ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Název:	Systém pro skórování Ultimate Frisbee zápasů - backend
Student:	Marek Dostál
Vedoucí:	Ing. Jiří Hunka
Studijní program:	Informatika
Studijní obor:	Softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	Do konce letního semestru 2016/17

Pokyny pro vypracování

Cílem práce je navrhnout a implementovat backend webové aplikace (dále jen aplikace) pro zaznamenávání frisbee zápasů a následné poskytování statistik. Aplikace je navrhována a vyvíjena v rámci dvou souběžných bakalářských prací, které budou sdílet společnou analýzu a návrh požadavků. Tato práce se zaměří na návrh a implementaci backendu.

1. Společně s Jaroslavem Veselým (realizuje bakalářskou práci zaměřenou na frontend) proveďte analýzu a návrh požadavků na aplikaci.
2. Na základě analýzy proveďte návrh backendu (datový model a business logika).
3. Proveďte implementaci backendu společně s komunikačním API serverem.
4. Navrhněte vhodnou sadu testů a svou implementaci jimi otestujte.

Seznam odborné literatury

Dodá vedoucí práce.

L.S.

Ing. Michal Valenta, Ph.D.
vedoucí katedry

prof. Ing. Pavel Tvrdík, CSc.
děkan

V Praze dne 11. listopadu 2015

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

Bakalářská práce

Systém pro skórování Ultimate Frisbee zápasů - backend

Marek Dostál

Vedoucí práce: Ing. Jiří Hunka

16. května 2016

Poděkování

Chtěl bych poděkovat všem, kteří se podíleli na vzniku této práce, zejména Karolíně Zubaté a Davidovi Viktorovi za cenné rady a korekturu textu. Dále bych chtěl poděkovat rodině a přátelům, kteří mne podporovali po celou dobu studia.

Prohlášení

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

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 46 odst. 6 tohoto zákona tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou, a veškeré jejich dokumentace (dále souhrnně jen „Dílo“), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla, a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu), licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 16. května 2016 .....

Odkaz na tuto práci

Dostál, Marek. *Systém pro skórování Ultimate Frisbee zápasů - backend.* Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2016.

Abstrakt

Tato bakalářská práce se zabývá návrhem a realizací backendu webové aplikace sloužící ke správě turnajů a tvorbě statistik v Ultimate Frisbee. Podstatou řešení je vývoj RESTového rozhraní v Pythonu a jeho nasazení do provozu pomocí uWSGI. Výsledkem je funkční backend nezávislý na frontendu.

Klíčová slova webová služba, REST API, Python, framework Falcon, Ultimate Frisbee

Abstract

This bachelor's thesis deals with design and realization of the web application backend. It will be used to help manage tournaments and generate statistics in Ultimate Frisbee. The essence of the solution lies in the development of REST API in Python and putting it into operation via uWSGI. Functional backend, which is independent of frontend, is the result of the project.

Keywords web service, REST API, Python, Falcon framework, Ultimate Frisbee

Obsah

Seznam obrázků

Seznam tabulek

Úvod

Létající talíř, tzv. frisbee, je pro mnoho lidí známým pojmem, ale o existenci kompetitivního kolektivního sportu, který se s tímto předmětem hraje už skoro 50 let, ví málokdo. Ultimate Frisbee se dnes hraje téměř ve všech koutech světa a jeho popularita v posledních letech raketově stoupá. Důsledkem je větší počet turnajů, na kterých se týmy mezi sebou utkávají. Organizace takové akce ale není vždy triviální záležitostí, proto je potřeba hledat možnosti, které pomohou k zefektivnění práce pořadatelů.

V případě menších turnajů se často setkáváme s řešením, které je pro jeden konkrétní případ dostačující. Výsledky zápasů se uchovávají například v textových souborech, excelovských tabulkách nebo na statických webových stránkách. Pro konkrétnější využití dat nebo dlouhodobější sledování statistik jsou ale předchozí metody naprosto nevhodné.

Hlavním cílem je navrhnout a implementovat backend pro webovou aplikaci, která bude zajišťovat správu turnajů, zaznamenávání zápasů a následné poskytování výsledků a statistik. Součástí práce bude analýza dosavadních i možných řešení, testování a nasazení na server. Backend, v mém případě webová služba, bude nabízet veřejné rozhraní pro implementaci klientských aplikací. Frontend bude vyvíjen v rámci souběžné bakalářské práce Jaroslava Veselého. Dílčími cíly je pak seznámení čtenáře s použitými technologiemi a sepsání dokumentace k API.

Motivací při tvorbě práce mi byla především skutečnost, že sám jsem aktivním hráčem Ultimate Frisbee. Z toho důvodu disponuji velkou snahou celý projekt dotáhnout do úspešného konce. Vznikající systém má pracovní název Catcher.

Kapitola 1

Úvod do problematiky

V této kapitole se seznámíme s pojmy, které jsou pro další postup v práci klíčové. Největší část se zaměří na popsání sportu Ultimate Frisbee, zbytek se věnuje techničtějším pojmům, konkrétně aplikačnímu rozhraní, protokolu HTTP a webové službě.

1.1 Co je to Ultimate Frisbee

Ultimate Frisbee je mladý a dynamicky se rozvíjející sport s létajícím talířem, který se hraje od roku 1968 [1]. Hraje jej přibližně sedm miliónů hráčů ve více než 80 zemích světa a jeho popularita rok od roku stoupá [2]. Během posledních několika let je běžné sledovat živé přenosy z amerických soutěží, především profesionální ligy AUDL [3], na sportovním kanálu ESPN [4]. Ultimate, jak se mu zkráceně říká, v roce 2015 dokonce získalo uznání od Mezinárodního olympijského výboru [5]. Nejčastěji se hraje v kategoriích open (muži), ženy, mix (smíšené týmy mužů a žen), junioři (do 19 let) a masters (nad 33 let).

1.1.1 Pravidla

Popularitě přidává fakt, že jde o hru celkem vzato nenáročnou na vybavení s jednoduchými pravidly:

> *Ultimate je kolektivní bezkontaktní sport, v němž vítězí tým, který má na konci hrací doby vyšší počet bodů. Hraje se na hřišti o rozměrech cca 100x37 metrů (délka fotbalového hřiště, polovina jeho šířky). Na obou koncích hřiště jsou vyznačeny koncové zóny o hloubce cca 18 metrů.*
>
> *V ultimate proti sobě hrají dva sedmičlenné týmy. Smyslem hry je pomocí přihrávek dopravit disk do soupeřovy koncové zóny a jeho chycením v zóně získat bod. Po chycení disku se hráč musí zastavit a do 10 vteřin disk přihrát spoluhráči. Povoleným pohybem hráče*

s diskem je pivotování, tedy otáčení se kolem vlastní osy s jednou nohou pevně na zemi. V ultimate hráči často střídají útok a obranu při ztrátě disku, ke které dochází záhozem disku do autu, na zem, jeho zachycením soupeřem nebo při dlouhém držení disku. Není povolen fyzický kontakt mezi hráči ani přetahování o disk. [1]

Obrázek 1.1: Fotografie z amerického časopisu TIME ze zápasu mezi univerzitními týmy z Pittsburgu a Floridy [6]

1.1.2 Spirit of the Game

Už od počátku je Ultimate Frisbee založeno na hodnotách, jež kladou zodpovědnost za fair play na samotné hráče. Očekává se vysoce kompetitivní hra, avšak bez ztráty vzájemné ohleduplnosti a vytracení radosti ze hry. Všechna provinění vůči pravidlům řeší samotní hráči. Jako jediná sportovní hra se tak obejde bez rozhodčích, a to i na nejvyšších soutěžích, kterými jsou mistrovství Evropy a světa [7].

Hraní fair play je otázka cti. Na každém turnaji je vyhlašována cena Spirit of the Game (dále jen „SOTG“), která je cenou pro ty, kteří se chovali nejčestněji. Po každém zápase se týmy navzájem ohodnotí v podobě číselného hodnocení a cenu pak získá tým s nejvyšším průměrem. Cena SOTG je ceněna obdobně jako 1. místo.

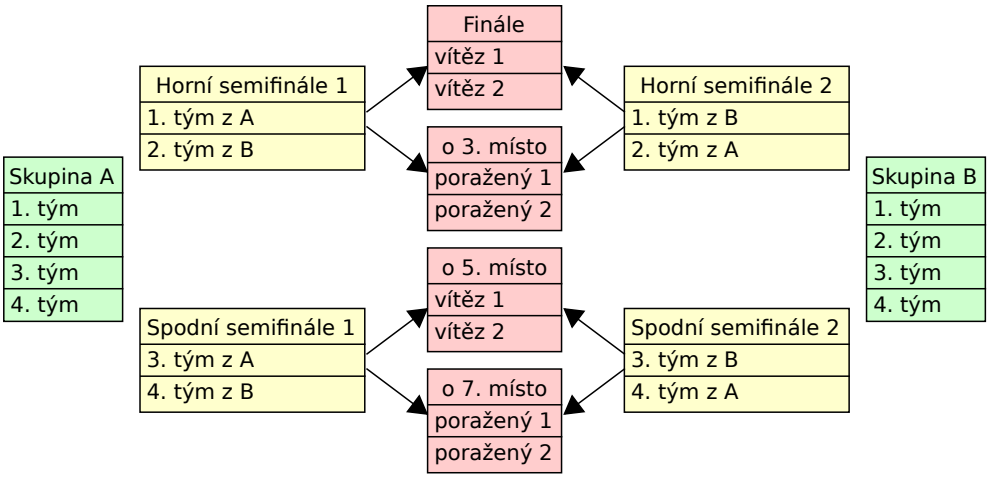
1.1.3 Ultimate v České republice

V České republice zastřešuje sporty s létajícím talířem již od roku 1991 [8] Česká asociace létajícícho disku (dále jen „ČALD“). V celé republice eviduje desítky zaregistrovaných klubů, tzv. oddíly, které se mezi sebou utkávají v rámci celého roku na turnajích. Jenom v loňském rok jich bylo na našem území přes třicet [9].

Většina turnajů nebo mistrovství trvá zpravidla více dnů, během kterých se odehrají desítky utkání. S přibývajícím počtem hráčů i fanoušků pak vzniká čím dál větší poptávka po online přenosech a statistikách z těchto akcí.

1.1.4 Jak vypadá turnaj

Běžný turnaj v Ultimate Frisbee se odehrává stylem *skupina-playoff*. Všechny týmy jsou rozlosovány do základních skupin, ve kterých se mezi sebou utkají. Celkový počet vítězství, popř. celkové skóre pak určí pořadí ve skupině a týmy jsou nalosovány do vyřazovacích zápasů, tzv. play-off. Vítězné týmy v těchto zápasech postupují turnajem nahoru a poražené bojují o co nejlepší umístění vzhledem k situaci, ve které jsou. Když využijeme analogii z jiných sportů, jde o zápasy typu semifinále, kde vítězný tým postoupí do finále a poražený do zápasu o třetí místo. Názornou ukázku turnaje lze vidět na obrázku 1.2.



Obrázek 1.2: Struktura jednoduchého turnaje, kterého se účastní 8 týmů

Dalším typem turnaje, který se běžně hraje, je liga. Týmy jsou pouze v jedné skupině, kde je po odehrání všech vzájemných zápasů určeno finální pořadí.

1.2 Aplikační rozhraní

Aplikační rozhraní se častěji označuje zkratkou API[1]. V informatice se tímto pojmem označuje sbírka procedur, funkcí, tříd nebo protokolů nějaké knihovny, jež mohou ostatní programátoři používat. Jde v podstatě o abstrakci, která popisuje rozhraní pro interakci s řadou programových celků, které programátor využívá namísto toho, aby je sám programoval. Určuje, jak budou knihovní funkce volány ze zdrojového kódu. API může mít následující vlastnosti.

- **Jazykově závislé API** - Lze volat pouze v daném programovacím jazyce, pomocí základních prvků jazyka.
- **Jazykově nezávislé API** - Jeho použití je možné ve více programovacích jazycích, což je žádoucí vlastnost služeb orientovaných na API, které nejsou vázány na určitý systém nebo proces. Tato služba pak může být poskytnuta jako webová služba.

1.3 Protokol HTTP

Protokol HTTP[2] je základním transportním protokolem na síti internet. Jde o protokol typu *požadavek-odpověď*. Se serverem naváže klient spojení, odešle mu požadavek a následně obdrží odpověď. Spojení je pak ukončeno. Požadavek představuje textový dokument, ve kterém dochází ke specifikování parametru dotazu. Konkrétní požadavek můžeme vidět na následující ukázce.

```
GET /api/player/1 HTTP/1.1
Host: catcher.zlutazimnice.cz
```

Na prvním řádku se nachází název HTTP metody (HTTP obsahuje řadu metod, které mají různý význam, více v tabulce 1.1), identifikace požadovaného zdroje a verze protokolu. Na následujicích řádcích najdeme tzv. „hlavičky“ (ve formátu *klíč: hodnota*), které specifikují parametry spojení. Jestliže chceme zaslat společně s požadavkem nějaká data k dalšímu zpracování, umístí se až za hlavičky a jeden prázdný řádek.

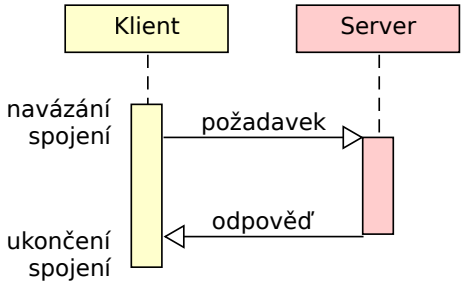
Odpověď serveru je opět textový dokument, ve kterém se nachází verze protokolu a stavový kód, který reprezentuje výsledek požadavku. Server může vrátit stavový kód z celkem pěti kategorií, jejich popis je uveden v tabulce 1.2. Za stavovým kódem je ještě krátký textový popisek. Na dalších řádcích jsou hlavičky a výsledek požadavku, který je oddělen prázdným řádkem, identicky jako v požadavku. Následující ukázka zobrazuje typickou odpověď z Catchera.

```
HTTP/1.1 200 OK
Server: nginx/1.6.3
```

[1] Application Programming Interface
[2] Hypertext Transfer Protocol

```
Date: Mon, 02 May 2016 19:18:18 GMT
Content-Type: application/json; charset=utf-8
Content-Length: 140
Connection: keep-alive
access-control-allow-origin: *
access-control-allow-headers: Content-Type,Authorization,X-Name
access-control-allow-methods: PUT,POST,DELETE,GET

{
  "ranking": null,
  "clubId": 11,
  "firstname": "Pavel",
  "caldId": 976,
  "lastname": "Matoušek",
  "number": 11,
  "nickname": "Maty",
  "id": 1
}
```



Obrázek 1.3: Sekvenční diagram HTTP komunikace; zdroj: autor na základě [10]

Metoda	**Význam**
GET	Základní metoda, jejíž účel je získat požadovaný zdroj. Neměla by mít žádné postranní efekty, jako tvorba nebo mazaní zdrojů.
POST	Metoda pro odesílání dat ke zpracování (data jsou vložena do těla požadavku). Tato metoda může být použita pro tvorbu nebo úpravu zdrojů.
DELETE	Metoda určená k mazání zdrojů.
PUT	Podobně jako POST slouží k odeslání dat ke zpracování. Zpravidla je tato metoda používána pro úpravu zdrojů.

Tabulka 1.1: Přehled vybraných metod HTTP protokolu v kontextu této práce [11]

Skupina kódů	Význam
1xx	Informační kódy.
2xx	Kódy označující úspěšné vykonání požadavku.
3xx	Přesměrování - tyto kódy označují odpověď, která obsahuje adresu, na které se nachází požadovaný zdroj.
4xx	Chybové kódy označující problém při zpracování způsobený klientem (odeslání nesprávných dat). Před opakováním požadavku je nutné opravit vstupní data.
5xx	Chybové kódy onzačující problém při zpracování, který vznikl na straně serveru. Požadavek je možné opakovat.

Tabulka 1.2: Kategorie návratových kódů HTTP protokolu [10]

1.4 Webová služba

Webová služba je zapouzdřená funkcionalita nějaké aplikace, kterou využívají další aplikace. Mluvíme tedy o strojové interakci, nevhodné například pro prohlížení ve webovém prohlížeči. S veřejně vystavenou webovou službou komunikují ostatní aplikace pomocí definovaného rozhraní. Pro realizaci API se dnes nejčastěji využívají dvě možnosti - SOAP [12] a REST [13].

Kapitola 2

Specifikace požadavků

Zadání této práce vzniklo původně z požadavku na rozšíření již fungující mobilní aplikace Catcher. Její autor mne na konci roku 2015 oslovil s nápadem vylepšit její serverou část a doplnit API, které by mohla využívat libovolná klientská aplikace. Po rychlé analýze jsme však oba došli k závěru, že bude lepší navrhnout nový backend, protože ten starý nebyl reálně rozšiřitelný.

Finálním řešením je tak webová služba, která poskytuje kompletní rozhraní, tzn. jde o zcela nezávislou vrstvu mezi frontendem a backendem. Protože výsledek mé práce má později nahradit nevyhovující backend staré aplikace, ponechal jsem celému projektu jméno Catcher.

2.1 Požadavky na Catchera

Protože se v komunitě hráčů i organizátorů turnajů jako aktivní hráč sám pohybuji, nebyl problém stanovit funkční i nefunkční požadavky na vznikající službu.

2.1.1 Funkční požadavky

Import a export dat Systém umožňuje import oddílů a jeho hráčů z databáze ČALD.

Rozpis zápasů Oprávněná role může vytvořit svůj vlastní turnaj a vložit seznam všech zápasů, které se budou hrát. Vytvořený rozpis zápasů systém již doplňuje automaticky na základě odehraných výsledků (např. vítěze semifinále automaticky posune do finále). Ve chvíli, kdy to bude již možné, doplní celkové pořadí turnaje. Automatické doplnění se týká i souhrnné tabulky vzájemných hodnocení v kategorii SOTG.

Zadávání dat v rámci turnaje Každý tým má možnost vytvořit vlastní soupisku na turnaj. Systém pak umožňuje v průběhu turnaje zadávat konkrétní údaje:

- průběžné skóre zápasů nebo jejich závěrečný výsledek
- skórující a asistující hráč
- hodnocení SOTG

Tvorba statistik Systém tvoří detailní statistiky hráčů, týmů a zápasů (obdržené a udělené body, počet asistencí, průměrná hodnota SOTG).

2.1.2 Nefunkční požadavky

Snadná rozšiřitelnost Už nyní evidujeme změny, o které je zájem, ale nejsou předmětem této práce. I proto je nutné projekt dokončit tak, aby byl v budoucnu snadno rozšiřitelný nebo modifikovatelný.

Nízká cena Cílem není vytvořit výdělečný projekt, ale fungující službu pro několik stovek hráčů a fanoušků v České republice. I proto je požadavkem použití volně dostupných knihoven a technologií.

Výkon Podle celkem jednoduchých odhadů lze usoudit, že aplikaci budou čekat výkyvy v provozu. Většina zápisů a čtění dat probíhá během samotných turnajů. Během špičky nebude počet požadavků za sekundu větší než několik desítek. I proto není na celkový výkon kladen žádný zvláštní požadavek. Systém by každopádně měl minimálně 95 % žádostí zpracovat do jedné sekundy.

Spolehlivost Spolehlivost je základem pro funkční běh Catchera. Velká část operací, včetně chybných budou zapisovány do souborů pro snadné odhalení chyb. Obnova musí být proveditelná ze zálohovacích souborů.

Bezpečnost Systém musí jednoznačně určit a ověřit uživatele, který přistupuje k rozhraní Catchera. Zároveň musí existovat možnost za chodu přidávat, odebírat nebo měnit oprávnění uživatelů. Pro tento projekt není nutné používat HTTPS[3] protokol.

Nároky na hardware Systém musí být schopen běžet na běžných serverech s ne více jak 1024 MB RAM[4].

Formát importu Pro import soupisek musí systém umět číst data ve formátu, který ČALD pro export používá.

Vytvoření dokumentace Pro vývoj klientských aplikací je potřeba vytvořit dostatečně detailní dokumentaci rozhraní.

[3] Hypertext Transfer Protocol Secure
[4] Random Access Memory

2.2 Uživatelské role

Nepřihlášený návštěvník Jde o nejčastější přístup k aplikaci. Slouží k zobrazení všech statistik (aktuální skóre, statistika všech hráčů, hodnocení SOTG). Nemůže žádná data vytvářet nebo modifikovat a nevyžaduje přihlášení.

Organizátor Účet pod touto rolí může vytvořit kdokoliv pouhou registrací. Po příhlášení lze přídávat nové turnaje a ty následně spravovat. Tím se rozumí tvorba rozpisu zápasů a seznamu účastníků. Dále může zadávat průběžné a výsledné skóre zápasů, skórující a asistující hráče a vidí na tabulku vzájemných hodnocení SOTG pro účely vyhlášení vítěze. Tato tabulka je pro ostatní role až do ukončení turnaje nezobrazitelná.

Klubový/oddílový účet Pro účely odevzdávání hodnocení SOTG a úprav v týmové soupisce je vytvořen pro každý oddíl jeden společný účet. Tento účet má možnost vytvořit pouze administrátor. Přihlašovací údaje pak získá zástupce klubu a je pouze na něm, kdo z jeho oddílu bude spravovat vytvořený učet. Součástí klubu může být více týmů, např. mužský a ženský tým nebo A-tým a B-tým.

Administrátor Má plnou kontrolu nad správou účtů a nad daty v databázi. Jeho možnosti vznikají sjednocením všech ostatních rolí.

2.3 Případy užití

Následující část popisuje nejběžnější případy užití. V návrhu rozhraní jde o jednu z nejdůležitejších součastí, protože určuje směr návrhu. Výsledné API by tak mělo pokrývat všechny níže uvedené případy. Seznam zachycuje jednotlivé požadavky a jejich popis:

Import

UC1: Import dat z databáze ČALD Doplňuje seznam oddílů a jejich hráčů do databáze Catchera. Manuálně lze stáhnout aktuální data z ČALD databáze a doplnit doposud aktuální data v databázi.

Uživatelé

UC2: Registruje nového uživatele Kdokoliv se může stát registrovaným uživatelem s rolí organizátor. Stačí se zaregistrovat s platným e-mailem. Klubové účty vytváří administrátor.

UC3: Autentizuje uživatele Určí skutečnou identitu uživatele. Jinými slovy jej přihlásí. Po úspěšném přihlášení je uživateli vrácen přístupový token pro vykonávaní dalších požadavků (více k sekci 5.4).

UC4: Odesílá zapomenuté heslo Při ztrátě hesla lze požádat o odeslání nového náhodně vygenerovaného na e-mail, který uživatel zadal při své registraci.

Oddíly

UC5: Vytvoří oddíl Uloží nový oddíl do databáze. Ukládá se u něj jméno, město a země původu.

UC6: Získá oddíly Vrátí informace o jednom nebo více oddílech.

UC7: Edituje oddíl Upraví informace o oddílu.

UC8: Smaže oddíl Smaže oddíl z databáze.

Hráči

UC9: Vytvoří hráče Vytvoří nového hráče, u kterého lze vytvořit vazba pouze k jednomu oddílu. Kromě jména a příjmení u něj lze uložit číslo dresu a jeho přezdívku.

UC10: Získá hráče Vrátí informace o jednom nebo více hráčích.

UC11: Edituje hráče Upraví informace o hráči.

UC12: Smaže hráče Smaže hráče z databáze.

Týmy

UC13: Vytvoří tým Vytvoří tým spadající pod konkrétní oddíl. Týmu se musí přiřadit stupeň a divize (např. A-tým ženy).

UC14: Získá týmy Vrátí informace o jednom nebo více týmech.

UC15: Edituje tým Upraví informace o týmu.

UC16: Smaže tým Smaže tým z databáze.

Turnaje

UC17: Vytvoří turnaj Uloží nový turnaj včetně účastníků turnaje, základních skupin a zápasů ve skupinách a play-off.

UC18: Edituje turnaj Upraví informace o turnaji nebo změní jeho stav.

UC19: Získá turnaje Vrátí informace o jednom nebo více turnajích.

UC20: Získá konečné pořadí Získá známé pořadí týmů na turnaji a pokud turnaj již skončil, zobrazí i hodnocení SOTG.

Týmové soupisky

UC21: Připsat hráče na soupisku Týmům na turnaji lze přiřadit hráče, tzn. vytvořit týmové soupisky. Hráčům ze soupisky pak lze připisovat body na turnaji. Každý hráč může hrát na turnaji pouze za jeden tým.

UC22: Získá soupisky Vrátí soupisku konkrétního týmu nebo seznam hráčů napříč celým turnajem, včetně jejich statistik a seřazení dle úspěšnosti.

UC23: Odebere hráče ze soupisky Jestliže hráč nedisponuje žádnými statistikami a nestihl ještě do žádného zápasu zasáhnout, je možné jej odstranit ze soupisky.

Skupiny a zápasy

UC24: Získá skupiny Vrátí informace o vybraných skupinách, tzn. seznam týmů včetně jejich vzájemného skóre a pořadí ve skupině.

UC25: Získá zápasy Vrátí stav, výsledné skóre, nebo hodnocení SOTG vybraných zápasů.

UC26: Upraví zápas Umožňuje zahájit nebo ukončit zápas. Dále lze editovat výsledné skóre nebo hřiště, na kterém se bude hrát.

UC27: Vytvoří nový bod v zápase Každý bod, který se odehraje, může evidovat skórujícího a nahrávajícího hráče a informaci, zda šlo o bod typu Callahan[5].

UC28: Smaže body v zápase V případě potřeby lze smazat dříve vložené body.

SOTG na turnaji

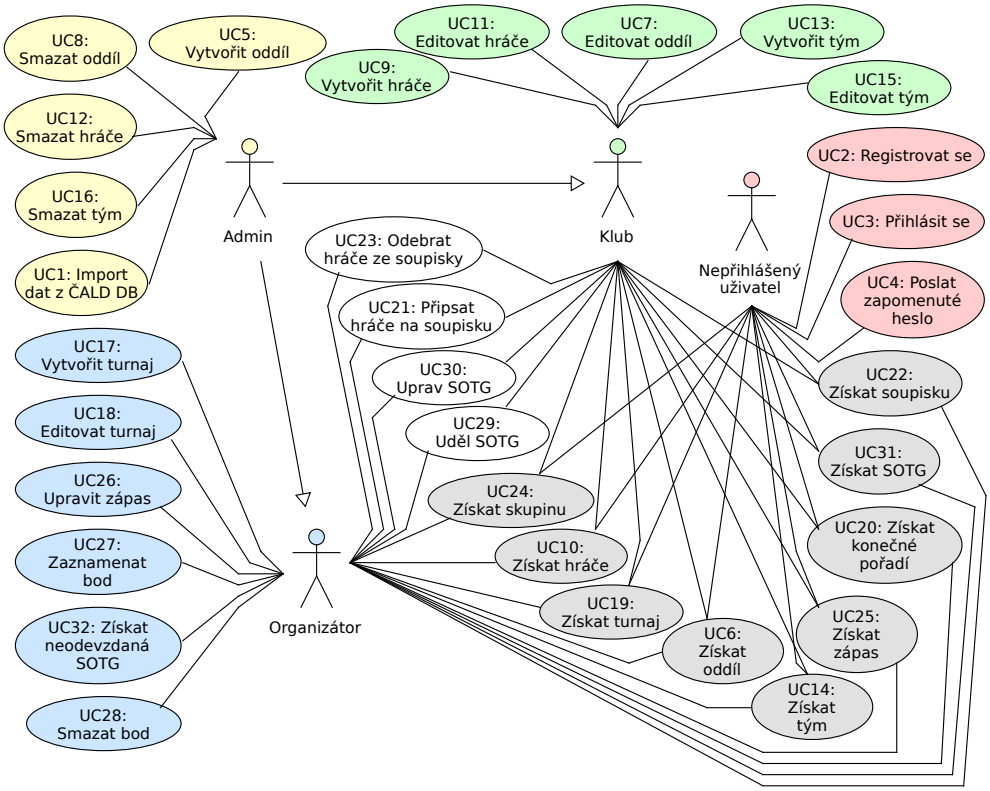
UC29: Ohodnotí soupeře v daném zápase hodnocením SOTG Po každém zápase může uživatel z klubového účtu ohodnotit soupeře hodnocením SOTG. Toto hodnocení se skládá z několika bodovaných kategorií. V celkovém součtu jde o hodnocení na stupnici od 0 do 20.

UC30: Upraví již jednou odevzdané hodnocení SOTG Před ukončením turnaje umožňuje změnit hodnocení.

UC31: Získá všechna odevzdaná hodnocení SOTG Do ukončení turnaje viditelné pouze pro organizátora.

[5] Body, při kterých skórujícímu hráči nahraje soupěř, jsou označovány jako Callahan. Když využijeme hokejovou terminologii, je to něco jako vlastní gól.

UC32: Získá všechna neodevzdaná hodnocení SOTG Před ukončením turnaje lze zkontrolovat, zda jsou všechna hodnocení již odevzdaná. V opačném případě zobrazí seznam týmů, které hodnocení neodevzdaly.



Obrázek 2.1: Use case diagram

Kapitola 3

Analýza

Jelikož původní záměr zněl rozšířit mobilní aplikaci Catcher, měli bychom se v této kapitole zabývat jejím průzkumem. Neopomeneme ani jiné podobné projekty. V druhé části se podíváme na možnosti našeho řešení.

3.1 Existující řešení

Mobilní aplikace Catcher

V roce 2013 [14] byla zveřejněna první verze této aplikace pro mobilní telefony s operačním systémem Android. Autory jsou dva aktivní hráči Jiří Voseček a Ondřej Burkert. Aplikace si od svého vzniku získala velkou popularitu ve frisbee komunitě a dnes je používána na většině českých turnajů. Umí zadávat výsledky zápasů včetně podrobného průběhu bodů, rozhodovat o umístění v základních skupinách, počítat statistiky hráčů, odevzdávat a zveřejnit vzájemná hodnocení SOTG týmů na turnaji. Dokáže omezeně importovat oficiální soupisky týmů na nadcházející turnaje spadající pod Českou asociaci létajícího disku.

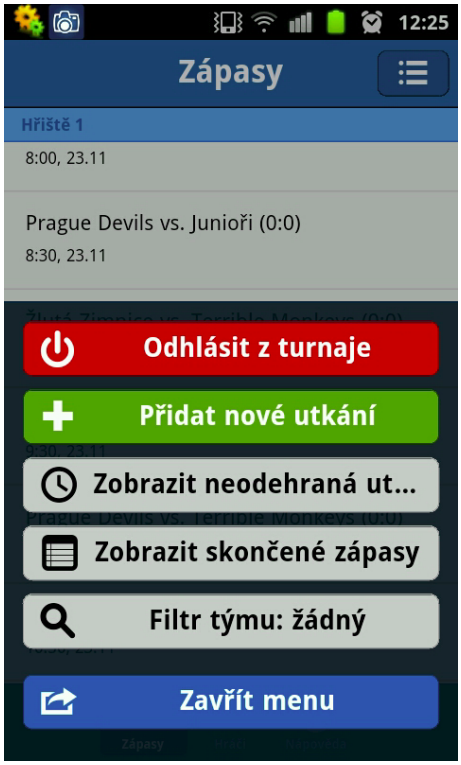
Kromě mobilní aplikace Catcher byl vytvořen backend v PHP [15], který ukládá a zpracovává data na serveru. Ten však není objektově orientovaný a všechna jeho eventuální rozšíření jsou velmi náročná.

Výhody

- Backend je využíván webovou aplikací (frisbee.cz) a mobilní aplikací na OS Android.

- Přes webový prohlížeč lze přistupovat k emulátoru aplikace.

- Systém je již mezi organizátory turnajů zaběhnutý a na Google Play[6] má již více než 1000 stáhnutí [16].
- Umí importovat soupisky na oficiální turnaje ČALD.



Obrázek 3.1: Mobilní aplikace Catcher [16]

Nevýhody

- Import soupisek z databáze ČALD je pouze poloautomatický. Administrátor musí před každým turnajem import spustit a zkontrolovat, zda se data uložila validně. Tomuto stavu nepřispívá ani fakt, že databáze ČALD, ze které se import provádí, je velmi špatně navržena.
- Nedisponuje žádným rozhraním, které by mohly používat další klientské aplikace.
- Automaticky nedoplňuje týmy v zápasech play-off do dalších kol (např. vítěze semifinále do finále apod.).

[6] Online distribuční služba, kde jsou k dispozici aplikace pro mobilní telefony a tablety s OS Android.

- Při vytváření turnaje neprobíhá žádná kontrola validity dat. Lze tak vytvořit spoustu chybných utkání, které se pak musí opravovat.

Ultimate Organizer

Aplikace zveřejněná v roce 2004 [17], napsaná v PHP a používaná na velkých akcích jako mistrovství světa nebo Evropy pro sledování výsledků a statistik. Poskytuje tvorbu a správu turnajů, týmů, hráčů, výsledků, rozpisů atd. Podporuje vícejazyčnost, tisk rozpisů v PDF, přístup pro mobilní telefony a spoustu dalších funkcí. Dokáže zobrazit podrobné detaily o průběhu zápasů.

Výhody

- Rozsáhlá aplikace poskytující nepřeberné množství možností.
- Aplikace má za sebou již více než 10 let fungování.
- Poskytuje přístup pro mobilní telefony s webovým prohlížečem.

Nevýhody

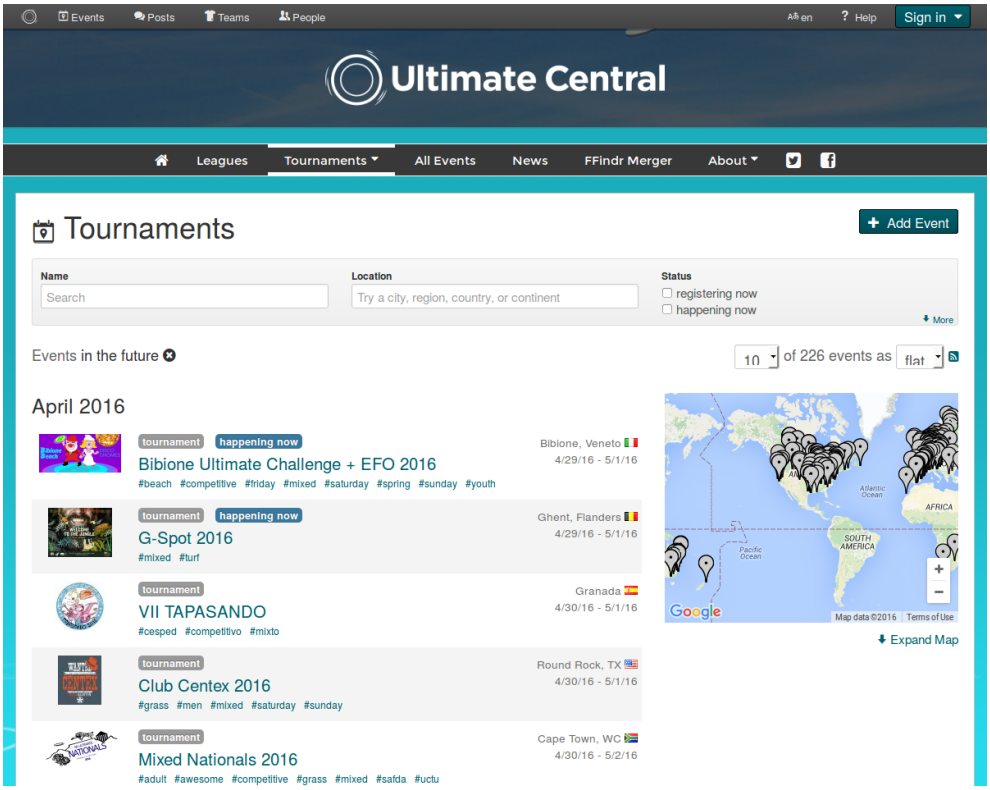
- Nefunguje jako webová aplikace. Uživatel si musí stáhnout zdrojové kódy a aplikaci si sám nasadit na svém počítači nebo serveru.
- Neposkytuje žádné rozumné rozhraní, tudíž pro ni nelze vytvořit žádného klienta.
- Nelze sledovat dlouhodobější statistiky napříč týmy a hráči. Všechny výsledky se týkají pouze jednoho konkrétního turnaje.

Ultimate Central

Webová aplikace [18], která slouží především ke správě soupisek na turnajích. Pořadatel turnaje vytvoří událost v kalendáři a zapíše všechny účastnící se týmy. Ty pak musí odevzdat svoje soupisky a potvrdit náležité dokumenty, které jsou na některých turnajích povinností (např. zřeknutí se možnosti žalovat organizátora aj.). Kromě toho dokáže ukládat výsledky zápasů a hodnocení SOTG.

Výhody

- Umožňuje vytvořit jednoduchou webovou stránku pro každý turnaj, na které lze shrnout důležité informace pro všechny účastníky.
- Aplikace je dostupná ve webovém prohlížeči.



Obrázek 3.2: Webová aplikace Ultimate Central

Nevýhody

- Neslouží ke statistikám. Do systému se zapisují pouze výsledky a hodnocení SOTG.
- Vyžaduje registraci a přihlašení každého jednotlivého hráče na turnaji.
- Stejně jako obě předchozí aplikace neposkytuje rozhraní, které by mohla využít například mobilní aplikace.

3.1.1 Výsledek

Průzkum existujících řešení naznačil, že náš projekt má smysl. Stále chybí aplikace, která by byla zároveň plně automatizovaná (před započetím turnaje se vytvoří rozpis a ten se již doplňuje na základě zadaných výsledků), měla rozhraní pro libovolné klienty a dokázala využít data z databáze ČALD.

3.2 Možnosti řešení

Protože už od počátku známe požadavek na možnost připojení klienta v podobě mobilní nebo webové aplikace, je nutné poskytnout žadatelům o přístup rozhraní, které dostatečně pokryje jejich požadavky. Z úvodní kapitoly víme, že webová služba se dělí na dva typy - SOAP a REST.

3.2.1 SOAP

Protokol SOAP (*Simple Object Access Protocol*) vystavuje aplikační logiku jako službu a je proveden v syntaxi jazyka XML [19]. Výhodou standardního značkovacího jazyka je jeho rozšíření a značná podpora ve většině programovacích jazycích. Syntaxe XML je pro člověka relativně čitelná, přestože bývá zdlouhavá, avšak počítač musí parsování dat věnovat více času i paměti.

Nevýhodou je nestálost prostředí. Ve chvíli, kdy dojde v SOAP API ke změně, klient může přestat pracovat, protože si nemusí umět poradit s odpovědí, přesněji s novou strukturou přijímaného XML souboru. Podobně je potřeba upravovat při změnách API i strukturu požadavku.

3.2.2 REST

Protokol REST (*Representational State Transfer*) je architektonický styl návrhu API popsaný Royem Fieldingem [20]. Je založený na vystavování a jednotném přístupu ke zdrojům[7], kterými jsou určitá data. Přistupuje se k nim pomocí HTTP metod GET, POST, DELETE a PUT.

RESTová služba používá pro přenos dat řadu formátů, nejstandardnějšími jsou XML a JSON [21]. JSON je datově méně náročnější a stejně jako XML je podporován v mnoha programovacích jazycích.

Zdroj

Architektonický styl REST je postaven na základním prvku – zdroji. Jde o logický objekt, který lze vyjádřit smysluplnou reprezentací, a s kterým lze manipulovat pomocí zpřístupněných metod. Každý zdroj má svůj unikátní identifikátor URI[8].

Požadavky na RESTové rozhraní

Aby se mohlo nějaké aplikační rozhraní považovat za RESTové (tzv. RESTful), musí splňovat několik základních předpokladů definovaných Fieldingem [20]. V této práci uvedeme ty nejdůležitější.

[7] resources
[8] Uniform Resource Identifier

Architektura klient-server Rozděluje zodpovědnost mezi různé části. Klientské aplikace se nemusí starat o správu dat a server o jejich prezentaci. Z toho vyplývá možnost vyvíjet obě kompomenty (klient a server) nezávisle na sobě.

Bezstavovost Aplikační stav je udržován na straně klienta. Všechny požadavky tak obsahují pouze informace nutné k jeho zpracování.

Jednotné rozhraní Pomocí unikátního identifikátoru lze získat reprezentaci nebo manipulovat s libovolným zdrojem. Klient musí být schopen s daným zdrojem manipulovat na základě jeho reprezentace. Zprávy by měly být dostatečně popisné.

3.2.3 Shrnutí

Z předcházejícího srovnání je patrné, že na flexibilní a v budoucnu a rozšiřitelné rozhraní se spíše hodí REST. Klienti nebudou tolik náchylní na úpravy API a navíc jim bude možné data poskytnout ve formátu JSON, který se pro jejich přenos hodí daleko více.

Kapitola 4

Návrh

Kdybychom navrhovali SOAP službu, vzali bychom všechny případy užití (use cases) a k nim vytvořili konkrétní metody. V REST je návrh trochu složitější. Musíme nejprve identifikovat všechny zdroje, se kterými se bude pracovat, a umožnit k nim přístup pomocí standardizovaných metod. V návrhu vycházíme ze tří ustálených modelů:

- doménový model
- datový model
- use case diagramy

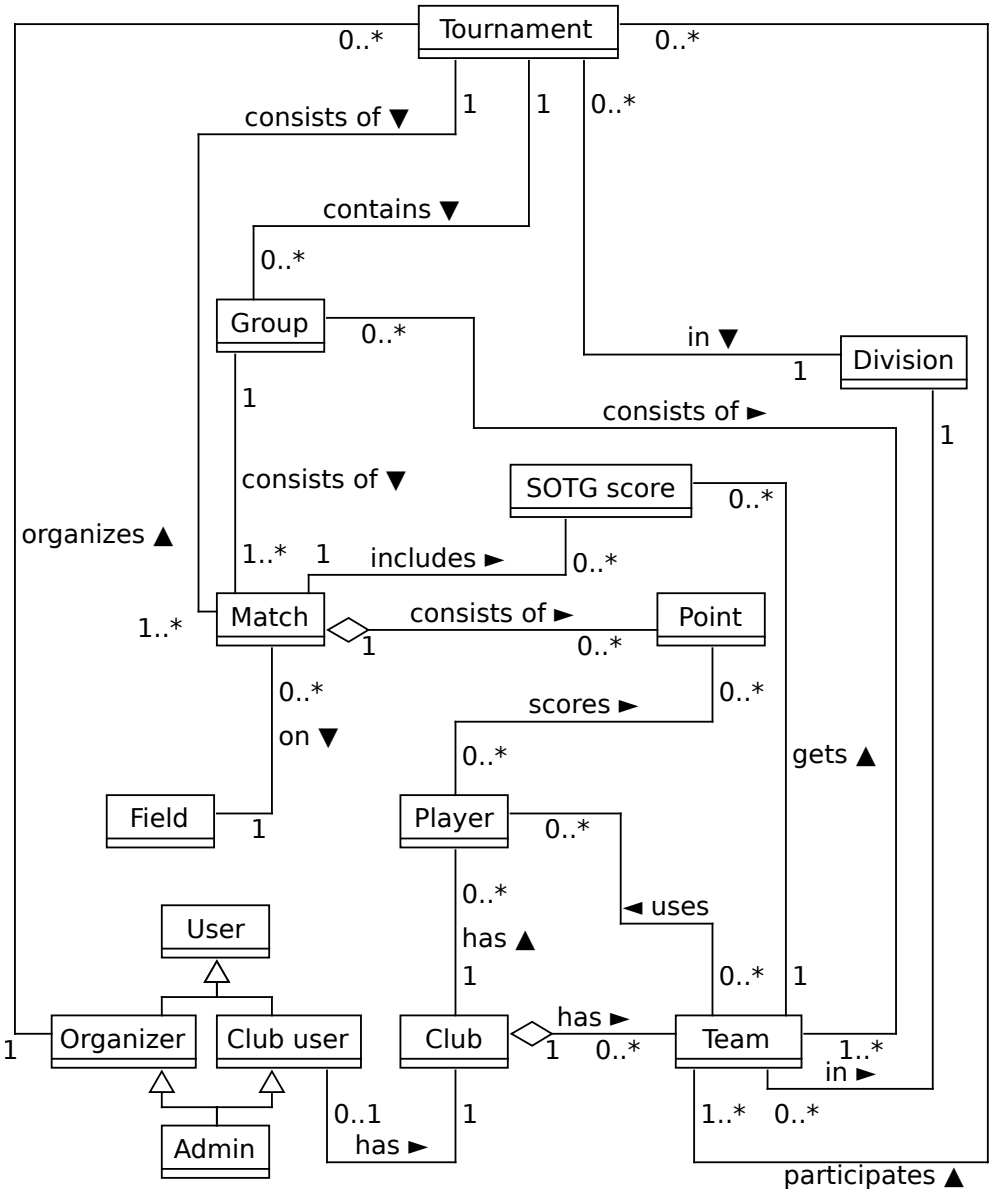
První dva modely nám slouží k identifikování jednotlivých zdrojů. Doménový model poskytuje klíčové entity a vztahy mezi nimi. Jde o vysokoúrovňový pohled na problematiku. Nezbytné detaily dořešíme pomocí datového modelu. Akce, které nejdou pokrýt pomocí standardních metod manipulujících se zdroji, nám mohou poskytnout use case diagramy. My si vystačíme s případy užití a jejich diagramem, který jsme zpracovali v sekci 2.3. Diagramy všech tří modelů jsou tedy v této práci zpracovány.

4.0.4 Doménový model

Tvorba doménového modelu (na obrázku 4.1) vychází ze zadání. Obsahuje následující entity, jež modelují základní představu o systému.

Uživatel (user) Uživatel je identifikován e-mailem a heslem nebo tokenem (více v sekci 5.4). Zároveň disponuje jednou ze tří rolí (`club`, `organizer` a `admin`).

Oddíl (club) Oddíl je logický prvek, který zastřešuje více týmů. Správu oddílu provádí klubový účet.



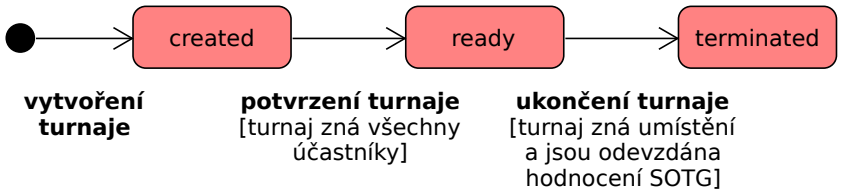
Obrázek 4.1: Doménový model

Tým (team) Tým je vázán na oddíl. Jeho identifikátory jsou divize a stupeň týmu (např. A-tým).

Divize (division) Kategorie, ve které se pořádají turnaje, nebo je složen tým. Například juniorská divize.

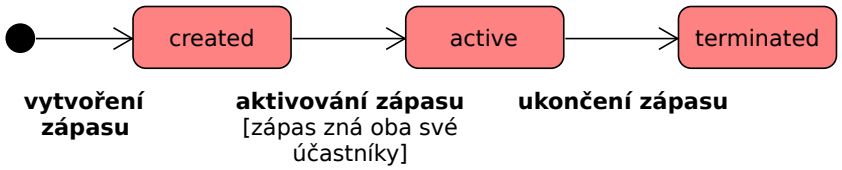
Turnaj (tournament) Turnaj může nabývat několika stavů (stavový diagram na obrázku 4.2). Po vytvoření turnaje lze až do potvrzení (pří-

znak `ready`) měnit týmy, které se turnaje zúčastní. Po něm mohou klubové účty odevzdávat soupisky za svoje týmy a na turnaji je možné aktivovat zápasy. Ve chvíli, kdy všechny týmy odevzdají svoje hodnocení SOTG a je známo finální pořadí týmů, lze turnaj ukončit (příznak `terminated`). Do té doby skryté výsledky hodnocení SOTG se stanou veřejnými. Turnaj je spravován administrátorem nebo uživatelem s rolí `organizer` (tzv. organizátor).



Obrázek 4.2: Stavový diagram turnaje

Zápas (match) Zápas má, stejně jako turnaj, tři stavy (obrázek 4.3). Aktivováním zápasu je umožněno zadat konečný výsledek nebo postupně vkládat jednotlivé body. Ukončením (příznak `terminated`) se konečný výsledek uloží a následně se promítne v dalším postupu turnajem (vítězný tým posune do dalšího zápasu, týmům se přidá výhra ve skupině apod.).



Obrázek 4.3: Stavový diagram zápasu

Bod (point) Zápas je složen z několika bodů. Každý bod je záznamem o skórování. Uchovává informaci, jaký hráč a komu nahrával. Zápas může být ukončen i bez jednotlivých bodů zadáním konečného skóre.

Skupina (group) Jde o několik týmů, které hrají zápasy pouze mezi sebou. Po odehrání všech utkání ve skupině se podle pravidel určí pořadí a na jeho základě se týmy posunou turnajem dál.

Hřiště (field) Zápas se musí někde hrát. K tomu slouží hřiště.

Hodnocení SOTG (SOTG score) Po každém utkání musí tým ohodnotit soupeře hodnocením Spirit of the Game. Zjednodušeně jde o číselné vyjádření na stupnici od 0 do 20.

4.0.5 Datový model

Pro ujasnění entit a vazeb mezi nimi byl vhodný doménový model. Jeho konkrétní implementací je model datový. Ten je dekomponován, tj. zbaven M:N vazeb, a oproti doménovému modelu obsahuje nové (relační) objekty včetně atributů. Kompletní datový model je dostupný v příloze B.

4.1 Identifikace zdrojů

Abychom mohli namapovat zdroje na konkrétní URI[9], musíme je nejprve jednoznačně určit. Mnoho programátorů zde postupuje tak, že vychází z datového modelu a použije prosté mapování *tabulka = zdroj*. Návrh je sice rychlý, ale nemusí být dostatečný. Navíc vytváří příliš těsnou vazbu mezi interní datovou reprezentací a logickou strukturou zdrojů. Zdroje v této práci vycházejí z doménového modelu.

4.1.1 Základní typy zdrojů

Podle [10] se zdroje dělí na tři základní typy:

Dokument Reprezentace jednoho konkrétního zdroje v určitém formátu.

Kolekce Množina zdrojů stejného typu. Jedná se o skupiny samostatných entit.

Kontroler Nezávislé metody aplikace. Akce, které nelze vyjádřit pomocí standardních CRUD[10] metod HTTP protokolu.

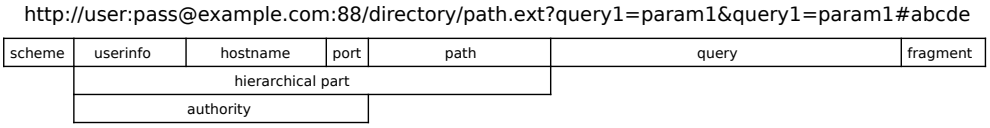
4.1.2 Návrh URI

Návrh se týká cesty v hierarchické části URI. Pro udržení kvality je doporučené se při návrhu držet několika zásad. Cesta by podle [10] měla:

- mít konzistentní a logickou strukturu
- mít dobře znázorněnou hierarchickou strukturu
- být nezávislá na zvolené technologii
- transparentně používat HTTP metody a kódy
- používat:
  - jednotná čísla pro označení konkrétních zdrojů

[9] Uniform Resource Identifier
[10] Create, Read, Update, Delete - vytváření, čtení, úpravy a mazání

  - množná čísla pro kolekce
  - slovesa pro kontrolery



Obrázek 4.4: Jednotlivé komponenty URI; zdroj: autor na základě [22]

S těmito zásadami byl vytvořen následující návrh na mapování operací, které lze provádět nad zdroji. Identifikátor značíme `{id}` nebo `{ide}` (druhý identifikátor je textový). V cestách je vynechán prefix `/api`, který odděluje aplikační a dokumentační část.

- `/login`
  - **popis:** Přihlašuje uživatele. Po přihlášení vrátí přístupový token (více v sekci 5.4).
  - **metoda:** POST
  - **pokrývá:** UC3
- `/forgotten-password`
  - **popis:** Odešle nové náhodně vygenerované heslo na e-mail uživatele.
  - **metoda:** POST
  - **pokrývá:** UC4
- `/users`
  - **popis:** Slouží k vytváření nových uživatelů.
  - **metoda:** POST
  - **pokrývá:** UC2
- `/user/{id}`
  - **popis:** Spravuje konkrétního uživatele. Lze mu nastavit nový e-mail nebo heslo.
  - **metoda:** PUT
  - **pokrývá:** UC21

- `/clubs`
  - **popis:** Kolekce oddílů.
  - **metoda:** GET, POST
  - **pokrývá:** UC5, UC6
- `/club/{id}`
  - **popis:** Operace nad konkrétním oddílem. Možnost uložit město a zemi původu.
  - **metoda:** GET, PUT, DELETE
  - **pokrývá:** UC6, UC7, UC8
- `/club/{id}/players`
  - **popis:** Kolekce všech klubových hráčů.
  - **metoda:** GET
  - **pokrývá:** UC10
- `/club/{id}/teams`
  - **popis:** Kolekce všech týmů, které zastřešuje konkrétní oddíl.
  - **metoda:** GET
  - **pokrývá:** UC14
- `/players`
  - **popis:** Kolekce hráčů.
  - **metoda:** GET, POST
  - **pokrývá:** UC9, UC10
- `/player/{id}`
  - **popis:** Operace nad konkrétním hráčem.
  - **metoda:** GET, PUT, DELETE
  - **pokrývá:** UC10, UC11, UC12

- `/teams/`
  - **popis:** Kolekce týmů.
  - **metoda:** GET, POST
  - **pokrývá:** UC13, UC14
- `/team/{id}`
  - **popis:** Operace nad konkrétním týmem.
  - **metoda:** GET, PUT, DELETE
  - **pokrývá:** UC14, UC15, UC16
- `/divisions`
  - **popis:** Kolekce divizí.
  - **metoda:** GET
- `/tournaments`
  - **popis:** Kolekce turnajů, kterou lze filtrovat pomocí pěti různých parametrů.
  - **metoda:** GET, POST
  - **pokrývá:** UC17, UC19
- `/tournament/{id}`
  - **popis:** Správa turnaje. Možnost turnaji upravit příznaky `ready` a `terminated`.
  - **metoda:** GET, PUT
  - **pokrývá:** UC18, UC19
- `/tournament/{id}/standings`
  - **popis:** Zobrazuje výsledné pořadí týmů na turnaji, včetně kategorie SOTG.
  - **metoda:** GET
  - **pokrývá:** UC20
- `/tournament/{id}/players`
  - **popis:** Slouží pro správu soupisek na turnaji.
  - **metoda:** GET, POST, DELETE
  - **pokrývá:** UC21, UC22, UC23

- `/tournament/{id}/teams`
  - **popis:** Slouží pro správu týmů na turnaji.
  - **metoda:** GET, PUT
- `/tournament/{id}/matches`
  - **popis:** Kolekce zápasů, kterou lze filtrovat pomocí šesti různých parametrů.
  - **metoda:** GET
  - **pokrývá:** UC25
- `/tournament/{id}/groups`
  - **popis:** Kolekce skupin.
  - **metoda:** GET
  - **pokrývá:** UC24
- `/tournament/{id}/group/{ide}`
  - **popis:** Konkrétní skupina.
  - **metoda:** GET
  - **pokrývá:** UC24
- `/tournament/{id}/spirits`
  - **popis:** Kolekce všech hodnocení SOTG. Možnost filtrovat hodnocení pro jeden konkrétní tým.
  - **metoda:** GET
  - **pokrývá:** UC31
- `/tournament/{id}/missing-spirits`
  - **popis:** Seznam týmů, které hodnocení SOTG neodevzdaly.
  - **metoda:** GET
  - **pokrývá:** UC32
- `/match/{id}`
  - **popis:** Správa utkání.
  - **metoda:** GET, PUT,
  - **pokrývá:** UC25, UC26

- `/match/{id}/points`
  - **popis:** Zobrazuje detailní popis zápasu. Konkrétně jeho odehrané body seřazené chronologicky. Dále poskytuje ukládání nových bodů a úpravu stávajících (například změna hráče, který skóroval).
  - **metoda:** GET, POST, PUT, DELETE
  - **pokrývá:** UC25
- `/match/{id}/spirits`
  - **popis:** Vrátí detail jednoho zápasu včetně hodnocení SOTG obou týmů. Kromě toho umožňuje odevzdat hodnocení SOTG.
  - **metoda:** GET, POST, PUT
  - **pokrývá:** UC25, UC29, UC30

Kapitola 5

Implementace

5.1 Výběr technologií

V době, kdy existuje obrovské množství programovacích jazyků, jejich ekosystémů a frameworků je důležité se umět „nespálit“. Vybrat špatné technologie totiž může znamenat prodloužení vývoje, zvýšení celkových nákladů nebo dokonce ukončení projektu. Proto je nutné tuto část dostatečně analyzovat a svědomitě zvážit, aby nedošlo k omylu, který by zabraňoval úspěšnému dokončení. Pro adekvátní výběr tak bylo potřeba specifikovat příslušná kritéria, která by jasně porovnala všechny zvažované technologie.

Podpora webových služeb

Při tvorbě webové aplikace nemůže být ani řeč o technologii, která by pro použití na webu nebyla vhodná. Technologie musí umět pracovat s mapováním URI a HTTP metod na zdroje a pracovat s běžnými formáty dat v internetové komunikaci (např. JSON).

Dokumentace a uživatelská podpora

Pro pochopení technologie a její správné používání je mnohdy potřeba znát detaily, které se uživatel dozví z dokumentace. Ta dokáže výrazně zkrátit učící křivku a programátorovi tak šetří čas. Pokud dokumentace nestačí, může její nedostatky zachránit dostatečně široká uživatelská základna. Ta zároveň snižuje riziko zestárnutí zvolené technologie.

Znalost technologie

Pokud již programátor některou z technologií zná, může mu tato znalost ušetřit mnoho času při vývoji. Nemusí totiž novou technologii nijak složitě zavádět nebo se učit její použití. Vhodným výběrem se také může vyvarovat situacím, kdy až v průběhu implementace narazí na neřešitelné problémy. Zavrhnout

neznámé technologie ale může znamenat přijít o možnost používat nástroje, které jsou problémům šité na míru.

Výkon

I když služba nepočítá s vetším provozem čítajícím stovky požadavků za vteřinu, není důvod používat technologie, které jsou pomalé nebo spotřebovávají příliš mnoho prostředků.

5.2 Zvolené technologie

5.2.1 Programovací jazyk Python

Python [23] je víceúčelový skriptovací jazyk navržený v roce 1991 [24] Guido van Rossumem. Je vyvíjen jako open source projekt a je bezplatně dostupný pro většinu platforem (Unix, Windows, Mac OS). V distribucích Linuxu je většinou součástí základní instalace. Díky velkému množství knihovních modulů z různých oblastí má velice široké uplatnění. Běžně jej používají ve firmách jako Google, Dropbox, YouTube, Red Hat, Cisco, Facebook nebo Microsoft [25].

Protože jde o dynamický interpretovaný jazyk, jeho zdrojový kód není nutné předem překládat překladačem do strojového kódu. Je to zároveň hybridní jazyk, protože umožňuje používat různá programovací paradigmata, včetně objektově orientovaného, imperativního, procedurálního nebo v omezené míře i funkcionálního [26]. I když je Python mnohokrát označován za skriptovací jazyk, jeho návrh umožňuje psaní rozsáhlých a plnohodnotných aplikací včetně GUI. Podporuje také dynamickou kontrolu datových typů.

Je to jazyk, který se velmi snadno učí a bývá považován za jeden z nejvhodnějších programovacích jazyků pro začátečníky. Pomáhá tomu jeho jednoduchá syntaxe a čistota kódu. Na rozdíl od jiných jazyků bývá jeho zdrojový kód mnohokrát krátký a dobře čitelný. Je tak zároveň vhodný pro výuku i využití v praxi. Podle PYPL indexu[11] je Python celosvětově nejrychleji roustoucí programovací jazyk co do oblíbenosti za posledních pět let [27] a v dubnovém žebříčku roku 2016 drží druhé místo mezi jazyky Java [28] a PHP.

Po stránce výkonu je na tom Python relativně dobře, protože mnoho na výkon náročných knihoven je implementováno v jazyce C [29]. V porovnání s ostatními interpretovanými jazyky je na tom samotný Python taky dobře. Fakt, se kterým ale musíme počítat, je ten, že dynamicky interpretované jazyky jsou obecně pomalejší než kompilované.

[11] PopularitY of Programming Language Index je vytvořen analyzováním toho, jak často jsou tutoriály jednotlivých programovacích jazyků hledány na Googlu.

5.2.2 Webový framework Falcon

Falcon [30] je minimalistický webový framework pro vývoj aplikačních backendů a jejich API s otevřeným zdrojovým kódem, populární pro svoji neuvěřitelnou rychlost. Falcon podporuje architektonický styl REST, což znamená, že mapuje zdroje a jejich metody do HTTP protokolu.

framework	req/sec	μs/req	výkon
Falcon (0.3.0)	21,858	46	8x
Bottle (0.12.8)	12,583	79	4x
Werkzeug (0.10.4)	4,708	212	2x
Pecan (0.8.3)	3,442	291	1x
Flask (0.10.1)	2,837	352	1x

Tabulka 5.1: Výkonnostní test několika podobných webových frameworků pro CPython 2.7.9 [31]. Jde o implementaci jazyka Python, kterou používá Catcher.

Falcon disponuje minimálním množstvím závislostí na jiných knihovnách a díky jeho flexibilitě je ho možné používat ve většině verzí Pythonu[12]. Kromě toho umí pracovat s WSGI[13]. Nevýhodou Falconu je menší uživatelská základna na rozdíl od konkurenčního Flasku, jenž patří mezi nejpopulárnější webové frameworky pro Python.

Na následujících řádcích je ukázka použití Falconu v mé práci. Jde o metodu GET pro kolekci oddílů.

```
class Clubs(Collection):
    def on_get(self, req, resp):
        '''
        metoda getClubs() vrátí všechny kluby
        pomocí předpřipravených dotazů a data
        uloží do odpovědi
        '''
        clubs = Queries.getClubs()
        collection = {
            'count' : len(clubs),
            'items' : clubs
        }
        req.context['result'] = collection
```

[12] Vývoj v Pythonu poznamenalo v roce 2008 vydání nové verze Python 3.0, která je částečně zpětně nekompatibilní.

[13] WSGI je komunikační protokol mezi aplikací napsanou v Pythonu a webovým serverem.

5.2.3 Databáze MySQL

MySQL [32] je relační databáze, se kterou probíhá komunikace pomocí jazyka SQL [33], jak už její název napovídá. Díky svému výkonu, snadné použitelnosti (lze ji nainstalovat na Linux, Windows, OS X a další) a faktu, že jde o volně šiřitelný software (je k dispozici i pod komerční licencí), je MySQL velmi populární. Je součástí velmi oblíbené kombinace základního softwaru na serverech známou pod zkratkou LAMP[14].

Pro správu databáze se používá zpravidla příkazový řádek nebo lze separátně stáhnout a nainstalovat nástroj zvaný MySQL Workbench [34]. Ten byl použit i v této práci při návrhu databázového modelu.

Od svého vzniku v roce 1995 [35] se od MySQL odpojilo několik alternativních větví, tzv. forků. Mezi nejznámější případy patří MariaDB [36] a Percona [37].

5.2.4 Peewee ORM

Peewee [38] je implementace ORM[15] pro jazyk Python. Obsahuje podporu pro databáze SQLite [39], PostgreSQL [40] a MySQL. Je otevřeným softwarem a jeho zdrojový kód je dostupný na GitHubu [41]. Použití Peewee zde:

```
import peewee

class Tournament(peewee.Model):
    '''reprezentace tabulky tournament v DB'''
    id = peewee.PrimaryKeyField()
    name = peewee.CharField(db_column='name')

'''získání konkrétního záznamu s id = 1'''
tournament = Tournament.get(id=1)
'''tisk jména turnaje'''
print tournament.name
```

Co je to ORM

ORM je programovací technika, která zajišťuje, že data z relační databáze se konvertují do objektů v OOP[16]. Programátor tak pracuje s perzistentními objekty místo toho, aby psal SQL dotazy. Pokročilejší implementace ORM dokonce dokážou využít objektovou dědičnost, což relační databáze nepodporují.

Technika je často kritizována, protože programátory odnaučuje psát SQL dotazy a jde většinou o zbytečnou režii navíc. Mnoho ORM nástrojů totiž slo-

[14] Linux, Apache, MySQL, PHP
[15] Object-relational mapping
[16] Object-oriented programming

žitější dotaz přeloží do jazyka SQL tak neúčinně, že může být mnohonásobně pomalejší, než kdyby jej programátor napsal v SQL sám. V závěru této práce jsem použití ORM zhodnotil.

5.2.5 Verzovací systém Git

Git [42] je nástroj vytvořený Linusem Torvaldsem[17]. Slouží k distribuované správě verzí libovolných digitálních informací, například zdrojových kódů. Výraznou výhodou je možnost spolupráce velkého množství programátorů na jednom softwarovém projektu. Každý programátor má adresář s projektem na svém lokálním disku a všechny změny sdílí s centrálním repozitářem, ke kterému mají přístup i všichni ostatní programátoři. Ti si pak mohou stáhnout všechny změny, ke kterým v projektu došlo.

Ruku v ruce jde s popularitou Gitu nahoru i služba GitHub. Ta nabízí bezplatný i komerční hosting pro repozitáře softwarových projektů. Zdarma, a díky tomu populární v dané oblasti, je GitHub pro open source projekty. Funguje od roku 2008 a hostuje už více jak 10 miliónů repozitářů [43]. Poskytuje mnoho dalších funkcionalit, jako například možnost diskutovat nad kódem nebo zasílat notifikace o změnách.

5.2.6 Webový server Nginx

Nginx [44] je webový server a reverzní proxy[18], který pracuje s běžnými protokoly. První oficiální verze se objevila v roce 2004, kterou vyvinul ruský softwarový inženýr Igor Sysoev [45]. Zaměřuje se na vysoký výkon a nízké nároky na pamět. Je používán velkými firmami díky propracované možnosti rozložení zátěže. Z českých firem je to například Seznam.cz [46].

Dnes je již s 25 % druhým nejpoužívanějším webovým serverem v prvním miliónu nejzatíženějších webů na světě. Napříč celým webem je pak na třetím místě s přibližně 16 % [47]. I když je Apache HTTP Server [48] stále výraznou jedničkou, jejich rozdíl se neustále zmenšuje. Nginx je open source.

5.3 Adresářová struktura

Adresářová struktura popsaná v této sekci odpovídá struktuře, kterou lze najít i na GitHubu. Projekt se dělí na dvě části, v jedné se nachází aplikační logika a v druhé je jednoduchá webová stránka s dokumentací.

Celou aplikační logiku najdeme v adresáři `/catcher`, kde se nachází REST-ové zdroje a jejich mapování na URI, mapování relační databáze na objekty nebo zdrojové kódy zajišťující autentizaci a autorizaci uživatelů. Dokumentace se nachází v adresáři `/html`.

[17] tvůrce Linuxového jádra

[18] Reverzní proxy se používá pro zvýšení výkonu webového serveru. Rozděluje vstupující provoz na více serverů, například vyvažuje zátěž serverů zapojených v clusteru.

Konfigurační soubory jsou uloženy v adresáři `/catcher/config`. Nachází se zde přístupové údaje k databázím (produkční a testovací) nebo e-mailovému účtu pro zasílání nového hesla. Konfigurační soubory nejsou z důvodu bezpečnosti uloženy v repozitáři na Gitu.

```
/ ..... kořenová složka
├── /bin ..... spustitelné skripty
├── /catcher ..... aplikační logika
│   ├── /api ..... vrstva spravující API
│   ├── /config ..... konfigurační soubory
│   ├── /logger ..... logování aplikace
│   ├── /models ..... mapování relační databáze na OOP
│   ├── /resources ..... REST zdroje
│   ├── /test ..... testy
│   └── /restapi.py ..... spouštěcí skript
├── /html ..... adresář dostupný z webu (dokumentace)
├── /nginx ..... konfigurační soubory webového serveru
├── /sql ..... SQL skripty
├── /LICENSE ..... licence softwaru
├── /README.md ..... výčet závislostí a ostatní informace
└── /VERSION ..... aktuální verze
```

Obrázek 5.1: Struktura webové služby včetně adresáře pro webovou dokumentaci.

5.4 Bezpečnost

Zabezpečit webovou aplikaci je dnes opakovaně diskutovaným problémem. Mnoho systémů již postihlo napadení nebo únik důležitých dat. Navíc neexistuje stav, kdy je možné o své práci prohlásit, že je naprosto bezpečná. Případní útočníci jsou totiž stále vynalézavější a hledají každou chybu v systému. Aplikaci je tedy nutné obohatit o co nejvíce ochranných prvků a držet se zásad, které jejich možnosti minimalizují.

5.4.1 Zabezpečená komunikace

Základem všech aplikací, ve které se můžou nacházet citlivá nebo jinak důležitá data, je použití protokolu HTTPS[19], který zabezpečuje spojení šifrováním pomocí SSL[20] nebo TLS[21]. Důsledkem je znemožnění odposlechu, např. přihla-

[19] Hypertext Transfer Protocol Secure
[20] Secure Sockets Layer
[21] Transport Layer Security

šovacích údajů. Moje práce HTTPS protokol nevyužívá, ale v případě nasazení do ostrého provozu bych jej použil.

5.4.2 Autentizace

Autentizace je procesem, kdy se ověří identita uživatele. Protože jedním ze základních pilířů RESTové služby je jeho bezstavovost, je potřeba s každým požadavkem posílat i data, která slouží k autentizaci. Zdroje a jejich metody, které jsou ze své podstaty veřejné a nevyžadují autentizaci, jsou tohoto procesu zbaveny.

Mým řešením je posílání přístupového tokenu (náhodně generovaný hash o 32 znacích) v hlavičce požadavku `Authorization`. Uživatel se před dotazováním služby přihlásí pomocí svého e-mailu a hesla. V případě úspěšného přihlášení je mu vrácen stavový kód 200 a přístupový token, který může používat po dobu jeho platnosti.

5.4.3 Autorizace

Autorizace je proces, který na svém konci schválí uživatele k provedení nějaké akce. Každý uživatel má přiřazenou roli, která mu povoluje nebo zakazuje konkrétní operace. V případě, že se jedná o organizátora (role `organizer`), je ověřováno, zda pracuje pouze se svými vytvořenými turnaji. Každý turnaj si totiž v databázi uchovává informaci o tom, kým byl vytvořen. Klubové účty (role `club`) mají právo volat požadavky týkající se týmů, které zastřešuje jejich klub. Uživatel s rolí `admin` disponuje přístupem ke všem operacím.

5.5 Reprezentace dat

Pro komunikaci ze strany klienta i serveru používám jediný formát – JSON. Jde o zápis dat, který je nezávislý na platformě, a je přímo určený pro přenos dat. Vstupní data, kterými může být libovolná datová struktura, transformuje do řetězce. Data v JSON formátu mohou být hierarchicky řazena nebo organizována v polích. Pro lepší pochopení je na následujících řádcích uveden praktický příklad. Jde o návratová data z GET požadavku na adrese `/api/divisons`. Obsahem atributu `items` je pole s jednotlivými divizemi.

```
{
 "count":5,
 "items":[
   {
     "division":"junior",
     "id":5
   },
   {
     "division":"masters",
     "id":4
   },
```

```
    {
      "division":"mixed",
      "id":3
    },
    {
      "division":"open",
      "id":1
    },
    {
      "division":"women",
      "id":2
    }
  ]
}
```

V Pythonu se formátu JSON velmi podobá datová struktura `dict()` [49]. Vzájemnou konverzi poskytuje několik knihoven, některé jsou dokonce specializovány pouze na přeměnu v jednom směru. Já používám ve většině případech standardní knihovnu `json` [50] nebo ručně doinstalovanou `ujson` [51].

5.6 Vytvoření turnaje

Vytvoření turnaje je jednou z nejsložitějších operací systému. V metodě POST jsou vloženy:

- základní informace o turnaji
- hřiště, na kterých se bude hrát
- seznam utkání s detaily (hřiště, čas, domácí a hostující tým) a informací, kam postupuje vítěz i poražený nebo na jakých místech v celkovém pořadí skončí
- skupiny s postupovým klíčem, podobně jako v utkáních

Než se data uloží, je nejprve nutné zkontrolovat validnost dat. Musí se zrevidovat, zda:

- nechybí data
- nedochází k časové i místní kolizi zápasů
- na turnaji nehrají neexistující týmy
- struktura turnaje je hratelná, nedochází k nelogickým situacím (například do finále postoupí tři týmy)

Data navíc musí být ukládána atomicky, tzn. že při odhalení chyby v průběhu ukládání dat je celá transakce vrácena zpět a data nejsou uložena. Atomičnost operací je využívána v mnoha dalších operacích, ne jenom při vytváření turnaje.

5.7 Průběh turnaje

V okamžiku, kdy má turnaj příznak `ready`, je možné odehrávat zápasy. V jejich průběhu lze vkládat body jednotlivě nebo rovnou celé výsledky. Ať už si uživatel vybere jakkýkoliv způsob, po každé změně jsou přepočítávány statistiky hráčů, aby byly co nejaktuálnější. Po skončení zápasu se vyhodnotí vítěz a poražený a následně jsou automaticky doplněni do dalšího kola na základě postupového klíče, jenž organizátor vložil již při tvorbě turnaje.

Pokud vyřazovacím zápasům předchází skupina, po ukončení každého zápasu se kontroluje, zda je již kompletně odehraná. Pokud ano, týmy jsou podle pořadí ve skupině, které je opět přepočítáváno po každém zápase, posunuty v rámci turnaje do dalšího kola.

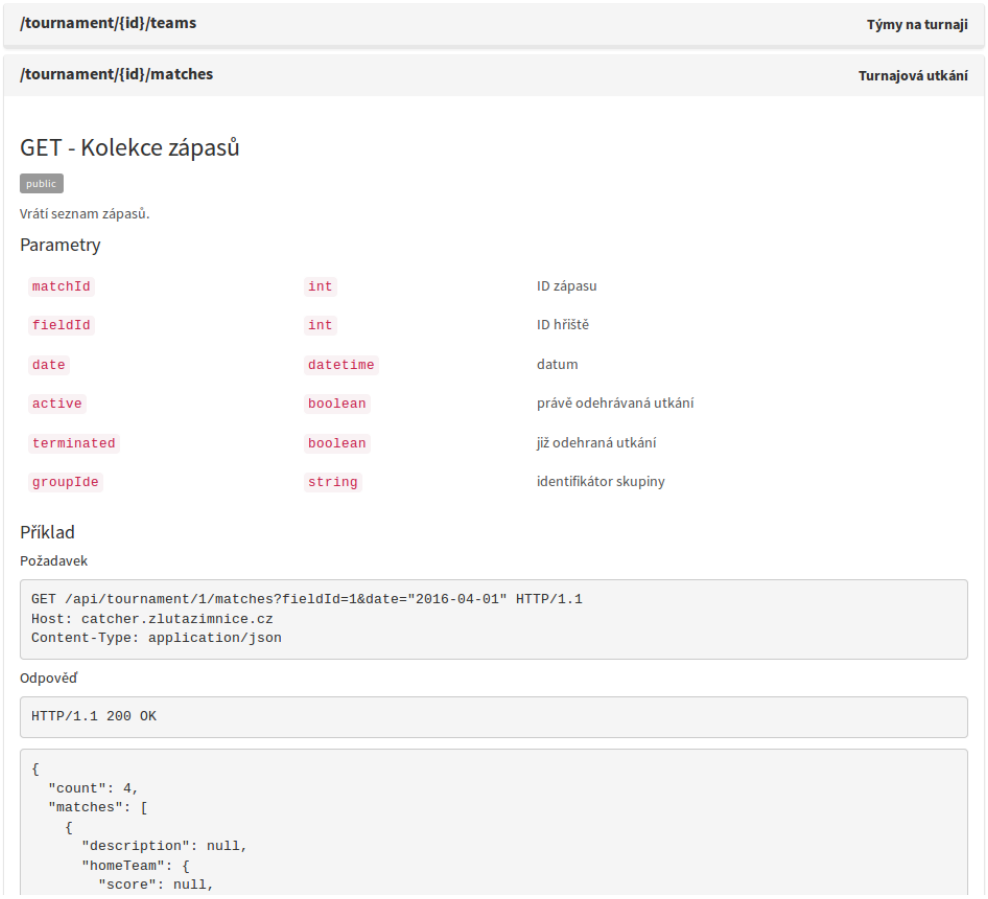
5.8 Hodnocení SOTG

Hodnocení se odevzdává po každém ukončeném utkání. Všechny hodnoty jsou uloženy do databáze a navíc se přepočítá tabulka s průměrnými hodnoceními, ze kterých pak vychází celkové pořadí v této kategorii. Každá změna hodnocení znamená opětovné přepočítání.

5.9 Uživatelská dokumentace

I když sepsání uživatelské dokumentace není požadavkem této práce, je vhodné celé rozhraní popsat. Programátoři, kteří budou v budoucnu pracovat na klientských aplikacích, mohou z tohoto zdroje čerpat všechny potřebné informace k tomu, aby API kompletně využili.

Dokumentace, dostupná na adrese `http://catcher.zlutazimnice.cz`, poskytuje seznam všech zdrojů a metod, které lze volat. Popisuje způsob autentizace, formát komunikace, parametry dotazů nebo strukturu jejich odpovědí. Dokumentace byla využívána Jaroslavem Veselým k vývoji frontendu. Jak dokumentace vypadá můžete vidět na obrázku 5.2.



/tournament/{id}/teams — Týmy na turnaji

/tournament/{id}/matches — Turnajová utkání

GET - Kolekce zápasů

public

Vrátí seznam zápasů.

Parametry

matchId	int	ID zápasu
fieldId	int	ID hřiště
date	datetime	datum
active	boolean	právě odehrávaná utkání
terminated	boolean	již odehraná utkání
groupIde	string	identifikátor skupiny

Příklad

Požadavek

```
GET /api/tournament/1/matches?fieldId=1&date="2016-04-01" HTTP/1.1
Host: catcher.zlutazimnice.cz
Content-Type: application/json
```

Odpověď

```
HTTP/1.1 200 OK
```

```
{
  "count": 4,
  "matches": [
    {
      "description": null,
      "homeTeam": {
        "score": null,
```

Obrázek 5.2: Ukázka z dokumentace

Kapitola 6

Testování

Jednou z nejdůležitejších součástí vývoje je testování. Pomáhá zkoumat funkčnost softwaru a v případě bezchybného a úplného testování zajišťuje, že výsledná aplikace neobsahuje chyby a má požadované vlastnosti. Testování se často provádí ve více iteracích, například pravidelně před vydáním nové verze softwaru. Součástí testování je pak především reportování nalezených chyb.

Testy se dělí na tzv. black-box a white-box testy. Při testování černé skříňky nemáme informace o vnitřní podobě a porovnáváme pouze správnost výstupních dat po zadání vstupních. Do testovaného subjektu se nelze podívat, vidíme jen to, jak se chová navenek. Cílem tohoto typu testování je analyzovat software z pohledu uživatele. Na opačné straně se při znalosti implementace používá testování bílé skříňky. Sice tato metoda zastiňuje pohled uživatele, ale tester je schopen lépe určit, kde hledat případné chyby. Při částečné znalosti softwaru se ještě používá pojem testování šedé skříňky (gray-box). Neznáme například přesnou podobu zdrojového kódu, ale jen algoritmus použitý v aplikaci.

Testy se také dělí na automatizované a manuální. Méně nákladnou a rychlejší cestou je zpravidla automatizované testování, kdy se automaticky spouští velké množství testů, často i s velkým množstvím vstupních dat. Tyto testy se používají v místech, kde dochází k opakování stejného nebo velmi podobného scénáře. Ne vždy se ale používá pouze automatizované testování. Když test vyžaduje lidský úsudek, rozdílné přístupy nebo jej není nutné pravidelně opakovat, používá se manuální testování či kombinace obojího.

6.1 Testování Catchera

Testování softwaru je stále velmi podceňovaný obor a často je tato fáze vývoje odsunuta na druhou kolej nebo zcela vynechána. Kromě toho, že psaní testů se ukázalo jako nejrychlejší způsob odhalování chyb, bylo pak ještě více překva-

pující, že při použití techniky TDD[22] se znatelně zefektivnil vývoj. Už při psaní zdrojového kódu jsem tak mohl jednoduše spouštět již vytvořené testy a znát jejich stav. Díky specifickému popisu získání a manipulace se zdroji v architektuře REST jsem většinu testů podřídil tomuto rozdělení a prováděl jsem tzv. testování funkcionalit – z pohledu uživatele, ověřování úzce zaměřených scénářů.

V Pythonu se nejrozšířenější framework pro automatizované testování jmenuje Unittest [52]. Já jsem však využil testovací nástroj, jenž přímo poskytuje webový framework Falcon. Je potřeba zmínit, že tento nástroj většinu metod dědí od základního Unittestu a jeho možnosti pouze rozšiřuje pro lepší a rychlejší testování REST API.

Automatizovanými testy jsou v Catcheru pokryty všechny zdroje s jejich metodami (celkem jich je 66). Typicky jsou testy po několika sdružovány do jedné třídy, ve které je možnost vytvořit specifické testovací prostředí. V knihovně `unittest` jsou pro tyto účely často používány metody `setUp()` a `tearDown()`, které jsou volány před, respektive po každém testu. Důsledek tohoto chování je zajištění velmi důležitého aspektu v testování – nezávislosti testů. Bez něj by mohly testy pracovat nad stejnou množinou dat a navzájem si „kazit“ výsledky. Nejčastější podoba testů pro Catchera je vidět v následující velmi zjednodušené ukázce kódu:

```
class TestClubs(falcon.testing.TestBase):

    def setUp(self):
      '''Spustí se před každým testem,
      zpravidla naplní testovací databázi daty.'''

    def tearDown(self):
      '''Spustí se po každém testu,
      zpravidla vyčistí testovací databázi.'''

    def testGet(self):
      '''Testuje požadavek GET — získá kolekci všech oddílů.'''
      response = self.simulate_request('GET', '/api/clubs')
      self.assertEqual(self.srmock.status, HTTP_200)

    def testPost(self):
      '''Testuje požadavek POST — vytvoří nový oddíl.'''
      response = self.simulate_request('POST', '/api/clubs', body)
      self.assertEqual(self.srmock.status, HTTP_201)
```

Kromě porovnávání návratových HTTP kódů se v mých testech zkoumá podoba a obsah návratových dat, správné vyhazování výjimek nebo stav testovací databáze před a po volání metod.

[22] Test-driven development - technika, kdy se testy píší ještě před samotným vývojem.

Kapitola 7

Nasazení

K testování a provozu webové aplikace jsem získal virtualizovaný server, ke kterému přistupuji pomocí SSH [23].

Virtuální privátní server

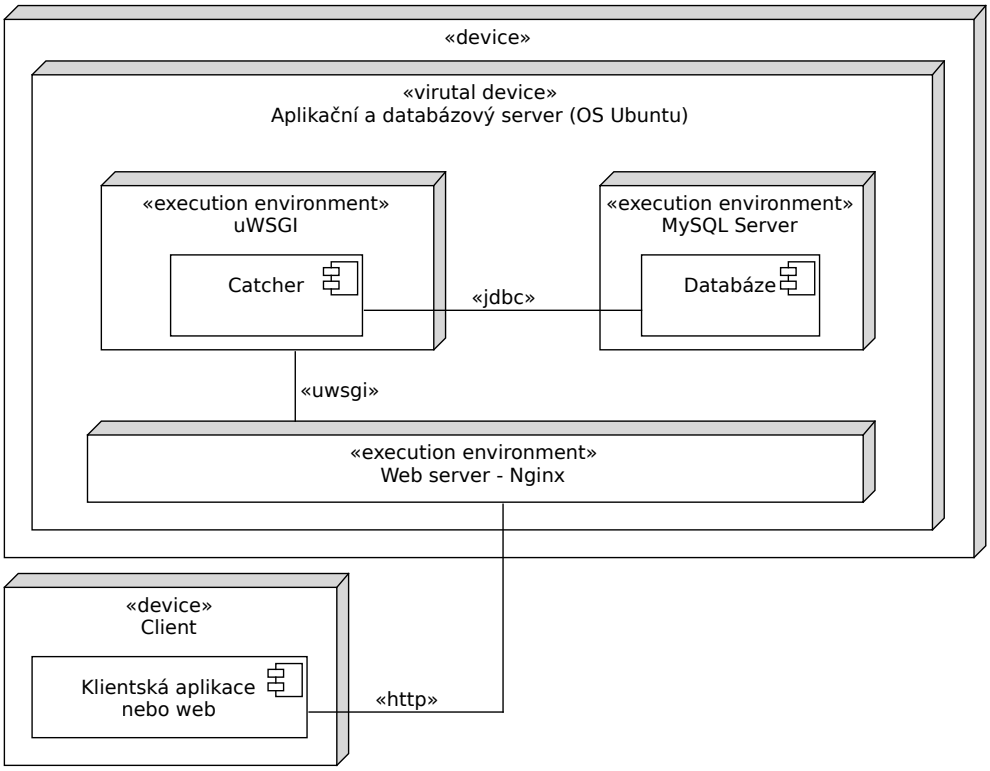
Jde o server běžící na virtualizovaném hardwaru, který se často označuje zkratkou VPS. Bývá poskytován hostingovými společnostmi, které svůj fyzický stroj nabízejí více klientům najednou. Každému zákazníkovi vymezí prostor s vlastní instancí operačního systému, jednotlivými hostingovými službami a vlastní konfigurací. Nevýhodu je sdílení fyzických prostředků, poněvadž při nedostatečném nastavení může docházet k ovlivňování cizích VPS. Výhodou je naopak levnější pořízení ve srovnání s celým fyzickým serverem.

7.1 Softwarové požadavky

Aplikace běží na VPS s OS Ubuntu [53] verze 15.10. Na veřejné adrese virtuálního serveru je dostupná RESTová webová služba Catcher, s níž může zvnějšku komunikovat libovolná aplikace, která má připojení k internetu. Jako prostředník mezi klientskou aplikací a uWSGI 2.0.12 [54], jehož úkolem je udržovat Catchera, zde běží webový server Nginx 1.9.3. Data se ukládají na databázový server MySQL verze 5.6.28.

Pro běh aplikace byly stáhnuty všechny potřebné balíčky včetně pipu [55], jenž slouží pro správu nesystémových knihoven v Pythonu. Detailní seznam všech závislostí obsahuje soubor `README.md` uložený v rodičovském adresáři projektu. Lepší správu souborů na serveru mi pak zajistil program Midnight Commander [56].

[23] Program, který umožňuje komunikaci zabezpečeným protokolem. Používá TCP/IP.

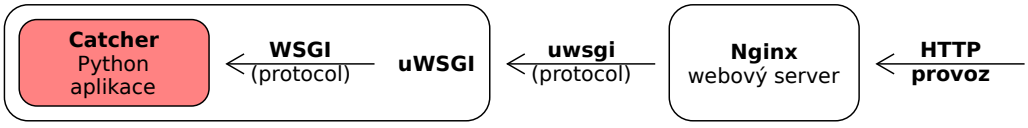


Obrázek 7.1: Diagram nasazení

7.2 uWSGI

Aby bylo lépe pochopitelné zapojení uWSGI do celého procesu nasazení, popíšeme si jej trochu detailněji.

Projekt uWSGI je malý program, který se stará o management procesu aplikace. Funguje jako prostředník mezi webovým serverem a aplikací, se kterými komunikuje vlastním uwsgi protokolem, respektive standardním WSGI. Detail komunikace je vidět na obrázku 7.2.



Obrázek 7.2: Komunikace mezi webovým serverem a aplikací v Pythonu; zdroj: autor na základě [57]

uWSGI úloha

Spuštění webové aplikace provadím následujícím příkazem. Proces poslouchá na portu 8080, odkud přijímá zprávy od webového serveru, a po celou dobu běží na pozadí.

```
$ uwsgi --socket localhost:8080 --wsgi-file ./restapi.py --callable api &
```

7.3 Konfigurace

Konfigurační soubor

Protože všechny moje zdrojové kódy byly zveřejněny v repozitáři na GitHubu, nebylo by správné, aby obsahovaly jakákoliv hesla. Pro tyto potřeby byl vytvořen konfigurační soubor, který mám zazálohovaný na vlastním cloudovém úložišti a pomocí programu wget [58] si jej stahuji do projektu vždy, když je změněn. Tento konfigurační soubor obsahuje hesla k produkční a testovací databázi nebo k e-mailovému účtu, který používám pro obnovu hesla uživatelů.

Databáze

Pro přístup do databáze bylo potřeba vytvořit uživatele s právem zapisovat, číst a mazat. Tento účet se nadále autentizuje heslem, které je uložené v konfiguračním souboru. Příkazy pro vytvoření uživatele v SQL jsou následující:

```
CREATE USER 'catcher-server'@'localhost' IDENTIFIED BY 'tajne_heslo';
GRANT SELECT, INSERT, UPDATE, DELETE
    ON catcher. * TO 'catcher-server'@'localhost';
```

Nginx

Protože disponuji pouze jednou veřejnou adresou na VPS, bylo potřeba webový server nakonfigurovat tak, aby dokázal obsluhovat požadavky webové aplikace i požadavky na zobrazení dokumentace. Konfigurace na serveru byla následující:

```
upstream catcher_uwsgi {
    # adresa a port, na které přeposílám požadavky pro Catchera
    server localhost:8080;
}

server {
    # číslo portu na ipv4
    listen       80;
    # číslo portu na ipv6
    listen       [::]:80;
    # jméno serveru
    server_name  catcher.zlutazimnice.cz *.catcher.zlutazimnice.cz;
    charset      utf-8;
```

```
    index        index.html;

    # maximální velikost uploadu
    client_max_body_size 75M;

    # dokumentace na catcher.zlutazimnice.cz
    location / {
        # adresář s obsahem webu
        root   /var/www/catcher;
    }

    # webová aplikace na catcher.zlutazimnice.cz/api
    location /api {
        include     uwsgi_params;
        # upstream pro spojení s uwsgi
        uwsgi_pass  catcher_uwsgi;
    }
}
```

Pro vývoj na svém počítači jsem použil trochu odlišnou konfiguraci, protože jsem nemusel na jedné adrese provozovat dokumentaci i aplikaci. V prvních fázích vývoje jsem dokonce ani Nginx používat nemusel, protože uWSGI umožňuje simulovat chování webového serveru. Pro produkční nasazení ale není tento postup doporučen.

7.4 Údržba

Standardem pro větší aplikace by měl být monitorovací systém, který sleduje zátěž jednotlivých komponent (webová aplikace, databáze, webový server atd.) a v případě výpadku informuje administrátora SMS zprávou nebo e-mailem. Pro potřeby Catchera jsem však využil daleko jednodušího řešení – webové aplikace Uptime Robot [59], která v pravidelných intervalech „oťukává“ mnou definovavné adresy a ověřuje jejich dostupnost. Pro použití v omezené míře je zdarma a v situaci, kdy je Catcher nedostupný, mě informuje e-mailem.

Dále je nutné pro případ výpadku pravidelně zálohovat databázi. Místo manuálního zálohování programem `mysqldump` [60], lze použít Cron[24], který by pravidelně tento program spouštěl. V případě výpadku by tak stačilo aplikaci opravit a nahrát poslední zálohu. Catcher aktuálně disponuje pouze manuálním zálohováním databáze.

[24] Jde o softwarového démona, který slouží k automatickému spouštění periodicky se opakujících příkazů a procesů.

Závěr

V práci jsem pro lepší pochopení uvedl čtenáře do problematiky, analyzoval již fungující i možná řešení a provedl návrh backendu, ze kterého vzešlo řešení vytvořit webovou službu nezávislou na webové aplikaci. Službu se podařilo implementovat pomocí jazyka Python a úspěšně nasadit na virtuální privátní server. Důležitým výsledkem je také sepsání dokumentace a vytvoření automatizovaných testů pro lepší odhalování chyb, jimiž mohu usnadnit práci nejen sobě, ale i všem dalším programátorům, kteří se budou na vývoji v budoucnu podílet.

Teď, když je celá práce za mnou, mohu s odstupem prohlásit, že více času jsem se měl věnovat návrhové části. Systém měl být daleko více objektově orientovaný. Ve zdrojových kódech dochází k nelogickému umístění částí kódu, které se měly nacházet na vhodnějších místech, například v třídních metodách a ne ve funkcích apod. Kromě toho je určitě prostor pro zlepšení webové služby po stránce bezpečnosti, především použití protokolu HTTPS.

Hlavním cílem práce bylo navrhnout a implementovat backend pro webovou aplikaci. I přesto, že moje původní představa týkající se malého rozsahu práce byla poměrně naivní, úspěšně jsem dokončil hlavní i dílčí cíle. Webová aplikace kolegy Jaroslava Veselého využívá mé řešení a na základě naší debaty máme další nápady, jak aplikaci rozšířit. V průběhu analýzy jsme také zjistili, že podobná aplikace pro jednoduchou správu turnajů v Ultimate Frisbee chybí. Pokud tedy úspěšně dokončenou aplikaci zveřejníme, vyplníme tím mezeru na trhu.

Literatura

[1] Co je Ultimate? *Česká Asociace Létajícího Disku* [online]. [cit. 2016-03-27]. Dostupné z: http://cald.cz/co-je-ultimate

[2] What is Ultimate? *USA Ultimate: About Ultimate* [online]. 2015 [cit. 2016-03-27]. Dostupné z: http://www.usaultimate.org/about/

[3] *The AUDL: American Ultimate Disc League* [online]. c2016 [cit. 2016-03-27]. Dostupné z: http://theaudl.com/

[4] *ESPN: The Worldwide Leader in Sports* [online]. c2016 [cit. 2016-03-27]. Dostupné z: http://espn.go.com/

[5] KOTĚŠOVEC, Petr. ČALD získala oficiální uznání od Českého Olympijského Výboru. In: *Česká Asociace Létajícího Disku* [online]. 2015 [cit. 2016-03-27]. Dostupné z: http://cald.cz/novinky/cald-ziskala-oficialni-uznani-od-ceskeho-olympijskeho-vyboru

[6] BERNACCHI, Chris a Bryan WALSH. Don't Let the IOC Ruin Ultimate Frisbee. *TIME* [online]. 2015 [cit. 2016-03-28]. Dostupné z: http://time.com/3982671/dont-let-the-ioc-ruin-ultimate-frisbee/

[7] Spirit of the Game. *Frisbee.cz* [online]. c2014 [cit. 2016-05-16]. Dostupné z: http://www.frisbee.cz/spirit-of-the-game.html

[8] Historie ultimate. *Česká Asociace Létajícího Disku* [online]. [cit. 2016-03-28]. Dostupné z: http://cald.cz/historie-ultimate

[9] Kalendař. *Česká Asociace Létajícího Disku* [online]. [cit. 2016-03-28]. Dostupné z: http://cald.cz/kalendar

[10] KOUDELKA, Jakub. *Metodika návrhu REST API*. Praha, 2013. Diplomová práce. Vysoká škola ekonomická v Praze. Vedoucí práce Lukáš Burkoň.

[11] GOURLEY, David a Brian TOTTY. *HTTP: the definitive guide*. 1st ed. Sebastopol, CA: O'Reilly, 2002, xviii, 635 p. ISBN 15-659-2509-2.

[12] CURBERA, Francisco, et al. Unraveling the Web services web: an introduction to SOAP, WSDL, and UDDI. *IEEE Internet computing*, 2002, 6.2: 86.

[13] RICHARDSON, Leonard a Sam. RUBY. *RESTful web services*. Farnham: O'Reilly, c2007. ISBN 05-965-2926-0.

[14] VOSEČEK, Jiří, Ondřej BURKERT a Petr KOTĚŠOVEC. Mobilní aplikace pro skórování na turnajích. In: *Česká Asociace Létjícího Disku* [online]. 2013 [cit. 2016-04-14]. Dostupné z: http://cald.cz/novinky/mobilni-aplikace-pro-skorovani-na-turnajich

[15] PHP and MySQL Manual. STOBART, Simon a Mike VASSILEIOU. *PHP and MySQL Manual Simple, yet Powerful Web Programming*. London: Springer London, 2004, s. 3. ISBN 978-0-85729-404-3.

[16] Catcher. *Google Play* [online]. c2016 [cit. 2016-04-21]. Dostupné z: https://play.google.com/store/apps/details?id=com.ulti.catcher

[17] Ultimate Organizer. *SourceForge* [online]. 2016 [cit. 2016-04-14]. Dostupné z: https://sourceforge.net/projects/ultiorganizer/

[18] *Ultimate Central: Find and manage ultimate frisbee events all over the world* [online]. [cit. 2016-04-17]. Dostupné z: http://ultimatecentral.com/

[19] LIGHT, Richard B. *Presenting XML*. Indianapolis, IN: Sams.net Pub., c1997. ISBN 15-752-1334-6.

[20] FIELDING, Roy Thomas. *Architectural styles and the design of network-based software architectures*. Irvine, 2000. ISBN 0-599-87118-0. Disertační. University of California, Irvine. Vedoucí práce Richard Taylor.

[21] IHRIG, Colin J. *Pro node.js for developers*. Berkeley: Apress, 2013, s. 263-270. Expert's voice in Web development. ISBN 978-1-4302-5860-5.

[22] BERNERS-LEE, Tim; FIELDING, Roy; MASINTER, Larry. *Uniform resource identifiers (URI): generic syntax*. RFC 3986, 1998. Dostupné z: http://www.hjp.at/doc/rfc/rfc3986.html

[23] *Python.org* [online]. c2001-2016 [cit. 2016-03-06]. Dostupné z: https://www.python.org/

[24] VAN ROSSUM, Guido. *A Brief Timeline of Python*. In: The History of Python [online]. 2009 [cit. 2016-03-06]. Dostupné z: http://python-history.blogspot.cz/2009/01/brief-timeline-of-python.html

[25] COCHRANE, Ken. *Best Python Companies to Work For*. In: DZone [online]. 2012 [cit. 2016-03-07]. Dostupné z: https://dzone.com/articles/best-python-companies-work

[26] KUCHLING, A. M. Functional Programming HOWTO: Introduction. PYTHON SOFTWARE FOUNDATION. *Python 2.7.11 documentation* [online]. c1990-2016 [cit. 2016-03-06]. Dostupné z: https://docs.python.org/2/howto/functional.html

[27] PYPL Index. *PYPL PopularitY of Programming Language* [online]. 2016 [cit. 2016-03-06]. Dostupné z: http://pypl.github.io/PYPL.html

[28] GOSLING, James, Bill JOY, Guy L. STEELE, Gilad BRACHA a Alex BUCKLEY. *The Java® language specification*. Java SE 8 edition. Addison-Wesley, 2000. ISBN 978-013-3900-699.

[29] RITCHIE, Dennis M. The Development of the C Language. *Second History of Programming Languages conference*. Cambridge, Mass., 1993. Dostupné také z: https://www.bell-labs.com/usr/dmr/www/chist.pdf

[30] *Falcon: The minimalist Python WSGI framework* [online]. [cit. 2016-03-06]. Dostupné z: http://falconframework.org/

[31] Benchmarks. *Falcon: The minimalist Python WSGI framework* [online]. [cit. 2016-03-06]. Dostupné z: http://falconframework.org/

[32] *MySQL* [online]. 2016 [cit. 2016-03-06]. Dostupné z: https://www.mysql.com/

[33] *Microsoft Developer Network: Structured Query Language (SQL)* [online]. 2016 [cit. 2016-03-06]. Dostupné z: https://msdn.microsoft.com/en-us/library/ms714670%28v=vs.85%29.aspx

[34] *MySQL: MySQL Workbench* [online]. c2016 [cit. 2016-03-07]. Dostupné z: https://www.mysql.com/products/workbench/

[35] DRIES, Buytaert. The history of MySQL AB. In: *Dries Buytaert: On digital experiences, Open Source, startups & the future* [online]. c1999-2016 [cit. 2016-03-06]. Dostupné z: http://buytaert.net/the-history-of-mysql-ab

[36] *MariaDB.org: The MariaDB Foundation* [online]. c2016 [cit. 2016-03-13]. Dostupné z: https://mariadb.org/

[37] *Percona: The Database Performance Experts* [online]. c2006-2016 [cit. 2016-03-13]. Dostupné z: https://www.percona.com/

[38] LEIFER, Charles. *Peewee* [online]. [cit. 2016-03-16]. Dostupné z: http://docs.peewee-orm.com

[39] *SQLite* [online]. [cit. 2016-03-21]. Dostupné z: https://www.sqlite.org/

[40] *PostgreSQL* [online]. [cit. 2016-03-21]. Dostupné z: http://www.postgresql.org/

[41] *GitHub* [online]. c2016 [cit. 2016-05-10]. Dostupné z: https://github.com/

[42] SOFTWARE FREEDOM CONSERVANCY. *Git* [online]. [cit. 2016-05-14]. Dostupné z: https://git-scm.com/

[43] DOLL, Brian. 10 Million Repositories. In: *GitHub: Blog* [online]. 2013 [cit. 2016-05-14]. Dostupné z: https://github.com/blog/1724-10-million-repositories

[44] *Nginx* [online]. [cit. 2016-04-26]. Dostupné z: http://nginx.org/

[45] Changes. *Nginx* [online]. 2004 [cit. 2016-04-27]. Dostupné z: http://nginx.org/en/CHANGES

[46] Top by ISP Seznam.cz. *Free SEO Analysis* [online]. [cit. 2016-04-26]. Dostupné z: http://www.seomastering.com/hosting/Seznam.cz/

[47] February 2016 Web Server Survey. *Netcraft* [online]. 2016 [cit. 2016-04-27]. Dostupné z: http://news.netcraft.com/archives/2016/02/22/february-2016-web-server-survey.html

[48] APACHE SOFTWARE FOUNDATION. *Apache HTTP Server Project* [online]. c1997-2016 [cit. 2016-05-14]. Dostupné z: https://httpd.apache.org/

[49] Dictionary. PYTHON SOFTWARE FOUNDATION. *Python 2.7.11 documentation: Built-in Types* [online]. c1990-2016 [cit. 2016-05-06]. Dostupné z: https://docs.python.org/2/library/stdtypes.html#dict

[50] json. PYTHON SOFTWARE FOUNDATION. *Python 2.7.11 documentation: JSON encoder and decoder* [online]. c1990-2016 [cit. 2016-05-06]. Dostupné z: https://docs.python.org/2/library/json.html

[51] ujson 1.35. PYTHON SOFTWARE FOUNDATION. *Python: Pypi* [online]. c1990-2015 [cit. 2016-05-06]. Dostupné z: https://pypi.python.org/pypi/ujson

[52] Unittest. PYTHON SOFTWARE FOUNDATION. *Python 2.7.11 documentation* [online]. c1990-2016 [cit. 2016-04-26]. Dostupné z: https://docs.python.org/2.7/library/unittest.html

[53] CANONICAL LTD. *Ubuntu* [online]. c2016 [cit. 2016-05-14]. Dostupné z: http://www.ubuntu.com/

[54] uWSGI 2.0.13.1. PYTHON SOFTWARE FOUNDATION. *Python: Pypi* [online]. c1990-2015 [cit. 2016-05-06]. Dostupné z: https://pypi.python.org/pypi/uWSGI

[55] pip 8.1.2. PYTHON SOFTWARE FOUNDATION. *Python: Pypi* [online]. c1990-2015 [cit. 2016-05-06]. Dostupné z: https://pypi.python.org/pypi/pip

[56] EDGEWALL SOFTWARE. *Midnight Commander* [online]. [cit. 2016-04-26]. Dostupné z: https://www.midnight-commander.org/

[57] PRAUS, Petr. Produkční nasazení Django aplikací na Cherokee pomocí WSGI. In: *Zdroják.cz* [online]. 2010 [cit. 2016-04-29]. Dostupné z: https://www.zdrojak.cz/clanky/produkcni-nasazeni-django-aplikaci-na-cherokee-pomoci-wsgi

[58] GNU Wget. FREE SOFTWARE FOUNDATION. *GNU Operating System* [online]. c2010, aktualizováno 27.4.2016 [cit. 2016-04-26]. Dostupné z: https://www.gnu.org/software/wget/

[59] *Uptime Robot* [online]. c2016 [cit. 2016-05-14]. Dostupné z: https://uptimerobot.com

[60] Mysqldump: A Database Backup Program. *MySQL* [online]. c2016 [cit. 2016-04-23]. Dostupné z: http://dev.mysql.com/doc/refman/5.7/en/mysqldump.html

Příloha **A**

Seznam použitých zkratek

API Application Programming Interface

CRUD Create, read, update and delete

CSS Cascading Style Sheets

ČALD Česká asociace létajícího disku

GUI Graphical user interface

HTTP Hypertext Transfer Protocol

HTTPS Hypertext Transfer Protocol Secure

JSON JavaScript Object Notation

OOP Object-oriented programming

ORM Object-relational mapping

RAM Random Access Memory

REST Representational State Transfer

SOTG Spirit of the Game

SOAP Simple Object Access Protocol

SQL Structured Query Language

SSL Secure Sockets Layer

TLS Transport Layer Security

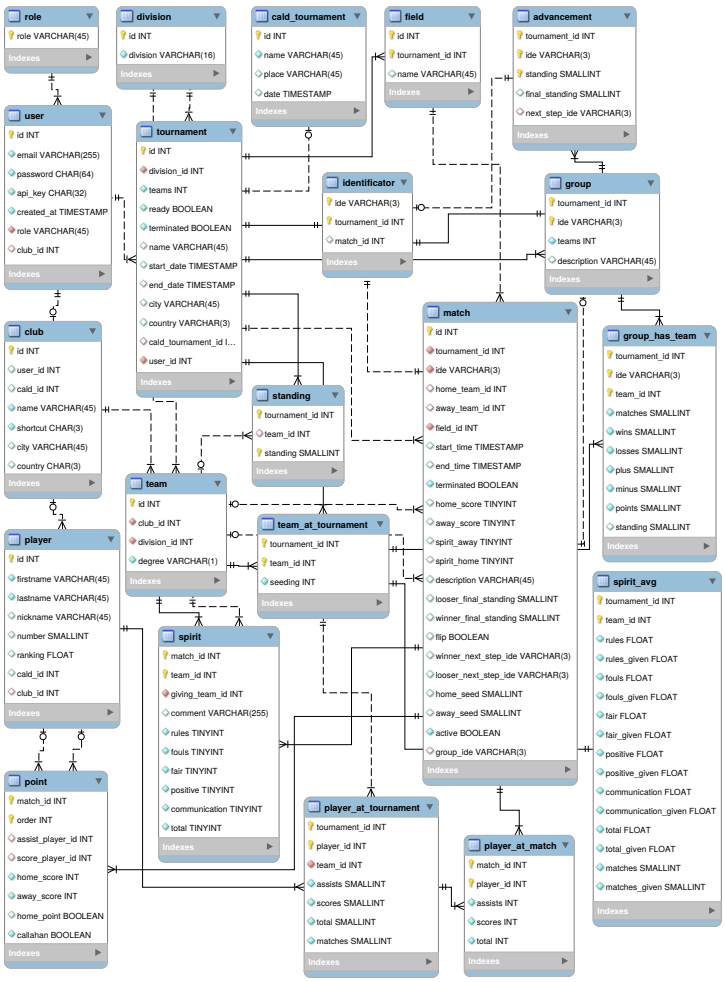
URI Uniform Resource Identifier

VPS Virtual private server

XML Extensible markup language

Příloha B

Datový model



Obrázek B.1: Datový model

Příloha C

Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `src`
  - `impl` ..... zdrojové kódy implementace
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF