisky týmů nejsou nijak omezené a na velikosti týmů tedy nezáleží. Statistiky se také neukládají v závislosti na herním čase, a tedy je lze využít, přestože například hokej a hokejbal mají dle pravidel rozdílný herní čas (hokej má třikrát 20 minut [17], hokejbal má třikrát 15 minut [18]). Lze ovšem podotknout, že i pokud by týmy měly fixní velikost, tak by stále bylo možné využít balíček i pro hokejbal a florbal, protože hokej má stejný počet hráčů na hřišti jako hokejbal i florbal [19].

1.2.2 Nefunkční požadavky

Nefunkční požadavky jsou takové, které neudávají, co má systém umět (neboli nehovoří o funkcionalitách), ale udávají například v čem má systém fungovat atp. Pro balíček byly specifikovány tyto nefunkční požadavky:

- Balíček má být realizovaný pro OS Android.
- Balíček má fungovat i bez připojení k internetu.
- Balíček má informace o hráčích získávat z jádra.
- Balíček má být spustitelný pouze za pomoci jádra.
- Balíček má být lokalizovaný do českého a anglického jazyka.

1.3 Rešerše existujících řešení

Dle požadavků jsem se při hledání existujících řešení zaměřoval na následující kritéria:

- Aplikace musí umožňovat sdílení přes internet
- Aplikace musí fungovat i v režimu bez připojení k internetu
- Aplikace podporuje hokej nebo hokejbal

- Aplikaci lze využít na OS Android

V internetovém obchodě Google Play [2] lze nalézt několik mobilních aplikací pro systém Android, řešících obdobný problém. V jejich popisu lze ale nalézt, že nesplňují některé požadavky, které vedoucí této bakalářské práce od aplikace požaduje.

První zkoumanou aplikací je Tournament Manager [9] V popisu aplikace lze nalézt „*WARNING: This App requires Internet Connection to manage your Tournaments*“ (tj., že aplikace vyžaduje internetové připojení), což je v rozporu s kritériem, že aplikace má fungovat v režimu bez připojení k internetu (tzv. „offline“). Dále tato aplikace neukládá žádné statistiky specifické pro hokej a nedokáže pracovat s hráči v týmech. Aplikace není specializovaná na žádné sporty, takže ukládání speciálních statistik zde nelze provést v žádném sportu. Lze ji ovšem využít v OS Android a umožňuje sdílení výsledků prostřednictvím internetu.

Další zkoumanou aplikací je aplikace jménem Bracket Maker & Tournament App [10]. V jejím popisu je opět zmíněno, že pro správu turnajů vyžaduje internetové připojení, což je v rozporu s nutností funkčnosti v režimu bez připojení. Na druhou stranu ovšem poskytuje podporu pro hokej. V hokeji ale neumožňuje ukládání některých hráčských statistik, které jsou požadovány. Aplikace jinak splňuje všechna kritéria.

Další je aplikace Tournament Manager (jiná aplikace, než výše uvedená, ale se stejným názvem) [11] V jejím popisu je uvedeno následující: „*By default, applications are installed in the following sports: football, volleyball, basketball, handball.*“, aplikace tedy neobsahuje připravenou možnost správy turnajů a zápasů s pravidly hokeje. Je ovšem využitelná na OS Android a lze s ní pracovat v režimu offline. Aplikace ale nenabízí možnost sdílení výsledků.

Dále byly vyhledávány aplikace, které nejsou nabízené přímo pro systém Android. Jako první lze uvést webovou aplikaci Konkuri [12]. Tato aplikace je sice využitelná přes internetový prohlížeč v OS Android, ovšem její využití na malém zařízení je velmi nepřehledné a nepohodlné a lze tedy považovat toto kritérium za nesplněné. Dále, jelikož jde o webovou aplikaci, nelze ji využít v režimu bez internetového připojení, a tedy nesplňuje ani toto kritérium. Ostatní kritéria aplikace splňuje.

U žádné z těchto aplikací také není zmíněno řešení pomocí samostatných balíčků pro určité sporty, které by byly samostatně stahovatelné. Dá se tedy předpokládat, že řešení pomocí balíčků zatím není pro Android poskytováno.

Řešení, které se nejvíce blíží požadavkům vedoucího, je jeho vlastní aplikace. Tato aplikace ale také nesplňuje požadavky na modularitu a na to, aby bylo možné spravovat různé sporty a v nich specifické statistiky.

Bylo tedy zjištěno, že žádná existující aplikace nevyhovuje poměrně přesným požadavkům vedoucího.

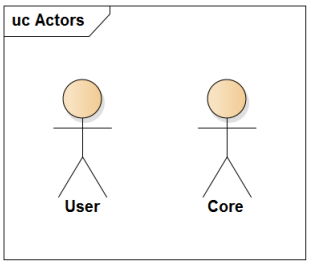
1.4 Případy užití

V rámci analýzy jsou specifikovány také případy užití. Dle [14] *„diagramy případů užití jsou behaviorální diagramy použité k zachycení, specifikaci a vizualizaci požadovaného chování systému“* (překlad autora). Umožňují tedy přesnější návrh, jak bude uživatel se systémem pracovat a jaké všechny funkcionality mu systém má nabízet. Nejprve byli identifikování aktéři, což jsou entity, které nějakým způsobem pracují s navrhovaným systémem, a poté byly specifikovány samotné případy užití.

1.4.1 Aktéři

V balíčku byli identifikováni dva aktéři (viz obrázek 1.2). Prvním je uživatel, což je běžný uživatel aplikace, má přístup ke všem funkcionalitám, které aplikace nabízí pomocí uživatelského rozhraní. V aplikaci neexistují žádná speciální oprávnění, takže zde není žádný správce nebo třeba platící uživatel.

Druhým aktérem je jádro. Jelikož je aplikace modulární, musí balíček poskytovat potřebné informace jádru aplikace, a tedy jádro může využít některé funkčnosti balíčku.



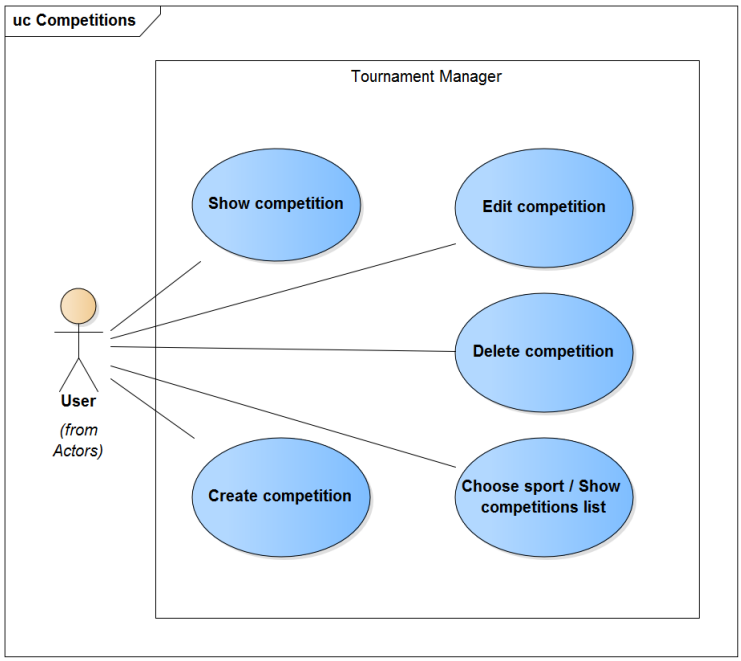
Obrázek 1.2: Aktéři

1.4.2 Diagramy a popis případů užití

Případy užití aplikace jsou rozděleny do několika diagramů. Tyto diagramy jsou členěny tematicky. Níže lze nalézt stručné popisy všech případů užití. U některých případů byly definovány rozsáhlejší popisy a další kritéria – to lze najít v přiloženém dokumentu.

1.4.2.1 Případy užití vztahující se k soutěžím

Případy užití soutěží obsahují akce, mezi něž se řadí vytvoření soutěže, zobrazení soutěže, upravení soutěže, smazání soutěže a zobrazení všech soutěží ve sportu. Pro balíček jsou relevantní všechny tyto případy užití, s výjimkou zobrazení všech soutěží ve sportu, což zajišťuje jádro aplikace.



Obrázek 1.3: Případy užití soutěží

Create competition – uživatel má možnost v aplikaci vytvořit soutěž. Při vytváření soutěže se aplikace dotáže na název soutěže, datum počátku, datum konce, typ a poznámku.

Edit competition – uživatel zvolí u soutěže možnost pro úpravu. Aplikace zobrazí stejný formulář, jako při vytváření soutěže a po upravení ji uloží.

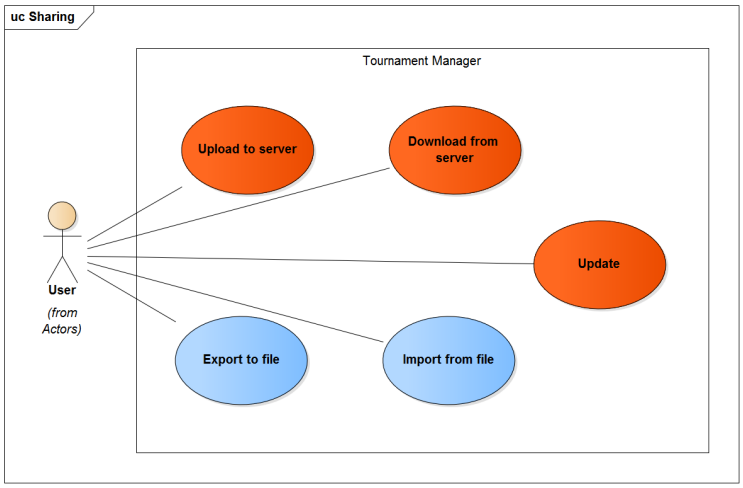
Delete competition – uživatel zvolí u soutěže její odstranění. Aplikace zkontroluje, zda je soutěž prázdná, a pokud ano, odstraní jí.

Choose sport/Show competitions list – každá soutěž probíhá v konkrétním sportu. Uživatel má možnost zvolit si sport a aplikace mu ukáže seznam soutěží, které se konají právě v tomto sportu.

Show competition – uživatel si ze seznamu vybere soutěž a aplikace mu zobrazí její detail. Součástí detailu bude její název, datum začátku a konce soutěže, poznámka, počet účastnících se hráčů a počet turnajů.

1.4.3 Případy užití vztahující se ke sdílení

Na obrázku 1.4 lze vidět případy užití, které souvisí se sdílením soutěží. Jedná se tedy o komunikaci s webovou částí aplikace, což zahrnuje nahrávání soutěží na server, jejich stahování, ale také třeba exportování do souboru. Všechny zobrazené případy jsou ovšem volány z jádra a podporovány knihovnou, takže na implementaci balíčku budou mít minimální dopad.



Obrázek 1.4: Případy užití sdílení

Upload to server – uživatel zvolí u soutěže možnost pro její nahrání na server. Aplikace nahraje soutěž a uživateli zobrazí odkaz pro sdílení, který může někdo jiný, komu ho uživatel předá, využít ke stažení soutěže.

Download from server – pokud má uživatel odkaz pro sdílení soutěže, může jej využít ke stažení soutěže ze serveru. Uživatel zadá odkaz do aplikace, a ta stáhne danou soutěž.

Update – uživatel zvolí možnost pro aktualizaci soutěže. Aplikace zajistí stažení nových úprav, které jsou na serveru a nemá je uživatel stažené a zároveň odešle na server úpravy, které uživatel udělal a ještě nejsou na serveru.

Export to file – uživatel exportuje soutěž do souboru, aby ji mohl později ze souboru nahrát do aplikace.

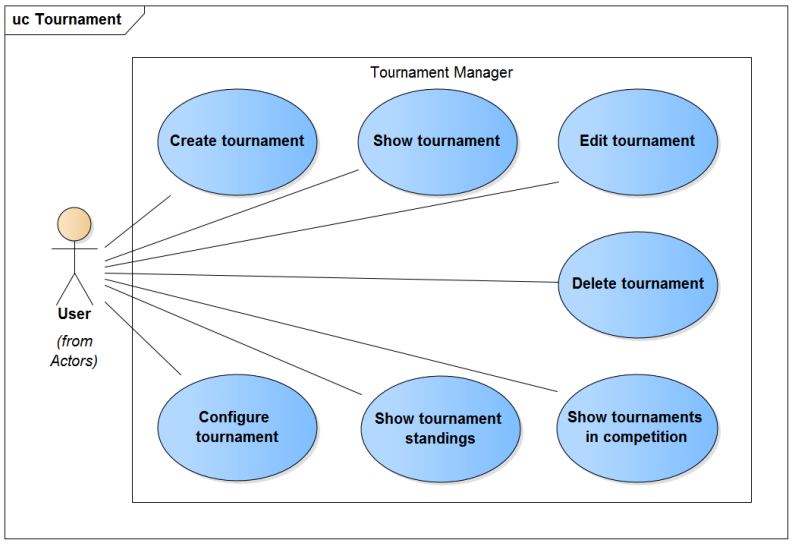
Import from file – uživatel nahraje do aplikace soutěž, která byla dříve exportována.

1.4.3.1 Případy užití vztahující se k turnajům

Případy užití vztahující se k turnaji (viz obr. 1.5) jsou pro balíček důležité všechny, protože je balíček spravuje. Kromě klasických případů, jako je vytvoření turnaje, zobrazení atp. je zde také „Configure tournament“, tedy konfigurace turnaje, což je nastavení bodového ohodnocení týmů za odehrané zápasy. Jedná se v podstatě o nastavení atributů entity *Configuration*, kterou lze nalézt v doménovém modelu (obr. 1.10).

Create tournament – uživatel vytvoří v soutěži nový turnaj. Aplikace se při vytváření dotáže uživatele na název turnaje, začátek a konec turnaje a na poznámku.

Show tournament – uživatel si v seznamu turnajů v soutěži vybere jeden a aplikace mu zobrazí detail tohoto turnaje. Detail obsahuje název turnaje,



Obrázek 1.5: Případy užití turnajů

datum jeho začátku a konce, počet týmů, hráčů a zápasů v turnaji a doplňující poznámku.

Edit tournament – uživatel může upravit základní informace o turnaji (tedy opět název turnaje, začátek a konec turnaje a poznámku).

Delete tournament – uživatel může v seznamu turnajů vymazat turnaj, pokud ovšem neobsahuje žádné zápasy, týmy ani hráče.

Show tournament standings – uživatel si v detailu turnaje zvolí zobrazení výsledků. Aplikace zobrazí seznam týmů seřazený dle počtu získaných bodů. Dále se v seznamu výsledků zobrazí počet výher, proher a remíz, počet vstřelených gólů a počet inkasovaných gólů.

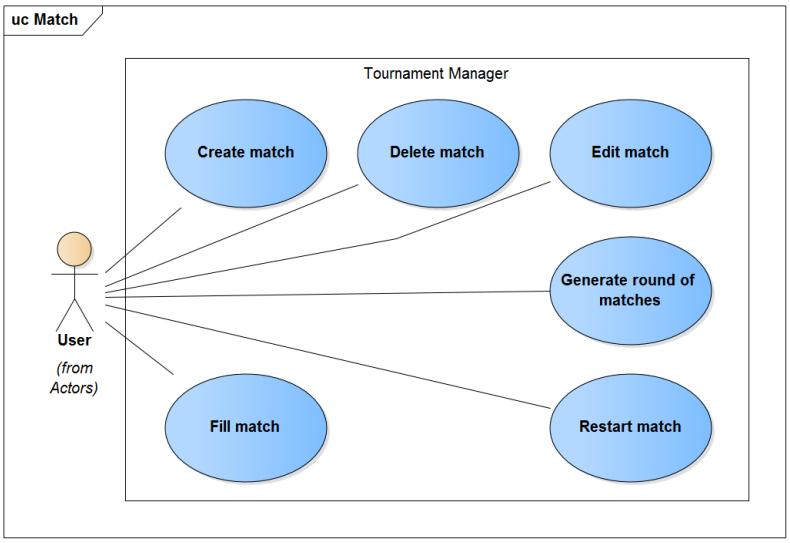
Show tournaments in competition – uživatel má možnost zobrazit si seznam všech turnajů v dané soutěži.

Configure tournament – aplikace umožňuje nastavení bodového zisku týmů za zápas. Lze nastavit bodový zisk za výhru, prohru a remízu. Na rozdíl od hry squash je zde nutné ale nastavit i bodový zisk za výhru, prohru a remízu v prodloužení a za výhru a prohru po nájezdech.

1.4.3.2 Případy užití vztahující se k zápasům

Co lze provádět se zápasy je možné vidět na obrázku 1.6. Opět se jedná o klasické operace pro celkovou správu zápasů (jako je jejich přidávání a odebírání), ale lze také vygenerovat celé kolo zápasů, restartovat zápas, což mu vymaže skóre a nastaví ho jako neodehraný, a vyplnit statistiky zápasu.

Create match – uživatel může vytvořit zápas v turnaji. Aplikace se ho při vytváření dotáže na oba týmy, periodu, kolo, datum a poznámku.



Obrázek 1.6: Případy užití zápasů

Delete match – aplikace umožňuje vymazat jakýkoliv zápas. To zahrnuje odstranění všech statistik, které s ním souvisejí.

Edit match – uživatel může i po vytvoření zápasu měnit jeho základní údaje (už ale nelze měnit týmy).

Generate round of matches – aplikace umožňuje uživateli vygenerovat celé následující kolo zápasů (v jednom kole hraje každý tým s každým právě jednou).

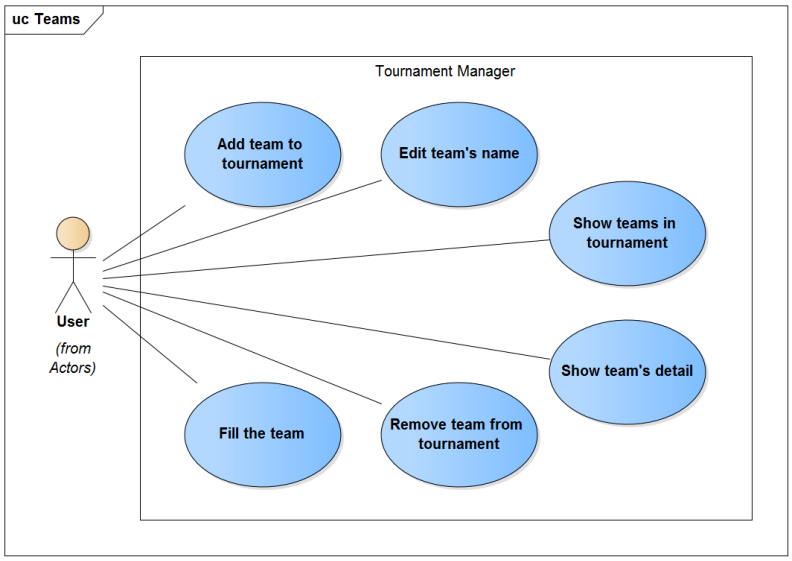
Restart match – uživatel má možnost restartovat zápas. Aplikace nastaví zápas jako neodehraný a vymaže všechny jeho statistiky.

Fill match – jedná se o vyplnění statistik zápasu. V balíčku pro hokej to funguje tak, že uživatel vyplní celkové skóre zápasu (jaký tým dal kolik gólů) a zaškrtne, zda se jednalo o prodloužení, nájezdy, nebo zda zápas skončil v normálním hracím čase. Poté přejde na soupisky týmů, které může libovolně upravit (změnit hráče v týmech) a vyplní jim statistiky v zápase. Poté vše uloží, čímž aplikace zároveň nastaví zápas jako odehraný.

1.4.3.3 Případy užití vztahující se k týmům

Na obrázku 1.7 lze vidět případy užití vztahující se k týmům. Jedná se o správu týmu v turnaji. Všechny týmy totiž existují pouze na úrovni turnaje a neexistují žádné globální týmy, které by bylo možné registrovat do turnajů a počítat jejich statistiky například v rámci celé soutěže. O všechny zobrazené případy se stará balíček.

Add team to tournament – uživatel má možnost v turnaji vytvořit tým. Při vytváření se ho aplikace pouze dotáže na název týmu. Uživatel ho zadá a aplikace tým vytvoří.



Obrázek 1.7: Případy užití týmů

Edit team's name – uživatel má možnost i po vytvoření upravit název týmu. Aplikace se ho pouze dotáže na nový název.

Show teams in tournament – uživatel si nechá ukázat seznam týmů v turnaji. Aplikace zobrazí všechny týmy společně s jejich hráči.

Show team's detail – uživatel si rozkliknutím týmu nechá zobrazit jeho detail. V detailu jsou vidět všichni hráči v týmu s možností jejich odstranění z tohoto týmu.

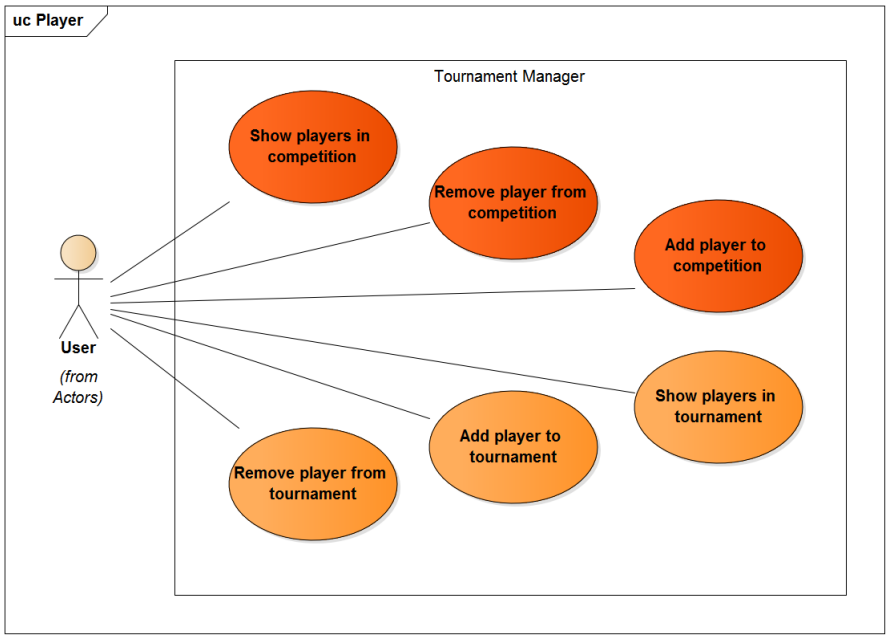
Remove team from tournament – uživatel může odstranit tým z turnaje. To mu aplikace ale umožní pouze v případě, že se tým neúčastní žádných zápasů.

Fill the team – uživatel může do týmu přidat hráče. To je možné provést v detailu týmu.

1.4.3.4 Případy užití vztahující se k hráčům

Na obrázku 1.8 jsou vidět případy užití aplikace vztahující se k hráčům. Z diagramu byly záměrně vynechány případy užití, které zajišťuje jádro aplikace (vytvoření hráče, upravení hráče atp.). Ostatní případy již zajišťuje balíček. To jsou v podstatě případy, kdy se s hráčem operuje v rámci soutěže a turnaje, konkrétně tedy jde o jejich přidávání, odstraňování a zobrazování v soutěži a turnaji.

Show players in competition – aplikace uživateli zobrazí seznam hráčů v soutěži. Tento seznam zároveň obsahuje agregované statistiky (jejich seznam lze nalézt v popisu požadavků) v rámci soutěže.



Obrázek 1.8: Případy užití hráče

Add player to competition – uživatel má možnost přidat hráče do soutěže. Hráč je poté viditelný v seznamu hráčů.

Remove player from competition – uživatel může smazat hráče ze soutěže, pokud se ovšem neúčastní žádného turnaje.

Show players in tournament – aplikace uživateli zobrazí seznam hráčů v turnaji. Tento seznam opět obsahuje agregované statistiky, ovšem nyní pouze v rámci turnaje.

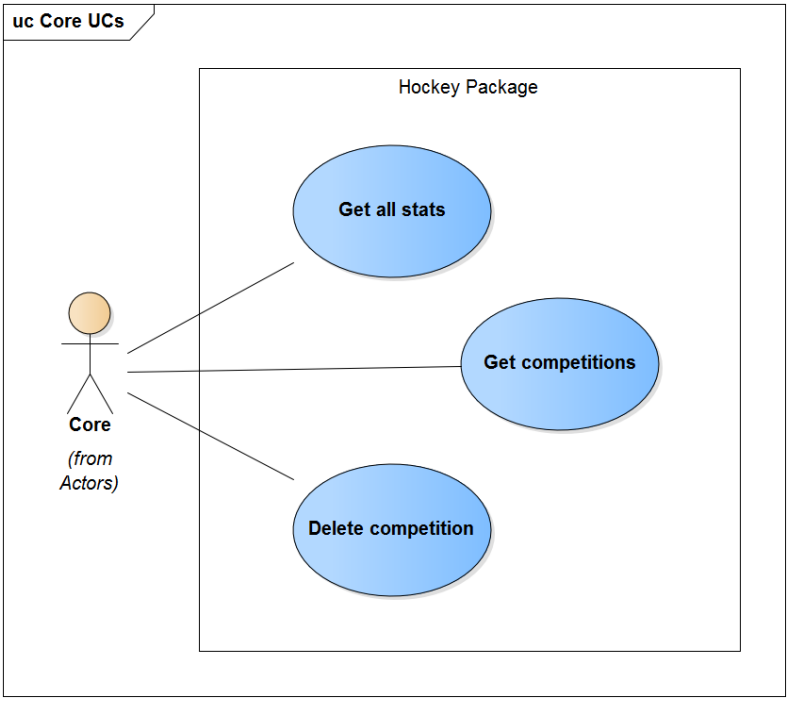
Add player to tournament – uživatel má možnost přidat hráče do turnaje. Hráč je poté viditelný v seznamu hráčů. Lze přidat pouze hráče, kteří jsou zaregistrovaní v nadřazené soutěži.

Remove player from tournament – uživatel může smazat hráče z turnaje, pokud se neúčastnil žádných zápasů a není v žádném týmu.

1.4.4 Případy užití jádra

Na obrázku 1.9 lze vidět případy užití balíčku, které využívá jádro aplikace. Tyto případy užití jsou vztažené přímo k balíčku a ten je tedy všechny implementuje. Na diagramu je vidět, že se jedná hlavně o případy užití spojené se soutěžemi. Přestože tedy balíček jako takový neposkytuje uživateli možnost zobrazit všechny soutěže, dává jádru možnost tyto soutěže získat, aby je mohlo uživateli zobrazit.

Get all stats – Jádro potřebuje mít možnost získat agregované statistiky napříč soutěžemi, aby je mohlo zobrazit uživateli. Jádro tedy pošle dotaz na



Obrázek 1.9: Případy užití jádra

tyto statistiky do balíčku a balíček odešle agregované statistiky zpět.

Get competitions – Jádro se dotáže balíčku na všechny jeho soutěže, aby je mohlo zobrazit uživateli. Balíček odešle seznam soutěží do jádra.

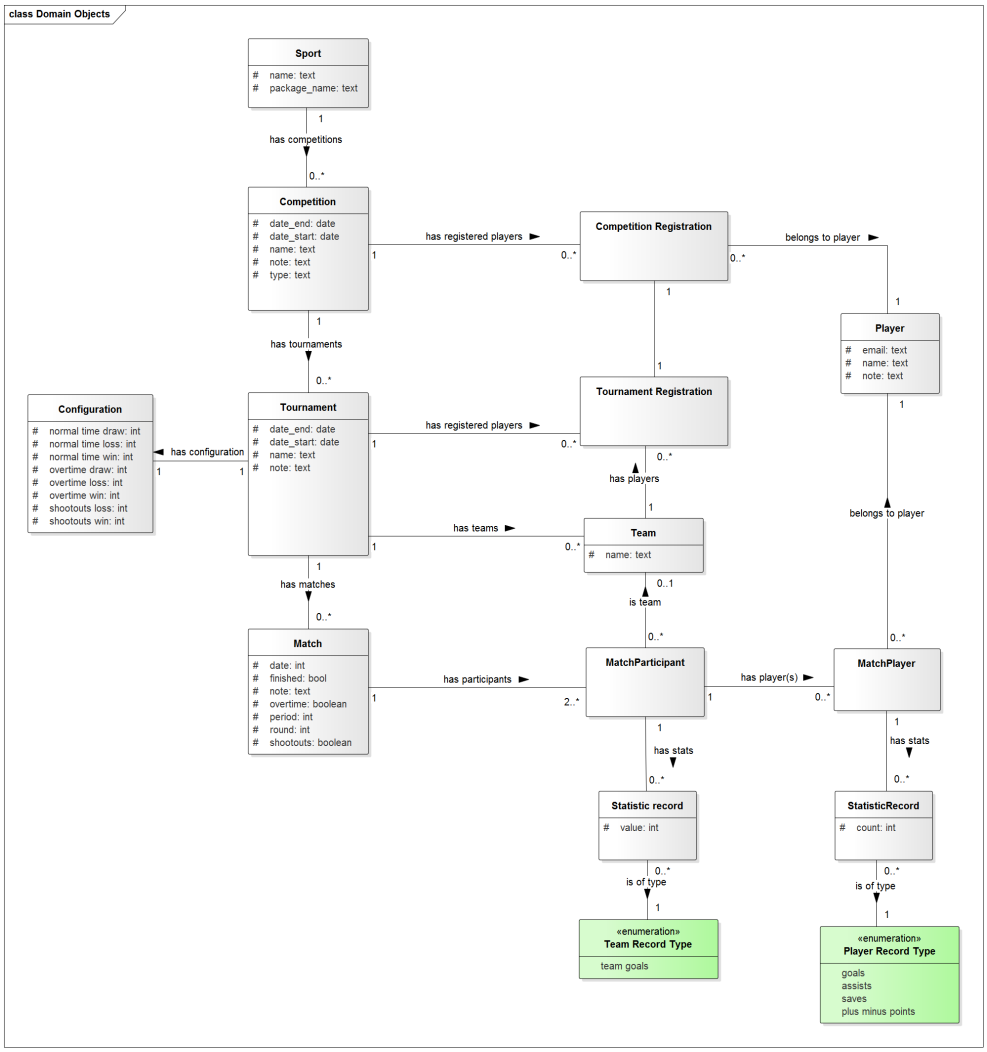
Delete competition – Jádro iniciuje smazání soutěže. Odešle tedy požadavek do balíčku. Balíček se pokusí smazat soutěž, a pokud se mu to povede, potvrdí úspěšné odstranění. Pokud ne (v případě, že soutěž obsahuje turnaje nebo hráče), odešle informaci o chybě.

1.5 Doménový model

Doménový model aplikace, který je na úrovni statistik upravený, aby odpovídal specifikacím balíčku, lze vidět na obrázku 1.10. V doménovém modelu je vidět hierarchie specifikovaná v požadavcích, která se skládá ze soutěží (na obrázku jako *Competition*), v soutěžích obsažených turnajů (*Tournament*) a v turnajích obsažených zápasů (*Match*).

Dále je z doménového modelu vidět myšlenka hierarchického vkládání hráčů. Hráče je potřeba vložit do soutěže a teprve poté je možné vložit ho do turnaje. Obdobně to funguje se zápasy - zápasů se mohou účastnit pouze hráči, kteří jsou zaregistrovaní v turnaji.

Na první pohled nemusí být zcela jasné, proč existuje entita *MatchParti-*



Obrázek 1.10: Doménový model

cipant (neboli účastník zápasu), jelikož se může zdát, že by stačilo pouze k zápasu napojit dva týmy. Pokud by se tak ovšem učinilo, nebylo by možné splnit požadavek, že týmy v turnaji musí být možné upravovat s tím, že se změní soupiska pouze nadcházejících zápasů. Také by nebylo možné splnit požadavek, aby bylo možné soupisky týmů v zápase kdykoliv změnit bez narušení ostatních zápasů. Takto lze k entitě účastníka zápasu navázat konkrétní soupisku v tomto zápase skrz entitu *MatchPlayer*. Tato entita je pouze dekompozicí vazby M:N mezi účastníkem zápasu a hráčem. Je zde ale vyobrazená, protože se k této vazbě vztahují statistiky konkrétního hráče v konkrétním zápase.

Statistiky hráčů v hokejovém zápase jsou samostatné entity. Každá instance poté reprezentuje přesně jednu statistiku (například počet gólů kon-

krétního hráče v konkrétním zápase) a váže se na výčet všech typů ukládaných statistik. Tento návrh respektuje strukturu, která je udávaná knihovnou, a jejíž hlavní výhodou je, že při přidání sledované statistiky se nemusí tento model nijak měnit (pouze přibude instance *PlayerRecordType*) a nemusí se tedy ani dělat zásahy do struktury databáze. Stejně jsou navrženy statistiky týmu v konkrétním zápase. Výčet konkrétních ukládaných statistik lze nalézt v kapitole Návrh.

Entita *Configuration* obsahuje všechny nutné parametry pro nastavení bodového hodnocení týmů v zápasech. Jsou zde číselné atributy popisující bodový zisk za výhru, za prohru a za remízu. V hokeji je potřeba myslet ale i na možnost prodloužení a nájezdů, kdy tento bodový zisk může být rozdílný, a proto jsou zde také atributy pro výhru, prohru a remízu v prodloužení. Nájezdy ovšem nemohou skončit remízou, a tedy neexistuje atribut ukládající body za remízu při nájezdech. Díky atributům *overtime* a *shootouts* v entitě *Match* lze poté při počítání bodů zjistit, jaké bodové ohodnocení mají jednotlivé týmy za zápas dostat.

Kapitola 2

Návrh

V této kapitole se zabývám návrhem aplikace. Nastiňuji celkový návrh aplikace (tedy komunikaci balíčků a jádra) v OS Android, dále návrh architektury balíčku jako takového a návrh i ve vztahu k webové části.

Poté detailněji rozebírám návrh některých jednotlivých tříd a převedení doménového modelu na tyto třídy. Toto dělám pouze u vybraných tříd z důvodu rozsahu práce.

Nakonec pomocí sekvenčního diagramu ukazuji, jak probíhá komunikace jednotlivých tříd naskrz architekturou.

Nejdříve ale popisuji vybrané součásti z technologie Androidu, které byly využity a o kterých se zmiňuji jak v kapitole návrhu, tak v kapitole realizace, a je tedy potřeba je uvést.

2.1 Technologie Android

V této sekci stručně vysvětluji některé pojmy a principy, které byly při programování využity a jsou tedy poměrně důležité k pochopení toho, jak aplikace funguje (především tedy na prezentační vrstvě).

2.1.1 Aktivita

Dle [15] „*V Androidu, Aktivita je okno, které obsahuje uživatelské rozhraní aplikace. Aplikace může mít nula nebo více aktivit.*“ (překlad autora). Aktivita je tedy jednotlivá obrazovka, kterou uživatel vidí na displeji. Aktivita využívá vzhled uživatelského rozhraní, který je vytvořený v jazyce XML a při načtení ho aktivita zobrazí.

2.1.1.1 Životní cyklus aktivity

V OS Android má aktivita jasně definovaný životní cyklus, který je při programování potřeba brát v úvahu a využívat funkce, které s ním pracují. Třída

`Activity`, ze které musí všechny aktivity dědit, definuje několik funkcí, které jsou volány, když se s aktivitou dějí změny v jejím životním cyklu. Tyto funkce zároveň výstižně ukazují, jak takový životní cyklus probíhá a jsou následující:

- `onCreate()` – volána ve chvíli, když je aktivita poprvé vytvořena.
- `onStart()` – volána, když je aktivita zobrazena uživateli.
- `onResume()` – volána, když aktivita začne s uživatelem interagovat.
- `onPause()` – volána, když je aktuální aktivita pozastavena a předchozí je znovu spuštěna.
- `onStop()` – volána v případě, že aktivita již není viditelná uživatelem.
- `onDestroy()` – volána před tím, než je aktivita zničena systémem.
- `onRestart()` – volána, když je aktivita spouštěna poté, co byla před tím zastavena.

Seznam funkci je citován z [15].

2.1.1.2 Intent

Pro vytvoření funkční aplikace je nutné, aby se postupně spouštěly různé aktivity a aby si při tom mohly mezi sebou předávat nějaké informace. Ke spuštění nové aktivity slouží objekt typu `Intent` (česky „Záměr“). Tento intent obsahuje různá kritéria a jako celek je předávaný do aktivity při jejím spuštění. Nejdůležitější součástí intentu je příslušná akce [16], která v podstatě říká „co je záměrem; co se má vykonat“, a pak nějaké další doplňky, které příjemci intentu chceme předat.

Samotné spuštění aktivity poté může probíhat metodou `startActivity()`, které se předá zvolený intent. „*Android pak najde aktivitu, která nejlépe splňuje kritéria záměru, a předá jí záměr ke zpracování*“ [16].

Druhou možností spuštění je metoda `startActivityForResult()`, která funguje podobně, ovšem musí se jí kromě intentu předat také číslo volající aktivity. Po skončení takto vytvořené aktivity se na původní – volající – aktivitě zavolá callback metoda `onActivityResult()`. Tímto způsobem lze získat výsledek vyvolané aktivity.

2.1.2 Fragment

Fragment byl do OS Android uveden až s verzí 3.0 [3] a je to samostatná část uživatelského rozhraní, či chování aktivity. Aktivita může obsahovat libovolné množství fragmentů, které s aktivitou sdílejí její životní cyklus. Hlavní výhodou fragmentů je jejich možnost znovuvyužití. Fragmenty sice existují vždy s nějakou aktivitou, ale nic nebrání využití stejné třídy definující tento

fragment i v jiné aktivitě. Příkladem může být fragment zobrazující nějaký seznam, přičemž je tento fragment využit ve dvou různých aktivitách pro dva různé seznamy.

2.1.3 Služba

„Služby se v systému Android používají pro dlouho běžící procesy, které mají být nadále spuštěné, dokonce i když již nejsou spojené s žádnou aktivitou.“ [16]. Služby se tedy dají využívat například při přehrávání hudby, kdy hudba běží i po „zhasnutí telefonu“, nebo například pro získání dat, které má aktivita zobrazit. Získání dat například z databáze totiž může trvat delší dobu, a proto je potřeba ho nevázat přímo na aktivitu. Zároveň to umožňuje „poslat pro data“ pouze jednou, přestože aktivita projde několika fázemi životního cyklu, než vůbec data získá.

2.1.4 Context

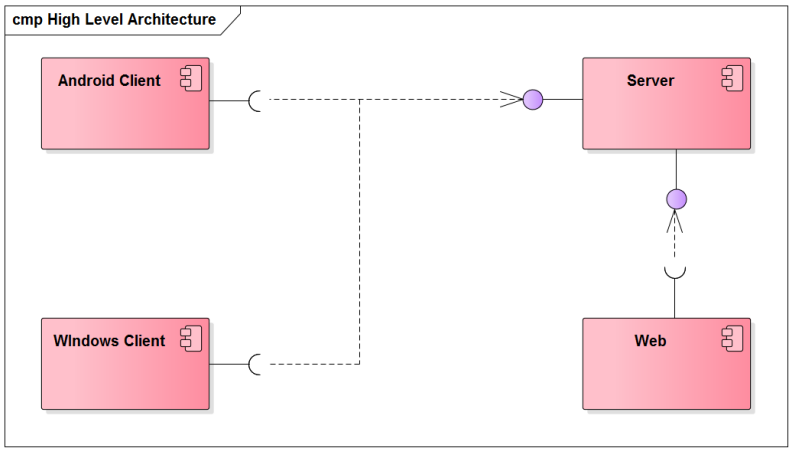
`Context` je objekt často využívaný v OS Android pro předávání reference na aplikaci [15]. Předání této reference je potřeba například v případě přístupu k databázi, nebo při dynamickém vytváření prvků uživatelského rozhraní.

2.2 Architektura

V této sekci se zabývám architekturou aplikace. Rozdělil jsem ji na tři podsekce, v nichž vždy přiblížím architekturu z trochu jiného úhlu pohledu, lépe řečeno z jiné úrovně. Nejdříve ji popisuji z té nejvyšší úrovně, tedy komunikace jednotlivých klientů a webové části. Poté přibližuji, jak vypadá celý mobilní klient, tedy jádro a balíčky a na závěr popisuji návrh architektury balíčku jako takového.

2.2.1 Komponenty aplikace

Aplikace je rozdělena na mobilní a webovou část, což lze přehledně vidět na obrázku 2.1. Jak je z diagramu patrné, mobilní část se skládá ze dvou klientů. Jedním z nich je aplikace pro OS Windows Mobile a druhým je aplikace pro OS Android. Obě tyto aplikace budou využívat stejného serverového API (rozhraní, které mohou klienti využívat), které umožňuje ukládání a synchronizaci soutěží. Server bude také poskytovat vlastní webové rozhraní, na kterém bude možné publikovat výsledky a statistiky soutěží nahrané na něj pomocí API. Pro více informací o serverové části viz [5]. Pro další informace o klientu pro OS Windows Mobile viz [7].



Obrázek 2.1: Komponenty Aplikace

2.2.2 Android klient

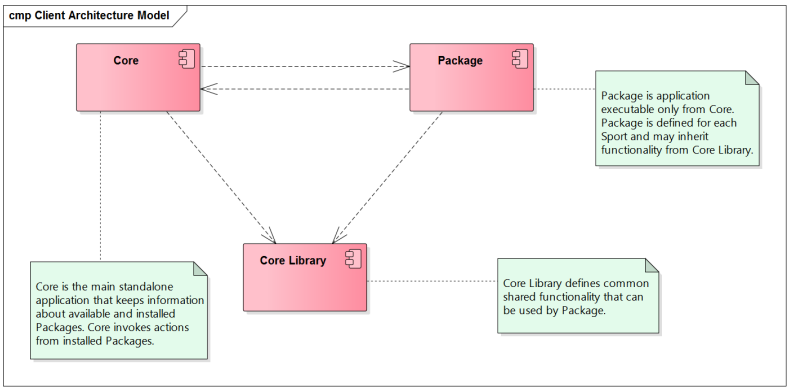
Klient pro OS Android se skládá ze tří hlavních částí, které lze vidět na obrázku 2.2. Tyto části tedy jsou „Core“, nebo-li jádro aplikace, „Package“, což jsou jednotlivé balíčky a „Core library“, tedy knihovna jádra, která poskytuje různá rozhraní a třídy.

Jádro je hlavní samostatná část aplikace, která je spustitelná přímo ze seznamu aplikací v telefonu. Jádro si drží informaci o tom, které balíčky jsou v telefonu nainstalovány a umožňuje jejich spuštění. Obsluhuje například evidenci hráčů, kteří musejí být stejní pro všechny balíčky apod.

Balíček funguje jako samostatná aplikace, která je ovšem spustitelná pouze z jádra. Jádro spustí aktivitu balíčku (detail soutěže nebo vytvoření nové soutěže), a poté již balíček funguje samostatně. Uživatel má proto dojem, že se pohybuje v jedné aplikaci.

Pro každý sport, který má specifická pravidla a statistiky – rozdílné od již existujících balíčků, je nutné vytvořit samostatný balíček. Balíček pro hokej lze bez větších obtíží využít i pro příbuzné sporty právě proto, že statistiky jsou u těchto sportů stejné (hokej, hokejbal, florbal) Zároveň s balíčkem pro hokej byl vyvíjen i balíček pro sport squash. Pro více informací o tomto balíčku viz [8] Aplikace je ale funkční s jakýmkoliv počtem balíčků, a proto si uživatel může stáhnout pouze ty balíčky, které potřebuje.

Třetí částí klienta pro OS Android je knihovna, která definuje sdílené funkcionality, které mohou balíčky i jádro využívat. Mezi ně se řadí například synchronizace se serverem, serializace soutěže a definice různých rozhraní pro datovou a doménovou vrstvu (o tom více v popisu architektury balíčku). Knihovna také definuje základní objekty jak na datové vrstvě, tak na doménové vrstvě a poskytuje abstraktní třídy využitelné pro uživatelské rozhraní. V knihovně jsou také definované konstanty pro komunikaci mezi jádrem a



Obrázek 2.2: Komponenty Android klienta

balíčky, které jsou využívány při startování balíčků a předávání informací.

Komunikace mezi jádrem a balíčky je oboustranná. Balíčky totiž potřebují mít přístup k hráčům, kteří jsou uloženi v jádru a jádro zase potřebuje například přístup k seznamu soutěží, který je uložený v každém balíčku. Jádro také spouští jednotlivé balíčky, podle toho, jakou soutěž si uživatel vybere.

Více informací o jádře a o knihovně lze nalézt v diplomové práci, která se tímto zabývá (viz [6]).

2.2.3 Balíček pro hokej

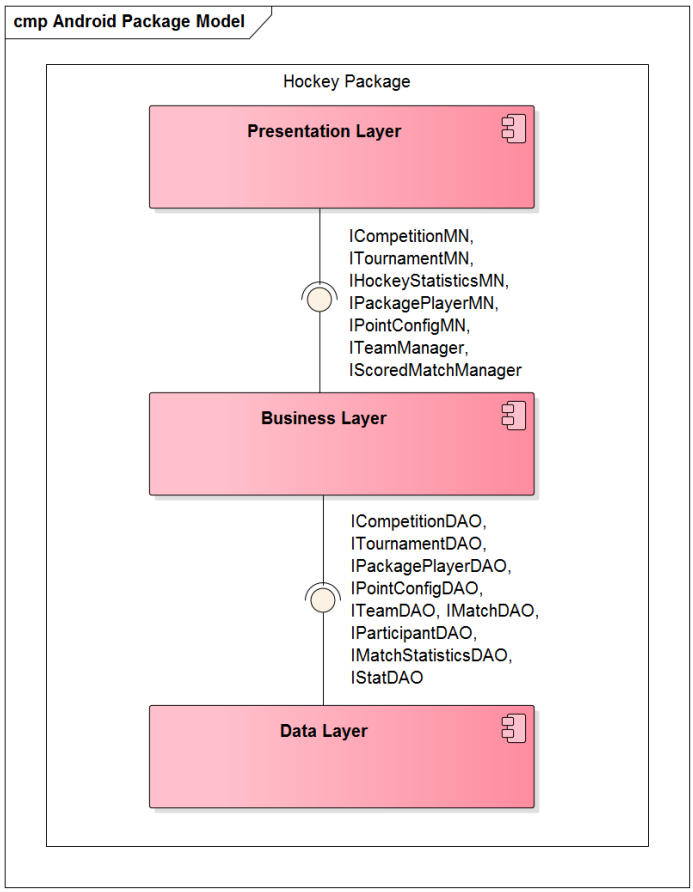
Jak lze vidět na obrázku 2.3, tak architektura balíčku je klasická třívrstvá. Obsahuje prezentační vrstvu, doménovou vrstvu a datovou vrstvu.

Datová vrstva balíčku implementuje rozhraní, která jsou definovaná knihovnou. K tomu je potřeba vytvořit další rozhraní, která budou umožňovat ukládání i entit specifických pro hokej. Mezi tato specifická DAO (Data access object - tj. objekty poskytující abstraktní rozhraní pro přístup k databázi) se řadí rozhraní na ukládání bodové konfigurace turnaje (počet bodů týmu za zápas) a speciálních statistik pro zápas (konkrétně prodloužení a nájezdy). Pro ukládání dat bude datová vrstva využívat databázi SQLite, což je databáze podporovaná systémem OS Android.

Doménová vrstva implementuje logiku, která se skládá hlavně z využívání správných DAO rozhraní pro zkompletování entit a jejich poskytnutí prezentační vrstvě. Tuto logiku doménová vrstva poskytuje opět skrze rozhraní, která jsou definovaná knihovnou a jsou rozdělená do jednotlivých tzv. managerů. Pro potřeby balíčku je třeba opět implementovat některé managery, jejichž rozhraní není definované knihovnou. To jsou konkrétně managery obsluhující agregaci statistik hokeje, ukládání speciálních statistik pro zápas (prodloužení a nájezdy) a bodovou konfiguraci turnaje.

Prezentační vrstva spravuje uživatelské rozhraní a interakci s uživatelem, tedy například přechody mezi jednotlivými obrazovkami, validace formulářů

atp. Pomocí rozhraní jednotlivých managerů na doménové vrstvě získává a ukládá potřebná data, která zobrazuje uživateli.



Obrázek 2.3: Komponenty balíčku

2.3 Model tříd

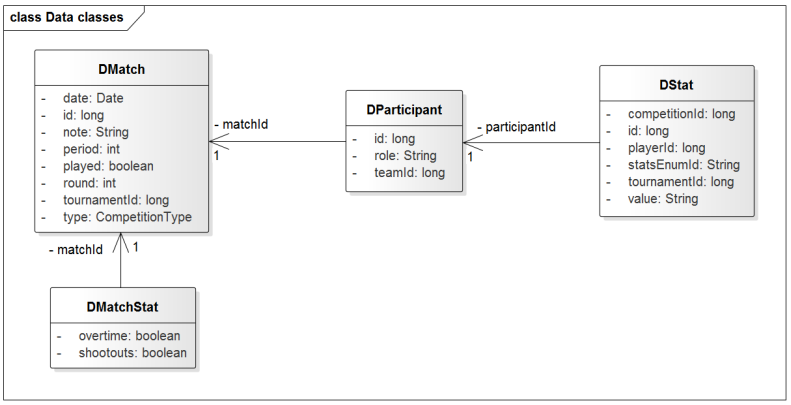
Z důvodu rozsahu nepopisuji všechny třídy v aplikaci. Pouze popisuji některá řešení, která jsou buď specifická pro balíček, nebo nebyla z doménového modelu (1.10) převedena stylem jedna ku jedné.

Třídy pro objekty byly rozděleny na dva typy. Datové a doménové třídy. Datové třídy v podstatě kopírují databázi a slouží hlavně pro uložení informací. Aby se daly jednoduše rozlišit, jsou označeny velkým D na začátku jejich názvu. V doménové vrstvě se potom tyto třídy skládají dohromady a doménové objekty se pošlou na prezentační vrstvu, která si o ně požádala, aby je mohla zobrazit.

Příkladem může být tým, který jako entita na datové vrstvě obsahuje pouze základní informace o týmu, tedy jeho název, vlastní id (identifikátor) a id turnaje. Tým jako objekt na doménové vrstvě ovšem už obsahuje i seznam hráčů, kteří do tohoto týmu patří.

2.3.1 Datová vrstva

Na obrázku 2.4 lze vidět čtyři entity na datové úrovni, pomocí kterých jsou ukládány všechny statistiky zápasu. Ty jsem vybral, protože je na nich vidět značná změna oproti doménovému modelu (1.10). Takto bylo zjednodušeno ukládání statistik, aby nebylo zbytečně moc tabulek a s tím spojených entit na datové úrovni. V následujícím textu srozumitelně popisuji význam všech entit.



Obrázek 2.4: datové entity

Entita *DStat* slouží k uložení statistik. Sama o sobě reprezentuje jednu statistiku v konkrétním zápase. Tato třída je definovaná v knihovně a jelikož je velmi obecná, je možné s ní uložit libovolný počet a typy statistik hráče v zápase a týmu v zápase, aniž by bylo potřeba ji v balíčku upravovat. Jaké typy statistik se ukládají, je pomocí výčtu (enumeration) uloženo přímo v balíčku. Hokejový enumeration obsahuje tyto hodnoty: participates, team_goals, goals, assists, plus_minus_points, outcome, interventions (zákroky – v angličtině jako *saves*, v uživatelském rozhraní také pojmenováno jako *saves*, pouze v kódu pojmenováno *interventions* z důvodu špatného překladu). Všechny tyto typy statistik kromě *team_goals* se vztahují k hráči, což znamená, že DStat s těmito typy statistik má vyplněny všechny atributy včetně playerId. Pokud chci ukládat statistiky týmu v zápase, tak se v entitě DStat pouze nevyplní playerId a nechá se nastaveno na null.

Pomocí DStat se tedy ukládají jak statistiky hráčů (v doménovém modelu 1.10 jako *Statistic record* provázaný na *MatchPlayer*), tak statistiky týmů (v doménovém modelu 1.10 jako *Statistic record* provázaný na *MatchPartici-*

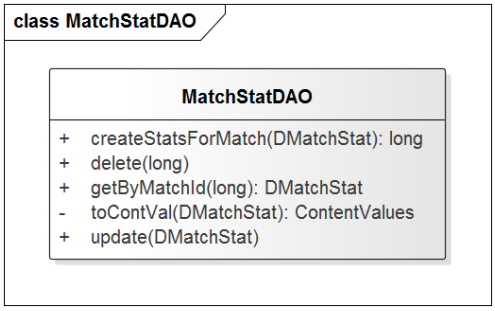
pant). Tento přístup byl v hokeji zvolen z důvodu, že statistiky hráče a týmu v zápase se v zásadě neliší. Vždy jde pouze o uložení jedné celočíselné hodnoty, a proto nemá v programu smysl mít pro tyto statistiky dvě samostatné třídy. Aby je ale stále bylo možné odlišit, tak musí z typu statistiky být zřejmé, zda se jedná o týmovou statistiku, či statistiku jednotlivce. To je ukládáno ve výčtu u každého typu statistiky pomocí příznaku *forPlayer*.

Další entitou je zde DParticipant, která kopíruje entitu z doménového modelu s názvem *MatchParticipant*. V tomto balíčku musí být ke každému zápasu vázány přesně dvě instance. Jedna, která reprezentuje domácí tým a má tedy roli nastavenou na *home*, a druhá, která reprezentuje hosty a roli má nastavenou na *away*. Zároveň je vždy navázána na existující tým v turnaji pomocí *teamId*, protože zápasu se mohou účastnit pouze platné týmy. *TeamId* musí být vždy vyplněno, jelikož hokej je týmový sport a nepřipouštíme, aby se turnaje účastnili jednotlivci.

Dále je zde knihovní entita *DMatch*, která reprezentuje zápas. Vzhledem k potřebě uložit k zápasu, zda bylo prodloužení, či dokonce nájezdy, musí být pro hokej vytvořena další entita nazvaná *DMatchStat* obsahující tyto informace. Ta je pomocí matchId navázána přímo na zápas. Informace o prodloužení a nájezdech není možné uložit pomocí DStat, protože DStat se neváže přímo na zápas. Entita tedy obsahuje atributy *overtime* a *shootouts*. Atribut *shootouts* má silnější platnost, a pokud je nastaven na true, tak bez ohledu na atribut *overtime* se považuje zápas za dokončený až v prodloužení.

Na datové vrstvě jsou DAO třídy. Ty jsou dostupné pomocí singleton třídy *DAOFactory* (třída, která má pouze jednu instanci v programu), která poskytuje instance všech DAO tříd v balíčku k využití doménové vrstvě. Tyto DAO třídy slouží pro ukládání, upravování, mazání a získávání dat z databáze. Každá DAO třída obsluhuje jednu entitu a jak již bylo řečeno, většina z nich implementuje rozhraní, která jsou definovaná v knihovně. Tato rozhraní definovaná knihovnou jsou: *ICompetitionDAO, IMatchDAO, IPackagePlayerDAO, IParticipantDAO, IStatDAO, ITeamDAO* a *ITournamentDAO*. Každé z těchto rozhraní je tedy v balíčku implementováno jednou DAO třídou. Rozhraní jsou využívána doménovou vrstvou a slouží tedy k propojení datové a doménové vrstvy.

Jednou z tříd, které jsou určeny přímo pro balíček hokeje a nemají tedy definované rozhraní knihovnou, je třída *MatchStatDAO*, kterou lze vidět na obrázku 2.5. Tato třída implementuje rozhraní *IMatchStatDAO* definované balíčkem a je specifická, protože obsluhuje databázové operace s entitou specifickou pro hokej – *DMatchStat* (viz obr. 2.4). Z názvů metod vyplývá, že nabízí možnost tuto statistiku vytvořit, upravit, smazat nebo najít dle id zápasu. Poté je zde privátní funkce „toContVal“, která musí sloužit pro převedení entity DMatchStat do speciálního formátu, využívaného databází SQLite k ukládání. Druhým rozhraním, které je definované balíčkem je *IPointConfigDAO*. Toto rozhraní je implementované třídou *PointConfigDAO* a nabízí funkce potřebné pro ukládání entity *DPointConfiguration*.



Obrázek 2.5: Třída MatchStatDAO

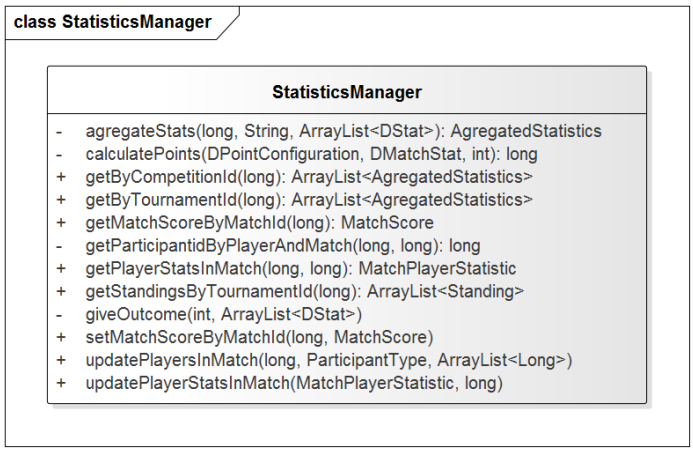
2.3.2 Doménová vrstva

V doménové vrstvě se objekty plní pomocí tříd z datové vrstvy. Tyto objekty poté již zobrazuje prezentační vrstva, a proto musí všechny implementovat `Parcelable`. Slouží to k tomu, aby v prezentační vrstvě mohly být odeslány ze služby do fragmentu.

Tyto objekty jsou vytvářeny v tzv. managerech (již byly zmíněny v popisu architektury balíčku). Managery jsou třídy, které poskytují rozhraní pro práci s doménovými objekty. Například *TeamManager* poskytuje všechny funkce nutné pro správu týmů - jejich nalezení, vytvoření, změny atd. Většina managerů implementuje rozhraní, která jsou definovaná v knihovně. Tato rozhraní definovaná knihovnou jsou: *ICompetitionManager, IPackagePlayerManager, IScoredMatchManager, ITeamManager* a *ITournamentManager*. Jedním z managerů, které neimplementují rozhraní dané knihovnou je například *StatisticsManager* implementující rozhraní *IHockeyStatisticsManager* definované balíčkem (manager viz obr. 2.6). Lze vidět, že poskytuje především funkce na práci se statistikami. Konkrétně se jedná o agregaci statistik pro soutěž či turnaj, vypočítání pořadí týmů a týmových statistik nebo o nastavování a získávání statistik konkrétního hráče v daném zápase. Tento manager pracuje s objekty statistik, které jsou navržené speciálně pro hokej a obsahují vybrané statistiky dle požadavků specifikovaných v analýze.

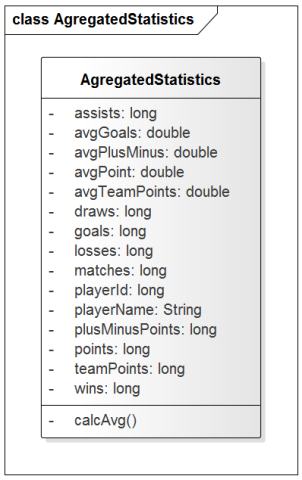
Managery jsou dostupné k využití na prezentační vrstvě opět pomocí singleton třídy *ManagerFactory*, která poskytuje instance všech managerů v balíčku. Tato třída si totiž ukládá instance tříd, které implementují všechny výše zmíněná knihovní rozhraní a rozhraní specifická pro hokej. Prezentační vrstva tedy může využívat tato rozhraní poskytovaná doménovou vrstvou.

Jako příklad doménového objektu, který nemá ekvivalent na datové vrstvě a je tedy skládán až na doménové vrstvě, uvádím objekt *AgregatedStatistics*, který lze vidět na obrázku 2.7. Tento objekt obsahuje všechny agregované statistiky, které se zobrazují v seznamu hráčů v soutěži a v turnaji. Má všechny vybrané statistiky pro hokej, které se dají vypočítat ze statistik, jež jsou za zápas ukládány. Na datové vrstvě neexistuje její ekvivalent a je vytvářena



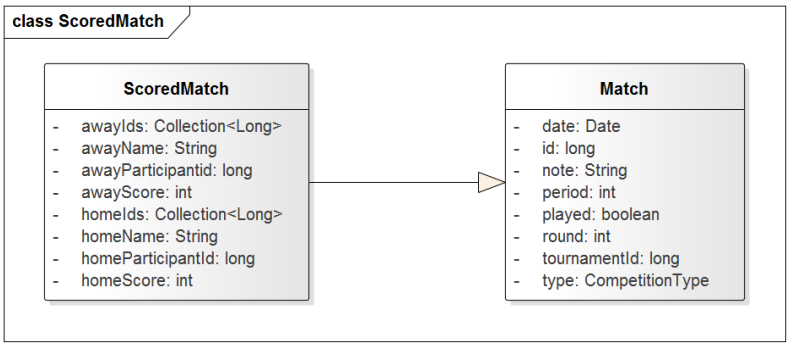
Obrázek 2.6: Třída StatisticsManager

v doménové vrstvě, konkrétně v manageru statistik, agregací ze všech statistik poskytovaných datovou vrstvou.



Obrázek 2.7: Třída AgregatedStatistics

Dále se zde vyskytuje objekt *ScoredMatch* (viz obr. 2.8), který je opět skládán až v doménové vrstvě. Jak lze vidět, tato třída dědí od třídy Match, která v podstatě kopíruje její ekvivalent na datové vrstvě. Tento objekt obsahuje vedle klasických údajů o zápase také údaje o výsledku, seznamu účastníků a účastnících se týmech. Manager dedikovaný na správu zápasů tedy funguje tak, že je-li dotázán na zápas, sestaví takovýto objekt ScoredMatch tím, že najde zápas, najde účastnící se týmy, najde skóre atd. a teprve poté předá sestavený objekt do prezentační vrstvy.



Obrázek 2.8: Třída ScoredMatch

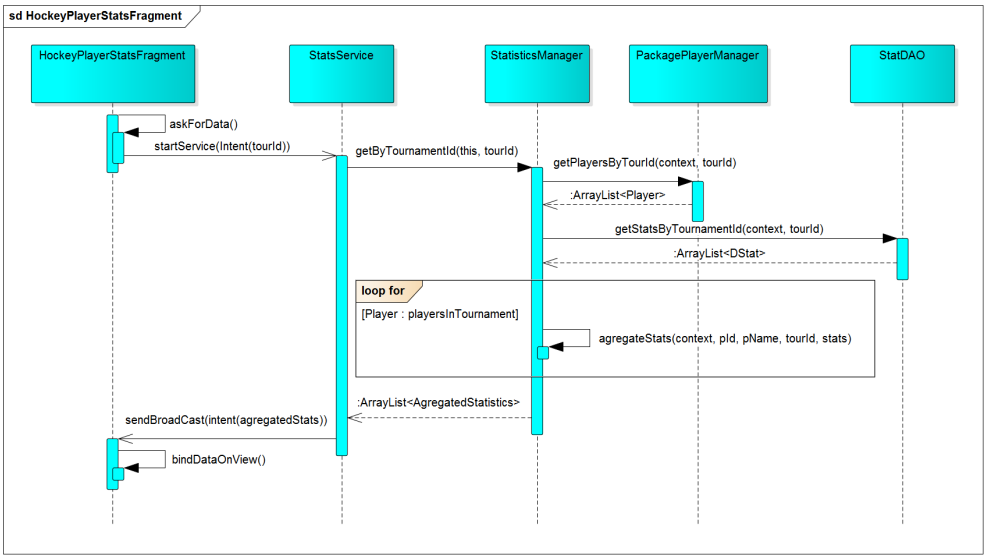
2.4 Model využití tříd

Pro znázornění již zmiňované agregace statistik jsem se rozhodl zařadit do práce sekvenční diagram, který názorně zobrazuje získání agregovaných statistik a je na něm vidět, jak probíhá komunikace skrz celou třívrstvou architekturou (viz obrázek 2.9). Pro zvýšení přehlednosti diagramu je vynechána komunikace s *factory* třídami.

Na diagramu lze vidět následující průběh: ve fragmentu pro zobrazení agregovaných statistik je zavolána funkce, aby se fragment dotázal na data. V této funkci se vytvoří intent, do něj se přidá id turnaje a spustí se jím služba. Služba podle akce definované v intentu zjistí, že se jedná o dotaz na statistiky do turnaje a získá z něj id tohoto turnaje. Získá si tedy instanci manageru pro statistiky a zavolá na něj funkci „getByTournamentId“, kde v argumentech předá id turnaje a kontext (v tomto případě je kontextem samotná služba, takže předá sama sebe).

Manager pro statistiky získá od manageru pro hráče všechny hráče v turnaji pomocí id turnaje. Poté si vyžádá instanci DAO pro statistiky a od ní získá všechny statistiky v daném turnaji.

Dále následuje iterace přes všechny hráče v turnaji, kdy se pro každého hráče agregují jeho statistiky. Do služby se tedy vrací pole všech agregovaných statistik v rámci turnaje. Služba vytvoří další Intent, do kterého přidá tyto statistiky a pomocí broadcastu (neboli rozeslání zprávy všem naslouchajícím) odešle intent. Fragment pro statistiky tento intent zachytí a identifikuje, že je určený pro něj a zavolá na sobě funkci pro zobrazení statistik. V této funkci je již zobrazí uživateli tím, že vyplní získané agregované statistiky do tabulky.



Obrázek 2.9: Sekvenční diagram agregace statistik

Kapitola 3

Realizace a testování

Tato kapitola je zaměřená na realizaci návrhu a následné testování. Nejdříve je uvedeno několik příkladů kódu a tyto příklady jsou popsány. Dále je uvedeno, jak probíhalo testování, jaké typy testování byly použity a jaké mělo toto testování výsledky. Nakonec se zmiňuji o uživatelské příručce, která byla v rámci realizace vytvořena, a o dokumentaci tříd a metod.

3.1 Realizace

Zde lze nalézt vybrané ukázky kódu z balíčku pro hokej. Mezi ukázky jsem zařadil aktivitu, která zobrazuje detail zápasu, funkci pro získání statistik v soutěži a nakonec funkci, která agreguje statistiky. Všechny tyto ukázky jsou ovšem upraveny, aby se přehledně vešly do této práce, a tedy přesný zdrojový kód lze nalézt pouze na přiloženém datovém nosiči.

Je ale také nutné zmínit, že serializace soutěží a jejich synchronizace se serverem nebyla implementována, protože rozhraní, které měla knihovna pro podporu synchronizace poskytovat, nebyla v době psaní této práce připravena.

3.1.1 Aktivita zobrazení detailu zápasu

V kódu (kód 3.1) lze vidět aktivitu `ShowMatchActivity`. Tato aktivita zobrazuje detail zápasu a dědí z knihovní třídy `AbstractTabActivity`. Tato knihovní třída je využita, protože má již nastavené zobrazování záložek, které lze využít k zobrazení výsledku zápasu a statistik hráčů do dvou různých záložek.

Třída `ShowMatchActivity` má několik privátních proměnných. Statická proměnná `MATCH_ID` slouží k uložení id zápasu do intentu v metodě `newStartIntent`. V proměnné `matchId` je uloženo id zápasu, který aktivita zobrazuje (to se získá z intentu, kterým byla aktivita nastartována). Proměnná `v` obsahuje *View* („*This class represents the basic building block for user interface components*“ [3], neboli speciální datový typ Androidu, který drží informace o rozložení uživatelského rozhraní) aktivity. Dále je zde `pager` a `adapter`.

`Pager` spravuje zobrazování jednotlivých záložek a `adapter` obsahuje všechny fragmenty, které jsou v aktivitě zobrazeny právě jako tyto záložky. Pomocí tohoto `adapteru` lze z aktivity přistupovat k zobrazovaným fragmentům.

```
public class ShowMatchActivity
extends AbstractTabActivity {
    private static final String MATCH_ID = "match_id";
    private long matchId;
    private View v;
    private ViewPager pager;
    private DefaultViewPagerAdapter adapter;

    @Override
    protected View injectView(ViewGroup parent)

    public static Intent newStartIntent(Context context,
    long matchId)

    @Override
    protected void onCreate(Bundle savedInstanceState)

    @Override
    protected PagerAdapter getAdapter
    (FragmentManager manager)

    @Override
    public boolean onCreateOptionsMenu(Menu menu)

    @Override
    public boolean onOptionsItemSelected(MenuItem item)
}
```

Zdrojový kód 3.1: Aktivita pro zobrazení zápasu

Mezi metody se zde řadí přetížená metoda `injectView`, která zde pouze do proměnné `v` přiřadí View aktivity.

Dále je zde statická metoda `newStartIntent`, která vytvoří `Intent`, pomocí kterého lze nastartovat tuto aktivitu. Zmíněnou metodu poté využívá fragment, který zobrazuje seznam zápasů. Po kliknutí na zápas s její pomocí získá intent, s nímž poté nastartuje aktivitu na zobrazení zápasu.

Metoda `onCreate`, která je volaná při vytvoření aktivity, je zde přetížená a nastavuje `pager` a `adapter`.

Další metodou je `getAdapter`, kterou je nutné implementovat pro knihovní

```
@Override
public boolean onOptionsItemSelected(MenuItem item){
  if (item.getItemId() == R.id.action_finish){
    MatchScore score = fragment1.getScore();
    if(score.isShootouts() &&
    (score.getHomeScore() == score.getAwayScore())){
      Snackbar.make(...).show(); //shows error
      return super.onOptionsItemSelected(item);
    }
    ArrayList<...> homeStats = fragment2.getHomeList();
    ArrayList<...> awayStats = fragment2.getAwayList();

    Intent intent = MatchService.newStartIntent(...);
    intent.putExtra(..., score);
    intent.putExtra(..., homeStats);
    intent.putExtra(..., awayStats);
    startService(intent);
    finish();
  } else if(item.getItemId() == R.id.action_edit) {
    fragment2.editAll();
  }
  return super.onOptionsItemSelected(item);
}
```

Zdrojový kód 3.2: Funkce onOptionsItemSelected

aktivitu, ze které tato aktivita dědí. Tato funkce vrací nový adaptér, ve kterém jsou již vytvořeny správné fragmenty, které se mají v aktivitě zobrazovat.

Metoda `onCreateOptionsMenu` se stará o vytvoření menu v pravém horním rohu aktivity.

Jako poslední je zde metoda `onOptionsItemSelected` (viz zdrojový kód 3.2 – získání fragmentů z adapteru bylo kvůli přehlednosti nahrazeno pouze slovy „fragment1“ a „fragment2“). Tato metoda je volána v případě, že bylo stisknuté některé tlačítko v menu. U této aktivity menu obsahuje dvě možnosti – uložení statistik a upravení všech statistik naráz. Zajímavý může být právě průběh uložení statistik. Jelikož jsou statistiky zobrazovány ve dvou fragmentech a je potřeba uložit je stisknutím jednoho tlačítka, nemohou být statistiky ukládány přímo z fragmentů. Proto, v případě stisku tlačítka pro uložení, je na obou fragmentech zavolána metoda, přes kterou aktivita získá z fragmentů jejich statistiky. Následně je zkompletuje, ověří, že jsou validní (to znamená, že pokud je zvolené prodloužení, není nastavená remíza) a pokud ano, tak je odešle do služby pro uložení.

Při stisknutí tlačítka pro upravení všech statistik hráčů současně se naopak

```
private agregateStats(Context context, long plId,
String pName, ArrayList<DStat> allStats)
{
  ...
}

public getByCompetitionID(Context context,long compId)
{
  ArrayList<Player> compPlayers = ManagerFactory.
  getInstance().packagePlayerManager.
  getPlayersByCompetition(context,compId);

  ArrayList<AggregatedStatistics> res =
  new ArrayList<>();

  ArrayList<DStat> competitionStats = DAOFactory.
  getInstance().statDAO.getStatsByCompetitionId(context,
  compId);

  for( Player p : compPlayers ){
    res.add( agregateStats(context, p.getId(),
    p.getName(), competitionStats) );}

  return res;
}
```

Zdrojový kód 3.3: Funkce pro získání statistik v soutěži

na fragmentu, který obsahuje statistiky hráčů, zavolá metoda `editAll()`. Ten v této metodě spustí novou aktivitu a poté, co se mu vrátí nově upravené statistiky, je nahraje do svých tabulek.

3.1.2 Získání agregovaných statistik dle id soutěže

Ve zdrojovém kódu (kód 3.3) lze vidět, jak probíhá získání agregovaných statistik pro soutěž. Funkce v tomto zdrojovém kódu se nachází ve třídě *StatisticsManager*. Pro přehlednost jsou ze zdrojového kódu odstraněny definice návratových hodnot.

Funkce `getByCompetitionId` je veřejná a je využívána službou poskytující statistiky. Tato funkce potřebuje v parametrech přijmout `Context` – to je potřeba pro předání do DAO tříd – a id soutěže. Funkce vrací seznam agregovaných statistik ve formě pole obsahujícího instance třídy `AggregatedStatistics`. Jak již bylo zmíněno v kapitole popisující návrh, tak tato třída obsahuje sta-

tistiky vždy pro jednoho konkrétního hráče. To tedy prezentační vrstvě stačí pro zobrazení tabulky s těmito statistikami, protože může pouze jednotlivé instance zobrazit v tabulce jako jednotlivé řádky.

Funkce probíhá tím způsobem, že nejdříve pomocí manageru, který je určený k obsluze hráčů, získá seznam všech hráčů, kteří jsou zaregistrovaní v soutěži. Poté od DAO třídy `StatDAO` získá všechny statistiky, které se vztahují k soutěži a následně pro každého hráče v této soutěži volá funkci `agregateStats`, která vždy získá statistiky pro tohoto hráče a vytvoří z nich jednu instanci třídy `AggregatedStatistics` obsahující vyplněné údaje, kterou v návratové hodnotě vrátí. Původní funkce všechny tyto navrácené statistiky vloží do jednoho seznamu, který poté předává zpět do prezentační vrstvy.

3.1.3 Agregace statistik

Ve zdrojovém kódu (kód 3.4) lze vidět, jak probíhá agregace statistik. Z důvodu délky kódu jsou některé části a některá klíčová slova vynechána (např `break` apod.). Zobrazená funkce je privátní ve třídě `StatisticsManager` a jejími atributy jsou: `Context`, id a jméno hráče, jehož statistiky mají být agregovány, a seznam statistik, ze kterých se má agregovat.

Funkce probíhá tak, že se nejdříve do proměnné `playerStats` vytvoří seznam pouze těch statistik, které se vztahují k hráči, a vytvoří se pomocné proměnné pro všechny statistiky, které se agregují. Tyto proměnné se při vytvoření všechny nastaví na nulu. Dále se pro každou ze statistik v seznamu vyhodnotí, o jakou statistiku se jedná – zda jsou to góly, účast, asistence, +/- body, zákroky nebo výsledek. V případě prvních pěti zmíněných statistik pouze dojde k přičtení dané hodnoty.

Pokud se ale jedná o výsledek, tak je potřeba vyřešit, kolik týmových bodů bude hráči přičteno. To se řeší způsobem, že se získá konkrétní bodová konfigurace turnaje a entita `DMatchStat`, která obsahuje informaci o tom, zda byl zápas dohrán v normálním hracím čase, v prodloužení či po nájezdech. Tyto informace se společně s výsledkem, který je reprezentovaný celým číslem vloží do funkce `calculatePoints`, která vrací počet bodů k připočtení. Ty se poté přičtou a následně se ještě podle čísla výsledku přičte výhra, prohra nebo remíza.

Poté, co jsou všechny tyto statistiky sečteny, tak funkce vrací novou instanci třídy `AggregatedStatistics`, do které vloží spočítané statistiky.

3.2 Testování

Aplikace byla testována dvěma způsoby. Pro otestování doménové a datové vrstvy byly vytvořeny Unit testy (*Jednotkové* testy), které testují managery a DAO třídy. Dále byly provedeny uživatelské testy, které měly otestovat aplikaci jako celek a zároveň práci s prezentační vrstvou. Další informace lze nalézt v následujícím textu.

```
private agregateStats(Context context,long plId,
String pName,ArrayList<DStat> allStats)
{
  ArrayList<DStat> playerStats = ...
  long matches = 0, wins = 0, draws = 0, losses = 0,
  goals = 0, assists = 0, plusMinusPoints = 0,
  teamPoints = 0, interventions = 0;
  for(DStat stat : playerStats){
    long value = Long.parseLong(stat.getValue());
    switch (StatsEnum.valueOf(stat.getStatsEnumId())){
      case goals:
        goals += value;
      case participates:
        matches++;
      case assists:
        assists += value;
      case plus_minus_points:
        plusMinusPoints += value;
      case interventions:
        interventions += value;
      case outcome:
        DPointConfiguration pointConfiguration = ...
        DMatchStat matchStat = ...

        teamPoints += calculatePoints((int)value,
        pointConfiguration,matchStat);
        switch ((int)value){
          case 1:
            wins++;
          case 2:
            draws++;
          case 3:
            losses++;
  } } } }

  return new AggregatedStatistics(plId,pName,matches,
  wins,draws,losses,goals,assists,plusMinusPoints,
  teamPoints,interventions);
}
```

Zdrojový kód 3.4: Funkce pro agregaci statistik

3.2.1 Automatické testy

Automatické testování lze na OS Android provádět buď klasickými Unit testy pro jednotlivé třídy, nebo tzv. instrumentačními testy. Tyto instrumentační testy je nutné použít v případě, kdy testovaná třída využívá různé nástroje z Android SDK (Software development kit, neboli sada vývojových nástrojů), které v Unit testech nelze využívat. Instrumentační testy musí probíhat na zařízení nebo emulátoru s OS Android, ve kterém poté lze tyto nástroje využívat.

V tomto případě nelze využít klasické Unit testy, protože managery i DAO třídy využívají nástroje z Android SDK (`Context`, `Intent`,...). Nebylo ale nutné využít přímo instrumentační testy, které trvají dlouho kvůli opakovanému zapínání emulátoru atp. Místo toho byl využit nástroj *Robolectric* [4]. Tento nástroj dokáže instrumentační testy převést na Unit testy, protože přepisuje Android SDK třídy ve chvíli, kdy jsou nahrávány, díky čemuž lze tyto testy spouštět jako klasické Unit testy („*Robolectric makes this possible by rewriting Android SDK classes as they're being loaded and making it possible for them to run on a regular JVM*“ [4]).

Byly napsány testy pro všechny managery na doménové vrstvě a všechny DAO třídy na datové vrstvě. Při testování managerů byl využit framework *Mockito*, pomocí kterého byly napodobeny DAO třídy, které jsou jimi využívány. Na datové vrstvě toto již nebylo potřeba, protože *Robolectric* zvládá vytvoření databáze při testech, a tedy DAO třídy šlo testovat bez dalších příprav.

Pro testování každého manageru a každé DAO třídy byla vždy vytvořena třída se stejným názvem a s příponou „Test“, která obsahuje testovací funkce. Tyto testy pokrývají většinu funkcí managerů i DAO tříd. Na obrázku 3.1 lze vidět, kolik bylo v každé třídě testovacích metod. Celkem se jedná o 47 testů rozdělených mezi všechny tyto třídy a všechny testy byly úspěšné.

Ve zdrojovém kódu 3.5 lze vidět ukázku jedné z testovacích tříd. Obsah samotných testů byl záměrně vynechán, protože by ukázka byla příliš dlouhá. Jak je vidět z anotace nad samotnou třídou, tak třída funguje s podporou již zmíněného nástroje *Robolectric*. Poté je zde výčet několika rozhraní, která `StatisticsManager` využívá, a proto je potřeba je napodobit. Ty mají anotace *@Mock*.

Dále zde lze vidět, že třída obsahuje metody `setUp` a `tearDown`. První metoda je volána před všemi testy a druhá je volána po nich. Metoda `setUp` obsahuje všechny úkony, které je nutné udělat před započetím testů. Mezi to se řadí vytvoření mock objektů (objekty, které napodobují jiné třídy), které se poté vloží do *factory* tříd, aby testovaný `StatisticsManager` nepřistupoval k opravdovým DAO a manager třídám. Také je zde i vytvoření nějakých testovacích dat. Metoda `tearDown` naopak obsahuje nastavení *factory* tříd do původního stavu.

Jednotlivé testovací metody mají anotaci *@Test* a jsou následující:

Test	Time el...	Usage D...	Usage ...	Usage Af...	Results
CompetitionDAOTest (fit.cvut.org.cz.hockey.data.DAO)	0,19 s	22 734 ...	275 0...	297 762...	P:2
CompetitionManagerTest (fit.cvut.org.cz.hockey.business.managers)	21s	173 01...	14 86...	187 878...	P:4
ExampleUnitTest	0 s	0 Kb	14 86...	14 863 ...	P:1
MatchDAOTest (fit.cvut.org.cz.hockey.data.DAO)	0,24 s	40 428 ...	297 7...	338 190...	P:3
MatchManagerTest (fit.cvut.org.cz.hockey.business.managers)	0,48 s	61 829 ...	187 8...	249 708...	P:2
MatchStatisticsDAOTest (fit.cvut.org.cz.hockey.data.DAO)	0,76 s	-226 90...	338 1...	111 286...	P:3
PackagePlayerDAOTest (fit.cvut.org.cz.hockey.data.DAO)	1s	42 293 ...	111 2...	153 579...	P:3
PackagePlayerManagerTest (fit.cvut.org.cz.hockey.business.managers)	0,12 s	6 327 Kb	249 7...	256 035...	P:6
ParticipantDAOTest (fit.cvut.org.cz.hockey.data.DAO)	0,36 s	44 382 ...	153 5...	197 961...	P:2
PointConfigDAOTest (fit.cvut.org.cz.hockey.data.DAO)	0,29 s	33 015 ...	197 9...	230 977...	P:1
PointConfigManagerTest (fit.cvut.org.cz.hockey.business.managers)	0,08 s	0 Kb	256 0...	256 035...	P:2
StatDAOTest (fit.cvut.org.cz.hockey.data.DAO)	0,22 s	39 118 ...	230 9...	270 095...	P:2
StatisticsManagerTest (fit.cvut.org.cz.hockey.business.managers)	0,24 s	6 329 Kb	256 0...	268 692...	P:6
TeamDAOTest (fit.cvut.org.cz.hockey.data.DAO)	0,16 s	33 015 ...	270 0...	303 110...	P:1
TeamManagerTest (fit.cvut.org.cz.hockey.business.managers)	0,11 s	0 Kb	268 6...	268 692...	P:3
TournamentDAOTest (fit.cvut.org.cz.hockey.data.DAO)	0,22 s	39 118 ...	303 1...	342 228...	P:2
TournamentManagerTest (fit.cvut.org.cz.hockey.business.managers)	0,08 s	6 335 Kb	268 6...	275 028...	P:4

Obrázek 3.1: Automatické testování

- `testGetAllAgregated` – v této metodě se vytvoří testovací statistiky a testuje se, zda metoda `getAllAgregated` vrací správný počet hráčů a u nich správné statistiky. Poté se také ověří, zda se v metodě nevolají některé nevhodné funkce na DAO třídě spravující statistiky.
- `testGetByCompId` – zde se opět připraví testovací statistiky a poté se volá metoda `getByCompetitionId`. U navráceného seznamu statistik se pak ověřuje jeho velikost a správné hodnoty statistik.
- `testGetByTourID` – tato funkce probíhá stejně jako `testByCompId`, volá se ale metoda `getByTournamentId`, místo `getByCompetitionId`.
- `testGetStandings` – zde se připraví opět nějaké testovací statistiky a zápasy a poté se zavolá funkce `getStandingsByTournamentId`. Tato funkce by měla vrátit statistiky týmů seřazené podle týmových bodů. V navráceném seznamu se tedy kontroluje počet týmů, zda jsou týmy správně seřazené a správnost některých ze statistik.
- `testMatchScoreCallsDAO` – v této metodě se ověřuje, že po zavolání funkce `getMatchScoreByMatchId` je na příslušné DAO třídě vyvolána funkce `getByMatchId`.
- `testSetMatchScore` – tato funkce ověřuje, zda byla zavolána na DAO třídě spravující statistiky zápasu funkce `update` (je nutné nastavit prodloužení a nájezdy) a zda byla na DAO třídě spravující statistiky dvakrát zavolána funkce `update` (pro nastavení počtu gólů každého z týmů).

```
@RunWith(RobolectricGradleTestRunner.class)
@Config(constants = BuildConfig.class, sdk = 21)
public class StatisticsManagerTest {
  @Mock
  IPackagePlayerManager mockPlayerManager;
  @Mock
  IStatDAO mockStatDAO;
  @Mock
  IMatchDAO mockMatchDAO;
  @Mock
  IPointConfigDAO mockPointConfigDAO;
  @Mock
  IMatchStatisticsDAO mockMatchStatisticsDAO;
  @Mock
  IParticipantDAO mockParticipantDAO;
  @Mock
  ITeamManager mockTeamManager;
  @Mock
  IScoredMatchManager mockMatchManager;

  @Before
  public void setUp() throws Exception

  @Test
  public void testGetAllAgregated() throws Exception

  @Test
  public void testGetByCompId() throws Exception

  @Test
  public void testGetByTourId() throws Exception

  @Test
  public void testGetStandings() throws Exception

  @Test
  public void testMatchScoreCallsDAO() throws Exception

  @Test
  public void testSetMatchScore() throws Exception

  @After
  public void tearDown() throws Exception
}
```

Zdrojový kód 3.5: Testovací třída StatisticsManagerTest

3.2.2 Uživatelské testy

Pro testování uživatelského rozhraní a zároveň celého balíčku bylo využito uživatelské testování. Byl vytvořen seznam testovacích případů (*Test case*), které byly vyzkoušeny na mobilních telefonech Huawei P7-L10 (verze OS Android 4.4.2) a Samsung Galaxy S4 (verze OS Android 5.0.1). Celý seznam těchto navržených testů lze nalézt v přiloženém souboru.

Testy byly rozděleny na relativně malé „jednotky“ a ty byly zařazeny do jednotlivých úrovní hierarchie aplikace (hierarchie je blíže popsána v kapitole *Analýza*). Úrovně jsou tyto:

- seznam soutěží,
- přehled soutěže,
- přehled turnaje,
- zápas.

Tento přístup umožňuje lepší pokrytí aplikace, než navržení rozsáhlého celkového průchodu. V jednotlivých vrstvách hierarchie se totiž možnosti v balíčku hodně rozdělují a jedním průchodem by bylo obtížné je dobře pokrýt. Více delších průchodů by pak způsobovalo opakované procházení stejných částí v různých testech.

V tabulce 3.1 lze nalézt ukázku jednoho testovacího případu. Tento případ se nachází na úrovni zápasu a testuje práci s úpravou zápasových rozpisek a jejich ukládání.

Tabulka 3.1: Test case - Přidání hráčů do rozpisek a odstranění

Krok	Test Case – Přidání hráčů do rozpisek a odstranění	Očekávaný výsledek
1)	V zápase přepněte na hráče	Zobrazí se seznam hráčů v zápase
2)	Stiskněte tlačítko „+“	Zobrazí se dialog
3)	Zvolte libovolný tým	Zobrazí se dostupní hráči pro přidání
4)	Vyberte libovolné hráče a potvrďte	Hráči byli přidáni do soupisky zvoleného týmu v konkrétním zápase
5)	Dlouze stiskněte hráče	Zobrazí se dialog
6)	Zvolte odstranění hráče	Hráč byl odstraněn
7)	Uložte zápas	Zobrazí se seznam zápasů
8)	Znovu otevřete zápas	Soupiska je stále stejná

Při testování takovéhoto případu se vždy provede jeden krok a zkontroluje se očekávaný výsledek. Pokud se opravdový výsledek liší od očekávaného, do tabulky se k tomuto kroku zapíše, co přesně se stalo. Pokud je očekávaný výsledek shodný s opravdovým, zapíše se, že je krok v pořádku.

Na příkladu v tabulce je vidět, že se jedná o relativně malý testovací případ, který testuje pouze jednu funkcionalitu, a nejedná se o rozsáhlý průchod, který by zahrnoval soutěže, turnaje i zápasy. Testovacích případů přibližně stejného nebo většího rozsahu bylo připraveno celkem 22. To stačí k pokrytí většiny funkcionalit balíčku.

Při uživatelském testování byla odhalena chyba, která způsobovala za specifických podmínek pád balíčku. Chyba byla následně odstraněna a při dalším uživatelském testování již proběhly všechny testy v pořádku. Lze tedy toto testování prohlásit za úspěšné jak z důvodu odhalení chyby, tak z důvodu, že balíček již projde všechny testovací případy v pořádku.

3.3 Dokumentace a uživatelská příručka

Zdrojový kód může být někdy sám o sobě obtížně pochopitelný a v aplikaci nemusí být bez návodu některé věci jasné. Ke zdokumentování balíčku byla proto vytvořena dokumentace programu (tříd a metod) a uživatelská příručka. To by mělo pomoci jak uživateli, tak vývojáři s pochopením všeho, co je potřeba.

3.3.1 Dokumentace

K vytvoření programátorské dokumentace byl využit program *Javadoc*, který umožňuje bez větších problémů vytvořit přehlednou dokumentaci celého projektu. Stačí pouze s danou strukturou komentovat funkce a třídy přímo ve zdrojovém kódu a *Javadoc* vytvoří dokumentaci v podobě stránek ve formátu HTML, které se dají jednoduše proklikat.

Ve zdrojovém kódu 3.6 lze vidět ukázku struktury těchto komentářů. V ukázce je část rozhraní `IHockeyStatisticsManager`. Je zde komentář před definicí samotné třídy, kde je napsaný její stručný popis. Dále je v kódu vidět ukázka popisu jedné funkce. K funkci lze přidat popis jejích parametrů (začíná anotací *@param*) a její návratové hodnoty (anotace *@return*). K funkci také lze přidat její stručný popis, ovšem jelikož se v tomto případě jedná pouze o rozhraní, tak je popisem návratové hodnoty a parametrů funkce definovaná dostatečně.

Tyto HTML stránky lze nalézt na přiloženém datovém nosiči.

```
/**
 * Created by atgot_000 on 3. 5. 2016.
 * Interface for manager, that works with
 * all kinds of hockey statistics
 */
public interface IHockeyStatisticsManager {

    /**
     *
     * @param context application context
     * @return aggregated statistics for all players
     * in application throughout all competitions
     */
    ArrayList<AggregatedStatistics> getAllAggregated
    (Context context);

    ...
}
```

Zdrojový kód 3.6: Ukázka dokumentace Javadoc

3.3.2 Uživatelská příručka

Pro uživatele byla vytvořena uživatelská příručka, která slouží k lepšímu pochopení a k nalezení všech možností balíčku. Příručka vychází z jednotlivých obrazovek. U každé obrazovky je popsáno, co lze na obrazovce vidět a co je možné zde provést. Jsou tam tedy popsány vždy funkce všech tlačítek. Dále jsou v příručce u jednotlivých akcí, které může uživatel provést na obrazovce, zmíněny podmínky, které daná akce vyžaduje. Například, že pro vytvoření zápasu je nutné, aby byly nejdříve vytvořeny dva týmy v turnaji atp. Také jsou v příručce popsány použité zkratky na obrazovkách se statistikami a výsledky. Příručka má 17 stran a lze ji nalézt ve formátu *pdf* na přiloženém datovém nosiči.

Závěr

V mé bakalářské práci jsem se zabýval vývojem samostatného balíčku pro hokej a příbuzné sporty do aplikace Tournament Manager. Aplikace jako celek je dělaná pro vedoucího práce, který bude jejím uživatelem, a její vývoj probíhal v týmu pěti studentů, v němž každý měl na starost její určitou část. Práce byla rozdělena do tří částí dle procesu vývoje aplikace – analytická část, část návrhu a část realizace a testování.

V analytické části jsem se zabýval zmapováním procesu odehrání jednoho turnaje, ze kterého vzešla specifikace požadavků. Poté jsem se věnoval existujícím řešením a zhodnotil jsem jejich klady a nedostatky. Nakonec jsem popsal případy užití balíčku a vytvořil doménový model pro balíček, vycházející z doménového modelu aplikace. Touto částí je splněn cíl práce zahrnující analýzu požadavků.

V návrhové části jsem nejdříve stručně shrnul použité principy technologie Android a poté jsem popsal architekturu aplikace jak z pohledu celé aplikace, tak z pohledu samotného balíčku. Následně jsem ukázal návrh několika vybraných tříd a popsal jednotlivé vrstvy architektury balíčku. Nakonec jsem popsal komunikaci skrz celou architekturu, od prezentační až k datové vrstvě. Tato část naplňuje cíl navržení architektury a seznámení se s principy jádra.

V části realizace a testování jsem ukázal různé části balíčku a ty jsem popsal. Dále jsem se zabýval jak automatickým, tak uživatelským testováním a na závěr jsem zmínil vytvoření uživatelské a programátorské dokumentace. Touto částí byly splněny cíle zahrnující implementaci balíčku, dokumentaci a vytvoření uživatelské příručky.

Vytvořený balíček splňuje specifikované požadavky a je možné ho využít i pro příbuzné sporty, i když mezi nimi nelze zatím rozlišit. Balíček maximálně využívá knihovnu a správně komunikuje s jádrem, kterému nejen poskytuje správná data, ale také je od něj získává.

Balíček nenabízí profesionální možnosti pro velké turnaje, ovšem nabízí možnost snadné správy turnajů a zápasů, což je ideální pro skupiny přátel, a tedy i pro vedoucího, na něhož byla celá aplikace cílená. Hlavní cíl práce –

vytvoření balíčku pro podporu hokeje a příbuzných sportů – byl tedy splněn.

V době psaní této práce není funkční komunikace se serverem, a tedy nelze sdílet soutěže. To je způsobeno tím, že v knihovně zatím nebyla vytvořena rozhraní, která jsou potřeba a která by balíček využíval.

Práci je možné rozšířit vypracováním synchronizace se serverem ve chvíli, kdy na to bude knihovna připravená. Také je vhodné doplnit možnost zakládat soutěže v různých sportech v tomto balíčku, aby je jádro zobrazovalo do různých seznamů. To bude možné po upravení jádra aplikace, protože toto v současnosti neumožňuje. Do budoucna by bylo také například možné upravit a rozšířit funkcionality balíčku tak, aby podporovaly i profesionálnější turnaje.

Literatura

[1] ALHIR, Sinan Si, *UML in a Nutshell* 1st ed. United States of America: O'Reilly & Associates, Inc. ©1998. ISBN 1-56592-448-7

[2] GOOGLE, Google Play, *play.google.com* [online], ©2015, [cit. 6. 12. 2015], dostupné z: <https://play.google.com/store/>

[3] GOOGLE, Android Developers, *http://developer.android.com/* [online], [cit. 6. 5. 2016], dostupné z: http://developer.android.com/index.html

[4] ROBOLECTRIC, *http://robolectric.org/* [online], [cit. 8. 5. 2016], dostupné z: http://robolectric.org/

[5] HACURA, Michal. *Tournament Manager – Synchronizace* Praha: ČVUT, 2016. Diplomová práce, ČVUT, Fakulta informačních technologií, Katedra softwarového inženýrství. Unpublished.

[6] NĚMEČEK, Josef. *Tournament Manager – Jádro aplikace* Praha: ČVUT, 2016. Diplomová práce, ČVUT, Fakulta informačních technologií, Katedra softwarového inženýrství. Unpublished.

[7] MÁCA, Vlastimil. *Tournament Manager – klient pro Windows 10 Mobile* Praha: ČVUT, 2016. Diplomová práce, ČVUT, Fakulta informačních technologií, Katedra softwarového inženýrství. Unpublished.

[8] CHMEL, Václav. *Tournament Manager – Balíček pro squash* Praha: ČVUT 2016. Bakalářská práce, ČVUT, Fakulta informačních technologií, Katedra softwarového inženýrství.

[9] POQUESOFT, Tournament Manager, In: Google, *Google Play* [online], 28. listopadu 2014, [cit. 6. 12. 2015], dostupné z: <https://play.google.com/store/apps/details?id=com.poquesoft.mistorneos>

[10] MILEYENDA ENTERTAINEMENT, Bracket Maker & Tournament App, In: Google, *Google Play* [online], 23. listopadu 2015, [cit. 6. 12. 2015], dostupné z: <https://play.google.com/store/apps/details?id=com.mileyenda.manager>

[11] CYGNUS SOFTWARE, Tournament Manager, In: Google, *Google Play* [online], 19. listopadu 2015, [cit. 6. 12. 2015], dostupné z: <https://play.google.com/store/apps/details?id=eu.uvdb.entertainment.tournamentmanager>

[12] KOINEMA SRL, *Konkuri*, [online], ©2009-2016 [cit. 12. 5. 2016] dostupné z: <http://www.konkuri.com/>

[13] STILLER, Evelyn, LEBLANC, Cathie; *Project-Based Software Engineering: An Object-Oriented Approach*, 1st ed. United States of America: Addison-Wesley, ©2002. ISBN 0-201-74225-X

[14] OTERO, Carlos E., *Software Engineering Design: Theory and Practice*. 1st ed. United States of America: Taylor & Francis Group, ©2012. ISBN 978-1-4398-5168-5

[15] LEE, Wei-Meng, *Android Application Development* 1st ed. United States of America, Indianapolis: Wiley Publishing, Inc. ©2011. ISBN 978-1-118-01711-1

[16] MURPHY, Mark L., *Android 2: Průvodce programováním mobilních aplikací*, Vydání první, Brno: Computer press, a. s., Czech edition compyright ©2011. ISBN 978-80-251-3194-7

[17] ČESKÝ SVAZ LEDNÍHO HOKEJE. *Pravidla ledního hokeje 2014 - 2018* [online]. 2014, [cit. 12. 5. 2016] dostupné z: <http://www.cslh.cz/text/119-pravidla-ledniho-hokeje.html>

[18] INTERNATIONAL STREET & BALL HOCKEY FEDERATION. *OFICIÁLNÍ PRAVIDLA 2008*. [online] Přeložil Českomoravský svaz hokejbalu. vyd. roku 2008 [cit. 12. 5. 2016] dostupné z: <http://plzenlitice.klubweb.cz/document/download/id/2>

[19] ČESKÁ FLORBALOVÁ UNIE. *Pravidla florbalu a jejich výklad*. [online]. 2014. [cit. 12. 5. 2016] dostupné z: <https://www.ceskyflorbal.cz/cfbu/predpisy/pravidla-florbalu>

Příloha A

Seznam použitých zkratek

API Application programming interface

DAO Data access object

id Identifikátor

JVM Java virtual machine

OS Operační systém

SDK Software development kit

Test case Testovací případ

UML Universal markup language

Unit test Jednotkový test

XML Extensible Markup Language

Příloha **B**

Obsah přiloženého CD

- `src` ..... adresář se zdrojovými kódy
- `apk` ..... adresář s instalovatelným balíčkem a prototypem jádra
- `thesis`
  - `BP_Ondrej_Kosut.pdf` ..... text práce ve formátu PDF
  - `BP_Ondrej_Kosut.tex` ..... zdrojová forma práce ve formátu LaTeX
- `others`
  - `documentation.zip` ..... dokumentace
  - `use_cases.docx` ..... podrobnější specifikace některých případů užití
  - `user_guide.pdf` ..... uživatelská příručka ve formátu PDF
  - `test_cases.xlsx` ..... testovací případy pro uživatelské testování