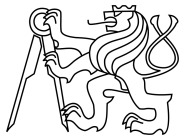
ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ



ZADÁNÍ DIPLOMOVÉ PRÁCE

Název:	Tournament Manager - Synchronizace
Student:	Bc. Michal Hacura
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

Cílem práce je vytvořit serverovou část systému Tournament Manager pro synchronizaci záznamů a statistik z více klientských zařízení a jednoduchou webovou prezentaci výsledků. Tournament Manager je aplikace především pro OS Android umožňující organizaci turnajů v různých sportech, evidování výsledků a individuálních statistik hráčů.
Cíle práce:
- Analyzujte požadavky na serverovou část z pohledu synchronizace klientů a webové prezentace.
- Proveďte rešerši dostupných hostingů a technologií pro serverovou část systému.
- Navrhněte architekturu a strukturu serverové části systému.
- Implementujte a řádně otestujte serverovou část systému pro synchronizaci včetně webové prezentace statistik a výsledků.
- Serverovou část řádně zdokumentujte, především z pohledu rozhraní pro synchronizaci.

Seznam odborné literatury

Dodá vedoucí práce.

L.S.

Ing. Michal Valenta, Ph.D.
vedoucí katedry

prof. Ing. Pavel Tvrdík, CSc.
děkan

V Praze dne 16. listopadu 2015

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

Diplomová práce

Tournament Manager - Synchronizace

Bc. Michal Hacura

Vedoucí práce: Ing. Zdeněk Rybola

17. února 2017

Poděkování

Rád bych poděkoval vedoucímu práce Ing. Zděnku Rybolovi za podporu a pomoc při vedení této práce. Dále děkuji Václavu Chmelovi, Ondřeji Košutovi, Vlastimilu Mácovi a Josefu Němečkovi za výbornou spolupráci a pohodový přístup při vývoji celého systému Tournament Manager.

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

Hacura, Michal. *Tournament Manager - Synchronizace.* Diplomová práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2017.

Abstrakt

Práce se zabývá vývojem serverové části systému Tournament Manager umožňující synchronizaci dat z více klientských zařízení a jednoduchou prezentaci výsledků na webu. Tournament Manager je systém, skládající se z klientů pro mobilní zařízení a synchronizačního serveru, umožňující organizování sportovních turnajů a evidování dosažených výsledků a statistik. V práci je zachycen kompletní proces vývoje sestávající z analýza funkčností, návrhu jednotlivých částí, implementace a testování. Výsledkem je plně funkční server připravený k integraci do zbytku systému.

Klíčová slova server, sdílení dat, synchronizace mobilních zařízení, REST, JSONPath, CodeIgniter, PHP

Abstract

This thesis focuses on development of server part of Tournament Manager system providing data synchronization across multiple client devices and simple presentation of results on web. Tournament Manager is system consisting of clients for mobile devices and synchronization server, used to organize sport tournaments and track achieved results and statistics. The thesis contains

complete process of software development consisting of functionality analysis, design of individual parts, implementation and testing. Result is a fully functional server prepared for its integration to the rest of system.

Keywords server, data sharing, mobile devices synchronization, REST, JSONPath, CodeIgniter, PHP

Obsah

Seznam obrázků

Seznam tabulek

Úvod

Vývoj informačních technologií za posledních 30 let s sebou přinesl spoustu nových zařízení, které nyní považujeme za neodmyslitelnou součást každodenního života. Jednou z výrazných změn bylo rozšíření chytrých telefonů a jejich hromadné připojení k internetu. Nárůst potenciálních zákazníků zaregistrovalo nepřeberné množství firem i organizací a rychle na trh dodávají spoustu aplikací a nástrojů, které uživatelé chytrých telefonů mohou využívat na každém svém kroku. Nyní již existují aplikace pro drtivou většinu sociálních sítí, webových portálů, bank a institucí. Dále vzniká množství zábavného a uměleckého obsahu. Od přehrávačů multimediálního obsahu streamovaného po internetu, přes virtuální knihovny plné elektronických verzí slavných literárních děl až po aplikace ulehčující život.

Tento trend je jedním z důvodů vzniku systému Tournament Manager, jehož úkolem má být usnadnění organizace sportovních turnajů, evidence výsledků a výpočet statistik. Systém se má skládat z několika klientských aplikací pro mobilní zařízení a serveru, který bude zajišťovat sdílení dat mezi nimi. Na vývoji systému se podílí tým lidí a v tuto dobu je vyvíjen pro operační systém Android a pro univerzální platformu Windows. Systém by měl umožňovat implementaci jednotlivých sportů v podobě rozšiřujících balíčků, kterými bude možné rozšířit sadu nabízených sportů v klientských aplikacích. Tato práce se zabývá vývojem zmiňovaného serveru, který má vedle sdílení dat umožňovat také zobrazení těchto dat v podobě webových stránek. Vývoj jednotlivých částí systému je kromě této práce zachycen v několika dalších samostatných pracích: Android klient[1], Android balíčky[2][3] a Windows 10 Mobile klient[4].

Rozbor zadání a struktura práce

Zadání této práce (dostupné na začátku tohoto dokumentu) je rozděleno na několik bodů. Aby bylo možné zadání práce splnit, což je samozřejmě cílem

této práce, je nutné splnit všechny dílčí body.

První bodem zadání práce je analyzovat požadavky na serverovou část z pohledu synchronizace klientů a webové prezentace. Bude nutné projít požadavky na celý systém a identifikovat procesy, které vyžadují účast serverové části. Následně bude definována sada požadavků na serverovou část systému. Tento bod bude splněn v první polovině kapitoly 1 Analýza.

Druhým bodem zadání je provedení rešerše dostupných hostingů a technologií pro serverovou část systému. V této práci bude provedena rešerše dostupných hostingů, jejíž součástí bude vyhodnocení použití jednotlivých možností pro účely serverové části systému. Rešerši hostingů bude možné najít na konci kapitoly 3 kdy již bude jasné, jaké jsou nároky výsledného řešení. Rešerše technologií bude rozdělena na analýzu existujících řešení, která bude součástí kapitoly 1 Analýza, a analýzu dostupných technologií v kapitole 2 Návrh.

Třetím bodem zadání je navrhnout architekturu a strukturu serverové části systému. Návrhem architektury a struktury serverové části se bude zabývat samostatná kapitola 2 Návrh. V této kapitole bude nejprve řešena architektura jako celek a následně budou řešeny její jednotlivé části.

Čtvrtým bodem zadání práce je implementovat a řádně otestovat serverovou část systému pro synchronizaci včetně webové prezentace statistik a výsledků. Serverová část bude implementována podle návrhu, který vznikne v rámci stejnojmenné kapitoly. Průběh a způsob implementace bude popsán v kapitole 3, kde budou uvedeny i použité nástroje. Součástí této kapitoly budou i ukázky zajímavých částí kódu. Samotnou implementaci bude možné najít na přiloženém CD. Testována bude především část zajišťující komunikaci s klientskými aplikacemi, ale pozornost bude věnována i další složitým částem serverové části. Průběh testování včetně ukázek testů bude možné najít v kapitole 4 Testování.

Pátým a posledním bodem zadání je serverovou část řádně zdokumentovat, především z pohledu rozhraní pro synchronizaci. Podrobná dokumentace rozhraní pro synchronizaci včetně ukázek použití bude realizována v podobě samostatného dokumentu. Tento a další důležité dokumenty bude možné najít na přiloženém CD.

Kapitola 1

Analýza

Důležitou částí vývoje software je přesná identifikace a analýza činnosti, kterou bude výsledný produkt provádět. Jelikož bude server jednou z částí většího systému, bude nutné nejprve pohlížet na celý server, aby bylo možné pochopit veškeré souvislosti. V této kapitole budou zkoumány jednotlivé činnosti v systému, které by mohly souviset se serverovou částí. Na základě těchto činností a očekávaných výsledků bude možné vytvořit si představu o celkovém chování serveru. Toto chování bude zapsáno jako sada požadavků. Dále bude možné definovat entity, se kterými bude server pracovat. Následující kapitola se zabývá odhalením a přesným popsáním právě těchto znaků chování, čímž vznikne specifikace, kterou by měl výsledný server přesně splňovat.

1.1 Rozbor obecných požadavků na systém

Na začátku vývoje systému byla zadavatelem dodána sada požadavků na celý systém Tournament Manager včetně několika typických případů užití. Požadavky popisují především chování systému jako celku a většina se zaměřuje na podobu doménového modelu a způsobu práce s jednotlivými entitami v aplikacích pro mobilní zařízení.

Rozbor požadavků, které se vztahují k doménovému modelu systému, je možné najít v práci zabývající se návrhem aplikace pro Android[1]. Aby bylo možné pochopit problematiku systému a strukturu dat, nahrávaných na server, bude základní princip modelu nastíněn i zde. Systém má být schopen evidovat soutěže, což má být nejvyšší úroveň, kterou bude možné nahrát na server. Dále má být systém schopen evidovat hráče. Hráče je možné přidat do soutěže. Soutěže mají být následně členěny do turnajů. V turnaji je možné vybrat podmnožinu hráčů z odpovídající soutěže a tuto podmnožinu do turnaje přidat. Taková podmnožina hráčů se považuje za účastníky tohoto turnaje. V rámci turnaje má systém umožňovat rozdělení hráčů do týmů. Tyto týmy jsou poté rozřazeny do zápasů. V soutěži je možné vybrat, že se jedná o soutěž jednotlivců. V takové případě se krok rozdělení hráčů do týmu neprovádí a

zápasu se účastní každý hráč sám za sebe. Požadavky specifikují i jakým způsobem mají být vytvářeny záznamy na jednotlivých úrovních, ale z pohledu serveru jsou tyto body nezajímavé.

Pro tuto práci jsou ovšem zajímavé takové požadavky, které se vztahují k funkcím a způsobu implementace serveru. Požadavky pocházející z sady požadavků dodané zadavatelem budou označené písmenem *G* (z anglického General - obecný). Tyto požadavky budou přeneseny do výsledné sady požadavků na server v původním znění, případně budou upřesněny nebo rozděleny na více dílčích požadavků. Provázání obecných a výsledných požadavků bude zachyceno tabulkou pokrytí na konci části 1.2.

Většina obecných požadavků je kladena na celý systém, ačkoliv některé z obecných požadavků definují funkce přímo serverové části systému. Ta je v požadavcích označována jako „webová aplikace“ nebo „server“.

1.1.1 Funkční požadavky

Funkčními požadavky se rozumí takové požadavky, které popisují přímo funkce aplikace, tedy co má aplikace umět nebo dělat. Funkční požadavek by neměl definovat jak požadovaného výsledku nebo funkce dosáhnout.

GF1 Prezentace dat Webová aplikace bude umožňovat zobrazení výsledků soutěže.

GF1.1 Forma prezentace Výsledky budou zobrazovány formou webových stránek.

GF1.2 Zveřejnění dat Systém umožní sdílet odkaz na zobrazení výsledků dané soutěže.

GF1.3 Zabezpečení dat Systém umožní zaheslovat zobrazení výsledků dané soutěže.

GF2 Sdílení dat Webová aplikace bude umožňovat sdílení dat soutěže mezi uživateli.

GF2.1 Nahrání dat Systém umožní pushnout data soutěže na server.

GF2.2 Přístup k datům Systém umožní nasdílet pushnutá data s jiným uživatelem prostřednictvím e-mailu.

GF2.3 Zabezpečení dat Sdílení bude možné zaheslovat.

GF2.4 Stažení dat Systém umožní pullnout sdílená data.

GF2.5 Správa sdílení Systém bude řešit synchronizaci sdílených dat.

1.1.2 Nefunkční požadavky

Nefunkční požadavky se nezaměřují na funkce aplikace, ale spíše na její obecné chování. Mezi nefunkčními požadavky proto mohou být takové, které vyžadují použití určité technologie, určují nějaké limity na výkon aplikace nebo rozhodují o škálovatelnosti aplikace.

GNF1 Struktura systému Systém by se měl skládat z mobilní aplikace sloužící pro hlavní práci se systémem a daty a webové aplikace umožňující synchronizaci dat mezi více mobilními klienty a prezentaci dat dalším osobám.

GNF2 Způsob realizace Webová aplikace bude řešena objektově v PHP.

1.1.3 Postřehy k obecným požadavkům

Již při pohledu na tuto sadu požadavků se začíná rýsovat základní představa o funkcích serverové části a její spolupráci se zbytkem systému. U požadavků, které jsou popisují chování celého systému, bude nutné rozhodnout jaký vliv mají na serverovou část a tuto dílčí funkce definovat je jako samostatný požadavek. Příkladem může být požadavek GF2.5 Správa sdílení, ze které není jasné do jaké míry má být akce automatizovaná a která část systému bude synchronizaci řídit. Některé z požadavků bude nutné upřesnit. Jedná se například o požadavek *GF1.1 Forma prezentace*, ze kterého není jasné, jakým způsobem má prezentace pracovat s modulárním konceptem klientů. Toto vše bude vyřešeno konzultacemi se zadavatelem a vývojáři ostatních části systému Tournament Manager.

1.2 Sběr a analýza požadavků na server

U serveru, který má být výsledkem této práce, se očekává perfektní funkčnost a zajištění všech potřebných funkcí, které budou v rámci systému Tournament Manager potřeba. Aby bylo možné takový výsledek zajistit do maximální možné míry, je nutné věnovat dostatek času, pozornosti a úsilí již na začátku vývoje. Právě v této části vývoje totiž existuje riziko, že bude některá z funkcí serveru opomenuta nebo špatně pochopena, a případné opravy pak mohou být velice problematické a nákladné. V následující části práce proto dojde k identifikaci všech procesů, které zahrnují práci se serverem. Následně bude definována sada funkčních a nefunkčních požadavků na server.

1.2.1 Identifikace procesů

V rámci systému Tournament Manager bylo identifikováno několik různých procesů. Většina z nich se vztahuje ke klientským aplikacím, kde se má odehrávat většina operací systému, což odpovídá požadavku GNF1 Struktura

systému od zadavatele. V této části práce budou představeny ty z procesů, které naprosto zřejmě budou vyžadovat účast serverové části systému.

1.2.1.1 Organizování turnaje

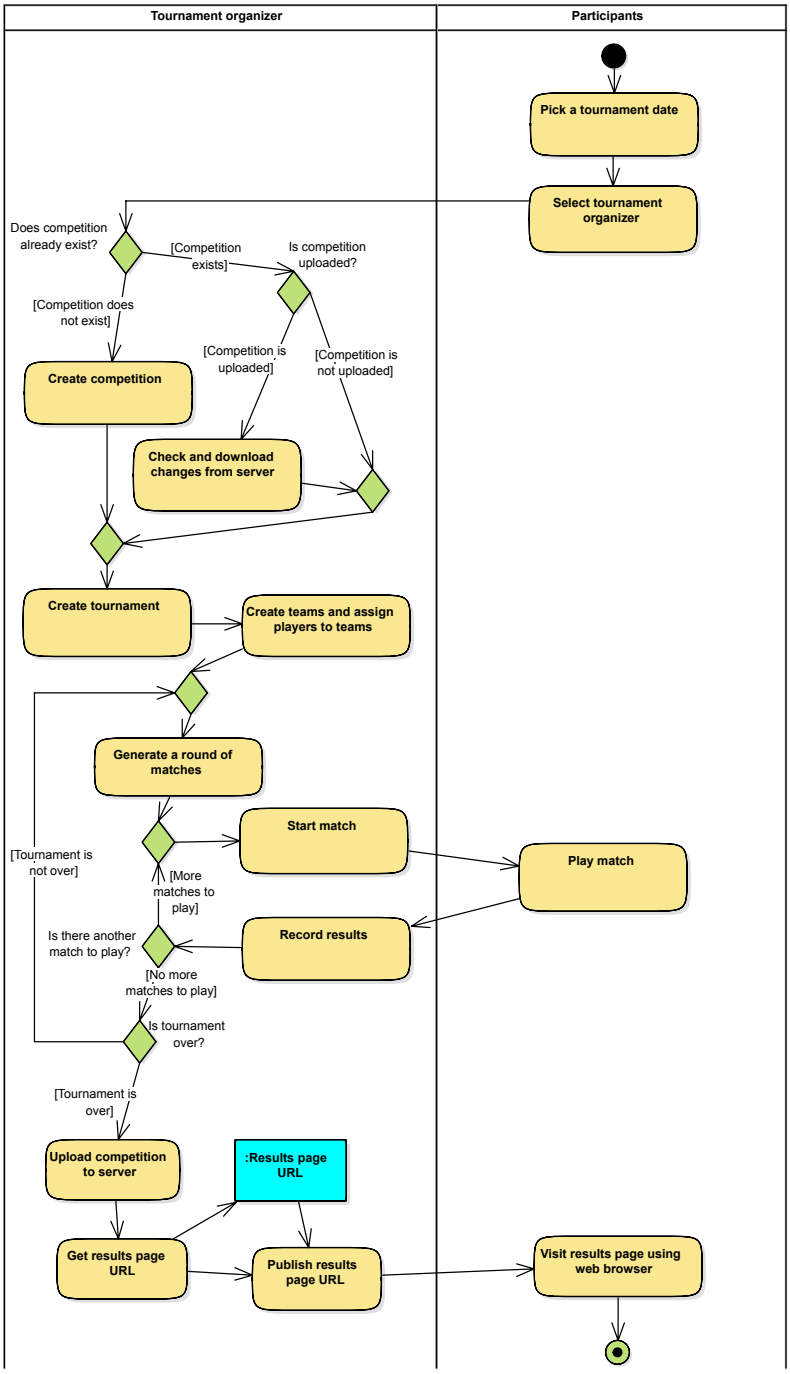
Jeden z hlavních procesů v systému Tournament Manager je organizování soutěže, resp. turnaje. Začátkem procesu je dohodnutí termínu mezi účastníky turnaje. Následně je vybrána osoba, která se bude starat o organizaci celého turnaje a evidenci výsledků. Pro dohodnutí termínu konání turnaje a výběr organizátora se předpokládá užití samostatných komunikačních prostředků mimo systém Tournament Manager. Pověřená osoba nejdřív zajistí, aby v systému existovala soutěž, v rámci které bude turnaj pořádán, a poté v ní připraví nový turnaj. Následně v turnaji vytvoří týmy, do kterých rozřadí hráče. Poté nechá vygenerovat první kolo zápasů mezi jednotlivými týmy. V následující části jsou postupně odehrány všechny vygenerované zápasy. Na konci každého zápasu organizátor zapíše dosažené výsledky do systému. V případě zájmu nebo potřeby je možné vygenerovat další kolo zápasů a v turnaji pokračovat. Po ukončení turnaje organizátor nahraje nová data na server. Systém mu poté umožní získat odkaz na stránku s výsledky soutěže, který může zveřejnit dle svého uvážení. Pomocí tohoto odkazu se účastníci turnaje mohou pomocí svého internetového prohlížeče podívat na svoje statistiky, výsledky v rámci turnaje nebo zjistit aktuální pořadí v rámci soutěže. Proces je zachycen v podobě diagramu na obrázku 1.1.

1.2.1.2 Sdílení soutěže mezi uživateli

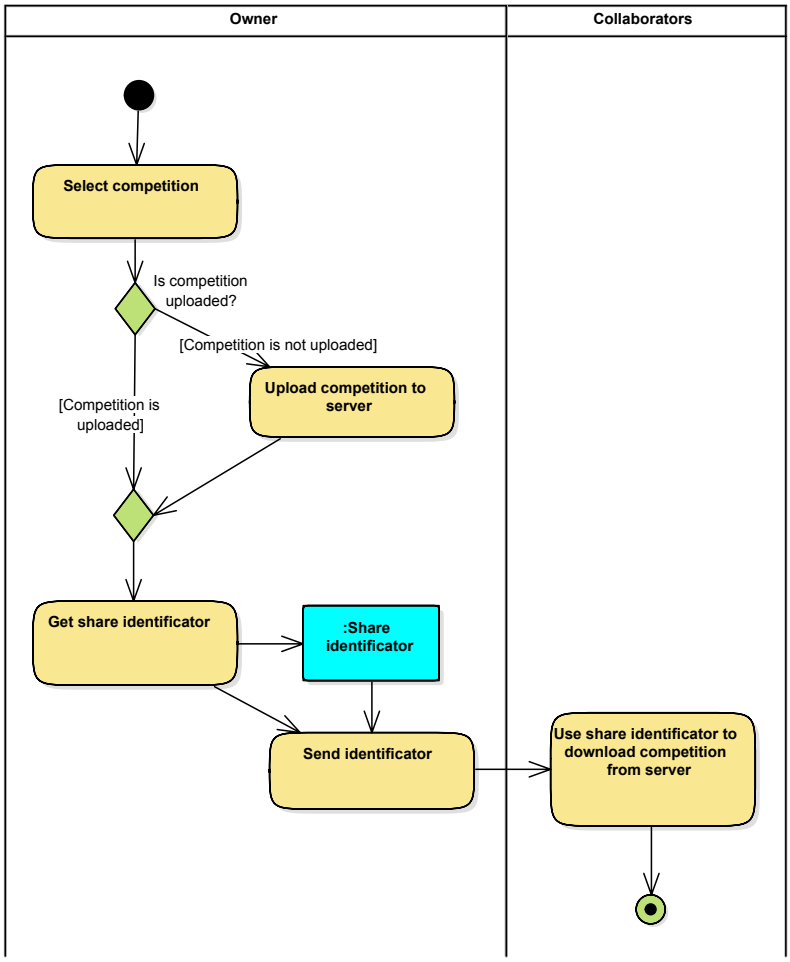
Dalším z identifikovaných procesů pracujících se serverem je proces sdílení soutěže mezi uživateli. Na začátku procesu je nutné zajistit, aby byla soutěž nahraná na serveru. Poté je možné získat identifikátor a ten předat dalšímu uživateli. Proces sdílení soutěže je dokončen, jakmile druhý uživatel vloží obdržený identifikátor do své klientské aplikace a stáhne soutěž ze serveru. Oba uživatelé nyní mají lokální kopii soutěže a zároveň mají možnost editovat její obsah. Proces je zachycen v podobě diagramu na obrázku 1.2.

1.2.1.3 Vývoj balíčku

Posledním z procesů je vývoj nového rozšiřujícího balíčku. Vývojář balíčku se musí nejdříve rozhodnout, jaké sporty chce ve svém balíčku implementovat a pro jakou platformu má být balíček určen. Důvodem vývoje balíčku může být chybějící sport v systému nebo přenesení již existujícího balíčku na další platformu. Jakmile je vše vybráno, vývojář navštíví sekci pro vývojáře na serveru, kde nový balíček zaregistruje. Při registraci obdrží unikátní název pro balíček a klíč pro komunikaci s synchronizačním rozhraním. Dalším krokem vývojáře je stažení startovního balíčku pro vybranou platformu, který by již měl obsahovat většinu kódu pro načtení knihovny a komunikaci s jádrem, včetně



Obrázek 1.1: Organizování turnaje - process diagram

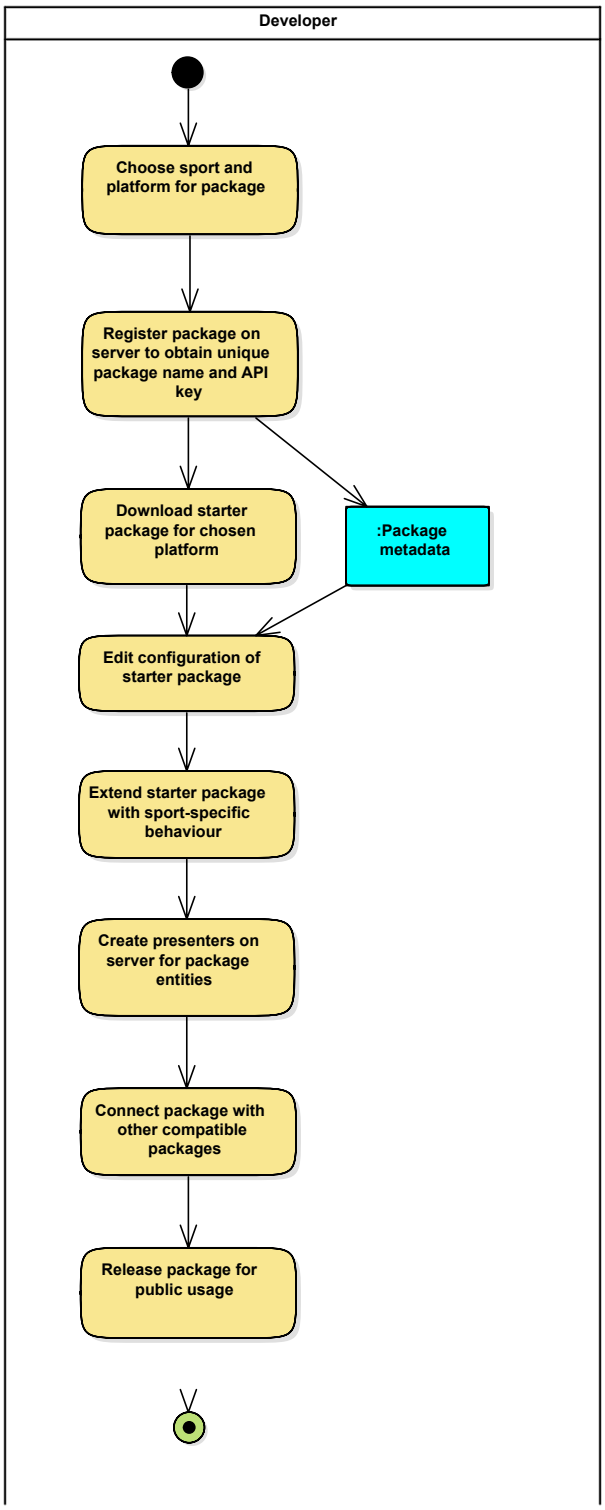


Obrázek 1.2: Sdílení soutěže - process diagram

několika ukázek. Úkolem vývojáře je nyní nastavit balíček podle dat obdržených při registraci a začít s implementací nového sportu. Jakmile je balíček dokončen, je nutné vytvořit na serveru presentery pro zobrazení výsledků soutěží z nového balíčku a propojit balíček s ostatními kompatibilními balíčky. Kompatibilní balíček může být například implementace stejného sportu pro jinou platformu. Posledním krokem je již pouze vydání balíčku pro veřejnost. Proces je opět zachycen v podobě diagramu, tentokrát na obrázku 1.3.

1.2.2 Definice požadavků

V minulé části bylo popsáno, jaké procesy má systém podporovat. Nyní je nutné izolovat a definovat jednotlivé požadavky na server. Tyto požadavky budou sloužit jako měřítko při hodnocení funkčnosti výsledného serveru.



Obrázek 1.3: Vývoj balíčku - process diagram

F1 Nahrání dat Server umožní nahrát data soutěže z klientů na server.

F1.1 Úprava dat Server klientům umožní aktualizaci dat na serveru.

F1.2 Dílčí část Server nebude při aktualizaci vyžadovat opětovné nahrání všech dat soutěže a umožní nahrání pouze nových nebo upravených dat.

F2 Uchovávání dat Server bude uchovávat data soutěže na serveru.

F2.1 Verze uchovávaných dat Server bude uchovávat vždy pouze aktuální verzi dat.

F2.2 Budoucí balíčky Server bude schopen uchovávat data soutěží ze všech aktuálních i budoucích balíčků bez nutnosti zásahu do funkčnosti a kódu serveru.

F3 Stažení dat Server umožní stáhnout data soutěže ze serveru do klientů.

F4 Synchronizace Server umožní zkontrolovat, zda je verze dat na serveru novější než na klientské aplikaci.

F4.1 Přepsání dat Server neumožní přepsat data na serveru jejich starší verzí.

F5 Sdílení dat Server umožní získat a měnit data na serveru pouze oprávněným osobám.

F5.1 Autorství dat Server bude považovat za autora dat osobu, která data na server nahraje. Tato osoba obdrží při nahrání dat sadu přístupových tokenů.

F5.2 Udělení práv Server bude považovat všechny osoby, které prokážou znalost přístupového tokenu, za oprávněné.

F6 Prezentace dat Server umožní prezentovat výsledky soutěže na internetu.

F6.1 Forma prezentace Výsledky budou prezentovány v podobě HTML dokumentů přístupných pomocí internetového prohlížeče.

F6.2 Aktuálnost dat Server bude prezentovat výsledky vždy v závislosti na poslední nahrané verzi dat. Server nemusí uchovávat historii výsledků.

F6.3 Zabezpečení prezentace Server umožní přistupovat k výsledkům pouze osobám, které vlastní odkaz na stránku s výsledky. Pro přístup k datům bude vyžadován odpovídající přístupový token.

F6.4 Přizpůsobení prezentace Server umožní vývojářům balíčků přizpůsobit formu prezentovaných dat bez nutnosti upravovat strukturu nebo kód serveru.

F7 Správa balíčků Server bude uchovávat přehled všech sportovních balíčků v systému.

F7.1 Registrace balíčku Server umožní vývojáři balíčku registrovat nový balíček na serveru.

F7.2 Kompatibilní balíčky Server umožní propojovat mezi sebou kompatibilní balíčky. U kompatibilních balíčku je zaručena schopnost jednoho balíčku zpracovat data soutěže balíčku druhého.

F7.3 Autentizace Server umožní nahrávat nová a měnit nebo mazat stávající data na serveru pouze balíčkům registrovaným na serveru.

F8 Příručka Server bude obsahovat příručku pro vývojáře balíčků, která bude obsahovat návody a postupy potřebné k implementaci nového balíčku.

NF1 Způsob realizace Server bude řešen objektově v jazyce PHP.

NF2 Komunikace s klienty Server bude pro komunikaci s klienty používat komunikační protokol, který je podporován na platformách OS Android a Windows 10 Mobile.

1.2.2.1 Pokrytí obecných požadavků

Výsledná sada požadavků na server pokrývá všechny obecné požadavky, které dodal zadavatel projektu. Vztahy mezi jednotlivými sadami požadavků jsou zachyceny v tabulce 1.1.

Požadavek GF2 Sdílení dat byl rozšířen na několik přesněji definovaných požadavků. Požadavky F1 až F3 popisují práci s daty, požadavky F4 a F5 popisují funkce spojené se sdílením a zabezpečením sdílení.

Požadavek GF2.2 Přístup k datům byl týmem analyzován a rozdělen na několik požadavků v rámci celého systému. Pro server vznikl požadavek F5.2 Udělení práv. Šíření přístupového tokenu emailem zajistí klientské aplikace.

Požadavek GF2.3 Zabezpečení dat byl rozdělen na několik požadavků definujících zabezpečení sdílení a práci s přístupovými tokeny. Možnost zaheslovat sdílení bude řešena nutností doložit přístupový token, což je zachyceno požadavky F5 Sdílení dat a F5.2 Udělení práv.

Požadavek GF2.5 Správa sdílení byl pro server minimalizován na možnost zjištění, zda je na serveru dostupná novější verze než má klientská aplikace. Funkce byla zachycena jako požadavek F4 Synchronizace. Pro zajištění konzistence dat na serveru byl navíc přidán požadavek F4.1 Přepsání dat.

Požadavek GNF1 Struktura systému není v sadě požadavků pro server zastoupen, protože ze strany serveru bude splněn již existencí samotného serveru.

Tabulka 1.1: Pokrytí obecných požadavků

Obecný požadavek	Požadavek na server
GF1	F6
GF1.1	F6.1
GF1.2	F6.3
GF1.3	F6.3
GF2	F1 - F5
GF2.1	F1
GF2.2	F5.2
GF2.3	F5, F5.2
GF2.4	F3
GF2.5	F4
GNF1	-
GNF2	NF1

1.3 Analýza existujících řešení

Existuje hned několik webových služeb a jiných technologií, ze kterých je možné se při vývoji serveru inspirovat a použít je jako kvalitativní měřítko. Některé z těchto služeb mohou dokonce splňovat požadavky definované v kapitole 1.2 a bylo by možné je použít pro výsledné řešení. Z tohoto důvodu bylo vyhledáno několik služeb umožňujících sdílení dat mezi mobilními aplikacemi. Tyto služby byly analyzovány a porovnány se základní sadou požadavků, které jsou nutné pro úspěšnou integraci do systému Tournament Manager.

Nalezené řešení musí splňovat všechny z následujících požadavků, aby bylo možné jej považovat jako vhodné pro implementaci serverové části systému:

- Řešení musí umožňovat sdílení dat klientů.
- Řešení musí umožňovat klientským aplikacím práci v offline režimu.
- Řešení musí být schopné komunikovat s klienty pro OS Android a UWP.
- Řešení musí být schopné prezentovat výsledky nahraných soutěží veřejnosti v podobě webových stránek.

Je vhodné, aby výsledné řešení bylo psané v jazyce PHP nebo Java, protože údržba výsledného systému bude po dokončení realizace plně v režii zadavatele. Tento požadavek není kritický a pokud by měl být jedinou překážkou, je možné ho po konzultaci se zadavatelem ignorovat. Na internetu bylo nalezeno několik služeb, které by mohly splňovat výše definované požadavky. Následuje krátký popis každé z nich, včetně kontroly splnění definovaných požadavků.

1.3.1 Google Firebase

Služba Google Firebase[5] je rozsáhlým systémem nabízející skoro vše od cloudových služeb po jednoúčelové služby pro přihlašování. Jednou z částí služby Firebase je Realtime Database. Jedná se o NoSQL databázový systém přístupný přes automaticky generované API. Toto řešení je možné použít při sdílení dat mezi mobilními aplikacemi nebo jako backend pro jednoduché javascriptové prezentace. Služba se navíc sama stará o synchronizaci dat mezi klienty a umožňuje používat poslední synchronizovanou verzi v offline režimu.

- Firebase splňuje požadavky na sdílení dat včetně možnosti práce v offline režimu.
- Z mobilních zařízení momentálně Google Firebase oficiálně podporuje operační systémy Android a iOS. Použití pro UWP je teoreticky možné přes REST API, způsob použití by ale byl výrazně odlišný od implementace na Androidu. Požadavek na podporu je tedy označen jako nesplněný.
- K datům nahraným v databázi je možné přistupovat i z webu a je proto možné vytvořit webovou prezentaci, která by byla umístěná na hostingu Google Firebase a prezentovala by výsledky nahraných soutěží. Požadavek na prezentaci je splněn.

Google Firebase se jeví jako skoro perfektní možnost pro realizaci serverové části systému Tournament Manager. Bohužel, aktuálně neexistuje jeho přijatelná podpora pro platformu UWP a nelze ho proto přijmout.

1.3.2 Couchbase Mobile

Služba Couchbase Mobile[6] je další z cloudových databázových řešení. Podobně jako v případě Google Firebase se jedná o NoSQL databázový systém s automaticky generovaným API. Také zde je na klientských zařízeních uchovávaná lokální kopie databáze a v případě offline režimu je možné data stále editovat. Po připojení na internet dojde k automatické synchronizaci dat.

- Couchbase Mobile splňuje požadavky na sdílení dat včetně možnosti práce v offline režimu.
- Couchbase Mobile v tuto dobu oficiálně podporuje mobilní operační systémy iOS a Android. Aktuálně není dostupná podpora pro Windows Phone nebo UWP, ačkoliv existuje podpora pro desktopové aplikace běžící na OS Windows. Požadavek na podporu je tedy označen jako nesplněný.

- K datům nahraným v databázi je možné přistupovat přes REST API, které je přístupné přes web. Je tedy možné napsat webového klienta, který bude načítat data soutěží a následně je prezentovat v podobě webových stránek. Webový klient by bylo nutné udržovat odděleně od databázového systému, protože Couchbase Mobile nabízí pouze databázový systém. I přes tento nedostatek lze požadavek uznat za splněný.

Couchbase Mobile je bohužel další službou, která pouze o kousek nevyhovuje definovaným požadavkům a je nutné ji odmítnout. Na rozdíl od Google Firebase je možné najít informace o připravované podpoře UWP platformy a lze ji v blízké době očekávat.

1.3.3 Realm Mobile Platform

Služba Realm Mobile Platform[7] je další službou, která nabízí databázový systém přístupný přes API. Na rozdíl od předchozích uvedených služeb je Realm Mobile Platform prezentována jako offline-first, tedy zaměřena především na offline režim s vestavěnou podporou synchronizace. Tento koncept odpovídá potřebám systému Tournament Manager nejvíce z uvedených možností. Služba vyžaduje provozování vlastního serveru, přes který dochází k synchronizaci mobilních aplikací.

- Realm Mobile Platform splňuje požadavky na sdílení dat včetně možnosti práce v offline režimu.

- Realm Mobile Platform v tuto dobu oficiálně podporuje mobilní operační systémy iOS a Android. Pro zmíněné dvě je k dispozici podpora i přes projekt Xamarin, ale podpora pro UWP dostupná není. Požadavek na podporu je tedy označen jako nesplněný.

- K datům nahraným v databázi je možné přistupovat přes REST API, které je přístupné přes web. Je tedy možné napsat webového klienta, který bude načítat data soutěží a následně je prezentovat v podobě webových stránek. Protože je vyžadováno provozování vlastního serveru, je pravděpodobné, že by na stejném serveru byla provozována i prezentační část serveru. Požadavek lze brát za splněný.

Velkým rozdílem proti předchozím zmíněným je offline-first přístup k udržování dat. U předchozích řešení byl offline režim považován za dočasný výpadek služby a bral se jako nouzový stav. Zde je offline režim považován za standardní stav, což více odpovídá způsobu používání systému Tournament Manager. Také toto řešení bohužel naráží na podpoře platformy UWP.

1.4 Doménový model

V minulé části bylo definováno, co se od serveru očekává, nyní je třeba analyzovat, s jakými objekty bude server pracovat. V této části práce bude nejprve převzat doménový model používaný v klientských aplikacích a ten bude analyzován a následně upraven pro použití na serveru.

1.4.1 Doménový model klientů

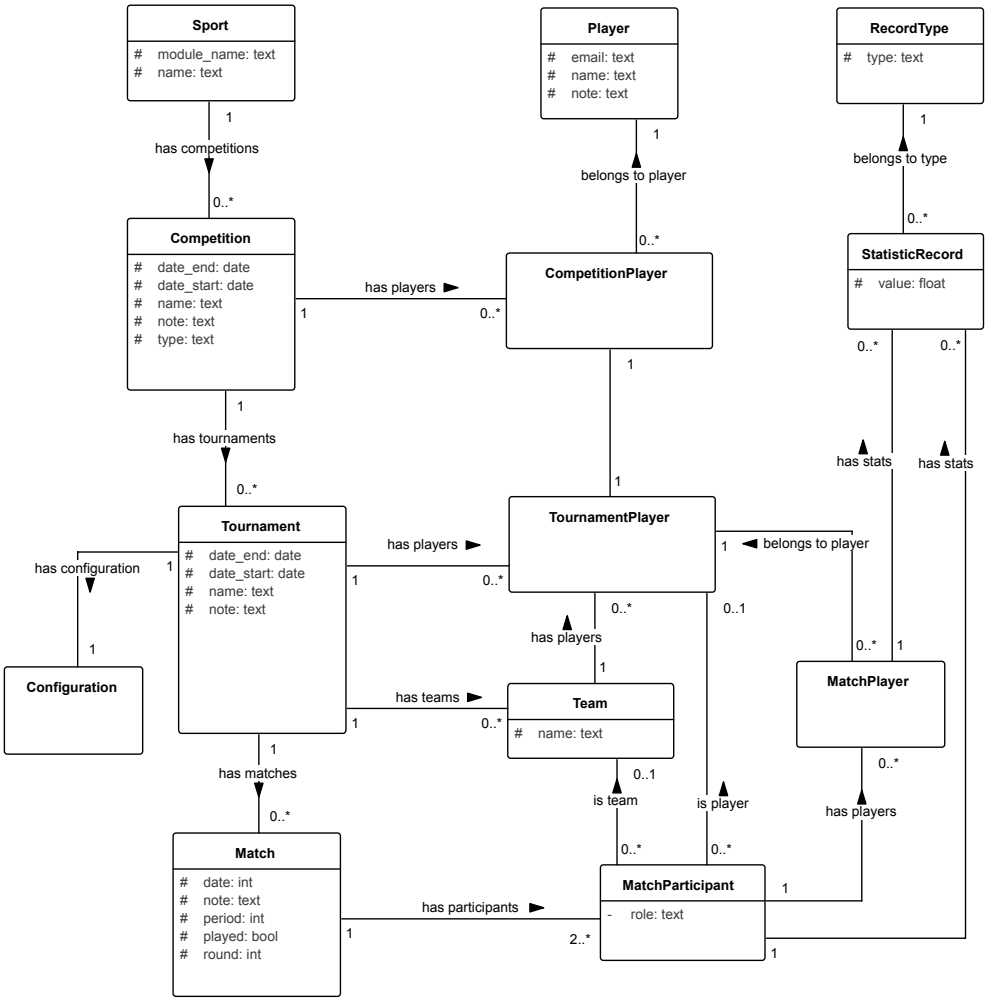
Na celý systém je kladen jeden zásadní požadavek, kterým je možnost rozšíření systému o nové sporty v podobě rozšiřujících modulů. Aby nebylo nutné v modulech opakovat stále se opakující části kódu, mají být společné funkce a části kódu držené v podobě knihovny, která má sloužit jako základ funkčnosti balíčků. Od vývojáře nového balíčku se očekává maximální využití částí knihovny a implementace pouze rozdílných prvků a funkcí. Tuto funkčnost bylo nutné brát v úvahu již při návrhu doménového modelu celého systému. V rámci vývoje mobilních klientů byl navržen doménový model (viz Obrázek 1.4), zachycující všechny entity v systému. V případě konkrétních entit, jakými jsou například soutěž, tým nebo hráč, je model snadno pochopitelný. Soutěž se skládá z turnajů, turnaje se skládají ze zápasů, u každého zápasu jsou účastníci, vše je přesně tak, jak je vyžadováno v dodaných požadavcích na systém. Obdobně bylo vytvořeno několik imaginárních entit, které slouží pro upřesnění vztahů mezi entitami. Díky těmto entitám je možné zachytit nejen účast hráče v zápasu, ale zároveň je možné uchovat i jeho dosažené výsledky. Obdobně je možné zjistit složení týmu pro jeden určitý zápas. Podrobný popis doménového modelu klientských aplikací je možné najít v práci zabývající se klientem pro Android[1].

1.4.2 Výsledný doménový model

Při vytváření modelu pro serverovou část systému bylo nutné pohlížet na systém jiným způsobem. O správu dat v systému se budou starat klientské aplikace, úkolem serveru bude uchovávat aktuální verze těchto dat. Doménový model pro server je zachycen na obrázku 1.5.

Package Entita Package představuje rozšiřující balíček pro klientské aplikace. Každý z balíčků je vytvořen pro právě jednu platformu a je provázán s právě jedním vývojářem, který má na starosti jeho správu. Balíčky je možné mezi sebou propojovat, čímž dochází k zachycení jejich kompatibility. U kompatibilních balíčků se očekává jejich schopnost zpracovat data druhého.

Developer Entita Developer představuje vývojáře balíčků a slouží především pro určení autorství balíčku. Developer může být zároveň administráto-

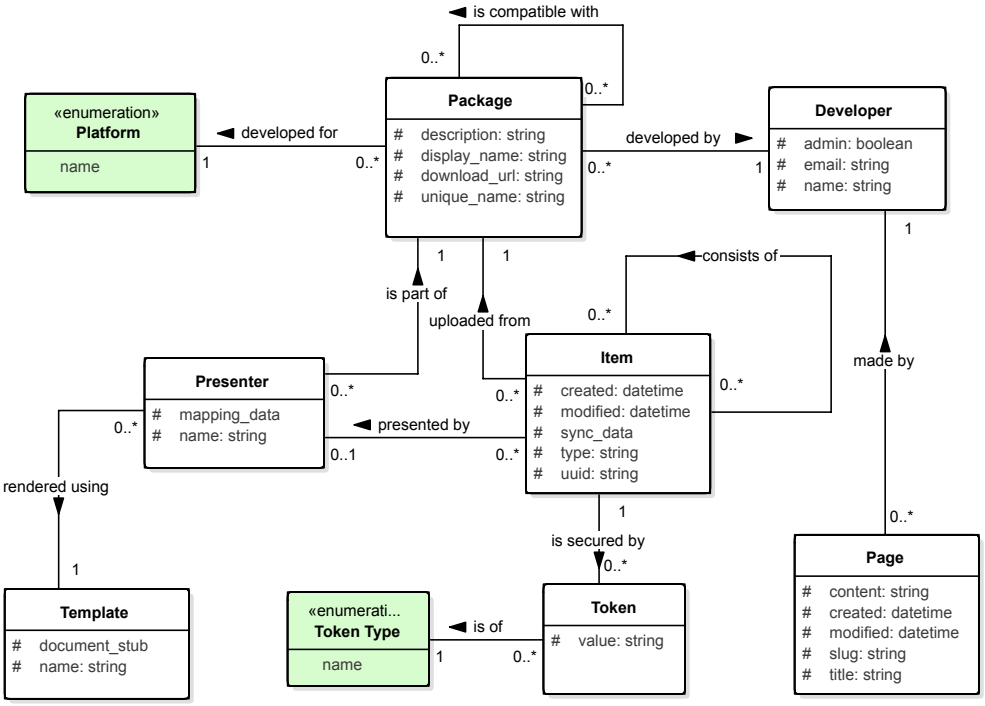


Obrázek 1.4: Doménový model klientských aplikací

rem serveru, čímž mu bude umožněno spravovat balíčky a obsah příručky pro vývojáře.

Platform Každý z balíčků je vyvíjen pro jednu cílovou platformu. Aby bylo možné balíček nasadit, je nutné, aby byla pro tuto platformu vytvořena také klientská aplikace. Enumeration Platform představuje všechny platformy, které jsou v rámci systému podporovány.

Item Entita Item je nejdůležitější entitou celého serveru. Představuje položku nahranou na server. Z pohledu serveru není důležité, zda se jedná o soutěž, turnaj, tým nebo jinou z entit používaných v klientech. Pro server jsou si všechny tyto entity rovny a práce s nimi je identická. Přesný popis proč a jak se došlo k použití této entity je možné najít v části 1.4.2.1



Obrázek 1.5: Doménový model serveru

Vznik entity Item. Každý item se odkazuje na balíček, ze kterého byl nahrán, a definuje, který presenter se má použít k jeho zobrazení. Ke každému itemu dále patří sada přístupových tokenů, které budou použity při autorizaci jednotlivých operací s itemy. Itemy je navíc možné do sebe skládat.

Token Entita Token slouží pro ověření přístupu a pro každý Item jich existuje hned několik. Každý Token patří právě jedné instanci entity Item. Typ tokenu je definován vazbou na záznam v enumeration Token Type.

Presenter Entita Presenter představuje převodník jednotlivých Itemů na datovou strukturu, kterou je schopen server zobrazit. Jedná se o prvek, který bude hrát důležitou roli v části serveru zajišťující prezentaci výsledků. Pro určení výsledného vzhledu a struktury se presenter odkazuje na jednu z šablon.

Template Entita Template je další důležitou částí prezentace výsledků. Pro sjednocení vzhledu a formy prezentovaných dat bude existovat několik šablon, které bude možné pro zobrazení dat použít.

Page Entita Page představuje jednotlivé stránky, ze kterých bude sestavena výsledná příručka pro vývojáře balíčků. Kromě atributů potřebných pro

zobrazení stránky obsahuje základní informace, kdy byla vytvořena nebo editována, a zároveň odkaz na osobu, která stojí za jejím vytvořením.

1.4.2.1 Vznik entity Item

Nejlepším vysvětlením podstaty entity Item je následovat kroky, které předcházeli vzniku této entity. Jednotlivé kroky jsou v zjednodušené formě zachyceny v následujícím textu.

První způsob uložení dat vycházel z nejjednoduššího způsobu ukládání celé soutěže jako jednoho dále nedělitelného celku. Celý proces sdílení měl být postaven na jednom kusu dat, který by se přenášel mezi serverem a klienty například v podobě komprimovaného archivu. Struktura dat v rámci tohoto archivu by nebyla nijak definovaná a každý balíček by si do něj mohl přidat, co uzná za vhodné. Nevýhodou tohoto způsobu by ovšem byla nutnost přenosu velkého objemu dat kvůli libovolné změně. Teoreticky stačila oprava jednoho překlepu v poznámce u týmu a přesto by bylo nutné na server nahrávat znovu všechna data o celé soutěži, což by bylo v rozporu s požadavkem F1.2 Dílčí část. V případě mobilních zařízení, kde je velmi často přenos dat přes mobilní sítě zpoplatněn, by se jednalo o výrazný nedostatek a proto začalo hledání alternativy.

Zřejmým krokem bylo rozdělení soutěže na více částí. Možnost rozdělení soutěže podle doménového modelu používaného v klientech byla zamítnuta skoro okamžitě. Doménový model používaný u klientů spoléhá na možnost definovat novou entitu v novém balíčku. Takto je možné definovat například nové statistiky, přidat upřesňující informace u týmu a různé další úpravy. Toto by ale vyžadovalo v případě uvedení nového balíčku pro klientské aplikace, změnit také část serveru, což je v rozporu s požadavkem F2.2 Budoucí balíčky.

Výsledným řešením byl způsob ukládat všechny nahrávané entity jako jednu obecnou entitu. Pokud tato obecná entitu zvládne uložit libovolnou stávající entitu v systému Tournament Manager, nebudou jí dělat problémy ani entity, které vzniknou v budoucnu. Tato obecná entita byla pojmenována Item. Důležitým atributem entity Item je atribut syncData, který představuje blíže nespecifikovanou kolekci, kam je možné uložit všechny atributy ukládaných entit. Díky tomuto způsobu ukládání dat nebude nutné při představení nové entity v klientských aplikacích měnit strukturu serverové části systému. Protože se tímto převodem přijde o přirozenou informaci o typu nahrávaných dat, je součástí itemu atribut type, kam je možné uložit informaci o typu ve formě textového řetězce. Pomocí atributu type budou klientské aplikace schopné zjistit jakým způsobem deserializovat získaná data. Aby nedošlo k podobnému problému, který nastal již při prvním způsobu ukládání dat, bude umožněno mezi sebou jednotlivé itemy propojovat. Každý item může obsahovat kolekci dalších itemů (např. tým bude obsahovat kolekci hráčů). Server bude umožňovat přidání jednoho itemu do více než jedné kolekce (např. účast

týmu v zápasech), díky čemuž bude možné zachytit libovolnou strukturu, která může v systému Tournament Manager vzniknout.

Kapitola 2

Návrh

Následující kapitola se zabývá návrhem architektury a způsobu realizace serverové části systému Tournament Manager. V první polovině kapitoly bude navržena architektura serveru a budou zvoleny technologie, které budou použity při implementaci serveru. V druhé polovině budou navrženy jednotlivé vrstvy a části serveru.

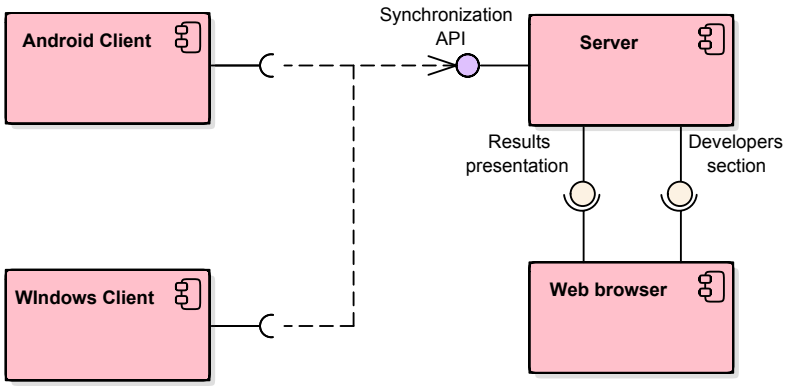
2.1 Návrh architektury

V této kapitole bude popsána architektura nejprve z pohledu celého systému Tournament manageru, kde bude zachycen vztah s ostatními pracemi. Architektura systému je uvedena pro znázornění konzumentů jednotlivých rozhraní, které bude serverová část systému nabízet. Následovat bude popis architektury samotného serveru. Důraz bude kladen na vysvětlení komunikace mezi jednotlivými vrstvami, popis vrstev bude spíše jednoduchý. Každá z vrstev serveru bude podrobně popsána v samostatných částech kapitoly 2 Návrh.

2.1.1 Architektura systému

Architektura systému (viz Obrázek 2.1) byla naznačena již v úvodu této práce. Systém Tournament Manager se skládá z celkem tří aplikací: klient pro Android, klient pro Windows a synchronizačního serveru. Jednotlivé klientské aplikace jsou dále rozděleny na jádro, knihovnu a balíčky sportů, ale tato úroveň architektury systému již v rámci této práce znázorněna není. Architekturu klientských aplikací je možné najít v příslušných pracích [1][4].

Velice důležitým požadavkem na systém je schopnost klientů pracovat bez připojení k internetu. Uživatelé systému budou veškerou práci provádět na klientských aplikacích, které budou mít nainstalované na svých mobilních zařízeních. Komunikace se serverem bude provedena na žádost uživatele v případě sdílení soutěže (funkce sdílení může být použita také pro účely zálohování) nebo pro prezentaci výsledků na webu. Pro komunikaci s klientskými aplika-



Obrázek 2.1: Architektura systému

cemi bude na serveru vytvořeno synchronizační rozhraní, které bude vycházet z návrhového stylu REST.

Server dále nabízí další dvě rozhraní, která jsou určena pro konzumaci pomocí internetového prohlížeče. První webové rozhraní je určeno pro prezentaci výsledků soutěží. Toto rozhraní bude zobrazovat data nahraná na server v podobě HTML dokumentů. Prezentace výsledků je určena především pro účastníky turnajů. Druhé webové rozhraní je určeno pro vývojáře systému Tournament Manager. Pomocí toho rozhraní bude možné přistupovat k sekci pro vývojáře, kde bude možné registrovat nové balíčky. Součástí sekce pro vývojáře bude také příručka, ve které bude možné najít rady a návody k vývoji balíčku pro klientské aplikace.

2.1.2 Architektura serveru

Server bude navržen jako třívrstvá aplikace, aby byla umožněna jeho jednodušší rozšiřitelnost a udržitelnost[8]. Třívrstvá architektura je velice častým programovacím stylem, který aplikaci rozděluje na tři samostatné vrstvy: datovou, business a prezentační. Tyto vrstvy spolu komunikují přes předem definovaná rozhraní a je tak možné upravit či zcela vyměnit jednu z těchto vrstev bez nutnosti upravovat zbytek aplikace. Jednotlivé vrstvy serveru a vztahy mezi nimi jsou zachyceny diagramem na obrázku 2.2.

Datová vrstva je nejnižší úrovní aplikace. Vrstva se stará o selekci a perzistenci dat, většinou v kombinaci s nějakým databázovým systémem. Na datové vrstvě budou vytvořeny Data Access objekty (zkráceně „DAO“), pomocí kterých bude možné přistupovat k jednotlivým entitám serveru. Každé DAO se bude starat pouze o svou entitu. Pro přístup k datové vrstvě budou pro jednotlivá DAO vytvořena odpovídající rozhraní (např. `ItemDAOInterface` nebo `AccountDAOInterface`), ve kterých budou zachyceny metody DAO tříd. Základem pro rozhraní DAO tříd bude rozhraní `BaseDAOInterface`, které bude

obsahovat popis obecných metod společných pro všechna DAO, jakými jsou například metody `findAll`, `findWhere` nebo `insert`.

Business vrstva leží na vrstvě datové a obsahuje veškerou logiku aplikace. Úkolem business vrstvy serveru bude dodat do aplikace veškeré postupy, pomocí kterých budou vyřizovány příchozí požadavky. Vrstva se bude skládat z tříd pro jednotlivé entity, které budou obsahovat implementace operací s danou entitou (např. `ItemLogic` nebo `AccountLogic`). Při realizaci jednotlivých operací bude využívána datová vrstva, přes kterou budou načítána data z databáze. Komunikace s datovou vrstvou bude umožněna přes výše popsané rozhraní. Na rozdíl od datové vrstvy, se v rámci business vrstvy očekává úzká spolupráce jednotlivých tříd logiky při zpracování přijatého požadavku. Přesný popis business vrstvy bude možné najít v části 2.5 Business vrstva. Pro komunikaci s business vrstvou budou pro třídy logiky vytvořena odpovídající rozhraní (např. `ItemLogicInterface` nebo `AccountLogicInterface`), popisující příslušné metody. Obdobně jako v datové vrstvě budou mít i zde rozhraní společný základ v podobě `AbstractLogicInterface`. Toto rozhraní bude definovat společné metody především pro integraci business vrstvy do struktury serveru.

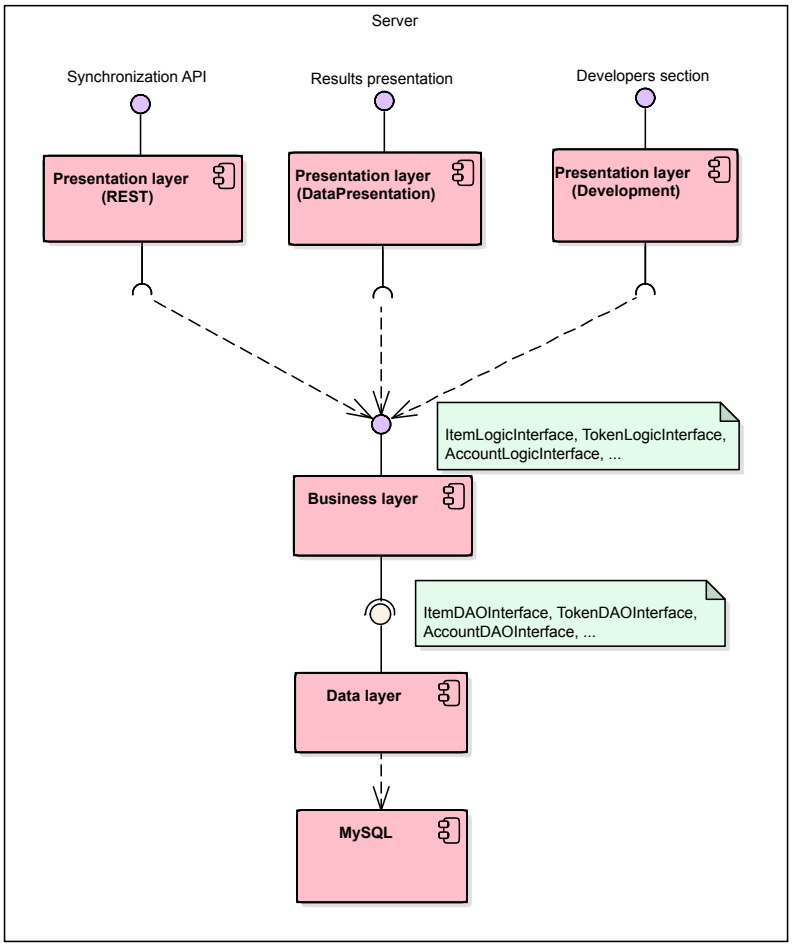
Prezentační vrstva je nejvyšší vrstvou aplikace a slouží k interakci s uživatelem. Typicky se jedná o GUI, ale může se jednat také o API používané jinou aplikací. Úkolem prezentační vrstvy je načíst vstup od uživatele, předat ho odpovídající metodě v business vrstvě aplikace a nakonec uživateli zobrazit výsledek. Prezentační vrstva serveru bude rozdělena na tři části: synchronizační API, prezentaci výsledků a sekci pro vývojáře. Přesný návrh funkcí jednotlivých částí prezentační vrstvy bude řešen v části 2.6 Prezentační vrstva.

2.2 Použité technologie

K implementaci serveru bude nutné použít hned několika různých technologií a nástrojů. V této části budou tyto technologie vybrány a popsány. Při vývoji samozřejmě budou použity i různé nástroje usnadňující práci s kódem nebo testovaní aplikace. Většina těchto nástrojů bude popsána až v následující kapitole 3 Realizace.

2.2.1 Programovací jazyk

Požadavek NF1 Způsob realizace vcelku výrazně usnadnil výběr jazyka, ve kterém bude server napsán. Volba PHP je ovšem naprosto logická a pochopitelná. Jazyk PHP je momentálně nejpoužívanějším jazykem u webových stránek a aplikací, u kterých se neočekává velmi vysoká návštěvnost, což je většina [9][10]. Z analýzy je zřejmé, že velká část výpočtů a funkcí systému bude prováděna na klientských aplikacích a server bude provádět především



Obrázek 2.2: Architektura serveru

operace nutné k synchronizaci. Synchronizace bude prováděna na příkaz uživatele a lze očekávat, že bude prováděna řádově jednou za několik dní. Taková návštěvnost ani zdaleka nepřekračuje [11] návštěvnost jiných webových aplikací, které jsou postaveny na PHP.

2.2.2 Datové uložiště

Pro efektivní uložení nahrávaných dat na serveru bude použito některé z hotových databázových řešení. Samozřejmě je možné přijít s vlastním způsobem uchovávání dat například v podobě souborů, ale takové řešení je vhodné implementovat až pokud existující databázová řešení nejsou dostatečná nebo naopak pokud by správa databáze přinesla více obtíží než užitku (např. uložení konfigurace jednoduché webové prezentace).

Při výběru vhodného databázového řešení se dostáváme k zásadní otázce

SQL nebo NoSQL. Aktuálně jsou databázové systémy založené na SQL v silné převaze oproti všem ostatním NoSQL řešením [12]. To ale samozřejmě neznamená, že by NoSQL systémy byly horší. Jedná se o odlišné technologie, které je vhodné používat v jiných situacích. S velkou pravděpodobností také existují aplikace, které běží na SQL databázích, ačkoliv by u nich bylo vhodnější využít některé z NoSQL řešení.

Obě z uvedených řešení mají své výhody i nevýhody. Zásadními výhodami NoSQL databází je možnost uložit informace s předem neznámou strukturou (např. v podobě Key-Value záznamu) a excelentní výkonnost a škálovatelnost. Výhodami SQL je možnost ukládat data v předem definovaných strukturách (tabulka/relace), které je možné dále propojovat pomocí vztahů (cizí klíče). SQL databáze dále disponují možností provádět operace v transakcích, díky čemuž je možné provádět rozsáhlé změny v datech a nebát se narušení integrity dat v případě chyby [13].

Možnost ukládat data s předem neznámou strukturou by se pro účely této práce velice hodila, požadavky a navržený doménový model ovšem více odpovídají relačním SQL databázím. Využití najde především možnost definovat vztahy mezi jednotlivými relacemi a provádění operací v transakcích. Vysoká výkonnost NoSQL není vzhledem k předpokládané návštěvnosti tak velkým přínosem.

2.2.3 PHP framework

Základní technologie již byly vybrány, nyní je nutné vybrat framework, pomocí kterého bude server vytvořen. Použití frameworku není nutností a samozřejmě je možné napsat celý kód serveru kompletně od základu. Tímto způsobem lze vytvořit extrémně efektivní a výkonnou aplikaci, která bude naprosto přesně splňovat vše, co se od ní nyní očekává. Právě slovo „nyní“ je v předchozí větě důležité. Celá architektura aplikace je v rukou jednoho vývojáře a lze očekávat její optimalizaci pro aktuální potřeby. Případné budoucí úpravy nebo rozšíření se mohou ukázat jako velmi problematické a mohou vyžadovat velkou míru rozsáhlých úprav. Vývoj aplikace je možné přizpůsobit tomuto riziku a server psát s povědomím o případném budoucím rozvoji. V praxi by to ale znamenalo napsat si vlastní sadu nástrojů a služeb, tedy v podstatě napsat vlastní framework. Z tohoto důvodu byla možnost vývoje od základu zamítnuta rovnou.

2.2.3.1 Přehled frameworků

Při vývoji systému bude použit některý z existujících PHP frameworků. Aktuálně jich existuje nepřeberné množství, každý zaměřený na jiný typ systému nebo způsob používání. Některé frameworky se snaží dosáhnout extrémního výkonu, jiné si berou za cíl udržet maximálně přehledný kód. Pro serverovou část systému Tournament Manager bude vybíráno mezi následujícími čtyřmi frameworky.

První framework, který zde bude představen, je Symfony 2 [1]. Jedná se o velice oblíbený framework používaný také u velmi navštěvovaných webových stránek a služeb, jako například Spotify [14]. Jedná se o rozsáhlý framework využívající MVC strukturu, který je možné dále rozšiřovat pomocí modulů nazývaných bundles. Většina těchto modulů je vytvářena komunitou okolo tohoto frameworku, ale základní nástroje jsou vyvíjeny přímo autory frameworku. Na serveru Knp Bundles [2] je k dispozici již skoro 3000 těchto bundlů, umožňující rozšířit Symfony 2 framework o správu uživatelů, stránkování a filtrování obsahu nebo komunikaci se sociálními sítěmi. Framework je standardně dodáván s ORM Doctrine, implementujícího vzor Data Mapper. Doctrine vytváří pro všechny entity v systému repozitáře, pomocí kterých je možné vyhledávat, vytvářet a měnit veškerá data v databázi. Repozitáře obsahují velké množství metod vytvořené na míru entitám, složitější dotazy je možné zapsat pomocí QueryBuilderu. Dokumentace k tomuto frameworku je velice dobře zvládnutá a případné problémy lze řešit i s rozsáhlou komunitou, která je rozložena po celém světě. Na internetu je k dispozici nepřeberné množství tutoriálů nebo řešených dotazů.

Druhým frameworkem je poměrně nový framework Laravel [3]. Tento framework ohromnou rychlostí nabývá na oblíbenosti a momentálně je jedním z nejoblíbenějších [15]. Laravel je založen na podobném principu jako Symfony a dokonce používá množství jeho bundlů jako základ. Opět se jedná o robustní řešení s velkým množstvím již existujících částí. Framework je dodáván s ORM Eloquent, které implementuje vzor ActiveRecord. S databází se pracuje přímo přes entitu, resp. model. Práce s ORM Eloquent je rychlá a jednoduchá, ale není vhodná na složitější dotazy, které vyžadují spojování tabulek a jiné operace. Pro složitější aplikace je možné nahradit Eloquent například za Doctrine. Úroveň dokumentace je na podobné úrovni jako v případě Symfony a obdobně rozsáhlá je také komunita okolo tohoto frameworku.

Třetím frameworkem je CodeIgniter 3 [4]. Tento framework jde na celý proces vývoje webových stránek jinak než předchozí zmíněné. Jedná se o méně rozsáhlý framework, který obsahuje základní MVC strukturu a pouze základní sadu nástrojů. Je tak vhodný pro stránky a aplikace, kde by rozsáhlý framework spíše brzdil svou mohutností a většina jeho částí by nebyla nikdy použita. Všechny části frameworku jsou pak vytvořeny jako minimální možné. Validace formulářů je pouze základní, routování požadavků je velice jednoduché až primitivní. Framework není standardně dodáván s žádným ORM řešením, obsahuje pouze základní DB driver, který umožňuje používat SQL, MySQL, SQLite nebo podobnou databázi přes jedno rozhraní. Nástavbou této knihovny je pak DB Forge, pomocí kterého je možné vytvořit a editovat tabulky. Dokumentace není tak rozsáhlá jako v předchozích případech, ale díky

[1] Symfony 2. `https://symfony.com/`

[2] Knp bundles. `http://knpbundles.com/`

[3] Laravel. `https://laravel.com/`

[4] CodeIgniter 3. `https://www.codeigniter.com/`

jednoduchosti frameworku to ani není potřeba. Celá dokumentace se navíc podobá spíše návodu nebo příručce a řešení problému nebo učení nových věcí je tak pohodlnější než u předchozích frameworků. U tohoto frameworku je navíc nutné zmínit, že množství jeho principů lze dnes považovat za omezené nebo zastaralé. Jako největší problém lze považovat absenci podpory jmenných prostorů v PHP. Díky jednoduchosti frameworku je ale možné velké množství prvků dodatečně přidat bez větších obtíží. Momentálně je vyvíjena zcela nová verze CodeIgniter 4, která má většinu těchto problémů vyřešit.

Čtvrtým a posledním zde uvedeným frameworkem je Nette [5]. Jedná se o framework českého původu, rozsáhlostí někde na úrovni frameworku Laravel. Nette využívá MVP strukturu a klade důraz na krásu kódu z programátorského hlediska. Lze o něm říci, že často řeší věci odlišně od své konkurence a často tak přichází s výrazně lepší nebo pohodlnější možností. Na Nette vychází spoustou komunitou vydávaných rozšíření, obdobně jako je tomu i prvních dvou zmíněných frameworků. Standardně se framework dodává s vlastní knihovnou, která vychází z řešení NotORM. Přístup k datům je řešen přes speciální API, díky kterému se postupně skládá dotaz. V případě potřeby je framework samozřejmě možné rozšířit o některé z ORM řešení. Dokumentace je asi největší slabinou tohoto frameworku. Kromě výčtu a základního popisu tříd totiž skoro žádná neexistuje. Nette spoléhá na svou komunitu, která je velmi aktivní a znalá. Bohužel, skoro veškeré rady a nápady této komunity jsou soustředěny na několika málo webových fórech a řešení i vcelku triviálního problému se stává obtížným. Většina příspěvků je navíc psaná v češtině nebo slovenštině a vyhledávání v nich je tak výrazně složitější, než je tomu v případě anglických příspěvků pro předchozí frameworky.

Samozřejmě nebylo možné zde zmínit všechny frameworky, které je možné pro vývoj v jazyce PHP použít. Existuje množství velmi kvalitních frameworků, které jsou v některých ohledech lepší, než ty výše uvedené. Za zmínku rozhodně stojí frameworky Yii [6], CakePHP [7], Zend [8] nebo Phalcon [9]. Každý z těchto osmi zmíněných frameworků je kvalitní produkt, na kterém pracuje množství schopných lidí a který je v tuto dobu využíván na tisících webových stránek. První čtyři frameworky byly vybrány závislosti na zkušenostech autora práce, různorodosti výběru a míry používání PHP komunitou.

2.2.3.2 Porovnání a výběr frameworku

Vybrané frameworky se liší hned v několika bodech. Největším rozdílem je určitě rozsah těchto frameworků, který má vliv na výkon výsledné aplikace ale také na rychlost a pohodlí při vývoji této aplikace.

[5] Nette. `https://nette.org/`
[6] Yii. `http://www.yiiframework.com/`
[7] CakePHP. `https://cakephp.org/`
[8] Zend. `https://framework.zend.com/`
[9] Phalcon. `https://phalconphp.com/`

Porovnávat frameworky po výkonové stránce je obtížný úkol. Na internetu je možné najít velké množství různě řešených testů a benchmarků. Většina z těchto testů sleduje množství vyřízených požadavků za určitý časový úsek (většinou za 1 sekundu). U každého testu se ovšem příznivci jednotlivých frameworků ozývají, že testy nejsou platné, protože jejich oblíbený framework podává lepší výsledky při jiné konfiguraci nebo jiném způsobu použití. Toto je bohužel pravda, každý framework vyžaduje určitou míru zkušeností, aby byl využíván naplno, a ani tak není možné frameworky srovnávat přímo, protože každý z nich je vhodný na jinou činnost. Pro účely této práce bylo vybráno několik různých benchmarků, které sledovali především výkon v základní konfiguraci a minimálním úkolu (obdoba Hello World programu). Jedním z těchto testů je PHP Framework Benchmark [10], který spravuje uživatel kenjis. Z tohoto benchmarku je jasně patrná rychlost CodeIgniteru proti Symfony 2 nebo Laravelu. Obdobný výsledek ukazuje i benchmark na stránkách techempower.com [11]. Bohužel ani jeden z těchto benchmarků neobsahuje framework Nette. Toto je způsobeno faktem, že Nette je ve světě v podstatě neznámým frameworkem. Zajímavým průzkumem, který paradoxně funguje jako důkaz tohoto tvrzení, je anketa [15] z roku 2015 publikovaná na webu SitePoint. Framework Nette v této anketě dopadl velice dobře (umístění v top 3), ale je nutné se podívat kdo stojí za těmito výsledky. Prvním zajímavým faktem je počet hlasů podle země. Počet hlasů z České republiky a Slovenska převyšuje počet hlasu z libovolné jiné země. Druhým zajímavým faktem je země původu hlasů pro framework Nette, protože ze zmíněných dvou zemí pochází více než 97% hlasů pro Nette.

Pro implementaci bude použit framework CodeIgniter 3, především kvůli jeho vysokému výkonu a dobré dostupnosti rad, návodů a rozšíření. Díky své nízké rozsáhlosti a vysoké upravitelnosti je vhodný jako základ pro skoro libovolnou architekturu. Ostatní frameworky nabízejí spoustu nástrojů pro vytváření bezpečných formulářů, podporu asynchronního načítání webových stránek a podobných funkcí, díky kterým je možné vytvořit perfektní webové prezentace. Tyto nástroje by ovšem pro účely synchronizačního serveru byly využity pouze okrajově a ve většině případů by nedošlo k jejich využití vůbec.

2.2.4 Google Identity Toolkit

U každého balíčku bude vedena informace o jeho autorovi. Za autora se považuje vývojář, který balíček registroval na serveru. Z toho důvodu bude registrace balíčku vyžadovat registraci a přihlášení vývojáře do sekce pro vývojáře.

Balíčky budou vyvíjeny jako rozšíření pro klientské aplikace systému Tournament Manager. V této době jsou vyvíjeny klientské aplikace pro platformy Android a Windows. Lze předpokládat, že vývojáři balíčků budou tuto plat-

[10] PHP Framework Benchmark. `https://github.com/kenjis/php-framework-benchmark`

[11] TechEmpower. `https://www.techempower.com/benchmarks/#section=data-r13&hw=ph&test=fortune&l=4fthbz`

formu používat i k jiným činnostem a projektům a tedy že vlastní uživatelský účet na alespoň jedné z těchto platforem. Bylo by proto vhodné, kdyby sekce pro vývojáře umožňovala přihlášení pomocí takového účtu a nezatěžovala vývojáře zbytečnou registrací.

Aby nebylo nutné implementovat přihlašování kompletně od základů, bude použita služba Google Identity Toolkit [12] (zkráceně Gitkit), která umožňuje během pár minut rozšířit aplikaci o přihlašování pomocí různých účtů třetích stran. Před integrací služby do aplikace je nutné přes Google API Manager [13] vytvořit pro aplikaci projekt, registrovat v něm novém rozhraní a vygenerovat certifikát pro komunikaci s přihlašovacím rozhraním. Přesný postup je uveden v návodu pro zprovoznění služby Gitkit [14].

Přihlašování přes Gitkit je složeno ze dvou fází. První fáze začíná vyvoláním přihlašovací obrazovky, která se již nachází na serverech společnosti Google. Na této obrazovce si uživatel vybere, pomocí které služby se chce do aplikace přihlásit. Vybráním jedné ze služeb se nastartuje další část služby Identity toolkit, která již zajišťuje komunikaci se servery vybrané služby. Jakmile uživatel vyplní přihlašovací údaje a povolí použití svého účtu v aplikaci, vrací se dění zpět na servery Google. Zde začíná druhá fáze, kterou je zpracování získaných dat a jejich převod na univerzální identifikátor. Ten je vrácen zpět do aplikace, kde již může být použit libovolným způsobem.

2.3 Databázové schéma

Jelikož byl pro uložení dat vybrán SQL databázový systém, je nutné doménový model serveru převést na databázové schéma, podle kterého budou později vytvořeny jednotlivé tabulky a vztahy mezi nimi. Schéma je zachyceno na obrázku 2.3.

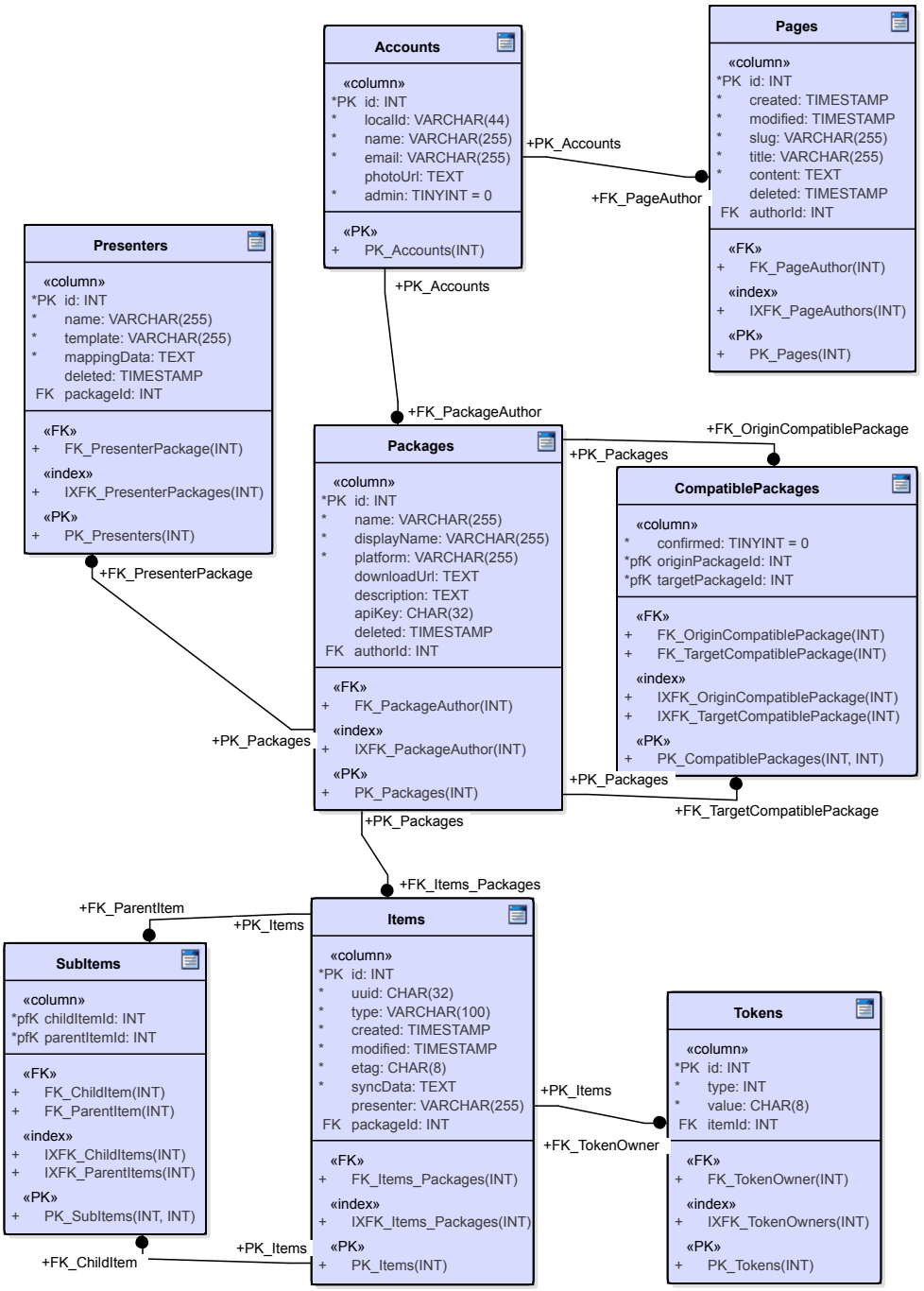
Accounts Tabulka Accounts vychází z původní entity Developer. Entita byla přejmenována z důvodu lepší spolupráce se službou Gitkit. Accounts obsahuje atributy `name`, `email` a `photoUrl`, které budou potřeba pro zobrazení uživatelského profilu na serveru v sekci pro vývojáře. Tabulka dále obsahuje atribut `localId`, který je vyžadován při identifikaci uživatele přes Gitkit. Kromě těchto identifikačních hodnot obsahuje tabulku také atribut `admin`, který určuje, zda má účet přidělená administrátorská práva.

Items Tabulka Items uchovává entitu Item. Nejdůležitějším z atributů této tabulky je atribut `syncData`, který uchovává data itemu jako JSON

[12] Google Identity Toolkit. `https://developers.google.com/identity/toolkit/`

[13] Google Developers Console API Library. `https://console.developers.google.com/apis/library?project=_`

[14] Configure Google Identity Toolkit. `https://developers.google.com/identity/toolkit/web/configure-service`



Obrázek 2.3: Databázové schéma

string. Na první pohled by toto řešení mohlo být rozporu s 1. normální formou, která by měla být splněna v rámci normalizace databáze. Již při řešení doménového modelu na straně 15 bylo zmíněno, že struktura těchto dat není předem známa. SyncData navíc nebudou nikdy použita k selekci nebo jiné operaci v SQL dotazech a lze je tedy považovat za ucelený blok dat, což 1NF vyhovuje. Atribut `type` slouží klientům pro zpětné rozpoznání itemu a jeho deserializaci. Tabulka dále obsahuje atribut `uuid`, který slouží jako jedinečný identifikátor v rámci celého systému. Další atribut, který stojí za zmínění, je `etag`. Tento atribut je používán v procesu synchronizace k identifikaci verze. Tabulka Items obsahuje cizí klíč do tabulky Packages, aby bylo jasné, odkud byl Item nahrán. Odkaz na Presenters není proveden přes cizí klíč, ale vyhledávání vhodného presenteru bude provedeno až při samotném vykreslení itemu. Přesný popis této akce je v sekci 2.6.3. Provázání mezi itemy je řešeno přes dekompoziční tabulku Subitems.

Packages Tabulka Packages reprezentuje stejnojmennou entitu z původního doménového modelu. Platforma, na kterou je balíček vyvíjen, je ukládána přímo do této tabulky jako string atribut `platform`, který se při načítání v kódu nahrazuje hodnotou z Enumeration.

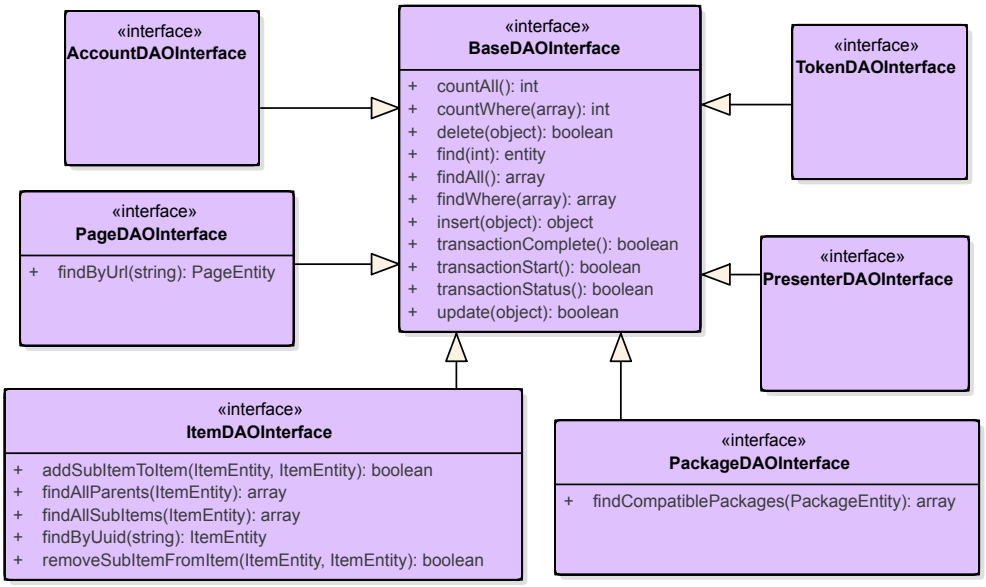
Pages Tabulka Pages obsahuje stránky vývojářské příručky odpovídající entitě Page. Tabulka kromě atributů entity obsahuje také cizí klíč do tabulky Accounts, který udává autora stránky.

Presenters Tabulka Presenters uchovává všechny presentery v systému. Tabulka obsahuje atributy `name`, který je použit při výběru vhodného presenteru pro item, a `mappingData`, který obsahuje předpis pro konverzi Item na data použitelná v šabloně. Obdobně jako u vztahu Package - Platform i zde je odkaz na šablonu nahrazen atributem `template` obsahující název šablony. Ten je v kódu převáděn na Enumeration. Práce s presentery je vysvětlena v sekci 2.6.3.

Tokens Tabulka Tokens udržuje všechny přístupové tokeny k itemům. Typ tokenu je ukládán jako číselný atribut `type`, které je po načtení převáděn na Enumeration. Tabulka kromě atributu `value`, udávající hodnotu přístupového tokenu, obsahuje také cizí klíč do tabulky Items, pomocí kterého je možné přiřadit přístupové tokeny k itemu.

2.4 Datová vrstva

Datová vrstva serveru se bude skládat z objektů, umožňujících přístup k datům v databázi. Tyto objekty budou zajišťovat čtení a zápis pro jednotlivé entit v systému (např. `Item` nebo `Token`).



Obrázek 2.4: Rozhraní datové vrstvy

Aby bylo možné přistupovat k datové vrstvě bez nutnosti přesně znát jednotlivé DAO třídy, budou pro ně vytvořena rozhraní (např. `ItemDAOInterface`). Ty budou rozšiřovat společné rozhraní `BaseDAOInterface`, které bude popisovat metody společné pro všechny DAO třídy. Přehled jednotlivých rozhraní včetně jejich metod je zachycen v podobě diagramu na obrázku 2.4.

Pro každé z těchto rozhraní bude vytvořena také odpovídající implementace. Základem pro všechny třídy bude třída `BaseDAO`. Ta bude obsahovat implementaci základních obecných operací se záznamy v databázi, jako například `insert(entity)` nebo `findWhere(condition)`.

Z třídy `BaseDAO` budou vycházet DAO třídy, které již budou pracovat s konkrétními entitami. Některé z těchto tříd, jako například `PresenterDAO`, nebudou obsahovat žádné další metody a vystačí si pouze se základními metodami. Opakem bude například třída `ItemDAO`, která bude kromě základních metod nabízet metody pro snadnější selekci dat (např. `findByUuid(uuid)`) nebo pro práci s kolekcemi (např. `addSubItemToItem(childItem, parentItem)`).

Pro komunikaci s databázovým systémem bude použita třída `CI_DB_Driver`, která je součástí frameworku CodeIgniter. Díky této třídě nebude nutné psát v metodách dotazy v jazyce SQL, ale bude možné využít vestavěného QueryBuilderu. Dotaz do databáze se bude vytvářet postupným voláním metod builderu. Po sestavení dotazu se QueryBuilder postará o převod na odpovídající SQL syntaxi. Výhodami tohoto řešení je snadný přechod na jiný databázový systém prostým vyměněním DB driveru.

2.5 Business vrstva

Chování serveru bude implementováno pomocí sady tříd, které budou tvořit business vrstvu aplikace. Na této vrstvě bude řešena veškerá logika serveru.

Na business vrstvu lze pohlížet jako na seznam všech dostupných operací, které server umožňuje. Tyto operace jsou rozčleněny do tříd podle entit, se kterými pracují. V případě business vrstvy už ale není vyžadováno, aby jedna třída pracovala s pouze jednou entitou. Míra propojení jednotlivých tříd je na vcelku vysoké úrovni a například pro vyřízení požadavku na úpravy itemu, kdy bude potřeba u požadavku ověřit API klíč i přístupový token, spolu můžou komunikovat až tři třídy logiky.

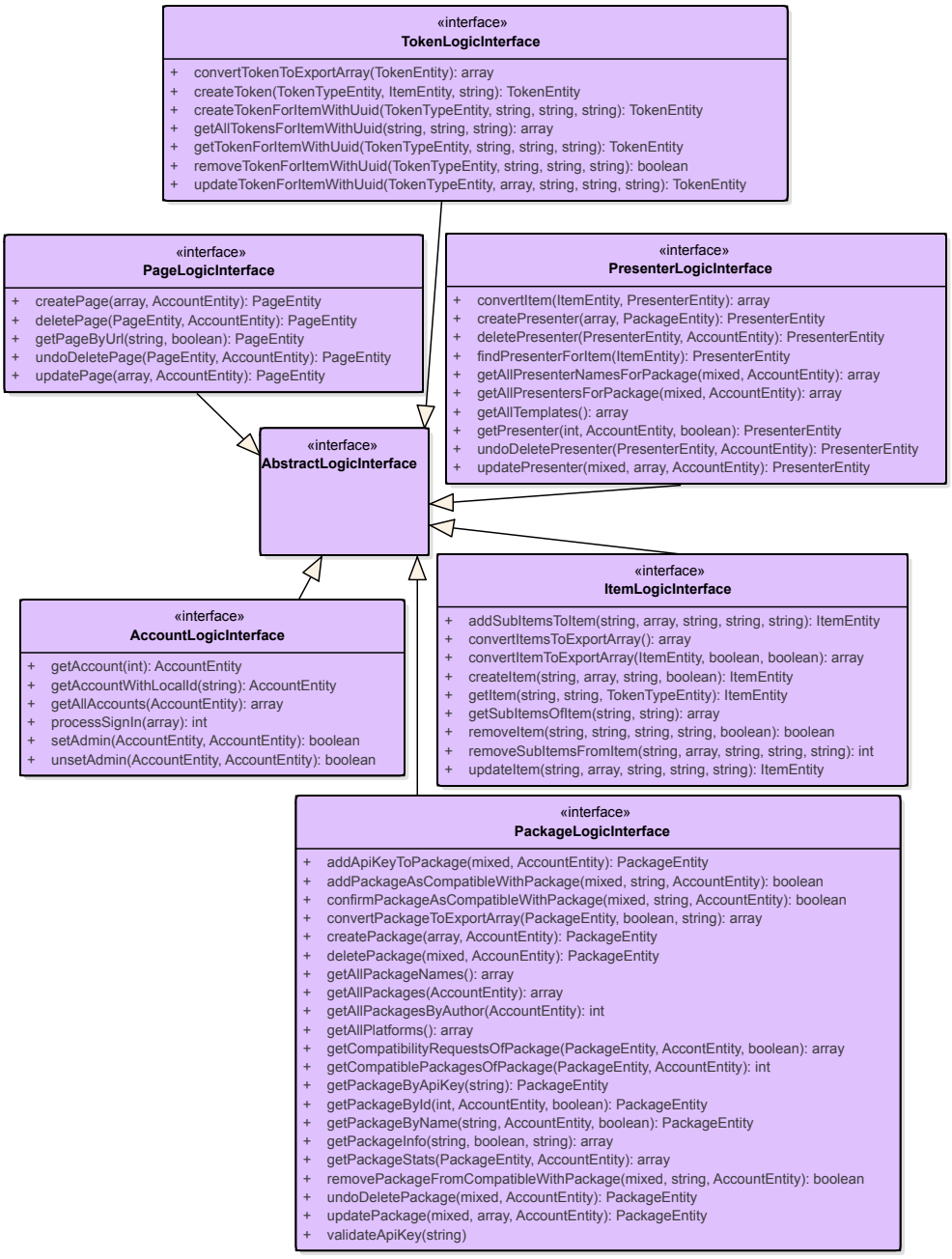
K business vrstvě se bude opět přistupovat přes rozhraní, aby bylo možné využívat třídy logiky ze všech částí prezentační vrstvy serveru. I zde je vytvořen jeden základní interface `AbstractLogicInterface`, který bude sloužit jako základ všech ostatních rozhraní. V případě business vrstvy sice nebude definovat žádné metody, ale pro udržení konzistentního přístupu k návrhu a pro případný budoucí rozvoj použito bude. Důležitá jsou rozhraní pro konkrétní logiky. Přehled jednotlivých rozhraní včetně jejich metod je zachycen v podobě diagramu na obrázku 2.5.

Pro jednotlivá rozhraní bude opět vytvořena i příslušná implementace. Základem tříd business vrstvy bude třída `AbstractLogic`, která bude obsahovat základní strukturu potřebnou k propojení s CodeIgniter frameworkem. Rozšířením třídy `AbstractLogic` budou vznikat konkretní třídy obsahující logiku spojenou s určitou entitou (např. `ItemLogic`). Každá z tříd je unikát a ačkoliv je možné mezi třídami najít určitou podobnost, například přítomnost metod pro vytvoření a úpravu záznamu, není možné jejich strukturu nijak obecně popsat.

Business vrstva bude často využívat data uložená v databázi. K těm se dostane přes datovou vrstvu, se kterou bude komunikovat pomocí rozhraní definovaných v předchozí části návrhu. Kromě datové vrstvy budou třídy business vrstvy při provádění operace využívat také různé další externí knihovny a nástroje.

2.6 Prezentační vrstva

Třetí a poslední vrstvou serveru je prezentační vrstva. Ta se stará o interakci s uživatelem nebo klientskými aplikacemi. Protože bude sbírat požadavky z různých zdrojů, je rozdělena na tři samostatná komunikační rozhraní. První rozhraní slouží pro vývojáře a umožňuje správu registrovaných balíčků, editaci presenterů a přístup k vývojářské příručce. Druhé rozhraní slouží pro komunikaci s klientskými aplikacemi. Pomocí tohoto rozhraní bude možné provádět synchronizaci soutěží. Třetí rozhraní je určeno pro účastníky soutěží, kteří si chtějí prohlédnout výsledky soutěží.



Obrázek 2.5: Rozhraní business vrstvy

Všechna tři rozhraní v prezentační vrstvě se budou skládat z controllerů. Controller je třída, která dostane na vstup příchozí požadavek a má se postarat o jeho zpracování. Součástí požadavku je sada parametrů, pomocí kterých je požadavek upřesněn. Požadavkem tedy může být například zobrazení detailu balíčku a parametrem tohoto požadavku bude unikátní název balíčku. O zpracování takového požadavku se postará metoda `detail(name)`, která bude součástí controlleru `Packages`. Tato metoda nejdříve provede kontrolu vstupu, například porovnáním proti regulárnímu výrazu. Následně předá vyřizování požadavku business vrstvě (v našem příkladu třídě `PackageLogic`), která ověří práva pro provedení takového požadavku a zajistí načtení dat balíčku pomocí datové vrstvy (v našem příkladu třídou `PackageDAO`). Data balíčku následně vrátí zpět do controlleru, který se postará o převedení dat na HTML dokument a ten vrátí jako odpověď uživateli.

O přesměrování požadavku na správnou metodu správného controlleru se bude starat Router, který je součástí frameworku. Routeru z každého požadavku zjistí na kterou URL je směrován a jaká HTTP metoda byla použita pro jeho odeslání. Podle těchto údajů bude schopen přesně vybrat metodu, která se o vyřízení takového požadavku postará.

2.6.1 Sekce pro vývojáře

První rozhraní bude určeno pro vývojáře a správce systému Tournament Manager. K tomuto rozhraní se bude přistupovat přes webový prohlížeč. V sekci pro vývojáře bude možné vytvořit a spravovat rozšiřující balíčky, měnit konfiguraci presenterů a prohlížet příručku pro vývojáře. Administrátorům zde bude umožněno spravovat všechny balíčky v systému a držet příručku pro vývojáře v aktuální podobě.

Pro většinu operací bude vyžadováno přihlášení vývojáře. Zde bude využívána služba Gitkit, zmíněná v sekci 2.2. Pro usnadnění komunikace s touto knihovnou bude vytvořen `SecuredController`, vycházející z původního `CI_Controller`. Třída `SecuredController` bude obsahovat metodu `requireActualAccount()`, která provede načtení aktuálního účtu z session dat a v případě jeho nepřítomnosti provede redirect na přihlašovací obrazovku. Všechny další controllery budou rozšiřovat `SecuredController` a budou tak mít k této metodě přístup. Pro načtení informací o přihlášeném uživateli tak bude stačit tuto metodu zavolat na začátku vyřizování požadavku.

Jednotlivé operace ve vývojářské sekci lze rozdělit na zápis a čtení. Metody, které budou vyřizovat operace čtení, nejprve ověří přihlášení uživatele (pokud bude vyžadováno) a následně zavolají příslušnou metodu z business vrstvy. Výsledek tohoto volání převedou pomocí šablon na HTML dokument, který vrátí zpět uživateli. Pro operace zápisu budou metody před zavoláním business vrstvy ještě zpracovávat formuláře, pomocí kterých budou od uživatele získávat data. Formuláře budou definované jako součást šablon a k jejich ověřování bude použita knihovna `CI_Form_Validation`, která je součástí fra-

meworku. Pomocí této knihovny je možné popsat jednoznačná pravidla pro jednotlivé položky formuláře a framework se již sám postará o jejich kontrolu a informuje uživatele v případě jejich porušení. Jakmile bude formulář vyplněn správně a metoda bude mít jistotu o správnosti vyplněných dat, pokračuje ve volání business vrstvy stejně jako v případě operací čtení.

2.6.2 Synchronizační API

Druhé rozhraní prezentační vrstvy serveru je synchronizační API. K tomuto rozhraní se přistupuje pomocí protokolu HTTP a využívat jej budou klientské aplikace systému Tournament Manager. Pomocí tohoto rozhraní bude možné nahrávat data na server nebo z něj aktuální podobu dat stahovat.

Controllery tohoto rozhraní budou vycházet z třídy `API_Controller`, která bude obsahovat metody usnadňující práci s požadavky. Třída `API_Controller` bude obsahovat metody pro načítání a zpracování HTTP hlavičky požadavku, ze které bude načítat například autentizační údaje.

Obdobně jako ve vývojářské sekci i zde lze každou metodu rozdělit na několik kroků. Prvním krokem je ověření dat požadavku, následuje volání business vrstvy, vyhodnocení vrácené hodnoty a odeslání odpovědi zpět klientské aplikaci.

2.6.2.1 Zabezpečení

Synchronizační rozhraní serveru bude používat dva ochranné prvky, které mají zabránit neoprávněnému čtení a úpravě dat na serveru. Tyto kontrolní prvky jsou navrženy jako kompromis mezi sílou zabezpečení a snadností použití.

Prvním bezpečnostním prvkem bude API klíč. Každý z balíčků, který bude komunikovat se synchronizačním rozhraním serveru, musí být registrovaný na serveru a musí mít vygenerovaný API klíč. Tento klíč bude klient přikládat ke každému požadavku, který jakkoliv upravuje data na serveru. Hlavním důvodem použití tohoto prvku je omezení podvodné komunikace, která by mohla přicházet vně systému Tournament Manager a u které hrozí, že by mohla poškodit data na serveru. Pokud by náhodou došlo k úniku API klíče balíčku mimo systém, bude nutné co nejdříve klíč deaktivovat a vygenerovat pro balíček nový klíč. Deaktivací klíče dojde okamžitě k zastavení funkce sdílení a synchronizace všem uživatelům postiženého balíčku a bude proto nutné formou opravného patche balíček aktualizovat.

Druhým bezpečnostním prvkem bude přístupový token. Token bude umožňovat přístup k datům pouze oprávněným uživatelům systému. Token bude mít podobu krátkého alfa-numerického řetězce textu, který bude opět přikládán k požadavku. Přístupový token bude generovaný serverem, který je bude uchovávat ve své databázi. Každý token se bude odkazovat na item, kterému patří, a bude definovat jakého je typu. Na serveru budou existovat tři typu přístupových tokenů: Owner (majitel), Write (zápis) a Read (čtení). Token typu

owner je určen pouze pro autora nahraného itemu a bude umožňovat veškeré operace s itemem, včetně jeho mázání. Token typu Write je určen pro osoby, se kterými chce autor item sdílet, a bude umožňovat čtení a úpravu itemu. Token typu Read je určen především pro účastníky turnajů a bude umožňovat pouze čtení itemu. Autor itemu obdrží při nahrání itemu na server sadu tokenů, jeden od každého typu. Další šíření tokenů je čistě na jeho uvážení. V případě nechtěnému úniku nebo vyzrazení tokenu je možné hodnotu tokenu změnit. Nová hodnota může být opět vygenerována serverem nebo je možné zadat hodnotu vlastní, což umožní jeho snadnější zapamatování a sdílení.

Token i ApiKey budou k požadavku přiloženy pomocí HTTP hlavičky Authorization. Specifikace HTTP umožňuje tuto hlavičku používat v rámci různých schémat, což umožňuje různé způsoby použití napříč celým internetem. Pro účely systému Tournament Manager bude použito vlastní schéma AccessToken, v rámci kterého bude možné odeslat Token a ApiKey, případně pouze jeden z uvedených. Autorizační hlavička bude mít následující tvar: `AccessToken apikey="<API-KEY>", token="<TOKEN>"`. Načítání jednotlivých položek z této hlavičky bude součástí třídy `API_Controller` a bude tak k dispozici ve všech controllerech synchronizačního rozhraní.

2.6.2.2 Struktura

Návrh komunikačního rozhraní bude vycházet z architektonického stylu REST. Bude se tedy skládat z několika zdrojů (definovaných pomocí URL) nad kterými bude možné provádět CRUD operace. Těmito zdroji budou Items, Tokens a Packages, což odpovídá entitám potřebným pro proces sdílení.

Všechny zdroje budou až na drobné rozdíly pracovat stejným způsobem. Podle zdroje bude požadavek předán příslušnému controlleru (např. `Items`). Následně bude podle druhu operace a použité HTTP metody vybrána správná metoda třídy. O jaký druh operace se jedná je dáno pomocí URL, kam byl požadavek zaslán. Pokud je požadavek poslán na URL odpovídající tvaru `<resource-name>/`, jedná se o požadavek pracující s kolekcí jako celkem. Takto je možné načíst přehled všech záznamů nebo přidat nový záznam do kolekce. Pokud je požadavek poslán na URL odpovídající tvaru `<resource-name>/<ID>`, jedná se o požadavek pracující s jedním určitým záznamem (definovaný podle ID). Tímto způsobem je možné získat detail záznamu, změnit ho nebo ho smazat.

Jak bylo uvedeno výše, jednotlivé zdroje se od tohoto obecného způsobu mohou mírně lišit. Některé operace nemusí podporovat vůbec nebo naopak budou podporovat další speciální operace. V rámci zdroje Items bude možné spravovat kolekci subitems jednotlivých itemů. Pro přidání itemu do kolekce subitems bude použita HTTP metoda PUT, pro mazání bude použita HTTP metoda DELETE. Zdroj Items bude navíc rozšířen o kontrolu verzí. Jednotlivé itemy budou mít vždy vygenerovaný Etag, což bude alfa-numerický identifikátor verze. Do jisté míry si lze etag představit jako hash vypočítaný pro

soubor. Při úpravě nebo mazání bude nutné spolu s požadavkem poskytnou Etag verze, kterou má klient aktuálně k dispozici. Pokud se bude poskytnutý Etag shodovat s Etagem na serveru, pak bude požadavek proveden, protože klient vycházel z poslední verze. Pokud se Etagy shodovat nebudou, pak bude požadavek zamítnut, protože klient vychází ze starých dat a mohlo by dojít k nechtěnému přepsání verze, o které klient nevěděl. Průběh úpravy itemu je zachycen na sekvenčním diagramu 2.6. Aby mohl server zjistit, zda je jeho lokální verze aktuální, je možné spolu s požadavkem `GET/items/<UUID>` poslat také Etag. Pokud je verze aktuální, server informuje klienta, že nedošlo k žádným změnám, v opačném případě mu rovnou pošle verzi aktuální. Průběh porovnání verzí je zachycen na sekvenčním diagramu 2.7.

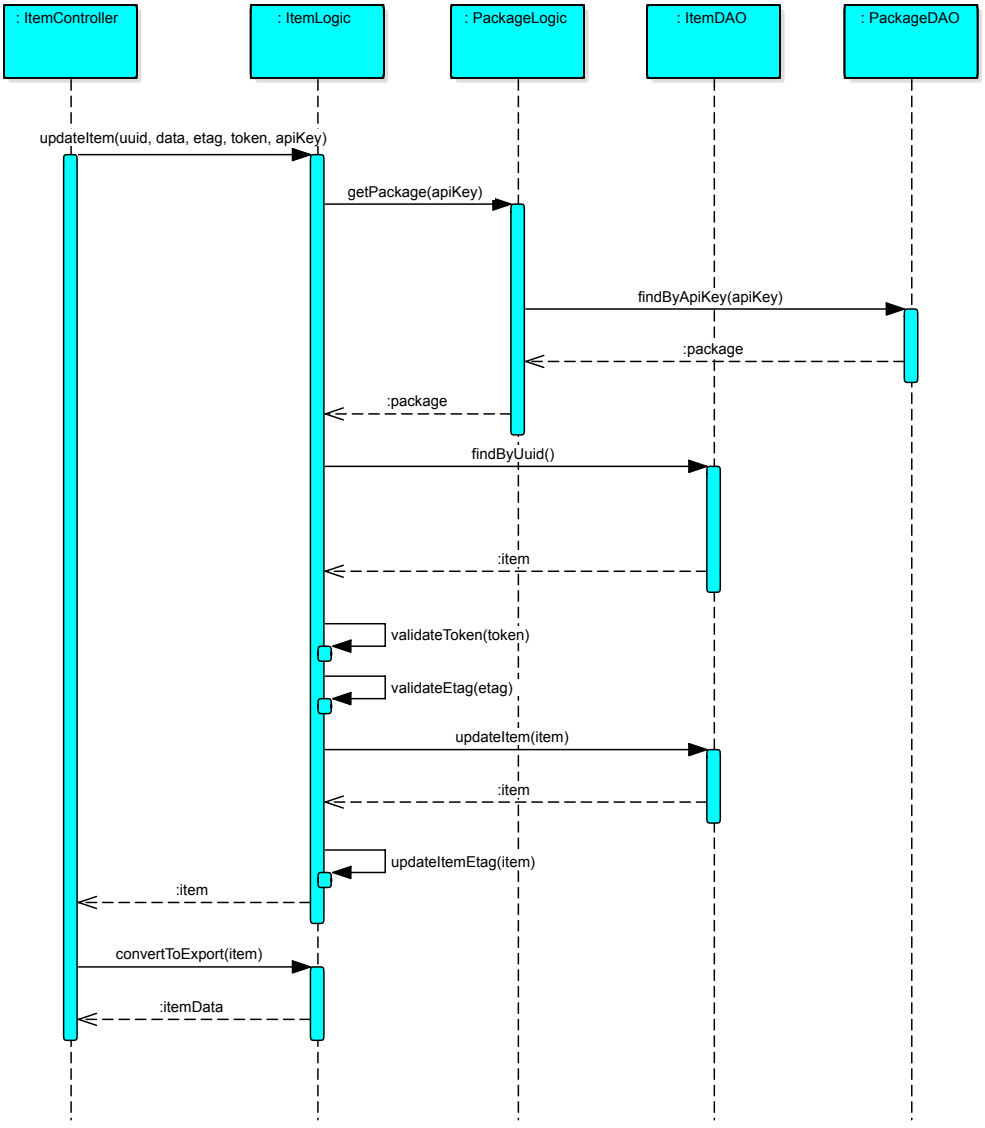
2.6.2.3 Komunikace

Většina operací v rámci synchronizačního API bude používána samostatně. Při nahrání soutěže na server klient odešle všechna data soutěže pomocí `POST/items/<UUID>`. Server vytvoří požadovaný item, uloží do něj poskytnutá data a klientovi zpět vrátí sadu přístupových tokenů. Obdobným způsobem je řešeno stažení aktuální verze ze serveru. Klient pomocí `GET/items/<UUID>` stáhne data ze serveru a ty si odpovídajícím způsobem uloží do lokální databáze. Výjimkou jsou operace nahrání změn na server. V rámci této operace by správně měl klient nejdříve zjistit, zda na serveru není nová verze pomocí `GET/items/<UUID>`. Pokud je na serveru novější verze, musí si tuto verzi před nahráním nových změn stáhnout. Nyní je již jisté, že je lokální verze aktuální a může pokračovat v nahrání změn na server pomocí `PUT/items/<UUID>`. Operace nahrání změn na server je zachycena sekvenčním digramem na obrázku 2.8.

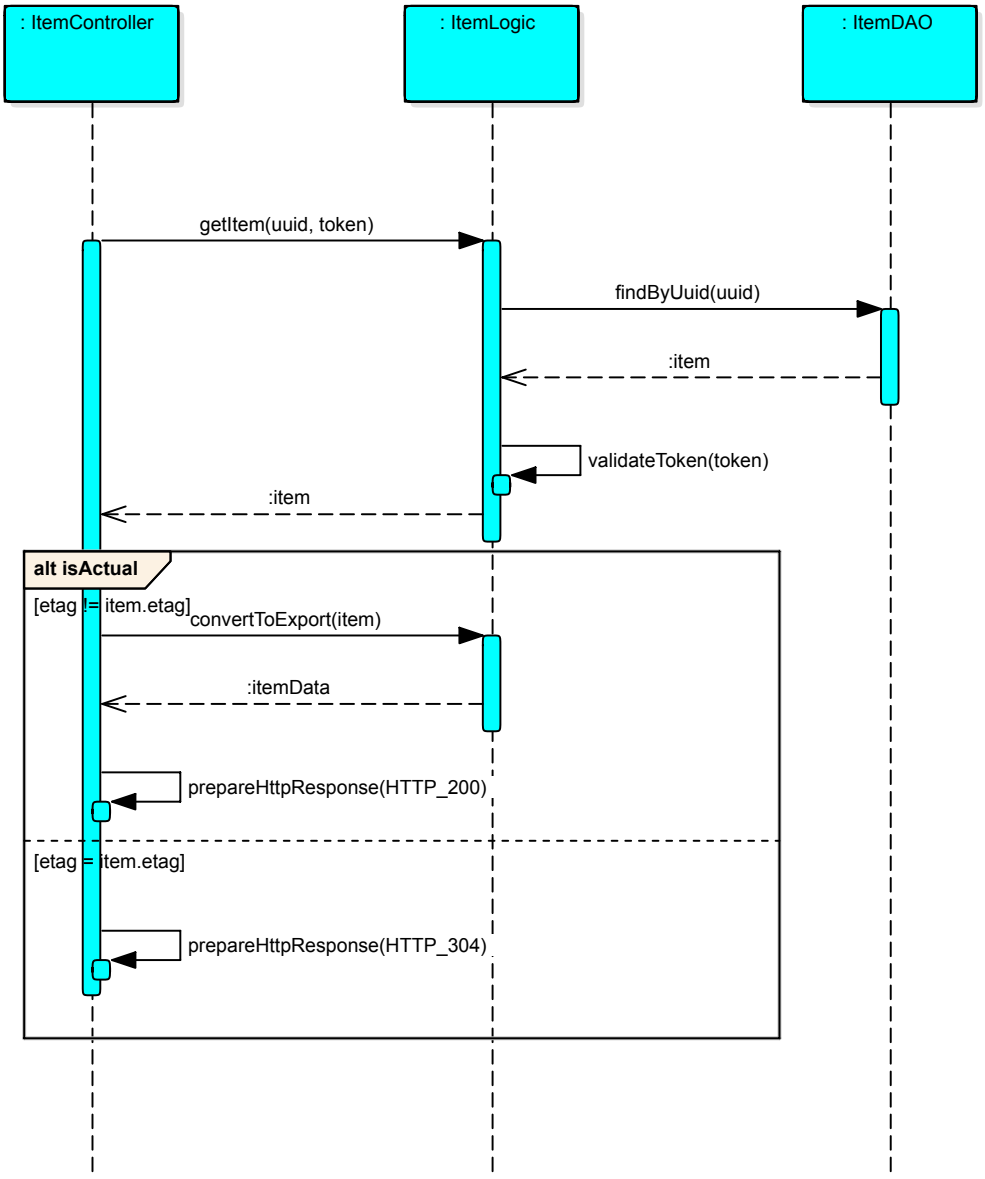
2.6.3 Prezentace dat

Třetím a posledním rozhraním prezentační vrstvy je Prezentace dat. Toto rozhraní je určeno především pro hráče, kteří se účastní soutěží vedených v systému Tournament Manager. K tomuto rozhraní se přistupuje pomocí webového prohlížeče.

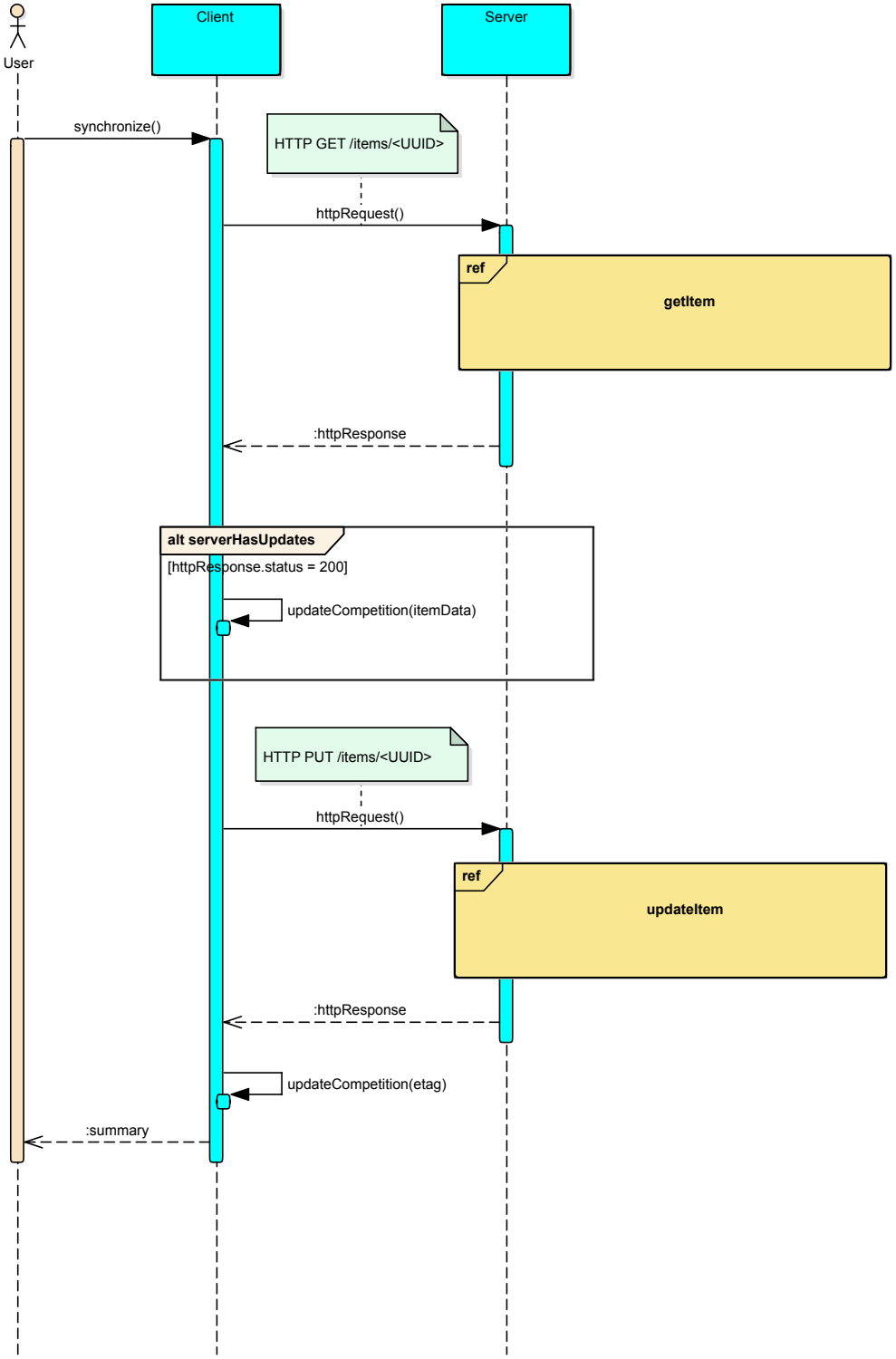
Rozhraní pro prezentaci dat bude obsahovat pouze jeden controller `Items` obsahující jednu metodu `render`. Tato metoda nejprve pomocí třídy `ItemLogic` získá item, který má být vykreslen. Následně pomocí třídy `PresenterLogic` získá presenter vhodný pro vykreslení zvoleného itemu a ten spolu se samotným itemem předá znovu třídě `PresenterLogic` k provedení konverze. Jakmile se konvertovaná data vrátí zpět do metody `render`, bude stačit už jen tato data vykreslit pomocí šablony definované v presenteru a výsledný HTML dokument vrátit zpět uživateli. Tento proces je zachycen sekvenčním diagramem na obrázku 2.9.



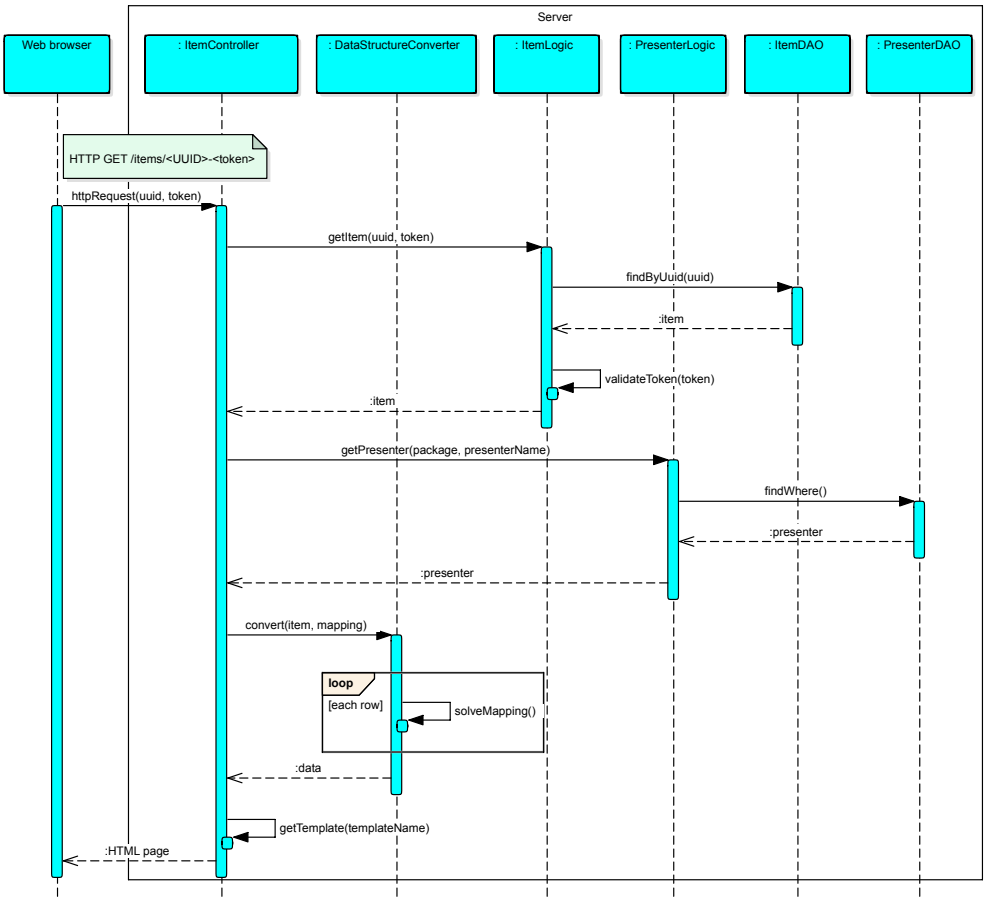
Obrázek 2.6: Aktualizace itemu



Obrázek 2.7: Získání itemu



Obrázek 2.8: Synchronizace soutěže



Obrázek 2.9: Vykreslení itemu

2.6.3.1 Zobrazení dat neznámé struktury

Skoro ve všech částech vývoje serveru se naráželo na fakt, že server o struktuře nahrávaných dat nic bližšího nevěděl. Zatím bylo možné data považovat za nedělitelný celek a tak je také ukládat nebo zpracovávat. Pro prezentaci dat je ale potřeba nahraná data procházet, zobrazovat a dokonce nad nimi provádět agregační nebo aritmetické operace.

Na serveru bude vytvořeno několik různých šablon, optimalizovaných pro různé entity v systému Tournament Manager. Bude tak existovat například šablona `player_detail` nebo `match_1v1`. Každá z těchto šablon bude na svém vstupu očekávat určitou datovou strukturu. Vývojář si z dostupných šablon vybere tu, která svou formou nejvíce odpovídá entitě, kterou chce prezentovat.

Aby bylo možné zobrazit item pomocí vybrané šablony, bude nutné z dat itemu vytvořit datovou strukturu, kterou šablona vyžaduje. K tomuto úkolu bude sloužit nástroj `DataStructureConverter`. Vstupy tohoto nástrojem jsou item, ze kterého se mají čerpat data, a mappingData, která si lze představit

jako návod, pomocí kterého se bude vytváře výsledná struktura. Jednotlivé kroky mappingDat budou zapsány ve formátu `«target>": «value>"`. Obě hodnoty budou zapsány ve formátu, který bude vycházet z technologie JSONPath, kterou publikoval Prof. Dr.-Ing. Stefan Gössner [16]. JSONPath funguje, jak již název může napovídat, obdobně jako XPath pro XML. Pomocí řetězce je možné zapsat cestu napříč datovou strukturou. Příklad mappingDat je možné si prohlédnou na ukázce 2.1, která znázorňuje data potřebná pro převod itemu hráče pro použití v šabloně `player_detail`.

Ukázka kódu 2.1: mappingData - player_detail

```
{
  "player.name": "#item.syncData.name",
  "player.email": "#item.syncData.email",
  "player.note": "#item.syncData.note"
}
```

2.6.3.2 Konverze dat

Při konverzi itemu bude `DataStructureConverter` postupně procházet mappingData a při každém kroku uloží na cílovou pozici získanou hodnotu. Pro ukázku 2.1 tedy v prvním kroku nejprve vytvoří položku `player` a v ní vytvoří položku `name`. Následně na tuto pozici uloží hodnotu atributu `name` ze syncDat itemu. V dalším kroku bude již položka `player` existovat, takže do ní pouze přejde a vytvoří v ní novou položku `email`, kam uloží odpovídající hodnotu z itemu. Konvertor takto pracuje dál, dokud nejsou zpracována celá mappingData.

O něco složitější bude práce s kolekcemi. Postup konvertoru bude opět předveden na vzorovém případu. Ukázka 2.2 obsahuje mappingData pro zobrazení týmu pomocí šablony `team_detail`. První krok těchto mappingData bude zpracován již známým způsobem. Při vyhodnocování druhého kroku již dochází k mírné odlišnosti. Nejprve budou vybrány subItemy konvertovaného itemu. Ty budou následně předány funkci `where`, která provede jejich filtraci. Výstupem této funkce bude kolekce obsahující ty položky původní kolekce, které vyhovují dané podmínce `type.equals('player')`. Konvertor tedy pro každou položku kolekce subItems spustí vyhodnocení této podmínky. Při vyhodnocování bude vybrán atribut `type`, který bude předán další funkci `equals`. Ta porovná hodnotu na vstupu s další podmínkou a vrátí boolean hodnotu. V tomto případě je podmínka definována staticky pomocí stringu, nicméně funkci equals bude možné jako parametr předat další výraz, který by byl před porovnáním opět vyhodnocen. Kolekce, která bude výsledkem celého tohoto volání, se nakonec uloží na pozici `team.members`. Konvertor pokračuje následujícím krokem. Tento krok se bude od předchozích lišit již při výběru cílové pozice. Protože je na pozici `team.members` kolekce, je nutné konvertoru

říct, že má následující krok udělat pro každý prvek této kolekce, což bude zapsáno instrukcí `#each`. Hodnoty v jednotlivých krocích už budou zjištěny klasickým způsobem.

Ukázka kódu 2.2: mappingData - team_detail

```
{
  "team.name": "#item.syncData.name",
  "team.members":
    "#item.subItems.where(type.equals('player'))",
  "team.members.#each.name": "#node.syncData.name",
  "team.members.#each.email": "#node.syncData.email"
}
```

Kromě zmíněných funkcí `where` a `equals` bude konvertor nabízet mnoho dalších funkcí, pomocí kterých bude možné provádět selekci dat nebo jejich vyhodnocení. Pro práci s kolekcemi budou dostupné funkce pro přístup přes index (např. `first` nebo `nthElement`) nebo funkce `map`, která umožňuje každý prvek transformovat. Pro agregaci dat budou dostupné funkce pro hledání extrémů (`min` a `max`) nebo pro výpočet (`avg`, `sum` a `count`). V neposlední řadě budou dostupné funkce pro základní aritmetické operace (např. `add` nebo `divide`) a základní logické operace (např. `exists` nebo `both`). Všechny funkce a jejich použití bude popsáno v samostatné dokumentaci.

Kapitola 3

Realizace

Realizace serveru začala přípravou vývojového prostředí. Nejdříve bylo používáno prostředí lokálního serveru, kde bylo možné připravit framework a implementovat první dvě vrstvy. S vývojem sekce pro vývojáře bylo nutné přejít na server, který byl dosažitelný ze serverů Google. V opačném případě fungovala služba Gitkit pouze v jakémsi nouzovém režimu. Server byl vyvíjen na osobním hostingu autora práce. Na konci realizace bude provedena rešerše dostupných hostingů, která může fungovat jako základní přehled pro zadavatele, až se rozhodne přesunout server do vlastní správy.

Proces realizace začal přípravou potřebných technologií. Na server byla nasazena základní distribuce frameworku CodeIgniter 3, která byla propojena s nově vytvořenou MySQL databází. Následně byla upravena struktura frameworku pro použití třívrstvé architektury.

3.1 Datová vrstva

Nejprve byla implementována datová vrstva serveru. Jako základ byla použita třída `CI_Model`, která byla rozšířena implementací rozhraní `BaseDAOInterface`. Tím vznikla třída `BaseDAO` obsahující metody pro komunikaci s databází z původního modelu a základní operace specifikované v `BaseDAOInterface`.

S připraveným základem bylo možné pokročit k implementaci DAO tříd pro jednotlivé entity. Pro každou entitu byl systém rozšířen o migraci, která vytvořila v databázi tabulku potřebnou pro uložení dat. Následně byla vytvořena třída, jejíž atributy odpovídají atributům vytvořené tabulky. Tato třída slouží jako container, pomocí kterého je možné jednotlivé záznamy posílat mezi vrstvami. Následovala implementace odpovídající DAO třídy.

3.2 Business vrstva

S vytvořenou datovou vrstvou bylo možné pokročit v architektuře výše na business vrstvu. Business vrstva je řešena pomocí samostatných tříd a do frameworku je integrována v podobě knihovny.

Nejprve byla implementována základní třída `AbstractLogic`, která se stará o propojení s frameworkem. Implementace tohoto propojení je řešena v konstruktoru třídy. Protože business vrstva bude vyžadovat ke své práci vrstvu datovou, bylo nutné přijít se způsobem, jak k datové vrstvě přistupovat. K tomuto účelu je využit ServiceLocator, který je součástí frameworku. Způsob použití ServiceLocatoru je zachycen v ukázce 3.1. S dostupným propojením s datovou vrstvou již bylo možné začít s implementací jednotlivých tříd logiky.

Ukázka kódu 3.1: přístup k třídám v CodeIgniter

```
// autoload.php
...
$autoload['libraries'] = array(
  'logic/PackageLogic' => 'PackageLogic',
  'logic/ItemLogic' => 'ItemLogic',
  ...
  );

$autoload['model'] = array(
  'PackageDAO' => 'PackageDAO',
  'ItemDAO' => 'ItemDAO',
  ...
  );

// usage in ItemLogic.php
...
$this->CI->PackageLogic->validateApiKey($apiKey);
...
$this->CI->ItemDAO->find($id);
```

3.3 Sekce pro vývojáře

Prvním realizovaným rozhraním byla vývojářská sekce. Toto rozhraní je nutné pro umožnění registrace balíčku a vygenerování API klíče, který bude vyžadován pro komunikaci se synchronizačním rozhraním.

Nejprve bylo nutné připravit `SecuredController`, který obsahoval metody potřebné pro ověření přihlášeného uživatele. Spolu s tímto controllerem byla připravena jednoduchá knihovna `GitkitLibrary` (viz ukázka 3.2) obalující klienta Google Identity Toolkit. Poté došlo k postupné implementaci jednotlivých funkcí vývojářské sekce. Pro každou funkci byla nejprve vytvořena metoda

v příslušném controlleru, která byla následně doplněna o odpovídají šablonu. Při vytváření šablon byl použit front-end framework MaterializeCSS[15].

Ukázka kódu 3.2: metoda getActualAccountData třídy GitkitLibrary

```
public function getActualAccountData() {
  $gitkitUser = $this->gitkitClient->getUserInRequest();

  if (!$gitkitUser) {
    return NULL;
  }

  $userData = [
    'localId' => $gitkitUser->getUserId(),
    'name' => $gitkitUser->getDisplayName(),
    'email' => $gitkitUser->getEmail(),
    'photoUrl' => $gitkitUser->getPhotoUrl()
  ];

  return $userData;
}
```

3.4 API

S existující vývojářskou sekcí bylo možné pokročit k vývoji synchronizačního rozhraní. To bylo nejdříve zachyceno v podobě dokumentace, aby bylo možné paralelně se serverem vyvíjet i klienty pro mobilní zařízení. Dokumentace popisuje jednotlivé zdroje API a všechny operace, které je možné s nimi provádět. Popis operace obsahuje výpis podporovaných parametrů, popis struktury těla požadavku, popis struktury odpovědi a přehled HTTP stavových kódu, které může operace vrátit. Dokumentace je dostupná na přiloženém CD.

Ačkoliv výchozí `CI_Controller` umožňuje odpovídat na požadavky formou HTTP Response bez použití HTML šablony, toto řešení je trochu kostrbaté. Z tohoto důvodu byl použit balík CodeIgniter Rest Server [16], který výrazně zjednodušuje implementaci REST rozhraní v CodeIgniteru. Tento balík obsahuje třídu `REST_Controller`, která má být použita jako rodič controllerů REST rozhraní. V controllerech pak stačí na konec názvu připsat název HTTP metody (např. `detail_get` nebo `detail_put`) a `REST_Controller` se již postará o výběr správné metody pro příchozí požadavek. Pro synchronizační rozhraní serveru bude `REST_Controller` rozšířen na `API_Controller`, který

[15] MaterializeCSS. `http://materializecss.com/`

[16] CodeIgniter Rest Server. `https://github.com/chriskacerguis/codeigniter-restserver`

bude přidávat metody určené pro načítání API klíče a přístupového tokenu z hlavičky HTTP požadavku.

S připraveným zázemím bylo možné přejít k implementaci jednotlivých controllerů synchronizačního rozhraní. Vytvořená dokumentace značně usnadnila vývoj rozhraní. Při vývoji bylo nutné neustále zkoušet chování rozhraní. Pro zasílání zkušebních požadavků byl použit nástroj REST Console [17], který funguje jako rozšíření do internetového prohlížeče Google Chrome. Pomocí tohoto nástroje je možné jednoduše vytvořit požadavek, včetně všech HTTP hlaviček. Přijatá odpověď je následně analyzována a zobrazena v jednoduše čitelné formě. Pro názornost je přiložena ukázka 3.3, která zachycuje operaci načtení itemu. Tuto operaci je možné najít v podobě sekvenčního diagramu na obrázku 2.7 na straně 40.

3.5 Prezentace dat

Jakmile server umožňoval nahrání itemu, bylo možné začít s vývojem rozhraní umožňující prezentaci dat. Nejdůležitější částí tohoto rozhraní je třída `DataStructureConverter`, jejíž funkce je popsána již v kapitole 2 Návrh v části 2.6.3 Prezentace dat na straně 38. Při vývoji tohoto rozhraní byla třída `PresenterLogic` z business vrstvy doplněna o metodu `convertItem(item, presenter)`, která se stará o vytažení mappingDat z presenteru a jejich použití při konverzi itemu.

Třída `DataStructureConverter` je řešena jako samostatný prvek, který postupným průchodem mappingDat převádí vstupním item na výslednou datovou strukturu. Celý proces konverze je založen na souhře dvou metod `solveMapping` a `getValue`. Metoda `solveMapping` se stará o zpracování jednoho řádku mappingDat. Jejím úkolem je najít správnou pozici ve výsledné datové struktuře, kam se má uložit výsledek výrazu. Jakmile je nalezena správná pozice, spustí se vyhodnocení výrazu pomocí metody `getValue`. Tato metoda postupně vykonává instrukce výrazu a aplikuje je na hodnotu v paměti. Instrukce jsou rozlišeny na step (krok) a func (volání funkce). Instrukce step provede krok datovou strukturou. Příkladem může být výraz `"person.homeAddress.street.name"`, který udává cestu, složenou z kroků, na jejímž konci je název ulice. Instrukce func provede zavolání některé z interních funkcí konvertoru. Příkladem může být výraz `"apples.add(oranges)"`, který znázorňuje součet dvou druhů ovoce. Používání třídy `DataStructureConverter`, resp. celých presenterů je zachyceno v příručce pro prezentaci, která je dostupná na přiloženém CD.

3.5.1 Vzorová data

Aby bylo možné odladit chování třídy `DataStructureConverter` a zbytek celé prezentace dat, bylo nutné vytvořit sadu testovacích dat, která by do největší

[17] REST Console. `http://restconsole.com`

Ukázka kódu 3.3: metoda detail_get třídy Items

```
public function detail_get($uuid) {
  $token = $this->getToken();
  if ($token === NULL) return;

  log_message('debug','API Request: Get item "'.$uuid.'"('.
      $token.') ('.$this->input->ip_address().').');

  try {
    $item = $this->ItemLogic->getItem($uuid, $token);
  } catch (AccessDeniedException $e) {
    $this->processAccessDenied($e);
    return;
  } catch (NotFoundException $e) {
    $this->response(API_Error::get($e->getNotificationName())
        , self::HTTP_NOT_FOUND);
    return;
  }

  $etag = $this->input->get_request_header('If-None-Match',
      TRUE);

  if (!is_null($etag) && $item->etag === trim($etag, '"')) {
    $this->response(NULL, self::HTTP_NOT_MODIFIED);
    return;
  }

  $includeSyncData = ($this->input->get('syncData') === 'true
      ');
  $recursive = ($this->input->get('recursive') === 'true');

  $itemData = $this->ItemLogic->convertItemToExportArray(
      $item, $includeSyncData, $recursive);

  $this->response($itemData, self::HTTP_OK);
}
```

možné míry představovala možné datové struktury v systému Tournament Manager. Protože je struktura nahrávaných dat plně v režii autora balíčku, byl jako vzor vybrán balíček pro hokej [2]. V době vytvoření těchto dat nebyl žádný z klientů připraven pro komunikaci se serverem a testovací data proto vycházela z předpokládaného formátu pro synchronizaci.

3.5.2 Příprava šablon

Aby byl celý proces prezentace dat smysluplný, bylo nutné vytvořit na serveru šablony, pomocí kterých jsou konvertovaná data vykreslena. V rámci implementace rozhraní pro prezentaci dat byly vytvořeny šablony pro jednotlivé entity používané v klientských aplikacích. Je velmi pravděpodobné, že server bude postupně rozšiřován o další šablony, které umožní vykreslení speciálních turnajů nebo specifických prvků některých sportů.

Pro vytvoření šablon bylo představeno několik abstraktních komponent, pomocí kterých jsou šablony vytvářeny. Jednou z takových komponent je tabulka umožňující řazení podle vybraných sloupců. Pro funkci řazení sloupců bylo použito javascriptové řešení tablesorter [18]

3.6 Přehled hostingů

Při vytváření webové aplikace se prakticky vždy počítá s jejím nasazením na server připojený k síti. Většinou se jedná o připojení k internetu, ale velké organizace často vytváří vlastní intranety s lokálními servery. Aplikace a webové stránky psané v PHP lze nasadit na libovolný server, na kterém je dostupný interpretr PHP.

Server je možné provozovat v rámci vlastního hardware a internetového připojení. Cenu tohoto řešení lze určit vcelku přesně. Je třeba sečíst cenu použitého HW a dále uvážit cenu internetového přípojení o dostatečné rychlosti a spotřebované energie. Výsledný server je následně nutné správně nakonfigurovat a výhledově i udržovat, což obnáší pravidelné údržby, zálohování a řešení případných problémů. Samozřejmě jsou případy, kdy je provozování vlastního serveru výhodné, ale v případě malých projektů se rozhodně nejedné o vhodné řešení.

Další možností je využít některý z dostupných hostingů. Ty jsou většinou řešeny v podobě cloudového řešení typu PaaS. Typický web-hosting se skládá z Unix-based operačního systému a některé ze serverových služeb (Apache, NGINX apod.), která je již správně nakonfigurovaná. Uživatel hostingu dostane pouze prostor na disku, kam nahraje svojí PHP aplikaci pomocí FTP, SCP nebo Git přístupu. Většina hostingů umožňuje uživateli částečně upravit chování hostingu, jako např. změnit používanou verzi PHP nebo způsob zobrazování chyb. Vcelku často je k hostingu dodáván přístup k dalším službám. Typickým příkladem takové služby je e-mailový server nebo plánování událostí pomocí Cron. Ceny hostingů jsou různé v závislosti na velikosti diskového prostoru, rychlosti připojení a míře dodávaných služeb. Cenu může ovlivnit také kvalita služeb, především garantovaná dostupnost. Je možné najít hostingy, které jsou zdarma a slouží jako startovní můstek pro nové projekty. Diskový prostor je výrazně omezen, ale většinou je dostatečný pro množství základ-

[18] tablesorter. `http://tablesorter.com`.

ních webových stránek. Dalším častým omezením je limit na datový přenos. Toto omezení souvisí s rozsahem webových stránek a jejich návštěvnosti. Tyto limity jsou ale opět nastaveny tak, aby začínající weby neměly výrazné problémy.

Existuje i kombinace předchozích dvou možností v podobě pronájmu celého serveru. Tehdy se jedná o cloudové řešení typu IaaS. Poskytovatel takové služby má veškerý HW ve svých serverovnách a uživateli je umožněn přístup k nim pomocí RDP, SSH nebo jiných metod vzdáleného ovládání. Uživatel tak může s tímto serverem pracovat jako kdyby ho měl vedle sebe, ale nemusí řešit nákup a opravy HW ani jeho provozování. Poskytovatel dále může celý server připravit podle požadavků uživatele a pomoct s řešením případných problémů. Ceny těchto cloudů jsou sice vyšší, než je tomu v případě webového hostingu, ale ani vzdáleně nedosahují na cenu provozování vlastního serveru srovnatelné kvality.

3.6.1 Výběr hostingu pro výsledný server

Pro účely systému Tournament Manager je vyžadován server přístupný z internetu. Z důvodu nepříliš velké rozsáhlosti serverové části systému stačí pro nasazení serverové části systému klasický web-hosting. Nic ale nebrání případnému přechodu na jinou z výše uvedených možností, například z důvodu lepší kontroly nad zálohováním dat. Existuje spousta webových hostingů rozdílných kvalit. V této práci není možné pokrýt většinu z nich a už vůbec ne všechny. Služby zde uvedené byly vybrány na základě zkušeností autora nebo podle referencí z internetu.

První možností je WEDOS Internet, a.s.[19] Za posledních 5 let roste jeho obliba nesrovnatelně s ostatními a aktuálně se jedná o nejrozšířenejší hosting pro českou doménu [17]. Společnost aktuálně nenabízí webhosting zdarma, který by bylo možné použít v rámci této práci, ale hosting NoLimit je více než dostatečný a při pohledu na poskytované služby a možnosti je jeho cena skoro zanedbatelná.

Druhou možností je FORPSI, resp. společnost INTERNET CZ, a.s.[20] Dle statistik dostupných na internetu má společnost stojící za tímto hostingem na svém účtu nejvíce registrací českých domén [18]. Bohužel, ani FORPSI nenabízí webhosting zdarma, nicméně jejich základní webhosting Easy je srovnatelný parametry, cenou i kvalitou služeb s předchozím uvedeným hostingem. Zajímavostí hostingu od FORPSI je možnost provozovat na něm aplikace napsané v jiném jazyce než PHP, jako jsou například .NET, Python nebo Ruby.

Třetí možností je společnost Red Hat, Inc. a jejich OpenShift [21]. Podobně jako FORPSI nabízí OpenShift prostor pro aplikace psané ve několika různých jazycích. V čem se OpenShift výrazně liší od předchozích dvou zmíněných, je

[19] WEDOS Hosting. `https://hosting.wedos.com`
[20] FORPSI. `https://www.forpsi.com/`
[21] OpenShift. `https://www.openshift.com/`

způsob jak se platí za poskytované služby. Platba se neprovádí za určité období, ale za chování aplikace. Do ceny se promítá doba běhu aplikace v kombinaci s výkonem serveru, na kterém aplikace běží, nebo velikost používané databáze. Ceny za jednotlivé body se liší v závislosti na zvoleném plánu. Společnost aktuálně nabízí možnost využít hosting zdarma, ale s příchodem nové verze systému jsou ohlášeny změny v plánech.

Čtvrtou možností je Heroku [22], nyní udržované společností Salesforce.com, Inc. Služba je velice podobná službě OpenShift, zásadním rozdílem je přítomnost Free plánu, který je sice určen převážně pro vývoj aplikací, ale je možné ho využívat i dlouhodobě.

[22] Heroku. `https://www.heroku.com/`

Kapitola 4

Testování

Součástí vývoje software by vždy mělo být i jeho testování. Nejvyšší pozornost byla věnována synchronizačnímu API, které je nejrozsáhlejší částí serveru. Většina situací a stavů byla testována uživatelsky již při implementaci serveru. Autor práce a ostatní členové týmu se snažili přijít s různými dotazy, většinou obsahující úmyslnou chybu, které by mohly server dostat do neošetřeného stavu. Takový stav byl samozřejmě několikrát objeven a server byl ihned adekvátně ošetřen.

Pro server bylo vytvořeno také několik automatických testů. K jejich vytvoření byl použit nástroj ci-phpunit-test [23], který umožňuje použití testovacího frameworku PHPUnit [24] v CodeIgniteru. Testy jsou součástí systému a je možné je najít ve složce `application/tests`. Členění této složky odpovídá členění složky `application` (např. testy pro `application/libraries/DataStructureConverter.php` jsou součástí souboru `application/tests/libraries/DataStructureConverter\_test.php`).

4.1 Unit testy

Unit testy byly vytvořeny pro ty části systému, u kterých dochází k složitějším výpočtům a akcím. Testována byla především business vrstva systému a další důležité části knihovny. Jednou z takových částí je například třída `DataStructureConverter`, která zajišťuje funkčnost prezentace výsledků. Pro tuto třídu byly vytvořeny testy, které ověřují korektní provedení různých konverzí. Příklad takové testu je možné si prohlédnou v ukázce 4.1.

Pomocí unit testů byly otestovány také třídy tvořící business vrstvu serveru. U těchto testů byly datová třídy nahrazeny mockem, aby bylo možné testovat chování business tříd samotných. Na ukázce 4.2 je zachycen jeden z takových testů pro třídu `PageLogic`, ve kterém je použit mock třídy `PageDAO`.

[23] ci-phpunit-test. `http://kenjis.github.io/ci-phpunit-test/`.
[24] PHPUnit. `https://phpunit.de/`

Ukázka kódu 4.1: DataStructureConverter - unit test

```
public function test_convert_collection_iterator() {
  $item = (object) ['members' => [
      'boys' => [
        ['name' => 'James', 'surname' => 'Doe', 'age' =>
            13]],
      'girls' => [
        ['name' => 'Jane', 'surname' => 'Doe', 'age' => 9],
        ['name' => 'Susan', 'surname' => 'Doe', 'age' => 16]
      ]
    ]];
  $mappingData = json_encode([
    "people" => "#item.members.flatten()",
    "people.#each.fullName" => "concat(concat(#node.name,
        string(' ')), #node.surname)",
    "jamesAge" => "#root.people.where(fullName.equals('James
        Doe')).first().age",
    "janeAge" => "#root.people.where(fullName.equals('Jane
        Doe')).first().age",
    "susanAge" => "#root.people.where(fullName.equals('Susan
        Doe')).first().age",
  ]);

  $result = $this->obj->convert($item, $mappingData);

  $this->assertArrayHasKeyThatEquals('jamesAge',13,$result);
  $this->assertArrayHasKeyThatEquals('janeAge',9,$result);
  $this->assertArrayHasKeyThatEquals('susanAge',16,$result);
}
```

4.2 Integrační testy

Nástroj ci-phpunit-test umožňuje kromě klasických unit testů také virtualizovat příchozí požadavek a testovat, jakým způsobem ho systém zpracuje. Takto je možné testovat propojení jednotlivých vrstev aplikace a chování aplikace jako celku. Tento typ testů byl použit pro testování synchronizačního API, aby bylo zajištěno chování odpovídající vytvořené dokumentaci.

Ukázka kódu 4.2: PageLogic - unit test

```
public function test_getPageByUrl_returnDeleted() {
  $testUrl = 'the-url';

  $page = (object) [
    'id' => '42',
    'url' => $testUrl,
    'title' => 'test page',
    'content' => '<p>test content</p>',
    'authorId' => 1,
    'deleted' => date("Y-m-d H:i:s", strtotime("NOW - 1day"))
    ];

  $returnCollection = [$page];

  $pageDAOMock = $this->getMockBuilder('PageDAO')->getMock();
  $pageDAOMock->method('findWHere')->willReturn(
      $returnCollection);

  $this->verifyInvokedOnce(
      $pageDAOMock,
      'findWhere',
      [['url' => $testUrl]]
    );

  $this->obj->CI->PageDAO = $pageDAOMock;

  $actual = $this->obj->getPageByUrl($testUrl, TRUE);

  $this->assertEquals($page, $actual);
}
```

Ukázka kódu 4.3: API Controller Items - integrační test

```
/**
 *  @depends test_detail_post_valid_data
 */
public function test_detail_get_existing($metadata) {
  $this->request->setHeader('Authorization', 'AccessToken
      token="'.$metadata['token'].'"');

  $output = $this->request('GET', '/api/v1/items/7
      d748d7e419940392ec500f54eab66f2');

  $this->assertResponseCode(200);

  $responseEntity = json_decode($output, TRUE);

  $this->assertArrayHasKeyThatEquals('uuid', '7d748d7e
      -4199-4039-2ec5-00f54eab66f2', $responseEntity);
  $this->assertArrayHasKeyThatEquals('etag', $metadata['etag'
      ], $responseEntity);
  $this->assertArrayHasKeyThatEquals('type', 'test_item',
      $responseEntity);
  $this->assertArrayNotHasKey('syncData', $responseEntity);
  $this->assertArrayNotHasKey('subItems', $responseEntity);
  }
```

Závěr

Cílem této práce bylo vytvořit serverovou část systému Tournament Manager, která by umožňovala synchronizaci dat mezi klientskými aplikacemi systému a prezentaci výsledků soutěží na webu. V práci je zaznamenán kompletní proces vývoje, v rámci kterého byl server vytvořen.

- Byla provedena analýza jednotlivých procesů systému, ze kterých bylo definováno očekávané chování serveru ve formě funkčních a nefunkčních požadavků.
- Na základě požadavků byl server navržen. V návrhu bylo vyřešeno, jakým způsobem bude server implementován a jaké technologie budou použity k jeho realizaci.
- Dle návrhu byl server do posledního bodu implementován. Výsledný server byl otestován pomocí jednotkových a integračních testů.

Výsledkem práce je plně funkční server připravený k zapojení do systému Tournament Manager. Server splňuje všechny požadavky, které byly definovány na začátku této práce a plně odpovídá navržené architektuře. Rozhraní pro synchronizaci je zdokumentováno a nyní již vyčkává na první použití. Rozhraní pro prezentaci výsledků je vybaveno základní sadou šablon, které by měly stačit pro zobrazení všech aktuálně implementovaných balíčků. Server je připraven na případné rozšíření o nové šablony.

Budoucnost serveru

Věřím, že dokončením serveru nekončí má účast na vývoji systému Tournament Manager. Již před dokončením této práce se začalo řešit rozšíření serveru o další rozhraní, které by umožňovalo centralizovanou distribuci balíčků systému.

Za zvážení bude také stát přechod na novější verzi frameworku CodeIgniter, jehož verze 4 je momentálně dostupná pro testování komunitou a lze očekávat její vydání v řádech měsíců. Díky oddělení logiky serveru do samostatných knihoven by měl být přechod na novou verzi vcelku jednoduchým úkolem.

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] Máca, V.: *Tournament Manager – klient pro Windows 10 Mobile*. Diplomová práce, Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2017, [cit. 2017-01-01].

[5] Firebase Realtime Database. In: *Firebase* [online]. Únor 2017, [cit. 2017-02-13]. Dostupné z: `https://firebase.google.com/docs/database/`

[6] Couchbase Mobile. In: *Couchbase* [online]. 2017, [cit. 2017-02-13]. Dostupné z: `https://www.couchbase.com/nosql-databases/couchbase-mobile`

[7] Realm Mobile Platform. In: *Realm* [online]. 2017, [cit. 2017-02-13]. Dostupné z: `https://realm.io/products/realm-mobile-platform/`

[8] Hanselman, S.: A reminder on "Three/Multi Tier/Layer Architecture/Design"brought to you by my late night frustrations. In: *Scott Hanselman Blog* [online]. Červen 2004, [cit. 2017-01-26]. Dostupné z: `http://www.hanselman.com/blog/A-Reminder-On-Three-Multi-Tier-Layer-Architecture-Design-Brought-To-You-By-My-Late-Night-Frustrations.aspx`

[9] Usage of server-side programming languages for websites. In: *W3Techs* [online]. 2016, [cit. 2016-12-22]. Dostupné z: https://w3techs.com/technologies/overview/programming_language/all

[10] Server-side programming languages market position report. In: *W3Techs* [online]. 2016, [cit. 2016-12-22]. Dostupné z: https://w3techs.com/technologies/market/programming_language

[11] Programming languages used in most popular websites. In: *Wikipedia* [online]. Leden 2017, [cit. 2017-01-26]. Dostupné z: https://en.wikipedia.org/wiki/Programming_languages_used_in_most_popular_websites

[12] DB-Engines Ranking. In: *DB-Engines* [online]. jan 2017, [cit. 2017-01-27]. Dostupné z: http://db-engines.com/en/ranking

[13] Buckler, C.: SQL vs NoSQL: How to Choose. In: *sitepoint* [online]. Září 2015, [cit. 2017-01-27]. Dostupné z: https://www.sitepoint.com/sql-vs-nosql-choose/

[14] Spotify Engineering (@SpotifyEng): Twitter post: *http://spotify.com is powered by the @symfony framework and many great packages from http://packagist.org using Composer by @seldaek*. Květen 2015, [cit. 2017-01-27]. Dostupné z: https://twitter.com/SpotifyEng/status/595972071609339905

[15] Skvorc, B.: The Best PHP Framework for 2015: SitePoint Survey Results. In: *SitePoint* [online]. 2015, [cit. 2016-12-23]. Dostupné z: https://www.sitepoint.com/best-php-framework-2015-sitepoint-survey-results/

[16] Gössner, S.: JSONPath - XPath for JSON. *Mechanik, das Web und der ganze Rest*, Září 2007, [cit. 2017-02-14]. Dostupné z: http://goessner.net/2007/09/jsonpath-xpath-for-json.html

[17] Webhosting. In: *cz.nic* [online]. 2016, [cit. 2016-12-27]. Dostupné z: https://stats.nic.cz/stats/hosting/

[18] Domény podle registrátorů. In: *cz.nic* [online]. 2016, [cit. 2016-12-27]. Dostupné z: https://stats.nic.cz/stats/domains_by_registrar/

Příloha **A**

Seznam použitých zkratek

API Application programming interface

CRUD Create, read, update, delete

DAO Data access object

Gitkit Google Identity Toolkit

GUI Graphical user interface

JSON JavaScript Object Notation

ORM Object-relational mapping

REST Representational state transfer

URL Uniform resource locator

UWP Univeral Windows Platform

Příloha **B**

Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `docs`
  - `synchronization_api.pdf` .. dokumentace synchronizačního rozhraní
  - `data_presentation.pdf` ..... příručka pro prezentaci výsledku
- `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