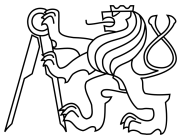
ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ



ZADÁNÍ DIPLOMOVÉ PRÁCE

Název:	Tournament Manager - klient pro Windows 10 Mobile
Student:	Bc. Vlastimil Máca
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 klienta aplikace Tournament Manager pro OS Windows 10 Mobile. Aplikace by měla umožnit organizování soutěží a turnajů v různých sportech, evidenci zápasů, jejich výsledků, složení týmů a statistik hráčů. Aplikace musí podporovat implementaci různých sportů formou samostatných balíčků.

Cíle práce:
- Analyzujte odlišnosti systému Android a Windows 10 Mobile s ohledem na funkce aplikace Tournament Manager.
- Navrhněte potřebné úpravy architektury a návrhu aplikace Tournament Manager pro Android pro její implementaci pro Windows 10 Mobile.
- Implementujte jádro aplikace a vzorový balíček pro vybraný týmový sport.
- Implementovanou aplikaci řádně otestujte.
- Zdokumentujte jádro aplikace z pohledu jádra a z pohledu implementace nového balíčku, zdokumentujte vzorový balíček.

Seznam odborné literatury

Dodá vedoucí práce.

L.S.

Ing. Michal Valenta, Ph.D.
vedoucí katedry

prof. Ing. Pavel Tvrdík, CSc.
děkan

V Praze dne 9. prosince 2015

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

Diplomová práce

Tournament Manager - klient pro Windows 10 Mobile

Bc. Vlastimil Máca

Vedoucí práce: Ing. Zdeněk Rybola

17. února 2017

Poděkování

Rád bych poděkoval vedoucímu práce, panu Ing. Zdeňku Rybolovi, za výborné vedení práce a ochotu. Pak bych rád poděkoval svým kolegům z týmového projektu, panu Josefu Němečkovi, Michalu Hacurovi, Ondřeji Košutovi a Václavu Chmelovi za skvělou týmovou spolupráci. V neposlední řadě bych rád poděkoval své rodině za poskytnutí ideálních podmínek ke studiu a vytvoření této práce.

Prohlášení

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

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů, zejména skutečnost, že České vysoké učení technické v Praze má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona.

V Praze dne 17. února 2017 .....

Odkaz na tuto práci

Máca, Vlastimil. *Tournament Manager - klient pro Windows 10 Mobile*. Diplomová práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2017.

Abstrakt

Autor se v této práci zabývá tvorbou aplikace pro systém Windows 10 Mobile. V prvních kapitolách se zabývá analýzou existující modulární aplikace Tournament Manager pro OS Android. V další kapitole provádí návrh architektury aplikace a vzorového modulu. V posledních kapitolách se pak zabývá popisem realizace, testování a dokumentace aplikace. Výsledkem práce je částečně modulární aplikace nejen pro Windows 10 Mobile, ale díky vývoji na Univerzální Platformě Windows také pro desktopová zařízení se systémem Windows 10.

Klíčová slova Mobilní aplikace, Tournament Manager, Windows 10, UWP, C#, modulární aplikace, modul, sport

Abstract

The author of this work deals with creating of application for OS Windows 10 Mobile. In the first chapters author analyzes the existing modular application Tournament Manager for OS Android. In the next chapter, the author performs design of application architecture and model module. In the last chapters author describes the implementation, testing and documentation of

application. The result is a partially modular application for Windows 10 Mobile, but thanks to the usage of Universal Windows Platform also for desktop devices with Windows 10.

Keywords Mobile application, Tournament Manager, Windows 10, UWP, C#, modular application, module, sport

Obsah

Úvod **1**

1 Cíl práce **3**

2 Analýza **5**
2.1 Analýza Android aplikace a sběr požadavků 5
2.2 Rešerše existujících aplikací pro OS Windows 10 Mobile 11
2.3 Návrh uživatelského rozhraní aplikace 14
2.4 Doménový model 18

3 Návrh **23**
3.1 Seznámení se s vývojem pro OS Windows 10 Mobile 23
3.2 Struktura platformy 28
3.3 Struktura aplikace 29
3.4 Návrh knihovny aplikace 33
3.5 Návrh jádra 49
3.6 Návrh modulu 54
3.7 Postup pro vývoj modulu a jeho distribuci do aplikace 58

4 Realizace **61**
4.1 Ukázka GUI aplikace 61
4.2 Ukázka kódu 62
4.3 Vytvořené SDK 62
4.4 Lokalizace Aplikace 63

5 Testování a dokumentace **67**
5.1 Jednotkové testy 67
5.2 Uživatelské testy 67
5.3 Integrační testy 68
5.4 Dokumentace 69

Závěr **71**

Literatura **73**

A Seznam použitých zkratek **75**

B Obsah přiloženého CD **77**

Seznam obrázků

2.1 Ukázka aplikace MSN Sport. 12
2.2 Ukázka aplikace TourneyMan. 13
2.3 Diagram přechodů obrazovek. 15
2.4 Doménový model. 18

3.1 Architektura platformy. 28
3.2 Architektura aplikace. 31
3.3 Architektura knihovny aplikace 34
3.4 Přehled tříd knihovny aplikace. 34
3.5 Databázový model. 36
3.6 Rozhraní pro fungování architektury aplikace. 37
3.7 Rozhraní DAO tříd. 39
3.8 Rozhraní Managerů. 40
3.9 Architektura jádra aplikace. 49
3.10 Diagram tříd pro jádro aplikace. 50
3.11 Architektura modulu aplikace. 55
3.12 Diagram tříd pro modul aplikace. 55

4.1 Vlevo - GUI aplikace pro Android. Vpravo - GUI aplikace pro W10M 61

Úvod

I přes ohromné množství mobilních aplikací pro systém OS Android, nebyla nalezena aplikace plně vyhovující požadavkům na evidenci statistik sportovních aktivit zadavatele této práce. To vedlo k tomu, že vytvořil vlastní aplikaci pro Android, která se stala hlavní inspirací týmového projektu. Společný týmový projekt, jehož součástí je tato práce, se skládá ze tří diplomových a dvou bakalářských prací, jejichž cílem je vytvořit platformu pro aplikaci Tournament Manager sloužící pro evidenci statistik v různých sportech. Hlavní částí této platformy je aplikace pro OS Android[1]. K této aplikaci jsou řešeny implementace dvou balíčků pro sporty Hokej[2] a Squash[3]. Další částí projektu je aplikace pro Windows 10 Mobile, jejíž tvorbou se zabývá tato práce. Spojujícím prvkem těchto aplikací je synchronizační webový server[4], starající se o synchronizaci dat mezi zařízeními a o prezentaci dat na webu.

Kapitola 1

Cíl práce

Cílem této práce je vytvořit aplikaci Tournament Manager pro OS Windows 10 Mobile.

Aplikace vychází ze stejnojmenné aplikace pro systém Android a musí splňovat všechny kladené požadavky zmíněné v zadání. Zejména je to možnost samostatně a odděleně vytvářet balíky pro jednotlivé sporty a zachovat funkční požadavky aplikace pro systém Android.

Pro úspěšné vytvoření aplikace je nutné provést několik analýz. Nejdříve proběhne analýza funkčností a vlastností aplikace pro OS Android. Dále bude vytvořena rešerše existujících podobných aplikací na OS Windows 10 Mobile. Na základě analýzy a této rešerše bude proveden návrh uživatelského rozhraní aplikace. Poslední analytickou částí bude popsání Doménového modelu.

Po analýzách bude řešen návrh aplikace. Nejdříve proběhne seznámení se s technologií pro vytváření aplikací v OS Windows 10 Mobile. Následně bude proveden návrh řešení modulární architektury aplikace. Poté bude proveden návrh jednotlivých částí aplikace. Na základě tohoto návrhu pak bude provedena implementace aplikace a jednoho ukázkového balíčku.

Na závěr, po dokončení implementace bude aplikace i vytvořený balíček důkladně otestován a bude vytvořena dokumentace aplikace i vytvořeného modulu, včetně příručky pro tvorbu dalších balíčků.

Kapitola 2

Analýza

V rámci této kapitoly bude provedeno několik analýz. Nejdříve bude provedena analýza vlastností a funkčností mobilní aplikace Tournament Manager pro OS Android, která vznikala v rámci tohoto společného týmového projektu. Budou analyzovány její funkční a nefunkční požadavky, se zaměřením zejména na modulární funkčnost aplikace, kterou bude dále v práci nutné navrhnout pro OS Windows 10 Mobile.

Následně bude provedena rešerše podobných existujících aplikací pro systém Windows 10 Mobile. V rámci této rešerše budou analyzovány aplikace z oblasti sportu, zejména aplikace zaměřující se na evidenci sportovních statistik.

Poté bude představen návrh uživatelského rozhraní, který je v obou aplikacích podobný.

Nakonec bude popsán doménový model, který byl vytvořen v rámci společné práce na týmovém projektu a bude sloužit jako základ pro návrh aplikace.

2.1 Analýza Android aplikace a sběr požadavků

Tato sekce obsahuje analýzu aplikace pro OS Android a z ní vyplývající požadavky na aplikaci pro OS Windows 10 Mobile.

Požadavky jsou rozděleny na funkční a nefunkční. Funkční požadavky jsou požadavky, které popisují jaké funkce musí aplikace splňovat. Neřeší jak je těchto funkcí dosaženo. Nefunkční požadavky jsou pak požadavky které popisují technická omezení, které musí aplikace splňovat.

Analýza aplikace bude orientována zejména na systémové funkční požadavky aplikace, které souvisí se systémem OS Android a architekturou aplikace, jelikož ty pravděpodobně budou v aplikaci pro OS Windows 10 Mobile odlišné. Tyto požadavky budou označeny jako FS[1]. Zbylými funkčními po-

[1] FS - Funkční Systémové požadavky

žadavky se analýza nebude zabývat příliš podrobně, protože budou vesměs stejné jako v aplikaci pro OS Android a již jsou detailně popsány v diplomové práci Josefa Němečka[1].

Architektura aplikace pro OS Android, na kterou bude zaměřena tato analýza, je architektura modulární. To je taková architektura, která se skládá z několika oddělitelných částí. Většinou mívá jednu hlavní část, které se říká jádro a která vše řídí. Zbylým částem se pak říká balíčky či moduly a ty se k jádru připojují a přidávají mu funkčnost. V této práci bude pro části architektury preferováno označení modul. Výhoda této architektury je, že je možné do aplikace moduly přidávat či odebírat dle potřeby. Druhou velkou výhodou této architektury je možnost vyvíjet její části nezávisle.

Aplikace byla analyzována postupným průchodem aplikací. Průchod se skládá z několika kroků. Tím prvním byla instalace aplikace a její první spuštění, kde byly zkoumány vlastnosti a chování aplikace v rámci systému Android. Následujícími kroky pak byly analýzy jednotlivých obrazovek. Poznatky a požadavky z těchto obrazovek byly sdruženy a zaznamenány podle třech hlavních úrovní aplikace - soutěže, turnaje a zápasy.

Analýza aplikace byla realizována na zařízení Meizu M9 s OS Android 4.0.1. a velikostí displaye 3.5". Jedná se tedy o starší zařízení, s minimální podporovanou verzí OS Android. Tuto analýzu lze tedy také považovat za ověření funkčnosti s minimální nutnou konfigurací.

2.1.1 Účel aplikace

Aplikace Tournament Manager slouží uživatelům k organizaci a evidenci jejich sportovních aktivit - většinou týmových sportů. Příkladem typického použití aplikace může být následující scénář.

Uživatel aplikace, si jde každý víkend zahrát florbal se svými přáteli. Tito přátelé takto spolu hrají pravidelně a chtějí si evidovat statistiky, aby věděli, jak se zlepšují a jak si kdo vede.

Uživatel tedy v aplikaci eviduje tuto florbalovou soutěž a její hráče. Do ní si každý víkend vytvoří turnaj a v rámci turnaje se hrají zápasy. Před začátkem turnaje uživatel v aplikaci vygeneruje náhodné rozložení týmů a poté vygeneruje zápasy. Takže má přehled, kdo s kým a kdy bude hrát. Po dokončení zápasu uživatel vždy zaznamená výsledek zápasu a statistiky hráčů. Další víkend uživatel znovu vytvoří turnaj, ale tentokrát už může hráče do týmů rozdělit podle jejich předchozích výsledků, tak aby týmy byly vyvážené.

V případě, že se uživatel nemůže turnaje zúčastnit, má možnost soutěž vyexportovat do souboru a předat někomu dalšímu, kdo bude pokračovat ve vedení statistik. Aby se i ostatní uživatelé mohli kdykoliv podívat na své statistiky, měla by aplikace umožnit nahrát soutěž na synchronizační webový server, kde budou detaily soutěže neustále k nahlédnutí.

2.1.2 Instalace a první spuštění

Aplikace pro OS Android se instaluje pomocí souborů s příponou `apk`, které jsou celkem tři. Prvním je soubor s jádrem aplikace `TournamentManager.apk`, dalším je soubor s modulem aplikace `TM Hockey.apk` a posledním je soubor s modulem aplikace `TM Squash.apk`.

Nejdříve je instalován hlavní soubor s jádrem aplikace. Aplikace se zobrazí jako běžná spustitelná aplikace v hlavní nabídce aplikací zařízení. Po spuštění takto nainstalované aplikace se zobrazí obrazovka s nadpisem „Soutěže“, ve které je uživatel informován, že nebyl nalezen žádný sport, a má pokračovat instalací modulu. Aplikace v tuto chvíli, kdy není nainstalován žádný modul, umožňuje pouze vytváření hráčů v aplikaci. Ke každému hráči je třeba zadat jméno a email, jinak nelze hráče vytvořit. K hráči je možné zadat ještě poznámku. Hráč tedy není závislý na nainstalovaném modulu a je spravován jádrem aplikace.

Po doinstalování druhého souboru, s názvem `TM Hockey.apk`, se vizuálně v hlavní nabídce aplikací zařízení nic nezmění. Až v nastavení zařízení a seznamu nainstalovaných aplikací je možné vidět obě nainstalované aplikace. Aplikaci „TM Hockey“ tedy není možné spustit samostatně, je možné ji ale samostatně nainstalovat.

Po spuštění aplikace s nainstalovaným modulem `TM Hockey.apk` se opět zobrazí obrazovka s nadpisem „Soutěže“, tentokrát už obsahující dvě záložky - hokej a florbal.

Po doinstalování posledního souboru, s názvem`TM Squash.apk`, se opět pro uživatele v zařízení nic nezměnilo a aplikace „TM Squash“ je viditelná pouze v nastavení zařízení, v seznamu nainstalovaných aplikací.

Spuštěním hlavní aplikace Tournament Manager se opět uživatel dostane na obrazovku s nadpisem „Soutěže“ a nyní už je vidět v záložce dalších pět sportů - badminton, plážový volejbal, squash, tenis a volejbal.

Uživatel už také v nastavení aplikace vidí seznam přístupných sportů, které má možnost jednotlivě vypnout či zapnout. Také má zpřístupněnou možnost v obrazovce "Soutěže"vytvářet soutěže.

Následuje výčet funkčních a nefunkčních požadavků plynoucích z této části analýzy.

F Hráči Aplikace eviduje hráče. U hráče eviduje jméno, email a poznámku. Hráče bude možné upravit a smazat. Smazat hráče nepůjde, pokud už bude v nějaké soutěži.

F Řazení hráčů Aplikace umožňuje řadit hráče.

F Sporty Aplikace bude umožňovat vypínat a zapínat jednotlivé sporty. Vypnuté sporty nebude zobrazovat v přehledu soutěží.

NF Architektura Aplikace by měla mít modulární architekturu. Aplikace by pro uživatele měla vypadat jako jeden celek.

NF Rozšiřitelnost Aplikace by měla být rozšiřitelná pomocí samostatně instalovatelných modulů, které do aplikace dodají sady sportů. Moduly by ale neměly být samostatně spustitelné.

2.1.3 Soutěže

Uživatel v aplikaci může vytvářet soutěže. Soutěž je možné vytvořit pomocí formuláře nebo importem ze souboru. K soutěži je evidován název, datum začátku a konce, poznámka, u některých sportů také typ soutěže - jednotlivci a týmy.

Přechodem do detailu soutěže zmizí v aplikaci hlavní menu a nahradí ho šipka vlevo, znamenající přechod zpět. To je známkou přechodu do modulové aplikace. Známkou přechodu je i jiná animace během přechodu mezi obrazovkami. Po minimalizaci aplikace je v seznamu spuštěných aplikací systému Android vidět pouze hlavní aplikace „Tournament Manager“. Znovu spuštěním této aplikace se aplikace vrátí tam, kde byla ukončena, tedy do modulové části aplikace. Pomocí speciální aplikace pro prohlížení spuštěných procesů je pak vidět, že je spuštěna jádrová i modulová aplikace. Aplikace tedy pro běžného uživatele vypadá celistvě, jako jedna aplikace.

Soutěž, evidovaná v modulové aplikaci, je zobrazena v seznamu soutěží v části jádra a aplikace jádra umožňuje přejít do modulové aplikace. V seznamu soutěží je na soutěži také možné vyvolat kontextovou nabídku, kde je možné přejít do editace soutěže, smazat soutěž a exportovat soutěž. V Aplikaci tedy existuje obousměrná komunikace mezi jádrem aplikace a modulem.

Na obrazovce detailu soutěže je možné přidat hráče do soutěže. Hráči jsou přidáni výběrem ze seznamu všech hráčů v zařízení. U hráčů je v detailu soutěže vidět tabulka se statistikami. Dále je v detailu soutěže k vidění seznam turnajů, které je možné řadit a přidávat. Soutěž je také možné editovat, avšak už není možné změnit typ soutěže.

Z tabulky hráčů je možné se dostat na detail hráče, který je spravován jádrem aplikace. To jde poznat podle menu v levém horním rohu obrazovky. V detailu hráče je možné vidět seznam jeho soutěží a agregované statistiky ze všech jeho soutěží.

V této části analýzy byly zaznamenány následující funkční a nefunkční požadavky.

F Soutěže Aplikace musí evidovat soutěže. U soutěže eviduje její název, datum začátku a konce, typ soutěže a poznámku k soutěži. Soutěž bude možné editovat, ale už nebude možné změnit její typ. Soutěž bude možné mazat, pouze pokud nebude mít žádné turnaje a hráče.

F Hráči v soutěži Aplikace bude evidovat hráče v soutěži. Do soutěže může být přidán hráč, který je evidován v aplikaci. Mazat hráče ze soutěže půjde pouze pokud se nebude účastnit žádného turnaje.

F Import a Export soutěže Aplikace umožňuje importovat a exportovat soutěž z a do souboru.

NF Jednotný vzhled Aplikace by měla vždy vypadat jako jeden celek a to i po přechodu do modulu.

Byly také zaznamenány následující funkční požadavky na architekturu aplikace.

FS Komunikace částí architektury Jádro aplikace a modul spolu musí být schopni navzájem komunikovat.

FS Povinnosti jádra Jádro musí modulu umožnit zobrazit detail hráče a poskytnout seznam hráčů, které eviduje.

FS Povinnosti modulu - Soutěže Modul musí umožňovat zobrazení detailu soutěže, vyvolání editace či smazání soutěže ve sportu. Dále modul musí poskytovat jádru seznam soutěží pro zobrazení jejich přehledu.

FS Povinnosti modulu - Hráči Modul musí jádru poskytnout agregované statistiky hráče pro sport, aby je jádro mohlo zobrazit v detailu hráče.

2.1.4 Turnaje

V této části aplikace už se jedná vždy o práci uvnitř modulu a s jádrem aplikace už komunikace neprobíhá. Následuje už pouze seznam zjištěných funkčních požadavků.

F Turnaje Aplikace bude umožňovat vytvářet turnaje v soutěži. U turnaje bude evidovat název, datum začátku a konce, poznámku. Turnaj bude možné editovat a mazat. Nebude možné mazat turnaj, který má nějaký tým nebo zápas.

F Bodový zisk U turnaje by aplikace měla umožňovat nastavit bodový zisk - tj. přidělení počtu bodů za výsledek zápasů. Každý turnaj by měl mít nastaven výchozí bodový zisk.

F Hráči v turnaji Do turnaje by mělo být možné přidat hráče, kteří jsou evidováni v soutěži. Bude možné hráče i odebírat, ale pouze pokud už nejsou v nějaké soupisce.

F Týmy v turnaji Do turnaje by mělo být možné evidovat týmy. Každý tým má název. Týmy bude možné evidovat, pouze pokud je turnaj součástí týmové soutěže.

F Soupisky V aplikaci bude možné ke každému týmu přidat hráče a tvořit tak soupisky. Členy týmu se mohou stát pouze hráči v turnaji, kteří ještě nejsou v žádném jiném týmu. Soupisky bude možné kdykoliv měnit, ale změny se projeví pouze u nedohraných zápasů.

F Generování soupisek Aplikace by měla umožňovat vygenerovat složení týmů podle různých kritérií (např. náhodně nebo vyváženě, podle statistik hráčů).

2.1.5 Zápasy

Zápasy jsou evidovány uvnitř turnaje a detail zápasu je odlišný v každém sportu. Nejsou zde žádné zajímavé funkční požadavky na systém či jiné nefunkční požadavky. Následuje tedy rovnou seznam zjištěných funkčních požadavků.

F Zápasy Aplikace by měla umožňovat evidovat zápasy v turnaji. Každý zápas bude mít alespoň dva účastníky a jejich výsledné skóre. Zápas může být dohraný nebo nedohraný.

F Soupisky v zápasu U každého zápasu jsou evidovány soupisky jeho účastníků. U dohraného zápasu už se soupisky účastníků nemění. U nedohraného zápasu soupisky reflektují aktuální stav soupisek týmů. V rámci každého zápasu je možné do soupisek dodatečně přidat hráče, který není v soupisce ani jednoho účastníka.

F Rozdělení zápasů Zápasy by měly být rozděleny do kol a period. Kdy každé kolo obsahuje několik period a každá perioda obsahuje několik zápasů. V rámci jednoho kola by měl hrát každý s každým a v rámci jedné periody by měly být do zápasů rozděleny všechny týmy.

F Generování zápasů Aplikace by měla umožňovat generovat celá kola zápasů, tak, aby hrál každý s každým.

F Statistiky Statistiky budou evidovány pro jednotlivé hráče, ale i pro týmy jakožto účastníky zápasů. Týmové statistiky pak platí pro všechny členy týmu. Statistiky se budou počítat pouze z dohraných zápasů.

F Řazení statistik Hráče a týmy bude možné řadit podle jejich agregovaných statistik na úrovni turnaje i soutěže.

2.1.6 Požadavky na aplikaci pro OS Windows 10 Mobile

Závěrem zbývá doplnit požadavky týkající se systému Windows 10 Mobile. Jsou to následující nefunkční požadavky.

NF Lokalizace Aplikace by měla být snadno lokalizovatelná a měla by podporovat alespoň češtinu a angličtinu.

NF Připojení k internetu Aplikace bude pro účely synchronizace vyžadovat připojení k internetu.

NF Verze OS Windows 10 Mobile Aplikace by měla fungovat na verzi 10.0.10586 Windows 10 Mobile, s kterou byla vydána první zařízení s tímto systémem.[5]

2.2 Rešerše existujících aplikací pro OS Windows 10 Mobile

Tato sekce obsahuje rešerši existujících aplikací pro OS Windows 10 Mobile.

Po hledání aplikací v aplikaci Store App, skrze kterou se aplikace instalují, podle klíčových slov jako jsou „Sport“, „Tournament“, „Manager“ a dalších byly nalezeny pouze dvě aplikace, částečně vyhovující funkčním požadavkům na aplikaci. Je to pravděpodobně způsobeno tím, že OS Windows 10 Mobile je nový a poměrně málo rozšířený systém, takže množství aplikací není tak velké jako u OS Android.

Součástí rešerše jsou tedy dvě aplikace pracující se sportovními statistikami.

2.2.1 Kritéria hodnocení

U zkoumaných aplikací bude vyhodnocena míra splnění funkčních požadavků, kdy budou upřednostněny zejména požadavky na evidenci různých sportů, turnajů, zápasů a statistik.

Stejnou měrou bude rešerše také zaměřena na zkoumání uživatelského rozhraní a organizaci obsahu. Výsledné poznatky rešerše aplikací z této oblasti tak budou moci být využity k porovnání s uživatelským rozhraní aplikace pro OS Android a dalšímu návrhu designového řešení uživatelského rozhraní v kapitole o návrhu aplikace.

2.2.2 MSN Sport

První aplikací je aplikace MSN Sport od vydavatele Microsoft. Aplikace slouží k sledování přehledů statistik sportů, týmů a hráčů a je tak zajímavá zejména kvůli uživatelskému rozhraní. Uživatel si vybere své oblíbené sporty, které chce sledovat a k těm poté vidí přehled chystaných zápasů a turnajů, důležité statistiky a informace.

2.2.2.1 Uživatelské rozhraní a funkčnost

Uživatelské rozhraní aplikace je složeno z několika částí.

Obrázek 2.1: Ukázka aplikace MSN Sport.

První částí je levé hlavní menu s typickým menu tlačítkem. Obsahem levého menu je první úroveň obsahu aplikace, což jsou v tomto případě vybrané sporty, které chce uživatel sledovat.

Druhou částí, nacházející se u horního okraje obrazovky, je panel s nadpisem obsahující název aktuální stránky a tlačítko pro vyhledávání. Názvem aktuální stránky je například název týmu či jméno hráče.

Třetí částí, která se nachází pod nadpisem je pak panel s tzv. taby - tedy záložkami, které slouží jako rozcestník další úrovně obsahu. Aktivní záložka je označena silnějším stylem písma a pokud je záložek více, než kolik se vejde na šířku obrazovky, jsou záložky rotovány.

Hlavní a největší částí uživatelského rozhraní je pak samotný obsah. Základem obsahu jsou vždy velké, dobře čitelné a formátované informace doplněné ikonami a obrázky.

Poslední částí je pak panel, nacházející se u spodní hrany obrazovky. Na tomto panelu se nachází akce pro práci s obsahem ve formě tlačítek s ikonami.

Nejčastějším typem obsahu jsou seznamy. Ty jsou řešeny formou karet s výraznými nadpisy a několika dodatečnými informacemi.

Otevřením detailu karty dojde ke změně nadpisu, obsahu záložek a obsahu hlavní části uživatelského rozhraní. Hlavní menu zůstává stejné a slouží tak jako pevný bod aplikace.

Pro navigaci zpět slouží hardwarové tlačítko zpět a je jediným způsobem jak se lze vrátit na předchozí úroveň.

2.2.2.2 Zhodnocení

Aplikace díky dobrému formátování obsahu, velkému množství ikon a obrázků působí příjemně. Občas se však uživatel může cítit ztracený, obzvlášť, když se

Obrázek 2.2: Ukázka aplikace TourneyMan.

zanoří do nižších úrovní obsahu. Postupně se totiž dá přecházet mezi týmy, hráči, zápasy a po několika přechodech uživatel může ztratit přehled, kde se nachází. Jinak ale uživatelské rozhraní a struktura celkem odpovídá struktuře uživatelského rozhraní aplikace pro OS Android.

Z funkčního hlediska aplikace MSN Sport nesplňuje požadavky na evidenci vlastních aktivit jako jsou turnaje a statistiky a proto je tedy jako alternativa nevyhovující.

Celkově je aplikace zpracovaná dobře a může sloužit jako inspirace pro návrh designu uživatelského rozhraní, jelikož příkladně splňuje požadavky na vzhled aplikací pro OS Windows 10 Mobile, což je způsobeno tím, že se jedná o aplikaci vydanou tvůrcem operačního systému.

2.2.3 TourneyMan

Druhou aplikací, kterou je možné zařadit do této rešerše, je aplikace TourneyMan. Aplikace slouží k velmi jednoduché evidenci statistik. Uživatel v aplikaci založí turnaj a eviduje do něj hráče. Poté vygeneruje zápasy a zaznamená kdo vyhrál.

2.2.3.1 Uživatelské rozhraní a funkčnost

Aplikace má vzhledem k malému množství funkcí jednoduché uživatelské rozhraní. Rozhraní se skládá z horního panelu, který obsahuje seznam záložek. Záložky jsou zde velmi velké a přehledné. Pod horním panelem se pak nachází přímo hlavní obsah. Obsahovou část aplikace tvoří zejména seznamy, které jsou podobně jako v aplikaci MSN Sport řešeny formou karet. V dolní části obrazovky se pak nachází ovládací panel s akcemi. Tlačítka akcí jsou oproti aplikaci MSN Sport zarovnané na střed a ikonky jsou v kroužku.

Aplikaci tvoří jen jedna úroveň obsahu rozčleněná do záložek. Pohyb v aplikaci je tak velmi přehledný a jednoduchý.

2.2.3.2 Zhodnocení

Na aplikaci je vidět, že je vytvořena pro starší verzi systému Windows Phone, takže inspiraci pro design uživatelského rozhraní zde hledat nelze. U této aplikace by do budoucna mohlo hrozit i její stažení ze Store App, v případě, že by se společnost Microsoft rozhodla tvrději prosazovat doporučené designové postupy. Případný další rozvoj aplikace je vzhledem k jejímu zastarání rovněž nejistý. Z tohoto hlediska je tedy nevyhovující.

Z funkčního hlediska aplikace splňuje pouze požadavky na evidenci zápasů, generování zápasů podle účastníků turnaje a evidenci jedné statistiky - skóre. Nesplňuje však požadavek na evidenci různých sportů, sledování podrobnějších hráčských statistik, evidenci hráčů a týmů. Aplikace tedy částečně vyhovuje funkčním požadavků, avšak ne v plném rozsahu. Celkově je tedy aplikace jako alternativa nevyhovující.

2.2.4 Vyhodnocení rešerše

Ani jedna z aplikací v této rešerši nesplňuje všechny funkční požadavky na aplikaci a jako případné alternativy jsou tedy nevyhovující.

Vzhledem k nedostatku aplikací se sportovní tématikou pro systém OS Windows 10 a neexistující alternativní aplikaci, má vytvoření nové aplikace rozhodně smysl.

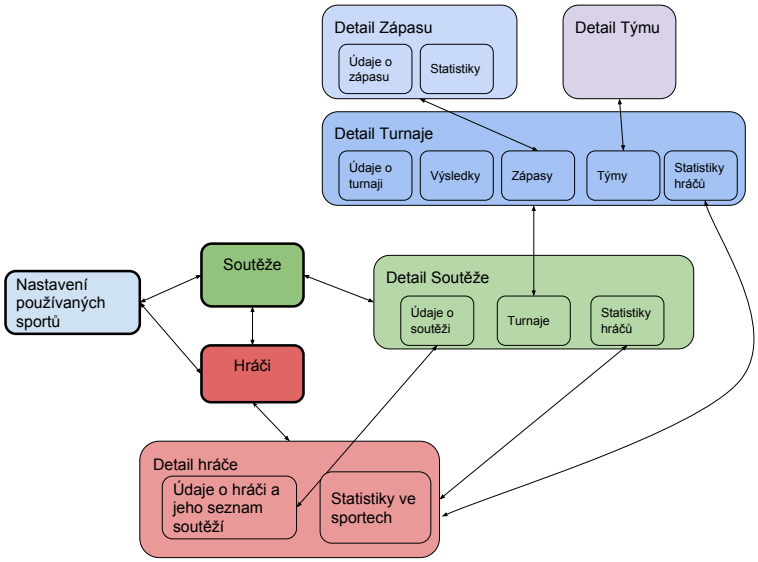
2.3 Návrh uživatelského rozhraní aplikace

V této sekci je popsán návrh uživatelského rozhraní aplikace pro OS Windows 10 Mobile.

Návrh vychází z funkčních požadavků, předchozí rešerše existujících aplikací a z uživatelského rozhraní pro systém OS Android.

Tato část návrhu uživatelského rozhraní se zabývá rozdělením rozhraní do obrazovek, jejich propojením, a informacemi, které mají zobrazovat. Tento návrh se tedy od aplikace pro OS Android příliš lišit nebude a proto zde bude pouze krátce shrnut, aby měl čtenář povědomí o struktuře aplikace. Detailnější popis tohoto návrhu je k vidění v práci Josefa Němečka[1].

Aplikace pro operační systém OS Android na rozdíl od aplikace MSN Sport, která byla popsána v rešerši výše, nabízí hlavní úroveň menu pouze v jádře aplikace a nikoliv už pak v modulech. Jak plyne z rešerše, v aplikacích pro OS Windows 10 Mobile je zvykem mít přístupné toto menu všude a tak také bude navrženo uživatelské rozhraní pro tuto aplikaci. Obrazovky dostupné z tohoto menu jsou na diagramu 2.3 vyznačeny silnějším okrajem.



Obrázek 2.3: Diagram přechodů obrazovek.

Pro navigaci po aplikaci bude využito dotykového ovládání pro přechody směrem vpřed a tlačítka zpět pro přechody směrem vzad. Tlačítko zpět může být řešeno hardwarově či softwarově, ale je vždy dostupné.

Následuje krátký popis jednotlivých obrazovek a diagram jejich přechodů, k vidění na obrázku 2.3.

2.3.1 Nastavení používaných sportů

První obrazovkou, kterou uživatel uvidí po prvním spuštění bude obrazovka nastavení používaných sportů. Zde budou uživateli zobrazeny dostupné sporty v aplikaci a uživatel bude moct zapnout ty sporty, které bude chtít používat. Potvrzením vybraných sportů uživatel přejde na přehled soutěží. Tato obrazovka bude vždy dostupná z hlavního menu aplikace.

2.3.2 Soutěže

Přehled soutěží je hlavní obrazovkou aplikace a bude první obrazovkou, kterou uživatel uvidí při každém dalším spuštění aplikace. Bude tedy rovněž dostupná z hlavního menu. Na této obrazovce bude vidět seznam sportů a jejich soutěží. Uživatel zde bude moci vytvořit, editovat , mazat a řadit soutěže. Vybráním jedné ze soutěží uživatel přejde na detail soutěže. Ke každé soutěži bude zobrazen její název a datum konání. Přechodem na tuto obrazovku bude

vždy vymazána historie procházení a uživatel se tak vždy bude moci vrátit do výchozího stavu.

2.3.3 Hráči

Další obrazovkou, dostupnou z hlavního menu bude obrazovka Hráčů. Zde bude seznam hráčů registrovaných v zařízení. Uživatel zde bude moci vytvářet, editovat, mazat a řadit hráče, tak jak je to definováno ve funkčních požadavcích. Vybráním jednoho z hráčů v seznamu uživatel přejde do obrazovky detailu hráče.

2.3.4 Detail hráče

Na obrazovce detailu hráče budou dvě záložky.

Údaje o hráči Na této záložce budou zobrazeny informace o hráči jako je jeho email či přehled odehraných soutěží v jednotlivých sportech. Do každé soutěže, které se hráč účastnil bude možné přejít.

Statistiky ze sportů Na této záložce bude možné vidět souhrnné statistiky hráče za každý sport v aplikaci.

2.3.5 Detail soutěže

Na obrazovku detailu soutěže bude hráč přesměrován po vybrání soutěže v seznamu soutěží. Obrazovka soutěže se bude skládat z několika záložek.

Údaje o soutěži Tato záložka bude zobrazovat základní informace o soutěži jako je místo konání a poznámka, počet turnajů a tak dále. Z této záložky bude možné přejít na editaci soutěže a také bude možné vyvolat export soutěže do souboru.

Turnaje Záložka s turnaji bude zobrazovat seznam turnajů v soutěži. Na této záložce bude možné turnaje přidávat, editovat, odebírat, řadit a přecházet na jejich detail. Ke každému turnaji bude zobrazen jeho název a datum konání.

Hráči Záložka s hráči bude obsahovat tabulku, která bude sloužit jako seznam hráčů v soutěži a zároveň bude zobrazovat statistiky těchto hráčů. Hráče bude možné přidávat, odebírat a řadit podle statistik. Vybráním některého z hráčů bude možné přejít na na jeho detail.

2.3.6 Detail turnaje

Na detail turnaje bude uživatel přesměrován po vybrání jednoho z turnajů v seznamu turnajů na obrazovce detailu soutěže. Obrazovka detailu turnaje bude rozdělena do záložek, podobně jako obrazovka detailu soutěže.

Údaje o turnaji Na této záložce budou zobrazeny údaje o turnaji jako je datum konání, počet zápasů, počet účastníků a tak dále. Uživatel na této záložce bude moci přejít na editaci turnaje či na editaci bodového zisku turnaje.

Výsledky Na záložce s výsledky bude zobrazena tabulka účastníků turnaje a jejich statistik. Ve výchozím stavu budou seřazeni podle statistiky, která určuje celkové pořadí účastníků od nejlepšího po nejhoršího, ale účastníky bude možné řadit i podle ostatních statistik.

Zápasy Záložka se zápasy bude obsahovat seznam zápasů v turnaji. Uživatel bude moci zápasy přidávat, generovat, editovat, mazat a resetovat. Vybráním jednoho zápasu uživatel bude přesměrován na zobrazení jeho detailu.

Týmy Záložka s týmy bude zobrazovat seznam týmů a jejich členů. Na této záložce bude možné týmy vytvářet, mazat a generovat jejich složení. Vybráním jednoho z týmů bude možné přejít do jeho detailu.

Hráči Záložka s hráči bude zobrazovat tabulku, sloužící jako seznam hráčů v turnaji. Tabulka bude zobrazovat agregované statistiky hráčů z jejich odehraných zápasů. V této tabulce bude možné hráče řadit a bude možné je přidávat a odebírat. Vybráním hráče bude možné přejít na detail hráče.

2.3.7 Detail týmu

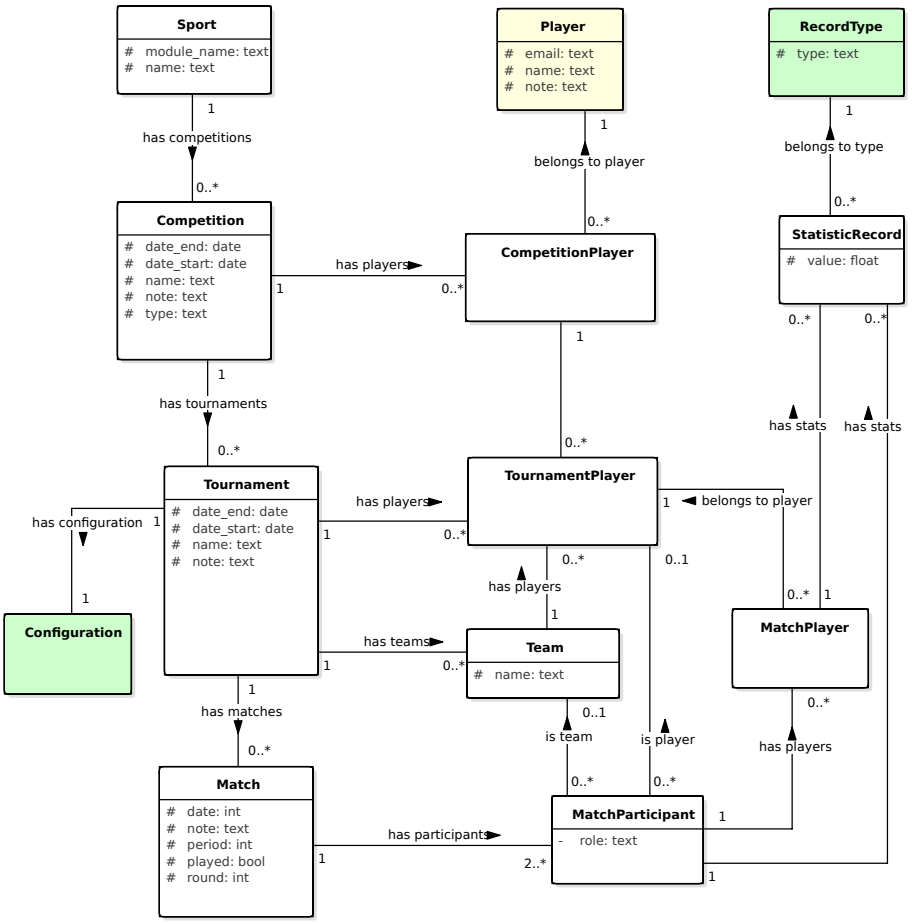
V obrazovce detailu týmu bude uživateli umožněno měnit složení týmu přidáním nebo odebráním jeho členů. Toto je konečná obrazovka, z které už nelze přejít na další.

2.3.8 Detail zápasu

Na obrazovce detailu zápasu budou celkem dvě záložky.

Údaje o zápasu Na této záložce budou zobrazeny údaje o zápasu jako je například skóre nebo jiný stav zápasu. Tyto údaje bude možné pro rychlejší ovládání rovnou měnit. Údaje bude možné na této stránce také uložit, čímž dojde k označení zápasu jako odehraného.

Statistiky Na této záložce budou zobrazeny statistiky hráčů jednotlivých účastníků formou tabulky. Jednotlivé řádky tabulek bude možné editovat samostatně a nebo všechny najednou, pro rychlejší zadávání. Také zde bude možné přidávat či odebírat hráče z jednotlivých účastníků.



Obrázek 2.4: Doménový model.

2.4 Doménový model

Na závěr analytické části práce bude popsán a vysvětlen doménový model aplikace.

Doménový model byl navržen společnými silami všech členů týmového projektu a zobrazuje entity, jejich vlastnosti a vztahy, s kterými aplikace pracuje. Model vychází z funkčních požadavků aplikace a požadavků zadavatele na způsob, jakým bude aplikace používána.

Tento model pak bude hlavním opěrným bodem při návrhu aplikace, ale je také důležitý pro čtenáře, aby získal povědomí o kontextu obsahu aplikace.

Sport Entita sportu představuje různé sporty, které uživatel aplikace provozuje a pro které budou evidovány jednotlivé soutěže, kterých se účastní.

Sporty budou v aplikaci rozděleny do modulů, jelikož mají každý svá

specifika. Některé sporty si jsou však z hlediska funkčních požadavků aplikace velmi podobné a v podstatě se liší pouze názvem a několika maličkostmi. Výroba separátních modulů pro tyto podobné sporty by byla zbytečná a proto bude možné mít v jednom modulu více podobných sportů.

Player Entita hráče představuje osobu, s kterou uživatel aplikace někdy hrál nebo bude hrát nějakou soutěž.

Každý uživatel evidovaný v zařízení má jméno, email a poznámku. Email je nutný, kvůli jednoznačné identifikaci hráče napříč různými soutěžemi pro případ sdílení soutěží mezi více uživateli.

Na obrázku je tato entita zvýrazněna žlutě a to znamená, že je spravována jádrem aplikace.

Competition Entita soutěže představuje dlouhodobější událost ve sportu. Tato událost se skládá z několika dílčích událostí, z kterých se shromažďují a vyhodnocují statistiky hráčů. Například v hokeji by se mohlo jednat o jednu ligu.

Soutěž má datum začátku a konce, název a typ. Typ soutěže je většinou jednotlivci nebo týmy, není ale omezen pouze na tyto dva a každý modul si může přidat vlastní typy soutěže. Soutěž pak může mít několik turnajů, což jsou dílčí události, zmíněné výše a několik hráčů, kteří se jí účastní. Musí to být ale výběr z hráčů, evidovaných v zařízení.

CompetitionPlayer Entita hráče v soutěži představuje osobu, která se účastní dané soutěže a už je evidována v zařízení uživatele. Z obrázku je patrné, že jeden hráč se může účastnit více soutěží.

Tournament Entita turnaje představuje událost ve sportu, ve které se účastníci postupně utkávají mezi sebou. Na konci této události je vyhodnoceno pořadí účastníků podle výsledků jejich klání. Vezmeme-li jako příklad opět hokej a soutěž by byla jedna liga, turnaj by pak mohla být jedna sezóna této ligy.

Turnaj má, podobně jako soutěž, název, datum začátku a konce a také poznámku. Každý turnaj má konfiguraci bodů. Dále turnaj může mít hráče, kteří jsou registrovaní v soutěži turnaje. Pokud se jedná o týmovou soutěž, může mít turnaj týmy. Turnaj také může mít několik zápasů.

TournamentPlayer Entita hráče v turnaji představuje hráče, který se účastní turnaje a zároveň je účastníkem soutěže, ve které se turnaj koná.

Hráč, který se účastní turnaje se pak může účastnit zápasů, nebo být součástí týmů, pokud se jedná o týmovou soutěž. Pokud se hráč účastní zápasu, budou k němu také evidovány statistiky ať už se bude zápasu účastnit jako jednotlivec, nebo jako člen týmu.

Team Entita týmu představuje skupinu hráčů, kteří hrají spolu v daném turnaji. Skrze entitu účastníka zápasu pak týmy mohou mít také vlastní statistiky.

Každý tým má svůj název a hráče, kteří jsou jeho členy. Týmy jsou vytvořeny pro každý turnaj samostatně a nejsou přenášeny mezi jednotlivými turnaji.

Configuration Entita konfigurace turnaje představuje nastavení, podle kterého se hráčům, potažmo týmům, v rámci daného turnaje přidělují počty bodů za jejich výsledky. Podle těchto bodů se pak může určovat pořadí účastníků či hodnoty dalších statistik.

Jak plyne z násobnosti entit na obrázku, každý turnaj by měl mít alespoň jednu konfiguraci bodů.

Na obrázku je také vidět, že tato entita je zvýrazněna zeleně. To znamená, že její vlastnosti záleží na každém sportu a modulu a nejsou předem definované. Entita je tedy definovaná až v modulu.

Match Entita zápasu představuje událost ve sportu, ve které se mezi sebou utkají typicky dva účastníci turnaje. Výsledkem této události je pak určitý výsledek, většinou formou bodového skóre a statistiky účastníků, které byly tvořeny během klání těchto dvou účastníků.

Každý zápas by měl mít alespoň dva účastníky a také může mít další informativní vlastnosti o jeho průběhu či výsledku. Ty už jsou však různé pro každý sport a musí být řešeny separátně a ne v obecném doménovém modelu aplikace.

Participant Entita účastníka zápasu představuje buď tým nebo hráče (záleží na typu soutěže), který se účastnil zápasu. Jedná se tedy o vazební entitu mezi zápasem a jeho účastníky.

Každý účastník má svou roli v zápase, což většinou bývá strana domácích nebo strana hostů. Účastník, například celý tým, pak také může mít týmové statistiky za zápas.

MatchPlayer Entita hráče v zápase slouží jako záznam o tom, který hráč hrál v zápase za kterého účastníka. Entita je evidována jak pro hráče, který se účastní zápasu jako jednotlivec v soutěži jednotlivců, ale i pro všechny hráče v týmu, kteří se účastní jako členové týmu v týmové soutěži.

Entita je vytvořena a navázána na účastníka zápasu v momentě, kdy je dohrán zápas a dojde k evidenci statistik. Tím je dosaženo toho, že i když se v budoucnu změní složení hráčů v týmu, v odehraných zápasech se již složení týmů nezmění, jelikož je zaznamenáno touto entitou.

Entita je tedy vazební entitou, navázanou na hráče v turnaji a na sadu statistik, které mu byly po dohrání zápasu zaznamenány.

StatisticRecord Entita statistického záznamu představuje jednu statistickou hodnotu, která je připsána hráči či účastníkovi zápasu.

Každý účastník či hráč v zápasu může mít několik zaznamenaných statistik za jeden zápas. Každý statistický záznam má ještě určen typ následující entitou.

RecordType Entita typu statistického záznamu představuje název pro hodnotu statistického záznamu a určuje tím typ tohoto záznamu.

Na obrázku je entita vyznačena zelenou barvou. Je tak vyznačena ze stejného důvodu, jako nastavení turnaje. Tedy, že se jedná o entitu, která je různá pro každý sport, a je detailně definována až v modulu.

Kapitola 3

Návrh

V úvodu této kapitoly proběhne seznámení se s vývojem pro OS Windows 10 Mobile. V této kapitole bude následně popsána architektura celé platformy, kde budou znázorněny vazby jednotlivých komponent platformy. Dále bude popsán návrh architektury aplikace jako celku a návrh jednotlivých částí této architektury. Poté bude popsán návrh vzorového modulu.

3.1 Seznámení se s vývojem pro OS Windows 10 Mobile

S uvedením operačních systémů Windows 10 a Windows 10 Mobile pro mobilní zařízení v roce 2015 společnost Microsoft představila Univerzální Platformu Windows. Jak již plyne z názvu, jejím cílem je umožnit vytváření aplikací, které lze spustit na různých typech zařízení podporujících systémy Windows 10 bez nutnosti výrazně přizpůsobovat tyto aplikace jednotlivým typům zařízení.

Je tak možné vytvořit jednu aplikaci, kterou je možné nainstalovat na telefon s Windows 10 Mobile, tablet či počítač s Windows 10, ale i exotičtější zařízení jako je např. Microsoft HoloLens a stále bude vypadat a fungovat správně. Tím je uživateli umožněno použít vhodné zařízení pro splnění úkolu, který potřebuje vyřešit, a není omezován operačním systémem, které zařízení používá. [6]

3.1.1 Historie a směr UWP

Společnost Microsoft myšlenku univerzálních aplikací poprvé ukázala v operačním systému pro mobilní zařízení Windows Phone 8.1 a systému pro PC Windows 8, kde byly známé jako Univerzální Aplikace. V těchto aplikacích byl také poprvé představen obchod aplikací.

Tyto univerzální aplikace sice používaly společný kód pro datovou a logickou vrstvu aplikace, ale musely definovat prezentační vrstvu pro mobilní a

desktopový systém zvlášť. Také bylo nutné ověřovat přítomnost API specifických pro zařízení a tomu přizpůsobovat uživatelské rozhraní a funkce. Aplikace se musela kompilovat zvlášť pro mobilní systém a zvlášť pro desktopový systém. Do obchodu s aplikacemi tedy bylo nutné nahrát dvě verze aplikace.[6]

Tato myšlenka byla ve Windows 10 a Windows 10 mobile ještě zdokonalena a přinesla několik nových postupů, prvků a vlastností popsaných v následujících sekcích.

Důležitou a stěžejní funkcí systému Windows 10 Mobile, kterou chce zaujmout zákazníky je funkce Continuum, která umožňuje připojit k mobilnímu telefonu externí monitor, klávesnici a myš a využívat ho jako stolní počítač. Tato funkce plně využívá možností UWP a Microsoft ji vnímá jako klíčovou pro další vývoj OS a mobilních telefonů.[7]

Každých několik měsíců Microsoft UWP aktualizuje. Většinou jsou vylepšeny a optimalizovány stávající funkce a přidány nové funkce, které se týkají propojení mobilních a stolních zařízení.

3.1.2 Vlastnosti UWP

V této sekci bude popsáno několik vlastností a prvků UWP, které řeší neduhy předchozích Univerzálních aplikací pro systémy Windows Phone 8.1 a Windows 8.

3.1.2.1 Distribuce aplikací a cílová zařízení

Aplikace jsou zabaleny a distribuovány v tzv. Appx balíčku. Jedná se o obdobu Apk balíčku u aplikací pro OS Android.

To umožňuje bezpečnou, snadnou instalaci aplikace a její případné aktualizace. Aplikace v tomto formátu jsou nahrány na jednotný obchod aplikací jménem Store, který je společný pro všechny UWP aplikace a všechny typy zařízení. Dalším způsobem jak nahrát aplikaci do zařízení je pouze pomocí vývojářského softwaru VisualStudio či použitím příkazové řádky a scriptů dodaných s Appx balíkem aplikace.[8]

Aplikace pro UWP se tedy nevytváří pro operační systém, ale pro cílovou skupinu zařízení. V případě aplikace Tournament Manager je cílovým zařízením především mobilní telefon.

Vzhledem k možnostem UWP a možnosti pokrytí desktopových a tabletových zařízení využívající standardní OS Windows 10 téměř bez práce, by bylo až trestuhodné tuto možnost nevyužít a znemožnit použití aplikace na těchto zařízeních. Obzvláště, když množství uživatelů standardního systému Windows 10 je násobně větší než množství uživatelů používajících Windows 10 Mobile.

3.1.2.2 Adaptivní uživatelské rozhraní

Důležitým prvkem UWP umožňujícím použití jedné aplikace na různých typech zařízení je Adaptivní uživatelské rozhraní a Efektivní pixel.

Úkolem Efektivního pixelu je zajistit, aby text o jedné velikosti (např. 24 pixelů) vypadal pořád stejně na zařízeních s různou velikostí displaye, různou hustotou pixelů a s různou vzdáleností pozorovatele od displaye. Text tak má vždy správnou velikost a dobře se čte.

Dalším prvkem Adaptivního UI je možnost změnit rozložení UI prvků a jejich vlastností na základě šířky okna aplikace v efektivních pixelech. Tím je umožněno přizpůsobit zobrazení aplikace tak, aby byla vždy efektivně využita veškerá dostupná plocha. Není tak nutné vytvářet celou prezentační vrstvu znovu. I tato možnost je však stále podporována, jelikož je to v některých případech nevyhnutelné.

Adaptivní UI pokrývá nejen výstup a zobrazení aplikace ale také vstup. UI prvky jsou schopné reagovat na všechny možné typy vstupních zařízení od dotykových jako je prst či stylus, přes standardní klávesnici a myš až po ovladače herních konzolí.[6]

3.1.3 Programování a aplikační model

V této části bude popsáno jaké programovací jazyky jsou použity při tvorbě UWP aplikací, jak vypadá typická architektura aplikace, a jaké jsou dostupné frameworky.

3.1.3.1 Programovací jazyky

Při vytváření UWP aplikace se používají zejména dva programovací jazyky - C# a XAML.

C# Jazyk C# je objektově orientovaný typový programovací jazyk pro obecné použití. Byl vyvinut společností Microsoft a používá se v programování .NET Frameworku a tedy v programování aplikací pro systém Windows. Je proto velmi známý a rozšířený a v programování pro UWP je to primární programovací jazyk.[8]

XAML Jazyk XAML je rozšiřitelný značkovací jazyk založený na XML. Rovněž byl vyvinut společností Microsoft a slouží k vytváření prezentačních vrstev aplikací, podobně jako slouží HTML k vytváření webových stránek. V UWP je tedy využit v prezentační vrstvě. [9]

3.1.3.2 Vývojové prostředí

Aplikace pro UWP se stejně jako jiné aplikace pro .NET framework vyvíjí ve vývojovém prostředí Microsoft Visual Studio 2015. Z čehož plyne následující struktura vytvářené aplikace.

Solution Základním prvkem Visual Studia je Solution, což je struktura sloužící k organizování projektů a komponent a představuje vytvářenou aplikaci.

Projekt Projekt ve Visual Studiu představuje spouštěcí aplikaci, knihovnu tříd, knihovnu unit testů nebo jinou součást potřebnou pro správné fungování Solution. Projekty jsou tedy jednotlivé části vytvářené aplikace.

Projekty mezi sebou mají závislosti - tzv. reference a projekt je zkompilován až poté, co jsou zkompilovány jeho reference. Není proto možné, aby na sebe projekty odkazovaly vzájemně, jelikož by nebylo možné určit pořadí kompilace projektů. Na to je potřeba myslet při návrhu struktury aplikace.

3.1.3.3 MVVM

V aplikacích UWP a celkově v aplikacích používajících XAML pro tvorbu uživatelského prostředí je využit návrhový vzor MVVM, tedy Model-View-ViewModel.

Jedná se o vzor, zejména pro zobrazovací část aplikace, podle kterého se aplikace skládá ze tří vrstev.

Model Tato vrstva představuje veškerou logiku aplikace a zajišťuje získání dat. Model může obsahovat další dělení a vrstvy, které pro tento návrhový vzor nejsou důležité.

View Vrstva představující zobrazovací část aplikace. Definuje vzhled a obsah jednotlivých obrazovek. Vrstva `View` by se měla starat pouze o to jak data zobrazit.

ViewModel Vrstva `ViewModel` se nachází mezi vrstvou Model a vrstvou `View` a stará se o transformaci dat z Modelu pro `View` tak, aby měla `View` co nejjednodušší práci se zobrazováním dat.

MVVM také definuje vztahy mezi jednotlivými vrstvami. `View` by měla používat jeden nebo více `ViewModelů`. `ViewModel` nemá o existenci `View` nic vědět a měl by pouze vystavovat ucelená data, příkazy, které umí provést, a případně informovat o změnách a událostech pomocí notifikací a událostí. Dále by pak `ViewModel` měl mít referenci na Model a Model by zase o existenci `ViewModelu` neměl nic vědět.

Pro tento vzor existuje několik frameworků pro usnadnění jeho implementace a pro vývoj aplikace bude vhodné jeden vybrat.

Nejznámějším frameworkem pro MVVM je Prism[2], který vyvíjela společnost Microsoft, ale nedávno byl uvolněn jako Open Source[10]. Prism byl

[2]Knihovna Prism. Dostupná na: https://github.com/PrismLibrary/Prism

vytvořen zejména pro WPF, kde umožňuje i vytváření modulárních aplikací. Jeho verze pro UWP je ale o tyto možnosti ošizena.

Další alternativou k frameworům jsou takzvané toolkity, což jsou spíše sady nástrojů, z kterých si vývojář vybere pouze to co potřebuje. Příkladem může být Cimbalino toolkit[3]. K tomu však není k nalezení mnoho ukázek a dokumentace je pouze formou (ne zrovna dobře popsaným) přehledem tříd a metod.[CITE]

Nejatraktivnější možností a zvoleným vítězem je tak Template10[4], což je kombinace frameworku a toolkitu. Template10 je částečně frameworkem, jelikož upravuje chod aplikace a přidává třídy pro správu chodu aplikace. Zároveň je také toolkitem, protože obsahuje komponenty, které vývojář může využít, ale nemusí. Jsou to například různé ovládací prvky pro prezentační vrstvu či validační mechanizmus pro prezentační i logickou vrstvu a mnoho dalších.

O vývoj a správu Template10 se starají zejména zaměstnanci Microsoftu Jerry Nixon a Derren May, kteří se podílejí na tvorbě pomůcek a návodů pro vývoj v UWP a díky tomu mají i náhled do zákulisní tvorby UWP. Template10 přebírá mnoho prvků z frameworku Prism, má velkou podporu komunity UWP a je velmi aktivně rozvíjen.[11]

3.1.3.4 Databáze v UWP

Pro ukládání dat je pro UWP vyvíjena nová verze EntityFramework, což je velmi často používané ORM pro vývoj aplikací v C#. Ta je však stále ve vývojové fázi a není kompletní.[12] Jako alternativu je však možné využít jednodušší balík pro práci s databází SQLite jménem SQLite.Net-PCL[5].

Knihovna poskytuje třídu, která vytvoří připojení k databázi a následně umožňuje vytváření tabulek podle entit, vkládání, úpravy a mazání entit z databáze a také jednoduché migrace pro aktualizaci tabulek databáze podle entit, které do nich budou ukládány.

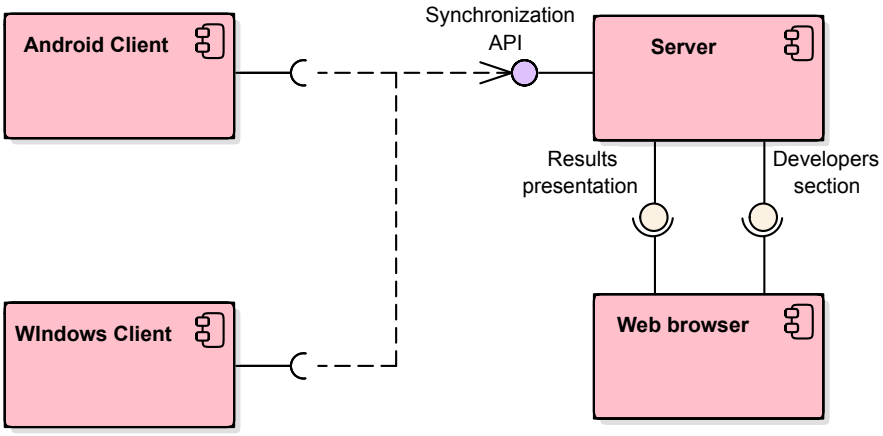
Jelikož tato knihovna poskytuje metody pro komunikaci s databází, kde jako generický parametr slouží typ entity, podle kterého se určí tabulka pro další práci a podle kterého se vytváří instance vracených objektů z databáze, není možné zde využívat interface. V celé aplikaci tak bude nutné pracovat s konkrétními typy entit.

Knihovna má nevýhodu v tom, že neumožňuje definici vztahů mezi entitami, takže bude potřeba tyto závislosti řešit v aplikaci svépomocí.

[3]Knihovna Cimbalino Toolkit. Dostupná na: https://cimbalino.org/

[4]Knihovna Template10. Dostupná na: https://github.com/Windows-XAML/Template10/

[5]Knihovna SQLite.Net-PCL. Dostupná na: https://github.com/oysteinkrog/SQLite.Net-PCL



Obrázek 3.1: Architektura platformy.

3.2 Struktura platformy

V této části návrhu je zachycena struktura celé platformy, kde je vidět, jaké mezi sebou mají vazby jednotlivé části platformy Tournament Manager.

Jak již víme z úvodu, platforma má tři hlavní části:

- klientská aplikace Tournament Manager pro OS Android.
- synchronizační server s webovou prezentací,
- klientská aplikace Tournament Manager pro OS Windows 10 Mobile.

Na obrázku 3.1 je vidět, že klientské aplikace jsou propojené se synchronizačním serverem, který je propojen s prezentačním webovým serverem. Synchronizační server slouží k synchronizaci a sdílení dat mezi klientskými zařízeními.

Aby aplikace mohly soutěže sdílet, musí spolu být struktury jejich sportů a soutěží vzájemně kompatibilní. To znamená, že struktura exportované soutěže ze zdrojového sportu musí být převoditelná na strukturu soutěže cílového sportu. Tato společná převoditelná struktura je definována synchronizačním serverem, který definuje strukturu kontejnerů, do kterých má být soutěž převedena. Tyto kontejnery jsou však definovány takovým způsobem, aby umožnily jistou volnost v struktuře dat pro všechny možné požadavky sportů. Je tedy ještě nutné, aby jednotlivé sporty mezi sebou měly dohodnutou detailnější strukturu, kterou budou moci oba vytvořit a přijmout.

Synchronizační server je pak schopen zobrazit synchronizované soutěže pomocí webového prezentačního serveru. K tomu je ještě zapotřebí, aby tvůrce aplikace na serveru nastavil, jakým způsobem se mají synchronizovaná data převést do prezentované webové stránky. Tyto principy a popisy jsou podrob-

něji popsány v diplomové práci Michala Hacury[4], která se zabývá tvorbou tohoto synchronizačního a webového prezentačního serveru.

3.3 Struktura aplikace

V této sekci bude popsán návrh architektury aplikace.

Navrhovaná architektura bude architektura modulární. To je architektura, umožňující rozdělení aplikace na oddělitelné části, pomocí kterých lze výslednou aplikaci skládat. Výhodou modulární architektury je zejména možnost nezávislého a odděleného vývoje aplikace více vývojáři. To jsou, společně s nefunkčními požadavky z analýzy aplikace pro OS Android, hlavní požadavky na architekturu aplikace.

Před samotným návrhem bude popsán způsob a fungování modulární architektury aplikace pro OS Android, následně bude proveden průzkum možností a omezení UWP v oblasti modularity. Na základě zjištěných možností a omezení bude proveden návrh architektury tak, aby splňovala co nejvíce výše zmíněných požadavků. V každé části návrhu bude dodržena snaha o co největší přiblížení-se aplikaci pro OS Android.

3.3.1 Řešení modulární architektury v OS Android

Modulární architektura v aplikaci pro OS Android je řešena pomocí více propojených aplikací. Operační systém Android umožňuje nastavit aplikace jako součást hlavní aplikace. Tyto pod-aplikace jsou pak spouštěny v rámci hlavní aplikace a v přehledu spuštěných aplikací a se tváří jako jedna hlavní aplikace. V seznamu nainstalovaných aplikací lze tyto pod-aplikace nalézt po otevření záložky s hlavní aplikací.

Tyto aplikace lze tedy vnímat jako moduly a je možné je instalovat a mazat ze zařízení dle potřeby. Každá pod-aplikace má vlastní manifest, což je soubor s definicí vlastností aplikace, ve kterém je určeno, která aplikace je hlavní aplikací.

Spuštění modulu se vyvolá příkazem z hlavní aplikace, který je rozeslán všem aplikacím v zařízení. Aplikace, která příkazu rozumí, ho zachytí a provede. Tím dojde ke spuštění modulu, který se přenese do popředí a hlavní aplikace se přenese na pozadí.

Hlavní aplikace tedy komunikuje s modulem pomocí tzv. broadcast příkazů. Modul může komunikovat s aplikací, která funguje na pozadí také pomocí příkazů.

3.3.2 Modulární architektura a UWP

V této sekci budou prozkoumány možnosti vytvoření modulární aplikace v UWP. Bohužel, UWP v době tvorby tohoto návrhu nepodporuje vytváření aplikací

s doplňky, moduly či pluginy, které by bylo možné do aplikace dodávat samostatně. Také neumožňuje dynamické načítání externích knihoven aplikace a tedy i modulů.[13] Je proto nutné vyhodnotit možnosti UWP v oblasti propojení aplikací a možnosti architektury aplikace a vybrat nejlepší volbu.

Jelikož je UWP aplikace distribuovaná jako jeden Appx soubor, je nutné, aby všechny moduly byly buď uvnitř výsledné aplikace a nebo musí být moduly samostatné aplikace spouštěné z hlavní aplikace.

UWP umožňuje několik typů spuštění ostatních aplikací. Tím prvním je spuštění `LaunchUriForResultAsync`.[8] Spuštění aplikace začíná zavoláním metody, do které je předán identifikátor aplikace, která má být spuštěna a parametry pro spuštění aplikace. Cílová aplikace by se měla spustit, provést operaci a následně vrátit výsledek operace. Cílová aplikace se spustí v novém okně a po dokončení operace se zavře. Tento způsob spuštění modulu je podobný tomu, jak se spouští aplikace v systému Android. Problém ale je, že aplikace, na rozdíl od aplikace v systému Android, je stále obyčejná aplikace a musí být spustitelná i samostatně.

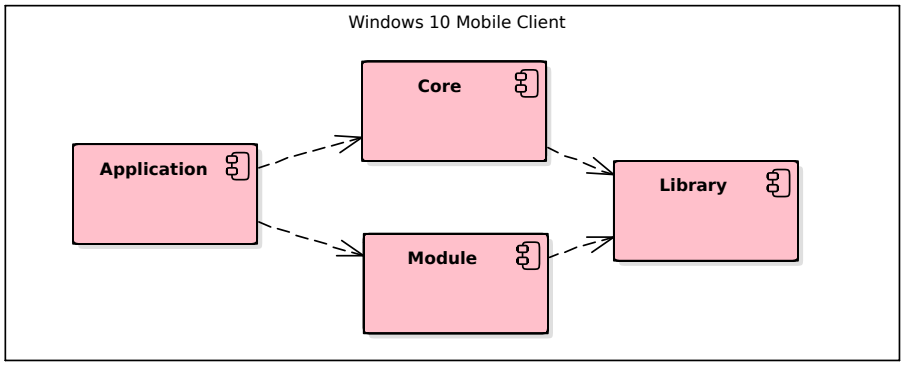
Spouštěcí aplikace navíc musí znát identifikátor spouštěné aplikace a v UWP momentálně není možné získat seznam aktuálně nainstalovaných aplikací a seznam jejich identifikátorů.[14] Aplikace se navíc na desktopových zařízeních spustí v novém okně, ale původní aplikace je pořád viditelná. To z hlediska používání není moc přívětivé.

Další možností jak spustit aplikaci je LaunchUriAsync.[8] To funguje podobně jako předchozí způsob, s tím rozdílem, že původní aplikace neočekává žádné výsledky od spuštěné aplikace a lze mezi oběma aplikacemi přecházet.

Pokud nebude modul samostatnou aplikací, musí být uvnitř jedné aplikace a to by znamenalo vzdát se nefunkčního požadavku na možnost doinstalovávat moduly zvlášť, dle požadavku uživatele. V aplikaci by byly vždy všechny moduly. Toto řešení umožní přecházet do modulu dle potřeby, vždy v rámci jednoho okna a aplikace bude vypadat jako jeden celek. Oproti předchozímu řešení není potřeba zabývat se přepínáním aplikací a řešení zobrazování oken. Na druhou stranu je potřeba vyřešit problém s vývojem modulu třetími stranami (tedy případnými zájemci o rozvoj aplikace) a jejich následující distribucí.

Jelikož UWP je nová a dynamicky se rozvíjející platforma, lze očekávat, že v budoucích verzích bude práce s tímto typem aplikací ještě vylepšena. V následující aktualizaci systému Windows je očekáváno přidání doplňků do webového prohlížeče Microsoft Edge, který je také vytvářen v UWP.[15] Je proto velmi pravděpodobné, že systém těchto rozšíření bude přestaven pro celou platformu. V momentální verzi Windows 10.0.10586 je však podpora omezená a je potřeba se rozhodnout pro jedno z výše popsaných řešení.

Zejména kvůli lepší uživatelské přívětivosti a nepotřeby řešení přepínání oken, které může být velmi problematické, byla zvolena možnost vytvoření jedné aplikace. Moduly v této aplikaci budou řešeny tak, aby se v budoucnu, pro případ, že dojde k rozvoji UWP v oblasti modularity, mohly snadno přesu-



Obrázek 3.2: Architektura aplikace.

nout do své aplikace či jiné formy rozšíření. Bude tedy nutné navrhnout způsob a postup, jakým budou moduly do aplikace vytvářeny a distribuovány. Toto rozhodnutí bylo diskutováno se zadavatelem a bylo jím schváleno.

3.3.3 Návrh architektury aplikace

Na obrázku 3.2 je vidět výsledný návrh architektury aplikace.

Stejně jako ve všech modulárních aplikacích, je i tato aplikace rozdělena na jádro a moduly. Navíc, stejně jako aplikace pro Android, obsahuje knihovnu se společnými předdefinovanými funkcemi a strukturami pro usnadnění vývoje modulů, pro jejich zmenšení a sjednocení jejich vzhledu.

Z distribučních vlastností UWP, popsaných v předchozí sekci, a z nutnosti řešit architekturu v rámci jedné aplikace plyne, že distribuční verze aplikace bude vždy obsahovat všechny existující moduly.

Architektura aplikace také musí umožňovat nezávislý vývoj dalších modulů jinými vývojáři. Dále by vývojáři měli mít možnost používat vlastní moduly bez nutnosti distribuce těchto modulů do centrální aplikace. To znamená, že musí existovat verze aplikace pro vývoj - tzv. SDK.

Následuje podrobnější popis významu jednotlivých částí architektury.

3.3.3.1 Spouštěcí část aplikace - Application

Část s názvem Application je hlavní částí aplikace, která se stará o obsluhu spuštění a inicializaci aplikace, což je realizováno třídou `App`. Dále bude nazývána jako spouštěcí část aplikace.

Jak je patrné z obrázku a z porovnání architektury s architekturou aplikace pro OS Android, tato spouštěcí část by mohla být součástí jádra aplikace. Spouštěcí část však byla oddělena, aby bylo možné snadno distribuovat jádro aplikace vývojářům. Aplikace je totiž kompilována jako celek a je proto nutné, aby spouštěcí část aplikace měla před kompilací reference na všechny

části aplikace (nemusí přímo, pokud je sama nepotřebuje, ale stačí zprostředkovaně, přes jiné části aplikace). To znamená, že do spouštěcí části musí být přidány reference na moduly, což je sada projektů s vlastními dalšími referencemi. Kdyby spouštěcí část byla součástí jádra aplikace, muselo by se celé jádro distribuovat vývojářům formou zdrojových kódů jen proto, aby mohli vývojáři tyto reference přidat, a následně aplikaci zkompilovat a spustit. Vývojáři by tím také získali možnost zasahovat do zdrojového kódu jádra a knihovny aplikace, což rozhodně není žádoucí.

Z těchto důvodů tedy byla spouštěcí část aplikace oddělena od jádra aplikace. Vývojáři do této spouštěcí části budou přidávat pouze reference na moduly a spouštět jádro aplikace, kterému předají seznam názvů sestavení veřejných částí modulů, vyskytujících se v aplikaci. Také na jádro přesměrují zbylé úkony aplikace, jako je například řízení navigace mezi obrazovkami, či různé inicializace aplikace.

Spouštěcí část také obsahuje ikonky aplikace, pro zobrazení v obchodu aplikací a další nepotřebné věci pro samotný vývoj. Tato část tedy nemusí být nutně součástí SDK, jelikož neobsahuje žádné důležité funkce a je pouze spojujícím článkem mezi jádrem a moduly.

3.3.3.2 Jádro aplikace

Další součástí aplikace je jádro. Jádro aplikace se stará o základní fungování celé architektury aplikace, tím že inicializuje, spouští a řídí komunikaci s jednotlivými moduly.

Jak plyne z analýzy, jádro aplikace také řeší zobrazení hlavní obrazovky a má ještě za úkol evidovat hráče a spravovat nastavení aplikace.

Pro práci s databází, entitami, zobrazením a zpracováním dat a komunikací s moduly využívá základní třídy z knihovny aplikace, takže ta je nedílnou součástí SDK i distribučního balíčku.

Jádro aplikace má pro okolní svět pouze jedno vstupní místo určené pro spouštěcí aplikaci a vše ostatní musí být skryto. S moduly pak komunikuje tak, že si je samo vytvoří a předá jim objekt s implementovaným rozhraním, přes které budou moduly s jádrem komunikovat.

Jádro aplikace tedy může být součástí SDK ve zkompilovaném tvaru, tj. ve formě DLL knihoven.

Moduly spouští podle toho, jestli si uživatel přeje používat jejich sporty. Z toho plyne, že při prvním spuštění aplikace jádro načte všechny moduly a jejich sporty a dá na výběr uživateli, které chce použít. Podle toho už pak při dalším spouštění načítá do paměti jenom ty moduly, které jsou opravdu potřeba. Tento proces je detailněji popsán v sekci návrhu logické vrstvy jádra aplikace.

3.3.3.3 Moduly aplikace

Moduly jsou části aplikace, obsahující třídu pro komunikaci s jádrem aplikace a vlastní implementaci jednotlivých sportů.

Každý modul je nutné přidat jako referenci do spouštěcí části aplikace a zaregistrovat ho k předání do jádra.

Moduly k implementaci sportů také používají společnou knihovnu aplikace. Tu by se měly snažit využívat co nejvíce, aby výsledná velikost aplikace byla co nejmenší a vzhled a chování bylo jednotné a případně opravitelné v jednom místě, tedy v knihovně.

Každý modul se bude skládat z několika projektů, které představují jednotlivé vrstvy modulu. Ty budou více popsány v samostatné sekci o návrhu knihovny aplikace a modulu.

K vytvořenému SDK bude přiložen ukázkový modul ve formě projektů se zdrojovými kódy, aby měl případný vývojář jednodušší začátek a mohl kopírovat kusy kódu, které mu budou vyhovovat.

3.3.3.4 Knihovna aplikace

Poslední částí architektury aplikace je společná knihovna aplikace. Ta obsahuje rozhraní, která umožňují fungování platformy a komunikaci jádra s moduly.

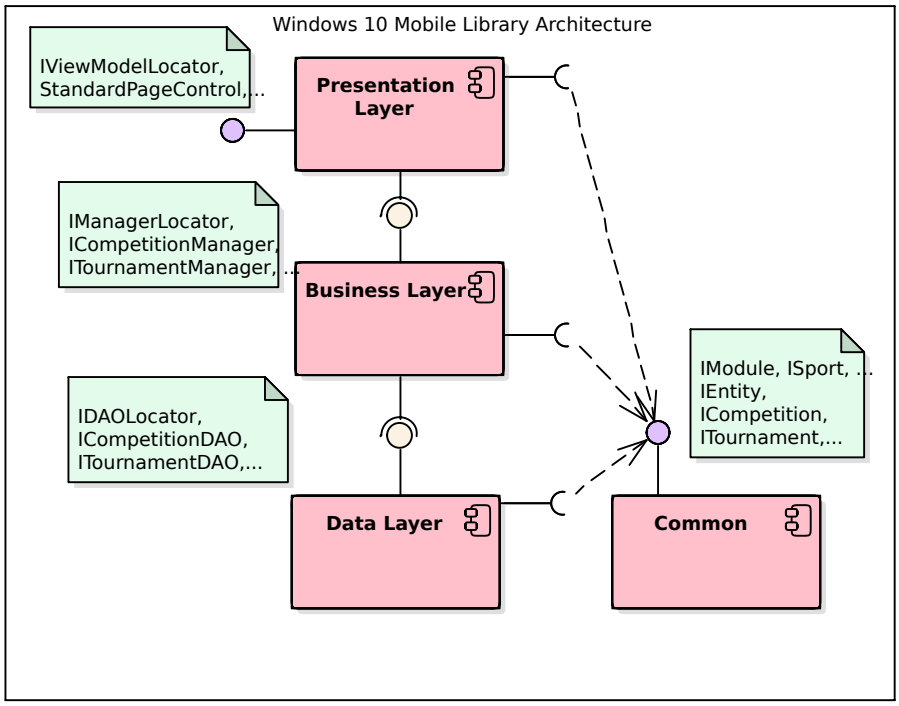
Také obsahuje základní implementace tříd, potřebných k fungování aplikace a modulů podle doménového modelu. Tím usnadní a urychlí vývoj a ušetří místo ve velikosti aplikace. Musí však být vytvořena dostatečně univerzálně, aby bylo možné její prvky snadno rozšiřovat v modulech, zároveň by ale neměla moduly příliš omezovat a svazovat. Detailnější návrh tříd knihovny a jejich fungování bude více popsáno v sekci o návrhu knihovny aplikace.

3.4 Návrh knihovny aplikace

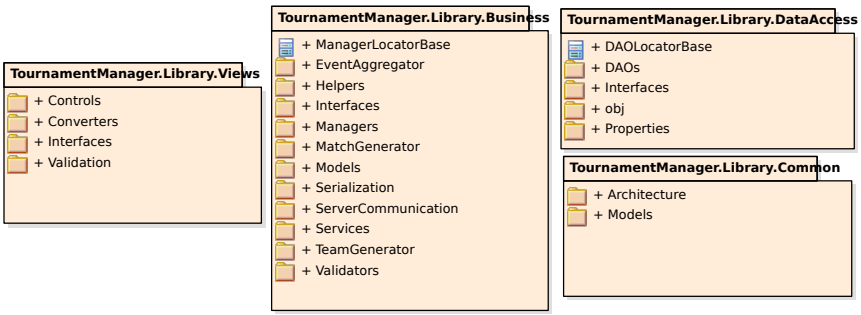
Knihovna obsahuje třídy a prvky společné pro použití v ostatních modulech.

Naprostá většina tříd, které knihovna bude obsahovat budou generické s typovými parametry tak, aby je mohly moduly a jádro použít s vlastními entitami, které musí pouze implementovat základní rozhraní entit z knihovny a nebo mohou rozšiřovat entity definované knihovnou. Generické třídy budou postupně zpřesňovány a nabízí tak několik úrovní potomků. Vývojář se tedy může rozhodnout, jestli chce pro práci s entitou použít už lépe připravenou třídu přímo pro danou entitu a nebo obecnější třídu pro obecnou entitu s tím, že potřebné metody si bude implementovat sám.

Z obrázku 3.3 je patrné, že knihovna kopíruje vícevrstvou architekturu, která je využita v ostatních částech aplikace. Datová vrstva architektury je rozdělena do dvou částí, kde první částí je tzv. společná vrstva, která obsahuje zejména entity, jejich rozhraní a rozhraní pro komunikaci částí aplikace a



Obrázek 3.3: Architektura knihovny aplikace



Obrázek 3.4: Přehled tříd knihovny aplikace.

celkové fungování architektury. Druhou částí je část pro přístup k datům, která poskytuje generické třídy pro jednotlivé entity, zajišťující práci s databází. Logická vrstva je pak vrstvou obsahující základní rozhraní a implementace generických tříd, tzv. managerů, které pracují s rozhraními tříd a entit z datové vrstvy a poskytují své metody nadřazeným prezentačním vrstvám. Na obrázku **??**je pak vidět celkový přehled tříd definovaných v knihovně aplikace.

3.4.1 Společná část knihovny

Společná část knihovny bude obsahovat zejména entitní třídy a jejich rozhraní. Každá entita bude implementovat návrhový vzor DTO a bude tedy obsahovat pouze vlastnosti a žádné metody. Vlastnosti navíc budou mít pomocí atributů řečeno, jak mají být ukládány do databáze. To je nutné, kvůli použití knihovny SQLite.Net-PCL pro práci s databází a také z tohoto důvodu je tato společná část součástí datové vrstvy.

Entity tedy budou sloužit jako transportní objekty mezi všemi vrstvami aplikace.

Instance entit budou vytvářeny zejména knihovnou SQLite.Net-PCL při načítání z databáze a jinak nebude jejich vytváření nijak redukováno, jelikož jsou to pouze kontejnery a neobsahují žádnou vlastní logiku. Pro nastavení výchozích hodnot pak vytváření instancí entit budou řídit třídy v logické vrstvě a ochranu před uložením nesprávných dat do databáze společně zajistí třídy v logické a datové vrstvě.

Ke každé entitě bude existovat rozhraní. To bude sloužit zejména k omezování generických parametrů v ostatních třídách.

Entity v knihovně aplikace jsou stejné jako entity v aplikaci pro OS Android. Jejich databázový model je vidět na obrázku 3.5.

Entity budou vytvářeny postupným děděním od základního předka všech entit, který definuje povinnost, aby entita měla identifikátor. Poté následuje předek pro entity určené ke sdílení. Tyto entity budou mimo svých vlastních vlastností, daných doménovým modelem, mít ještě vlastnosti nutné pro komunikaci se synchronizačním serverem. Konkrétně jsou to tyto.

UUID Což je univerzální identifikátor generovaný systémovou třídou Gu•id. Ten bude nastaven každé entitě v jejím konstruktoru.

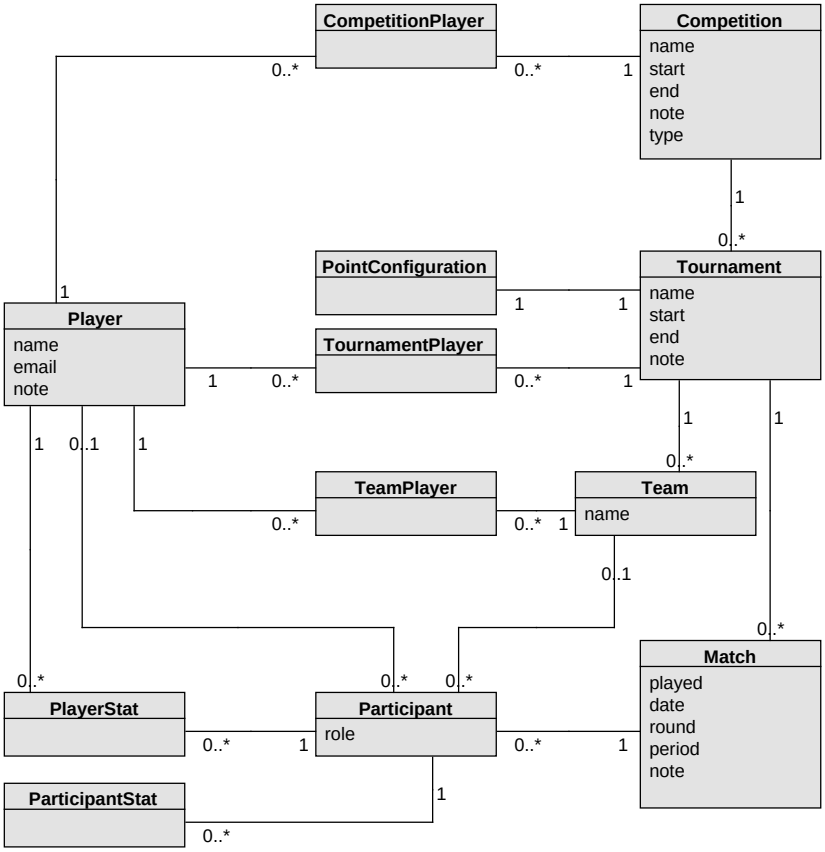
Tokeny Každá entita pro sdílení má tři vlastní tokeny, každý pro jinou úroveň oprávnění. Token je tedy jakýmsi ověřovacím údajem pro práci s entitou na serveru.

Etag Toto je údaj sloužící pro sledování změn entit na serveru. Při každé změně na serveru je entitě vygenerován nový etag, který je následně předán jejím nadřazeným entitám.

Typ entity Toto je textové označení typu entity, sloužící k rozpoznání typu entity v serializovaném stavu.

Last modified a Last synchronized Časové údaje o poslední změně entity uživatelem a poslední synchronizaci entity na serveru. Tyto údaje mohou sloužit pro další rozhodování při případné synchronizaci.

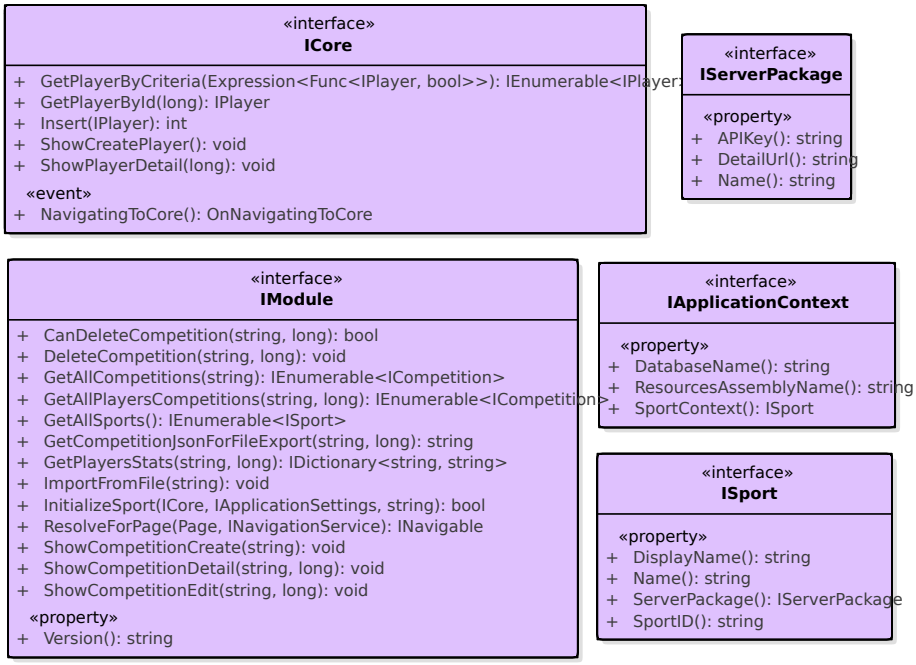
Z předka pro sdílené entity budou děděním vytvořeny entity `Competition`, `Tournament`, `Match`, `CompetitionPlayer` a `Team`. Zbylé entity budou vytvořeny děděním z předka všech entit.



Obrázek 3.5: Databázový model.

Při komunikaci se synchronizačním serverem pak budou entity pro sdílení vystupovat jako samostatné celky. Ostatní entity, jako jsou např. statistiky, pak budou během serializace přidány k jejich nejbližším entitám určeným pro sdílení formou synchronizačních dat. Tato problematika bude podrobněji popsána v sekci o návrhu logické vrstvy knihovny.

Toto rozložení entit na sdílené a obyčejné bylo rozhodnuto společně v rámci týmového projektu. Cílem je ušetřit data ukládaná v zařízení i data přenášená při případné komunikaci se serverem. Kdyby byla každá entita sdílená, musela by každá entita mít vlastní synchronizační data popsaná výše, které by bylo nutné ukládat a přenášet. Velikost těchto dat by rychle rostla u entit jako jsou účastníci a statistiky z důvodu jejich velkého množství a proto bylo rozhodnuto, že tyto data se budou řešit formou synchronizačních dat nejbližších sdílených entit.



Obrázek 3.6: Rozhraní pro fungování architektury aplikace.

3.4.1.1 Rozhraní pro fungování architektury aplikace

Součástí společné části budou kromě entit také rozhraní pro komunikaci jednotlivých částí aplikace. Ty se budou nacházet v této vrstvě, jelikož budou moci být, stejně jako entity, použity ve všech vrstvách aplikace.

IModule Toto rozhraní bude definovat metody a vlastnosti, které od modulu bude vyžadovat jádro aplikace. Jedná se například o poskytnutí seznamu sportů, inicializace sportu, vyřešení navigace na předanou stránku, poskytnutí seznamu soutěží, přesměrování na detail, editaci, vytvoření soutěže atd. Každý modul by měl implementovat právě jednu třídu podle tohoto rozhraní.

ICore Rozhraní definující metody, poskytované jádrem modulu. Zde je to zejména poskytnutí hráčů podle různých kritérií, vytvoření hráče, přesměrování na zobrazení detailu hráče či formulář pro vytvoření hráče. Jádro bude implementovat právě jednu třídu podle tohoto rozhraní.

IServerPackage Rozhraní definující vlastnosti, které vývojář modulu musí zadat, pro správnou registraci modulu do aplikace. Jedná se zejména unikátní název modulu a API klíč pro komunikaci se synchronizačním serverem. Oba tyto údaje vývojář získá registrací modulu na synchroni-

začním serveru. Více o této problematice a o procesu registrace modulu je k přečtení v práci Michala Hacury[4].

ISport Rozhraní, které představuje jeden sport implementovaný modulem. Modul pro každý sport vytvoří jednu instanci třídy s tímto rozhraním.

Rozhraní pro sport definuje název, který se má zobrazit uživateli v seznamu sportů, unikátní název sportu v rámci modulu, který si vývojář může zvolit, a instanci rozhraní IServerPackage. Také ještě obsahuje identifikátor, který je složen z unikátních názvů sportu a modulu. Vývojář tedy má možnost se rozhodnout, jestli bude na synchronizačním serveru registrovat celý modul a nebo samostatné sporty modulu.

IApplicationContext Toto je rozhraní definující kontext, který modul interně použije pro rozlišení aktuálně používaného sportu. Každý sport má v modulu jeden kontext a při přechodu z jádra do modulu pomocí jedné z metod v rozhraní `IModule` je podle předaného sportu zvolen aktuální kontext, podle kterého v modulu dojde k vytvoření a nastavení příslušných tříd v prezentační, logické a datové vrstvě tak, aby pracovaly s požadovaným sportem.

V každém kontextu je uložena informace o názvu databáze, která má být použita, názvu assembly, která obsahuje texty pro překlady a může mít instanci rozhraní `ISport`, ke kterému tento kontext patří. Vývojář pak může v modulu s tímto kontextem dále pracovat - např. ho předávat do tříd a podle aktuálního sportu upravovat chování těchto tříd, nebo vytvářet jiné třídy.

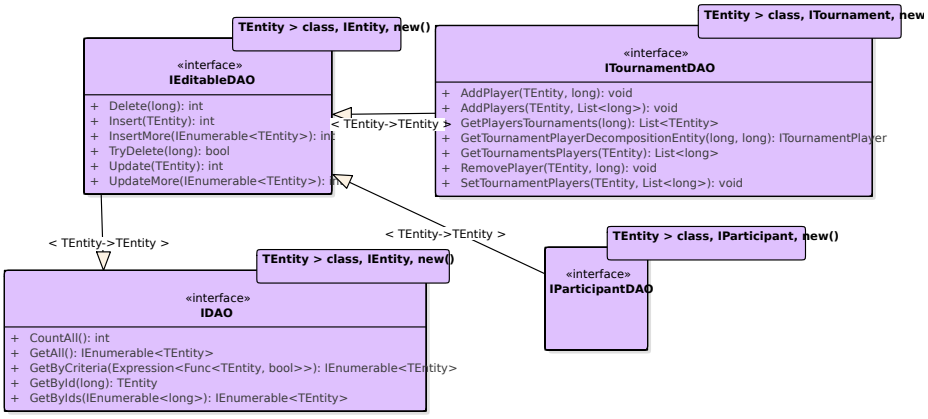
3.4.2 Datová část knihovny

V části knihovny pro přístup k datům je využit návrhový vzor Data Access Object, podobně jako je tomu v aplikaci pro OS Android. DAO třídy poskytují metody pro abstrakci přímé práce s databází. [16]

Každá DAO třída slouží pro práci s jednou entitou, případně s jejími vazebními entitami(případ entity `TournamentPlayer` a `TeamPlayer` a tedy `TournamentDAO` a `TeamDAO`.

Všechny tyto třídy budou generické, aby bylo možné snadno využít jejich základní funkce s entitami, které budou definovány v jednotlivých modulech a budou rozšiřovat základní entity knihovny.

DAO třídy v knihovně budou vystavěny hierarchií dědičnosti. Od základní abstraktní třídy budou děděním vznikat přesnější třídy, až po třídy pro použití s jednotlivými entitami. Tak, aby měl vývojář možnost volby několika úrovní, které může rozšířit. V případě, že entita nějak pracuje s hráčem (`Competition`, `Tournament`, `Team`), bude rozhraní ještě rozděleno na část rozhraní bez práce s hráčem a kompletní rozhraní s prací s hráčem. To je možné vidět na obrázku



Obrázek 3.7: Rozhraní DAO tříd.

3.7. Tím nebude vývojář nucen specifikovat entitu hráče, když bude chtít pracovat pouze se samotnou entitou.

V případě, že si vývojář nebude přát nic měnit, nebude muset nic rozšiřovat a konkrétní DAO třídy knihovny bude možné použít v modulu přímo. Jako generický parametr bude stačit dodefinovat typ entity, s kterým má třída pracovat.

Každá DAO třída bude mít jako parametr v konstruktoru instanci třídy `DBConnectionFactory`. Ta pro své vytvoření bude potřebovat název databáze, pro které bude následně vytvářet připojení, které je realizováno použitou knihovnou SQLite.Net-PCL. Třída pro připojení k databázi, která umožňuje klasické příkazy jako jsou insert, update, delete, get a další, které jsou potřeba pro každý úkon DAO třídy.

Název databáze pro třídu `DBConnectionFactory` bude získán z aktuálního kontextu modulu.

3.4.3 Logická vrstva

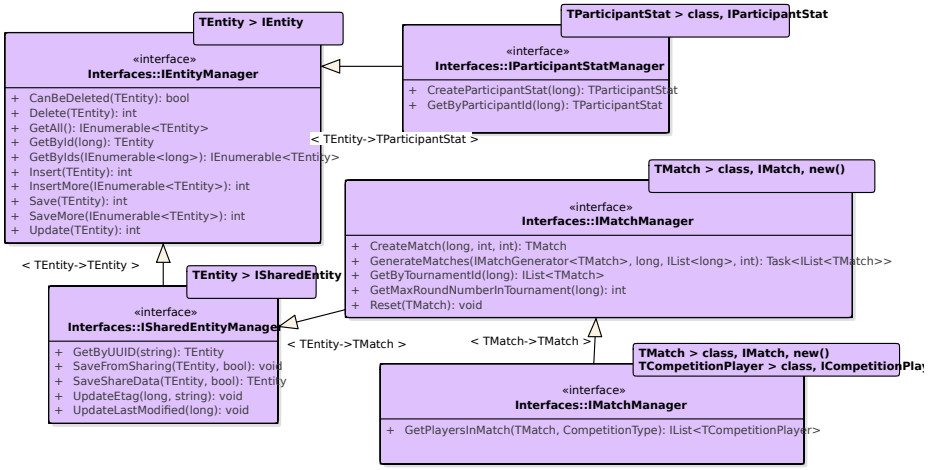
Logická vrstva knihovny je poměrně členitá a proto bude tato sekce rozdělena do dalších podsekcí, kde budou její jednotlivé části popsány.

3.4.3.1 Managery entit

Managery entit jsou třídy, které poskytují komplexní metody pro práci s entitami, zejména prezentační vrstvě, ale také dalším třídám v logické vrstvě. Ukázka jejich rozhraní a dědičnosti je vidět na obrázku 3.8.

Všechny tyto třídy jsou generické, podobně jako DAO třídy na datové vrstvě a umožňují tak snadné rozšíření a použití z modulů.

Každý manager poskytuje metody pro práci s jednou entitou, k tomu potřebuje zejména DAO třídu pro tuto entitu. Ta je mu poskytnuta jako roz-



Obrázek 3.8: Rozhraní Managerů.

hraní, které je definované v datové vrstvě knihovny. Když vývojář ve svém modulu rozšíří DAO z knihovny, bude toto rozšířené DAO stále splňovat rozhraní knihovny a vývojář ho tak bude moci předat tomuto manageru.

I když manager poskytuje metody pro práci s jednou entitou (v některých případech i více entit, když jde například o hráče a entitu, ve které hráč figuruje) a jeho rozhraní tedy vyžaduje jeden generický parametr, může ke své správné funkci potřebovat více entit. Konkrétní implementace rozhraní manageru tak může mít i více generických parametrů, než jen jednu entitu, pro kterou jsou vytvořeny.

Entity určené pro sdílení pak mají ještě speciálního předka, což je třída `SharingManager`. Ta poskytuje metody pro práci se sdílenými entitami a upravuje chování ostatních metod. Například při volání metody pro uložení entity aktualizuje její datum a čas poslední změny.

3.4.3.2 Validátory

Knihovna také obsahuje validátory a validační entity, které jsou postavené na základě knihovny `Template10.Validation`.

Validátory jsou třídy, které obsahují validační funkci, metody pro převod klasické entity na validační a převod validační entity na klasickou entitu.

Validační funkce, je funkce, které je jako parametr předána validační entita. Úkolem validační funkce je zaznamenat všechny chybně vyplněné položky do pole `Errors` předané validační entity.

Validační entity jsou tedy dekorátory[6] standardních entit a jako vše v knihovně, co souvisí s entitami, jsou generické a konkrétní entitu, kterou obalují, mají

[6]Dekorátor - návrhový vzor, podle kterého je existující objekt rozšířen o novou funkcionalitu.[17]

jako typový parametr. Tyto entity také dědí třídu `ValidatableModel` definovanou v `Template10.Validation`.

Validační entitě jsou při vytvoření validátorem nastaveny výchozí hodnoty a je jí předána validační funkce validátoru. Takto vytvořená validační entita je předána prezentační vrstvě, která už jen používá validačních metod entity a po úspěšném dokončení práce ji pomocí validátoru převede zpět na klasickou entitu a předá dál ke zpracování.

Pro použití v modulu bude možné využít přímo tříd definovaných v knihovně či rozšířit validátor, validační entitu a implementovat vlastní validační funkci. Pro správné zobrazení v prezentační vrstvě bude nutné v modulu doplnit texty případných chyb, které nalezne validační funkce.

3.4.3.3 Generátory

Součástí logické vrstvy knihovny budou také generátory zápasů a soupisek týmů.

Generátory zápasů jsou definovány rozhraním, které obsahuje jednu metodu. Vstupem této metody bude seznam identifikátorů účastníků (ID entity `Participant`) a číslo kola, pro které se má zápas vygenerovat, aby bylo možné měnit domácí a hosty pro každé sudé nebo liché kolo. Výstupem generátoru bude seznam vytvořených zápasů s nastavenou periodou a kolem a ke každému zápasu pak množina účastníků - typicky domácí a hosté.

Implementován bude generátor zápasů každý s každým, stejným způsobem jako je v aplikaci na OS Android. Bude použit i stejný algoritmus, popsaný v práci Josefa Němečka[1], aby se aplikace chovali stejně.

Generátory soupisek týmů jsou definovány podobným rozhraním jako generátory zápasů. Obsahuje jednu metodu, na jejímž vstupu je seznam týmů, seznam hráčů a seznam statistik, podle kterých bude generátor generovat. Výstupem pak bude seznam vygenerovaných týmů a ke každému týmu množina hráčů, kteří jsou jeho členy.

Implementován bude vyvážený generátor týmů, který náhodně seřadí prázdné týmy a hadovitě (od začátku do konce, od konce k začátku atd.) do nich bude přidávat hráče, seřazené podle předaných statistik od nejlepší k nejhorší.

3.4.3.4 ServerCommunication

Třídy v této sekci budou sloužit pro komunikaci se synchronizačním serverem.

Hlavní třídou, která bude řešit veškerou komunikaci se serverem bude statická třída `ServerCommunicator`. Tato statická třída bude implementovat všechny metody API synchronizačního serveru a bude pro vývojáře přístupovým bodem pro komunikaci se serverem. Parametry jednotlivých metod odpovídají požadavkům metod v API.

Pro komunikaci se serverem bude nutné vytvořit třídy, které budou odpovídat všem resource objektům, které mohou ze serveru přijít jako odpověď

na požadavek. Například to bude třída [ServerCommunicationItem] představující serverový `ItemResource`, do kterého budou pomocí serializérů, popsaných v následující sekci, serializovány entity určené pro sdílení. `ServerCommunicationItem` bude následně zkonvertován do formátu JSON a odeslán pomocí třídy `ServerCommunicator` na server.

Serverový `ItemResource` a jemu odpovídající třída `ServerCommunicationItem` pak mají tři hlavní části.

Synchronizační data Ty slouží pro uložení doménových dat entity - tedy např. názvy, poznámky, statistiky atd. Do synchronizačních dat mohou být serializovaná data z více entit než jen té aktuální.

SubItems To je množina dalších `ServerCommunicationItem` instancí (serializovaných entit), které mají vztah s aktuální entitou. Pro soutěž jsou to např. hráči v soutěži a turnaje.

Údaje pro sdílení Sem patří všechny údaje sdílených entit, které jsou nutné pro evidenci serializované entity na synchronizačním serveru. Jedná se o UUID, tokeny a etag, které již byli popsány v sekci o návrhu společné části knihovny.

Tímto způsobem tedy budou vytvořeny i třídy pro serverový `ErrorResource`, `PackageResource` a `TokenResource`.

Více o problematice komunikace se serverem je možné zjistit v práci Michala Hacury[4].

3.4.3.5 Serialization

Tato část knihovny bude obsahovat třídy typu `Serializer` a třídy typu `Loader`, které budou využity využity pro import a export soutěží do souboru a také budou moci být využity při komunikaci se serverem.

`Serializer` Třídy tohoto typu budou sloužit pro serializaci a deserializaci entit, určených pro sdílení, do kontejneru `ServerCommunicationItem`, s kterým pracuje synchronizační server. Kromě serializace a deserializace entit budou tyto třídy také umožňovat serializaci celých stromů entit. To se využije zejména při importu a exportu do souboru. V případě Soutěže tedy dojde k postupné serializaci hráčů a turnajů. Turnaje pak spustí postupnou serializaci týmů, zápasů atd. až dojde k serializaci celé soutěže.

`Loader` Třídy tohoto typu budou třídy, které budou zajišťovat správné uložení serializovaných entit do databáze. V metodě pro uložení do databáze dojde k deserializaci entity pomocí serializéru, následné vyhodnocení jejího stavu vůči databázi a uložení do databáze. Uložení bude realizováno pomocí managerů entit.

Oba typy těchto tříd budou mít v knihovně základní implementace, které bude možné použít v modulech přímo. Bude však možné procesy serializace, deserializace a uložení entit do databáze přetížit a změnit.

3.4.3.6 Helpers a Models

Třídy v této části logické vrstvy budou představovat komplexnější logiku, prováděnou nad několika entitami a jejich managery zároveň. Bude zde například třída `ScoredMatchHelper`, která zjednoduší získání potřebných informací pro zobrazení dat o zápase. Tato třída bude agregovat entitu zápasu, jeho účastníků (entity `Participant`) a jejich statistik (entity `ParticipantStat`) do sdružené entity, která se bude jmenovat `ScoredMatch`.

3.4.4 Prezentační vrstva

Tato část knihovny se zabývá usnadněním tvorby zobrazovací části aplikace. Každá obrazovka, která se bude zobrazovat uživateli je tvořena dvojicí tříd - `View` a `ViewModel`. V knihovně však nebudou řešeny předlohy pro celé obrazovky, jelikož části `View` z těchto dvojic nejsou moc dobře rozšiřitelné a upravitelné a dá se předpokládat, že v této oblasti se bude každý modul dosti lišit. Z tohoto důvodu bude tato část knihovny řešena pomocí předpřipravených komponent, z kterých se budou obrazovky skládat.

Prezentační vrstva knihovny tedy bude obsahovat přednastavené prvky pro sjednocení designu aplikace a snadnější vytváření obrazovek. Tento přístup také umožní snadné budoucí centrální opravování případných chyb, vylepšování funkčností či změny vzhledu.

Před tvořením návrhu této vrstvy knihovny bylo nutné vyřešit návrh grafické podoby obrazovek a jejich částí, které jsou popsány v následujících sekcích.

3.4.4.1 Návrh grafického uživatelského rozhraní

Jelikož musejí splňovat designové postupy pro aplikace UWP, mohou se lišit od aplikace pro OS Android. Při návrhu tedy bude snaha využívat takové prvky, aby byli uživatelé schopni mezi jednotlivými aplikacemi přecházet s co nejmenším zmatením.

Obrazovka aplikace bude rozvržena do čtyř hlavních částí, podobně jako obrazovky aplikace pro OS Android a aplikace MSN Sport.

První částí je levé menu s tzv. „Hamburger“ tlačítkem, které bude vyjíždět směrem doprava a bude se automaticky roztahovat v závislosti na velikosti obrazovky a velikosti okna aplikace. V menu budou odkazy na stránky v nejvyšší úrovni, popsaní v analytické části návrhu uživatelského rozhraní. Tato část je tedy velmi podobná té v aplikaci pro OS Android.

Druhou částí je v horní části obrazovky umístěná hlavička s popiskem - ta bude zobrazena na každé stránce a bude obsahovat nadpis stránky. Toto také koresponduje s aplikací pro OS Android.

Pod hlavičkou je pak místo pro třetí část, což je obsah stránky. Zde bude existovat několik různých druhů obsahů podle množství a typu dat k zobrazení. Všechny tyto obsahy by se měly přizpůsobovat velikosti obrazovky a vhodně využívat dostupnou plochu k zobrazení co nejvíce informací. Pro každý typ obsahu bude v knihovně vytvořena předloha pro jednoduché použití v modulu. Jednotlivé obsahy jsou popsány níže v této sekci.

Na závěr, pod obsahovou částí u spodního okraje obrazovky pak má své místo tzv. "Command Bar", tedy panel příkazů. Zde se nachází akce, které je možné provést nad aktuálním obsahem, či aktuálně zvolenou položkou obsahu. Hlavní akce budou zobrazeny přímo na panelu formou ikonky a popisku. Vedlejší akce pak budou schovány v menu, které se zobrazí po zmáčknutí na tlačítko s ikonkou tří teček. Každá hlavní akce také může po kliknutí zobrazit kontextové menu s volbou další - přesnější akce. Toto je od aplikace pro OS Android, kde se akce nachází v horní části obrazovky, odlišné a jedná se o specifický prvek aplikací pro OS Windows 10.

Následuje popis typů obsahů, které se budou vyskytovat na obrazovkách a jejich odlišnosti od aplikace pro OS Android.

Seskupený seznam položek Prvním obsahem bude tzv. "Semantic Zoom", což je specialita aplikací pro OS Windows 10 Mobile. Jedná se o seskupený seznam, který se skládá ze sekcí. Každá sekce má hlavičku s popiskem a seznam položek, které k této hlavičce patří. Na hlavičku sekce je možné kliknout a tehdy se tento sdružený seznam přepne do režimu zobrazení, kdy zobrazuje pouze seznam hlaviček sekcí. Po výběru sekce se přepne zpět na zobrazení sekcí s položkami, ale přesune se v seznamu tak, aby byla zvolená sekce vidět na horní straně obsahu. Díky tomu je možné se rychle orientovat a navigovat v seznamu s velkým množstvím sekcí a položek. Tento seznam bude využit pro zobrazení přehledu soutěží. Soutěže budou rozděleny do sekcí podle sportů, do kterých patří.

Seznam položek Druhým obsahem pak bude seznam položek. Ten bude umožňovat definovat způsob zobrazení jednotlivých položek, výběr více položek pro skupinové operace jako je například přidání více položek a po delším podržení položky či pravém kliknutí na položku bude zobrazovat kontextovou nabídku s možnými operacemi, které bude možné nad zvolenou položkou provést. Tento seznam položek bude obsažen i ve výše zmíněném sdruženém seznamu a lze proto všechny jeho vlastnosti a možnosti očekávat i tam. Jeho využití lze očekávat například i pro seznam hráčů, turnajů, zápasů a týmů.

Detail se záložkami Třetím obsahem bude detail. Detail se bude skládat z tzv. tabů. Jedná se o horizontální záložky, které umožňují rozdělit

obsah detailu do dalších sekcí, v kterých lze opět využít jeden ze zde zmíněných obsahů. Každá tato sekce v záložce má vlastní dolní panel s akcemi.

Tento prvek se v UWP nazývá Pivot a v základní podobě vypadá jinak, než ten v aplikaci pro OS Android. Jak ale bylo možné vidět v aplikaci MSN Sport z rešerše, je možné tento prvek upravit tak, aby vypadal podobně, jako ten pro OS Android. A tak bude při realizaci učiněno.

Tabulková data Dalším obsahem, bude zobrazení dat formou tabulky. Designové postupy pro UWP neposkytují přímo návod ani prvek, kterým lze řešit takovéto zobrazení dat a proto bude nutné vytvořit vlastní komponentu složenou z dostupných prvků. Popis jejího vytvoření se nachází v následující sekci.

Formulář V neposlední řadě bude jedním z obsahů také formulář. Formuláře budou obsahovat políčka nutná k vyplnění. Každé políčko bude mít nad sebou popisek, který říká o jaké políčko se jedná. Každé políčko bude umožňovat zobrazit informaci o chybném vyplnění hodnoty. Tato informace se bude zobrazovat pod políčkem, bude zvýrazněna červeným textem a opatřena zvýrazňující červenou ikonou.

Na konci každého formuláře budou vpravo dole dvě tlačítka - jedno pro potvrzení uložení a druhé pro návrat z formuláře, bez jeho uložení. Tlačítko pro potvrzení bude vlevo od tlačítka pro návrat a bude výraznější.

3.4.4.2 Předpřipravené prvky obrazovek

V této sekci budou popsány předpřipravené prvky pro vytváření obrazovek, které budou součástí prezentační vrstvy knihovny. Většinou se jedná o samostatné a vícekrát použitelné prvky v rámci jedné obrazovky, ale připraveny budou také rozvržení obrazovky s jedním typem obsahu.

Seznam Seznam bude komponenta, která bude mít předpřipravené rozvržení položky pro zobrazení entity, která má název, a datum začátku a konce, jelikož tyto entity budou nejčastěji zobrazovány v seznamech. Také bude mít předpřipravenou kontextovou nabídku pro editaci a mazání položek, včetně potvrzovacích dialogů. Vývojáři už pak bude pouze stačit doplnit správné texty a entity, případně si bude moct doplnit vlastní kontextovou nabídku, šablonu pro položku či potvrzovací dialogová okna.

Zobrazení času Tato komponenta bude sloužit pro zobrazení časových údajů. Bude se starat o převod centrálního času do lokálního, takže vývojáři bude stačit předat pouze data uložená v entitě, bez dalších úprav.

Zobrazení vlastnosti Tato komponenta bude sloužit pro zobrazení položek typu popisek - hodnota. Ty budou k vidění zejména v detailech. Kom-

ponenta se postará o vhodné stylování těchto informací a vývojář opět pouze dodá potřebná data.

Tabulka Komponenta tabulky bude sloužit zejména k zobrazení statistik.

Základní komponentou pro zobrazení tabulkových dat bude komponenta `Grid`. Ta ovšem neumožňuje zadat data formou kolekce a je proto nutné použít ji v kombinaci s komponentou pro seznam, kde na každém řádku bude `Grid` použit pro organizaci sloupců. Z toho plyne, že sloupce nad sebou budou muset udržovat stejnou šířku, aby na sebe navazovaly.

Pro správné a očekávané chování tabulky bude ještě nutné oddělit řádek hlavičky s názvy sloupců od řádků se záznamy. To umožní vertikální posun tabulky tak, aby název sloupců byl vždy navrchu. Také bude nutné oddělit první sloupec s názvem řádku od datové části záznamů a hlavičky sloupců.Díky tomu bude umožněn horizontální posun nadpisů sloupců společně s daty záznamů tak, aby byl pořád viděn název řádku. Tabulka také bude umožňovat řazení podle sloupců, kliknutím na název sloupce a díky použití seznamu na řádky tabulky, budou mít řádky stejné vlastnosti jako položky v obsahu seznamu, popsané výše.

Vývojář bude muset pro tabulku dodat dvě sady dat. První sada bude identifikátor sloupce a jeho název a druhá sada bude speciální sada entit, které budou obsahovat data pro všechny identifikátory sloupců. Dále pak bude vývojář muset poskytnout příkaz, kterému tabulka předá název sloupce a způsob, jakým mají být záznamy seřazeny (vzestupně či sestupně).

Počítadlo Tato komponenta bude sloužit pro zaznamenávání skóre. Bude obsahovat textové pole zobrazující aktuální číslo a dvě tlačítka - pro zvýšení či snížení tohoto čísla. Tato komponenta bude využita na detailech zápasů.

Základní Obrazovka Tato komponenta bude obsahovat připravené rozvržení pro obrazovku,která bude obsahovat titulek, místo pro obsah a místo pro spodní panel akcí. Do titulku vývojář doplní název. Obsah a spodní panel bude dále specifikovat pomocí ostatních komponent.

Obsah se záložkami Toto je komponenta pro obsah se záložkami. Bude se vkládat do komponenty pro základní obrazovku a bude zajišťovat správné styly záložek.

Záložka Komponenta záložka slouží pro vložení do obsahu se záložkami a obsahuje rozvržení na obsah a spodní panel. Díky tomu bude moci mít každá záložka vlastní spodní panel s vlastními akcemi. Vývojář tedy bude plnit obsah se záložkami těmito komponentami.

Záložka také bude obsahovat titulek, který bude zobrazovat komponenta pro obsah se záložkami v seznamu záložek.

Formulář pro validační entitu Toto je komponenta připravená pro zobrazení formuláře, pomocí kterého budou vyplňovány data do validační entity. Vývojář do této komponenty pouze doplní políčka, která bude uživatel vyplňovat a předá komponentě validační entitu.

Tato komponenta pak po zadání hodnoty do formuláře zajistí validaci entity, pomocí její validační funkce a případně zobrazí informace o chybné hodnotě a znemožní potvrzení formuláře.

Komponenta bude nabízet dvě hlavičky - jednu pro editační režim a druhou pro režim vytváření. Vývojář tak jen zadá správné texty a pomocí přepínače bude volit, jaký režim má komponenta zobrazit a tím ušetří vytváření formuláře pro editaci.

Zjednodušený formulář Komponenta podobná jako formulář, ale nebude pracovat s validační entitou. Pouze poskytne vzhled formuláře. Tedy nadpis, prostor pro políčka formuláře a tlačítka pro potvrzení či zrušení. Vše ostatní už si bude řídit vývojář sám.

Tato komponenta bude mít využití například pro obrazovky přidání hráčů do týmů, turnajů atd.

3.4.5 Komunikace vrstev

Komunikaci mezi vrstvami, ale i uvnitř vrstev bude zajišťovat rozhraní pro získání instancí požadovaných rozhraní. To se bude vyskytovat na každé vrstvě.

Jedná se o rozhraní, jejichž název končí slovem `Locator`. Toto rozhraní bude poskytovat generické metody pro získání implementace požadovaného rozhraní, které je předáno jako typový parametr. To z toho důvodu, že není možné dopředu určit množství , rozhraní a složitost tříd, která budou vytvořena a použita v modulech.

Jestli třída implementující toto rozhraní bude vždy vytvářet nové instance sama, jestli bude recyklovat již vytvořené instance nebo jestli instance bude získávat ještě jiným způsobem už bude záviset na situaci a také na uvážení vývojáře. Rozhraní je tedy spíše abstrakcí pro IoC kontejner[7].

Ve všech třídách by měl být preferován způsob, kdy jsou závislosti předávány do konstruktoru, tedy tzv. Constructor Injection[18]. Je však možné vyžadovat referenci na rozhraní třídy `Locator` a následně z něj získávat instance potřebných rozhraní (Tehdy se lokátor chová jako Factory nebo Service Locator [18]). To by ale mělo být použito jen v krajních případech, kdy počet závislostí je velmi velký nebo v případě, kdy závislost je vyžadována jen výjimečně.

[7] Inversion of control - technika v programování, spočívající v přesunutí logiky vytváření závislostí na jinou třídu[18]

Příkladem může být potřeba na logické vrstvě získat informace z několika entit zároveň. K tomu je potřeba několik managerů, které mají složité rozhraní s několika typovými parametry a konstruktor by tak byl velmi složitý.

Dalším příkladem je pak použití závislosti v jedné raritně používané metodě. Tehdy je zbytečné vyžadovat tuto závislost v konstruktoru a je lepší, i z hlediska výkonu, si o ni zažádat, až když bude potřeba.

V případě, kdy třída vyžaduje referenci na rozhraní třídy `Locator`, by mělo být v komentáři uvedeno, jaké rozhraní z něj bude získávat, pro případnou snazší dohledatelnost a testování tříd. V aplikaci pro OS Android se obdoba těchto třídy jmenuje `Factory`.

Každý `Locator` bude při svém vytvoření vyžadovat předání rozhraní `IApplicationContext`, tedy aplikačního kontextu, pomocí kterého bude moci rozpoznat, nad jakou databází a případně nad jakým sportem má aktuálně pracovat.

Na každé vrstvě bude existovat abstraktní třída, která toto rozhraní bude implementovat. Abstraktní třídy budou za uživatele řešit předávání aplikačního kontextu z vyšších vrstev a reakce na jeho změnu (například znovu vytvoření instancí pro správnou databázi). Také budou poskytovat mechanizmus pro ukládání již vytvořených a vyřešených požadavků na rozhraní. Tento mechanizmus však vývojář nebude muset využívat.

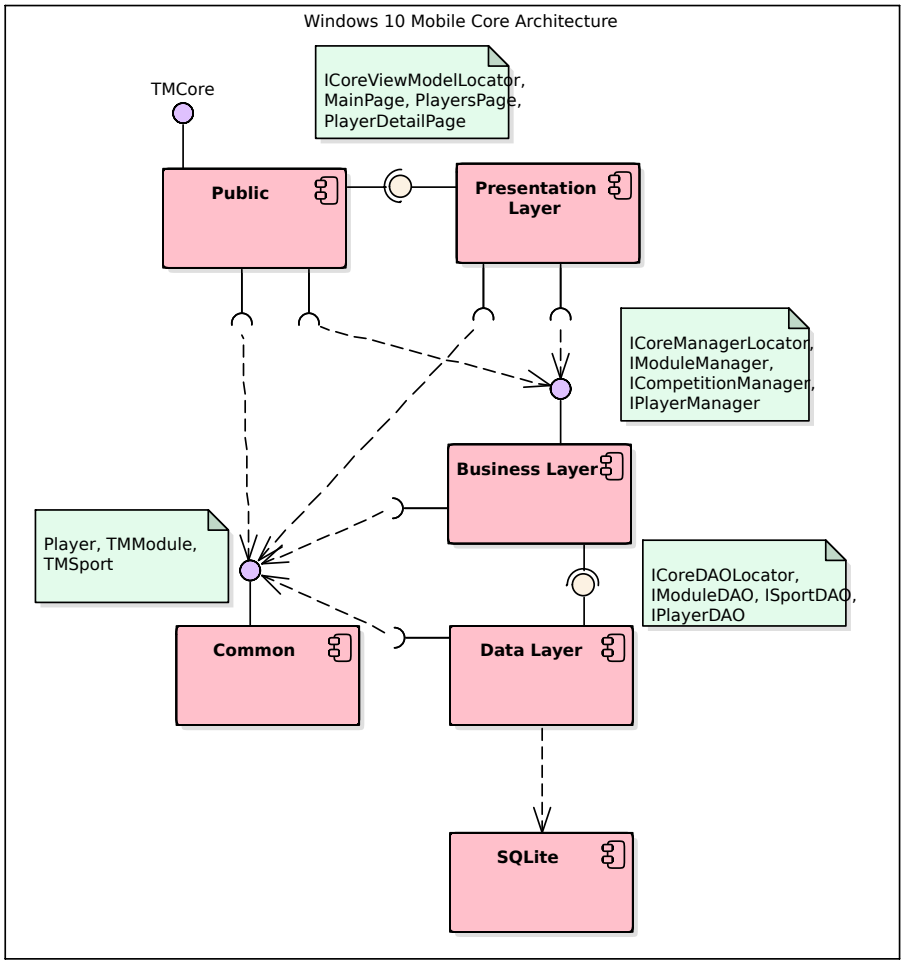
Datová vrstva Na této vrstvě to bude rozhraní `IDAOLocator`, které bude poskytovat metodu pro získání instance DAO třídy na základě typu rozhraní a typu entity, které budou předány jako typové parametry této metodě. Datová vrstva bude obsahovat abstraktní třídu s implementací tohoto rozhraní. Ta bude zajišťovat dodání potřebných závislostí pro vytvoření DAO tříd, což je zejména třída `DBConnectionFactory`.

Logická vrstva Zde je to rozhraní `IManagerLocator`, které poskytuje několik generických metod pro získání různých typů tříd. Jsou to metody pro získání entitních managerů, validátorů, serializátorů a dalších pomocných tříd.

Prezentační vrstva V této vrstvě je to rozhraní `IViewModelLocator`, které poskytuje metodu pro získání `ViewModel` tříd podle předaného typu. V případě tohoto `Locator` rozhraní bude pravděpodobně vždy vrácena nová instance `ViewModel` třídy, jelikož o cachování stránek se stará navigační služba aplikace.

Komunikace jednotlivých vrstev bude realizována přes tato rozhraní směrem od prezentační vrstvy k datové vrstvě, kdy každá vrstva si vytvoří `Locator` vrsty, kterou bude chtít využít.

Prezentační vrstva vytvoří `IViewModelLocator`. Ta, aby mohla vytvářet `ViewModel` třídy, které jsou závislé na managerech z logické vrstvy, si vytvoří



Obrázek 3.9: Architektura jádra aplikace.

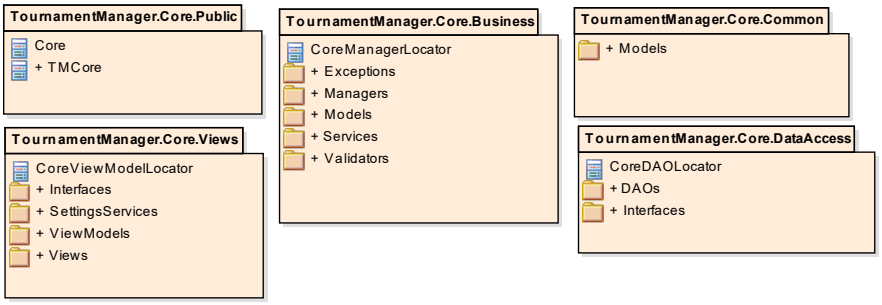
`IManagerLocator`, který tyto závislosti uspokojí. `IManagerLocator` pak vytvoří `IDAOLocator` z datové vrstvy, aby mohl uspokojit požadavky na vytvoření managerů. Tímto způsobem dojde k propojení vrstev a jejich komunikaci.

3.5 Návrh jádra

Tato sekce se bude zabývat návrhem jádra aplikace. Z obrázku je patrné, že se jedná o více vrstvou architekturu s pár zvláštnostmi, které budou dále popsány.

Na obrázku 3.9 můžeme vidět datovou vrstvu, která je rozdělena na dvě části. První je tzv. DAL[8] část, která se stará o ukládání dat do SQLite data-

[8]Data Access Layer - vrstva pro přístup k datům



Obrázek 3.10: Diagram tříd pro jádro aplikace.

báze pomocí knihovny popsané v sekci. Druhá je společná, tzv. Common část, která obsahuje entity. O Datové vrstvě se dá říct, že je částečně relaxovaná, jelikož její část, která zajišťuje ukládání do databáze a práci s databází smí použít pouze nadřazená logická vrstva, kdežto společnou část obsahující entity mohou použít i všechny vrstvy nad touto vrstvou.

Dále je na obrázku architektury k vidění standardní logická vrstva, která používá datovou vrstvu.

Následně logickou vrstvu pak využívá prezentační vrstva, která je také rozdělena do dvou částí. Tou první je standardní prezentační část obsahující třídy typu `View` a `ViewModel` pro zobrazení obrazovek uživateli, podle návrhového vzoru MVVM popsaného v sekci seznámení se s platformou UWP. Druhou částí je veřejná, tzv. public část. Ta slouží jako přístupový bod pro spouštěcí část aplikace a stará se o přesměrování úkonů na prezentační vrstvu a logickou vrstvu.

Každá z těchto jednotlivých částí bude popsána v následujících sekcích, včetně krátkého popisu tříd, které obsahují. Celkový diagram tříd jádra aplikace je možné vidět na obrázku.

3.5.1 Společná vrstva

Společná vrstva je součástí datové vrstvy a obsahuje zejména entity. Tato vrstva pro získání základních implementací využívá tříd a rozhraní ve stejně pojmenované části knihovny.

Konkrétně bude obsahovat následující entity.

Player Entita hráče, kterou podle funkčních požadavků musí evidovat jádro, obsahuje informace specifikované doménovým modelem.

TMSport Entita sportu, která slouží k uložení informací o sportu, aby jádro nemuselo kvůli názvům sportů načítat vždy všechny moduly. Každá entita sportu má entitu Modulu, ke které patří.

TMModule Entita modulu, která slouží k uložení informací o modulu jako je verze modulu a název modulu. Na základě těchto informací pak bude možné rozhodnout o znovunačtení informací ke sportům modulu.

3.5.2 Datová vrstva

Datová vrstva bude poskytovat DAO třídy, pomocí kterých bude moci logická vrstva dále přistupovat k entitám uložených v databázi. Budou to třídy, které budou potomky základních generických DAO tříd v knihovně a jako typové parametry budou mít nastaveny entity ze společné vrstvy. S těmi tedy tyto DAO třídy budou pracovat.

Konkrétně jsou to třídy `ModuleDAO`, `SportDAO` a `PlayerDAO` a jejich rozhraní. Oproti svým předkům bude mít navíc metodu jenom třída `SportDAO` pro získání sportu podle názvu modulu a identifikátoru sportu.

3.5.3 Logická vrstva

Logická vrstva jádra bude obsahovat třídy typu `Manager` starající se o práci s entitami. Pro jejich ukládání a načítání z databáze pak využijí příslušné DAO pro danou entitu. Pro implementaci těchto tříd rovněž budou využity základní třídy a rozhraní z knihovny.

Hlavní třídou v této vrstvě je `ModuleManager`. Ten poskytuje metody pro práci s moduly a jejich sporty. Také se stará o načtení modulů a sportů na základě předaného seznamu knihoven veřejných částí modulů ze spouštěcí části aplikace. Tento proces je popsán v následující sekci.

3.5.3.1 Proces načtení a inicializace modulů a sportů

Proces začíná tím, že třída TMCore ve veřejné části prezentační vrstvy jádra z instance třídy `IManagerLocator` získá instanci rozhraní `IModuleManager` a na tom spustí metodu pro načtení a inicializaci modulů.

Manager dostane na vstup metody seznam knihoven veřejných částí modulů. Manager zjistí, jestli od posledního spuštění byla aplikace aktualizována, jestli byl změněn jazyk aplikace[9] či jestli je databáze modulů prázdná. Pokud je některá z těchto podmínek splněna, Manager pokračuje načítáním modulů, jinak pokračuje inicializací povolených sportů, načtených z databáze.

Pro načítání modulů manager postupně prochází seznam knihoven modulů a pro každý modul spouští asynchronní proces načítání modulu. Po spuštění načítání všech modulů počká na jejich dokončení a pokračuje inicializací povolených sportů.

Proces načtení modulů začíná tím, že je načtena knihovna veřejné části modulu, podle předaného názvu. Z názvu knihovny veřejné části modulu je

[9] Jestli byl změněn jazyk aplikace, je nutné provést načtení názvů sportů do databáze pro správné zobrazení jmen sportů.

zjištěn název jmenného prostoru modulu. V tomto jmenném prostoru je pak nalezena třída s názvem `TMModule` implementující rozhraní `IModule` a je vytvořena její instance, která je zároveň uložena do seznamu spuštěných modulů. Následně jsou podle jmenného prostoru modulu načteny informace o modulu z databáze ve formě entity `TMModule`. Pokud tam záznam o tomto modulu není, je vytvořen, uložen a proces pokračuje načtením sportů modulu. Pokud tam záznam je a pokud došlo ke změně verze modulu, je modul aktualizován a proces pokračuje načtením sportů modulů. Pokud však ke změně verze nedošlo, pokračuje proces načtením sportů pouze, pokud došlo ke změně jazyka aplikace. Jinak proces pokračuje inicializací povolených sportů.

Proces pokračuje načtením sportů modulu. Na instanci rozhraní `IModule` je zavolána metoda pro získání všech sportů modulů. Pro každý sport je pak vyhledán záznam v databázi. Pokud sport v databázi je, jsou aktualizována jeho data v entitě `TMSport` a pokud je sport povolený, je na na instanci rozhraní `IModule` spuštěna inicializace sportu a sport je přidán do seznamu spuštěných a inicializovaných sportů. V rámci inicializace sportu je do modulu předána instance třídy `ICore`, aby mohl sport i modul komunikovat s jádrem. Pokud sport v databázi není je vytvořen a uložen jako nepovolený a proces pokračuje zpracováním dalšího sportu.

Po načtení modulů a jejich sportů proces pokračuje inicializací povolených sportů. Z databáze jsou získány všechny povolené sporty. Pro každý povolený sport je zjištěno, jestli už je v seznamu spuštěných a inicializovaných sportů. Pokud ano, je přeskočen, pokud ne, je vyhledána či vytvořena instance jeho modulu, je spuštěna jeho inicializace a je uložen do seznamu inicializovaných sportů. Tím je proces načtení a inicializace dokončen.

Třída `TMCore` poté podle toho, jestli je počet spuštěných sportů nulový, přesměruje aplikaci na zobrazení uvítací obrazovky a nebo na zobrazení hlavní obrazovky s přehledem sportů a soutěží.

Tento proces zajistí správné načtení modulů a sportů a jejich inicializaci. Manager prochází a načítá všechny moduly, pouze při prvním spuštění, aktualizaci aplikace či změně jazyka aplikace. Při dalším běžném spuštění už bude vědět, které moduly a sporty bude spouštět a nepoužívané moduly tak nebudou vůbec načteny do paměti. Při použití jediného modulu z patnácti sice zbylých čtrnáct modulů bude zabírat paměť v zařízení, ale při spuštění a chodu aplikace nebudou nijak načítány do operační paměti. Tím je tedy alespoň částečně splněn požadavek na možnost výběru modulů, které uživatel chce používat.

3.5.4 Veřejná vrstva

Veřejná část bude obsahovat třídu, která zajistí inicializaci jádra, předání seznamu modulů do logické vrstvy a případné přesměrování navigace na stránku do správného modulu či do vlastní prezentační vrstvy.

Na obrázku je možné vidět, že tato část by mohla být součástí prezentační části prezentační vrstvy. Byla však oddělena, pro případ budoucího převedení jádra aplikace do vlastní aplikace. Tehdy by stačilo pouze nahradit projekt této veřejné části spouštěcí částí aplikace, tu napojit na prezentační část a logickou vrstvu, stejně jako je napojena veřejná část. Tím by mělo být případné převedení do vlastní aplikace hotovo. Jedná se tedy o podobný vzor, který byl použit i v návrhu struktury aplikace, kde bylo použito podobné oddělení spouštěcí části od jádra aplikace, avšak z trochu jiného důvodu než zde.

3.5.5 Prezentační vrstva

Prezentační vrstva jádra aplikace bude obsahovat obrazovky, které jsou tvořeny dvojicemi tříd `View` a `ViewModel`. Jejich funkce již byla popsána v návrhu uživatelského rozhraní analytické části práce. Pro ukázku zde bude popsán návrh a fungování dvou ukázkových stránek. Zbylé stránky budou fungovat na stejném principu.

MainPage Hlavní stránka aplikace, zobrazuje seznam soutěží sdružený podle sportů. Ve `ViewModel` části této stránky bude pomocí instance rozhraní `IModuleManager` získán seznam všech sportů, seskupených s jejich soutěžemi. Soutěže budou realizovány entitou z logické vrstvy jádra, implementující rozhraní `ICompetition` z knihovny aplikace. Takto sdružená data budou předána k zobrazení ve sdruženém seznamu. Také bude třídě `View` poskytnuta sada příkazů pro práci se soutěží a sportem ze seznamu. `View` pak příkazy bude volat a jako parametr bude předávat zvolenou soutěž či sport.

FirstStartPage Stránka pro první spuštění aplikace, je zobrazena, když po spuštění aplikace není žádný sport povolen pro použití. Ve `ViewModel` části této stránky bude pomocí instance rozhraní `IModuleManager` získán seznam všech entit `TMSport`, které budou předány k zobrazení části `View`. `View` pomocí přepínačů bude sledovat označení uživatelem. Jakmile uživatel změní hodnotu přepínače, `View` o tomto kroku bude informovat `ViewModel`, ten změní hodnotu povolení daného sportu a rozhodne, jestli může být formulář potvrzen. Změny hodnot v seznamu sportů a změny hodnot indikující umožnění potvrzení formuláře sleduje třída `View` pomocí tzv. bindování[10] a přizpůsobuje jim obsah stránky.

3.5.6 Komunikace vrstev

Pro komunikaci budou využity třídy typu `Locator`, které se nachází na každé vrstvě a jejich úkoly jsou více popsané v návrhu knihovny aplikace.

[10]Binding - metoda pro propojení zobrazovací a datové části aplikace, tak že při změně hodnoty v jedné se automaticky změní ta druhá. [19]

Komunikace mezi vrstvami probíhá směrem od shora dolů a začíná u třídy `TMCore` ve veřejné části jádra.

Inicializace jádra a provázání jeho vrstev tedy bude probíhat následovně. Třída `TMCore`, na které spouštěcí část aplikace spustí inicializaci, si vytvoří aplikační kontext pro jádro, ve kterém definuje databázi, s kterou jádro bude pracovat. Následně vytvoří instanci třídy `CoreViewModelLocator` z prezentační vrstvy a předá jí aplikační kontext. Ta si kontext uloží, vytvoří instanci třídy `CoreManagerLocator` a také jí předá kontext. `CoreManagerLocator` si také kontext uloží a vytvoří instanci třídy `CoreDAOLocator` a rovněž ji předá kontext. Nyní jsou tedy všechny vrstvy provázané a téměř připravené pracovat.

Ještě je nutné, aby třída `TMCore` vytvořila a inicializovala instanci třídy `Core`, která implementuje rozhraní `ICore`, sloužící modulům pro komunikaci s jádrem. Tu následně předá do instance `CoreManagerLocator`(získanou z instance třídy `CoreViewModelLocator`) a ta ji bude dále distribuovat do třídy `ModuleManager`, který ji předá modulům při inicializaci.

Po těchto úkonech je jádro inicializované a připravené začít inicializovat a spouštět moduly, což je proces, který již byl popsán výše, v návrhu logické vrstvy aplikace.

Třídy typu `Locator` v datové vrstvě a v logické vrstvě budou vytvářet třídy až o ně bude požádáno a již vytvořené instance budou uschovány pro další žádosti. Třída `CoreViewModelLocator` na prezentační vrstvě pak bude vytvářet pro každou žádost o instanci vždy nové instance, z důvodů popsaných v sekci o návrhu knihovny.

3.6 Návrh modulu

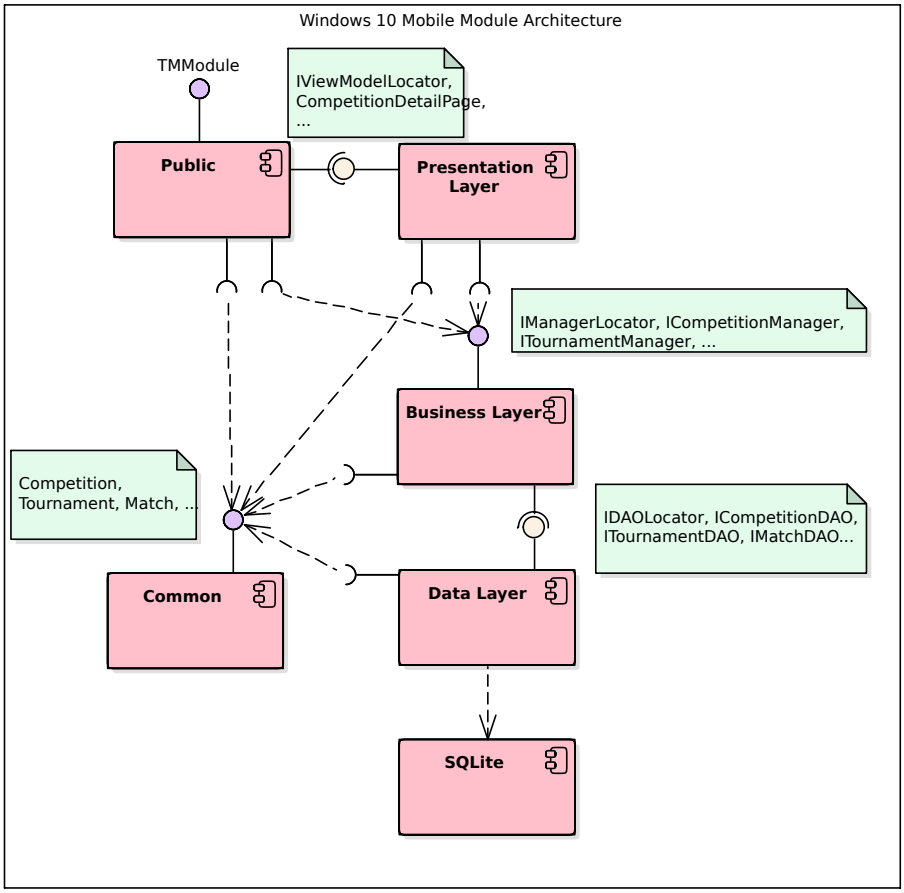
Pro návrh a realizaci jednoho vzorového modulu byl zvolen modul pro sporty typu hokej, který byl pro OS Android vytvořen v rámci práce Ondřeje Košuta[2] a poté byl upraven v práci Josefa Němečka[1].

Aby spolu moduly byly kompatibilní pro případnou synchronizaci, bude návrh modulu inspirován tímto modulem, zejména co se týče entit a databázového modelu.

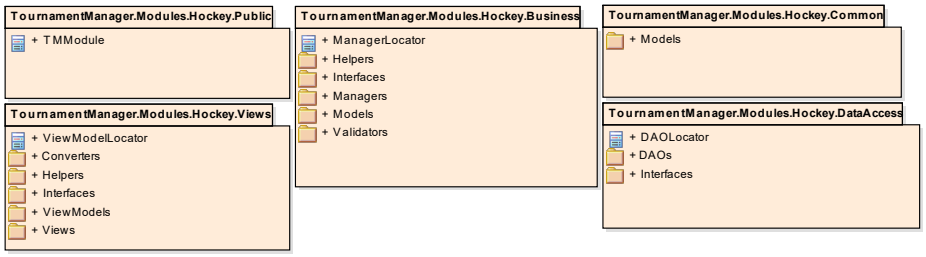
Architektura modulu, kterou je možné vidět na obrázku **??**, je ovšem od modulu pro OS Android odlišná, ale je velmi podobná architektuře jádra aplikace, která je popsaná v sekci 3.5. Od jádra aplikace se liší pouze třídou ve veřejné části prezentační vrstvy, která slouží jádru jako rozhraní pro komunikaci s modulem. Také se samozřejmě liší obsahem jednotlivých vrstev.

Jelikož se sporty hokej a florbal od sebe z hlediska doménového modelu liší pouze názvy, nebude potřeba v aplikaci brát ohledy na přepínání chování tříd či vyměňování tříd v závislosti na aktuálně zpracovávaném sportu.

Celkový diagram tříd modulu je možné vidět na obrázku



Obrázek 3.11: Architektura modulu aplikace.



Obrázek 3.12: Diagram tříd pro modul aplikace.

3.6.1 Společná vrstva

Společná vrstva bude obsahovat entity, které vzniknou děděním z entit definovaných v knihovně aplikace. Všechny entity mají stejné názvy a vlastnosti jako entity v knihovně i modulu pro OS Android.

Stejně jako v modulu pro OS Android budou oproti entitám v knihovně upraveny pouze entity pro statistiky. Do těch budou doplněny statistické údaje, které bude potřeba evidovat.

Pro entitu `PlayerStat` jsou to následující vlastnosti.

Assists Asistence hráče.

Goals Počet gólů.

PlusMinus Kanadské bodování.

Saves Zákroky hráče.

Poté jsou to tyto dvě vlastnosti pro entitu `Match`.

Overtime Boolean jestli se hrálo v prodloužení.

Shootouts Boolean jestli byly samostatné nájezdy.

A na závěr je to vlastnost `Score` pro entitu `ParticipantStat` a vlastnosti entity `PointConfiguration`, kdy každá vlastnost určuje počet bodů za určitý stav zápasu. Takže např. počet bodů za výhru v normálním čase nebo počet bodů za prohru na nájezdy.

3.6.2 Datová vrstva

Datová vrstva bude obsahovat DAO třídy pro všechny entity. Ve směs budou všechny vytvořeny děděním z DAO tříd knihovny bez větších úprav. Za to je možné vděčit návrhu DAO tříd v knihovně, které díky využití generických parametrů pro entity umožňují použití těchto tříd beze změn. Existenci vlastních DAO tříd na této vrstvě je však dobré zachovat, zejména z důvodu budoucí možnosti doplnění funkčnosti.

3.6.3 Logická vrstva

V logické vrstvě modulu budou děděním z knihovny aplikace vytvořeny entitní třídy typu Manager. Ty nebudou muset být modifikovány, až na pár výjimek, týkajících se statistik a specifických modulových záležitostí. Příkladem může být metoda pro vygenerování soupisek týmů ve třídě `TeamManager`, která po podědění z knihovny umí generovat soupisky pouze náhodně. Zde bude nutné doplnit podle jakých statistik a jak seřazených má generátor soupisek pracovat.

Dále v logické vrstvě budou použity validátory. Ty opět budou poděděny z knihovny aplikace a jako generické parametry jim budou doplněny entity ze

společné vrstvy modulu. Zde také budou muset být doplněny místa týkající se statistik. Jedná se zejména o výchozí nastavení validačních entit při jejich vytváření.

Také budou do logické vrstvy doplněny komplexnější třídy pro práci se statickými entitami. Bude to třída `StatisticsHelper`, která bude podpořena entitami `AggregatedStat` a `Standing` a bude se starat o agregaci statistik ze sportu, soutěží a turnajů. Velmi podobné konstrukty jsou k vidění i v modulu pro OS Android a nebudou zde tedy dále podrobně popisovány.

3.6.4 Prezentační vrstva

V prezentační vrstvě budou řešeny jednotlivé obrazovky modulu. Ty budou odpovídat obrazovkám navrženým v návrhu uživatelského rozhraní.

Modul bude řešit obrazovky od detailu soutěže až po detail zápasu, včetně všech stránek pro formuláře jako je přidání hráče do týmu, editace statistik hromadná, jednotlivá, atd. Každou stránku bude tvořit dvojice tříd `View` a `ViewModel`, stejně jako je tomu v jádře aplikace. Jednotlivé `View` budou postupně skládány pomocí předdefinovaných prvků a komponent z knihovny aplikace a k nim budou tvořeny `ViewModel` třídy, které pro ně budou připravovat data.

Každá třída typu `View` bude implementovat rozhraní `IView`, ve kterém definuje typ třídy `ViewModel`, který pro ní má být vytvořen.

3.6.5 Veřejná vrstva

Veřejná vrstva modulu bude obsahovat třídu, která bude implementovat rozhraní `IModule` definované ve společné části knihovny, zabývající se architekturou aplikace.

Tato třída se musí jmenovat `TMModule`, aby mohla být správně nalezena jádrem aplikace. Z toho plyne, že každý vývojář bude povinen vybavit modul touto veřejnou vrstvou a vytvořit v ní třídu s názvem `TMModule`, implementující rozhraní `IModule`.

Při vytvoření instance této třídy jádrem aplikace bude vytvořena třída typu `ServerPackage`, popsaná v návrhu knihovny. Následně budou vytvořeny dvě instance třídy `Sport`, rovněž popsané v knihovně, pro sporty hokej a florbal. Těmto dvěma instancím třídy `Sport` bude předána instance třídy `ServerPackage`. Tím bude modul připraven na inicializaci sportů ze strany jádra aplikace.

Jádro si po objevení modulu a jeho vytvoření vyžádá všechny jeho sporty a ty povolené uživatelem začne postupně inicializovat.[11] Pro každý inicializovaný sport bude v třídě `TMModule` vytvořen kontext aplikace s daným sportem a bude provedena inicializace provázání vrstev popsaná v následující sekci.

[11] Tento postup je popsán v sekci návrhu logické vrstvy jádra aplikace 3.5.3.1

3.6.6 Komunikace vrstev

Na závěr bude potřeba vrstvy provázat a k tomu budou sloužit třídy typu `Locator` na jednotlivých vrstvách tak, jak je popsáno v návrhu komunikace vrstev v knihovně aplikace.

Na datové vrstvě tedy bude potřeba vytvořit třídu `DAOLocator`, která bude DAO třídy poskytovat logické vrstvě. Ta vznikne děděním základní třídy z knihovny aplikace a bude nutné implementovat vytvoření instancí pro každou DAO třídu, její uložení pro další použití a přetížení metody pro získání instance, do které bude doplněno ověření jestli tato instance třídy vyhovuje požadavku a její případné vrácení.

Na logické vrstvě bude vytvořena entita `ManagerLocator`, která bude tyto třídy vytvářet a na žádost o jejich rozhraní je bude vydávat. Způsob, jakým bude fungovat, bude velmi podobný způsobu fungování třídy `DAOLocator`. Rovněž bude vytvořena děděním ze základní třídy v knihovně aplikace. Třída `ManagerLocator` mimo jiné také vytvoří instanci třídy `DAOLocator`, kterou bude potřebovat pro plnění konstruktorů `Manager` tříd.

Na prezentační vrstvě pak bude vytvořena třída `ViewModelLocator` děděním ze základní třídy v knihovně aplikace a bude se starat o vytváření všech tříd typu `ViewModel` v prezentační vrstvě. Bude nutné do ní zaregistrovat všechny třídy typu `ViewModel` a při dotazu na určitý typ bude nutné najít správný `ViewModel`, vytvořit ho a vrátit.

Při inicializaci propojení vrstev se nejdříve vytvoří instance třídy `ViewModelLocator`, které se předá instance rozhraní `ICore` pro komunikaci s jádrem a aplikační kontext. Instance třídy `ViewModelLocator` si tyto údaje uloží a vytvoří instanci třídy `ManagerLocator` a předá jí ty samé údaje. Instance třídy ManagerLocator pak vytvoří instanci třídy `IDAOLocator` a také jí předá kontext a instanci rozhraní `ICore`. Tím je dokončena inicializace vrstev a modul je připraven vytvářet instance tříd typu `ViewModel`, které vyžadují instance typu `Manager`, `Validator` a další z logické vrstvy a ty zase vyžadují instance typu `DAO`. Všechny tyto instance nyní mohou být vytvořeny jednotlivými lokátory.

3.7 Postup pro vývoj modulu a jeho distribuci do aplikace

Vzhledem ke zvolené architektuře aplikace, kdy je řešena formou jedné aplikace, je také nutné navrhnout způsob, jakým budou vývojáři vytvářet moduly a distribuovat je do distribuční verze aplikace.

Pro snazší vývoj modulu vznikne tzv. SDK. To bude obsahovat jádro a knihovnu aplikace ve zkompilovaném tvaru. Dále bude obsahovat základní spouštěcí část aplikace, která se postará o inicializaci jádra.

Aby byl vývoj dalších modulů pro vývojáře jednodušší, bude k SDK přidán zde vytvářený modul pro hokej, který bude spíše v neuceleném stavu a bude

sloužit jako velká ukázka kódu. Pro tento modul v SDK bude snaha ukázat co největší množství postupů a ukázek, jak lze řešit různé situace.

Pro vytvoření vlastního modulu pak bude dodána šablona pro vytvoření projektové struktury modulu do vývojového softwaru Visual Studio. Vytvořená struktura modulu bude prázdná a bude mít pouze správně nastavené reference na jádro a knihovnu aplikace.

Vývojář pak bude postupovat tím způsobem, že si rozbalí SDK balíček a do Visual Studia si přidá rozšíření pro generování struktury modulu. Dále ve Visual Studio otevře Solution rozbaleného SDK a vygeneruje si prázdnou strukturu pro vlastní modul. Pak začne se samotnou implementací, během které bude prohlížet kód ukázkového modulu a bude si odtud brát inspiraci, či přímo kusy kódů podle toho co zrovna bude tvořit ve svém modulu.

Pro správné zapojení modulu do jádra bude vývojář nejdříve muset ve veřejné části modulu vytvořit třídu s názvem `TMModule`, která bude implementovat rozhraní `IModule`. V jejím konstruktoru vytvoří sporty a přidá je do seznamu sportů ve třídě `TMModule`.

Poté zaregistruje název projektu veřejné části modulu do seznamu modulů ve spouštěcí části aplikace. Jakmile se mu podaří, aby se jeho sporty zobrazili v obrazovce nabídky sportů, může začít řešit funkcionalitu svého modulu. Je doporučeno začít od společné části (tedy entit) a postupovat po vrstvách směrem vzhůru, až k veřejné vrstvě.

SDK vývojář najde ke stažení na synchronizačním serveru, kde si bude muset také zaregistrovat svůj modul, aby získal potřebné unikátní identifikátory. Tento proces registrace a získání SDK je více popsán v práci Michala Hacury [4].

Jakmile vývojář modul dokončí, zabalí zdrojový kód modulu a odešle ho správci distribuční verze aplikace (buď emailem, nebo pomocí jiného mechanizmu). Ten zdrojový kód projde a případně vrátí k opravení kritických chyb. Poté správce distribuční verze nasadí modul vývojáře do verze aplikace určené k testování a bude jí distribuovat testerům a uživatelům, kteří budou mít zájem o testovací verzi aplikace. Po určité době, kdy už bude modul vyladěn a schválen uživateli, bude ho moct správce aplikace zabudovat do distribuční verze aplikace.

Pokud vývojář nebude chtít modul předat do distribuční verze a bude ho chtít používat pouze s přáteli, může si zkompilovat SDK s vlastním modulem a tuto zkompilovanou verzi SDK může svým přátelům předat jako aplikaci.

Kapitola 4

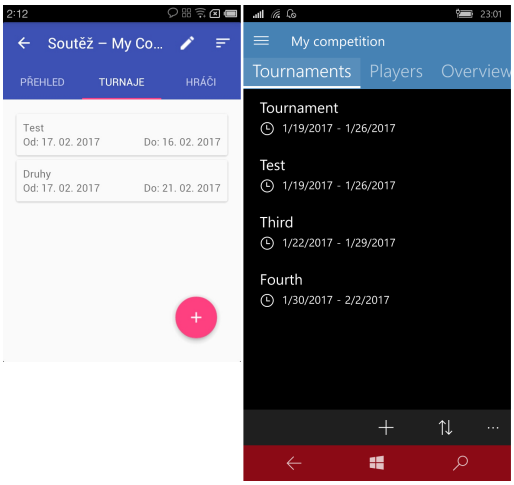
Realizace

V této kapitole bude ukázán výsledek realizace aplikace s ukázkou kódů zajímavých částí. Nakonec bude uvedeno pár statistických údajů z vývoje.

4.1 Ukázka GUI aplikace

Zde je na obrázku 4.1 vidět srovnání GUI aplikace pro OS Windows 10 Mobile ve srovnání s GUI Aplikace pro OS Android.

Aplikace jsou si dostatečně podobné a uživatel by neměl mít problém pracovat s oběma. Aplikace se také podobá aplikaci MSN Sport, jejíž snímek obrazovky je vidět v sekci 2.2 rešerše existujících aplikací.



Obrázek 4.1: Vlevo - GUI aplikace pro Android. Vpravo - GUI aplikace pro W10M

4.2 Ukázka kódu

Pro ukázky kódu byly zvoleny následující zajímavé části aplikace, které byly detailněji popsány v návrhu aplikace.

4.2.1 Rozhraní `IModule` v modulu pro hokej

V této sekci je k vidění ukázka implementace třídy základní a nejnutnější implementace třídy `TMModule`, tak jak je implementována v ukázkovém modulu pro sporty typu hokej. K implementaci byla využita základní třída poskytnuta knihovnou aplikace `TMModuleBase`. V ukázkách jsou postupně vidět konstruktor 4.1 a základní metody 4.2.

Vývojáře pak čeká kromě implementace těchto základních metod pro fungování modulu, zbytek metod definovaných v rozhraní `IModule`.

4.2.2 Obrazovka pro přidání hráče do soutěže

Zde je ukázána část `View` pro obrazovku modulu sloužící pro přidání hráče do soutěže. V ukázce 4.3 je vidět využití několika komponent, definovaných v knihovně aplikace. Díky tomu je zápis jinak poměrně složité obrazovky celkem jednoduchý.

Část `View` se skládá ze dvou částí. První je XAML část, která definuje vzhled a složení obrazovky a je vidět v ukázce. K ní ještě patří část, psaná v jazyce C#, tzv. codebehind, která se stará o řešení událostí jednotlivých částí obrazovky či řízení stavů a vzhledů jejích částí. Většinou je tato část nevyužita a obsahuje pouze definici typu třídy `ViewModelu`, který je pro `View` použit.

4.3 Vytvořené SDK

V rámci realizace aplikace bylo také vytvořeno SDK tak jak bylo popsáno v sekci 3.7. Balíček SDK je možné nalézt na přiloženém CD.

Pro další usnadnění práce vývojáři s registrací modulu do aplikace, bylo v projektu spouštěcí části SDK přidáno nastavení MSBuild, které se provede před každou kompilací. Automaticky před každou kompilací spouštěcí části aplikace projde složku se zkompilovanými knihovnami a najde knihovny, které začínají názvem `TournamentManager.Modules` a končí názvem `Public.dll`. Názvy těchto nalezených knihoven automaticky zaregistruje do kolekce modulů ke spuštění ve třídě `ModuleRegister`. To dělá tak, že před samotnou kompilací vygeneruje celý soubor s touto třídou.

Na vývojáře tak zbývá pro připojení modulu k jádru pouze přidat referenci na projekty veřejné a prezentační části modulu do spouštěcí části SDK, aby byl tento modul zkompilován před spouštěcí částí. Prezentační část modulu

Listing 4.1: Konstruktor třídy TMModule

```
//TMModule.cs

protected override Assembly CurrentAssembly { get; }
    = typeof(TMModule).GetTypeInfo().Assembly;

protected ServerPackage Package;

public TMModule()
{
    var localizationService =
        new LocalizationService(CurrentAssembly.GetName().
            Name);

    Package = new ServerPackage()
    {
        APIKey = "abcdefghijklmnopqrstuvw",
        Name = "uwp-hockey",
        DetailUrl = "testUrl",
    };

    Sports.Add(new Sport(Package, "hockey")
    {
        DisplayName =
            localizationService.GetString("uwp-
                hockey_HockeyDisplayName"),
    });

    Sports.Add(new Sport(Package, "florbal")
    {
        DisplayName
        = localizationService.GetString("uwp-
            hockey_FlorbalDisplayName"),
    });
}
```

musí být ze spouštěcí části referencována také, kvůli nutnosti objevení všech stránek přímo ze spouštěcí části aplikace.

4.4 Lokalizace Aplikace

Aplikace byla lokalizována do češtiny a angličtiny. Celkem je v jádře aplikace přes 50 textů k lokalizaci a v modulu pro hokej je pak přes 250 textů.

Texty se nacházejí v souborech s názvem `Resources.resw`, ve složce jménem `String` a v ní ve složkách pro jednotlivé kultury.

Pro použití odlišných popisků pro různé sporty je využito přepínačů altform. Soubor pojmenovaný `Resources.altform-sportID.resw` kde `sportID` bude identifikátor sportu, bude načten po přepnutí kontextu na daný sport a budou zobrazovány texty z tohoto souboru. Do souboru není nutné dávat všechny popisky, ale stačí jenom ty změněné oproti hlavní souboru `Resources.resw`. O přepnutí přepínače `altform` se stará třída `TMModule`, při přepínání kontextů.

Pro zjednodušení lokalizace textů bylo využito rozšíření pro VisualStudio jménem Multilingual App Toolkit. To umožňuje přeložit texty strojově pomocí překladače a pak už stačí jen opravit případné chyby. Také sleduje změny v souboru ve výchozím jazyce a v ostatních jazycích pak umožňuje zobrazit pouze změněné texty, což značně usnadňuje překladatelskou práci.

Listing 4.2: Základní metody třídy TMModule.

```
 //TMModule.cs
 ...
protected IViewModelLocator ViewModelLocator
        { get; set; }

/// <summary>
/// Updates context in Locators.
/// </summary>
protected override void SetCurrentContextToLocators(
    IApplicationContext newContext)
{
    ViewModelLocator.CurrentContext = newContext;
}

/// <summary>
/// On Sport Initialization creates ViewModelLocator if there
    is none present.
/// </summary>
protected override void InternalSportInitialize(
    ICore coreInstance,
    IApplicationSettings applicationSettings,
    string sportId, IApplicationContext currentContext)
{
    ViewModelLocator =
        ViewModelLocator ?? new ViewModelLocator(
                                applicationSettings,
                                coreInstance,
                                currentContext);
}

/// <summary>
/// Resolves page's viewModel
/// </summary>
protected override INavigable ResolveForPageInternal(
    Page page, INavigationService navigationService)
{
    var typedPage = page as IView;

    if (typedPage != null)
    {
        return
          ViewModelLocator.GetViewModel(
              typedPage.ViewModelType);
    }

    return null;
}
...
```

Listing 4.3: Základní metody třídy TMModule.

```
//AddCompetitionPlayerPage.xaml

<Page
    x:Class="TournamentManager.Modules.Hockey.Views.Views.
        AddCompetitionPlayerPage"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/
        presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:d="http://schemas.microsoft.com/expression/blend
        /2008"
    xmlns:lc="using:TournamentManager.Library.Views.Controls"
    xmlns:mc="http://schemas.openxmlformats.org/markup-
        compatibility/2006"
    x:Name="rootPage"
    mc:Ignorable="d">


    <lc:SimpleFormPageControl
        x:Uid="AddCompetitionPlayer"
        CancelClick="{x:Bind ViewModel.NavigateBack}"
        SubmitClick="{x:Bind ViewModel.SubmitForm}">
        <StackPanel Margin="12,6">
            <ListView
                x:Name="players"
                IsMultiSelectCheckBoxEnabled="True"
                ItemsSource="{x:Bind ViewModel.Players, Mode=
                    OneWay}"
                SelectionChanged="{x:Bind ViewModel.
                    Players_SelectionChanged}"
                SelectionMode="Multiple">
                <ListView.ItemTemplate>
                    <DataTemplate>
                        <StackPanel Orientation="Horizontal">
                            <TextBlock Text="{Binding Name}"
                                />
                            <TextBlock Text=" - " />
                            <TextBlock Text="{Binding Email}"
                                TextTrimming="
                                CharacterEllipsis" />
                        </StackPanel>
                    </DataTemplate>
                </ListView.ItemTemplate>
            </ListView>

            <HyperlinkButton
                x:Uid="AddCompetitionPlayer_LinkToNewPlayer"
                Margin="12,6"
                Click="{x:Bind ViewModel.GoToCreatePlayer}"
                Content="Create new player" />
```



```
        </StackPanel>
    </lc:SimpleFormPageControl>
</Page>

...
```

Kapitola 5

Testování a dokumentace

V této kapitole bude popsáno jakými metodami byla aplikace testována a jakým způsobem byla aplikace zdokumentována.

Během všech druhů testování bylo nalezeno několik chyb, které byly postupně odstraněny. Všechny testy, kromě jednotkových byly prováděny uživatelsky několika subjekty.

5.1 Jednotkové testy

K aplikaci bylo vytvořeno několik základních jednotkových testů, které ověřují především algoritmické části aplikace.

K realizaci jednotkových textů byl vybrán framework `xUnit`, zejména kvůli jeho vlastnosti, která umožňuje spouštět testy přímo na cílovém zařízení, kde aplikace bude nasazena.

Jednotkové testy jsou strukturovány tak, že ke každé testované třídě je vytvořena testovací třída. V této testovací třídě jsou pak vytvořeny podtřídy pro každou testovanou metodu. V rámci těchto podtříd pak jsou jednotlivé testové metody. Tento vzor umožňuje například dědění podtříd, čímž mohou získat například společnou inicializaci nebo společné testové metody.

Příkladem může být jednotkový test generátoru zápasů. Ten ověřuje několik vlastností generátor jako je například ověření, že v rámci kola hraje každý s každým, že generuje správné rozdělení při lichém i sudém počtu účastníků a že správně zaměňuje strany domácích a hostů pro sudá a lichá kola. Následuje ukázka metody tohoto jednotkového testu.

5.2 Uživatelské testy

Dále byla aplikace testována uživatelskými testy, které se zaměřovaly na funkčnost aplikace.

Listing 5.1: mappingData - player_detail

```
 //RoundRobinMatchGeneratorTest.cs
[Fact]
public async Task HandlesOddCountOfParticipants()
{
    var generator = new Business.MatchGenerator.
        RoundRobinMatchGenerator<Match>();

    var result = await generator.GenerateMatches(
        Participants5, 1);

    //10 matches
    Assert.Equal(10, result.Count);

    //Assert everyone with everyone
    var participants = result.Select(i => i.Value);
    var combinations = GetCombinations(participants);

    AssertAllWithAll(Participants5, combinations);
}
```

Pro účely testování bylo vypracováno několik scénářů, které postupně ověří splnění všech funkčních požadavků. Tyto scénáře jsou k vidění na přiloženém CD a ve směs se jedná o průchod aplikací vytvářením entit, jejich editací a mazání.

Testování bylo provedeno zejména dvěma personami. První personou je muž, 25 let, student. Tato persona nemá žádné zkušenosti se systémem Windows 10 Mobile a je uživatelem mobilního zařízení se systémem Android. Testování aplikace prováděla tato persona na stolním počítači se systémem Windows 10. Druhou personou je také muž, 25 let, student. Tato persona má zkušenosti se systémem Windows 10 Mobile a je vlastníkem mobilního zařízení s tímto systémem. Testování aplikace tedy prováděla na mobilním zařízení.

Připomínky, které měly persony byly postupně zapracovány. Jednalo se většinou o nezašednutí tlačítka, které pak způsobovalo problémy.

5.3 Integrační testy

Integrační testování aplikace je rozděleno do dvou částí. První část se bude zabývat testováním aplikace na různých zařízeních a bude zkoumat zejména dodržení vzhledu a funkčnosti. Druhá část integračních testů bude zaměřena na správné fungování modulů a jádra. Bude zkoumáno několik scénářů, ve kterých bylo vytipováno, že by se aplikace nemusela chovat správně.

5.3.1 Systémové integrační testy

Aplikace byla otestována na verzi systému Windows 10 1607 a verzi Windows 10 Mobile 10.0.10586 a verzi 10.0.14393. Aplikace funguje na všech zařízeních bez obtíží.

Zajímavé je, že pro kompilaci aplikace v režimu `Debug` je použit jiný druh kompilace než pro režim `Release` a tak bylo nutné vždy testovat oba tyto režimy.

Během testování režimů vzhledu, kdy bylo na desktopovém systému měněno rozlišení okna obrazovky se občas objevil problém, kdy nebylo možné na některých obrazovkách používat tzv. scrollování, tedy posouvání obrazovky pomocí posuvníků. Všechny tyto případy však byly postupně opraveny.

5.3.2 Aplikační integrační testy

Tyto testy se zabývají zkoumáním fungování jádra a modulů aplikace. Opět bylo vytvořeno několik scénářů, u kterých bylo odhadnuto, že by mohly být problémové. I tyto testy byly prováděny uživatelsky a i zde byly objeveny nedostatky, které byly opraveny. Seznam scénářů je rovněž k nalezení na přiloženém CD.

Jednalo se například o nesprávné přepínání kontextu sportu modulu, kdy uživatel přecházel z detailu soutěže do detailu hráče, odtud do detailu soutěže v jiném sportu a opět do detailu hráče a znovu do detailu prvního sportu a nakonec do detailu hráče. Mačkáním tlačítka zpět by se měly zobrazovat soutěže ve sportech tak jak byly navštíveny, aplikace ale v tomto případě nepřepínala aplikační kontext a ten tak zůstal na posledním navštíveném sportu. I tento neduh byl v aplikaci opraven.

5.4 Dokumentace

Dokumentace kódu se píše do komentářů, ve formátu XML, nad každou veřejně přístupnou či chráněnou třídu, metodu, vlastnost a proměnnou.

Nastavení projektu umožňuje zapnout generování dokumentace ve formátu XML. Součástí této funkcionality je i upozornění vývojáře na nezdokumentovanou část kódu. Tato funkcionalita byla zapnuta u všech projektů a pokrytí dokumentace je tak velmi dobré.

Pro zjednodušení práce s vyplňováním použitých parametrů, výjimek a dědičnosti bylo použito rozšíření GhostDoc, které generuje komentáře a tento popis vyplňuje automaticky.

Výsledná dokumentace pak byla vygenerována pomocí programu Doxygen a je k vidění na přiloženém CD.

Závěr

V práci byla provedena analýza aplikace Tournament Manager pro OS Android. Po analýze byla provedena rešerše existujících řešení a ta byla následována návrhem uživatelského rozhraní a popisem doménového modelu, které jsou podobné jako v aplikaci pro OS Android. V návrhu aplikace a jednoho vzorového modulu byly provedeny nutné architektonické změny, dané omezením systému Windows 10 Mobile, kvůli kterému nebylo možné splnit nefunkční požadavek na samostatné instalování modulů. Výsledná aplikace je tak alespoň vnitřně modulární a umožňuje oddělený vývoj modulů. Aplikace byla úspěšně implementována, otestována a zdokumentována.Protipólem k nesplnění zmíněného chybějícího nefunkčního požadavku může být, spustitelnost aplikaci i na systému Windows 10 pro desktopová zařízení, čehož bylo dosaženo díky vývoji aplikaci v Univerzální Platformě Windows.

Architektura aplikace je připravena na budoucí oddělení modulů z jedné aplikace do vlastních aplikací a s příchodem aktualizace systému Windows s názvem „Anniversary Update“, ve které byly představeny rozšíření pro aplikace, se tato možnost zdá jako nutný krok v budoucím rozvoji aplikace. Mimo jiné by bylo také vhodné doplnit společný postup pro obě aplikace, na základě kterého by mohli synchronizovat soutěže se synchronizačním serverem. V případě této aplikace je tento úkol plně v rukou vývojáře modulu.

Literatura

[1] Němeček, J.: *Tournament Manager – jádro aplikace*. Diplomová práce, Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2017, [cit. 2017-01-01].

[2] Košut, O.: *Tournament Manager - balíček pro hokej*. Bakalářská práce, Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2016, [cit. 2017-01-01].

[3] Chmel, V.: *Tournament Manager - balíček pro squash*. Bakalářská práce, Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2016, [cit. 2017-01-01].

[4] Hacura, M.: *Tournament Manager - Synchronizace*. Diplomová práce, Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2017, [cit. 2017-01-01].

[5] Foley, M. J.: Microsoft releases new Windows 10 Mobile test build that's likely RTM. *ZDNet*, 2015, [cit. 2016-04-02]. Dostupné z: `http://www.zdnet.com/article/microsoft-releases-new-windows-10-mobile-test-build-thats-likely-rtm/`

[6] Ekuan, M.: What's a Universal Windows Platform (UWP) app? 2016, [cit. 2016-04-05]. Dostupné z: `https://msdn.microsoft.com/en-us/windows/uwp/get-started/whats-a-uwp`

[7] Continuum for Phone. [cit. 2016-04-05]. Dostupné z: `https://msdn.microsoft.com/en-us/library/windows/hardware/mt608594(v=vs.85).aspx`

[8] Develop UWP Apps. 2016, [cit. 2016-04-05]. Dostupné z: `https://developer.microsoft.com/en-us/windows/apps/develop`

[9] What is XAML? *Microsoft Developer Network*, [cit. 2016-04-08]. Dostupné z: https://msdn.microsoft.com/en-us/library/cc295302.aspx

[10] Lagunas, B.: The Future of the Prism Library: Meet the New Team. 2015. Dostupné z: http://brianlagunas.com/future-prism-library-meet-new-team/

[11] Nixon, J.: Template10. [cit. 2016-04-05]. Dostupné z: https://github.com/Windows-XAML/Template10/wiki

[12] Miller, R.: EntityFramework - Roadmap. [cit. 2016-04-05]. Dostupné z: https://github.com/aspnet/EntityFramework/wiki/Roadmap

[13] Croft, J.: Windows 10 (Win 10 UWP VisualStudio project) modular application (like Prism.MEF) - Answer. [cit. 2016-04-05]. Dostupné z: http://stackoverflow.com/a/34086588

[14] Possible to get installed packages in UWP10 across all users - Answer. 2016, [cit. 2016-04-05]. Dostupné z: https://social.msdn.microsoft.com/Forums/en-US/b20cef2c-98a8-4d6f-8231-db856199f2c9/possible-to-get-installed-packages-in-uwp10-across-all-users?forum=wpdevelop

[15] Hachman, M.: New Microsoft Edge extensions may have a ripple effect on Windows gaming. *PCWorld*, 2016. Dostupné z: http://www.pcworld.com/article/3042730/windows/new-microsoft-edge-extensions-may-have-a-ripple-effect-on-windows-gaming.html

[16] Data Access Object Pattern. [cit. 2017-02-02]. Dostupné z: https://www.tutorialspoint.com/design_pattern/data_access_object_pattern.htm

[17] Decorator Pattern. [cit. 2017-02-02]. Dostupné z: http://www.oodesign.com/decorator-pattern.html

[18] Blumenauer, J.: Lost in Translation: Dependency Inversion Principle, Inversion of Control, Dependency Injection & Service Locator. *Applied Information Sciences Blog*, 2013. Dostupné z: https://blog.appliedis.com/2013/12/10/lost-in-translation-dependency-inversion-principle-inversion-of-control-dependency-injection-service-locator/

[19] Data Binding. [cit. 2017-02-02]. Dostupné z: https://www.techopedia.com/definition/15652/data-binding

Příloha **A**

Seznam použitých zkratek

GUI Graphical user interface

UI User interface

OS Operating System

XML Extensible markup language

XAML Extensible application markup language

UWP Universal Windows Platform

SDK Software Development Kit

UUID Universally Unique Identifier

DAO Data Access Object

JSON Javascript Object Notation

MSN Microsoft Network

DAL Data Access Layer

WM10 Windows 10 Mobile

Příloha **B**

Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `exe` ..... adresář se spustitelnou formou implementace
- `doc` ..... dokumentace implementace `.1 src`
  - `impl` ..... zdrojové kódy implementace
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF
  - `thesis.ps` ..... text práce ve formátu PS