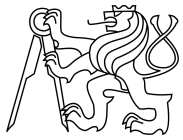
ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ



ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Název:	Tournament Manager - balíček pro squash
Student:	Václav Chmel
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 squashe 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 squashe.

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

Seznam odborné literatury

Dodá vedoucí práce.

L.S.

Ing. Michal Valenta, Ph.D.
vedoucí katedry

prof. Ing. Pavel Tvrdík, CSc.
děkan

V Praze dne 5. prosince 2015

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

Bakalářská práce

Tournament Manager - balíček pro squash

Václav Chmel

Vedoucí práce: Ing. Zdeněk Rybola

17. května 2016

Poděkování

Děkuji svému vedoucímu práce panu Ing. Zdeňkovi Rybolovi za věcné a podnětné připomínky při vedení 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 17. května 2016 .....

Odkaz na tuto práci

Chmel, Václav. *Tournament Manager - balíček pro squash.* Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2016.

Abstrakt

Tato práce popisuje analýzu, návrh, implementaci a testování balíčku pro squash do aplikace Tournament Manager pro operační systém Android. Tento balíček umožňuje uživateli zaznamenávání výsledků turnajů a soutěží ve squashi a monitorování s nimi spojených statistik.

Klíčová slova soutěž, turnaj, zápas, squash, zaznamenávání výsledků, statistiky, Tournament Manager, Android

Abstract

This thesis describes analysis, design, implementation and testing of squash package for Android application Tournament Manager. This package allows user to keep track of results of squash tournaments and competitons and to keep track of statistics that are valid for them.

Keywords competition, tournament, squash, result tracking, statistics, Tournament Manager, Android

Obsah

Seznam obrázků

Seznam ukázek kódu

*

Úvod

Squash je sport, který vznikl v polovině 19. století v Anglii.[1] Zpravidla se hraje ve dvou lidech, ale varianta čtyřhry je také možná. Hráči se pohybují v uzavřeném kurtu (místnosti), střídají se v odpalování míče pomocí pálek a snaží se, aby míč trefil přední stěnu mezi vymezenými čarami.

Existuje mnoho hráčů, nejen amatérských, kteří hrají squash pravidelně a mají zájem mezi sebou pořádat ligy, turnaje nebo jen zápasy. Systém do kterého by si mohli zaznamenávat výsledky a statistiky, bude proto velmi užitečný. V současnosti žádný ucelený systém pro pomoc hráčům neexistuje, případná stávající řešení nejsou vyhovující.

Praxe ukazuje, že by bylo užitečné, zadávat požadované údaje již během zápasu nebo těsně po jeho ukončení. Z tohoto důvodu těžko využijeme standardní počítač. Mnohem vhodnější řešení tedy přichází v podobě aplikace využitelné na mobilním zařízení. Tyto požadavky částečně splňuje aplikace pro operační systém Android s názvem Tournament Manager.

Cíle práce jsou zaměřeny na využití zkušeností a postřehů z výše jmenované aplikace, navržení balíčku doplněného o chybějící funkcionalitu pro squash do aplikace nově vznikající. Tato nová verze se bude skládat z jádra a aplikačních balíčků. Jádro bude spravovat balíčky, které budou zajišťovat funkcionalitu pro jednotlivé sporty. Balíčky budou samostatné aplikace a budou se také samostatně instalovat. Nebudou ovšem samostatně fungovat.

Cíle práce

Práce má za úkol splnit následující cíle:

- Seznámit se s jádrem aplikace Tournament Manager.
- Analyzovat požadavky na organizaci turnajů a evidenci výsledků ve squashi.

- Navrhnout architekturu a řešení balíčku pro squash.
- Implementovat dle provedeného návrhu balíček implementovat.
- Balíček otestovat.
- Vytvořit dokumentaci implementovaného řešení společně s uživatelskou příručkou.

Kapitola 1

Analýza

Tato kapitola se zabývá analýzou balíčku pro squash. Specifikujeme si názvosloví pro snadnější orientaci v problematice. Popíšeme základní business proces použití aplikace Tournament Manager a jejího balíčku pro squash. Poté uděláme rešerši existujících řešení. Specifikujeme si požadavky, které balíček bude splňovat. Popíšeme případy užití a závěrem se podíváme na doménový model aplikace.

1.1 Názvosloví

Ačkoli se může zdát, že sportovní názvosloví, které se zde používá, je na první pohled jednoduché, není tomu tak. Proto si popíšeme jednotlivé pojmy.

Soutěž. Soutěží se rozumí nejvyšší úroveň klasifikace opakujících se sportovních událostí (např. 1. fotbalová liga, extraliga ledního hokeje, Vaše liga ve squashi, apod.). Stejně tak je možné využít soutěž jako seriál dílčích událostí, např. pravidelná sportovní setkání kamarádů, apod. V rámci soutěže se nevyhodnocují výsledky, pořadí účastníků, apod. Sledují se však celkové statistiky jednotlivých účastníků. Soutěž se skládá z turnajů a patří do konkrétního sportu.

Turnaj. Turnaj představuje konkrétní sportovní událost, během které se sledují výsledky jednotlivých zápasů, vyhodnocuje se úspěšnost a určuje se vítěz a další pořadí. Např. konkrétní sezóna fotbalové či hokejové ligy, kolo v rámci Vaší ligy, nebo třeba konkrétní setkání sportovců formou turnaje.

Zápas. Zápas je jedno sportovní utkání dvou účastníků, které má výsledek.

Účastník. Účastníkem turnaje může být tým složený z jednotlivých hráčů, nebo přímo jednotliví hráči jako jedinci.

Kolo turnaje. Kolo je sled zápasů, každý účastník turnaje hraje s každým dalším účastníkem právě jednou. Kolo je organizováno do period. Příkladem kola je podzimní část fotbalové ligy.

Perioda turnaje. Perioda je sled zápasů v jednom kole, kdy každý účastník hraje jeden zápas. Příkladem je jeden hrací víkend ve florbalové lize.

1.2 Analýza statistik

Tato sekce zahrnuje námi zvolené statistiky, které budeme monitorovat ve squashi.

Squash je sport, který se hraje primárně na zápasy, budeme tedy monitorovat výsledek zápasu. Výsledek může mít tři formy: výhra, prohra, remíza. Budeme sledovat procentuální úspěšnost vyhraných zápasů. Squash hraje na sety. Budeme tedy monitorovat vyhrané a prohrané sety, které ve squashi udávají skóre. Dále budeme monitorovat průměrný počet vyhraných a prohraných setů na zápas a procentuální úspěšnost vyhraných setů. Tyto statistiky mohou uživatele zajímat z hlediska jejich individuální výkonnosti. Hráč vyhraje set, pokud dosáhne určitého počtu vyhraných míčků (zpravidla 15).Z toho vyplývá že budeme muset monitorovat i počet vyhraných a prohraných míčků. Stejně jako u setů ještě přidáme průměry počtu vyhraných a prohraných míčků na zápas.

1.3 Business proces užití aplikace

Business proces je aktivita nebo sousled aktivit sloužící k dosažení určitého cíle [2]. V softwarovém inženýrství se modelování business procesů používá k lepšímu pochopení problematiky a porozumění mezi dodavatelem a zákazníkem. Proces se většinou popisuje pomocí UML diagramu aktivit.

Nyní si popíšeme základní business proces užití aplikace a jejího squashového balíčku. Proces je jednoduchý a přímočarý, bude tedy stačit jeho slovní popis. Budeme z něj vycházet při zkoumání existujících řešení a dále pak ve specifikaci požadavků na balíček.

Dejme tomu, že uživatel bude chtít hrát amatérskou squashovou ligu s názvem Vaše liga. Proces začíná tím, že si uživatel v aplikaci vytvoří soutěž s výše uvedeným názvem.

1. Každý měsíc probíhá jedno kolo v 4-6 lidech.
2. Uživatel v soutěži vytvoří turnaj s názvem pro dané kolo.
3. Uživatel do soutěže přidá nové soupeře.
4. Uživatel přidá sebe a soupeře do turnaje pro dané kolo.

5. Uživatel odehraje zápas s některým ze soupeřů.

6. Uživatel tento zápas vytvoří v aplikaci, vyplní výsledek a uloží.

7. Uživatel opakuje kroky 5. a 6. dokud neodehraje všechny svoje zápasy v turnaji.

1.4 Rešerše existujících řešení

V této sekci se zaměříme na prozkoumání existujících aplikací. Za prvé na existující prototyp aplikace Tournament Manager a dále na ostatní aplikace, které najdeme na Google Play - platforma s aplikacemi na operační systém Android.

1.4.1 Prototyp aplikace Tournament Manager

Prototyp byl poskytnut vedoucím práce. Umožňuje vytvářet skupiny. Do skupiny se dají přidat hráči a lze zde organizovat turnaje. V zápasech je možno vidět statistiky a skóre turnajů, rozpis her a zúčastněné týmy. Po výběru nebo kliknutí na zápas lze zadat výsledky (ale nikoli statistiky). Negativa současné aplikace spočívají v nemožnosti rozlišení typu sportů. Zadávané statistiky jsou stavěné pro hokejbal.

Prototyp je nevyhovující, protože uživatel sice může vytvářet turnaje, ale zadávání výsledků a monitorované statistiky se už pro squash nehodí. Rozšíření prototypu s možností podpory více sportů by bylo složitější, než vytvoření nové aplikace. Tímto řešením se tedy zabývat nebudeme.

Prototyp posloužil jako dobrý nástroj pro pochopení problematiky.

1.4.2 Aplikace na Google Play

V této sekci se zaměříme na prozkoumání aplikací na Google Play. Budeme prohledávat aplikace, které jsou zdarma nejprve podle klíčového slova „tournament“ a poté podle klíčového slova „squash“.

The Tournament Manager. V aplikaci nelze podle našeho názvosloví založit soutěž. Top-level položka je turnaj. Lze založit turnaj několika typů. Co se týče vedení statistik, je možné do zápasu přidat pouze body a v tabulce se pak zobrazuje počet odehraných zápasů, body, počet výher, proher, remíz skóre nebo poměr setů a obdržené body. Pro squash se tedy tato aplikace nehodí. Další nevýhodou je, že maximum hráčů v turnaji je 32. Aplikace neumožňuje zakládat týmy. Odkaz: (3.5.2016)`https://play.google.com/store/apps/details?id=rockets.thetournamentmanager`.

Tournament Manager. Aplikace opět neumožňuje založit soutěž, ale jen turnaj. Účastníci mohou být pouze jednotlivci a po vytvoření zápasů je již nelze měnit. Skóre zápasu lze určit jen pomocí bodů. Pro squash tedy opět nevhodné. Aplikace sice umožňuje mezi sebou turnaje sdílet, ale nefunguje bez přístupu k internetu. Odkaz: (3.5.2016) `https://play.google.com/store/apps/details?id=com.poquesoft.mistorneos`.

Bracket Maker & Tournament App. Aplikace tentokrát umožňuje založit soutěž i turnaj. Soutěž může být týmová i pro jednotlivce. Po pochopení trochu neintuitivního ovládání lze do zápasu zadat výsledek ve formě setů, což je pro squash velice příjemné. Kromě toho, že některé statistiky se zobrazují v placené verzi, má aplikace opět velkou nevýhodu - nutnost přístupu k internetu. Pokud má uživatel přístup k internetu a je ochoten připlatit, není pro něj tato aplikace špatnou volbou. Bohužel ještě dnes existuje skupina uživatelů, kteří internet v mobilu nemají, a my je chceme zahrnout do cílové skupiny pro naši aplikaci. Odkaz: (3.5.2016) `https://play.google.com/store/apps\/details?id=com.mileyenda.manager`.

Squash Scorer. Jak už název napovídá, aplikace umožňuje pouze vytvářet zápasy a zadávat do nich výsledky. Úroveň soutěže a turnaje zde zcela chybí. Stejně jako náhled všech odehraných zápasů. Veškerá funkcionalita jako třeba zápas dvou týmů je obsažena až v placené verzi. Odkaz: (3.5.2016) `https://play.google.com/store/apps/details?id=squashscorer.medhurstt`.

1.4.3 Závěr rešerše

V rámci rešerše jsme se dozvěděli, že existují aplikace dvou typů:

1. Aplikace umožňující generování různých forem turnajů.
2. Aplikace umožňující ukládání statistik squashových zápasů.

Pomineme-li, že u 1. typu není dodržená hierarchie soutěž - turnaj - zápas, stejně neexistuje aplikace, která by oba typy vhodně kombinovala a nevyžadovala připojení k internetu. V následující části práce tedy vytvoříme vlastní aplikaci.

1.5 Cílová vize systému

Jak již bylo zmíněno výše, nová aplikace Tournament Manager se bude skládat z jádra a balíčků pro jednotlivé sporty. Také bude mít k dispozici server, na který budou moci uživatelé uploadovat soutěže a tím je mezi sebou sdílet. Problematikou serveru a sdílení se v této práci nebudeme vůbec zabývat.

Princip jádra a balíčků vymyslel Jakub Nižaradze ve svojí bakalářské práci *Mobchar - jádro aplikace*[3]. Jádro agreguje data, která mu balíčky poskytují. V případě Nižaradzeho jádro zobrazuje seznam postav hry na hrdiny napříč jednotlivými hrami. V našem případě jádro bude zobrazovat seznam všech soutěží pro jednotlivě sporty (balíčky). Po výběru soutěže, jádro předá řízení balíčku, tzn. detail soutěže a veškeré další obrazovky jsou už v režii konkrétního balíčku. Jádro aplikace Tournament Manager navíc bude řešit správu hráčů a synchronizaci soutěží se serverem. Problematice jádra se v detailu věnuje Josef Němeček ve svojí diplomové práci Tournament Manager - jádro aplikace [4].

Dalším prvkem této struktury je knihovní projekt tzv. „Knihovna“. Knihovna obsahuje společné prvky pro vývoj balíčků a je obsažena jak v jádře, tak v balíčcích. Původní záměr - ulehčit vývoj balíčků však nebyl naplněn a to vzhledem k tomu, že knihovna vznikala během této bakalářské práce, a na jejím vývoji se ještě podíleli právě Josef Němeček a také Ondřej Košut, jako součást své bakalářské práce *Tournament Manager - balíček pro hokej* [5].

1.6 Analýza požadavků

V této sekci se budeme zabývat analýzou požadavků pro náš balíček resp. aplikaci. Nejprve si vysvětlíme, co to požadavky jsou, k čemu jsou dobré a jak se dělí. Následně definujeme, jaké požadavky bude splňovat naše aplikace.

Podle [6] je softwarový požadavek popis konkrétní funkcionality, kterou bude cílový systém poskytovat nebo naopak omezení, které je na systém kladeno. Požadavky se často dělí na:

- Funkční - specifikují, jaké funkcionality by systém měl poskytovat, jak by se měl chovat k určitým vstupům nebo v určitých situacích a jaké by měl poskytovat výstupy.
- Nefunkční - specifikují omezení na funkce poskytované systémem nebo omezení na systém jako celek.

1.6.1 Funkční požadavky

Byly identifikovány následující funkční požadavky.

F1 Systém bude evidovat soutěže. Soutěž má svůj název. K soutěži je možné připojit poznámku pro bližší přiblížení soutěže. Také je možné evidovat období konání dané soutěže. V rámci soutěže bude možné určit, zda se jedná o soutěž týmů nebo jednotlivců. Soutěž nebude mít definovaný sport. Ten je definován příslušností k balíčku a v našem případě to bude automaticky squash.

F2 Systém bude evidovat turnaje. Pro turnaj je třeba evidovat jeho jméno a termín konání. Tento termín může být jeden konkrétní den (turnaj) nebo také celé období (sezóna ligy). Dále je možné evidovat poznámku pro bližší určení turnaje či jeho okolností. Každý turnaj je vždy evidován v rámci některé soutěže. Mezi jednotlivými turnaji v rámci jedné soutěže nejsou žádné přímé vazby s výjimkou agregování celkových soutěžních statistik.

F3 Systém bude umožňovat evidenci účastníků jednotlivých turnajů. Účastníkem turnaje může být tým složený z jednotlivých hráčů nebo přímo jednotliví hráči jako jedinci. Typ účastníků (tým/jednotlivci) je dán soutěží, ve které se turnaj koná. Tj. není možné kombinovat v jedné soutěži turnaje jednotlivců a týmů. Pro turnaj jednotlivců bude možné vybrat mezi hráči registrovanými do daného turnaje. Pro turnaj týmů bude možné vytvořit týmy platné pro daný turnaj.

F4 Systém bude umožňovat evidenci složení jednotlivých týmů. Tým je složený z jednotlivých hráčů registrovaných na úrovni turnaje. Soupisku týmu bude možné editovat manuálně - ručním výběrem hráčů. To umožní přidat do týmu nové hráče nebo stávající odebrat. Změna složení týmu ovlivní pouze budoucí zápasy. Změna nesmí ovlivnit již zaznamenané výsledky zápasů i osobní statistiky.

F5 Systém bude umožňovat evidenci jednotlivých zápasů v rámci turnaje. Každý zápas vždy probíhá mezi právě dvěma účastníky turnaje. V rámci zápasu se eviduje datum konání a výsledek. Výsledek může být zpočátku nevyplněn, pokud dosud nedošlo k odehrání. U zápasu se také může evidovat poznámka pro bližší určení zápasu. Systém musí umožnit editaci výsledku i po jeho prvním zaznamenání. Systém musí umožnit vymazat výsledek zápasu. Pak je zápas považován za neodehraný a je možné jej znovu vyplnit.

F6 Systém bude umožňovat přípravu rozlosování jednotlivých zápasů v rámci turnaje. Systém bude umožňovat přidávat zápasy do turnaje manuálně. Uživatel bude moci přidat jednotlivý zápas určením obou soupeřících účastníků (včetně určení domácího a hostujícího). Systém bude umožňovat automatické generování rozpisu zápasů formou každý s každým. Systém na přání uživatele vygeneruje jedno kolo turnaje, během kterého odehraje každý účastník jeden zápas s každým z dalších účastníků. Všechny vygenerované zápasy mají nevyplněný výsledek. Uživatel může kdykoli vygenerovat další kolo. To bude navazovat na předchozí, ve smyslu prohození domácích a hostujících účastníků. Systém bude umožňovat odstranit zápas z rozpisu. Systém by měl vizuálně odlišit jednotlivé periody a kola.

F7 Systém umožní přiřazovat registrované hráče do soutěží. Přiřazení hráčů do soutěže bude možné až po jejím založení. Systém si bude pamatovat pouze, kteří hráči jsou v jaké soutěži.

F8 Systém umožní přiřazovat hráče registrované v soutěži do jednotlivých turnajů v dané soutěži. Přiřazení hráče do turnaje bude možné až po jeho založení. Systém si bude pamatovat jen, kteří hráči jsou registrovaní v jakém turnaji.

F9 Systém bude evidovat osobní statistiky hráčů v rámci zápasů. Budou to vyhrané a prohrané sety a vyhrané a prohrané míčky. V případě týmů budou statistiky platné pro všechny účastníky daného zápasu.

F10 Systém bude evidovat agregované statistiky hráčů v rámci turnaje a soutěže. Podle provedené analýzy to budou: vyhrané a prohrané sety, vyhrané a prohrané míčky, počet výher, proher, remíz, průměrný počet vyhraných a prohraných setů za zápas, průměrný počet vyhraných a prohraných míčků za zápas, zápasová úspěšnost v % (poměr vyhraných a prohraných zápasů), úspěšnost vyhraných setů.

F11 Systém bude umožňovat konfiguraci bodového ohodnocení výher, proher a remíz.

F12 Systém poskytne jádru informace o uchovaných soutěžích.

F13 Systém poskytne jádru agregované statistiky pro konkrétního hráče napříč všemi soutěžemi.

1.6.2 Nefunkční požadavky

Byly identifikovány následující nefunkční požadavky:

N1 Aplikace bude řešená v Androidu.

N2 Aplikace bude fungovat i bez připojení k internetu.

N3 Aplikace bude informace o hráčích získávat z jádra.

N4 Aplikace nebude spustitelná samostatně, ale jen za pomocí jádra.

1.7 Případy užití

V následující sekci si popíšeme nejprve aktéry případů užití a pak i samotné případy užití, které vycházejí ze specifikovaných požadavků. Případy užití jsou poněkud přímočaré, proto je nebudeme popisovat pomocí scénářů, ale jen je shrneme v několika větách.

1.7.1 Aktéři

V našem případě budou naší aplikaci využívat dva aktéři. První je jádro, které se bude dotazovat na určitá data. Další pak už je fyzický uživatel, který bude využívat všechny ostatní funkce aplikace.

1.7.2 Případy

Následující sekce popisuje případy užití. Případy užití pro jádro můžeme vidět na diagramu 1.1. Případy užití pro uživatele jsou rozděleny z hlediska funkčních celků tj. případy týkající se soutěže diagram 1.2, týkající se turnaje 1.3, týkající se zápasu 1.6, týkající se týmů 1.4, týkající se hráčů 1.5.

1.7.2.1 Případy týkající se jádra

Získání listu všech soutěží. Jádro si zažádá o seznam všech soutěží uložených v balíčku. Balíček požadavek obslouží a vrátí požadovaná data. Způsob komunikace bude diskutován v následující kapitole.

Získání agregovaných statistik pro konkrétního napříč soutěžemi. Jádro si zažádá o statistiky pro konkrétního uživatele napříč všemi soutěžemi uloženými v balíčku. Balíček je napočítá a přeloží do formátu, aby je jádro mohlo dynamicky zobrazit. Potom je vrátí zpět do jádra.

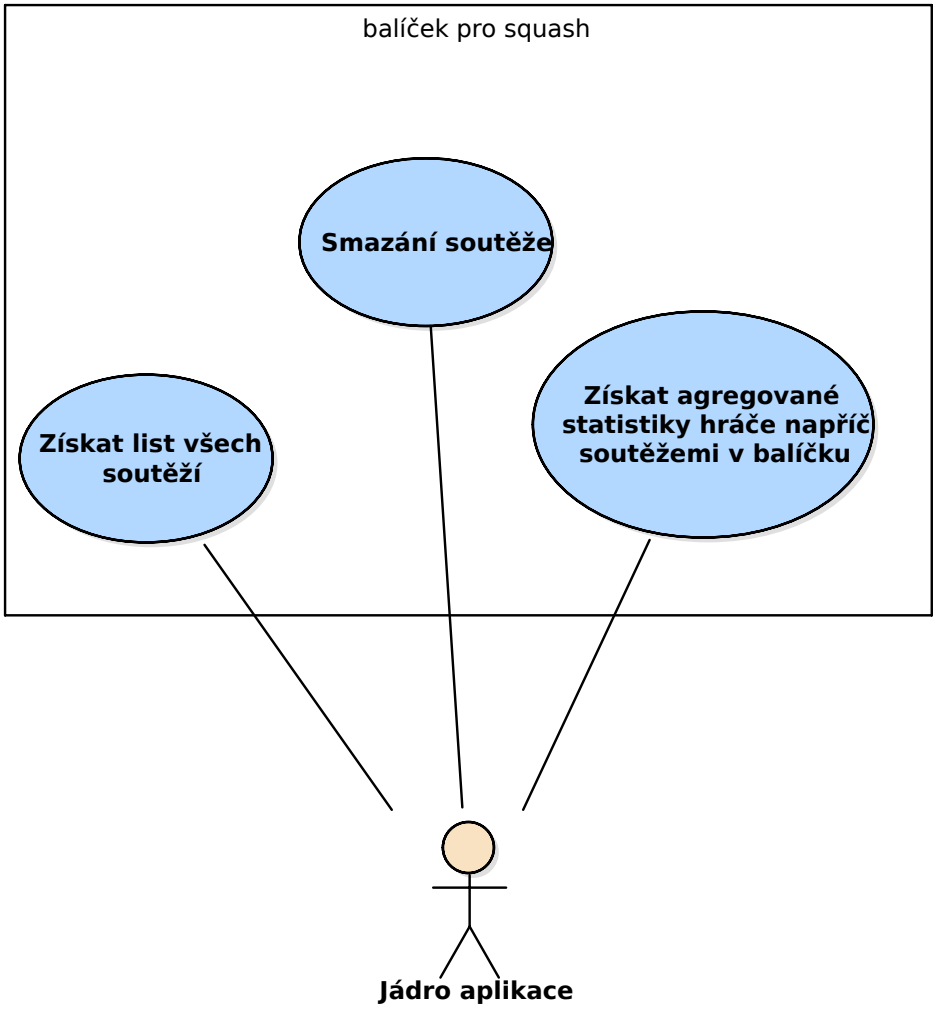
Smazání soutěže. Jádro zašle požadavek na smazání určité soutěže uložené v balíčku. Balíček se pokusí soutěž smazat. Soutěž lze smazat, pokud neobsahuje žádné hráče a turnaje. Výsledek vrátí zpět do jádra. Způsob komunikace bude diskutován v následující kapitole.

1.7.2.2 Případy týkající se soutěže

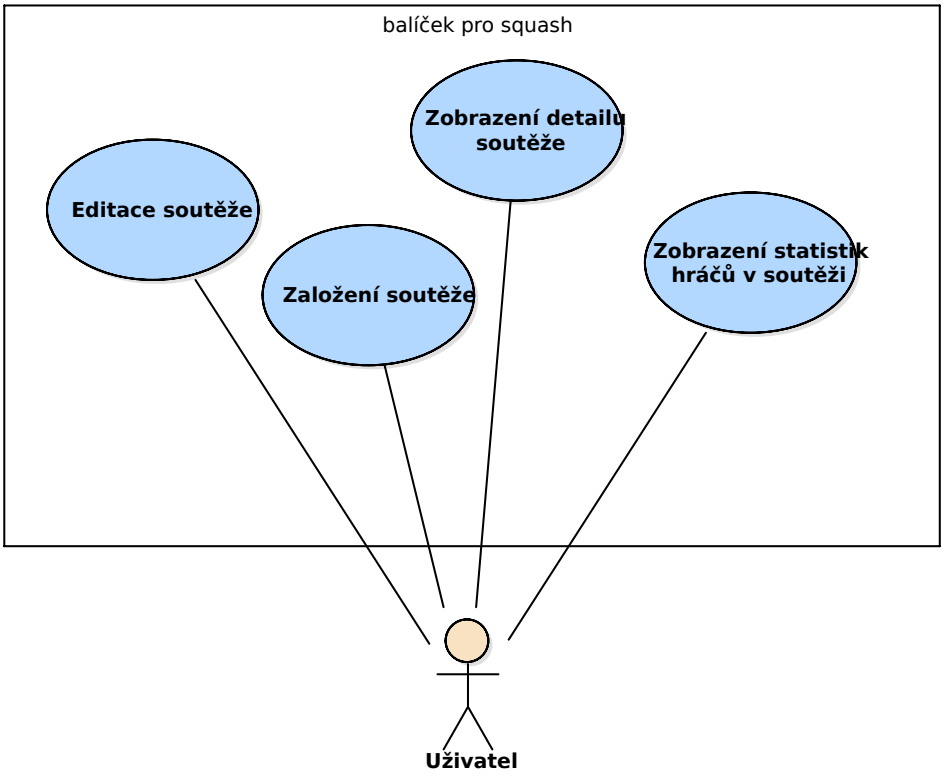
Založení soutěže. Po kliknutí na tlačítko přidat soutěž (nachází se v jádře) aplikace zobrazí formulář. Uživatel ho vyplní a zvolí uložit soutěž, aplikace zkontroluje platnost jména a poté buď zobrazí chybovou hlášku, nebo soutěž uloží.

Editace soutěže. Uživatel zvolí editovat soutěž. Aplikace mu zobrazí formulář s předvyplněnými daty ze založení soutěže. Pak již případ pokračuje jako v případě založení soutěže.

Zobrazení detailu soutěže. Uživatel se dostane na obrazovku s detailem soutěže. Aplikace mu detail zobrazí.



Obrázek 1.1: Diagram případů užití jádra.



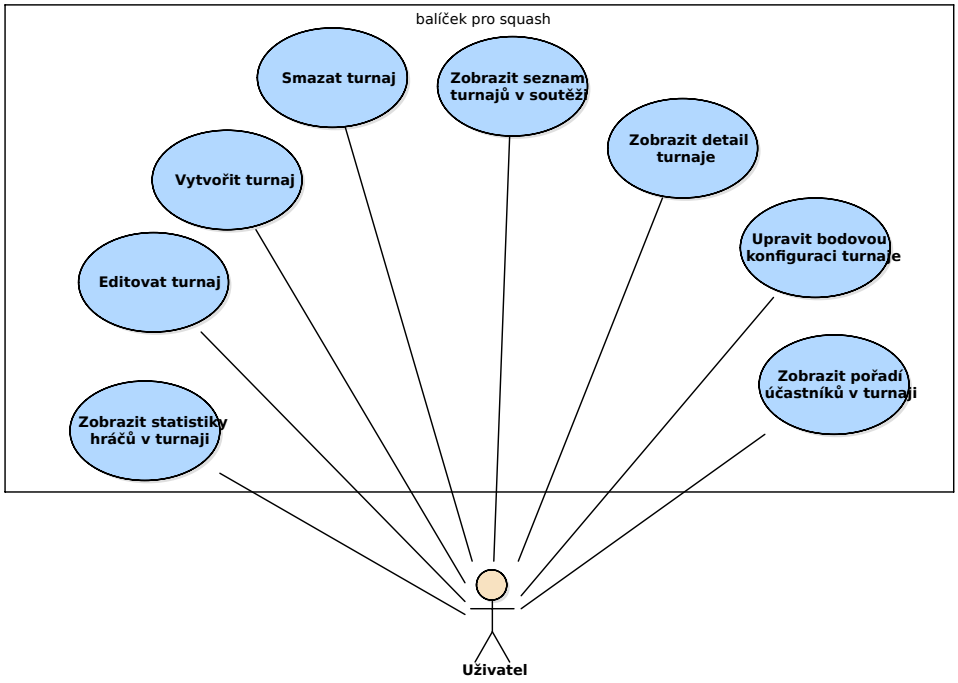
Obrázek 1.2: Diagram případů užití z pohledu soutěže.

Zobrazení statistik hráčů v soutěži. Uživatel se dostane na danou obrazovku. Aplikace zobrazí všechny hráče, kteří jsou v soutěži registrovaní a k nim napočítané statistiky napříč všemi turnaji, kterých se hráči účastní. Statistiky vychází z předchozí analýzy a liší se podle polohy zařízení. V případě, že se zařízení nachází ve svislé poloze (portrét), je zobrazen pro každého hráče počet výher, proher a remíz. Otočí-li se zařízení do vodorovné polohy (landscape), zobrazí se všechny statistiky.

1.7.2.3 Případy týkající se turnaje

Zobrazit statistiky hráčů v turnaji. Uživatel se dostane na danou obrazovku. Aplikace mu zobrazí seznam hráčů registrovaných v turnaji a k nim napočítané statistiky z daného turnaje. Statistiky jsou stejné jako u soutěže a zobrazují se stejným způsobem.

Vytvořit turnaj. Uživatel zvolí vytvořit turnaj. Aplikace mu zobrazí formulář a uživatel ho vyplní a následně zvolí volbu uložit. Aplikace zkontroluje vkládané údaje, pokud je vše pořádku, turnaj uloží. V opačném případě



Obrázek 1.3: Diagram případů užití z pohledu turnaje.

upozorní uživatele na případné nedostatky. Při uložení turnaje k němu uloží výchozí bodovou konfiguraci.

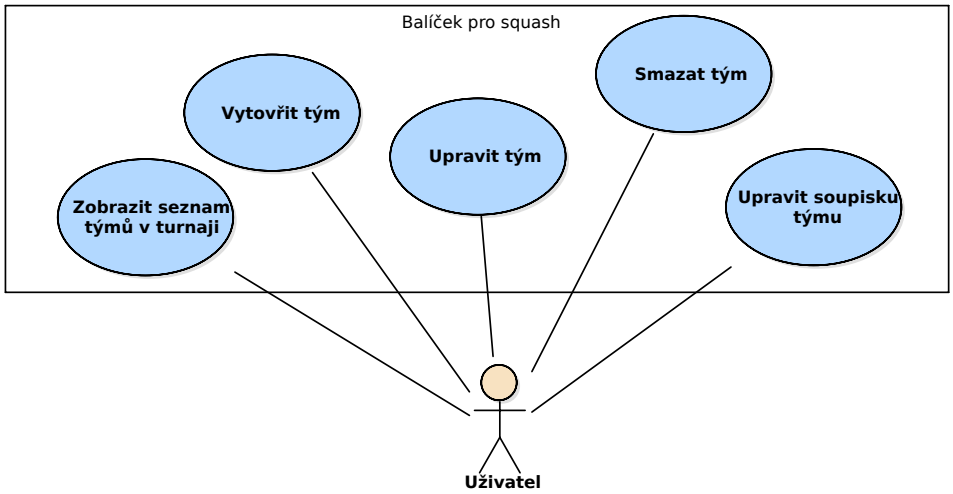
Editovat turnaj. Uživatel zvolí editovat turnaj. Aplikace mu zobrazí formulář pro editaci údajů turnaje s již předvyplněnými daty. Pak už případ pokračuje jako při vytvoření.

Smazat turnaj. Uživatel zvolí smazat turnaj. Turnaj lze smazat, pokud neobsahuje žádné hráče, týmy a zápasy. Aplikace se pokusí turnaj smazat, v případě neúspěchu informuje uživatele.

Zobrazit seznam všech turnajů v soutěži. Uživatel se dostane na danou obrazovku. Aplikace mu turnaje zobrazí.

Zobrazit detail turnaje. Uživatel se dostane na danou obrazovku. Aplikace mu detail turnaje zobrazí.

Změnit bodovou konfiguraci turnaje. Uživatel zvolí upravit bodovou konfiguraci turnaje. Aplikace mu zobrazí formulář s daty konfigurace uložené k turnaji. Uživatel má možnost data změnit, poté zvolí uložit a aplikace konfiguraci uloží.



Obrázek 1.4: Diagram případů užití z pohledu týmů.

Zobrazit pořadí účastníků v turnaji. Uživatel se dostane na danou obrazovku. Pokud je realizována soutěž jednotlivců, aplikace mu zobrazí seznam hráčů v turnaji, jinak seznam týmů. Seznam je zobrazen sestupně nejprve podle počtu bodů a poté podle rozdílu vyhraných a prohraných setů. V seznamu je název účastníka, počet bodů, výher, proher a remíz a jako poslední skóre, což je počet vyhraných setů ku počtu prohraných setů.

1.7.2.4 Případy týkající se týmů

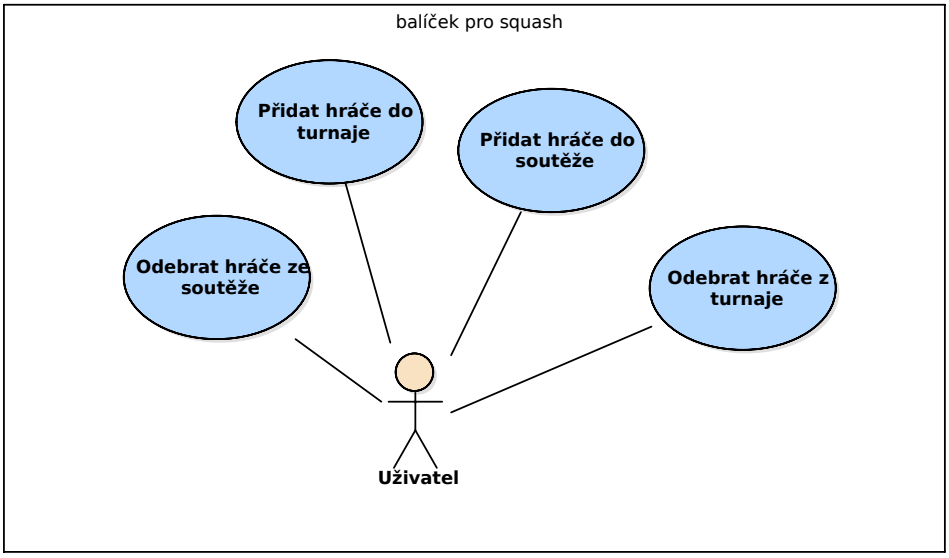
Zobrazit seznam týmů v turnaji. Uživatel se dostane na danou obrazovku, obrazovka je dostupná pouze v případě, že se jedná soutěž týmů. Aplikace mu zobrazí seznam týmů v turnaji a ke každému tým jeho hráče.

Vytvořit tým. Uživatel zvolí vytvořit tým. Aplikace mu zobrazí dialog na zadání názvu týmu. Uživatel ho vyplní, zvolí uložit a aplikace vytvoří nový tým.

Upravit tým. Uživatel zvolí upravit tým. Aplikace mu zobrazí dialog s předvyplněným názvem týmu. Pak už případ pokračuje, jako při vytvoření.

Smazat tým. Uživatel zvolí smazat tým. Tým lze smazat, pokud neobsahuje žádné hráče a nevyskytuje se v žádném turnaji. Aplikace se pokusí tým smazat, v případě neúspěchu informuje uživatele.

Upravit soupisku týmu. Uživatel otevře danou obrazovku. Zde má zobrazen aktuální soupisku týmu. Má zde možnost hráče odebírat a přidávat. Pro



Obrázek 1.5: Diagram případů užití z pohledu hráčů.

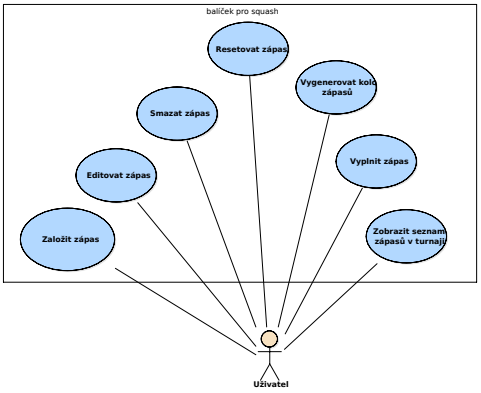
přidání hráčů, zvolí možnost přidat hráče. Aplikace mu zobrazí seznam hráčů, kteří existují v seznamu daného turnaje a nejsou v žádném týmu nebo patří do týmu, jehož soupiska je upravována. Aplikace v nabídce vynechá všechny hráče, kteří se již nachází na soupisce. Poté uživatel vybere hráče z této nabídky a zvolí uložit. Aplikace hráče uloží do aktuálně upravované soupisky. Když je uživatel se soupiskou hotov, zvolí uložit a aplikace nahradí soupisku týmu novou soupiskou.

1.7.2.5 Případy týkající se hráčů

Odebrat hráče ze soutěže. Uživatel zvolí odebrat hráče ze soutěže. Hráče lze odebrat, pokud se neúčastní žádného turnaje. Aplikace se pokusí hráče odebrat, v případě neúspěchu informuje uživatele.

Odebrat hráče z turnaje. Uživatel zvolí odebrat hráče z turnaje. Hráče lze odebrat, pokud se neúčastní žádného zápasu a není na soupisce žádného týmu. Aplikace se pokusí hráče odebrat, v případě neúspěchu informuje uživatele.

Přidat hráče do soutěže. Uživatel zvolí přidat hráče do soutěže. Aplikace mu zobrazí seznam všech hráčů uložených v jádře bez hráčů, co jsou již v soutěži. Uživatel vybere hráče, které chce přidat a zvolí uložit. Aplikace přidá hráče do soutěže.



Obrázek 1.6: Diagram případů užití z pohledu zápasů.

Přidat hráče do turnaje. Uživatel zvolí přidat hráče do turnaje. Aplikace mu zobrazí seznam všech hráčů registrovaných v soutěži bez hráčů, co jsou již v turnaji. Uživatel vybere hráče, které přidat a zvolí uložit. Aplikace přidá hráče do turnaje.

1.7.2.6 Případy týkající se zápasů

Přidat zápas. Uživatel zvolí přidat zápas. Aplikace zkontroluje, že turnaj obsahuje aspoň dva účastníky. Pokud ne, informuje uživatele, pokud ano zobrazí uživateli formulář. Uživatel ho vyplní a zvolí uložit. Aplikace zkontroluje, že je vyplněné kolo a perioda a účastníci zápasu se liší. Pokud je vše v pořádku zápas uloží, jinak informuje uživatele.

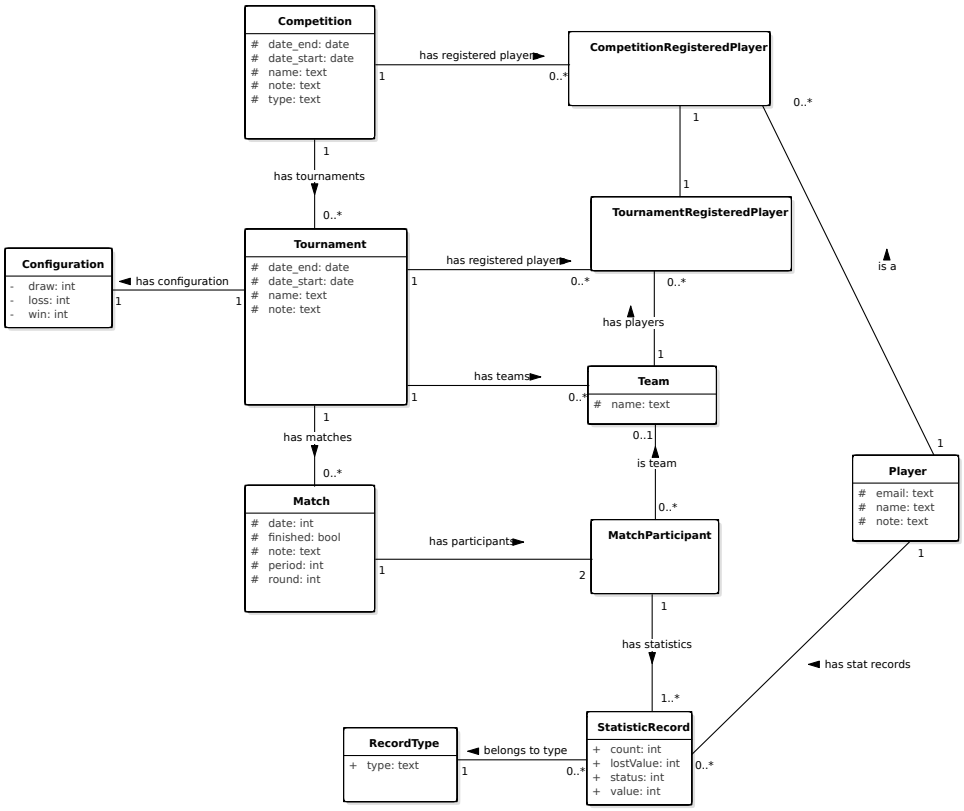
Editovat zápas. Uživatel zvolí editovat zápas. Aplikace mu zobrazí formulář s předvyplněnými daty. Dále případ pokračuje od zobrazení formuláře při vytvoření.

Smazat zápas. Uživatel zvolí smazat zápas. Na smazání zápasu nejsou kladena žádná omezení, proto aplikace zápas smaže.

Resetovat zápas. Uživatel zvolí resetovat zápas. Aplikace ho resetuje, tj. smaže jeho statistiky a soupisku a nastaví ho jako neodehraný.

Vygenerovat kolo zápasů. Uživatel zvolí vygenerovat kolo zápasů. Aplikace zkontroluje, že má turnaj aspoň dva účastníky. Pokud ne, informuje uživatele, jinak vytvoří nové kolo zápasů viz.sekce 1.1 Názvosloví.

Vyplnit zápas. Uživatel zvolí vyplnit zápas. Zde má k dispozici obrazovku, na které může přidávat a odebírat odehrané sety a měnit jejich skóre. Pokud



Obrázek 1.7: Doménový model.

se jedná o soutěž týmů má k dispozici další obrazovku, na které může měnit soupisku týmů pro daný zápas. Soupiska se mění stejně jako při změně soupisky týmů s tím rozdílem, že uživatel má na výběr pouze z hráčů, patřících aktuálně do daného týmu. Poté zvolí uložit a zápas se nastaví jako odehraný. Sety se uloží a soupiska jen pokud je dispozici, se uloží.

1.8 Doménový model

V této sekci je představen doménový model, který zachycuje vztahy mezi jednotlivými entitami s pohledu balíčku pro squash.

Competition. Tato entita představuje soutěž. Má svoje metadata dle požadavků, tedy název, datum počátku a konce, poznámku a typ. Typ specifikuje, zda se jedná o soutěž týmů, či jednotlivců. Soutěž má svoje registrované hráče. Skládá se z turnajů.

Tournament. Tato entita představuje turnaj. Má svá metadata dle požadavků, tedy název, datum počátku a konce a poznámku. Turnaj má opět svoje registrované hráče. Je to podmnožina hráčů registrovaných v turnaji. Turnaj si drží svojí konfiguraci bodů. Pokud turnaj patří do týmové soutěže, drží si přehled týmů. Turnaj se skládá z jednotlivých zápasů.

Configuration. Konfigurace určuje, kolik bodů obdrží účastník za výhru, prohru a remízu a obsahuje právě tyto tři položky.

Match. Tato entita představuje zápas. U zápasu je nutné evidovat datum, zda byl odehrán, poznámku, periodu a kolo. Zápas má vždy právě dva účastníky.

CompetitionRegisteredPlayer. Tato entita představuje hráče registrovaného do soutěže.

TournamentRegisteredPlayer. Tato entita představuje hráče registrovaného do turnaje.

Team. Tato entita představuje tým. U týmu je potřeba evidovat pouze název. Tým se skládá z hráčů registrovaných do turnaje a jeden takto registrovaný hráč může být pouze v jednom týmu.

MatchParticipant. Tato entita představuje účastníka zápasu. Účastník může být buď hráč, nebo tým. Toto je dané typem soutěže.

StatisticRecord. Tato entita představuje jednu konkrétní statistiku. Statistika má svůj typ. Co statistika ukládá se odvíjí od typu. Může to být výsledek zápasu, set, nebo účast v zápase. Statistiky se vždy váží na účastníka, účast v zápase se navíc ještě váže na konkrétního hráče.

RecordType. Tato entita představuje typ statistiky. Předpokládá se, že budou tři typy statistik. První bude Set. Ten ukládá výsledek počet vyhraných a prohraných míčků a výsledek setu pro jednoho účastníka. Další bude Zápas. Ten ukládá výsledek zápasu pro jednoho účastníka. Poslední bude Účast v zápase, ta se bude vázat na konkrétního hráče a ukládá, zda se účastnil zápasu nebo ne.

Player. Tato entita představuje hráče. Hráč má svůj název.

1.9 Shrnutí

V této kapitole jsme si představili základní business proces použití aplikace. Provedli jsme rešerši existujících řešení a zjistili jsme, že nám ani jedna aplikace nevyhovuje. Představili jsme si aplikaci Tournament Manager jako celek. Identifikovali jsme funkční a nefunkční požadavky na aplikaci našeho balíčku pro squash. Vydefinovali si jednotlivé případy užití, které mohou nastat a závěrem představili doménový model.

Kapitola 2

Návrh

Tato kapitola se zabývá návrhovou částí vývoje balíčku pro squash. Nejprve si představíme určitá specifika OS Android, která při tomto návrhu musíme vzít v úvahu. Následovat bude návrh architektury aplikace. Potom si uděláme úvod do některých komponent OS Android, které jsou nezbytné pro pochopení aplikace. Dále se podíváme na model tříd rozdělený do jednotlivých balíčků podle vrstev aplikace a jejich funkcionality. Představíme si databázový model a na závěr si popíšeme jakým způsobem mezi sebou jednotlivé vrstvy komunikují pomocí sekvenčního diagramu.

2.1 Specifika OS Android

Zde si popíšeme některá specifika OS Android, která musíme brát v úvahu při návrhu cílového systému.

Nevytěžovat příliš hlavní vlákno aplikace. Jakákoliv aplikace v Androidu běží na hlavním vlákně, jinak také nazývaném UI vlákně. Android je na toto vlákno velice citlivý, což znamená, že by se mělo vytěžovat jen nezbytně nutnou prací, tj. vázání dat na UI a prací s UI. Jakákoliv další práce (získávání dat, náročnější výpočty atd.) by měla probíhat v jiném vlákně. Pokud se hlavní vlákno příliš vytíží, zobrazí aplikace uživateli dialog, který ho informuje, že aplikace neodpovídá. Následně mu nabídne, zda si přeje počkat, nebo chce aplikaci ukončit. Není vhodné, aby toto uživatel někdy musel podstoupit. Pro naší aplikaci z toho vyplývá, že veškerá pracovní logika bude probíhat v jiných vláknech a výsledek se zašle do UI vlákna, kde se bude prezentovat uživateli. K tomu nám poslouží některé komponenty OS, které si popíšeme v sekci 2.3.

Snažit se efektivně používat dostupné zdroje. OS Android se používá pro různé formy mobilních zařízení (mobilní telefony, tablety apod.). Přestože

už dnes přestává platit, že tato zařízení nemají k dispozici stejné zdroje (výkon procesoru, velikost paměti) jako počítače, Android se stále snaží optimalizovat použití těchto zdrojů. My bychom tento nedostatek měli brát v potaz při vývoji naší aplikace.

Jelikož v naší aplikaci mají entity, podle doménového modelu z minulé kapitoly, velmi složitou strukturu a veliký stupeň zanoření (soutěž obsahuje turnaje, turnaje obsahují zápasy atd.), nebyla by úplně vhodná strategie nahrát si do paměti celou takovouto strukturu dat. Měli bychom využít lazy-loading. [1] Budeme-li brát v potaz předchozí odstavec, požadavky na plnění lazy-loadovaných referencí musí probíhat mimo hlavní vlákno. Většina našich operací spočívá v získávání dat z úložiště a jejich zobrazení uživateli. Kdybychom lazy-loading implementovali, probíhalo by to většinou následovně:

1. Entita zjistí, že má prázdnou referenci, kterou potřebuje. Následně na vedlejší vlákno pošle požadavek na určitá data.

2. Požadavek se zpracuje a výsledná data se uloží do entity.

3. Entita upozorní UI vlákno, že má připravená data a může je zobrazit uživateli.

Ačkoliv se tak na první pohled nemusí zdát, je tento proces zbytečně komplikovaný. Můžeme ho zjednodušit tím, že entitu z tohoto procesu úplně vyřadíme. UI bude tedy rovnou zasílat požadavky na data na vedlejší vlákno. To je bude zpracovávat a výsledná data zasílat zpět na UI vlákno. Z tohoto nám plyne, že pro naši aplikaci bude tedy výhodné a pro nás implementačně mnohem jednoduší, pokud vztahy mezi entitami nebudou a UI bude posílat požadavky na data, která bude potřebovat. Popsaný postup zohledníme při návrhu architektury pro naši aplikaci.

2.2 Architektura

V této sekci si definujeme možnosti architektury pro Android aplikace a navrhneme řešení pro náš balíček.

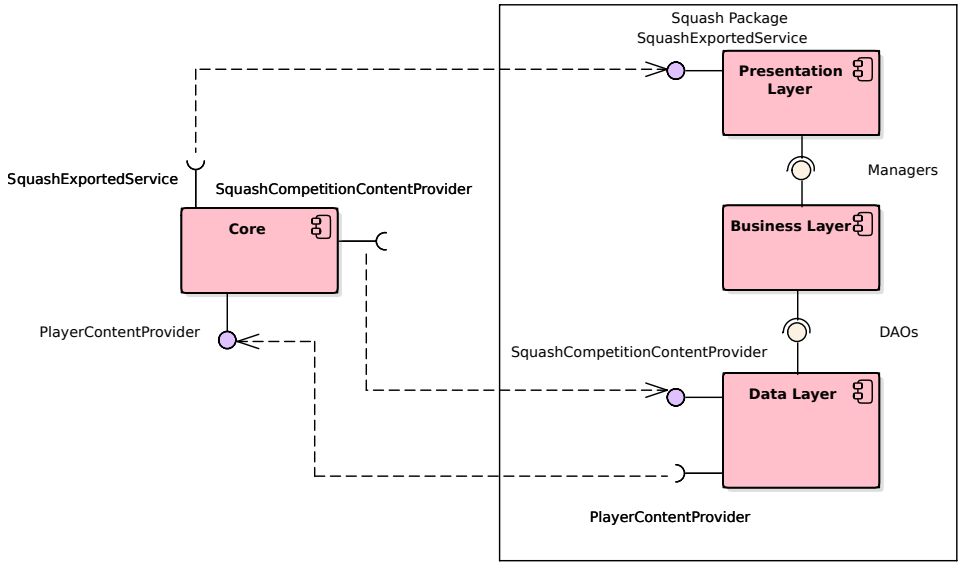
V případě Android aplikace máme dvě zásadní možnosti na zvolení architektury:

1. Jednovrstvou.

2. Vícevrstvou.

Jednovrstvá architektura se hodí pro rychlý vývoj jednoduchých aplikací. Zmiňujeme ji proto, že operační systém Android obsahuje prvky, které vývoj

[1] Třída si sice drží referenci na určitý objekt, ta je ale prázdná a naplní se, až když je potřeba.



Obrázek 2.1: Diagram komponetnt systému.

takových aplikací výrazně usnadňují, avšak ve vícevrstvé architektuře je nebude možno využít. Patří mezi ně například `CursorLoader`, který umožňuje získat data z databáze a zobrazit je uživateli. Systém se postará o asynchronnost a znovu-nahrání dat s minimálním přičiněním od developera.

Bohužel naše doména specifikovaná v předchozí kapitole je komplexní, pokud bychom zvolili jednovrstvou architekturu, byla by aplikace velice nepřehledná. Také by docházelo ke špatné údržbě a složitému nebo nemožnému rozšiřování. Aplikaci si proto rozdělíme do tří vrstev: prezentační, business a datové viz obrázek 2.1. Na obrázku můžeme také vidět komponenty pro komunikaci s jádrem, to bude vysvětleno v sekci 2.4. Existující manažery a DAO objekty budou vyobrazeny v rámci návrhu tříd v sekci 2.5.

Prezentační vrstva. Prezentační vrstva má za úkol předkládat data uživateli a zajišťovat jeho vstupy. V našem případě se bude prezentační vrstva skládat téměř výhradně z komponent Androidu. Nutné komponenty si popíšeme v sekci 2.3.

Business vrstva. Business vrstva má za úkol realizovat business logiku aplikace. Jak jsme si představili v sekci 2.1, entitní třídy nebudou navzájem propojeny, tzn. budou jen zřídka obsahovat reference na další entitní třídy. Například entita pro soutěž nebude mít v sobě uloženy všechny svoje turnaje. V našem zvoleném řešení budou manažery obsahovat vhodné metody, a to tak, aby se celá struktura dala poskládat dohromady. Entity budou tedy propojené přes tyto manažery. Příklad: budeme chtít získat turnaje dané soutěže, zavoláme

tedy příslušnou metodu třídy `CompetitionManager` a ona nám je vrátí. Hlavní úkol business vrstvy bude správně spravovat entity a vztahy mezi nimi, dále dopočítávat statistiky.

Datová vrstva. Datová vrstva má na starosti persistenci dat. Její hlavní úkol bude spravovat data ve vestavěné SQLite databázi zařízení.

2.3 Komponenty OS Android

V této sekci si uděláme úvod do některých komponent (tříd) tohoto systému, který bude nutný pro pochopení chodu systému o osobami, které OS Android neznají.

2.3.1 Fragment

Jedním ze základních stavebních prvků téměř každé aplikace pro OS Android je Fragment. Podle [7] je fragment komponent, který zabaluje určité chování nebo část uživatelského rozhraní nebo obojí. Fragment je akomodován v aktivitě viz níže. Fragment podléhá životnímu cyklu[2], který řídí OS Android. Diagram tohoto cyklu je k dispozici v příloze C na straně 55.

Příkladem fragmentu v naší aplikaci je *AggregatedStatsListFragment*, který zobrazuje uživateli seznam hráčů v turnaji nebo soutěži. Tento fragment v sobě zaobaluje část UI, protože určuje jakým způsobem se mají data zobrazit. Zaobaluje také chování dotazování na data v momentě, kdy je to zapotřebí.

2.3.2 Aktivita

Dalším stavebním prvkem je Aktivita. Podle [8] Aktivita reprezentuje jednu obrazovku, kde může uživatel vykonávat určitou činnost. Aktivita má svoje UI. Vždy, při spuštění další aktivity, se otevře nové okno. Aktivita v sobě může akomodovat fragmenty. Příkladem z naší aplikace je
`CompetitionDetailActivity`, která reprezentuje aktivitu detailu soutěže a obsahuje v sobě tři fragmenty. První zobrazuje metadata soutěže, druhý seznam turnajů v soutěži a třetí seznam hráčů v soutěži a jejich statistiky. Aktivita může být exportovaná, proto jí mohou pouštět ostatní aplikace. Aktivita má svůj vlastní životní cyklus. Jeho diagram lze nalézt v příloze C na straně pagereffig:aclife.

Většina věcí, které zvládne fragment, se dá udělat na úrovni aktivity. Použití fragmentů je ale nanejvýš vhodné a doporučované ze dvou hlavních důvodů. Za prvé použití fragmentů podporuje znovu použitelnost jejich zapouzdřené funkcionality, protože je jednoduše můžeme přidávat do libovolných ak-

[2] Životní cyklus je posloupnost metod, vyvolávaná OS v závislosti na viditelnosti daného prvku.

tivit. Za druhé toto řešení umožňuje vytvářet UI pro různá zařízení, tzn. aplikace může mít rozdílné UI pro mobilní telefon a tablet. Proto v naší aplikaci je drtivá většina funkcionality na prezentační vrstvě zapouzdřená do fragmentů a jednotlivé aktivity pak tyto fragmenty obsahují. To, že aktivita obsahuje pouze jeden fragment, vůbec ničemu neškodí.

2.3.3 Service, IntentService

Další často používanou komponentou je Service (služba). Služba slouží k dlouhotrvající práci na pozadí bez nutnosti interakce s uživatelem, a to i do té míry, že běží i po tom, co uživatel opustí aplikaci [9]. Služba také může být exportovaná, tj. mohou ji využít ostatní aplikace.

My budeme používat pro práci na pozadí třídu `IntentService`, potomka třídy Service. Tato třída za nás už obstarává veškeré věci týkající se vytváření vláken. Jedná se o singleton[3], který obsluhuje požadavky sekvenčně. Tzn. požadavek čeká, než se vyřídí předchozí požadavek. Toto řešení nám nebude vadit, jelikož budeme mít několik služeb. Naše požadavky se budou dotazovat jen do databáze. Budou rychlé a v jednu chvíli na jednu službu budou maximálně v řádu jednotek.

2.3.4 Intent

Intent je abstraktní systémová zpráva, která určuje jaká operace se má v OS provést [10]. Pomocí intentu se startují aktivity a služby. Do intentu se dají přibalit data. Další jeho využití je tedy pro zasílání dat mezi komponentami. Představíme si oba základní typy intentů a jejich využití v naší aplikaci.

Explicitní: Explicitní intenty posíláme přímo daným komponentám. Komponenty jsou určené svojí třídou. V našem případě se takto startují aktivity a služby. K tomu, aby mohl být komponentě poslán explicitní intent, musí na ní být vložen odkaz v Android manifestu.

Implicitní: Implicitní intenty se posílají komponentám zaregistrovaným v systému určitou sadou kvalifikátorů. Toto má vícenásobné využití. Například:

- Posílání intentů ostatním aplikacím.
- Chceme dát uživateli na výběr aplikaci, co intent zpracuje.
- Chceme intent doručit více komponentám najednou.
- Chceme intent doručit ve správný čas (toto bude náš případ).

[3]Třída, která má v aplikaci pouze jednu instanci.

V našem případě budeme používat jediný kvalifikátor, a to akci. Jedná se o unikátní řetězec napříč všemi aplikacemi. Často obsahuje název balíčku aplikace. Takto poslaný intent obdrží všechny komponenty, které jsou pro danou akci zaregistrované. V naší aplikaci se takto posílají intenty s výslednými daty v momentě, kdy služby dokončí svůj úkol.

2.3.5 BroadcastReceiver

Broadcast Receiver je jednoduchá komponenta, která obdrží intent a vykoná funkci, za jejímž účelem byla vytvořená. V našem případě tyto receivery obsahují jednotlivé fragmenty a zachytávají data vrácena ze služeb, aby je fragment mohl zobrazit uživateli.

2.3.6 Context

Context je v Android aplikaci jedna z nejpoužívanějších věcí. Tato věc se také velice špatně vysvětluje. Podle oficiálních stránek je context interface pro přístup ke globálním informacím o aplikaci [11]. Tento interface už implementují některé ostatní komponenty. Pro nás jsou důležité Aktivita a Služba. Context dovoluje provádět různé operace např. startování, aktivit a služeb, přístup k resources, dotazy na content provider (viz níže) a přístup k databázi.

O contextu se zmiňujeme, protože context je potřeba pro přístup k databázi, která se nachází v datové vrstvě. V té už žádné aktivity a služby nejsou, to je věc prezentační vrstvy a musíme tam tedy context umístit. Na context by se neměla držet reference, pokud už není potřeba, protože by se mohlo stát, že v době jejího užití již context není platný, a to by mohlo způsobit pád aplikace. Nejčistší řešení bude context dát jako parametr do business a do DAO metod. Toto je důvod, proč context je téměř ve všech metodách spodních vrstev naší aplikace.

2.3.7 ContentProvider

Content provider je komponenta, která slouží pro přístup datům a umožňuje sdílet data napříč aplikacemi. Content provider poskytuje data ve stejném formátu, obdobně jako SQLite databáze, proto jsme ho umístili na datovou vrstvu.

Mezi metody content provideru patří `insert`, `update`, `delete` a `query`. Tyto metody mají podobné parametry jako SQLite databáze. Content provider tyto parametry může a nemusí respektovat. Záleží na implementaci. Pokud je respektuje, tázající aplikace má větší volnost ve formulaci dotazů a manipulaci s daty, které má cílový provider na starosti. Content provider se většinou dotazuje databáze. Content provider má specifickou URI, na kterou směřují operace na tento provider. Operace jsou pak rozlišené pomocí této URI, použitých metod a jejich parametrů.

2.4 Komunikace jádro-balíček

V této sekci si v rychlosti shrneme komunikaci s jádrem.

Jak můžeme vidět na obrázku 2.1, jádro poskytuje balíčku komponentu `PlayerContentProvider`. Ta umožňuje balíčku získat informace o uložených hráčích v jádře. Tak je splněn požadavek, že hráči se získávají z jádra. Na oplátku balíček poskytuje jádru komponentu `SquashCompetitionContentProvider`. Ta umožňuje jádru získat informace o uložených soutěžích v balíčku. Tyto dva komunikační kanály získávají data přímo z databáze a nevyžadují žádné jejich zpracování. Ze jmenovaných důvodů mohou být tedy řešené pomocí content provideru. Balíček dále poskytuje jádru službu `SquashExportedService`. Ta umožňuje jádru, jednak danou soutěž smazat a dále získat statistiky hráče nasčítané přes všechny soutěže v balíčku. Toto nelze vyřešit pomocí content provideru, protože mazání nelze provádět za všech okolností a statistiky nejsou v databázi uložené v hotové podobě, ale musí se dopočítávat. Proto je řešení pomocí exportované služby vhodnější.

2.5 Struktura a popis tříd

V této sekci se budeme věnovat aplikaci z hlediska tříd. Nejprve si popíšeme balíček jako celek, potom shrneme základní informace o knihovně a nakonec postupně popíšeme jednotlivé vrstvy balíčku. Jelikož je balíček poměrně rozsáhlý (co se týká počtu tříd) byl by diagram tříd poněkud rozsáhlý a pro čtenáře nepřehledný. Systém tedy popíšeme z pohledu jednotlivých balíčků. Nejsou zde popisované jednotlivé třídy, jejich dokumentaci lze nalézt na přiloženém médiu.

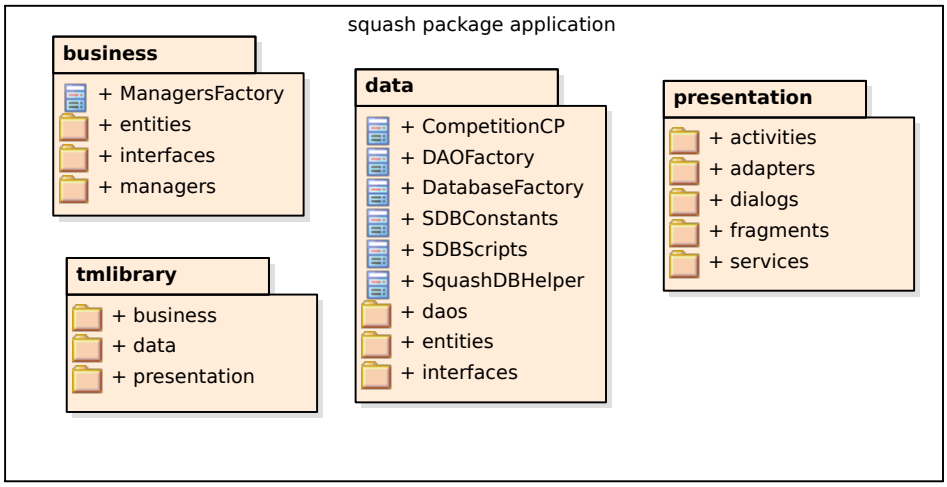
2.5.1 Aplikace jako celek

V této sekci popíšeme aplikaci jako celek z pohledu balíčků.

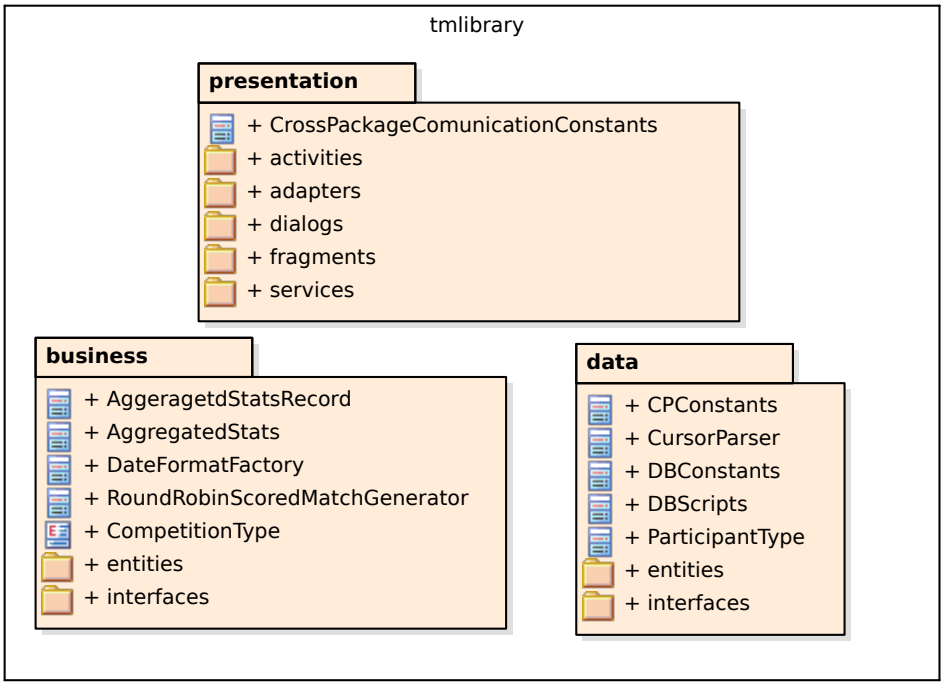
Jak si můžeme všimnout na obrázku 2.2, aplikace balíčku pro squash se skládá ze čtyř částí (jednotlivých balíčků). První tři tedy `business`, `data` a `presentation` reprezentují jednotlivé vrstvy systému popsané v sekci 2.2. Tyto balíčky budou podrobněji popsány v následujících sekcích. Čtvrtý balíček s názvem `tmlibrary` představuje „Knihovnu“ zmíněnou v předchozí kapitole. Jedná se ve skutečnosti o celý další knihovní projekt, který si v rychlosti popíšeme v sekci (2.5.2). Podrobnější informace o knihovně je možné najít v diplomové práci Josefa Němečka [4].

2.5.2 Knihovna

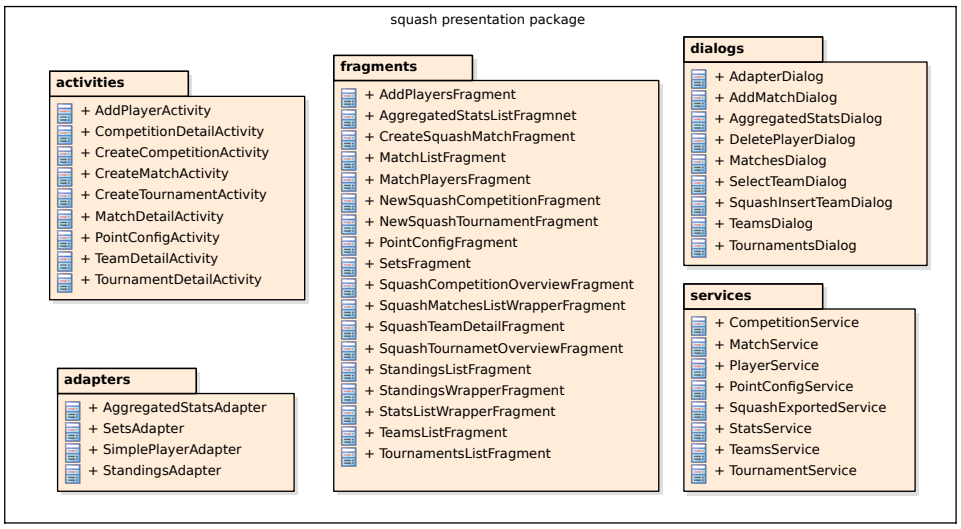
V této sekci si v rychlosti popíšeme knihovnu.



Obrázek 2.2: Balíčkový diagram tříd aplikace.



Obrázek 2.3: Balíčkový diagram tříd knihovny.



Obrázek 2.4: Balíčkový diagram tříd prezentační vrstvy.

Jak můžeme vidět na obrázcích 2.2 a 2.3 aplikace i knihovna dodržují stejnou strukturu. Nyní si v popíšeme jednotlivé vrstvy knihovny, do komponent zajdeme ve větším detailu při popisu jednotlivých vrstev aplikace.

Prezentační vrstva obsahuje dvě skupiny prvků napříč všemi komponentami. První skupina usnadňuje vývoj UI balíčku. Do této skupiny patří fragment `AbstractDataFragment`, který zajišťuje správný koloběh dat ve vztahu ke svému životnímu cyklu. Druhá pak obsahuje už přímo hotové obrazovky, které je nutné správně napojit na svůj zdroj dat. Příkladem je abstraktní třída `CompetitionOverviewFragment`, která reprezentuje obrazovku přehledu soutěže.

Business vrstva obsahuje hlavně společné entity napříč balíčky, dále pak rozhraní managerů, které jednotlivé balíčky implementují.

Datová vrstva obsahuje entity datové vrstvy, rozhraní DAO objektů, které jednotlivé balíčky implementují a třídy pro tvorbu a přístup společných databázových tabulek.

2.5.3 Prezentační vrstva

V této sekci si popíšeme prezentační vrstvu a jednotlivé skupiny tříd, které se na ní nacházejí. Diagram můžeme vidět na obrázku 2.4

Aktivity. Hlavní úkol aktivit v naší aplikaci je v sobě sdružovat fragmenty a umožňovat navigaci mezi nimi. Data si pak spravují fragmenty samy. Jedinou výjimkou je třída `MatchDetailActivity`, která v sobě sdružuje fragmenty pro detail zápasu (`SetsFragment` a `MatchPlayersFragment`). Zde je potřeba, aby se data z těchto fragmentů ukládala najednou.

Adaptéry. Adaptér je komponenta OS Android, která určuje, jak má vypadat položka v seznamu. Nezáleží na tom, jestli má seznam formu klasickou řádkovou, nebo formu dlaždic. V aplikaci máme několik seznamů (seznam hráčů, turnajů, zápasů). Adaptéry pro společné prvky jsou uloženy v Knihovně. Zde jsou adaptéry, které používá pouze naše aplikace.

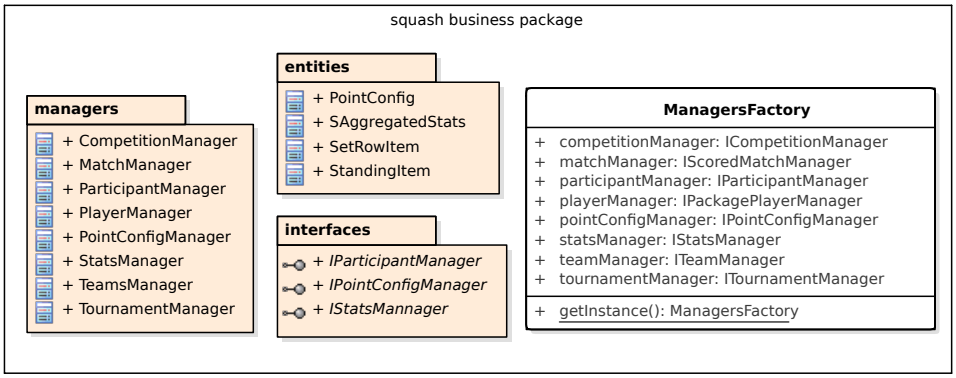
Fragmenty představují jednotlivé obrazovky naší aplikace. Samy si spravují svoje data a definují, jak má daná obrazovka vypadat.

Dialogy ačkoliv je v naší aplikaci, dialog ve zkutečnosti podtřída fragmentu, uvádíme je samostatně. Tyto třídy určují co mají dialogy mít za volby a jak se mají chovat.

Služby jsou zodpovědné za vyřizování požadavků prezentační vrstvy na business vrstvu. Byly již popsány v sekci 2.3.

2.5.4 Business vrstva

V této sekci si v krátkosti popíšeme business vrstvu aplikace. Můžeme jí vidět na obrázku 2.5.



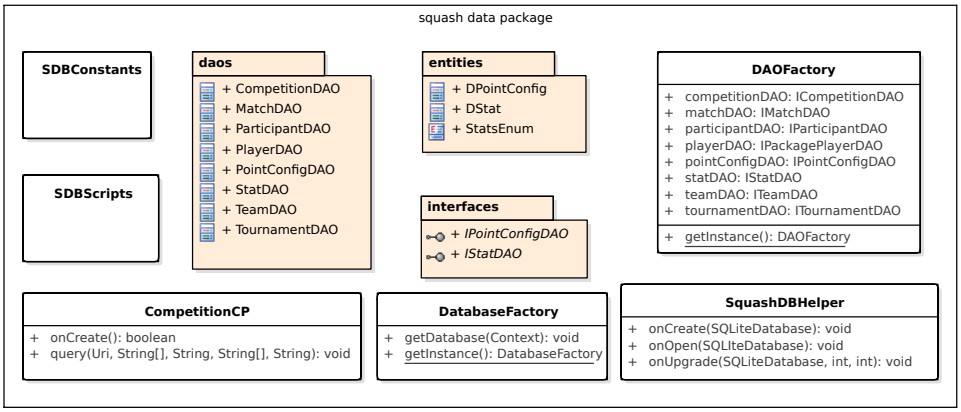
Obrázek 2.5: Balíčkový diagram tříd business vrstvy.

Třídy ve složce balíčku entities, představují jednotlivé entity. Zde jsou entity specifické pro squash. Ostatní jsou uložené v Knihovně. Oproti doménovému modelu nám přibyly další entity (`SAggregatedStats`, `SetRowItem`,

`StandingItem`), které přestavují uložené, či napočítané statistiky. Dále pak business vrstva obsahuje jednotlivá rozhraní, která definují předpis pro managery. Ty tento předpis implementují. Část používaných entit a interfejsů je opět uložená v knihovně. Příkladem manageru je třeba `PointConfigManager`, který spravuje bodovou konfiguraci jednotlivých turnajů a má metody `insert`, `update`, `delete` a `getById`.

2.5.5 Datová vrstva

V této sekci si v krátkosti popíšeme datovou vrstvu aplikace. Můžeme jí vidět na obrázku 2.6.



Obrázek 2.6: Balíčkový diagram tříd datové vrstvy.

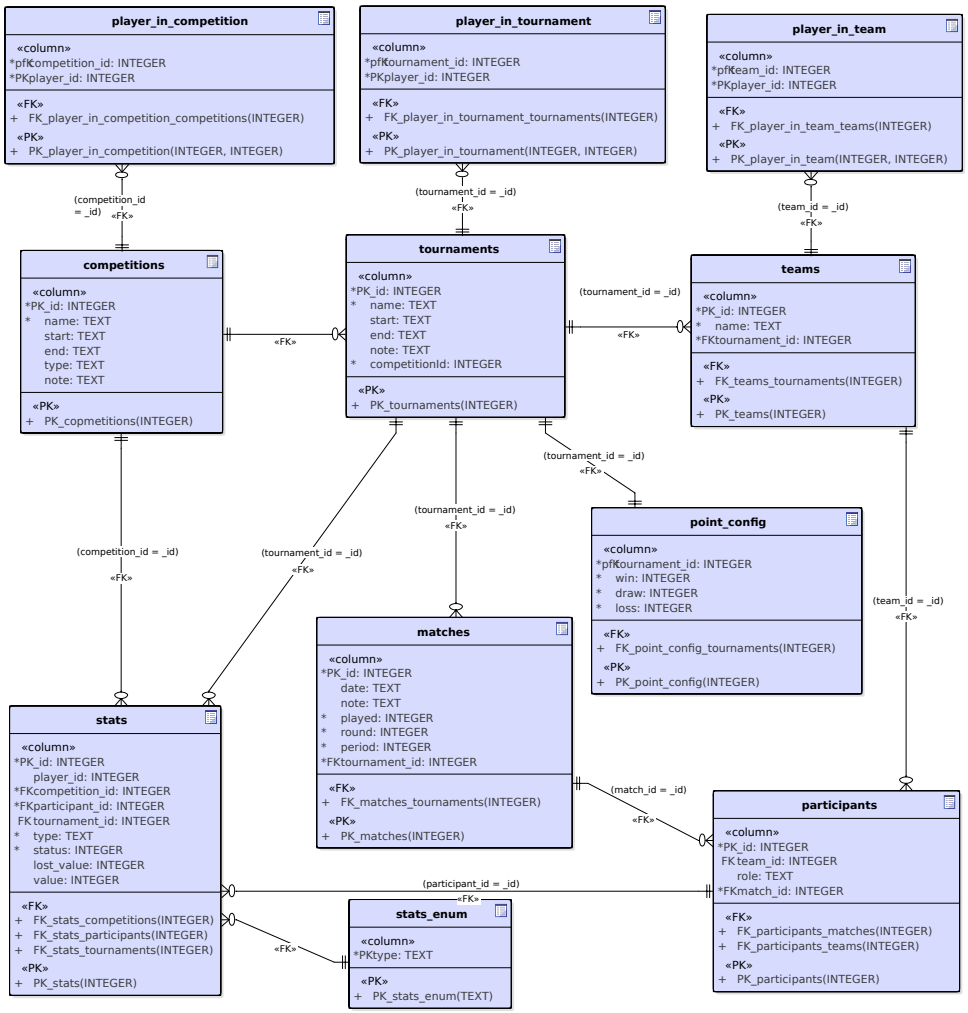
Datová vrstva obsahuje jednotlivé třídy pro správu databáze, rozhraní, která určují předpis DAO objektů a následně samotné DAO objekty na manipulaci s entitami. Opět zde platí, že část entit a rozhraní je uložená v knihovně. Příkladem DAO objektu je `ParticipantDAO`, které spravuje v databázi jednotlivé účastníky zápasů a poskytuje metody `insert`, `update`, `delete`, `getParticipantsByMatchId`, `getParticipantsByTeamId` a `getById`.

2.6 Databázový model

V této sekci si představíme a okomentujeme databázový model.

Databázový model vyobrazený na obrázku 2.7 je odvozený od doménového modelu představeného v předchozí kapitole. Nebude tedy třeba popisovat jednotlivé entity, jen ujasníme nejasnosti, které by mohly nastat.

Jak si můžeme všimnout v hojném počtu je zajištěna integrita pomocí cizích klíčů, ale ani v jednom případě cizí klíč neukazuje na hráče a tabulka hráčů zcela chybí. Tato tabulka chybí, neboť je uložena v jádře a my z ní



Obrázek 2.7: Databázový model aplikace.

získáváme data přes dříve zmíněný ContentProvider. Tím, že je uložená v jádře a tedy v úplně jiné databázi, nemůžeme na ní odkazovat pomocí cizích klíčů.

Můžeme si dále všimnout, že tabulka stats, do které se ukládají statistiky, obsahuje nadměrné množství vazeb (participant, tournament, match). Ačkoliv by tyto vazby nebyly nutné, protože bychom se ke statistikám dostali přes ty ostatní, přidání těchto vazeb nám urychluje a zjednodušuje práci.

2.7 Komunikace napříč vrstvami

V této sekci si ukážeme, jakým způsobem spolu komunikují vrstvy naší aplikace představené v sekci 2.2. Ukážeme si to na příkladu fragmentu, který

zobrazuje statistiky hráčů v soutěži. Sekvenční diagram si lze prohlédnout na obrázku 2.8.

Vše začíná v momentě, kdy si fragment zažádá o data metodou `askForData`. Tato metoda je zodpovědná za zaslání intentu operačnímu systému, který nastartuje příslušnou službu. Zde se jedná o intent explicitní. V případě tohoto fragmentu je dříve zmíněná metoda spjata s jeho životním cyklem.

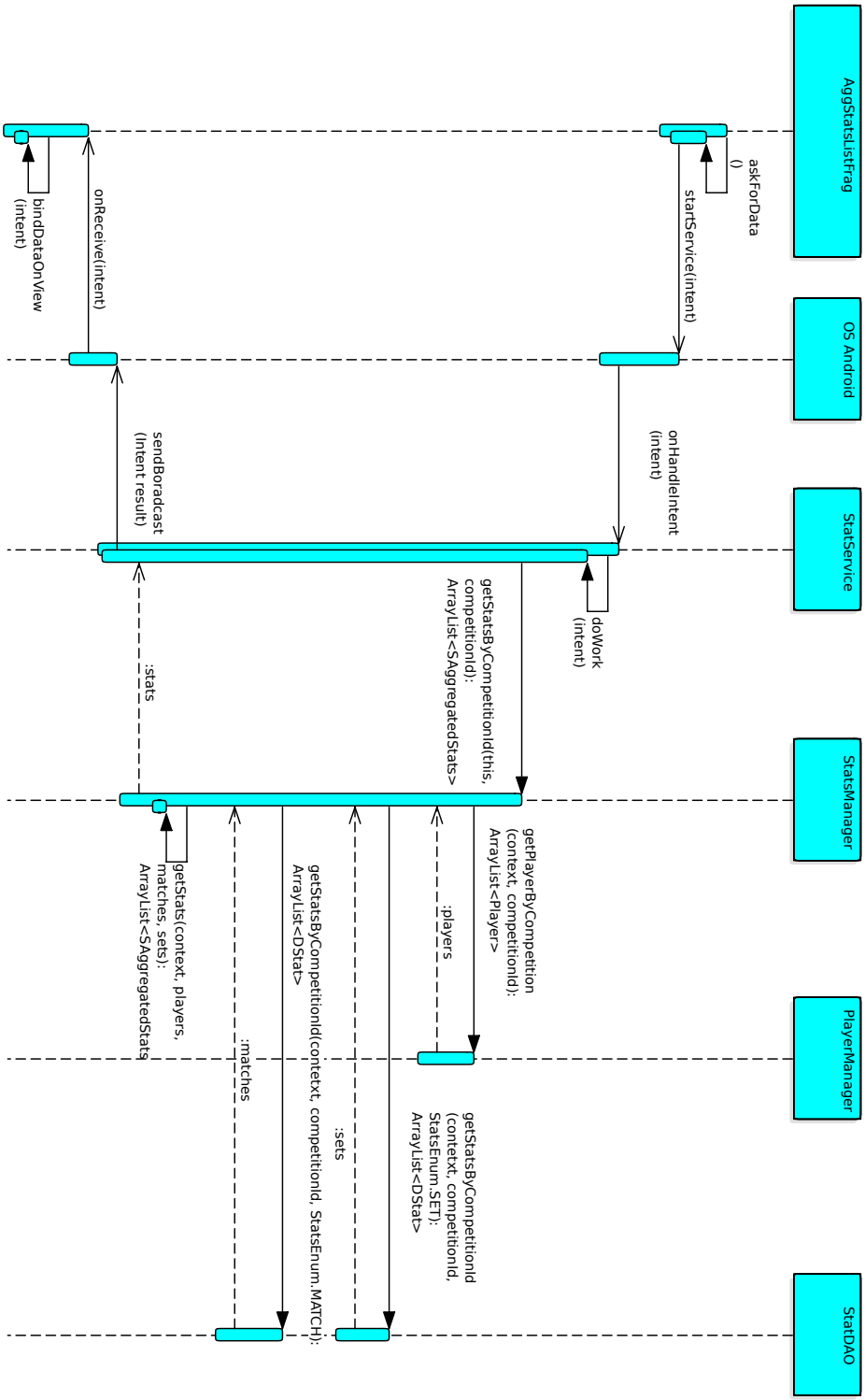
Operační systém nastartuje příslušnou službu a předá ji dříve zmíněný intent. Tento intent obsahuje veškerá data nutná k tomu, aby příslušný manager mohl provést svojí práci. V tomto případě se jedná o id soutěže, jejíž statistiky chceme získat. Služba provádí svojí hlavní práci v metodě `doWork`, kde se volají příslušné managery. I když to není do obrázku zaneseno, managery a DAO objekty jsou odstíněné pomocí rozhraní a přistupuje se k nim přes příslušné factory[4]. V tomto konkrétním příkladě se volá `StatsManager`, který má na starost získávání a dopočítávání statistik.

V managerech se provede příslušná logika (volání ostatních managerů a DAO objektů, zpracování dat) a vrátí se výsledek zpět do služby. Zde se získá pomocí třídy `PlayerManager`, list hráčů v dané soutěži a z databáze se pomocí DAO objektů získají statistiky setů a zápasů. Tato data se pošlou do metody `getStats`, která je zodpovědná za jejich korektní zpracování. Metody obsahují context, protože je potřeba pro přístup do databáze.

Řízení se vrátí zpět do služby. Služba vytvoří nový intent, který bude obsahovat výsledná data. Tomuto intentu nastaví akci, kterou získá ze svého spouštěcího intentu a vloží do něj výsledná data (statistiky). Služba tento intent pošle systému, který se ho pokusí doručit zpátky do fragmentu. Systém se data pokusí doručit příslušnému fragmentu podle akce, kterou obsahuje, zde fragmentu `AggregatedStatsListFragment`.

Fragment je buď v tuto chvíli naživu, tedy je na akci registrovaný, data přijme a zobrazí nebo není naživu a data se ztratí. Ztráta dat nevadí, protože to, že fragment není naživu, znamená, že má uživatel zobrazeno něco jiného a data nepotřebuje. V momentě, kdy uživatel chce data zobrazit, celý proces proběhne znovu. Na doručování dat ze služeb se používá implicitní intent. Pokud bychom použili explicitní intent a služba by ho poslala ve chvíli, kdy je fragment zničený, mohlo by to skončit například pádem aplikace.

[4]Podle [12] je factory návrhový vzor, který slouží k vytváření objektů. Abstraktní třída Factory definuje interfejsy, které bude factory vracet a její podtřídy doplní, které objekty budou dostupné za těmito intefejsy. V našem případě se jedná o to, co zde nazývají jako programátorský idiom Simple Factory tj. třída není abstraktní, ale rovnou za interfejsy doplňuje implementace.



Obrázek 2.8: Sekvenční model komunikace vrstev.

2.8 Shrnutí

V této kapitole jsme se věnovali návrhové části. Představili jsme si specifika OS Android (práce na pozadí a omezené zdroje), udělali jsme si návrh architektury (tři vrstvy - prezentační business a datová), udělali úvod do komponent OS Android, představili jsme si skupiny tříd v aplikaci a jejich funkcionalitu, dále databázový model a model komunikace.

Kapitola 3

Implementace a testování

V této sekci si popíšeme prostředky použité pro vývoj aplikace. Poté zrekapitulujeme, jakým způsobem byla aplikace otestována. Na závěr si ukážeme a okomentuje ukázku kódu.

3.1 Použité prostředky pro vývoj

V následující sekci si popíšeme prostředky použité pro vývoj aplikace.

Vývojové prostředí. Jako vývojové prostředí bylo použito Android Studio ve verzi 1.5.1. Naše aplikace je modul squash v projektu pro toto IDE. Projekt je k dispozici na přiloženém CD.

Emulátor. Pro vývoj aplikací na Android je vhodné použít emulátor mobilního telefonu. Byl použit emulátor od společnosti Genymotion[5], konkrétně emulátor telefonu HTC One s verzí androidu 4.3(API 18).

Repozitář a správa projektu. Pro správu projektu byl vytvořen repozitář na školním Gitlabu[6]. Kromě toho, že slouží jako cloudový repozitář pro ukládání zdrojových kódů, byly využity další funkcionality jako wiki pro sdílení obsahu (převážně analýza a návrh), issue tracking systém pro monitorování bugů a rozdělování úkolů mezi členy týmu.

3.2 Jednotkové testování

V této sekci si povíme něco o jednotkovém testování a ukážeme si, jakým způsobem proběhlo v naší aplikaci.

[5] `https://www.genymotion.com/`

[6] `https://gitlab.fit.cvut.cz/nemecjo2/tournamentmanager`

Podle [13] je jednotkové testování praktika, podle které se automaticky testují malé, nezávislé a oddělené kusy kódu. Většinou se jedná o metody, v méně častých případech třídy. Abychom mohli takovéto testy psát, musí být testované části oddělené nebo musí jít oddělit pomocí mockování[7].

V našem případě je jednotkovými testy pokrytý celý `StatsManager`, který počítá statistiky hráčů a pořadí v turnaji. Tato funkcionalita by se jinak testovala velmi složitě. Testování na OS Android běžně vyžaduje přítomnost emulátoru. Naštěstí existuje framework Robolelectric[8], který obsahuje stínové kopie Android tříd a tuto nezbytnost odbourává. Jako standardní testovací framework byl použit JUnit[9] a na mockování Mockito[10]. Celkem tedy vzniklo 5 jednotkových testů, které testovaly nejdůležitější funkcionalitu a všechny proběhly úspěšně.

3.2.1 Ukázka testovací třídy

V této sekci si ukážeme a popíšeme testovací třídu `StatsManagerTest`.

Jak můžeme vidět v ukázce kódu 1, třída obsahuje reference na potřebné managery a DAO objekty nutných pro chod třídy `StatsManager`. Metoda `setUp` proběhne před každým testem a má za úkol naplnit managery a DAO objekty testovací třídy namockovanými instancemi a vložit je do příslušné factory. Naopak metoda `tearDown` probíhá po každém testu a uvádí factory do původního stavu. Zbytek metod testuje jednotlivé metody třídy `StatManager` tak, aby byla pokryta celá její funkcionalita. Testovací třída obsahuje ještě privátní metody na přípravu dat do mockovaných instancí a assertování výsledků, protože výsledky chodí v kolekci.

Testy probíhají následovně:

1. Namockované instance se naplní vhodnými testovacími daty.
2. Zavolá se testovaná metoda.
3. Pokud metoda vrací výsledek, zkontroluje se jeho správnost. Pokud v dané metodě probíhá komunikace s metodami ostatních DAO objektů typů insert, update, zkontroluje se, že tyto metody byly volané ve správném počtu a se správnými argumenty.

3.3 Manuální testování

Aplikaci byla z velké části otestována manuálně. Následující sekce popisuje testovací scénáře, podle kterých tak bylo učiněno.

[7] Skutečné implementace tříd, se nahradí mock objekty, což jsou takové objekty, které pouze simulují chování dané třídy [14].

[8] `http://robolectric.org/`

[9] `http://junit.org/junit4/`

[10] `http://mockito.org/`

```
@RunWith(RobolectricGradleTestRunner.class)
@Config(constants = BuildConfig.class, sdk = 21)
public class StatsManagerTest {

  IPackagePlayerManager mockPlayerManager;
  IStatDAO mockStatsDAO;
  IParticipantDAO mockParticipantDAO;
  IMatchDAO mockMatchDAO;
  ITournamentDAO mockTournamentDAO;
  IPackagePlayerDAO mockPlayerDAO;
  ITeamManager mockTeamsManager;
  ICompetitionManager mockCompetitionManager;
  IPointConfigManager mockPointConfigManager;
  ITeamDAO mockTeamDAO;

  @Before
  public void setUp() throws Exception {...}

  @Test
  public void testGetAggregatedStats() throws Exception {...}

  @Test
  public void testGetSetsForMatch() throws Exception {...}

  @Test
  public void testUpdateStatsForMatch() throws Exception {...}

  @Test
  public void testGetPlayersForMatch() throws Exception {...}

  @Test
  public void testGetStandingsByTournament()
    throws Exception {...}

  @After
  public void tearDown() throws Exception {...}

}
```

Ukázka kódu 1: Ukázka třídy `StatsManagerTest`.

Založení soutěže. Uživatel klikne na tlačítko založit soutěž. Aplikace mu zobrazí formulář. Uživatel se pokusí uložit prázdnou soutěž. Aplikace zobrazí chybovou hlášku. Uživatel vyplní údaje a soutěž uloží. Na obrazovce soutěží squashe bude tato soutěž vidět.

Úprava soutěže. Uživatel se nachází na detailu soutěže a zvolí upravit soutěž. Aplikace mu zobrazí formulář s předvyplněnými detaily, avšak neumožní změnu typu soutěže. Uživatel smaže název a pokusí se soutěž uložit. Aplikace zobrazí chybovou hlášku. Uživatel změní údaje a soutěž uloží. Změna se promítne na obrazovce s přehledem soutěže.

Přidání hráčů do soutěže. Uživatel se nachází na obrazovce hráčů a statistik v soutěži. Zvolí přidat hráče. Aplikace mu zobrazí seznam hráčů v jádře bez hráčů, co už v turnaji jsou. Uživatel vybere hráče a zvolí uložit. Vybraní hráči se objeví v soutěži. Uživatel zkontroluje, že obrazovka přehledu odpovídá.

Smazání hráče ze soutěže. Uživatel se nachází na obrazovce hráčů v soutěži a má připraveného hráče A a B. A se nenachází v žádném turnaji. B se nachází aspoň v jednom turnaji. Uživatel zvolí smazat hráče B. Aplikace zobrazí chybovou hlášku. Uživatel zvolí smazat hráče A. Aplikace ho odstraní ze seznamu. Uživatel zkontroluje obrazovku přehledu soutěže.

Založení turnaje. Uživatel se nachází na obrazovce turnajů v soutěži. Uživatel klikne na tlačítko založit turnaj. Aplikace mu zobrazí formulář. Uživatel se pokusí uložit prázdný turnaj. Aplikace zobrazí chybovou hlášku. Uživatel vyplní údaje a turnaj uloží. Na obrazovce turnajů v soutěži se nový turnaj zobrazí. Uživatel zkontroluje obrazovku přehledu soutěže.

Úprava turnaje. Uživatel se nachází na detailu turnaje a zvolí upravit turnaj. Aplikace mu zobrazí formulář s předvyplněnými detaily. Uživatel smaže název a pokusí se turnaj uložit. Aplikace zobrazí chybovou hlášku. Uživatel změní údaje a turnaj uloží. Změna se promítne na obrazovce s přehledem turnaje.

Přidání hráče do turnaje. Uživatel se nachází na obrazovce hráčů v detailu turnaje jednotlivců. Zvolí přidat hráče. Aplikace mu zobrazí seznam hráčů z dané soutěže bez hráčů, kteří už v turnaji jsou. Uživatel vybere hráče a zvolí uložit. Noví hráči se objeví v seznamu hráčů v turnaji. Uživatel zkontroluje, že se hráči objeví i v tabulce pořadí turnaje. Uživatel zkontroluje obrazovku přehledu turnaje.

Odebrání hráče z turnaje. Uživatel se nachází v turnaji týmů na obrazovce se seznamem hráčů a má k dispozici hráče A a B. A není členem žádného týmu. B je členem některého týmu. Uživatel zvolí odstranit hráče B a aplikace zobrazí chybovou hlášku. Uživatel zvolí odebrat hráče A z turnaje a aplikace ho odstraní ze seznamu. Uživatel zkontroluje obrazovku přehledu turnaje.

Uživatel se nachází v turnaji jednotlivců na obrazovce se seznamem hráčů a má k dispozici hráče A a B. A se neúčastní žádného zápasu. B se účastní některého zápasu. Uživatel zvolí odebrat hráče A z turnaje a aplikace ho odstraní ze seznamu. Uživatel zkontroluje obrazovku přehledu turnaje a obrazovku pořadí účastníků v turnaji. Uživatel zvolí odstranit hráče B a aplikace zobrazí chybovou hlášku.

Založení týmu. Uživatel se nachází v turnaji týmů na obrazovce týmů. Uživatel zvolí založit tým. Aplikace mu zobrazí dialog pro vyplnění názvu. Uživatel zvolí uložit, aplikace zobrazí chybovou hlášku. Uživatel opět zvolí založit tým a tentokrát vyplní název. Tým se objeví v seznamu týmu. Nebudou vidět žádní hráči. Uživatel zkontroluje změnu v přehledu turnaje.

Úprava týmu. Uživatel se nachází v turnaji týmů, na obrazovce má k dispozici tým A, který se účastní některého zápasu. Uživatel zvolí upravit tým A. Aplikace mu zobrazí dialog pro vyplnění názvu s předvyplněným názvem. Uživatel ho smaže a zvolí uložit, aplikace zobrazí chybovou hlášku. Uživatel opět zvolí upravit tým A, tentokrát změní název. Uživatel zkontroluje, že se název změnil v seznamu týmů, v zápasech daného týmu a v tabulce pořadí turnaje.

Smazání týmu. Uživatel se nachází na obrazovce týmů v turnaji a má k dispozici tři týmy A, B, C. Tým A obsahuje aspoň jednoho hráče. Tým B se účastní aspoň jednoho zápasu. Tým C nemá žádné hráče a neúčastní se žádného zápasu. Uživatel zvolí smazat tým A. Aplikace zobrazí chybovou hlášku. Uživatel zvolí smazat tým B. Aplikace opět zobrazí chybovou hlášku. Uživatel zvolí smazat tým C. Aplikace ho odstraní ze seznamu. Uživatel zkontroluje, že se tým nenachází v tabulce pořadí a počet týmů na přehledu turnaje odpovídá.

Úprava soupisky týmu. Uživatel se nachází na obrazovce týmů a má k dispozici tým A, který obsahuje alespoň jednoho hráče, který se zúčastní mininmálně jednoho odehraného zápasu. Uživatel si zobrazí soupisku týmu A, odebere všechny hráče a zvolí uložit. Zkontroluje, že se nezobrazují u daného týmu a že se nezměnily jejich statistiky. Poté opět zvolí upravit soupisku a zvolí přidat hráče na soupisku. Aplikace mu zobrazí seznam hráčů. V tomto seznamu jsou hráči, kteří nejsou v žádném týmu nebo jsou v tomto týmu a byli ze soupisky odstraněni, ale soupiska ještě nebyla uložena. Uživatel označí

hráče, které chce přidat na soupisku a zvolí uložit. Hráči se objeví na soupisce. Poté zvolí uložit soupisku a zkontroluje, že hráči týmu jsou v seznamu týmů.

Založení zápasu. Uživatel se nachází na seznamu zápasů v turnaji, který nemá žádné účastníky. Uživatel zvolí vytvořit zápas, aplikace zobrazí chybovou hlášku. Uživatel přidá dva účastníky do turnaje a znovu zvolí založit zápas. Aplikace zobrazí formulář. Uživatel zvolí uložit. Aplikace zobrazí chybovou hlášku o periodě a kole. Uživatel vyplní periodu a kolo a zvolí uložit. Aplikace zobrazí chybovou hlášku. Uživatel nastaví účastníky zápasu a zvolí uložit. Aplikace zápas uloží. Uživatel zkontroluje, že se zápas objevil v seznamu zápasů a že nedošlo k žádné změně v pořadí účastníků ani ve statistikách hráčů turnaje. Uživatel zkontroluje změnu v přehledu turnaje.

Vygenerování kola zápasů. Uživatel se nachází na seznamu zápasů v turnaji, který nemá žádné účastníky. Uživatel zvolí vygenerovat kolo zápasů, aplikace zobrazí chybovou hlášku. Uživatel přidá tři účastníky do turnaje a znovu zvolí vygenerovat kolo zápasů. Aplikace vytvoří zápasy. Uživatel zkontroluje, že se v seznamu zápasů objevili tři zápasy a každý účastník hraje s každým dalším jednou. Uživatel zvolí vygenerovat další kolo. Aplikace ho vygeneruje. Uživatel zkontroluje, že se objevily další tři zápasy a pozice účastníků domácí a hosté je prohozená. Dále zkontroluje, že nedošlo k žádné změně v pořadí účastníků ani ve statistikách hráčů turnaje. Uživatel zkontroluje změnu v přehledu turnaje.

Úprava zápasu. Uživatel se nachází na seznamu zápasů v turnaji a má k dispozici zápas. Uživatel zvolí upravit zápas. Aplikace zobrazí formulář s předvyplněnými údaji a neumožní již měnit účastníky. Uživatel smaže periodu a kolo a zvolí uložit. Aplikace zobrazí chybovou hlášku o periodě a kole. Uživatel vyplní periodu a kolo a zvolí uložit. Aplikace zápas uloží. Uživatel zobrazí detail zápasu a zkontroluje, že upravené údaje souhlasí.

Resetování zápasu. Uživatel se nachází na seznamu zápasů v turnaji a má k dispozici odehraný zápas. Uživatel zvolí resetovat zápas. Aplikace zápas vynuluje. Uživatel zkontroluje, že u daného zápasu není zobrazené skóre. Dále, že v pořadí turnaje a ve statistikách hráčů došlo k patřičným změnám.

Smazání zápasu Uživatel se nachází na seznamu zápasů v turnaji a má k dispozici odehraný zápas. Uživatel zvolí smazat zápas. Aplikace zápas smaže. Uživatel zkontroluje, že se daný zápas již nenachází v seznamu. Dále, že v pořadí turnaje a ve statistikách hráčů došlo k patřičným změnám.

Vyplnění zápasu Uživatel se nachází v seznamu zápasů turnaje týmu a má k dispozici neodehraný zápas. V tomto zápase má každý tým dva hráče. Uživa-

tel zvolí zobrazit detail zápasu. Aplikace mu zobrazí obrazovku setů. Uživatel zvolí přidat set a zvolí uložit zápas. Aplikace mu zobrazí chybovou hlášku. Uživatel smaže jednu nulu a zvolí uložit zápas. Aplikace zobrazí chybovou hlášku. Uživatel vyplní výsledek setu korektně. Uživatel zvolí obrazovku soupisek a z každého týmu odstraní jednoho hráče. Poté zvolí možnost přidat hráče do libovolného týmu. Aplikace mu zobrazí seznam s odstraněným hráčem podle týmu, který zvolil. Uživatel se vrátí zpět a zvolí uložit zápas. Zkontroluje, že se správně změnilo skóre zápasu, pořadí týmů a statistiky účastněných hráčů.

3.4 Ukázky kódu

3.4.1 Uložení statistiky do databáze

V této sekci si ukážeme, jakým způsobem se statistika uloží do databáze.

```
private ContentValues convert(DStat stat){...}

@Override
public void insert(Context context, DStat stat) {
  SQLiteDatabase db =
↪  DatabaseFactory.getInstance().getDatabase(context);
  ContentValues cv = convert(stat);

  db.insert(SDBConstants.tSTATS, null, cv);
  db.close();
}
```

Ukázka kódu 2: Ukázka uložení statistiky do databáze.

V ukázce 2 můžeme vidět metodu `insert` třídy `StatDAO`. Přístup do databáze se získává pomocí pseudo-factory patternu popsaného v sekci 2.7, abychom případně mohli DAO automaticky testovat. Pro vložení se statistika musí vložit do instance třídy `ContentValues`. Tato instance obsahuje dvojice klíč-hodnota. Klíč je název sloupečku v databázi. Hodnota, je požadovaná hodnota sloupečku v databázi. Při vložení je potřeba uvést název tabulky a naplněná instance `ContentValues`. Prostřední parametr je zapotřebí pouze při vložení prázdných `ContentValues`, a to se nás netýká.

3.5 Shrnutí

V této kapitole jsme si představili prostředky použité pro vývoj (IDE, emulátor, Gitlab repozitář). Dále jsme popsali, jakým způsobem byla aplikace testována. (jednotkové testy, manuální testy) a na závěr uvedli a komentovali ukázku kódu vkládání statistik do databáze.

Závěr

V této práci jsme si popsali analýzu, vývoj, implementaci a testování balíčku pro squash do aplikace Tournament Manager pro operační systém Android. Výstupem této práce je funkční prototyp balíčku, který splňuje všechny požadavky specifikované v analýze, jak funkční, tak nefunkční. Tato práce tedy splnila všechny předepsané cíle zaměřené na seznámení se s jádrem aplikace Tournament Manager, analýzu požadavků na organizaci turnajů a evidenci výsledků ve squashi, navržení architektury a řešení a implementaci balíčku pro squash dle provedeného návrhu. Balíček byl následně otestován, byla vytvořena dokumentace implementovaného řešení a uživatelská příručka.

Prototyp je spolu s uživatelskou příručkou a dokumentací zdrojových kódů k dispozici na přiloženém CD. V budoucnu je možné řešit funkčnost sdílení soutěží mezi uživateli pomocí serveru. Určité prvky za tímto účelem již byly zakomponovány do balíčku i přes to, bude vyžadovat další úpravy. Další námět rozvoje aplikace může být zaměřen na možnosti využití balíčku i pro jiné obdobné sporty (např. stolní tenis), které bude umět rozlišovat.

Literatura

[1] WALLBUTTON, T.: *World Squash Federation One Hundred And Forty Years Of Squash*. World Squash, [online], 2016, [cit. 2016-05-02]. Dostupné z: `http://www.worldsquash.org/ws/wsf-information/squash-history/140-years-of-squash`

[2] ROUSE, M.: *business process*. TechTarget, [online], 2016, [cit. 2016-05-15]. Dostupné z: `http://searchcio.techtarget.com/definition/business-process`

[3] NIŽARADZE, J.: *MobChar - jádro aplikace*. Bakalářská práce, České vysoké učení technické v Praze, Fakulta informačních technologií, 2016.

[4] NĚMEČEK, J.: *Tournament Manager - jádro aplikace*. Diplomová práce, České vysoké učení technické v Praze, Fakulta informačních technologií, Unpublished.

[5] KOŠUT, O.: *Tournament Manager - balíček pro hokej*. Bakalářská práce, České vysoké učení technické v Praze, Fakulta informačních technologií, 2016.

[6] SOMMERVILLE, I.: *Software Engineering*. Harlow:Pearson Education Limited, 8 vydání, ISBN 0-321-31379-8.

[7] Android Developers: *Fragment*. [online], 2016, [cit. 2016-05-12]. Dostupné z: `http://developer.android.com/guide/components/fragments.html`

[8] Android Developers: *Activities*. [online], 2016, [cit. 2016-05-12]. Dostupné z: `http://developer.android.com/guide/components/activities.html`

[9] Tutorials Point Ltd.: *Android - Services*. [online], 2016, [cit. 2016-05-12]. Dostupné z: `http://www.tutorialspoint.com/android/android_services.htm`

[10] Tutorials Point Ltd.: *Android - Intents and Filters*. [online], 2016, [cit. 2016-05-12]. Dostupné z: http://www.tutorialspoint.com/android/android_intents_filters.htm

[11] Android Developers: *Context*. [online], 2016, [cit. 2016-05-12]. Dostupné z: http://developer.android.com/reference/android/content/Context.html

[12] FREEMAN, E.; FREEMAN, E.; BATES, B.; aj.: *Head First Design Patterns*. O'Reilly, 2004, ISBN 0596007124.

[13] FARCIC, V.; GARCIA, A.: *Test-Driven Java Development*. Packt Publishing Ltd., 2015, ISBN 978-1-78398-742-9.

[14] ROUSE, M.: *mock object*. TechTarget, [online], 2016, [cit. 2016-05-15]. Dostupné z: http://searchsoftwarequality.techtarget.com/definition/mock-object

Příloha **A**

Seznam použitých zkratek

UI Uživatelské rozhraní

OS Operační systém

IDE Vývojové prostředí

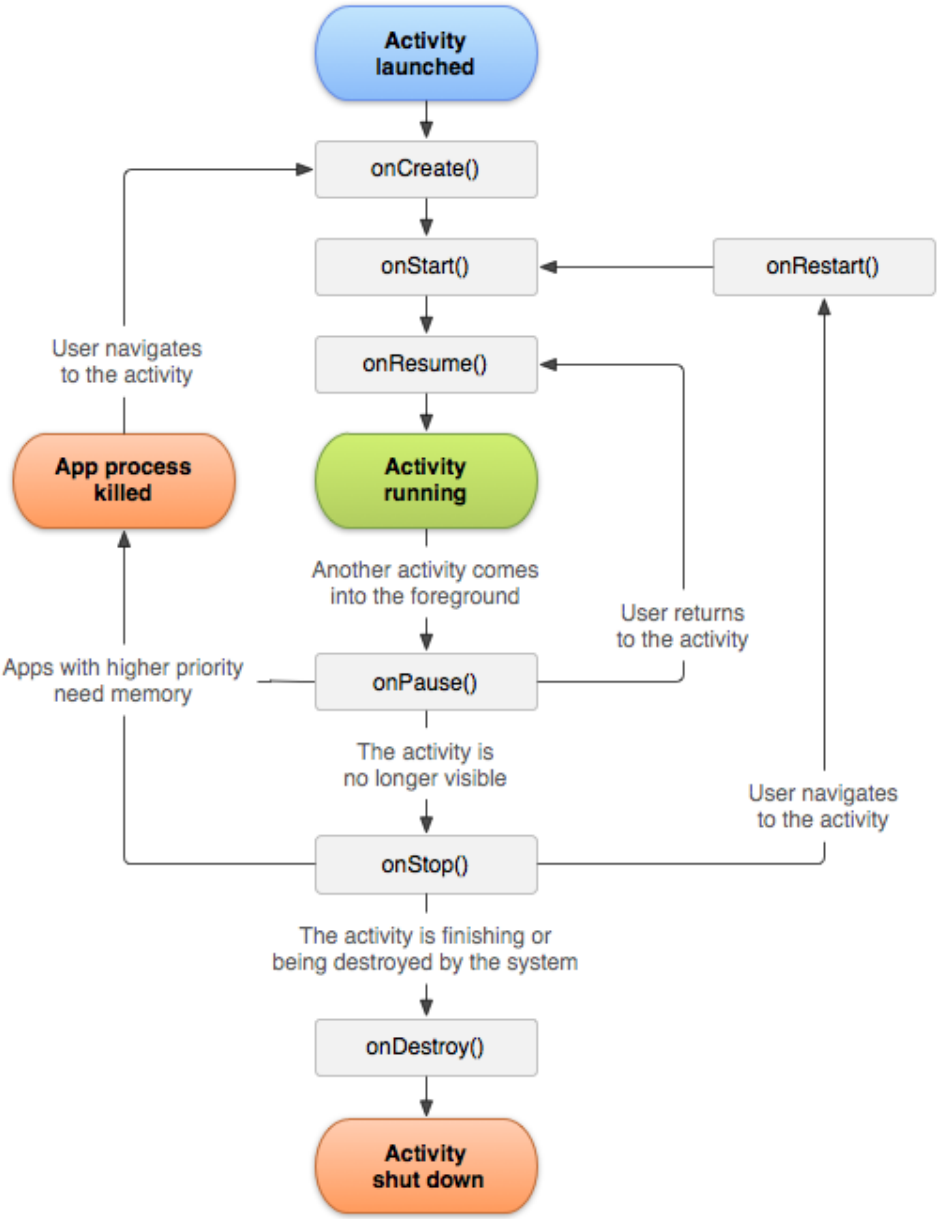
Příloha **B**

Obsah přiloženého CD

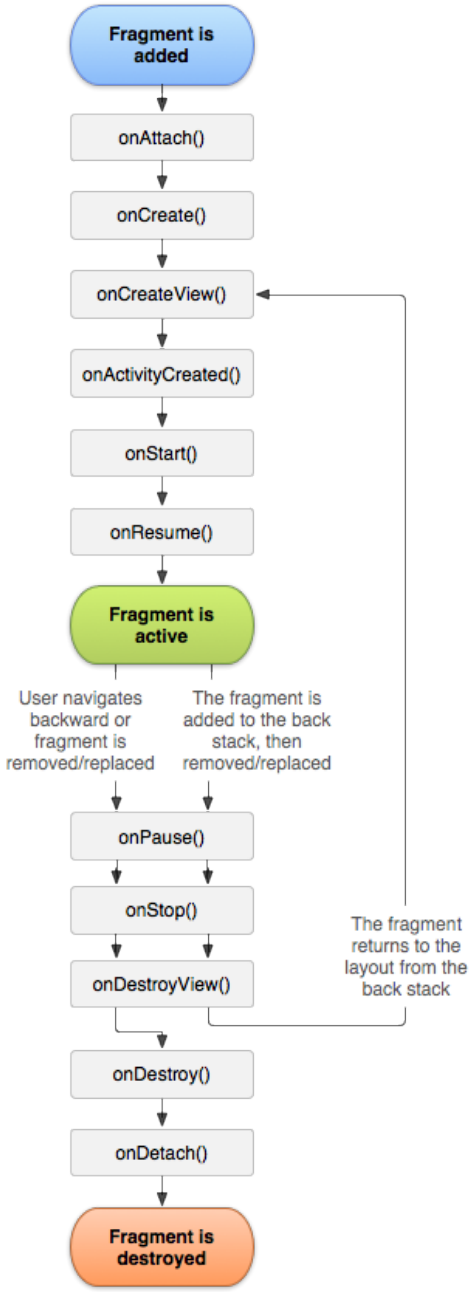
- `readme.txt` ..... stručný popis obsahu CD
- `apks` ..... adresář s připravenými aplikacemi
  - `core.apk` ..... prototyp jádra aplikace Tournament Manager
  - `squash.apk` ..... prototyp balíčku pro squash
- `prirucka.pdf` ..... uživatelská příručka
- `doc` ..... adresář s dokumentací zdrojových kódů
- `src`
  - `TournamentManager` ..... repozitář projektu, se zdrojovými kódy
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF

Příloha C

Diagramy životního cyklu



Obrázek C.1: Diagram životního cyklu aktivity



Obrázek C.2: Diagram životního cyklu fragmentu