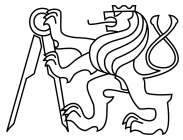
ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ



# ZADÁNÍ DIPLOMOVÉ PRÁCE

| | |
|---|---|
| **Název:** | Tournament Manager - jádro aplikace |
| **Student:** | Bc. Josef Němeček |
| **Vedoucí:** | Ing. Zdeněk Rybola |
| **Studijní program:** | Informatika |
| **Studijní obor:** | Webové a softwarové inženýrství |
| **Katedra:** | Katedra softwarového inženýrství |
| **Platnost zadání:** | Do konce letního semestru 2016/17 |

## Pokyny pro vypracování

Vytvořte jádro aplikace Tournament Manager pro mobilní zařízení s OS Android. Aplikace by měla umožnit organizování soutěží a turnajů v různých sportech, evidenci výsledků zápasů, složení týmů a statistik hráčů. Jádro aplikace musí umožnit implementaci balíčků pro různé sporty, jejichž realizace se bude lišit v závislosti na specifikách daných sportů.

Dílčí úkoly práce:
- Analyzujte společné požadavky na aplikaci z pohledu několika vybraných sportů.
- Navrhněte architekturu a strukturu jádra aplikace podporující realizaci různých sportů formou samostatných balíčků.
- Navrhněte knihovnu základních struktur a funkcí společných pro všechny uvažované sporty, kterou mohou využít jednotlivé balíčky.
- Implementujte jádro aplikace a knihovnu dle provedeného návrhu.
- Implementovanou aplikaci řádně otestujte.
- Zdokumentujte jádro aplikace z pohledu jádra a z pohledu implementace nového balíčku, zdokumentujte knihovnu základních struktur a funkcí.

## Seznam odborné literatury

Dodá vedoucí práce.

L.S.

Ing. Michal Valenta, Ph.D.
vedoucí katedry

prof. Ing. Pavel Tvrdík, CSc.
děkan

V Praze dne 2. prosince 2015

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

Diplomová práce

# Tournament Manager – jádro aplikace

***Bc. Josef Němeček***

Vedoucí práce: Ing. Zdeněk Rybola

9. ledna 2017

# Poděkování

Děkuji Ing. Zdeňku Rybolovi za cenné rady a pomoc při vypracování této diplomové práce. Dále bych rád poděkoval kolegům Michalu Hacurovi, Václavu Chmelovi, Ondřeji Košutovi a Vlastimilu Mácovi za výbornou spolupráci na projektu, jehož je tato diplomová práce součástí.

# 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 9. ledna 2017 .....

### Odkaz na tuto práci

Němeček, Josef. *Tournament Manager – jádro aplikace.* Diplomová práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2017.

# Abstrakt

Tato diplomová práce je součástí projektu, jehož cílem je návrh a implementace mobilní a webové aplikace, která umožňuje evidovat sportovní turnaje a jejich statistiky. Práce samotná se věnuje návrhu a implementaci aplikace pro mobilní zařízení s OS Android. Tato aplikace je modulární – jednotlivé moduly implementují různé sady sportů.

**Klíčová slova** Android, turnajový manažer, modulární aplikace, sport.

# Abstract

This master thesis is part of a project, which is supposed to design and implement mobile and web application, which allows user to record tournaments and statistics in different sports. This thesis focuses on design and implementation of application for mobile device with OS Android. This application is modular – each module implements different set of sports.

**Keywords** Android, tournament manager, modular application, sport.

# Obsah

# Seznam obrázků

# Seznam tabulek

# Seznam ukázek kódu

# Úvod

Tato diplomová práce je součástí projektu, který se skládá ze tří diplomových a dvou bakalářských prací. Jejich společným cílem je vytvořit mobilní a webovou aplikaci, která umožní uživatelům evidovat a sdílet sportovní turnaje a statistiky. Tato práce se zabývá návrhem a implementací modulární aplikace pro mobilní zařízení s OS Android, respektive jejích dvou hlavních modulů – jádra a knihovny.

Další práce se zabývají návrhem a implementací aplikace pro *Universal Windows Platform*[1], webovou a serverovou částí aplikace[2], implementací modulu pro sport hokej[3] a implementací modulu pro sport squash[4].

## Struktura práce

Tato práce se dělí na čtyři kapitoly. První kapitola se věnuje nejprve analýze různých sportů a poté především definici požadavků a případů užití – v této kapitole tedy specifikujeme naše očekávání od aplikace. Druhá kapitola se věnuje návrhu a popisu architektury aplikace. Popisujeme zde tedy, jakým způsobem můžeme dosáhnout našich očekávání z první kapitoly, a nahlížíme na architekturu aplikace z několika úhlů. Ve třetí kapitole si ukážeme některé úryvky z kódu a představíme některé technologie a jejich použití. V poslední čtvrté kapitole se věnujeme několika druhům testování, které jsou při vývoji software takřka nezbytné a které nám pomůže k rychlejšímu odhalování chyb.

## Cíle práce

Hlavním cílem této práce je tedy navrhnout a vytvořit jádro a knihovnu mobilní aplikace pro OS Android, která umožní evidovat a sdílet sportovní turnaje a statistiky. Jádro společně s knihovnou musí poskytovat základní funkcionalitu, která je mezi sporty sdílená. Specifika daných sportů implementují jednotlivé moduly a nejsou součástí jádra či knihovny.

Konkrétní dílčí cíle práce byly tedy definovány takto:

- analyzovat společné požadavky na aplikaci z pohledu několika vybraných sportů,
- navrhnout architekturu a strukturu jádra aplikace podporující realizaci různých sportů formou samostatných modulů,
- navrhnout knihovnu základních struktur a funkcí (společných pro všechny uvažované sporty), které mohou využít jednotlivé moduly,
- implementovat jádro aplikace a knihovnu dle provedeného návrhu,
- otestovat implementovanou aplikaci,
- zdokumentovat jádro aplikace z pohledu jádra a z pohledu implementace nového modulu,
- zdokumentovat knihovnu základních struktur a funkcí.

Kapitola 1

# Analýza

V této kapitole si nejprve zanalyzujeme sporty podle různých kritérií. Poté definujeme požadavky aplikace, z čehož nám budou ihned jasná naše očekávání od aplikace. Na základě těchto požadavků provedeme rešerši konkurence, abychom zjistili, zda již taková aplikace na trhu existuje či nikoli. Poté následují případy užití, tedy jak by měl uživatel aplikaci používat, a nakonec si představíme doménový model.

## 1.1 Dělení sportů

Na světě existuje obrovské množství sportů, které se mohou z pohledu sportovce velmi lišit. Tyto sporty se dají dělit podle různých kritérií. Podle jednoho z kritérií tak můžeme dělit sporty na individuální a kolektivní. Podle dalšího na míčové sporty (např. fotbal, volejbal), sporty zaměřené na přesnost (např. biatlon, lukostřelba), bojové sporty (např. judo, box), silové sporty (např. vzpírání) atd. Naším úkolem je co možná největší množství sportů analyzovat a pokusit se najít jejich společné charakteristiky, případně najít takové rozdělení, které nám v tom pomůže.

Jelikož je naším hlavním cílem evidovat sportovní statistiky, má pro nás největší smysl rozdělení z pohledu zaznamenání výsledku utkání. Na ten má největší vliv počet účastníků a podle něho můžeme rozdělit většinu sportů do dvou následujících kategorií – zápasové a závodní.

### 1.1.1 Zápasové sporty

Do této kategorie patří sporty, kde se vždy utkávají dva účastníci (týmy či jednotlivci). Tito účastníci svým výkonem většinou přímo ovlivňují výkon soupeře a tedy i výsledek utkání. Řadíme sem tedy např. veškeré míčové a bojové sporty. Dále však i např. lukostřelbu či šipky – účastníci sice svým výkonem neovlivňují soupeřův výkon, nicméně z historických důvodů se proti sobě

v těchto sportech utkávají také pouze dva účastníci, a proto je řadíme do této kategorie.

### 1.1.2 Závodní sporty

Do této kategorie patří sporty, kde se utkává větší množství účastníků najednou. Každý z těchto účastníků většinou podává individuální výkon, který je následně poměřován s výkony ostatních. Sem patří např. atletika, cyklistika, motorsport, golf, bowling a jiné. U běžeckých závodů v atletice nebo motorsportu sice výkony jednotlivých účastníků přímo ovlivňují výkony ostatních, ale z důvodu většího počtu účastníků v rámci jednoho závodu řadíme tyto sporty sem.

## 1.2 Požadavky

Celý systém se bude skládat z mobilní a webové aplikace. Mobilní aplikace je nezbytná součást systému, prostřednictvím které uživatel zadává veškerá data. Webová aplikace umožňuje synchronizaci dat mezi více uživateli a prezentaci dat dalším uživatelům a uživatel tuto aplikaci nemusí vůbec použít. Jelikož se webové aplikaci věnuje samostatná diplomová práce, budeme se zabývat pouze požadavky na mobilní aplikaci, a sice pro OS Android.

Obecným hlavním cílem aplikace je evidovat odehraná sportovní utkání včetně soupisek obou účastníků (v případě, že se jedná o kolektivní sport) a jejich individuálních či týmových statistik. Tato utkání se odehrávají v rámci turnaje. Tyto a další požadavky na aplikaci vznikly ve spolupráci se zadavatelem práce, který je také jejím cílovým uživatelem. Požadavky si rozdělíme na funkční a nefunkční.

### 1.2.1 Názvosloví

V následujících kapitolách budeme používat pojmy, které se sice mohou jevit jako zřejmé, ale ne vždy tomu tak je. Použitím následujících pojmů dále v práci myslíme vždy význam popsaný v této sekci, pokud nebude řečeno jinak. Tímto bychom se měli vyvarovat případným nedorozuměním.

**Soutěž** Nejvyšší úroveň opakujících se sportovních událostí pro daný sport. V rámci soutěže se utkávají jednotlivci či týmy, ale nevyhodnocují se výsledky. Příkladem mohou být florbalová superliga, 1. fotbalová liga, atp.

**Turnaj** Konkrétní sportovní událost v rámci soutěže, během které se evidují výsledky jednotlivých zápasů, vyhodnocuje se úspěšnost, určuje se vítěz atp. Příkladem mohou být konkrétní sezóna hokejové ligy nebo zkrátka turnaj, jak ho obecně chápeme – utkání několika účastníků o celkové vítěství. Turnajem

týmů se rozumí turnaj v rámci týmové soutěže, turnajem jednotlivců se rozumí turnaj v rámci soutěže jednotlivců.

**Tým** Sdružení několika hráčů pod jednotným jménem. Příkladem může být FK Dukla Praha v 1. fotbalové lize.

**Zápas** Utkání dvou či více (podle typu sportu) účastníků v rámci turnaje.

**Účastník** Tým či jednotlivec (podle typu soutěže) účastnící se zápasu.

**Kolo** Množina zápasů, ve které hraje každý účastník turnaje s každým právě jednou. Příkladem může být podzimní část 1. fotbalové ligy. Neplést tedy s kolem, jak je nazýván jeden hrací víkend právě např. v 1. fotbalové lize.

**Perioda** Množina zápasů, ve které hraje každý účastník turnaje právě jeden zápas (příp. jeden účastník odpočívá, pokud má turnaj lichý počet účastníků). Příkladem je jeden hrací víkend 1. fotbalové ligy.

**Hráč** Nejmenší stavební jednotka, ze které se skládají týmy (pokud se jedná o týmovou soutěž). Není třeba nijak dále vysvětlovat, příkladem může být Jaromír Jágr.

### 1.2.2 Funkční požadavky

Tyto požadavky definují funkcionalitu mobilní aplikace, popisují „co“ by měla či neměla dělat, nikoli však „jak“ by toho měla dosáhnout.

**F1 Hráči** Aplikace bude evidovat hráče. U hráčů evidujeme následující údaje: jméno, e-mailovou adresu a poznámku. Údaje o hráči bude možné kdykoli změnit. Hráče bude možné odstranit, pouze pokud se neúčastní žádné soutěže v žádném sportu.

**F1.1 Řazení hráčů** Aplikace umožní řadit hráče dle jména či e-mailové adresy. Ve výchozím stavu budou hráči seřazeni vzestupně podle jména.

**F2 Sporty** Aplikace umožní uživateli vybrat si sporty, které chce používat. Ostatní sporty se v aplikaci nebudou zobrazovat, dokud si je uživatel znovu nezvolí. Tento požadavek úzce souvisí s požadavkem N3 Rozšiřitelnost aplikace.

**F3 Soutěže** Aplikace bude evidovat soutěže. Soutěž vždy patří ke konkrétnímu sportu a evidujeme u ní následující údaje: název, datum začátku, datum konce, typ soutěže (soutěž jednotlivců či týmů) a poznámku. Údaje o soutěži bude možné kdykoli změnit. Soutěž bude možné odstranit, pouze pokud neobsahuje žádné hráče či turnaje.

**F3.1 Řazení soutěží** Aplikace umožní řadit soutěže v rámci daného sportu dle názvu, data začátku či data konce. Ve výchozím stavu budou soutěže seřazeny sestupně podle data konce.

**F3.2 Hráči v soutěži** Aplikace bude evidovat hráče, kteří se účastní dané soutěže. Do soutěže mohou být přidáni všichni hráči, kteří se jí ještě neúčastní. Odebrat můžeme pouze ty hráče, kteří se neúčastní žádného turnaje v rámci dané soutěže.

**F4 Turnaje** Aplikace bude evidovat turnaje. Evidujeme u něho následující údaje: název, datum začátku, datum konce a poznámku. Turnaj vždy patří ke konkrétní soutěži a utkávají se v něm jednotlivci či týmy v závislosti na typu dané soutěže. Údaje o turnaji bude možné kdykoli změnit. Turnaj bude možné odstranit, pouze pokud neobsahuje žádné hráče, týmy (pokud se jedná o týmový turnaj) či zápasy.

**F4.1 Řazení turnajů** Aplikace umožní řadit turnaje v rámci dané soutěže dle názvu, data začátku či data konce. Ve výchozím stavu budou turnaje seřazeny sestupně podle data konce.

**F4.2 Hráči v turnaji** Aplikace bude evidovat hráče, kteří se účastní daného turnaje. Do turnaje mohou být přidáni všichni hráči, kteří se účastní soutěže daného turnaje a kteří se daného turnaje ještě neúčastní. Odebrat můžeme pouze ty hráče, kteří se neúčastní žádného zápasu či nejsou součástí žádného týmu (pokud se jedná o turnaj týmů) v rámci daného turnaje.

**F4.3 Bodová konfigurace** Aplikace umožní nastavit tzv. bodovou konfiguraci turnaje – počet bodů za vítězství, remízu či prohru, v závislosti na typu sportu.

**F4.4 Pořadí** Aplikace zobrazí uživateli informaci o pořadí jednotlivců či týmů (pokud se jedná o turnaj týmů). Toto pořadí se odvíjí od pravidel daného sportu, většinou je dáno počtem získaných bodů. Uživatel bude mít možnost seřadit jednotlivce či týmy i podle jiných zobrazovaných statistik, např. počtu odehraných zápasů.

**F5 Týmy** Aplikace bude evidovat týmy, které se účastní daného turnaje (pokud se jedná o turnaj týmů). U týmu evidujeme pouze jeho název.

**F5.1 Soupiska** Aplikace bude evidovat soupisku týmu, tedy složení týmu z jednotlivých hráčů. Do týmu mohou být přidáni všichni hráči, kteří se účastní příslušného turnaje a současně nejsou na soupisce žádného jiného týmu v turnaji. Soupisku bude možné kdykoli změnit, tato změna se projeví při vytváření nových zápasů a neovlivní nijak stávající zápasy.

**F5.2 Generování soupisek** Aplikace umožní generovat soupisky týmů v turnaji. Hráči, kteří se účastní daného turnaje, budou rozdělení do týmů náhodně či na základě uživatelem vybrané hráčské statistiky.

**F6 Zápasy** Aplikace bude evidovat zápasy. Zápas vždy patří ke konkrétnímu turnaji a evidujeme u něho následující údaje: periodu, kolo, datum, poznámku a výsledek (v závislosti na druhu sportu). Aplikace umožní ruční vytvoření zápasu, kdy uživatel specifikuje výše zmíněné údaje (kromě výsledku) a účastníky zápasu. Výsledek zápasu je možné vyplnit kdykoli po vytvoření zápasu, do té doby se zápas považuje za neodehraný. Údaje o zápase bude možné kdykoli změnit. Zápas bude možné kdykoli odstranit. Aplikace také umožní „vynulovat“ zápas – tedy odstranit výsledek zápasu a považovat zápas znovu za neodehraný.

**F6.1 Účastníci** Aplikace bude evidovat účastníky zápasu. V případě, že se jedná zápas v rámci turnaje týmů, bude aplikace evidovat i složení týmu v daném zápase. Toto složení se může lišit od složení týmu v turnaji (někteří hráči např. nemusí do zápasu nastoupit apod.).

**F6.2 Hráči v zápase** Aplikace bude evidovat soupisky týmů v zápase (pokud se jedná o turnaj týmů). Tyto soupisky odpovídají soupiskám týmu v době vytvoření zápasu a je možné je kdykoli změnit. Tato změna nemá vliv na soupisku týmu v turnaji či ostatních zápasech.

**F6.3 Statistiky hráčů v zápase** Aplikace bude evidovat statistiky hráčů v zápase v závislosti na daném sportu. Sledované statistiky bude možné u jednotlivých hráčů účastnících se zápasu vyplnit současně s výsledkem zápasu.

**F6.4 Generování zápasů** Aplikace umožní generovat zápasy systémem každý s každým pro sporty, kde se utkávají dva účastníci. Aplikace vygeneruje jedno kolo zápasů, kde účastníky zápasů jsou všichni hráči v turnaji či týmy v turnaji (pokud se jedná o turnaj týmů). Při generování dalšího kola jsou prohozeny strany účastníků oproti kolu předchozímu.

**F7 Hráčské statistiky** Aplikace bude agregovat individuální hráčské statistiky. Statistiky budou agregovány ve třech úrovních: napříč všemi soutěžemi v rámci daného sportu, v rámci jedné soutěže a v rámci jednoho turnaje. Agregační funkce budou použity v závislosti na sledované hráčské statistice daného sportu, např. celkový počet odehraných zápasů či průměrný počet vstřelených branek na utkání.

**F7.1 Řazení hráčských statistik** Aplikace umožní řadit hráčské statistiky dle jakékoli vybrané statistiky.

**F8.1 Export** Aplikace umožní exportovat vybranou soutěž do souboru.

**F8.2 Import** Aplikace umožní importovat vybranou soutěž ze souboru.

### 1.2.3 Nefunkční požadavky

**N1 Minimální podporovaná verze OS Android** Aplikace bude schopna fungovat na OS Android verze 4.0 a vyšší[1].

**N2 Připojení k internetu** Aplikace nebude vyžadovat permanentní připojení k internetu. Připojení k internetu bude nutné pouze pro účely synchronizace soutěží.

**N3 Rozšiřitelnost aplikace** Aplikace bude rozšiřitelná pomocí instalace dalších aplikací, tzv. modulů. Jednotlivé moduly by měly nabízet podporu pro různé množiny sportů. Tento požadavek úzce souvisí s požadavkem F2 Sporty.

**N4 Lokalizace** Aplikace bude lokalizovaná do českého a anglického jazyka. Výběr jazyka je dán aktuálním nastavením telefonu. Pokud je jazykem telefonu čeština, je aplikace v češtině, ve všech ostatních případech je aplikace v angličtině.

## 1.3 Vymezení práce

Jelikož je cílem této práce navrhnout a implementovat jádro a knihovnu aplikace, nebudeme v rámci ní realizovat všechny funkční požadavky – některé z nich jsou totiž implementovány v modulech. Toto rozdělení můžeme vidět v tabulce 1.1. V tabulce je u jádra a knihovny uvedeno *Ano* tam, kde tyto komponenty přispívají při pokrývání daného požadavku větší měrou. Knihovna se totiž jinak podílí téměř na všech požadavcích a tato tabulka by tak neměla žádnou vypovídající hodnotu.

Nefunkční požadavky v rámci této práce realizujeme všechny.

[1] Nižší verze používá méně než 3 % všech uživatelů OS Android.[5]

Tabulka 1.1: Tabulka pokrytí funkčních požadavků

| **Požadavek** | **Jádro + knihovna** | **Modul** |
|---|---|---|
| F1 Hráči | Ano | Ne |
| F1.1 Řazení hráčů | Ano | Ne |
| F2 Sporty | Ano | Ano |
| F3 Soutěže | Ano | Ano |
| F3.1 Řazení soutěží | Ano | Ne |
| F3.2 Hráči v soutěži | Ne | Ano |
| F4 Turnaje | Ne | Ano |
| F4.1 Řazení turnajů | Ne | Ano |
| F4.2 Hráči v turnaji | Ne | Ano |
| F4.3 Bodová konfigurace | Ne | Ano |
| F4.4 Pořadí | Ne | Ano |
| F5 Týmy | Ne | Ano |
| F5.1 Soupiska | Ne | Ano |
| F5.2 Generování soupisek | Ano | Ano |
| F6 Zápasy | Ne | Ano |
| F6.1 Účastníci | Ne | Ano |
| F6.2 Hráči v zápase | Ne | Ano |
| F6.3 Statistiky hráčů v zápase | Ne | Ano |
| F6.4 Generování zápasů | Ano | Ano |
| F7 Hráčské statistiky | Ano | Ano |
| F7.1 Řazení hráčských statistik | Ne | Ano |
| F8.1 Export | Ano | Ano |
| F8.2 Import | Ano | Ano |

## 1.4 Rešerše existujících řešení

V této sekci si nejprve definujeme množinu požadavků, na základě kterých se pokusíme prozkoumat trh s existujícími aplikacemi. Následně si popíšeme jednotlivé nalezené aplikace a jejich případné rozdíly oproti těmto a dalším požadavkům. Nakonec si popsané aplikace shrneme a pokusíme se vybrat jednu, která by našim požadavkům přesně vyhovovala. V této sekci se na základě výše popsaných požadavků pokusíme prozkoumat existující aplikace. Pokud by se nám podařilo najít vyhovující aplikaci, bylo by pro nás výhodnější používat tuto již existující aplikaci, než se zabývat vytvářením nové.

### 1.4.1 Množina požadavků

Výše popsané požadavky jsou příliš striktní na to, abychom podle nich mohli hledat aplikaci s reálnou nadějí, že se nám ji podaří najít. Proto vybereme pouze malou podmnožinu požadavků, podle které budeme aplikaci hledat:

- aplikace podporuje evidenci soutěží, díky kterým je možné oddělit určité zápasy a týmy od ostatních zápasů a týmů,
- aplikace nesmí být zaměřena pouze na jediný sport, ale musí umožňovat evidenci soutěží ve více sportech,
- aplikace nesmí vyžadovat permanentní připojení k internetu, musí být schopna fungovat v tzv. offline režimu.

### 1.4.2 Aplikace v obchodě Google Play

Níže popsané aplikace byly nalezeny v obchodě Google Play po zadání klíčových slov „competition manager“ a „tournament manager“. Vzhledem k velkému počtu takových aplikací prozkoumáme jen několik nejlépe uživatelsky hodnocených.

#### 1.4.2.1 Virtual Competition Manager

Tato aplikace se zaměřuje pouze na soutěže a turnaje ve fotbale, což však poznáme pouze podle loga. Oproti námi požadované aplikaci nabízí navíc možnost vyřazovacího turnaje. Chybí zde však jakákoli možnost evidovat individuální statistiky.

Splnění podmnožiny požadavků:

- evidence soutěží – splněno,
- podpora více sportů – pouze zápasové sporty s právě dvěma účastníky,
- offline režim – splněno.

Odkaz do obchodu (21. 12. 2016): `https://play.google.com/store/apps/details?id=com.poquesoft.mistorneos`.

#### 1.4.2.2 Bracket Maker & Tournament App

Tato aplikace se jeví jako velmi propracovaná, opět má možnost vyřazovacího turnaje. Velká množina funkcionality (včetně např. individuálních statistik) je dostupná pouze v placené verzi. Její další nevýhodou je nutnost permanentního internetového připojení.

Splnění podmnožiny požadavků:

- evidence soutěží – splněno,
- podpora více sportů – splněno,
- offline režim – nesplněno.

Odkaz do obchodu (21. 12. 2016): `https://play.google.com/store/apps/details?id=com.mileyenda.manager`.

#### 1.4.2.3 TournamentManager

Tato aplikace je velmi snadná na používání a v zásadě splňuje naši podmnožinu požadavků. Nijak nás neomezuje ve výběru sportu, ovšem stejně jako u aplikace Virtual Competition Manager máme možnost evidovat pouze zápasové sporty s dvěma účastníky a nemáme žádnou možnost evidovat individuální statistiky.

Splnění podmnožiny požadavků:

- evidence soutěží – splněno,
- podpora více sportů – pouze zápasové sporty s právě dvěma účastníky,
- offline režim – splněno.

Odkaz do obchodu (21. 12. 2016): `https://play.google.com/store/apps/details?id=turnier.manager`.

#### 1.4.2.4 Best Tournament Manager

Tato aplikace podporuje pět vybraných sportů . Oproti našim požadavkům nabízí možnost konfigurace výchozího pořadí v turnaji – vítězem turnaje se tak např. nemusí stát tým s největším počtem získaných bodů. Aplikace však nemá možnost evidovat individuální statistiky či později přidávat týmy či zápasy k turnaji.

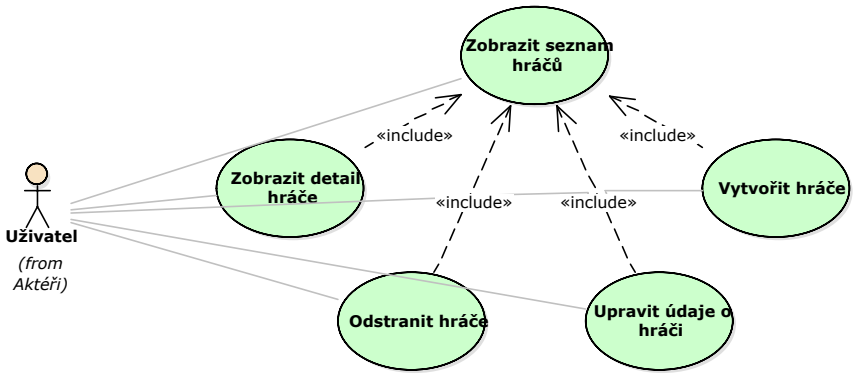
Splnění podmnožiny požadavků:

- evidence soutěží – splněno,
- podpora více sportů – splněno,
- offline režim – splněno.

Odkaz do obchodu (21. 12. 2016): `https://play.google.com/store/apps/details?id=com.mds.vbtm`.

### 1.4.3 Shrnutí

Podařilo se nám tedy najít aplikaci, která vyhovuje zadané podmnožině požadavků. Pokud bychom však uvážili i ostatní požadavky, nebude už vyhovovat žádná, a proto se budeme v následující části práce věnovat návrhu vlastní. Výše zmíněné aplikace nám však mohou posloužit jako inspirace.




Obrázek 1.1: Případy užití – hráči

## 1.5 Případy užití

V této sekci se budeme věnovat případům užití. Případy užití nám poslouží jako detailnější specifikace funkčních požadavků a také je můžeme využít jako základ pro tvorbu uživatelské příručky. Nejprve identifikujeme aktéry aplikace a poté samotné případy užití rozdělené do jednotlivých kategorií podobně jako byly řazeny funkční požadavky.

### 1.5.1 Aktéři

Jako jediného aktéra aplikace identifikujeme *uživatele*. Tento uživatel je běžný uživatel aplikace a má přístup ke všem funkcionalitám aplikace. V aplikaci neexistují žádné uživatelské role. Všechny následující případy užití jsou tedy dostupné pro tohoto aktéra. Většina z nich je navíc velmi jednoduchá, proto je nebudeme zbytečně dlouze rozepisovat.

### 1.5.2 Hráči

První skupinou případů užití jsou *hráči*. Následující případy užití se týkají požadavků F1 Hráči a F1.1 Řazení hráčů. Diagram těchto případů je na obrázku 1.1.

**UC1.1 Zobrazit seznam hráčů** Uživatel zvolí z menu možnost pro zobrazení hráčů a aplikace zobrazí seznam hráčů v aplikaci. Uživatel má možnost tento seznam seřadit podle údajů o hráči – jména či e-mailové adresy.

**UC1.2 Vytvořit hráče** Uživatel si *zobrazí seznam hráčů*, zvolí možnost přidat nového hráče a vyplní příslušné údaje. V případě, že jsou údaje vyplněné


Obrázek 1.2: Případy užití – nastavení

správně, uživatel je uložen a zobrazen v seznamu hráčů. V opačném případě je uživateli zobrazena hláška o nesprávnosti zadaných údajů.

**UC1.3 Zobrazit detail hráče** Uživatel si *zobrazí seznam hráčů* a zvolí možnost zobrazit detail konkrétního hráče. V detailu hráče jsou zobrazeny údaje o hráči a pro každý sport také seznam soutěží, kterých se účastní, a agregované hráčské statistiky pro daný sport.

**UC1.4 Upravit údaje o hráči** Uživatel si *zobrazí seznam hráčů* a zvolí možnost upravit údaje o konkrétním hráči (alternativou je *zobrazit detail hráče* a zvolit možnost upravit údaje o hráči). Validace údajů probíhá stejně jako v UC1.2.

**UC1.5 Odstranit hráče** Uživatel si *zobrazí seznam hráčů* a zvolí možnost odstranit konkrétního hráče. V případě, že se hráč neúčastní žádné soutěže v žádném sportu, je odstraněn. V opačném případě je uživateli zobrazena hláška, že se hráč účastní soutěže a tudíž nemůže být odstraněn.

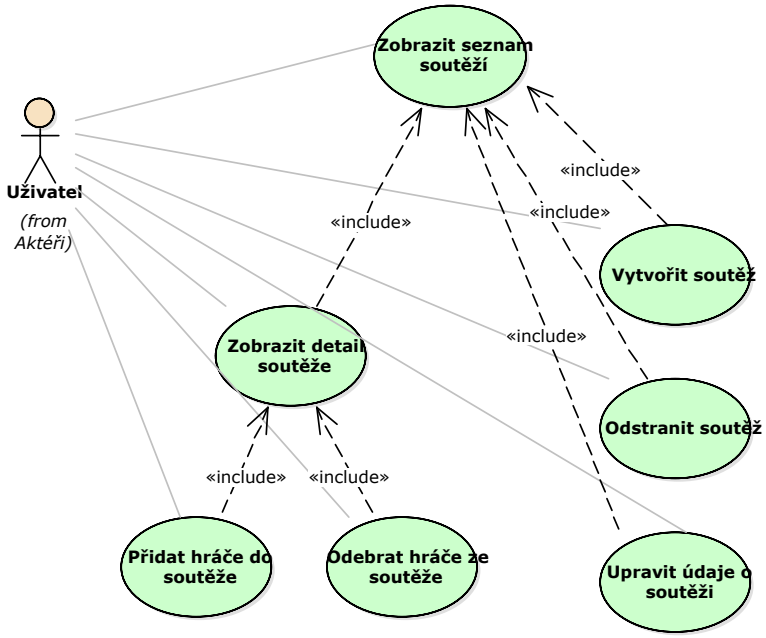
### 1.5.3 Nastavení

Druhou kategorií případů užití je *nastavení*. Následující případy užití se týkají požadavku F2 Sporty. Diagram případů užití této kategorie je na obrázku 1.2.

**UC2.1 Nastavit zobrazované sporty** Uživatel zvolí z menu možnost nastavení a aplikace zobrazí seznam dostupných sportů. Ve výchozím stavu jsou všechny sporty zvolené pro zobrazení. Uživatel má možnost zaškrtnutím a odškrtnutím zvolit sporty, které se mají v aplikaci zobrazovat. Toto nastavení má vliv na seznam sportů zobrazený v *detailu hráče* a *seznamu soutěží*.

### 1.5.4 Soutěže

Třetí kategorií případů užití jsou *soutěže*. Následující případy užití se týkají požadavků F3 Soutěže, F3.1 Řazení soutěží a F3.2 Hráči v soutěži. Diagram případů užití této kategorie je na obrázku 1.3.



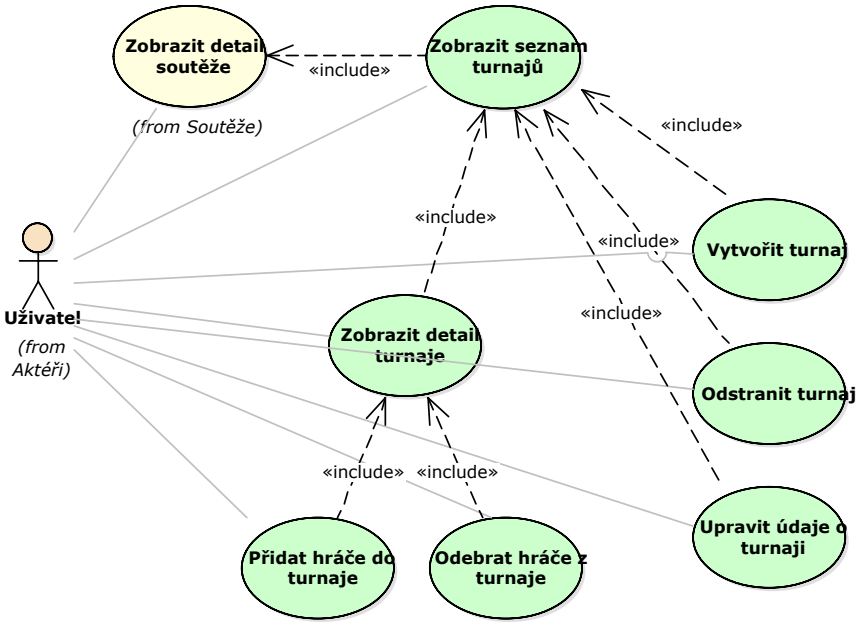

Obrázek 1.3: Případy užití – soutěže

**UC3.1 Zobrazit seznam soutěží** Uživatel zvolí z menu možnost pro zobrazení seznamu soutěží a aplikace zobrazí seznamy soutěží seskupené podle jednotlivých sportů. Uživatel má možnost tento seznam seřadit podle údajů o soutěži – názvu, data začátku či data konce.

**UC3.2 Vytvořit soutěž** Uživatel si *zobrazí seznam soutěží*, zvolí možnost přidat novou soutěž a vyplní příslušné údaje. V případě, že jsou údaje vyplněné správně, soutěž je uložena a zobrazena v seznamu soutěží. V opačném případě je uživateli zobrazena hláška o nesprávnosti zadaných údajů.

**UC3.3 Zobrazit detail soutěže** Uživatel si *zobrazí seznam soutěží* a zvolí možnost zobrazit detail konkrétní soutěže. V detailu soutěže jsou zobrazeny údaje o soutěži, seznam turnajů a seznam hráčů společně s jejich agregovanými statistikami pro danou soutěž.

**UC3.4 Upravit údaje o soutěži** Uživatel si *zobrazí seznam soutěží* a zvolí možnost upravit údaje o konkrétní soutěži (alternativou je *zobrazit detail soutěže* a zvolit možnost upravit údaje o soutěži). Validace údajů probíhá stejně jako v UC3.2. Typ soutěže nemůže být zpětně změněn.




Obrázek 1.4: Případy užití – turnaje

**UC3.5 Odstranit soutěž** Uživatel si *zobrazí seznam soutěží* a zvolí možnost odstranit konkrétní soutěž. V případě, že soutěž neobsahuje žádné turnaje ani hráče, je odstraněna. V opačném případě je uživateli zobrazena hláška, že soutěž není prázdná a tudíž nemůže být odstraněna.

**UC3.6 Přidat hráče do soutěže** Uživatel si *zobrazí detail soutěže* a zvolí možnost přidat hráče do soutěže. Aplikace nabídne uživateli všechny hráče, kteří ještě v této soutěži nejsou. Uživatel vybere požadované hráče a přidá tyto hráče do soutěže.

**UC3.7 Odebrat hráče ze soutěže** Uživatel si *zobrazí detail soutěže* a ze seznamu hráčů v soutěži zvolí možnost odebrat konkrétního hráče ze soutěže. V případě, že se hráč neúčastní žádného turnaje, je ze soutěže odstraněn. V opačném případě je uživateli zobrazena hláška, že se hráč účastní nějakého turnaje a tudíž nemůže být odstraněn.

### 1.5.5 Turnaje

Čtvrtou kategorií případů užití jsou *turnaje*. Následující případy užití se týkají požadavků F4 Turnaje, F4.1 Řazení turnajů, F4.2 Hráči v turnaji, F4.3 Bodová

konfigurace a F4.4 Pořadí. Diagram případů užití této kategorie je na obrázku 1.4.

**UC4.1 Zobrazit seznam turnajů** Uživatel si *zobrazí detail soutěže* a vybere možnost pro zobrazení seznamu turnajů v dané soutěži. Aplikace zobrazí seznam turnajů. Uživatel má možnost tento seznam seřadit podle údajů o turnaji – názvu, data začátku či data konce.

**UC4.2 Vytvořit turnaj** Uživatel si *zobrazí seznam turnajů*, zvolí možnost přidat nový turnaj a vyplní příslušné údaje. V případě, že jsou údaje vyplněné správně, turnaj je uložen a zobrazen v seznamu turnajů. V opačném případě je uživateli zobrazena hláška o nesprávnosti zadaných údajů.

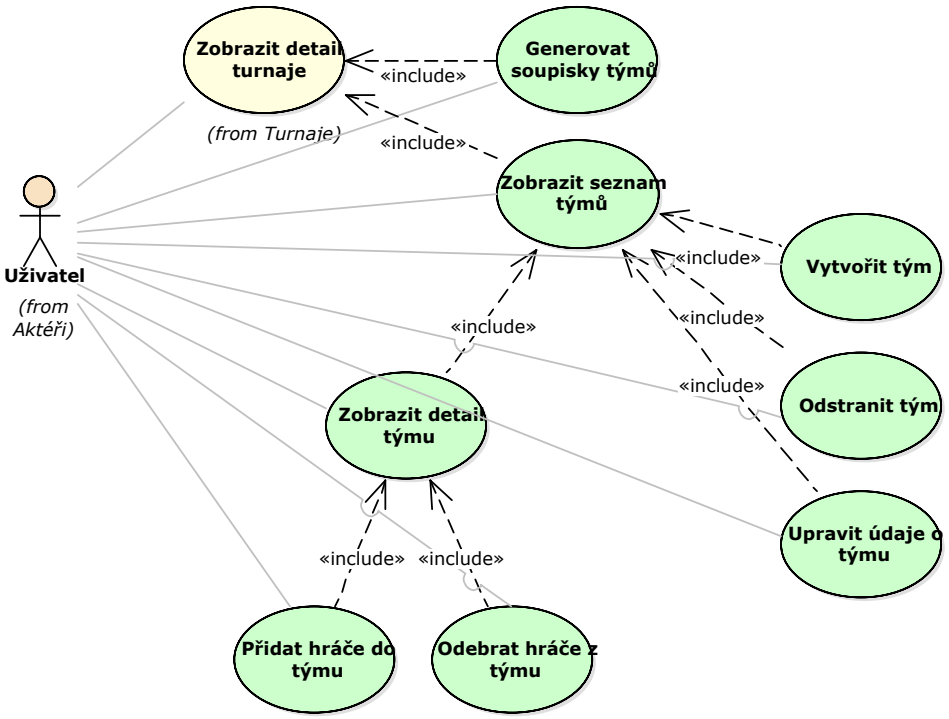
**UC4.3 Zobrazit detail turnaje** Uživatel si *zobrazí seznam turnajů* a zvolí možnost zobrazit detail konkrétního turnaje. V detailu turnaje jsou zobrazeny údaje o turnaji, pořadí účastníků, seznam zápasů, seznam týmů (pokud se jedná o turnaj týmů) a seznam hráčů společně s jejich agregovanými statistikami pro daný turnaj.

**UC4.4 Upravit údaje o turnaji** Uživatel si *zobrazí seznam turnajů* a zvolí možnost upravit údaje o konkrétním turnaji (alternativou je *zobrazit detail turnaje* a zvolit možnost upravit údaje o turnaji). Validace údajů probíhá stejně jako v UC4.2.

**UC4.5 Odstranit turnaj** Uživatel si *zobrazí seznam turnajů* a zvolí možnost odstranit konkrétní turnaj. V případě, že turnaj neobsahuje žádné zápasy, týmy (pokud se jedná o turnaj týmů) ani hráče, je odstraněn. V opačném případě je uživateli zobrazena hláška, že turnaj není prázdný a tudíž nemůže být odstraněn.

**UC4.6 Přidat hráče do turnaje** Uživatel si *zobrazí detail turnaje* a zvolí možnost přidat hráče do turnaje. Aplikace nabídne uživateli všechny hráče, kteří se účastní soutěže příslušné tomuto turnaji a ještě v tomto turnaji nejsou. Uživatel vybere požadované hráče a přidá tyto hráče do turnaje.

**UC4.7 Odebrat hráče ze turnaje** Uživatel si *zobrazí detail turnaje* a ze seznamu hráčů v turnaji zvolí možnost odebrat konkrétního hráče z turnaje. V případě, že se hráč neúčastní žádného zápasu ani není součástí žádného týmu (pokud se jedná o turnaj týmů), je z turnaje odstraněn. V opačném případě je uživateli zobrazena hláška, že se hráč účastní nějakého zápasu nebo je součástí nějakého týmu a tudíž nemůže být odstraněn.




Obrázek 1.5: Případy užití – týmy

### 1.5.6 Týmy

Pátou kategorií případů užití jsou *týmy*. Následující případy užití se týkají požadavků F5 Týmy, F5.1 Soupiska a F5.2 Generování soupisek. Stejně jako zmíněné požadavky se i tyto případy užití týkají pouze turnajů týmů. Diagram případů užití této kategorie je na obrázku 1.5.

**UC5.1 Zobrazit seznam týmů** Uživatel si *zobrazí detail turnaje* a zvolí možnost pro zobrazení seznamu týmů. Aplikace zobrazí seznam týmů včetně jeho složení.

**UC5.2 Vytvořit tým** Uživatel si *zobrazí seznam týmů*, zvolí možnost přidat nový tým a vyplní název týmu.

**UC5.3 Upravit údaje o týmu** Uživatel si *zobrazí seznam týmů*, zvolí možnost upravit konkrétní tým ze seznamu a upraví název týmu.

**UC5.4 Odstranit tým** Uživatel si *zobrazí seznam týmů* a zvolí možnost odstranit konkrétní tým ze seznamu. V případě, že se tým neúčastní žádného zápasu, je odstraněn. V opačném případě je uživateli zobrazena hláška, že se tým účastní nějakého zápasu a tudíž nemůže být odstraněn.

**UC5.5 Zobrazit detail týmu** Uživatel si *zobrazí seznam týmů* a zvolí možnost zobrazit detail konkrétního týmu. V detailu týmu je zobrazena soupiska – seznam hráčů daného týmu.

**UC5.6 Přidat hráče do týmu** Uživatel si *zobrazí detail týmu* a zvolí možnost přidat hráče do týmu. Aplikace nabídne uživateli všechny hráče, kteří se účastní daného turnaje a nejsou zatím na žádné soupisce. Uživatel vybere požadované hráče a přidá tyto hráče do týmu.

**UC5.7 Odebrat hráče z týmu** Uživatel si *zobrazí detail týmu* a ze seznamu hráčů v týmu zvolí možnost odebrat konkrétního hráče z týmu.

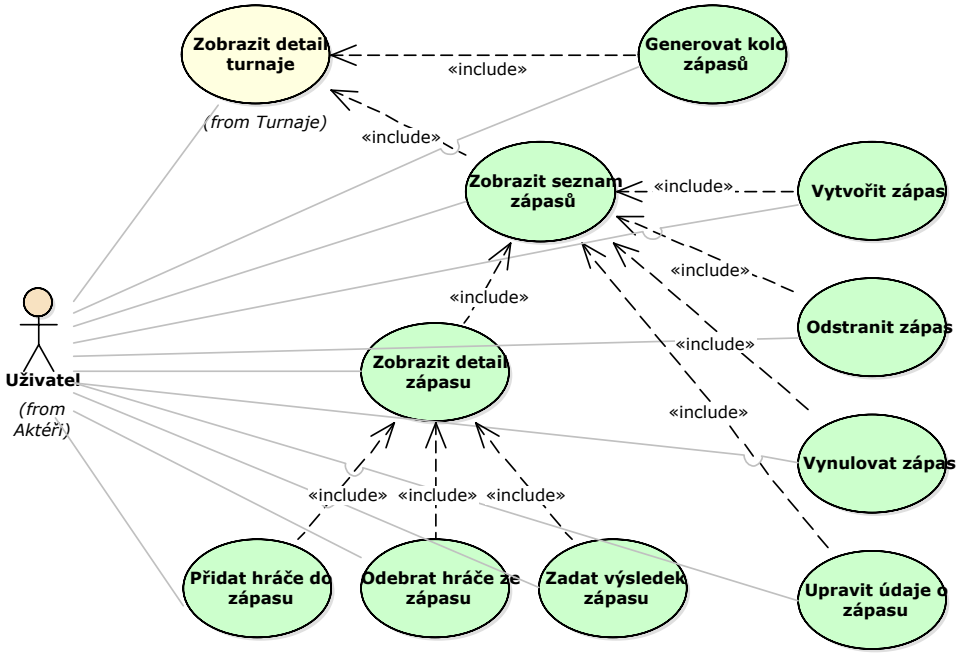
**UC5.8 Generovat soupisky týmů** Uživatel si *zobrazí detail turnaje* a zvolí možnost generovat soupisky týmů. Aplikace nabídne uživateli způsoby, jakými lze soupisky generovat. Základním způsobem je generovat soupisky náhodně, ostatní způsoby závisí na konkrétním sportu a evidovaných hráčských statistikách – jde nám totiž o generování přibližně vyrovnaných soupisek v závislosti na vybrané hráčské statistice. Pokud nebude v turnaji žádný tým nebo bude alespoň jeden z týmů obsahovat hráče na soupisce, aplikace soupisky nevygeneruje a zobrazí uživateli chybovou hlášku.

### 1.5.7 Zápasy

Šestou kategorií případů užití jsou *zápasy*. Následující případy užití se týkají požadavků F6 Zápasy, F6.1 Účastníci, F6.2 Hráči v zápase, F6.3 Statistiky hráčů v zápase, a F6.4 Generování zápasů. Diagram případů užití této kategorie je na obrázku 1.6.

**UC6.1 Zobrazit seznam zápasů** Uživatel si *zobrazí detail turnaje* a zvolí možnost zobrazit seznam zápasů. Aplikace zobrazí seznam zápasů včetně výsledků již odehraných zápasů.

**UC6.2 Vytvořit zápas** Uživatel si *zobrazí seznam zápasů*, zvolí možnost přidat nový zápas a vyplní příslušné údaje. V případě, že jsou údaje vyplněné správně, zápas je uložen a zobrazen v seznamu zápasů. V opačném případě je uživateli zobrazena hláška o nesprávnosti zadaných údajů.




Obrázek 1.6: Případy užití – zápasy

**UC6.3 Vygenerovat kolo zápasů** Uživatel si *zobrazí seznam zápasů*, zvolí možnost vygenerovat nové kolo zápasů. V případě, že turnaj obsahuje alespoň dva hráče (týmy – pokud se jedná o turnaj týmů), aplikace vygeneruje nové kolo neodehraných zápasů. V opačném případě zobrazí uživateli hlášku o nedostatečném počtu hráčů (týmů).

**UC6.4 Zobrazit detail zápasu** Uživatel si *zobrazí seznam zápasů* a zvolí možnost zobrazit detail konkrétního zápasu. Aplikace zobrazí údaje o zápase včetně výsledku a soupisek týmů (pokud se jedná o turnaj týmů) s hráčskými statistikami jednotlivých hráčů.

**UC6.5 Upravit údaje o zápase** Uživatel si *zobrazí seznam zápasů* a zvolí možnost upravit údaje o konkrétním zápase (alternativou je *zobrazit detail zápasu* a zvolit možnost upravit údaje o zápase). Aplikace zobrazí následující údaje: účastníky zápasu, kolo, periodu, datum a poznámku. Validace údajů probíhá stejně jako v UC6.2. Účastníci zápasu nemohou být zpětně změněni.

**UC6.6 Zadat výsledek zápasu** Uživatel si zobrazí detail zápasu. Aplikace mu umožní v závislosti na sportu zadat výsledek zápasu. Např. u zápasů v hokeji či florbale by uživatel zadal počet vstřelených branek obou účastníků,

u tenisu či volejbalu výsledky jednotlivých sad a u bowlingu skóre jednotlivých účastníků. Dále má uživatel možnost vyplnit hráčské statistiky hráčů obou týmů (pokud se jedná o turnaj týmů).

**UC6.7 Vynulovat zápas** Uživatel si *zobrazí seznam zápasů* a zvolí možnost vynulovat konkrétní zápas. Aplikace odstraní výsledek a všechny statistiky týkající se tohoto zápasu tak, že zápas bude neodehraný a tedy ve stavu, v jakém byl po jeho vytvoření.

**UC6.8 Odstranit zápas** Uživatel si *zobrazí seznam zápasů*, zvolí možnost odstranit konkrétní zápas a aplikace tento zápas odstraní.

**UC6.9 Přidat hráče do zápasu** Uživatel si *zobrazí detail zápasu* a zvolí možnost přidat hráče ke konkrétnímu týmu. Aplikace nabídne uživateli všechny hráče, kteří se zatím neúčastní tohoto zápasu. Uživatel vybere požadované hráče a přidá tyto hráče do zápasu.

**UC6.10 Odebrat hráče ze zápasu** Uživatel si *zobrazí detail zápasu* a ze seznamu hráčů zvolí možnost odebrat konkrétního hráče ze zápasu.

### 1.5.8 Statistiky

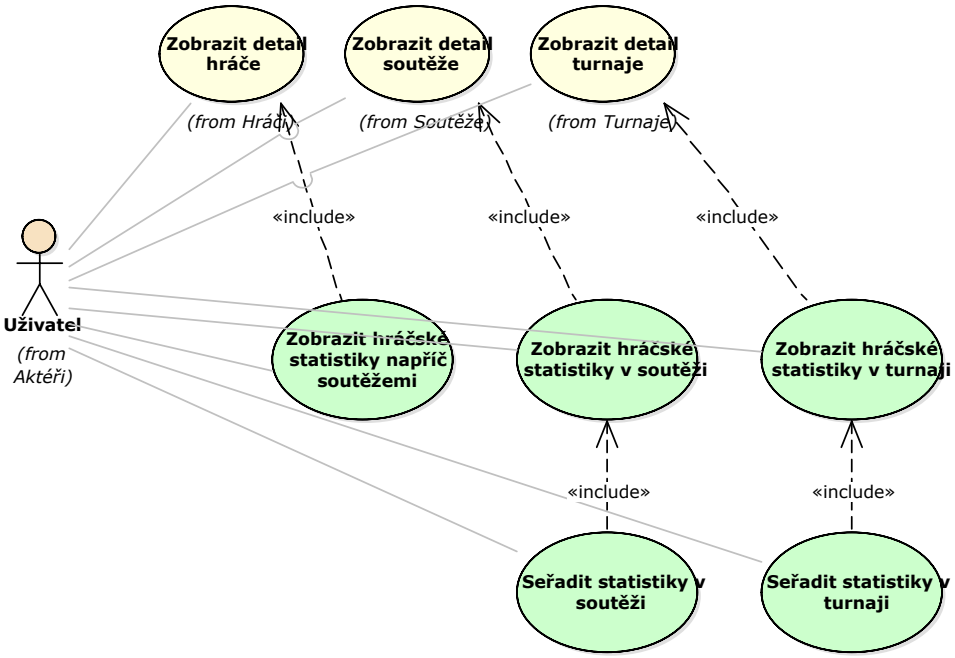
Sedmou kategorií případů užití jsou *hráčské statistiky*. Následující případy užití se týkají požadavku F7 Hráčské statistiky a F7.1 Řazení hráčských statistik. Diagram případů užití této kategorie je na obrázku 1.7.

**UC7.1 Zobrazit hráčské statistiky napříč soutěžemi** Uživatel si *zobrazí detail hráče* a vybere sport. Aplikace zobrazí agregované statistiky daného hráče napříč všemi soutěžemi ve vybraném sportu.

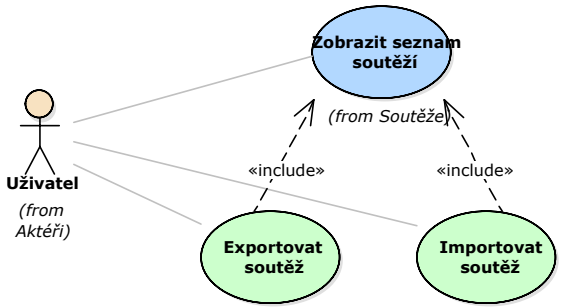
**UC7.2 Zobrazit hráčské statistiky v soutěži** Uživatel si *zobrazí detail soutěže* a zvolí možnost zobrazit hráčské statistiky. Aplikace zobrazí seznam hráčů s agregovanými statistikami v soutěži.

**UC7.3 Zobrazit hráčské statistiky v turnaji** Uživatel si *zobrazí detail turnaje* a zvolí možnost zobrazit hráčské statistiky. Aplikace zobrazí seznam hráčů s agregovanými statistikami v turnaji.

**UC7.4 Seřadit statistiky v soutěži** Uživatel si zobrazí *hráčské statistiky v soutěži*. Aplikace umožní uživateli seřadit seznam hráčů dle vybrané agregované statistiky.




Obrázek 1.7: Případy užití – statistiky




Obrázek 1.8: Případy užití – export a import

**UC7.5 Seřadit statistiky v turnaji** Uživatel si zobrazí *hráčské statistiky v turnaji*. Aplikace umožní uživateli seřadit seznam hráčů dle vybrané agregované statistiky.

### 1.5.9 Export a import

Osmou a zároveň poslední kategorií případů užití je *export a import*. Následující případy užití se týkají požadavků F8.1 Export a F8.2 Import. Diagram případů užití této kategorie je na obrázku 1.8.

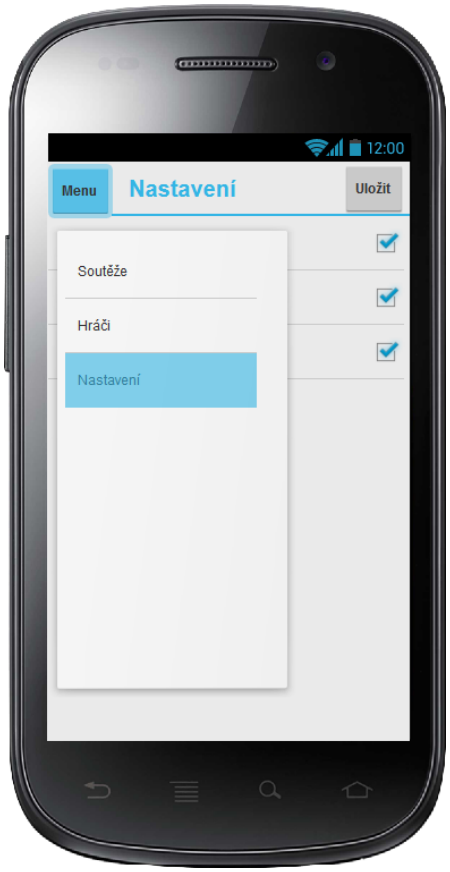
**UC8.1 Exportovat soutěž** Uživatel si *zobrazí seznam soutěží* a zvolí možnost exportovat konkrétní soutěž. Aplikace vytvoří a uloží soubor, jehož název bude tvořen názvem soutěže a časovým razítkem exportu, tedy aktuálním datem a časem.

**UC8.2 Importovat soutěž** Uživatel si *zobrazí seznam soutěží* a zvolí možnost importovat soutěž ze souboru. Aplikace prohledá příslušnou složku a nabídne uživateli soubory pro daný sport, které mohou být importovány. Uživatel si vybere soubor a aplikace zobrazí shrnutí importované soutěže, které bude obsahovat následující údaje: seznam turnajů (včetně počtu týmů, zápasů a hráčů), seznam nově vytvořených hráčů a seznam změněných hráčů oproti aplikaci. U každého hráče z posledního zmíněného seznamu bude uživatel moci vybrat, zda chce změnu aplikovat, nebo si ponechá v aplikaci stávající data. V případě, že uživatel import soutěže potvrdí, aplikace vytvoří novou soutěž identickou s exportovanou soutěží. Jediný rozdíl bude v názvu soutěže, který bude doplněn o časové razítko vytvoření, tedy aktuální datum a čas. V případě, že uživatel import soutěže zruší, vrátí se zpět do seznamu soutěží a soutěž importována nebude.

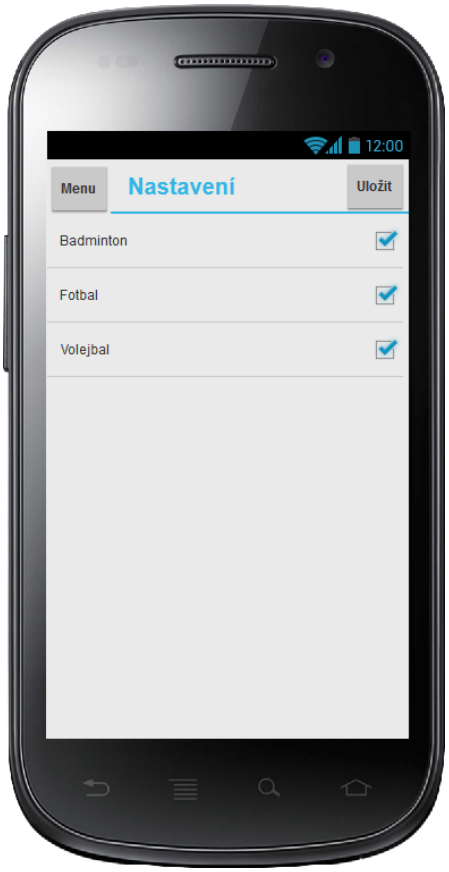
## 1.6 Návrh uživatelského rozhraní

Další klíčovou kapitolou návrhu a vývoje software je návrh uživatelského rozhraní. To popisuje způsob komunikace člověka se systémem – v tomto případě naší aplikací. Tento návrh vychází z předchozích dvou sekcí – požadavků a případů užití. V této sekci si nejprve popíšeme jednotlivé obrazovky, které bude aplikace obsahovat. Tyto obrazovky jsou rozděleny do jednotlivých kategorií podobně jako požadavky a případy užití. V každé kategorii si představíme nejprve její obrazovky a poté i graf přechodů mezi nimi. V poslední části této sekce nalezneme ještě pro úplnost obecnější graf přechodů, kde vynecháme podrobnosti popsané v jednotlivých kategoriích.

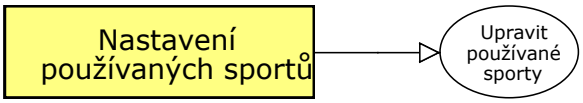
Návrh uživatelského rozhraní nemusí stoprocentně odpovídat výslednému uživatelskému rozhraní – pro návrh byla použita aplikace *Pencil*[2] a její paleta prvků dostupných pro návrh nepokrývá všechny prvky dostupné v Androidu. Kvůli stručnosti jsou v návrhu vynechány některé obrazovky či prvky. Chybí veškeré formuláře pro vytváření a úpravu hráčů, soutěží, turnajů, atd., protože jsou již detailně popsané v případech užití. Dalším chybějícím prvkem jsou




Obrázek 1.9: Hlavní menu




Obrázek 1.10: Nastavení




Obrázek 1.11: Nastavení – diagram přechodů

kontextová dialogová okna – jejich popisy se však nachází vždy u příslušné obrazovky.

### 1.6.1 Nastavení

Na obrázku 1.9 je hlavní menu aplikace. Z položek menu je vždy patrné, na jaké obrazovce se nacházíme (v tomto případě se tedy nacházíme v *Nastavení*).

Seznam dostupných sportů v aplikaci na obrázku 1.10 je první ze tří obrazovek dostupných z hlavního menu aplikace. Zaškrtnutím vybereme sporty, které chceme v aplikaci používat, a poté toto nastavení pomocí tlačítka *Uložit* v pravém horním rohu obrazovky uložíme. Touto obrazovkou pokrýváme případ užití UC2.1 Nastavit zobrazované sporty.

Pro úplnost doplníme diagram přechodů mezi obrazovkami v této kategorii na obrázku 1.11, přestože by to v tomto konkrétním případě nebylo nutné – máme zde pouze jedinou obrazovku s jednou akcí.

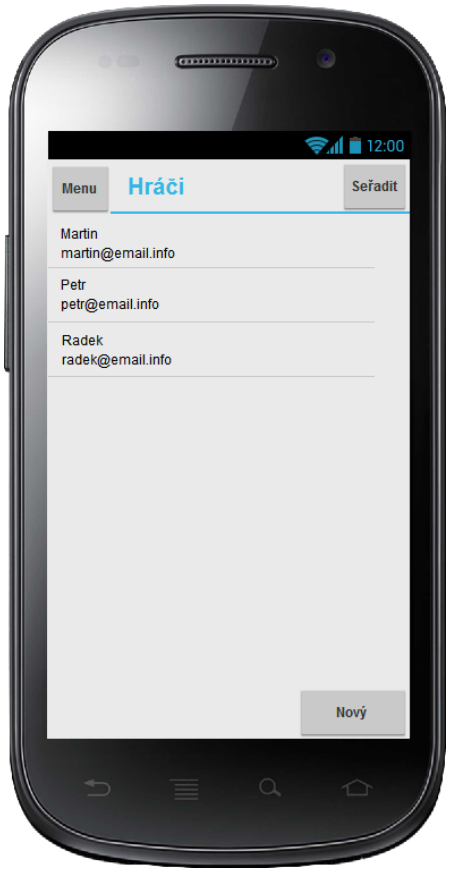
### 1.6.2 Hráči

Seznam hráčů na obrázku 1.12 je další ze tří obrazovek dostupných z hlavního menu aplikace. Hlavní menu aplikace vyvoláme stisknutím tlačítka *Menu* v levém horním rohu obrazovky či vysunutím z levé strany telefonu – toto platí pro všechny obrazovky, kde je tlačítko menu zobrazené. Pomocí tlačítka *Seřadit* v pravém horním rohu můžeme seřadit hráče – stisknutí vyvolá dialogové okno, kde si vybereme, zda chceme seřadit hráče dle jména či e-mailové adresy. Opakovaným výběrem změníme řazení ze sestupného na vzestupné a naopak. Dlouhým stiskem nad konkrétním hráčem vyvoláme dialogové okno, kde máme možnost přejít na editaci údajů o hráči či odstranit hráče. Tlačítko *Nový* v pravém dolním rohu umožňuje přidat nového hráče – stisknutí vyvolá formulář pro vyplnění údajů o hráči. Tato obrazovka odpovídá případu užití UC1.1 Zobrazit seznam hráčů, ovšem jsou z ní dostupné i tyto případy: UC1.2 Vytvořit hráče, UC1.4 Upravit údaje o hráči a UC1.5 Odstranit hráče. Obrazovky s formuláři pro vytvoření či editaci hráče jsou vynechány, jak již bylo zmíněno v úvodu kapitoly.

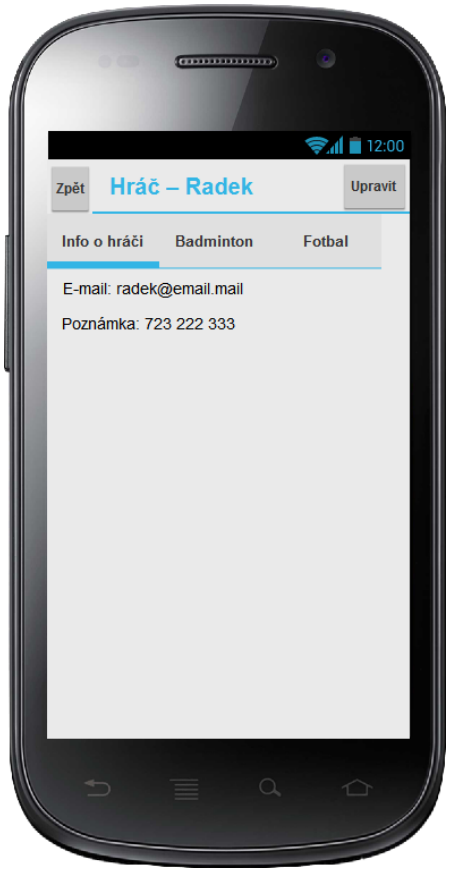
Na obrázku 1.13 je detail hráče. Tlačítkem *Zpět* v levém horním rohu obrazovky se vrátíme na seznam hráčů. Pomocí tlačítka *Upravt* v pravém horním rohu můžeme editovat údaje o hráči. Obě tato tlačítka zůstávají stejná pro všechny záložky této obrazovky. Pomocí jednotlivých záložek (na obrázku *Badminton* a *Fotbal*) se dostaneme k seznamu soutěží, kterých se hráč účastní, a agregovaným statistikám pro jednotlivé sporty. Tato obrazovka, jak je nejspíš zřejmé, vychází z případu užití UC1.3 Zobrazit detail hráče.

Na obrázku 1.14 je seznam soutěží a agregované statistiky v badmintonu. Tyto statistiky vždy závisí na sportu, údaje na obrázku jsou pouze pro ilustraci. Stisknutím konkrétní soutěže přejdeme na detail soutěže. Tato obrazovka vychází z případu užití UC7.1 Zobrazit hráčské statistiky napříč soutěžemi.

[2]Open source nástroj pro návrh Uživatelského rozhraní, dostupný z (25. 12. 2016): `http://pencil.evolus.vn/`.




Obrázek 1.12: Hráči

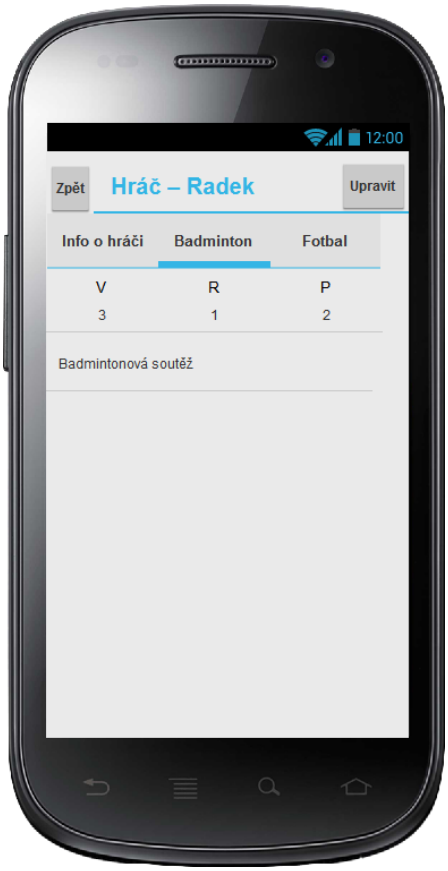



Obrázek 1.13: Detail hráče

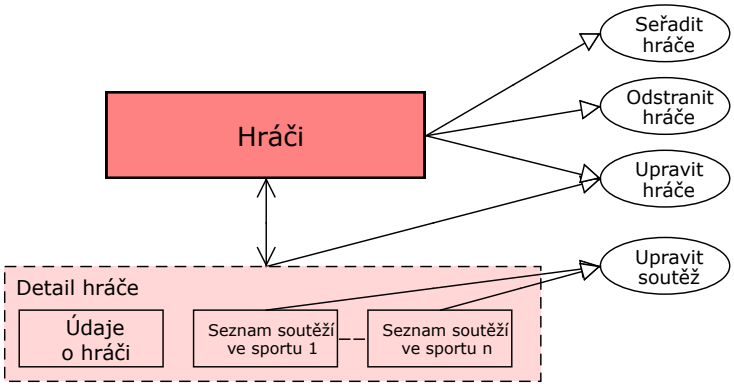
Pro přehlednost doplníme na obrázku 1.15 diagram přechodů mezi obrazovkami v této kategorii. Obdélníky s plným okrajem znamenají jednotlivé obrazovky, přerušovaným okrajem značíme souhrnné obrazovky, které se často skládají z několika dalších obrazovek, nicméně je nazýváme společným názvem. Elipsy poté znamenají dostupné akce. Šipky znázorňují možné přechody mezi obrazovkami, příp. možnost vyvolat akci. Tato notace je platná i pro všechny následující diagramy přechodů.

### 1.6.3 Soutěže

Seznam soutěží na obrázku 1.16 je poslední ze tří obrazovek dostupných z hlavního menu aplikace. Pomocí tlačítka *Seřadit* v pravém horním rohu můžeme seřadit soutěže – stisknutí vyvolá dialogové okno, kde si vybereme, zda chceme seřadit soutěže dle názvu, data začátku či data konce. Opakovaným výběrem změníme řazení ze sestupného na vzestupné a naopak. Stisknutím konkrétní soutěže přejdeme na detail soutěže. Dlouhým stiskem nad konkrétní soutěží




Obrázek 1.14: Detail hráče – soutěže a statistiky




Obrázek 1.15: Hráči – diagram přechodů


Obrázek 1.16: Soutěže


Obrázek 1.17: Detail soutěže

vyvoláme dialogové okno, kde máme možnost přejít na editaci údajů o soutěži, exportovat soutěž či odstranit soutěž. Tlačítko *Nová* v pravém dolním rohu umožňuje přidat novou soutěž – stisknutí vyvolá formulář pro vyplnění údajů o soutěži – a *Importovat* umožňuje importovat soutěž ze souboru – stisknutí vyvolá dialog s nabídkou souborů k importu, výběrem se dostaneme na shrnutí importu. Tato obrazovka pokrývá případy užití UC3.1 Zobrazit seznam soutěží, UC3.2 Vytvořit soutěž, UC3.4 Upravit údaje o soutěži, UC3.5 Odstranit soutěž a UC8.1 Exportovat soutěž. Obrazovky s formuláři jsou opět vynechány.

Na obrázku 1.17 je detail soutěže. Tlačítkem *Zpět* v levém horním rohu se vrátíme na seznam soutěží, ze kterého jsme se na tuto obrazovku dostali. Pomocí tlačítka *Upravit* v pravém horním rohu můžeme editovat údaje o soutěži. Obě tato tlačítka zůstávají stejná pro všechny záložky této obrazovky. Pomocí jednotlivých záložek (na obrázku *Turnaje* a *Hráči*) se dostaneme k dalším obrazovkám, které jsou popsány níže. Tato obrazovka, jak je nejspíš zřejmé,


Obrázek 1.18: Turnaje




Obrázek 1.19: Shrnutí importu

vychází z případu užití UC3.3 Zobrazit detail soutěže.

Na obrázku 1.19 je shrnutí importu. Tlačítkem *Zpět* v levém horním rohu se vrátíme na seznam soutěží, ze kterého jsme se na tuto obrazovku dostali. Na této obrazovce můžeme vidět, co všechno bude v rámci této soutěže importováno. Je zde seznam turnajů, kde je u každého uvedena informace o počtu hráčů, týmů a soutěží. Dále seznam nových hráčů, kteří budou do aplikace přidáni. Poslední položkou seznamu jsou změnění hráči. Tito hráči mají v souboru odlišná data vůči datům v aplikaci. Zde máme u každého hráče možnost rozhodnout, zda u něho chceme ponechat původní data či chceme aplikovat změny ze souboru. Tlačítkem *Importovat* potvrdíme import a soutěž je následně importována. Tato obrazovka odpovídá případu užití UC8.2 Importovat soutěž.

Na obrázku 1.18 je seznam turnajů v soutěži. Pomocí tlačítka *Seřadit* v pravém horním rohu můžeme seřadit turnaje – stisknutí vyvolá dialogové okno, kde si vybereme, zda chceme seřadit turnaje dle názvu, data začátku

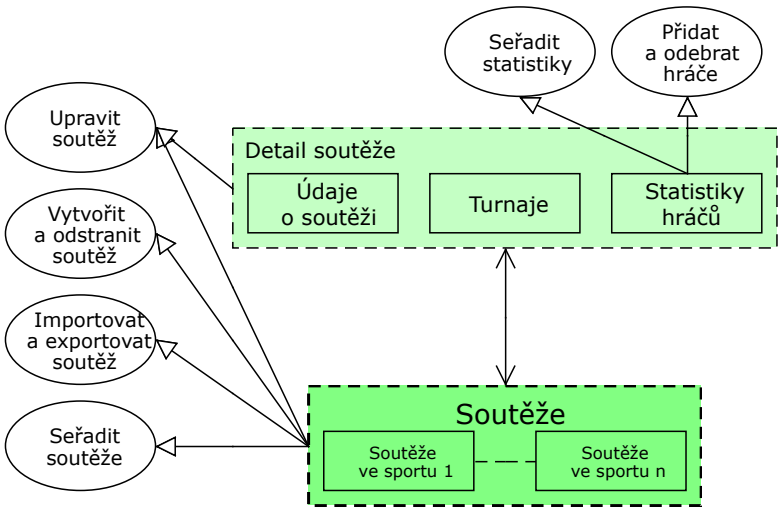

Obrázek 1.20: Hráči v soutěži




Obrázek 1.21: Přidat hráče

či data konce. Opakovaným výběrem změníme řazení ze sestupného na vzestupné a naopak. Stisknutím konkrétního turnaje přejdeme na detail turnaje. Dlouhým stiskem nad konkrétním turnajem vyvoláme dialogové okno, kde máme možnost přejít na editaci údajů o turnaji či odstranit turnaj. Tlačítko *Nový* v pravém dolním rohu umožňuje přidat nový turnaj – stisknutí vyvolá formulář pro vyplnění údajů o turnaji. Tato obrazovka pokrývá případy užití U4.1 Zobrazit seznam turnajů, UC4.2 Vytvořit turnaj, UC4.4 Upravit údaje o turnaji a U4.5 Odstranit turnaj. Obrazovky s formuláři jsou opět vynechány.

Na obrázku 1.20 jsou statistiky hráčů v soutěži. Statistiky můžeme seřadit stisknutím na vybraný sloupec. Opakovaným kliknutím na již vybraný sloupec změníme řazení ze sestupného na vzestupné a naopak. Stisknutím konkrétního hráče přejdeme na detail hráče. Dlouhým stiskem nad konkrétním hráčem vyvoláme dialogové okno, kde máme možnost odstranit hráče ze soutěže. Tlačítko *Přidat hráče* v pravém dolním rohu umožňuje přidat nové hráče do soutěže. Touto obrazovkou pokryjeme případy užití UC7.2 Zobrazit




Obrázek 1.22: Soutěže – diagram přechodů

hráčské statistiky v soutěži a částečně i UC7.4 Seřadit statistiky v soutěži.

Na obrázku 1.21 je přidání hráče do soutěže. Tlačítkem *Zpět* v levém horním rohu se vrátíme na statistiky hráčů v soutěži. Uživatel zaškrtnutím vybere požadované hráče a stiskne tlačítko *Přidat* v pravém dolním rohu, čímž se vrátí na předchozí obrazovku. Tato obrazovka tedy pokrývá případy užití UC3.6 Přidat hráče do soutěže a UC3.7 Odebrat hráče ze soutěže.

Pro přehlednost doplníme na obrázku 1.22 diagram přechodů mezi obrazovkami v této kategorii.

### 1.6.4 Turnaje

Na obrázku 1.23 je detail turnaje. Tlačítkem *Zpět* v levém horním rohu se vrátíme na detail soutěže, ze které jsme se na tuto obrazovku dostali. V pravém horním rohu se nacházejí tři tlačítka. Tlačítko *Upravit* umožňuje editovat údaje o turnaji. Další tlačítko *Konfigurovat bodový zisk* umožňuje, jak je z názvu patrné, nastavit např. body za vítězství, remízu či prohru, atd. Stisknutí posledního tlačítka *Generovat soupisky* vyvolá dialogové okno, kde si můžeme vybrat, zda chceme generovat soupisky náhodné či vyrovnané dle vybrané statistiky. Všechna čtyři tlačítka zůstávají stejná pro všechny záložky této obrazovky. Pomocí jednotlivých záložek (na obrázku *Pořadí* a *Hráči*) se dostaneme k dalším obrazovkám, které jsou popsány níže. Touto obrazovkou částečně pokryjeme případ užití UC4.3 Zobrazit detail turnaje. Máme odsud také možnost se dostat k UC5.8 Generovat soupisky týmů.

Na obrázku 1.24 je pořadí účastníků v turnaji. Kliknutím na vybraný sloupec můžeme seřadit účastníky dle vybraného sloupce. Opakovaným kliknutím

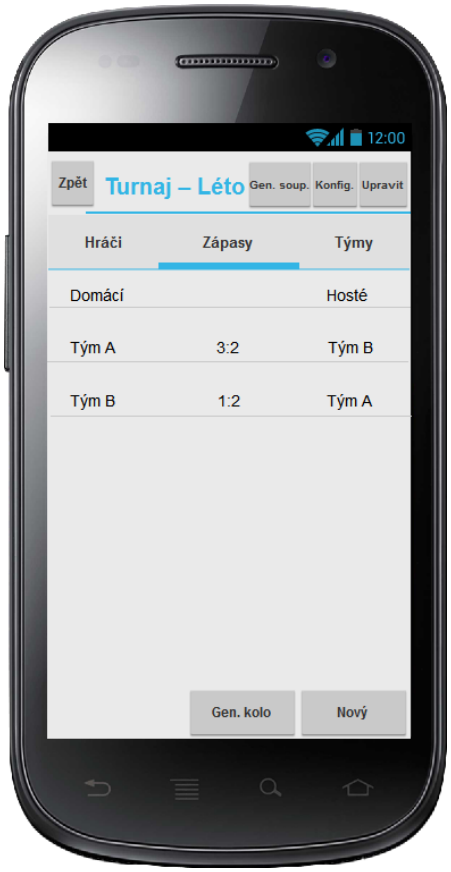

Obrázek 1.23: Detail turnaje


Obrázek 1.24: Pořadí

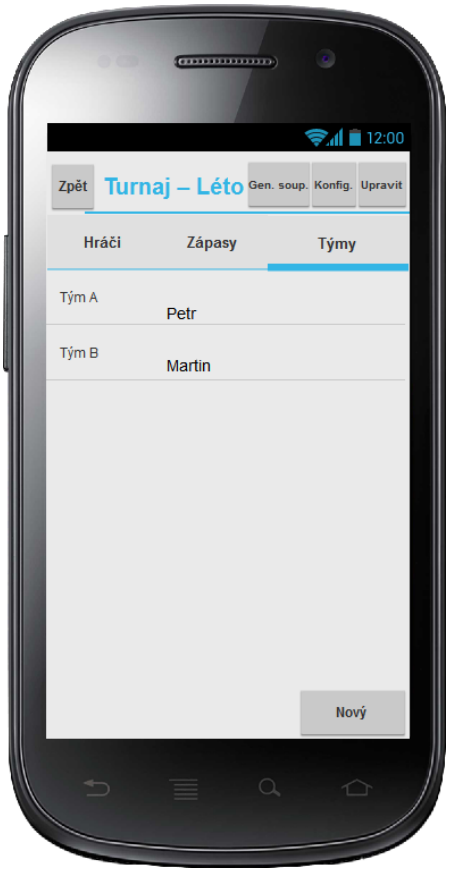
na již vybraný sloupec změníme řazení ze sestupného na vzestupné a naopak. Touto obrazovkou pokryjeme další část případu užití UC4.3 Zobrazit detail turnaje.

Na obrázku 1.25 je seznam zápasů v turnaji. Dlouhým stiskem nad konkrétním zápasem vyvoláme dialogové okno, kde máme možnost přejít na editaci údajů o zápase, vynulovat zápas či odstranit zápas. Stisknutím konkrétního zápasu přejdeme na detail zápasu. Tlačítko *Nový* v pravém dolním rohu umožňuje přidat nový zápas – stisknutí vyvolá formulář pro vyplnění údajů o zápase – a *Generovat kolo* umožňuje vygenerovat nové kolo zápasů. Tato obrazovka odpovídá případu užití UC6.1 Zobrazit seznam zápasů, dále z ní dostaneme k UC6.2 Vytvořit zápas, UC6.3 Vygenerovat kolo zápasů, UC6.5 Upravit údaje o zápase, UC6.7 Vynulovat zápas a UC6.8 Odstranit zápas. Formuláře pro vytvoření a editaci jsou opět vynechány.

Na obrázku 1.26 je seznam týmů v turnaji včetně soupisek jednotlivých týmů (pokud se jedná o turnaj týmů, jinak se tato obrazovka v detailu turnaje




Obrázek 1.25: Zápasy

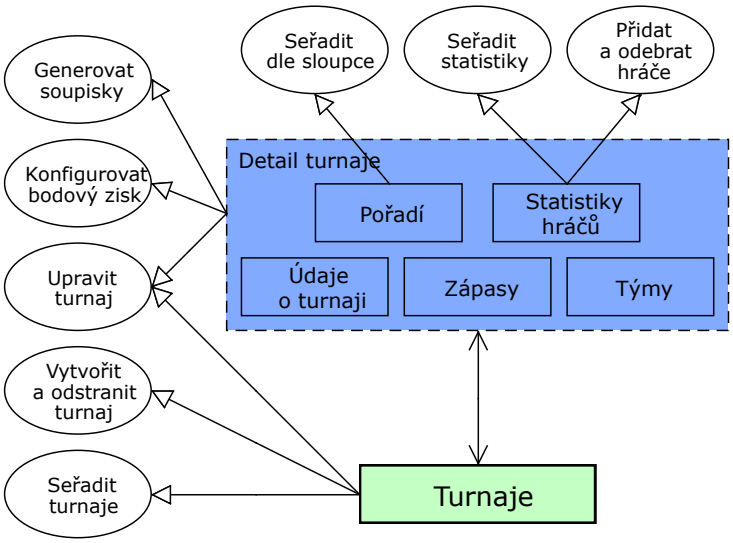



Obrázek 1.26: Týmy

nenachází). Stisknutím konkrétního týmu přejdeme na detail týmu. Dlouhým stiskem nad konkrétním týmem vyvoláme dialogové okno, kde máme možnost přejít na editaci názvu týmu či odstranit tým. Tlačítko *Nový* v pravém dolním rohu umožňuje přidat nový tým – stisknutí vyvolá formulář pro zadání názvu týmu. Touto obrazovkou pokryjeme případ užití UC5.1 Zobrazit seznam týmů a UC5.4 Odstranit tým. Obrazovky odpovídající případům užití UC5.2 Vytvořit tým a UC5.3 Upravit údaje o týmu jsou vynechány.

Obrazovky se statistikami hráčů v turnaji a přidáním hráčů do turnaje jsou vynechány, neboť jsou totožné s obrazovkami pro soutěž. Tyto obrazovky pokrývají případy užití UC4.6 Přidat hráče do turnaje, UC4.7 Odebrat hráče z turnaje, UC7.3 Zobrazit hráčské statistiky v turnaji a UC7.5 Seřadit statistiky v turnaji.

Pro přehlednost doplníme na obrázku 1.22 diagram přechodů mezi obrazovkami v této kategorii.




Obrázek 1.27: Turnaje – diagram přechodů

### 1.6.5 Týmy

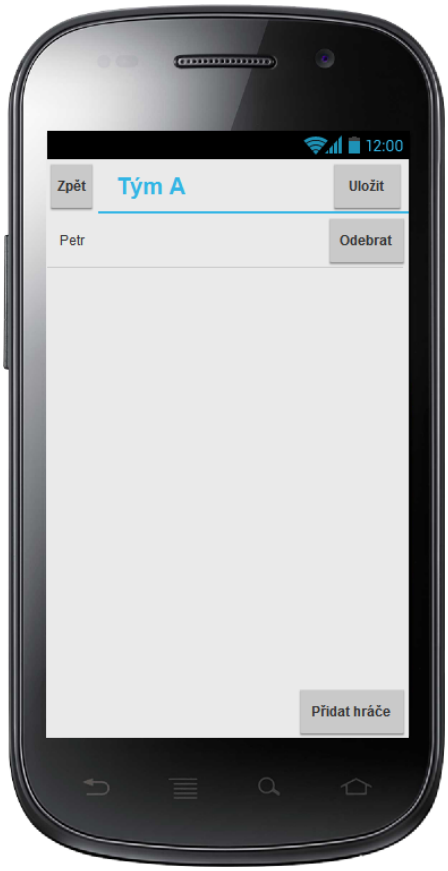
Na obrázku 1.28 je detail týmu. Tlačítkem *Zpět* v levém horním rohu se vrátíme na detail turnaje, ze kterého jsme se na tuto obrazovku dostali. Na této obrazovce se nachází soupiska týmu – tedy seznam hráčů, které obsahuje daný tým. Hráče můžeme ze soupisky odstranit kliknutím na tlačítko odstranit u konkrétního hráče. Stisknutím tlačítka *Uložit* v pravém horním rohu soupisku týmu uložíme a vrátíme se na detail turnaje, ve kterém se tento tým nachází. Pomocí tlačítka *Přidat hráče* v pravém dolním rohu můžeme přidat hráče do týmu. Tato obrazovka tedy odpovídá případu užití UC5.5 Zobrazit detail týmu.

Obrazovka pro přidání hráčů do týmu je vynechána, neboť je totožná s obrazovkou pro přidání hráčů do soutěže. Tato obrazovka pokrývá případy užití UC5.6 Přidat hráče do týmu a UC5.7 Odebrat hráče z týmu.

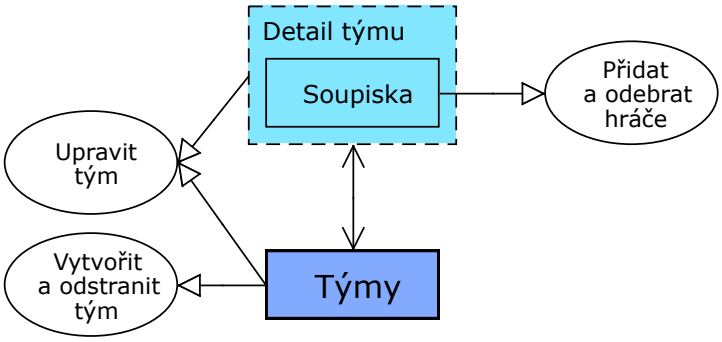
Pro přehlednost doplníme na obrázku 1.29 diagram přechodů mezi obrazovkami v této kategorii.

### 1.6.6 Zápasy

Na obrázku 1.30 je detail zápasu. Tlačítkem *Zpět* se vrátíme na seznam zápasů v turnaji, odkud jsme se na tuto obrazovku dostali. Stisknutím tlačítka *Uložit* uložíme veškeré změny provedené na této obrazovce a vrátíme se na seznam zápasů v turnaji, do kterého tento zápas patří. Obě předchozí tlačítka zůstávajíc stejná pro obě záložky na této obrazovce. Pomocí tlačítka *Upravit*

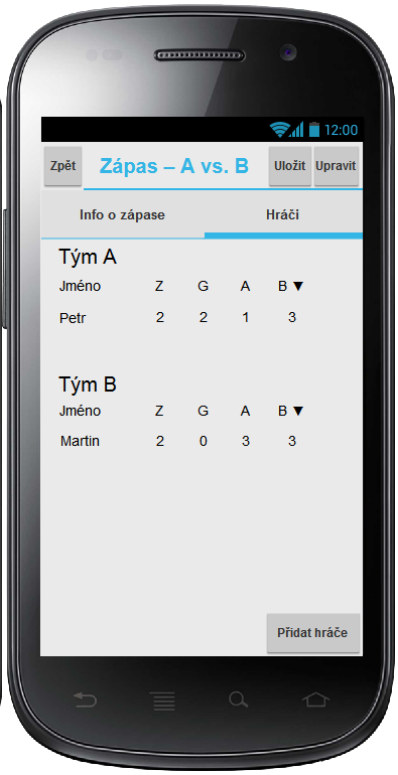



Obrázek 1.28: Detail týmu

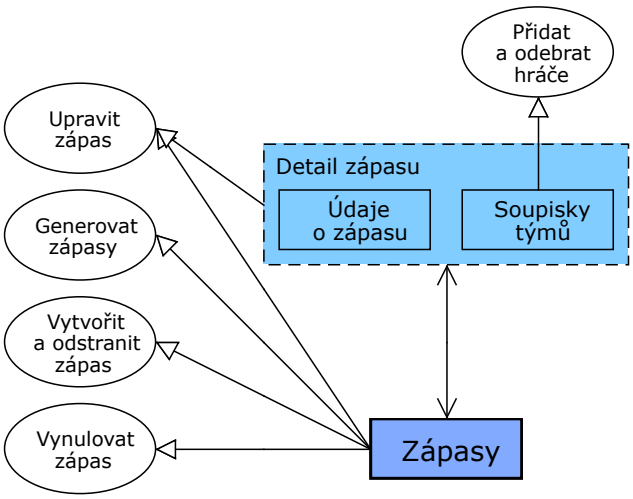



Obrázek 1.29: Týmy – diagram přechodů


Obrázek 1.30: Detail zápasu




Obrázek 1.31: Statistiky zápasu




Obrázek 1.32: Zápasy – diagram přechodů

v pravém horním rohu můžeme editovat údaje o zápasu (kolo, periodu, datum a poznámku). Tato obrazovka se může velmi lišit v závislosti na daném sportu. Tento obrázek může posloužit jako inspirace pro fotbal – můžeme zde velmi snadno upravovat výsledek pomocí tlačítek nad skóre a zaškrtávacích polí pro prodloužení a penalty. Pomocí záložky *Hráči* se dostaneme na další obrazovku. Touto obrazovkou tedy pokrýváme případ užití UC6.6 Zadat výsledek zápasu a částečně také UC6.4 Zobrazit detail zápasu.

Na obrázku 1.31 jsou soupisky jednotlivých týmů a statistiky jednotlivých hráčů v zápase. Tato obrazovka se opět může lišit v závislosti na daném sportu. Po stisknutí tlačítka *Upravit* v pravém horním rohu se zobrazí formulář, kde můžeme editovat statistiky všech hráčů v zápase najednou. Dlouhým stiskem nad konkrétním hráčem vyvoláme dialogové okno, kde máme možnost upravit statistiky daného hráče v zápase či odstranit hráče ze zápasu. Stisknutím tlačítka *Přidat hráče* v pravém dolním rohu vyvoláme dialogové okno, kde si vybereme, ke kterému týmu chceme hráče přidat, a následně můžeme přidat hráče k tomuto týmu. Obrazovka pro přidání hráče do týmu je totožná s obrazovkou pro přidání hráčů do soutěže. Touto obrazovkou pokryjeme zbylou část případu užití UC6.4 Zobrazit detail zápasu, resp. tu část se soupiskami týmů a hráčskými statistikami.

Obrazovka pro přidání hráčů do zápasu je vynechána, neboť je totožná s obrazovkou pro přidání hráčů do soutěže. Tato obrazovka pokrývá případy užití UC6.9 Přidat hráče do zápasu a UC6.10 Odebrat hráče ze zápasu.

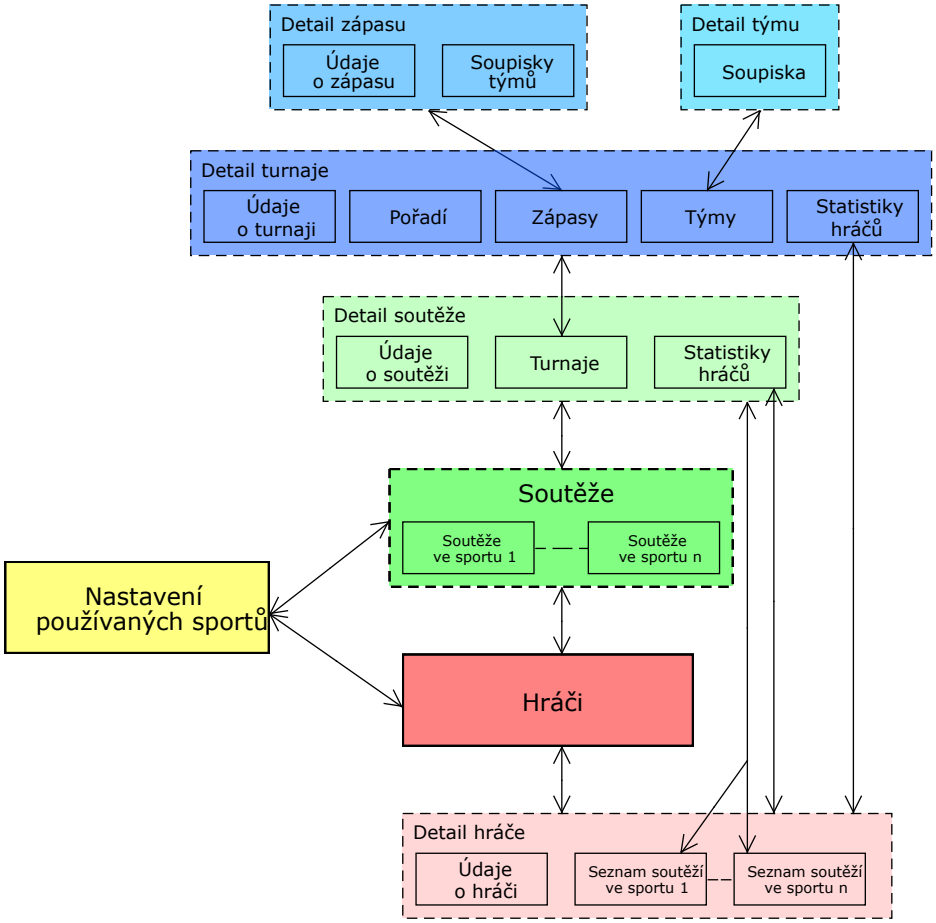
Pro přehlednost doplníme na obrázku 1.32 diagram přechodů mezi obrazovkami v této kategorii.

### 1.6.7 Diagram přechodů mezi obrazovkami

Nyní jsme si popsali obrazovky aplikace dle jednotlivých kategorií. Protože jsou tyto kategorie mezi sebou provázané, ukážeme si ještě diagram přechodů mezi jednotlivými kategoriemi. Na tomto obrázku 1.33 jsou kvůli zachování jisté přehlednosti vynechány veškeré akce dostupné v aplikaci.

## 1.7 Doménový model

V poslední sekci analýzy si představíme doménový model. Ten by měl doplnit představu o datech v aplikaci a vazbami mezi nimi. Jak víme z požadavků, aplikace by měla být rozšiřitelná pomocí modulů, které komunikují s jádrem aplikace. Třídy označené žlutě spravuje jádro a zeleně označujeme třídy, které se z definice budou lišit napříč moduly a nemají tedy žádnou výchozí definici. Ostatní třídy spravuje také modul, ale máme u nich danou výchozí strukturu tříd. Tato struktura sice není nijak striktně vyžadovaná (balíčky budou schopné fungovat i bez jejího dodržení), nicméně její použití usnadní používání aplikace – moduly by se měly lišit jen minimálně, a to ve věcech, které

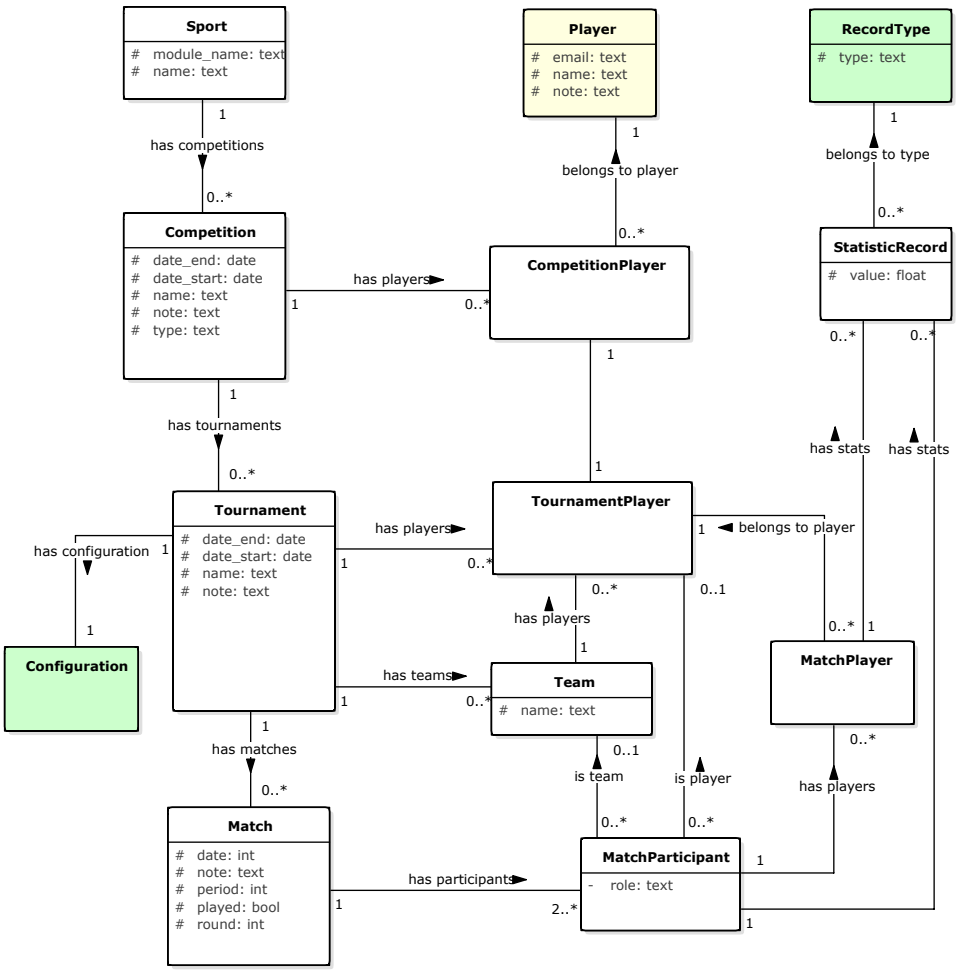



Obrázek 1.33: Aplikace – přechody mezi obrazovkami

jsou mezi sporty odlišné. V ostatních věcech by měly být moduly maximálně podobné.

**Sport** Tato entita představuje jeden z množiny sportů, které definuje modul. U sportu nás zajímá samotný název a dále název modulu, který ho implementuje. Sport může obsahovat libovolné množství soutěží.

**Player** Tato entita představuje hráče. Hráč má kromě jména e-mailovou adresu a poznámku. E-mailová adresa musí být unikátní, tzn. žádní dva hráči nesmějí mít stejnou e-mailovou adresu.




Obrázek 1.34: Doménový model aplikace

**Competition** Touto entitou je míněna soutěž. Soutěž má, jak už víme, datum začátku a konce, název, poznámku a typ – soutěž je buď týmová nebo individuální. Soutěže mohou obsahovat libovolné množství turnajů a hráčů.

**CompetitionPlayer** Tato entita představuje hráče přidané do soutěže, musí se tedy vztahovat k jedné konkrétní soutěži a jednomu konkrétnímu hráči.

**Tournament** Touto entitou se myslí turnaj. Turnaj má, stejně jako soutěž, datum začátku a konce, název a poznámku. Typ turnaje je daný typem soutěže, do které turnaj patří. K turnaji se váže jeho bodová konfigurace (ke každému turnaji právě jedna), libovolné množství zápasů, týmů a hráčů.

**TournamentPlayer** Tato entita představuje hráče přidané do turnaje, ti však již musí být i v příslušné soutěži. Vztahuje se tedy k jednomu konkrétnímu turnaji a jednomu hráči přidanému do soutěže.

**Configuration** Touto entitou je míněna bodová konfigurace turnaje. Zelená barva značí, že obsah této entity je čistě v režii modulu – každý sport se v tomto liší, např. v hokeji má smysl konfigurovat počet získaných bodů za vítězství po samostatných nájezdech na rozdíl od tenisu, kde toto smysl rozhodně nemá.

**Team** Tato entita představuje tým. Tým má pouhý název a je přiřazen k turnaji. Jak můžeme vidět z obrázku, týmy existují jen v rámci turnaje a nijak se nepřenáší mezi soutěžemi. Tým může obsahovat libovolný počet hráčů, kteří se účastní příslušného turnaje a tvoří jeho soupisku. Tým se může účastnit libovolného množství zápasů.

**Match** Touto entitou se myslí zápas. U něj evidujeme datum odehrání, poznámku, stav (zda již byl odehraný či nikoli), periodu a kolo. Zápas má libovolné množství účastníků, ovšem minimální počet jsou dva. Díky tomu, že horní hranice na počet účastníků není stanovena, nám do této struktury zapadají nejen *zápasové sporty*, ale také *závodní*.

**MatchParticipant** Tato entita představuje účastníka zápasu. U něj evidujeme roli, která v zápasovém sportu může sloužit např. k rozlišení domácího a hostujícího účastníka. Každý účastník se váže buď na právě jeden tým, nebo na právě jednoho hráče. V případě, že je účastníkem tým, udržuje si také soupisku týmu v daném zápase. Tato soupiska, jak je zřejmé z doménového modelu, nemusí odpovídat soupisce týmu jako takového – můžeme tedy evidovat, že některý hráč do zápasu nenastoupil apod. Účastník zápasu může obsahovat libovolný počet statistik, pomocí nich můžeme evidovat např. skóre zápasu v hokeji apod – držet skóre přímo u entity zápasu by sice bylo snazší, nicméně tímto způsobem jsme schopni zahrnout větší množství sportů pod jeden doménový model.

**MatchPlayer** Tato entita představuje hráče přidané do zápasu, ti však již musí být i v příslušném turnaji. Vztahuje se tedy k jednomu konkrétnímu zápasu a jednomu hráči přidanému do turnaje. K tomuto hráči, přidanému do zápasu, se váže libovolné množství statistik.

**StatisticRecord** Touto entitou je míněn zápis do statistik daného hráče v zápase. U této entity evidujeme pouze hodnotu. Typ statistiky je daný vazbou na *RecordType*.

**RecordType** Tato entita představuje typ statistiky. Každý sport má těchto typů několik a typy se samozřejmě v závislosti na sportu liší. U hokeje můžeme takto evidovat např. počet vstřelených gólů, počet zákroků brankaře či počet trestných minut. Množina evidovaných typů je tedy daná modulem, jak nám napovídá zelená barva. Na tuto entitu se váže libovolné množství zápisů do statistik, ať už individuálních či týmových.

Kapitola 2

# Návrh a popis architektury

V této kapitole se budeme věnovat návrhu a popisu architektury systému. Dříve než se však pustíme do samotné architektury, vysvětlíme si některé pojmy, které musíme pro pochopení návrhu znát. Poté si popíšeme samotnou architekturu. Na tu se dá pohlížet více způsoby – nejprve si představíme systémovou architekturu, která dělí systém na jádro, knihovnu a moduly, poté si představíme aplikační architekturu, která dělí jednotlivé moduly (např. jádro) do tří vrstev. Nakonec se budeme věnovat právě jednotlivým modulům, a sice jádru a knihovně.

## 2.1 Android

V této sekci si vysvětlíme některé pojmy specifické pro programování v Androidu. Tyto pojmy se používají především v prezentační vrstvě, ostatní vrstvy jsou napsané de facto ve standardní Javě.

### 2.1.1 Manifest

Každá aplikace v Androidu musí v kořenovém adresáři obsahovat soubor *AndroidManifest.xml*. V tomto souboru se nachází informace o aplikaci důležité pro OS Android. V tomto souboru můžeme např. nastavit název aplikace, ikonu aplikace či minimální verzi Android API, kterou aplikace vyžaduje.

Dále obsahuje seznam všech *Aktivit*, *Služeb* či *Poskytovatelů obsahu* v aplikaci. Můžeme také specifikovat, která aktivita se bude spouštět při zapnutí aplikace. Manifest může obsahovat libovolné množství dalších metadat.[6]

### 2.1.2 Activity

*Activity*, v češtině aktivita, znamená v Androidu obrazovku. Všechny aktivity musí dědit od `com.android.Activity`. Každá aktivita má přesně daný životní cyklus – od vytvoření přes start, samotný běh, zastavení až po ukončení. Pro

každý stav životního cyklu aktivity existuje metoda, kterou můžeme přetížit a ovlivnit tak chování aktivity v daném stavu.[7]

### 2.1.3 Fragment

Fragmenty pomáhají dělit aktivity na menší znovupoužitelné části. Fragmenty dědí od třídy `com.android.Fragment` a vždy patří do nějaké aktivity (či fragmentu). Stejně jako aktivita i fragment má daný životní cyklus.[8]

### 2.1.4 Adapter

Adaptér slouží k prezentaci seznamu dat stejného typu. Pokud chceme zobrazit např. seznam soutěží, usnadníme si práci, pokud vytvoříme adaptér schopný zobrazit jednu soutěž. Při vytváření uživatelského rozhraní poté vytvoříme tento adaptér, předložíme mu seznam soutěží a on se postará o jejich vykreslení.

### 2.1.5 Context

Kontext je v Androidu reference na aktuální aplikaci. Objekt této třídy potřebujeme např. při přístupu do databáze.

### 2.1.6 Intent

Intent, česky záměr či úmysl, je abstraktní popis operace, která má být provedena. Intent nám umožní spustit aktivitu či službu. K tomuto požadavku máme vždy možnost přidat data, která bude aktivita či služba vyžadovat.

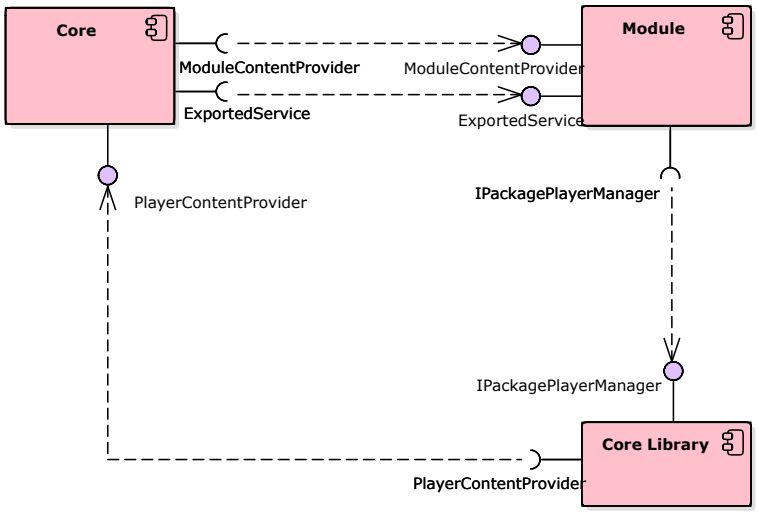
### 2.1.7 Service

Service, v češtině služba, se používá pro výpočty, které běží v samostatném vlákně – OS Android totiž přísně zakazuje, aby bylo vytěžováno hlavní vlákno aplikace. Pokud bychom toto vlákno vytěžovali příliš, aplikace nabídne uživateli ukončení aplikace, což rozhodně není žádoucí.

Služba se spouští asynchronně, běží na pozadí a když svou práci skončí, může např. odeslat výsledek fragmentu, který jej očekává. Služby mohou být exportované, což umožňuje ostatním aplikacím tyto služby používat. Právě exportovanou službu využijeme mimo jiné při komunikaci mezi jádrem a moduly.

### 2.1.8 Content Provider

Content Provider, česky poskytoval obsahu, umožňuje poskytovat přístup k datům podobně jako např. SQLite databáze. Výhodou je, že tento přístup je na rozdíl od služby synchronní. Tímto poskytovatelem pokryjeme druhou část




Obrázek 2.1: Systémová architektura aplikace

naší komunikace mezi jádrem a moduly. Content provider může být exportovaný, význam je stejný jako u služeb.

## 2.2 Systémová architektura

Jak jsme si již nastínili v úvodu kapitoly a jak můžeme vidět na obr. 2.1, systémová architektura rozděluje systém na jádro, knihovnu a jednotlivé moduly implementující různé sady sportů. Tato architektura by nám měla umožnit snadné rozšiřování aplikace o další a další sporty, mimo jiné je to jeden z požadavků zadavatele práce. V této sekci se budeme věnovat nejprve jádru aplikace, poté knihovně a nakonec samotnému modulu.

### 2.2.1 Jádro

Jádro aplikace obsluhuje pouze tři základní obrazovky dostupné z hlavního menu zmíněné v předchozí kapitole. Veškeré informace o hráčích a používaných sportech jsou uložené v jádru aplikace. Toto však neplatí např. o seznamu soutěží v daném sportu či agregovaných statistikách pro daný sport v detailu hráče. Tyto informace v jádru uložené nejsou a jádro je tedy musí získávat z jednotlivých modulů.

### 2.2.2 Knihovna

Knihovna aplikace je sada sdílených funkcí, tříd a rozhraní, pomocí kterých mezi sebou jádro a moduly komunikují. V knihovně najdeme např. třídy pro

téměř všechny entity z doménového modelu.

### 2.2.3 Modul

Modul aplikace, jak již bylo zmíněno, implementuje určitou sadu sportů. Modul musí implementovat některá rozhraní, aby mohla fungovat komunikace mezi ním a jádrem. Modul totiž není samostatně spustitelná aplikace a spouští se pouze z jádra – v případě, že nebude rozhraní implementované správně, nebude si jádro tohoto modulu všímat, případně nebude komunikace mezi modulem a jádrem fungovat stoprocentně.

## 2.3 Aplikační architektura

V této sekci se budeme věnovat architektuře jednotlivých modulů. Vzhledem ke komplexnosti aplikace nemá smysl uvažovat o jiné než vícevrstvé architektuře. Tato architektura je přehlednější, umožňuje snazší údržbu a např. výměna typu datového úložiště pro nás bude znamenat výměnu jedné vrstvy a nikoli přepsání celé aplikace. Z tohoto důvodu jsme se tedy rozhodli pro třívrstvou architekturu. Architekturu jednotlivých modulů samozřejmě nemůžeme ovlivnit, nicméně použití stejné architektury, jako použijeme my, dá vývojářům modulů množství výhod.

Nejvyšší vrstvou této architektury je vrstva *prezentační*, která prezentuje data uživateli a zajišťuje komunikaci uživatele s aplikací. Další vrstvou je vrstva *logická* nebo také business vrstva, která obsahuje samotnou logiku aplikace. Nejnižší vrstvou aplikace je vrstva *datová*, která zajišťuje ukládání a načítání dat z datového úložiště, kterým je v našem případě relační databáze. Všechny tyto vrstvy si nyní podrobněji popíšeme.

### 2.3.1 Prezentační vrstva

Mezi účely prezentační vrstvy patří samotná prezentace zpracovaných dat uživateli a komunikace mezi uživatelem a uživatelským rozhraním. O to se v Androidu starají především *aktivity*, *fragmenty*, ale i další třídy. Najdeme zde tak např. aktivity odpovídající jednotlivým obrazovkám, které obsahují *listenery*, česky posluchače, čekající na uživatelovu akci. Jakmile *listener* akci zachytí, může např. spustit jinou aktivitu či vykonat jakýkoli jiný kód.

### 2.3.2 Logická vrstva

Logická vrstva obstarává logiku aplikace. Najdeme zde tedy např. algoritmy generující vyrovnané soupisky či kolo zápasů. Také zpracování dat získaných z datové vrstvy, např. agregování statistik, se děje na této úrovni. Zkrátka téměř veškerá manipulace s daty (kromě samotného ukládání a načítání) se děje v rámci logické vrstvy.

### 2.3.3 Datová vrstva

Datová vrstva slouží k ukládání a načítání dat z datového úložiště, v našem případě z relační databáze SQLite. V této vrstvě se nachází třídy pro entity představující např. soutěž, turnaj nebo zápas. Pro mapování mezi relační databází a těmito entitami využijeme framework *ORMLite*. V jednotlivých třídách se poté pomocí anotací definuje např. název tabulky či sloupce, které je třeba ukládat do databáze.

### 2.3.4 Komunikace mezi vrstvami

Kromě komunikace v rámci vrstev probíhá také komunikace napříč vrstvami. Ta probíhá vždy směrem dolů, tzn. prezentační vrstva komunikuje přes definované rozhraní s logickou vrstvou a logická vrstva komunikuje přes definované rozhraní s datovou vrstvou. Vrstvy tedy mohou být snadno vyměněny za předpokladu, že implementujeme rozhraní konkrétní vrstvy.

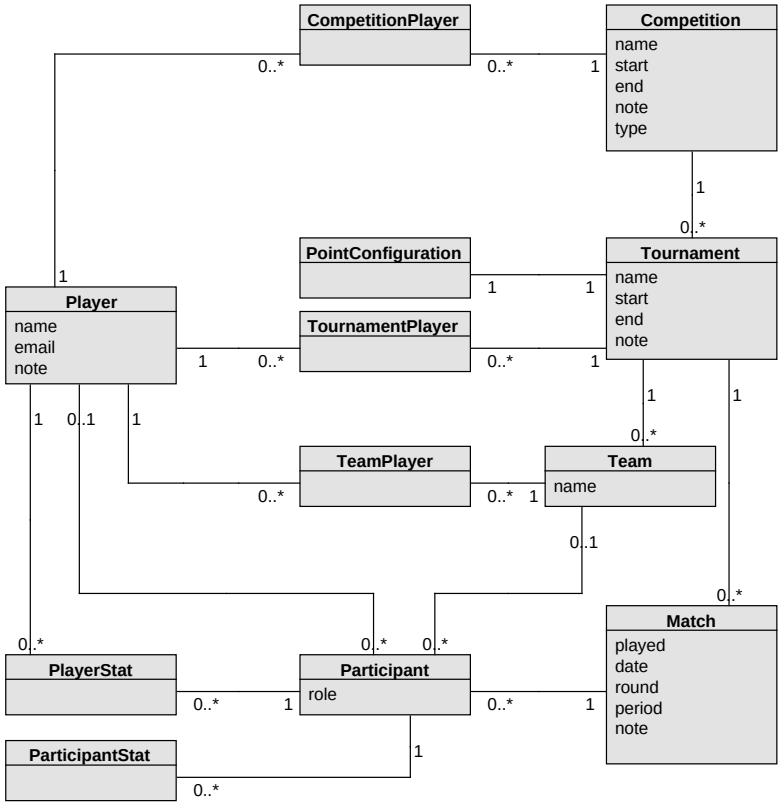
## 2.4 Databázový model

Databázový model na obr. 2.2 samozřejmě vychází z doménového modelu, ovšem najdeme zde některé odlišnosti. První rozdíl najdeme v chybějící tabulce pro sport. Předpokládáme, že modul sám oddělí soutěže z různých sportů od sebe (pokud implementuje více sportů) jiným způsobem, např. použitím odlišné databáze pro různé sporty. Další rozdíl najdeme u hráčských a týmových statistik. Zde kvůli jednoduchosti předpokládáme, že všechny hráčské statistiky budou jako sloupce tabulky *PlayerStat* a všechny týmové jako sloupce tabulky *ParticipantStat*. Tomuto databázovému modelu odpovídají entity v knihovně, ovšem modul nemá žádnou povinnost se tímto modelem řídit.

Nyní si uvedeme, jak tento model využívají moduly, které již implementované jsou. Na obrázku 2.3 vidíme pouze spodní část doménového modelu. Horní část je vynechána, neboť je totožná s předchozím obrázkem. Vazby na vynechané entity jsou znázorněny pomocí sloupců v tabulce – např. *tournamentId* u entity *Match* či *playerId* u entity *PlayerStat*. Pod názvem každé entity najdeme název modulu, odkud pochází – *Hockey* odpovídá modulu implementujícímu sporty na bázi hokeje, *Squash* odpovídá modulu implementujícímu sporty na bázi squashe a *Library* odpovídá entitám z knihovny vycházejícím z předchozího databázového modelu.

### 2.4.1 Hokej

Modul implementující sporty na bázi hokeje můžeme vidět na levé straně obr. 2.3. Tento modul rozšiřuje entitu *Match* a přidává k ní informaci o tom, zda byl zápas rozhodnut v prodloužení či samostatných nájezdech. Dále rozšiřuje

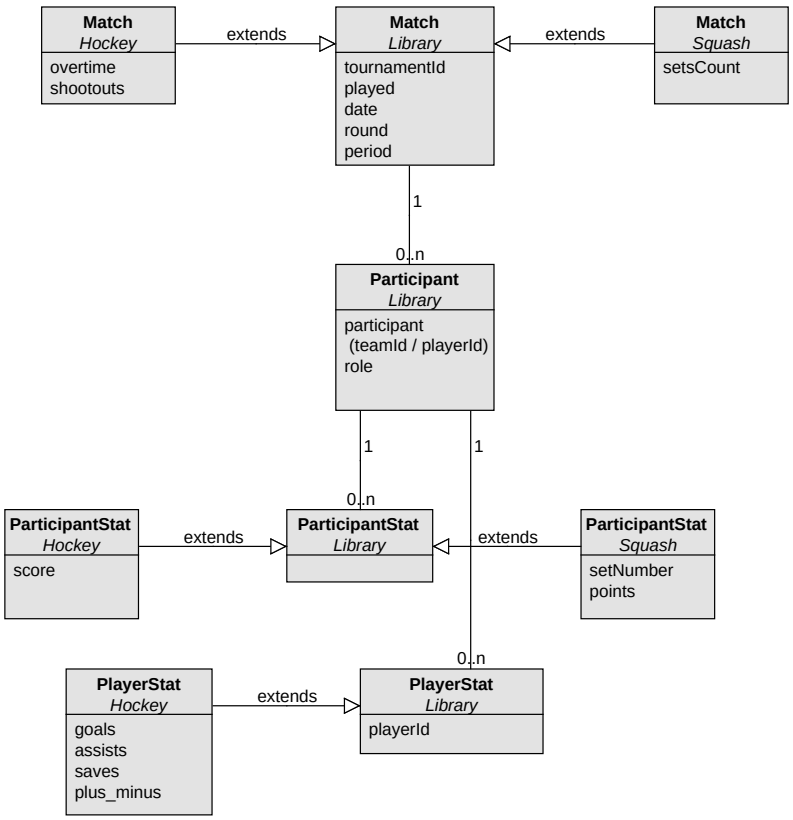



Obrázek 2.2: Databázový model

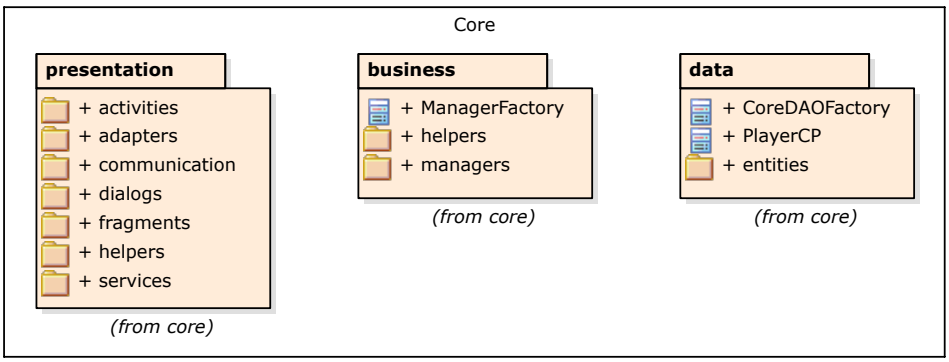
entitu *ParticipantStat*, kde je uvedeno skóre daného účastníka v zápase, tedy počet vstřelených branek. Poslední entitou, kterou tento modul rozšiřuje, je *PlayerStat*. Zde můžeme vidět výčet evidovaných hráčských statistik – počet vstřelených gólů, asistencí, zákroků a bodování plus mínus.

### 2.4.2 Squash

Modul implementující sporty na bázi squashe můžeme vidět na pravé straně obr. 2.3. Tento modul přidává k entitě *Match* informaci o počtu odehraných setů v zápase a k entitě *ParticipantStat* počet získaných bodů v konkrétním setu. Entita *PlayerStat* není rozšířena nijak, tento modul zatím nesleduje hráčské statistiky, ale pouze umožňuje udržovat informace o tom, kteří hráči se zápasu účastnili.




Obrázek 2.3: Databázový model – rozšíření v modulech




Obrázek 2.4: Jádro – diagram tříd

## 2.5 Jádro

V této sekci se budeme detailněji věnovat popisu jádra aplikace. Jádro je rozděleno na tři vrstvy tak, jak bylo popsáno v předchozí sekci. Toto rozdělení můžeme vidět na obrázku 2.4. Nyní se budeme opět věnovat jednotlivým vrstvám, ovšem tentokrát si je popíšeme konkrétněji v kontextu jádra aplikace. Nakonec se budeme věnovat komunikaci mezi těmito vrstvami.

### 2.5.1 Prezentační vrstva

Prezentační vrstva jádra obsahuje několik balíčků, které můžeme vidět na obrázku 2.4. Popíšeme si tedy jednotlivé balíčky.

#### 2.5.1.1 Activities

Tento balíček obsahuje čtyři aktivity nezbytné pro fungování jádra. Těmi jsou `MainActivity` a `PlayerDetailActivity`, které si nyní popíšeme, a dále `CreatePlayerActivity` a `ImportActivity`, jejichž účel je zřejmý – jedna slouží k vytvoření hráče, druhá zobrazuje shrnutí informací o importované soutěži.

**MainActivity** Toto je hlavní aktivita spouštěná při zapnutí aplikace. Tato aktivita obsahuje tři fragmenty, které tvoří jednotlivé obrazovky dostupné z hlavního menu aplikace – `SportsFragment` zobrazuje seznam soutěží pro jednotlivé sporty, `PlayersListFragment` slouží k zobrazení seznamu hráčů a `SettingsFragment` umožňuje nastavit používané sporty v aplikaci.

**PlayerDetailActivity** Tato aktivita, jak je zřejmé z jejího názvu, slouží pro zobrazení detailu hráče. K tomu využívá `PlayerDetailFragment` zobrazující samotné údaje o hráči a `PlayerSportFragment` zobrazující agregované statistiky a seznam soutěží v daném sportu, aktivita samozřejmě vytvoří pro každý používaný sport v aplikace jednu instanci tohoto fragmentu. Tato aktivita je specifická tím, že je *exportovaná*. To znamená, že ji mohou použít moduly např. při přechodu ze seznamu hráčů v soutěži do detailu hráče.

#### 2.5.1.2 Adapters

Tento balíček obsahuje adaptéry nezbytné pro zobrazení seznamu soutěží, hráčů a informací o importované soutěži.

#### 2.5.1.3 Communication

V rámci prezentační vrstvy se často odesílají požadavky či odpovědi s přibalenými daty. V tomto balíčku se nachází jediná třída `ExtraConstants` obsahující

klíče do těchto dat. Umístění klíčů na jedno místo nám zajistí používání stejných klíčů pro zápis i následné čtení.

#### 2.5.1.4 Dialogs

V tomto balíčku se nacházejí třídy starající se o dialogová okna. Tato okna jsou zobrazena např. při dlouhém stisku nad konkrétní soutěží či hráčem a dávají nám možnost si vybrat akci, kterou chceme nad tímto objektem vykonat. Příkladem takové akce může být editace či odstranění.

#### 2.5.1.5 Fragments

Tento balíček obsahuje fragmenty obsluhující jednotlivé obrazovky v jádře, některé příklady jsme si již uvedli v balíčku aktivit, ostatní není třeba vysvětlovat.

#### 2.5.1.6 Helpers

Tento balíček obsahuje třídy, které svým způsobem pomáhají s některou funkcionalitou ostatním třídám.

**FilesHelper** Tato třída slouží především importu a exportu soutěží a obsahuje metody pro načítání a ukládání souborů.

**PackagesInfo** Tato třída je klíčová z hlediska modularity aplikace – umí totiž prohledat seznam nainstalovaných aplikací v OS a najít v nich moduly naší aplikace. Jednotlivé moduly rozpozná pomocí metadat, která musí obsahovat definované klíče a hodnoty.

#### 2.5.1.7 Services

V tomto balíčku se nachází pouze třída `PlayerService`.

**PlayerService** Tato třída slouží pro práci s hráči. Prezentační vrstva ji využívá např. při zobrazení seznamu hráčů, kdy aktivita či fragment pošle požadavek této službě, která ho zpracuje a výsledek odešle zpět jako odpověď.

### 2.5.2 Logická vrstva

V logické vrstvě, jak lze vidět na obrázku 2.4, nalezneme dva balíčky a také jednu třídu, která se nachází přímo v samotném balíčku *business*, tedy kořenovém balíčku logické vrstvy. Této třídě se budeme věnovat nejdříve.

**ManagerFactory** Tato *factory*[3] slouží k získání instancí *managerů* umístěných v balíčku *managers*. Implementuje rozhraní `IManagerFactory`, které je definované v knihovně.

#### 2.5.2.1 Helpers

Tento balíček obsahuje jedinou třídu `JsonFileValidator`, která slouží k rozpoznání validních souborů k importu.

#### 2.5.2.2 Managers

V tomto balíčku se nachází balíček *interfaces* a třídy sloužící pro práci s entitami *Player* a *Setting* – `PlayerManager` a `SettingManager`. Tyto třídy implementují rozhraní definovaná právě v balíčku *interfaces*.

### 2.5.3 Datová vrstva

V této vrstvě nalezneme pouze třídy `CoreDAOFactory`, `PlayerCP` a balíček *entities*. Datová vrstva slouží k načítání a ukládání entit, v jádru aplikace jsou to entity *Player* a *Setting*. Ostatní entity spravují moduly samotné.

**CoreDAOFactory** Pomocí této třídy získáváme *DAO*[4] pro již zmíněné entity *Player* a *Setting*.

**PlayerCP** Content Provider, jak jsme popsali výše, umožňuje poskytovat přístup k datům. Tato třída slouží modulům, které skrze ní získávají informace o hráčích, kteří se v aplikaci nacházejí.

#### 2.5.3.1 Entities

V tomto balíčku se nachází pouze třída pro entitu *Setting*, entita *Player* se nachází v knihovně, protože ji kromě jádra používají i moduly.

### 2.5.4 Komunikace mezi vrstvami

Komunikaci mezi vrstvami můžeme rozdělit na dva odlišné případy – komunikaci mezi vrstvami v rámci jádra a komunikaci mezi vrstvami jádra a modulů. Těmto dvěma případům se budeme nyní věnovat.

[3] Návrhový vzor umožňující vytváření entit voláním metody *factory* místo konstruktoru entity.

[4] *Data Access Object* – objekt pro přístup k datům. Tento objekt umožňuje manipulovat s daty v databázi.

#### 2.5.4.1 Komunikace v rámci jádra

V rámci jádra probíhá komunikace mezi vrstvami směrem dolů, jak již bylo popsáno v jedné z předchozích sekcí. Prezentační vrstva využívá rozhraní `IManagerFactory` pro komunikaci s logickou vrstvou a logická vrstva využívá rozhraní `IDAOFactory` pro práci s vrstvou datovou.

#### 2.5.4.2 Komunikace jádro – modul

Komunikace samozřejmě probíhá i mezi jádrem a jednotlivými moduly. Tuto komunikaci jsme již nastínili na obr. 2.1. V rámci této komunikace se např. získává seznam soutěží, který se zobrazuje v jedné z hlavních obrazovek jádra či naopak seznam hráčů v modulu při přidávání hráčů do soutěže. Tuto komunikaci si ještě rozdělíme podle vrstev, na kterých probíhá.

Nejprve se budeme věnovat komunikaci mezi prezentačními vrstvami jádra a modulů. Moduly mají na prezentační vrstvě exportovanou službu, kterou jádro používá k získání následujících informací:

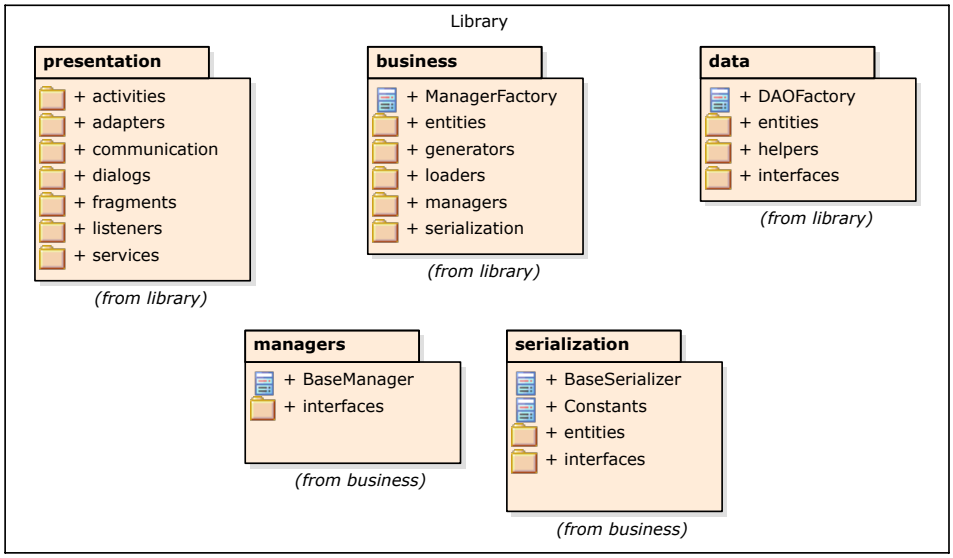
- seznamu všech soutěží,
- seznamu soutěží, kterých se účastní daný hráč,
- agregovaných statistik hráče v daném sportu,
- serializované soutěže pro export do souboru,
- informace o importované soutěži (tuto informaci zobrazuje jádro).

Kromě získávání informací má jádro také možnost spustit aktivitu pro vytvoření soutěže, aktivitu s detailem dané soutěže, službu pro smazání dané soutěže a službu pro import soutěže z daného souboru. Cesty k těmto aktivitám a exportované službě má modul uvedené v Manifestu. Modul má naopak možnost spustit aktivitu s detailem daného hráče.

Komunikace probíhá i mezi logickými vrstvami – moduly touto cestou získávají seznam hráčů. Moduly se k informacím o hráčích dostanou skrze rozhraní `IPackagePlayerManager`. Toto rozhraní je implementováno třídou `PackagePlayerManager`, která využívá ke komunikaci třídu `PlayerCP` z jádra. Jádro naopak komunikuje s moduly pomocí jejich *Content Provideru* při získávání informací o soutěžích daného hráče.

## 2.6 Knihovna

Knihovna obsahuje sadu sdílených funkcí, tříd a rozhraní. Tato sada by měla především usnadnit implementaci dalších modulů, které by tak implementovaly pouze specifika daných sportů a nikoli ty části, které jsou napříč sporty sdílené. Stejně jako jádro i knihovna je rozdělena na tři vrstvy, které si nyní jednotlivě rozebereme. Rozdělení můžeme vidět na obrázku 2.5.




Obrázek 2.5: Knihovna – diagram tříd

### 2.6.1 Prezentační vrstva

V této vrstvě nalezneme celkem 7 balíčků, které si nyní jednotlivě popíšeme. Tyto balíčky mohou být užitečné při implementaci nového modulu aplikace.

#### 2.6.1.1 Activities

V tomto balíčku nalezneme tři abstraktní třídy. Tyto třídy obsahují základ pro aktivity implementující obrazovky se záložkami (jako např. detail soutěže), se seznamem zaškrtávacích položek (např. při přidávání hráčů do turnaje) či jakoukoli jinou obrazovku vyžadující tzv. akční tlačítka v pravém horním rohu obrazovky.

#### 2.6.1.2 Adapters

Tento balíček obsahuje relativně velké množství tříd, které slouží k vykreslení seznamu různých entit – od abstraktní třídy pro seznam libovolných entit až po seznam zápasů, turnajů či týmů.

#### 2.6.1.3 Communication

Jak již bylo zmíněno u tohoto balíčku v rámci jádra – prezentační vrstva často komunikuje pomocí požadavků a odpovědí s přibalenými daty. V tomto balíčku nalezneme třídu `CrossPackageConstants`, která se využívá při komunikaci

mezi moduly, a třídu `ExtraConstants` využívanou při komunikaci v rámci komunikační vrstvy modulu.

#### 2.6.1.4 Dialogs

V tomto balíčku nalezneme třídy pro některá dialogová okna, která se mohou při implementaci hodit – prvním je dialogové okno s výběrem data, dalším je dialogové okno pro vytvoření týmu a posledním okno pro seřazení turnajů s výběrem sloupce, podle kterého chceme turnaje seřadit.

#### 2.6.1.5 Fragments

Balíček *fragments* obsahuje velké množství fragmentů. Najdeme zde fragmenty pro vytvoření následujících entit: soutěže, turnaje, zápasu či hráče. Dále tento balíček obsahuje fragmenty pro zobrazení detailu soutěže, turnaje či týmu. Nakonec zde máme fragment pro zobrazení seznamu zápasů, seznamu zaškrtávacích položek, seznam libovolných položek a obecný fragment s daty. Všechny tyto třídy jsou abstraktní a kromě seznamu zápasů všechny dědí od `AbstractDataFragment`. Ten nám usnadňuje odesílání požadavku o určitá data a následné přijetí odpovědi.

#### 2.6.1.6 Listeners

Zde najdeme jedinou třídu `PlayerDetailOnClickListener`, která umožní po kliknutí na položku odpovídající hráči spustit aktivitu s detailem hráče, která se nachází v jádru aplikace.

#### 2.6.1.7 Services

V tomto balíčku se nachází jediná třída, od které dědí např. `PlayerService` – tedy služba v jádru aplikace. Tato třída poskytuje informace o tom, zda se ještě určitý požadavek vyřizuje, či již byl vyřízen. Díky tomu můžeme uživateli zobrazit zprávu a aktuálním stavu požadavku a uživatel tak ví, že se aplikace nezastavila, ale stále pracuje.

### 2.6.2 Logická vrstva

Jelikož logická vrstva obsahuje samotnou logiku aplikace, budeme se jí věnovat detailněji než ostatním dvěma vrstvám. Ostatně již na obr. 2.5 jsme si kromě samotného balíčku *business* zobrazili i obsah balíčku pro práci s entitami (*managers*) a balíčku zajišťujícího serializaci (*serialization*).

**ManagerFactory** Toto je abstraktní třída, jejíž potomci by měli implementovat rozhraní `IManagerFactory`. Slouží k získání instancí *managerů* umístě-

ných v balíčku *managers* a také rozhraní `IDAOFactory`, pomocí kterého logická vrstva spolupracuje s datovou vrstvou.

#### 2.6.2.1 Entities

V tomto balíčku se nacházejí třídy pro entity používané v rámci logické vrstvy. Mezi tyto entity patří např. `PlayerAggregatedStats`, tedy třída pro agregované statistiky daného hráče. Tyto statistiky vznikají zpracováním dat z datové vrstvy.

#### 2.6.2.2 Generators

Tento balíček obsahuje mimo jiné tyto dvě třídy. První zajišťuje generování zápasů každý s každým. Metodě této třídy stačí předat seznam týmů či hráčů (v závislosti na typu soutěže) a číslo kola, které chceme vygenerovat. Druhá slouží ke generování vyvážených soupisek dle daných hráčských statistik.

#### 2.6.2.3 Loaders

V balíčku *loaders* se nacházejí entity používané při načítání informací o importované soutěži. Najdeme zde tedy např. entity *CompetitionImportInfo* či *PlayerImportInfo*. Kromě těchto tříd také nalezneme třídu pro tzv. konflikt, tedy rozdíl mezi informacemi o hráči ze souboru a z aplikace. Také zde najdeme pomocnou třídu, která vytváří instance konfliktů na základě dvou předaných hráčů.

#### 2.6.2.4 Managers

V tomto balíčku nalezneme třídy a rozhraní pro manažery veškerých entit z doménového modelu. V případě, že modul potřebuje tyto entity rozšířit, musí implementovat vlastní manažer. Ovšem k tomu může využít třídu `BaseManager` z tohoto balíčku, která je generická. Jejím rozšířením získá základní metody pro práci s vlastní rozšířenou entitou, jako je načtení, uložení, editace či smazání.

#### 2.6.2.5 Serialization

Na obrázku 2.5 nejsou zobrazeny všechny třídy, které se v tomto balíčku nachází. Kromě dvou zmíněných se zde nachází také abstraktní třídy pro serializaci soutěže, turnaje či týmu. Všechny tyto třídy obsahují základní sdílenou funkcionalitu, kterou mohou moduly při serializaci soutěže využít. V balíčcích *entities* se nachází entity potřebné pro serializaci – ta totiž probíhá formou převodu jakékoli struktury do třídy `ServerCommunicationItem`. Tato struktura je rekurzivní, obsahuje pole stejných tříd jako je struktura samotná.

### 2.6.3 Datová vrstva

V datové vrstvě nalezneme balíček *entities* s třídami pro veškeré entity z doménového modelu, dále balíček *helpers* s pomocnými metodami a konstantami a naposledy *interfaces* pro rozhraní využívané při např. komunikaci mezi logickou a datovou vrstvou.

**DAOFactory** Tato abstraktní třída je základem pro *factory* v modulu, pomocí kterého se získávají *DAO* pro jednotlivé entity doménového modelu.

**CompetitionCP** Tato abstraktní třída je základem pro *content provider* v modulu, pomocí kterého získává jádro informace o soutěžích daného hráče.

Kapitola 3

# Implementace

V této kapitole si nejprve představíme technologie použité při implementaci, poté popíšeme některé zajímavé algoritmy a výňatky z kódu a v poslední části kapitoly si představíme změny v modulu pro hokej a squash, které byly v rámci této práce taktéž realizovány.

## 3.1 Použité technologie

Během implementace jsme použili několik knihoven či frameworků, které stojí za zmínku a kterým se nyní budeme věnovat.

### 3.1.1 Gson

*Gson* je Java knihovna pro serializaci a deserializaci libovolných objektů do formátu *JSON* a zpět. Tato knihovna je, jak je již nejspíš zřejmé, využívaná při exportu soutěže do souboru a následném importu ze souboru zpět. Její použití je velmi snadné, jak můžeme vidět v ukázce kódu 3.1. Tato třída slouží jako mezičlánek serializace – celá struktura soutěže včetně turnajů, hráčů, apod. se převede na instanci `ServerCommunicationItem`. Tato instance obsahuje kromě různých polí i seznam instancí stejné třídy. Třída se poté serializuje do textového řetězce v metodě `toJson()`. Takto snadné použití serializuje celou instanci rekurzivně včetně všech polí. Odkaz do repozitáře knihovny (30. 12. 2016): `https://github.com/google/gson`.

### 3.1.2 ORMLite

*ORMLite* je Java knihovna poskytující objektově relační mapování. Díky této knihovně nemusíme v aplikaci psát SQL dotazy při práci s databází, ale využijeme dotazovacích metod nad objekty z této knihovny. Mapování je zajištěno pomocí anotací v třídě dané entity, jak můžeme vidět v ukázce kódu 3.2. Před názvem třídy je anotace `DatabaseTable`, kde máme možnost specifi-

kovat název tabulky a před každým atributem, který chceme v databázi mít, musíme uvést anotaci `DatabaseField`. Poté máme možnost specifikovat u každého atributu např. název sloupce v tabulce, zda může být hodnota atributu prázdná či zda se jedná o identifikátor. Kromě toho ORMLite vyžaduje, aby třída pro danou entitu měla i prázdný konstruktor. Nakonec je nezbytné pomocí `TableUtils.createTable(connectionSource, Player.class);` vytvořit tabulku pro danou entitu při vytváření databáze. Odkaz na tuto knihovnu (30. 12. 2016): `http://ormlite.com/`.

Ukázka kódu 3.1: Použití Gson

```
public class ServerCommunicationItem {
    public Long id;
    public String uid;
    ...
    public HashMap<String, Object> syncData;
    public List<ServerCommunicationItem> subItems = new ArrayList<>();

    public transient boolean serializeToken;
    public String toJson() {
        Gson gson = new GsonBuilder().serializeNulls().create();
        return gson.toJson(this);
    }
    ...
}
```

Ukázka kódu 3.2: Anotace u entit pro ORMLite

```
@DatabaseTable (tableName = DBConstants.tPLAYERS)
public class Player extends ShareBase implements Parcelable {
    @DatabaseField(generatedId = true, columnName = DBConstants.cID)
    private Long id;

    @DatabaseField(canBeNull = false, columnName = DBConstants.cNAME)
    private String name;

    @DatabaseField(unique = true, canBeNull = false, columnName =
        DBConstants.cEMAIL)
    private String email;

    @DatabaseField(columnName = DBConstants.cNOTE)
    private String note;

    public Player() {}
    ...
}
```

### 3.1.3 Espresso

*Espresso* je testovací framework umožňující psát automatické testy běžící na emulátoru OS Android. Díky němu tak můžeme testovat prezentační vrstvu aplikace včetně jednotlivých textů a hlášek, které se uživateli zobrazí. V Android Studiu je možné tyto testy vytvořit pomocí nástroje, kdy vývojář spustí *nahrávání*, což vyvolá spuštění aplikace na emulátoru. Vývojář poté prochází aplikaci tak, jak od testu očekává. V libovolný moment má možnost přidat verifikaci nějakého prvku – např. zda prvek na obrazovce existuje či nikoli nebo zda má daný prvek požadovanou textovou hodnotu. Po ukončení nahrávání je vygenerován spustitelný test, který toto chování dokáže opakovat. Příklady takových testů jsou v sekci 4.3.

### 3.1.4 Roboelectric

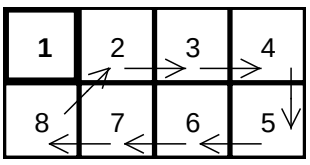
*Roboelectric* je další testovací framework, který nám naopak umožňuje spouštět automatické testy, které k běhu emulátor nevyžadují. Pomocí tohoto frameworku testujeme logickou a datovou vrstvu aplikace. Ukázky použití tohoto frameworku nalezneme v sekci 4.1 a 4.2.

## 3.2 Použité algoritmy

V této sekci si zmíníme dva zajímavé algoritmy, které jsme v aplikaci použili. Prvním algoritmem je generování zápasů každý s každým, tím druhým pak generování vyrovnaných soupisek.

### 3.2.1 Generování zápasů

Ke generování zápasů každý s každým používáme tzv. *Carousel-Berger systém*[9]. Aplikaci tohoto algoritmu můžeme vidět na ukázce kódu 3.3. Algoritmus si popíšeme na příkladu pro 8 účastníků. První perioda určí dvojice dle obr. 3.1 – hrají spolu vždy účastníci, kteří jsou na obrázku nad sebou. Dvojice pro další periodu jsou tvořeny posunem dle zmíněného obrázku. 1. účastník zůstává na svém místě, ostatní se posouvají. Tímto způsobem vytvoříme celkem 7 period – počet period je vždy o 1 menší, než počet účastníků.




Obrázek 3.1: Generování zápasů

Ukázka kódu 3.3: Algoritmus generování zápasů každý s každým

```
public class AllPlayAllMatchGenerator implements IMatchGenerator {
  @Override
  public List<Match> generateRound(List<Participant> parts, int
      round) {
    ArrayList<Match> matches = new ArrayList<>();
    int periods_number = parts.size() - (parts.size() + 1) % 2;
    int participants_even_number = parts.size() + parts.size() % 2;
    boolean oddness = parts.size() % 2 == 1 ? true : false;
    boolean round_oddness = round % 2 == 1 ? true : false;

    if (parts.size() <= 1)
       return matches;

    // array of participant indices - serves to generate matches
    ArrayList<Integer> arr = new ArrayList<>();
    for (int i = 0; i < participants_even_number; i++)
      arr.add(i);

    // each iteration generates one period
    int home_idx, away_idx;
    for (int period = 1; period <= periods_number; period++) {
      // each iteration generates one match
      for (int j = 0; j < participants_even_number / 2; j++) {
        if (round_oddness) {
          home_idx = arr.get(j);
          away_idx = arr.get(participants_even_number - 1 - j);
        } else {
          home_idx = arr.get(participants_even_number - 1 - j);
          away_idx = arr.get(j);
        }

        // skip if participants number is odd and participant does not
            exist
        if (oddness && (home_idx == parts.size() || away_idx ==
            parts.size()))
          continue;

        matches.add(createMatch(parts, home_idx, away_idx, period,
            round));
      }
      shift(arr);
    }
    return matches;
  }
  ...
}
```

### 3.2.2 Generování soupisek

Algoritmus pro generování soupisek je ještě jednodušší než předchozí algoritmus. Nejprve libovolně seřadíme účastníky a poté hráče dle vybrané statistiky. Následuje postupné přidělování hráčů jednotlivým týmům, nejprve ve směru vpřed, poté ve směru vzad. Poslednímu a prvnímu týmu jsou tedy v jednu chvíli přiděleni dva hráči po sobě.

Ukázka kódu 3.4: Algoritmus generování vyrovnaných soupisek

```
public class Balancedgenerator implements Igenerator {
    @Override
    public boolean generateRosters(List<Team> teams, Map<Long,
        Player> players, Map<Long, Double> stats) {
        if (teams.isEmpty())
            return false;

        for (Team t : teams)
            if (!t.getPlayers().isEmpty())
                return false;

        Collections.shuffle(teams, new Random(System.nanoTime()));
        int team_index = 0;
        boolean dir_forward = true;

        Map<Long, Double> sortedMap = MapComparator.sortByValue(stats);
        for (Map.Entry<Long, Double> e : sortedMap.entrySet()) {
            // Some players from competition are not in this tournament
            if (players.get(e.getKey()) == null)
                continue;

            teams.get(team_index).addPlayer(players.get(e.getKey()));

            if (team_index < teams.size()-1 && dir_forward) {
                team_index++;
            } else if (team_index == teams.size()-1 && dir_forward) {
                dir_forward = false;
            } else if (team_index == 0 && !dir_forward) {
                dir_forward = true;
            } else if (team_index <= teams.size()-1 && !dir_forward) {
                team_index--;
            }
        }

        return true;
    }
}
```

## 3.3 Úpravy v modulu pro hokej

V tomto modulu bylo původně možné evidovat soutěže pouze v hokeji. Aktuálně je možné evidovat soutěže i ve florbale, neboť je to sport typově úplně stejný jako hokej. Při spouštění modulu z jádra (či při získávání seznamu soutěží), musí jádro uvést kontext, ve kterém chce s tímto modulem pracovat – hokej či florbal. Seznam všech kontextů, které umí tento modul obsloužit je uveden v manifestu. Úpravy tohoto modulu probíhaly nejprve v prezentační vrstvě, kde byly sjednoceny všechny obrazovky tak, aby prvky pro vytváření či ukládání entit byly vždy na stejném místě a ovládání aplikace tak bylo snazší.

V logické vrstvě je změn několik. Tou první je využití sdílených *manažerů* z knihovny. Díky tomu máme v modulu jen kód specifický pro tento modul. Dále přibyl balíček *serialization*, který je schopný serializovat a deserializovat soutěž tohoto modulu včetně všech jejích součástí. Poslední hlavní změnou v této vrstvě je vytvoření balíčku *loaders*, který obsahuje třídy umožňující načíst informace o importované soutěži. Třídy tohoto balíčku samozřejmě využívají některé třídy z balíčku *serialization*.

Datová vrstva prošla relativně velkou změnou. Použitím *ORMLite* zmizely všechny původní *DAO* třídy. Namísto nich je zde *HockeyDAOFactory* rozšiřující *DAOFactory* z knihovny a je schopna díky generické metodě z knihovny poskytnout DAO pro libovolnou entitu.

Nezbytné bylo samozřejmě i upravit třídy pro testování. Ty totiž původně testovaly z velké části *DAO* třídy. Tyto třídy nám nyní poskytuje *ORMLite*, a tudíž nám stačí otestovat jejich spolupráci s *manažery*. Dále zde testujeme serializaci a načítání informací o importované soutěži.

## 3.4 Úpravy v modulu pro squash

V tomto modulu bylo původně možné evidovat pouze soutěže ve squashi. Aktuálně je však možné evidovat soutěže i v následujících sportech: v badmintonu, plážovém volejbale, tenise či volejbale. Další úpravy v tomto modulu byly podobné jako v modulu pro hokej. Prezentační vrstvy modulů pro squash i hokej byly sjednoceny tak, aby uživatel takřka nepoznal, že se reálně jedná o různé aplikace.

V logické vrstvě byly provedeny stejné změny jako v předchozím případě.

Stejně tak v datové vrstvě použitím *ORMLite* zmizely všechny původní *DAO* třídy. Namísto nich je zde *SquashDAOFactory* rozšiřující *DAOFactory*.

Testovací třídy prošly samozřejmě také změnou, nyní zde najdeme třídy pro testování serializace, načítání informací o importované soutěži a testování manažerů.

## 3.5 Implementace nového modulu

Jedním z cílů této práce je také zdokumentovat jádro z pohledu implementace nového modulu. V této sekci si tedy ve zkratce popíšeme, co vše je potřeba zajistit, abychom nový modul úspěšně zakomponovali do celého systému. Podrobný popis najdeme na přiloženém CD. Požadavky na modul si rozdělíme na několik částí.

### 3.5.1 Android Manifest

Jak již víme, tento XML soubor obsahuje informace o aplikaci důležité pro OS Android. Jádro od modulu vyžaduje, aby tento soubor obsahoval následující údaje:

- pod klíčem *application_type* hodnotu *TournamentManagerPackage*,
- pod klíčem *context_names* názvy všech modulem implementovaných sportů oddělené čárkou,
- pod klíčem *package_name* celou cestu balíčku, ve kterém je modul implementován (tato cesta je stejná s cestou uvedenou u kořenového tagu *manifest* v atributu *package*),
- pod klíčem *activity_create_competition* celou cestu k aktivitě pro vytvoření nové soutěže,
- pod klíčem *activity_detail_competition* celou cestu k aktivitě pro zobrazení detailu soutěže,
- pod klíčem *package_service* celou cestu k exportované službě.

### 3.5.2 Aktivita pro vytvoření nové soutěže

Modul musí obsahovat exportovanou aktivitu pro vytvoření nové soutěže. Ta obdrží při spuštění v přibalených datech pod klíčem *extra_sport_context* název sportu, tedy kontext, ve kterém má být soutěž vytvořena. Tato aktivita se používá i k editaci existující soutěže, v tomto případě je aktivitě předáno i id soutěže v klíči *extra_id*.

### 3.5.3 Aktivita pro zobrazení detailu soutěže

Modul musí dále obsahovat exportovanou aktivitu pro zobrazení detailu soutěže. Té je při spuštění opět předán kontext a id soutěže, jejíž detail chceme zobrazit.

### 3.5.4 Exportovaná služba

Exportovaná služba modulu musí být schopna odpovědět celkem na 7 požadavků:

- seznam všech soutěží,
- odstranění dané soutěže,
- agregované statistiky daného hráče,
- seznam soutěží, kterých se účastní daný hráč,
- serializaci soutěže,
- načtení informací o soutěži z daného souboru,
- import soutěže z daného souboru.

Odpovědi na tyto požadavky musí mít samozřejmě specifický formát, který najdeme v podrobném popisu na přiloženém CD.

### 3.5.5 Content provider

Modul musí dále implementovat *Content provider* poskytující jádru seznam soutěží, kterých se účastní daný hráč.

Kapitola 4

# Testování

Poslední kapitolou této práce je testování, které je nedílnou součástí vývoje software. Pomáhá totiž nejen odhalovat chyby včas, což snižuje náklady na jejich opravu, ale může také sloužit jako nástroj pro měření či zajištění kvality. Testy si v této kapitole rozdělíme podle rozsahu na jednotkové, integrační a systémové.

## 4.1 Jednotkové testy

Jednotkové testy testují samostatně jednotlivé a nezávislé třídy či metody[10]. Pomocí těchto testů pokryjeme mimo jiné třídy pro generování zápasů a generování vyrovnaných soupisek, které si popíšeme podrobněji. Všechny naše testy dědí od třídy `android.test.AndroidTestCase`. Android Studio nám poté umožní tyto spouštět a vyhodnocovat např. kolik procent kódu jsme testy pokryli. Dále díky tomu získáme přístup k metodám, jako jsou např. *assertEquals* nebo *assertNotNull*. Tyto a další metody nám umožní ověřit, zda jsou verifikované objekty dle požadavků.

### 4.1.1 Generování zápasů každý s každým

Část testu této třídy, resp. metody můžeme vidět na ukázce kódu 4.1. Najdeme zde nejprve metodu `setUp` s anotací `@Before`. Tato metoda je spouštěna před každým testovacím případem a slouží k nastavení prostředí. Na ukázce v této metodě vytváříme samotný generátor zápasů. Obdobně existuje anotace `@After`, kterou umístíme k metodě spouštěné po každém testovacím případě.

Další metody jsou již anotované pomocí `@Test`, tedy samotné testovací případy. Vidíme zde testovací případ, kdy se pokusíme vygenerovat kolo zápasů nejprve bez účastníků a poté pro jednoho účastníka, což samozřejmě není možné. Dalším případem ověřujeme, že při generování dvou kol zápasů každý s každým opravdu každý účastník odehraje právě jeden zápas s každým dalším účastníkem jako domácí a právě jeden zápas jako hostující účastník.

Ukázka kódu 4.1: Testování metody pro generování zápasů

```
public class AllPlayAllMatchGeneratorTest extends AndroidTestCase {
    private IMatchGenerator matchGenerator = null;

    @Before
    public void setUp() {
        matchGenerator = new AllPlayAllMatchGenerator();
        assertNotNull(matchGenerator);
    }

    /* Verify no match is generated for one or no participant. */
    @Test
    public void verifyNoMatchGenerated() {
        List<Participant> parts = new ArrayList<>();
        List<Match> round = matchGenerator.generateRound(parts, 1);
        assertNotNull(round);
        assertTrue(round.isEmpty());

        parts.add(new Participant(-1, 1, null));
        round = matchGenerator.generateRound(parts, 1);
        assertNotNull(round);
        assertTrue(round.isEmpty());
    }

    /* Verify that all participants play with all */
    @Test
    public void verifyAllPlayAll() {
        List<Participant> parts = new ArrayList<>();
        parts.add(new Participant(-1, 1, null));
        ...
        parts.add(new Participant(-1, 4, null));

        List<Match> round = matchGenerator.generateRound(parts, 1);
        List<Match> round2 = matchGenerator.generateRound(parts, 2);
        round.addAll(round2);

        assertNotNull(round);
        assertEquals(12, round.size());

        /* Verify that each participant play with each other
            participant twice,
        once as home and once as away */
        assertEquals(1, countMatchesForHomeAndAwayId(round, 1, 2));
        assertEquals(1, countMatchesForHomeAndAwayId(round, 2, 1));
        ...
    }
}
```

### 4.1.2 Generování vyrovnaných soupisek

Část testu této třídy, resp. metody můžeme vidět na ukázce kódu 4.2. V metodě `setUp` opět nejprve vytváříme samotný generátor soupisek. Následuje testovací případ, kde se snažíme generovat soupisky, ačkoli máme prázdný seznam týmů, což samozřejmě nelze. V dalším testovacím případu už testujeme samotné generování vyrovnaných soupisek. Nejprve si vytvoříme 4 týmy a poté 8 hráčů, kterým přidělíme individuální statistiku tak, že první hráč je v této statistice nejlepší, druhý je hned za ním, atd., dokud nedojdeme k poslednímu hráči, který je v této statistice nejhorší. Poté vygenerujeme vyrovnané soupisky a ověříme následující tři věci:

- vygenerování proběhlo úspěšně,
- každý tým obsahuje právě dva hráče,
- v jednotlivých týmech spolu hrají nejlepší s nejhorším, druhý nejlepší s druhým nejhorším, atd.

## 4.2 Integrační testy

Integrační testy slouží k testování spolupráce komponent v systému. V našem konkrétním případě takto testujeme *manažery* jednotlivých entit, které ke své práci potřebují databázi a DAO objekty z datové vrstvy. Příklad takového testu nalezneme v ukázce 4.3. Na této ukázce nalezneme v anotaci před názvem testovací třídy použití frameworku *Roboelectric*, který jsme zmiňovali v předchozí kapitole. Tento framework nám umožní např. získat *context*, tedy referenci na aktuální aplikaci, ačkoli aplikace neběží na žádném zařízení či emulátoru. Tuto referenci získáváme v metodě `setUp`.

Další metodou je `reset` s anotací `@After`. Tato metoda je tedy spuštěna po každém testovacím případu a v tomto případě nuluje všechny *Singleton*[5] objekty, které se ve třídě `ManagerFactory` nachází. Toto je nezbytné pro fungování dalších testovacích případů.

Testovacím případem uvedeným na zmíněné ukázce (reálně jich je v testovací třídě více, ovšem ostatní jsme z důvodu stručnosti vynechali) je smazání hráče. V tomto případě tedy nejprve vytvoříme hráče a ověříme, že byl opravdu vytvořen. Následně se pokusíme hráče smazat a ověříme, zda je při volání `playerManager.getAll()` vrácen prázdný seznam. Ověření, že byl hráč opravdu vytvořen, je důležité – pokud by se hráče nepodařilo vytvořit, metoda pro vrácení seznamu všech hráčů by vracela očekávaný prázdný seznam a test by tak indikoval správné chování, což by ale nebyla pravda.

[5] *Singleton*, tzv. jedináček, je návrhový vzor, který používáme v případě, kdy vyžadujeme, aby v celém systému existovala jediná instance objektu.

Ukázka kódu 4.2: Testování metody pro generování vyrovnaných soupisek

```
public class BalancedRostersGeneratorTest extends AndroidTestCase {
    private ITeamsRostersGenerator generator = null;

    @Before
    public void setUp() {
        generator = new BalancedTeamsRostersGenerator();
        assertNotNull(generator);
    }

    @Test
    public void verifyNoTeams() {
        assertFalse(generator.generateRosters(new ArrayList<>(),
        new HashMap<>(), new HashMap<>()));
    }

    @Test
    public void verifyRostersSize() {
        List<Team> teams = new ArrayList<>();
        Team team1 = new Team(1, "Team A");
        ...
        teams.add(team1);
        ...

        Map<Long, Player> playersMap = new HashMap<>();
        playersMap.put(1L, new Player(1, "Player 1", "", ""));
        ...

        Map<Long, Double> statsMap = new HashMap<>();
        statsMap.put(1L, 8D);
        ...
        statsMap.put(8L, 1D);
        assertTrue(generator.generateRosters(
                            teams, playersMap, statsMap));

        /* Verify each team has two players */
        assertEquals(2, team1.getPlayers().size());
        ...

        /* Verify player 1 is in team with 8, 2 with 7, and so on */
        for (Team team : teams) {
            Player first = team.getPlayers().get(0);
            Player second = team.getPlayers().get(1);
            assertEquals(9, first.getId() + second.getId());
        }
    }
}
```

Ukázka kódu 4.3: Testování třídy *PlayerManager*

```
@RunWith(RobolectricTestRunner.class)
@Config(constants = BuildConfig.class)
public class PlayerManagerTest extends AndroidTestCase {
    private Context context;
    private static IPlayerManager playerManager = null;

    private static final String name = "Martin";
    private static final String email = "martin@email.cz";
    private static final String note = "606 111 222";
    private static long playerId;

    @Before
    public void setUp() {
        context =
            Shadows.shadowOf(RuntimeEnvironment.application).getApplicationContext();
        playerManager =
            ManagerFactory.getInstance(context).getEntityManager(Player.class);

        /* Preconditions */
        assertNotNull(ManagerFactory.getInstance(context));
        assertNotNull(playerManager);
    }

    @After
    public void reset() {
        ManagerFactory.reset();
    }

    /**
     * Verify delete Player
     */
    @Test
    public void delete() {
        Player inserted = new Player(name, note, email);
        playerManager.insert(inserted);
        playerId = inserted.getId();
        assertTrue(playerId > 0);

        assertTrue(playerManager.delete(playerId));
        Player player = playerManager.getById(playerId);
        assertNull(player);

        players = playerManager.getAll();
        assertNotNull(players);
        assertTrue(players.isEmpty());
    }
    ...
}
```

Tabulka 4.1: Testovací scénář odstranění hráče

| **Kroky scénáře** | **Očekávaný výsledek** |
|---|---|
| Nainstalovat a spustit aplikaci | Aplikace je spuštěna |
| Přejít na seznam hráčů | Zobrazí se prázdný seznam |
| Stisknout tlačítko pro vytvoření hráče | Zobrazí se formulář pro vytvoření hráče |
| Vyplnit jméno a e-mailovou adresu | – |
| Stisknout tlačítko pro uložení hráče | Zobrazí se seznam hráčů s nově vytvořeným hráčem |
| Dlouze stisknout hráče | Zobrazí se dialogové okno s možnostmi pro editaci a smazání hráče |
| Vybrat možnost pro smazání hráče | Zobrazí se prázdný seznam |

## 4.3 Systémové testy

Systémové testy testují systém jako celek v reálném prostředí. Těmito testy otestujeme chování aplikace z pohledu uživatele – tedy přes uživatelské rozhraní. V této kategorii využijeme framework *Espresso*, který jsme zmiňovali v předešlé kapitole. Tento framework nám umožňuje procházet aplikaci v Android emulátoru tak, jako by to dělal reálný uživatel na mobilním zařízení s OS Android.

Pro napsání takových testů je potřeba definovat scénáře. Tyto scénáře vycházejí z případů užití. Příklady takových scénářů najdeme v tabulce 4.1. Ukázku některých funkcí napsaných za použití frameworku *espresso* najdeme na ukázce 4.4. Tyto statické funkce se nachází ve třídě `CommonFunctions` a ostatní testy je tak mohou využít. Na zmíněné ukázce můžeme vidět celkem tři funkce. První dvě jsou navigační – do seznamu hráčů se dostaneme pomocí `navigateToPlayers` a do formuláře pro vytvoření hráče pomocí funkce `navigateToPlayerCreate`. Naopak odstranění vybraného hráče ze seznamu nám umožňuje funkce `deletePlayerAtPosition`.

Před každou touto funkcí z `CommonFunctions` najdeme komentář, který kromě popisu významu funkce obsahuje i informaci, za jakých podmínek můžeme funkci použít. Např. u funkce `navigateToPlayers` vidíme, že lze použít pouze z jedné ze tří výchozích obrazovek – seznamu soutěží, hráčů a nastavení. Dále můžeme vidět, že i tyto funkce se volají navzájem – např. při navigaci do formuláře pro vytvoření hráče nejprve použijeme funkci pro navigaci do seznamu hráčů. Teprve potom se navigujeme do zmíněného formuláře.

Ostatní funkcionalita, která není pokryta automatizovanými testy, je pokryta testy manuálními.

Ukázka kódu 4.4: Některé sdílené funkce pro systémové testování

```
/**
 * Navigation to Players.
 * Can be used only from Competitions / Players / Settings screen.
 */
public static void navigateToPlayers() {
    openMenu();
    ViewInteraction appCompatCheckedTextView = onView(
            allOf(withId(R.id.design_menu_item_text),
                withText(R.string.players), isDisplayed()));
    appCompatCheckedTextView.perform(click());
}

/**
 * Navigation to Create Player.
 * Can be used only from Competitions / Players / Settings screen.
 */
public static void navigateToPlayerCreate() {
    navigateToPlayers();
    ViewInteraction floatingActionButton = onView(
            allOf(withClassName(is(
            "android.support.design.widget.FloatingActionButton")),
                isDisplayed()));
    floatingActionButton.perform(click());
}

/**
 * Deletion Player on specified position.
 * Can be used only from Players.
 */
public static void deletePlayerAtPosition(int position) {
    ViewInteraction recyclerView = onView(
            allOf(withId(R.id.recycler_view), isDisplayed()));
    recyclerView.perform(actionOnItemAtPosition(position,
        longClick()));

    ViewInteraction appCompatTextView = onView(
            allOf(withId(android.R.id.text1),
                withText(R.string.delete),
                    childAtPosition(
                            allOf(withId(R.id.select_dialog_listview),
                                    withParent(withId(R.id.contentPanel))),
                            1),
                    isDisplayed()));
    appCompatTextView.perform(click());
    SystemClock.sleep(2000);
}
```

# Závěr

Hlavním cílem této práce bylo navrhnout a vytvořit jádro a knihovnu mobilní aplikace Tournament Manager pro OS Android. Jak by mělo být z této práce patrné, tento cíl byl splněn. Aplikace přiložená na datovém nosiči je toho důkazem. Také se nám podařilo splnit jednotlivé dílčí cíle. U některých si to můžeme dokonce snadno ověřit – např. podporu realizace různých sportů formou samostatných modulů. Na přiloženém datovém nosiči jsou totiž kromě jádra aplikace i oba moduly implementující celkem sedm sportů. Po jejich instalaci tedy aplikace umožní evidovat soutěže a turnaje právě v těchto sportech.

## Budoucí rozvoj

Tento projekt samozřejmě nekončí odevzdáním této práce. V úvodu jsme zmínili další současně probíhající práce. Hlavní rozvoj, který se v současné chvíli nabízí, je propojení webové a serverové části aplikace s naší mobilní aplikací. To by mělo umožnit především snazší synchronizaci, kdy si uživatelé nemusí vyměňovat soubory s exportovanou soutěží, ale pomocí jednoduchého rozhraní získají ze serveru informace o dané soutěži.

Další možnosti se otevírají při implementaci nových modulů. Aktuální počet podporovaných sportů je pouze sedm a chybí např. podpora pro jeden z nejpopulárnějších sportů – fotbal. Rozšíření sady podporovaných sportů je tedy hlavní směr, kterým by se měla odvíjet další cesta tohoto projektu.

# Literatura


[1] MÁCA, V.: *Tournament Manager – klient pro Windows 10 Mobile*. Diplomová práce, České vysoké učení technické v Praze, Fakulta informačních technologií, Katedra softwarového inženýrství, 2017.

[2] HACURA, M.: *Tournament Manager – synchronizace*. Diplomová práce, České vysoké učení technické v Praze, Fakulta informačních technologií, Katedra softwarového inženýrství, 2017.

[3] KOŠUT, O.: *Tournament Manager – balíček pro hokej*. Bakalářská práce, České vysoké učení technické v Praze, Fakulta informačních technologií, Katedra softwarového inženýrství, 2016.

[4] CHMEL, V.: *Tournament Manager – balíček pro squash*. Bakalářská práce, České vysoké učení technické v Praze, Fakulta informačních technologií, Katedra softwarového inženýrství, 2016.

[5] Android Developers: *Dashboards*. [online], 2016, [cit. 2016-12-21]. Dostupné z: `https://developer.android.com/about/dashboards/index.html`

[6] Android Developers: *App Manifest*. [online], 2016, [cit. 2016-12-25]. Dostupné z: `https://developer.android.com/guide/topics/manifest/manifest-intro.html`

[7] Android Developers: *Activity*. [online], 2016, [cit. 2016-12-25]. Dostupné z: `https://developer.android.com/reference/android/app/Activity.html`

[8] Android Developers: *Fragment*. [online], 2016, [cit. 2016-12-25]. Dostupné z: `https://developer.android.com/reference/android/app/Fragment.html`

[9] FRANÇAISE DES ÉCHECS: *Le livre de l'arbitre*. Saint-Quentin-en-Yvelines: FFE, 2008 vydání, ISBN 978-2-915853-01-8.

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

Příloha **A**

# Seznam použitých zkratek

**DAO** Data access object – objekt pro přístup k datům

**GUI** Graphical user interface – grafické uživatelské rozhraní

**JSON** Javascript object notation – datový formát

**OS** Operační systém

**SQL** Structured query language – dotazovací jazyk pro práci s databází

**XML** Extensible markup language – obecný značkovací jazyk

Příloha **B**

# Obsah přiloženého CD

- `DP_Němeček_Josef_2017.pdf` ..... text práce ve formátu PDF
- `build` ..... zkompilované soubory
  - `Tournament Manager.apk` ..... jádro aplikace
  - `TM Hockey.apk` ..... modul pro sporty na bázi hokeje
  - `TM Squash.apk` ..... modul pro sporty na bázi squashe
  - `TMLibrary.jar` ..... knihovna aplikace
- `doc`
  - `TM - Nový modul.pdf` ..... příručka pro implementaci nového modulu
  - `Tournament Manager/` ..... vygenerovaná dokumentace aplikace
- `src`
  - `text/` ..... zdrojová forma práce ve formátu LaTeX
  - `Tournament Manager/` ..... zdrojové kódy implementace