FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

ZADÁNÍ DIPLOMOVÉ PRÁCE

Název:	Turnajový správce pro Ultimate Frisbee
Student:	Bc. Marek Dostál
Vedoucí:	Ing. Jiří Hunka
Studijní program:	Informatika
Studijní obor:	Webové a softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	Do konce letního semestru 2018/19

Pokyny pro vypracování

Cílem práce je vytvořit kvalitní uživatelské rozhraní a následně implementovat webovou aplikaci, která usnadní organizaci Ultimate Frisbee turnajů. Bude sloužit pro sběr výsledků a jejich poskytování veřejnosti. Téma práce je volným pokračování autorovy bakalářské práce, avšak cílem je navrhnout aplikaci od začátku s důrazem na uživatelské rozhraní. Celá práce bude těžit z předešlé zkušenosti autora.

1. Proveďte analýzu existujících řešení a návrh požadavků na aplikaci, zahrňte předešlou verzi vaší aplikace.
2. Na základě analýzy proveďte návrh uživatelského rozhraní webové aplikace, tento návrh podrobte vhodným testům.
3. Navrhněte vhodný backend, zvolte vhodnou architekturu.
4. Proveďte implementaci.
5. Navrhněte vhodnou sadu aplikačních a uživatelských testů finální aplikace.
6. Navrhněte možná budoucí rozšíření/vylepšení.

Seznam odborné literatury

Dodá vedoucí práce.

Ing. Michal Valenta, Ph.D.
vedoucí katedry

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
děkan

V Praze dne 11. ledna 2018

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Diplomová práce

Turnajový správce pro Ultimate Frisbee

Bc. Marek Dostál

Katedra softwarového inženýrství
Vedoucí práce: Ing. Jiří Hunka

8. května 2018

Poděkování

Chtěl bych poděkovat všem, kteří se podíleli na vzniku této práce, zejména Karolíně Zubaté 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 8. května 2018 .....

Odkaz na tuto práci

Dostál, Marek. *Turnajový správce pro Ultimate Frisbee*. Diplomová práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2018.

Abstrakt

Tato diplomová práce se zabývá analýzou, návrhem a realizací webové aplikace umožňující správu turnajů v Ultimate Frisbee. Podstatou řešení je vývoj aplikace v programovacím jazyce Python a její nasazení do provozu pomocí technologie Docker. Při návrhu byl kladen důraz na uživatelské rozhraní a jeho testování. Výsledkem je funkční webová aplikace.

Klíčová slova webová aplikace, Python, framework Flask, Docker, Ultimate Frisbee

Abstract

This master's thesis deals with analysis, design and realization of a web application that allows its users to manage Ultimate Frisbee tournaments. The essence of the solution lies in the development of the application in Python and putting it into operation via Docker technology. While designing the project, emphasis was put on user interface and its testing. The result of the project is a functional web application.

Keywords web application, Python, framework Flask, Docker, Ultimate Frisbee

Obsah

Seznam obrázků

Seznam tabulek

Úvod

Ultimate Frisbee je kompetitivní kolektivní sport hraný s létajícím talířem, tzv. frisbee. Aktivita, původně stvořená v USA, je tu již přes 50 let a během té doby se transformovala do populárního sportu, který bude dokonce v budoucích letech útočit na účast na Olympijských hrách.

S přibývající popularitou roste i počet týmů a soutěží, které se hrají téměř ve všech koutech světa. Pořádat jakoukoliv organizovanou akci přitom není jednoduchá záležitost, proto je třeba hledat možnosti, jak pořadatelům usnadnit práci.

Na menších akcích si můžeme často všimnout velmi snadných řešení, které jsou v konkrétních případech postačující. Například pro uchovávání výsledků postačí papír s tužkou, tabulka v počítači nebo statické webové stránky. Pro analýzu dlouhodobých statistik nebo jiné využití dat je ale tento postup nedostačující.

Hlavním cílem je tak navrhnout a implementovat webovou aplikaci, která bude zajišťovat správu turnajů, zaznamenávání výsledků a jejich následné poskytování veřejnosti. Součástí práce bude návrh uživatelského rozhraní a jeho testování. Dílčím úkolem je seznámení čtenáře s použitými technologiemi a zvolenou architekturou. Práce je volným pokračováním mé bakalářské práce z roku 2016, ze které v některých částech vycházím.

Při tvorbě mi byla motivací skutečnost, že já sám jsem aktivním hráčem Ultimate Frisbee. Práci dělám především pro přátele a kolegy, kteří tráví organizací turnajů příliš mnoho svého volného času. Vznikající systém má pracovní název Catcher.

Kapitola 1

Úvod do problematiky

1.1 Co je to Ultimate Frisbee

Ultimate Frisbee je poměrně mladým a stále se rozvíjejícím sportem, který se hraje s létajícím talířem. Vzniknul v 60. letech v USA a právě tam se již dnes hraje první profesionální liga AUDL [1]. Celosvětově jej hraje přibližně na 7 miliónů lidí a během několika přístích let se bude dokonce ucházet o místo na olympijských hrách [2]. Zajímavostí je, že v Ultimate Frisbee se hraje i smíšená kategorie (mix), tzn. ženy a muži dohromady. Jinak dalšími běžnými kategoriemi jsou open (muži), ženy, junioři (omezeni věkem) a masters (hráčky a hráči nad 33 let).

1.1.1 Pravidla

Ultimate je populární i díky tomu, že jeho pravidla jsou velmi jednoduchá. Hrají proti sobě dvě mužstva o sedmi hráčích na hřišti o rozměrech 100x37 metrů (délka fotbalového hřiště). Na obou koncích jsou zóny, podobné těm v rugby nebo americkém fotbalu, do kterých se skóruje. Bod je dosažen tím, že hráči útočícího týmu dopraví disk pomocí přihrávek do soupeřovy zóny a tam jej chytnou. Hráč, který má zrovna disk v držení, s ním nesmí běhat a musí jej do 10 vteřin odhodit. Pokud útočící tým ztratí disk (hozením do autu nebo na zem, zachycením přihrávky soupeřem, dlouhým držením disku), tak se útočícím týmem stáva soupěř. Hra je bezkontaktní.

1.1.2 Spirit of the Game

Sport je zcela výjimečný svým přístupem k hodnotám fairplay, protože už od počátku funguje zcela bez rozhodčích – od hráčů se očekává vzájemná ohleduplnost a smysl pro férovost. Všechny sporné situace řeší samotní hráči a to i na velmi kompetetivních soutěžích jako jsou třeba mistrovství světa.

Na turnajích se vyhlašuje cena Spirit of the Game (dále jako SOTG), ceněná obdobně jako první místo, která je určena nejčestnějšímu týmu. Cenu získává tým s nejvyšším průměrem hodnocení, které si týmy dávají navzájem po každém zápase. Jde o stupnici od 0 do 20 bodů.

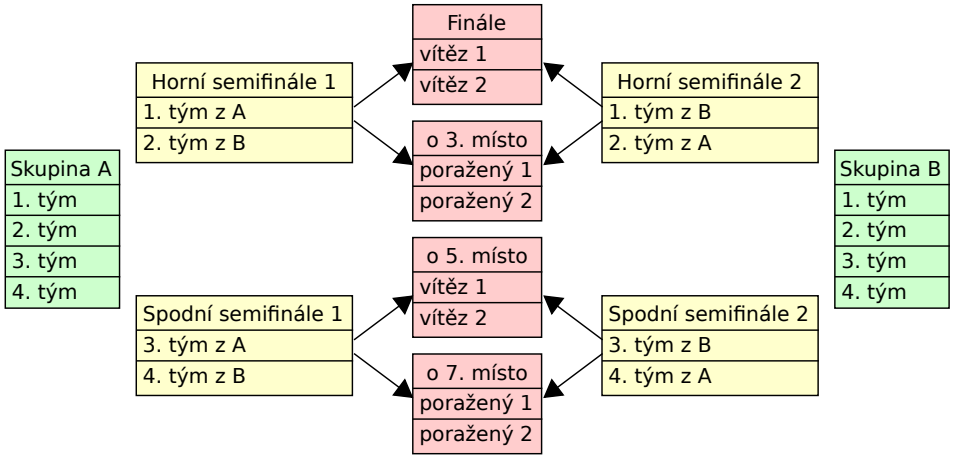
1.1.3 Ultimate v ČR

V České republice existuje několik desítek aktivních oddílů s největší koncentrací v Praze. Má to svůj důvod, protože Ultimate se v tuzemsku začalo hrát v 90. letech minulého století právě na půdě ČVUT [3]. Od té doby ale komunita hráčů raketově stoupá a Česká asociace létajícího disku (ČALD) se může za rok 2017 pyšnit 795 aktivními členy [4].

Většina soutěží v Ultimate probíhá o víkendech v podobě turnajů, na kterých se odehraje na desítky zápasů. S růstem sportu a hráčské základny roste poptávka po jejich online výsledcích.

1.1.4 Podoba typického turnaje

Nejčastější podoba Ultimate Frisbee turnajů je taková, že v první části se týmy utkají ve skupinách a podle jejich pořadí jsou doplněny do vyřazovací části turnaje. Na následujícím obrázku je takové schéma turnaje znázorněno.



Obrázek 1.1: Schéma typického turnaje pro 8 týmů [5]

1.2 Co je to webová aplikace

Jde o software, který je poskytován z webového serveru prostřednictvím počítačové sítě Internet nebo její obdobou – intranetem. V dnešní době je webová aplikace velmi populární, protože v podstatě vyžaduje pouze implementaci HTTP protokolu a je postavena na modelu klient-server, takže uživatelům

stačí vlastnit webový prohlížeč. Poněvadž prohlížeč neobsahuje žádnou logiku, nazývá se tenkým klientem.

Webové aplikace jsou často náhradou dřívějších aplikací typu klient-server, kde klientský program aplikací musel být instalován na osobních počítačích uživatele. Aktuální verze se tak spustí na serveru a všechny klientské aplikace – prohlížeče, pak přistupují k aktuální verzi. To přináší několik výhod:

- Nevyžaduje kooperaci klientských počítačů při zavádění nebo aktualizaci starší verze aplikace.
- Zrychluje vývoj a nasazení do provozu.
- Umožňuje se více a rychleji přizpůsobit požadavkům trhu.
- Kromě toho, že celý tento proces snižuje náklady na vývoj a údržbu, snižuje i nepřímé náklady jako jsou ceny za instalaci softwaru u zákazníků nebo zákaznickou podporu.
- Poskytuje platformní nezávislost a jednoduché použití.

Pro výstup webových aplikací se často používá HTML nebo XHTML kód [6], který je běžně podporován. Jde o statický dokument, který se často doplňuje dynamickými prvky, pro které se používá skriptovací jazyk JavaScript [7]. Ten může být využit například i pro validaci dat ve formulářích nebo dílčí výpočty, takže částečně obsahuje logiku aplikace. Pomocí JavaScriptu a techniky AJAX[1] [8] se také vytváří větší dojem desktopové aplikace, protože umožňují implementovat některé typické metody známé z desktopových aplikací, které standardní technologie prohlížečů nepodporují. Jde například o techniku drag and drop nebo aktualizaci dat bez nutnosti znovunačtení stránky, což může být pro uživatele rušivým elementem.

Aby webová aplikace mohla na serveru fungovat, vyžaduje webový server, který poslouchá na daném portu (typicky 80) a obsluhuje požadavky klientů. Mezi nejznámější webové servery patří Apache HTTP Server [9] nebo Nginx [10].

Pro implementaci aplikační logiky se používá nespočet různých programovacích jazyků, např. Java [11], PHP, Python [13], Ruby [14] atd.

[1] Asynchronous JavaScript and XML

Kapitola 2

Specifikace požadavků

Jak již bylo řečeno v úvodu, práce je volným pokračováním mé bakalářské práce, avšak cílem je navrhnout aplikaci od začátku s důrazem na uživatelské rozhraní. Protože výsledek této práce má nahradit stávající mobilní aplikaci, ponechal jsem práci pracovní název Catcher.

2.1 Požadavky na aplikaci

Při návrhu aplikace je kromě funkčích požadavků, které často tvoří konkurenční výhodu, nutné myslet i na požadavky nefukční. Ty mají mnohdy kritický vliv na fungování a kvalitu výsledného softwaru.

2.1.1 Funkční požadavky

Tvorba turnajů Základním požadavkem je tvorba turnaje. Tento proces se skládá z několika dílčích částí:

- výběr účastnících se týmů z databáze
- tvorba zápasů, včetně nastavení automatického posunování ve vyřazovací části turnaje, tzv. playoff (např. vítěz semifinále se doplní do finále a poražený do zápasu o třetí místo)
- tvorba skupin a vyhodnocení pořadí po odehrání všech zápasů v rámci této skupiny, včetně posunutí do předem určených zápasů

Správa turnaje V průběhu turnaje je možné zadávat konečné výsledky zápasů a hodnocení SOTG.

Tvorba statistik Systém udržuje data o všech odehraných turnajích – výsledky zápasů a průměrné hodnocení SOTG.

2.1.2 Nefunkční požadavky

Snadná rozšiřitelnost Návrh aplikace musí umožňovat snadné implementování dalších vlastností systému.

Nízká cena Úkolem je používat software i hardware tak, aby se minimalizovaly náklady na provoz. Vhodným řešením je používaní volně dostupných knihoven a technologií.

Výkon Nejvyšší zátěže bude aplikace dosahovat během víkendů, kdy probíhá velké množství turnajů. I přes nevyváženost provozu se ale bude jednat v kontextu běžného provozu webových aplikací o nízkozátěžový systém, protože počet požadavků za sekundu nebude větší než několik jednotek či desítek. Systém musí být schopen odpovědět na více než 95 % požadavků do jedné sekundy od přijetí.

Spolehlivost Systém musí dosahovat standardů v oblasti nasazování webových aplikací. Mezi ně patří:

- zálohování dat
- obnovení starších verzí aplikace
- logování vybraných situací
- informování administrátora v případě výpadku

Bezpečnost Vyžaduje se jednoznačné určení a ověření uživatelů.

Nároky na hardware Aplikace bude schopna pracovat na běžných virtuálních serverech, které neposkytují více jak 1024 MB RAM[2].

2.2 Uživatelské role

Nepřihlášený uživatel Návštěvník, jenž nemá žádné právo pro vytváření nebo modifikaci dat. Množina uživatelů s touto rolí bude majoritou. Jejich záměrem je pouze si prohlédnout statistiky (výsledky zápasů, hodnocení SOTG apod.). Jakýkoliv nepřihlašený uživatel se může zaregistrovat a následně přihlásit.

Přihlášený uživatel Uživateli se otevírá možnost vytvářet a spravovat svůj vlastní turnaj. Může vytvořit rozvrh zápasů, tzv. rozpis, upravovat seznam účastníků nebo má přístup ke všem hodnocením SOTG.

Administrátor Má povoleny všechny operace jako výše uvedené role s tím rozdílem, že bude mít přístup i k turnajům, které sám nevytvořil.

[2]Random Access Memory

2.3 Případy užití

V poslední části této kapitoly popíšeme nejběžnější příklady užití, které definují interakce mezi uživateli a systémem. Jedná se o důležitou součást softwarového inženýrství, neboť určuje směr návrhu.

Uživatelé

UC1: Registrovat nového uživatele Nepřihlášený uživatel se může zaregistrovat pomocí své platné e-mailové adresy. Ta je ověřena odesláním potvrzovacího odkazu.

UC2: Autentizovat uživatele Jinými slovy systém určí identitu uživatele a umožní mu přístup.

UC3: Resetovat heslo Při ztrátě nebo zapomenutí hesla lze zažádat o změnu hesla pomocí odkazu odeslaného na e-mailový účet.

UC4: Změnit heslo nebo e-mail Uživatel si může změnit e-mailovou adresu nebo heslo.

Týmy

UC5: Získat týmy Vrátí seznam všech dostupných týmů včetně divize nebo místa původu.

Turnaje

UC6: Vytvořit turnaj Uloží do databáze turnaj a základní informace jako je název, termín akce, místo konání apod.

UC7: Získat turnaje Vrátí data o jednom či více turnajích.

UC8: Editovat turnaj Modifikuje základní informace o turnaji.

UC9: Získat konečné pořadí Vrátí celkové pořadí týmů, včetně hodnocení SOTG, které je veřejně známé až po skončení turnaje.

UC10: Smazat turnaj Turnaji se pouze nastaví příznak „smazaný“, ale v databázi zůstane.

Nasazení

UC11: Přiřadit týmy k nasazení Nasazení se obecně snaží reflektovat sílu týmů a rozložit turnaj tak, aby se lepší týmy potkávali až v pozdější fázi turnaje. Například dva týmy s nasazením 1 a 2 se zpravidla nepotkají v prvním zápase na turnaji. Aby mohl být turnaj opětovně používán,

je vhodné pro tvorbu zápasů a skupin používat nasazení, které může být každý rok jiné. Dovoluje to také vytvářet turnaj ještě před tím, než známe všechny účastníky turnaje.

UC12: Získat nasazení Vrátí seznam seřazený podle nasazení, eventuálně s přiřazenými týmy.

UC13: Editovat nasazení Modifikuje seznam týmů v kontextu nasazení.

Skupiny

UC14: Vytvořit skupinu Skupina se definuje počtem týmu a nastavením, které týmy budou skupinu hrát.

UC15: Získat skupiny Vrátí data o jedné či více skupinách.

UC16: Editovat skupinu Modifikuje nastavení skupiny.

UC17: Smazat skupinu Fyzické smazání skupiny z databáze.

Zápasy

UC18: Vytvořit zápas Zápas se definuje kromě základních informací jako je datum, čas, doba trvání nebo místo také nastavením, jaké týmy budou zápas hrát. Protože v aplikaci se pracuje s nasazením, je možné přiřadit do zápasu týmy pomocí něj. Jsou tu ale i další atributy, kterými do něj lze „nadrátovat“ týmy podle výsledků z předchozích zápasů nebo skupin.

UC19: Vygenerovat skupinu zápasů Pro nejběžnější případy umožní aplikace vytvořit již nachystané skupiny zápasů.

UC20: Získat zápasy Vrátí data o jednom či více zápasech včetně hodnocení SOTG.

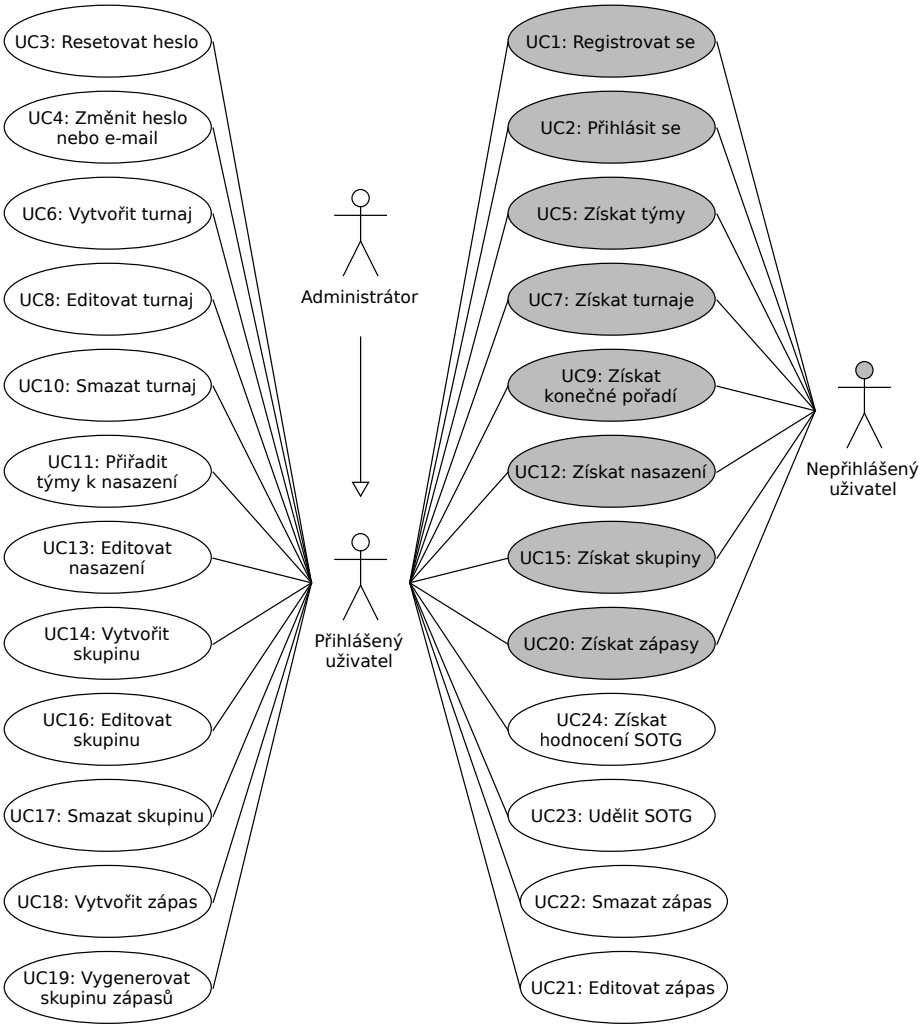
UC21: Editovat zápas Modifikuje nastavení zápasu.

UC22: Smazat zápas Fyzické smazání zápasu z databáze.

Spirit of the Game

UC23: Ohodnotit tým v zápase v kategorii SOTG Po zápase je umožněno organizátorům hodnotit oba dva týmy v kategorii Spirit of the Game. Hodnocení se skládá ze stupnice od 0 do 20 bodů. Toto hodnocení je veřejné až po skončení turnaje.

UC24: Získat souhrnné hodnocení SOTG Vrátí seznam týmů včetně všech hodnot hodnocení SOTG jako je suma obdržených a udělených bodů, počet odehraných zápasů a průměr.



Obrázek 2.1: Use case diagram

Kapitola 3

Analýza

V následující sekci se podíváme na několik podobných funkčních projektů. Zaměříme se především na jejich vlastnosti a uživatelské rozhraní. Využijeme při tom poznatky získané z magisterského předmětu MI-NUR, kde byla tato analýza předmětem týmové semestrální práce. Zbytek této kapitoly se bude zabývat možnostmi našeho řešení.

3.1 Existující řešení

Mobilní aplikace Catcher

Mobilní aplikace pro operační systém Android z roku 2013, jež byla vytvořena dvěma českými hráči Jiřím Vosečkem a Ondřejem Burkertem [15]. I díky tomu, že jejími autory jsou dva aktivní hráči, si ihned po zveřejnění našla své místo v komunitě hráčů Ultimate Frisbee. Aplikace umí pracovat s výsledky zápasů a skupin, hodnocením SOTG a statistikami jednotlivých hráčů.

Do úpravy oficiální databáze České asociace létajícího disku v roce 2017 byla aplikace dokonce schopná, i když omezeně, importovat oficiální soupisky týmů z ligových soutěží. I přes tuto ztrátu funkčnosti však aplikace dál zůstává standardem na českých turnajích.

Součástí aplikace je i backend v PHP, který se stará o zpracování a perzistenci dat na serveru. Možnosti rozšířit jej jsou však velmi náročné, protože aplikace nebyla navržena objektově a její rozsah by to vyžadoval.

Výhody

- Backend aplikace využívá webový portál frisbee.cz, který poskytuje data i uživatelům bez mobilní aplikace Catcher.
- Přes webový prohlížeč lze přistupovat k emulátoru[3] aplikace. Organizá-

[3]Software umožňující běh aplikací na jiné platformě, než pro kterou byly původně vytvořeny

Obrázek 3.1: Mobilní aplikace Catcher [16]

torům je tak umožněno dělat změny z běžného počítače.

- Aplikace má již přes 5 tisíc stáhnutí [16] na Google Play[4]
- Aplikace je na míru vytvořená pro Ultimate Frisbee. Nejde o žádný univerzální systém pro více sportů.

Nevýhody

- Aplikace je prakticky dále nerozšiřitelná.
- Neposkytuje žádné standardní rozhraní pro další klientské aplikace.
- Všechny zápasy jsou spojeny s konkrétnímy týmy. Každý další turnaj se tak musí celý vytvářet znovu.
- Neumí dynamicky dosazovat týmy do dalších částí turnaje podle jejich výsledků. Uživatel musí sám každý takový zápas nebo skupinu ručně vytvořit a vybrat jeho účastníky.

[4]Online distribuční služba, kde jsou k dispozici aplikace pro zařízení s operačním systémem Android.

- Vytvářet turnaje může pouze administrátor. Při tomto procesu navíc nedochází k žádnému validování dat. Masivní využívání aplikace je tak nemožné.

Ultimate Central

Webová aplikace [18] vytvořená v USA v roce 2010 [19] je již poměrně komplexní. Za poplatek dovoluje registrovaným uživatelům vytvářet stránky, pod kterými spravují vlastní obsah, turnaje a grafickou podobu. Aplikace umožňuje všechny základní vlastnosti podobných systémů: práci s výsledky a hodnocením SOTG.

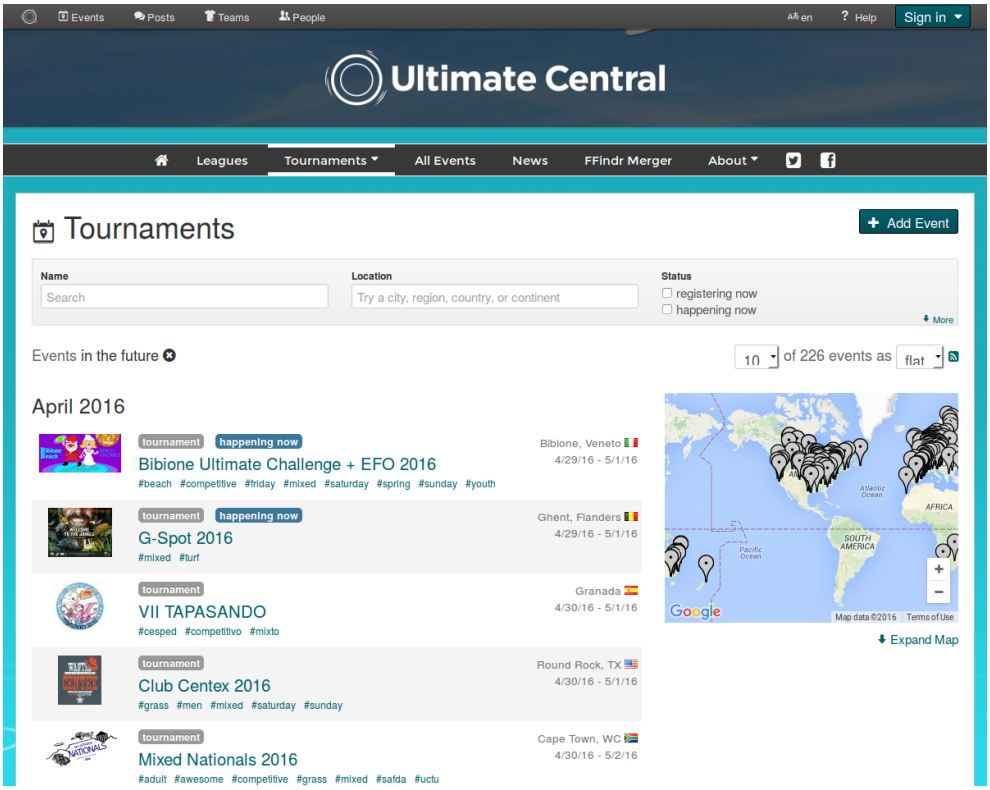
Největší přidaná hodnota je však v procesu komunikace s hráči, protože ti musí před turnajem provést akci potvrzení účasti a to dovoluje organizátorům zjistit informace o hráčích. Příkladem je následující proces registrace týmů a hráčů:

1. Registrovaný uživatel vytvoří turnaj se všemi potřebnými informacemi
2. Jednotlivé týmy vidí turnaj v kalendáři a mohou se rozhodnou na něj přihlásit
3. Organizátor rozhodne, které týmy se turnaje zůčastní
4. Jednotliví hráči pak potvzují svoji účast formulářem, ve kterém sdělují organizátorům kustomizovatelné informace. Např. jestli mají zájem o ubytování, typ večeří apod.

Na velkých akcích jako jsou mezinárodní mistrovství, kde je povinné odevzdávání soupisek nebo potvrzování podmínek účasti, je tato vlastnost zcela vyhovující. Pro menší turnaje je však celý tento proces nepříjemný.

Výhody

- Systém je dostupný ve webovém prohlížeči
- Poskytuje snadný nástroj pro přihlašování týmů. Nahrazuje posílání pozvánek do e-mailových schránek a následnou komunikaci.
- Umožňuje organizátorům vyžadovat po jednotlivých hráčích kustomizovatelnou operaci, např. vyplnění formuláře nebo potvrzení PDF dokumentu.
- Systém má za několik let provozu velkou databázi hráčů.



Obrázek 3.2: Webová aplikace Ultimate Central

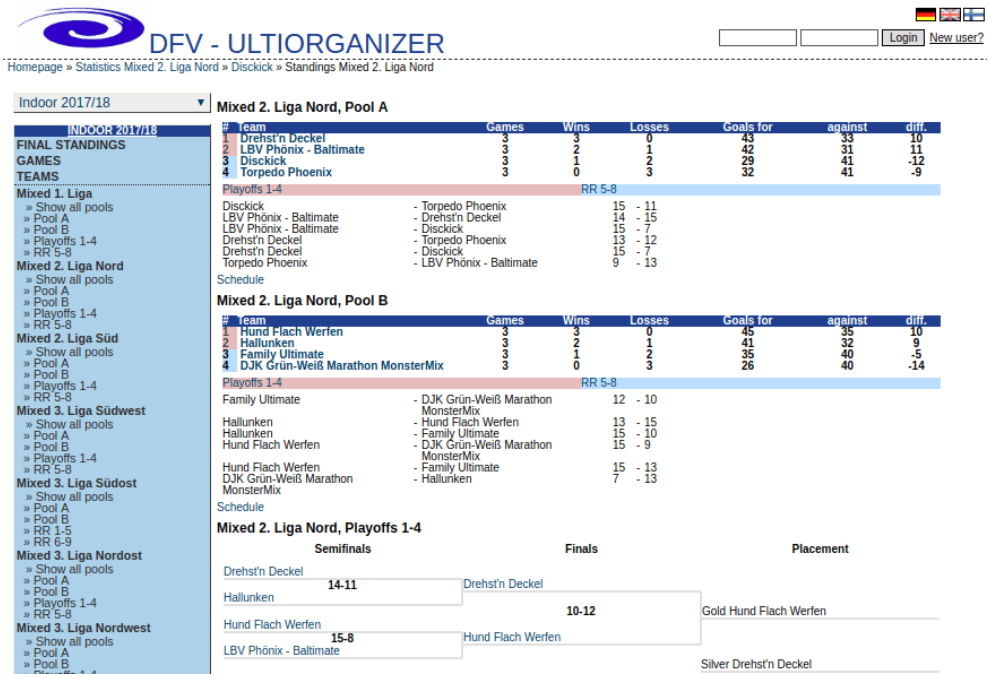
Nevýhody

- Neposkytuje sledování dlouhodobých statistik. Systém ukládá výsledky zápasu a hodnocení SOTG, ale je vhodný spíše pro správu účastníků než samotného turnaje.
- Vyžaduje registraci každého účastníka.
- Neposkytuje veřejné rozhraní pro klientské aplikace.
- Pro zobrazení výsledků má velmi nevhodné uživatelské rozhraní: z části nefunkční mapa, nevhodné pojmenování některých odkazů nebo tlačítek, chybějící zprávy o úspěšnosti nebo neúspešnosti operací.
- Některé vlastnosti jsou placené.

Ultimate Organizer

Aplikace napsaná v jazyce PHP pochází z roku 2004 [17]. Od té doby prošla překotným vývojem a poslední změny jsou datované na konci roku 2016 [17].

Často se používala pro velké akce jako jsou mistrovství světa nebo Evropy, avšak v poslední době její obliba klesá na úkor jiné aplikace – Ultimate Central. Dojem z Ultimate Organizer, který umožňuje všechny základní vlastnosti jako tvorbu rozpisu, zadávání výsledků nebo dokonce detailní průběhy zápasů, kazí pouze fakt, že nepracuje s hodnocením SOTG. Dále poskytuje vícejazyčnost, tisk rozpisů v PDF, synchronizaci s kalendářem a mobilní rozhraní pro zadávání výsledků.



Obrázek 3.3: Webová aplikace Ultimate Organizer

Výhody

- Jde o open-source software.
- Komplexní aplikace s nepřeberným množstvím funkcí.
- Více jak 10 let úspěšného fungování.
- Zčásti kustomizovatelná aplikace.

Nevýhody

- Nutné instalovat na vlastním serveru.

- Jde o webovou aplikaci, která nemá responzivní design. Mobilní rozhraní slouží pouze pro některé části systému. Pro většinu uživatelů s mobilním telefonem je tak používání aplikace nepohodlné.
- Neumí pracovat s hodnocením Spirit of the Game.
- Bez možnosti implementovat nového klienta, protože systém neposkytuje žádné rozumné rozhraní.
- Nesleduje statistiky dlouhodobě, ale pouze ke konkrétnímu turnaji.

Leaguevine

Jde o univerzální webovou aplikaci [20] pro několik sportů, včetně Ultimate Frisbee. Slouží pro správu výsledků a nabízí dokonce hodncení SOTG.

Výhody

- Neplacená aplikace.
- Poskytuje API rozhraní.
- Mobilní verze webové stránky.

Nevýhody

- Složitý proces vytvoření turnaje.
- Nesleduje statistiky dlouhodobě, ale pouze ke konkrétnímu turnaji.

Leverade, Konkuri a Enjore

Jeden španělský [21] a dva italské projekty [22] [23], z větší části placené, jsou zaměřené na desítky sportů nebo videoher. Ani jeden z nich nenabízí Ultimate Frisbee, tudíž ani podporu hodnocení Spirit of the Game.

Všechny tři aplikace jsou univerzálním pomocníkem pro tvorbu turnajů nebo lig a podle toho vypadají. Mají často nelogické uspořádání obsahu, který je připraven pro jiné sporty nebo univerzální popisky, které se pro některé sporty nehodí. Tvorba zápasů je také kvůli univerzálnosti z části omezená a neposkytuje všechny vyžadované scénáře.

Všechny tři projekty mají jednu nebo dvě mobilní aplikace pro uživatele operačních systému Android a iOS.

3.1.1 Výsledek

Průzkum potvrdil, že i přes velké množství projektů pro tvorbu sportovních soutěží, neexistuje aplikace, která by poskytovala jednoduchou a zároveň komplexní tvorbu rozpisů, automatizované doplňování do dalších fází turnaje nebo hodnocení SOTG.

3.2 Možnosti řešení

Před samotným návrhem aplikace se ještě krátce zaměříme na to, jaké jsou možnosti při vývoji webové aplikace, kterou jsme popsali v sekci 1.2.

3.2.1 Vícevrstvá architektura

Označuje aplikace, které jsou složeny z více vzájemně spolupracujících vrstev, které často pracují na různých místech (infrastruktuře). Nejznámějším případem je třívrstvá architektura, která umožňuje díky „prostřední“ – aplikační – vrstvě, narozdíl od dvouvrstvé, řazení požadavků do fronty, jejich plánování a další podobnou logiku. I tím může být dosaženo vyššího výkonu. Mluvíme o těchto třech vrstvách:

Datová Stará se o perzistenci dat a přístup k nim. Jde nejčastěji o databázi, ale kromě ní lze uvažovat i o souborovém systému, webové službě nebo jiné aplikaci.

Aplikační Provádí všechny klíčové funkce aplikace. Jde o logiku, výpočty a zpracovávání dat.

Prezentační Znázorňuje informace uživatelům a někdy kontroluje zadávané vstupy, které však nezpracovává.

3.2.2 Frameworky pro tvorbu webových aplikací

Prakticky každý vývoj webové aplikace zahrnuje stejné nebo velmi podobné funkční požadavky. Samotné programovací jazyky jejich podporu často přímo neposkytují, proto je nutné využít hotové knihovny, které tyto problémy pomáhají řešit. Ucelený soubor takových knihoven, který zajišťuje souhrnnou funkcionalitu potřebnou pro vývoj webové aplikace, tvoří tzv. framework. Kromě toho, že nabízí použití vybraných knihoven, definuje taky postup tvorby kódu. Pomáhá tak udržovat aplikační struktury, názvy nebo umístění souborů a adresářů. Jde tedy o celý koncept vývoje webové aplikace.

U rozsáhlejších projektů je použití frameworků vítaným rozhodnutím, protože usnadňuje správu, zjednodušuje vývoj, maximalizuje produktivitu a usnadňuje budoucí rozšíření aplikace. Ekonomickou otázkou bývá, zda použít vlastní nebo již hotové, otestované a často open source frameworky. Vyvinout vlastní

framework však bývá poměrně nákladnou záležitostí, nehledě na to, že nás nutí zaučovat vývojáře pro práci s ním. Druhá možnost tak přináší časovou úsporu a zlevnění vývoje. Vývojáři se mohou více zaměřovat na „business“ logiku než na samotnou architekturu.

3.2.3 Koncept MVC

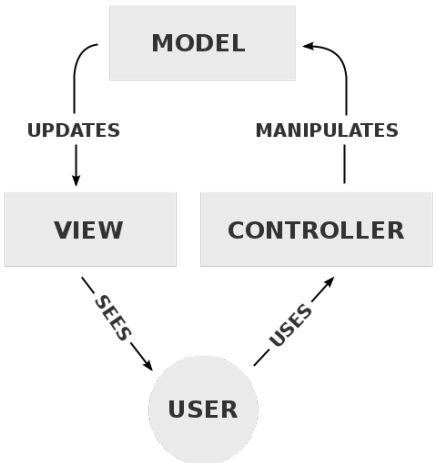
V kontextu webových aplikací se často setkáváme s pojmem *Model-View-Controller* [24]. Jedná se o návrhový vzor poprvé použitý v programovacím jazyce SmallTalk [6]. Někdy označován jako Model 2.

MVC umožňuje aplikaci rozdělit na tři na sobě nezávislé komponenty – datovou, aplikační a prezentační, kde každá z nich má svoje dílčí úkoly a spolu jsou ve spojení pomocí definovaného rozhraní.

Model Reprezentuje aplikační logiku, která řeší jádro webové aplikace. Validuje vstupní data a řeší přístup k nim. Části View umožňuje získat data.

View Má na starosti grafický nebo jiný výstup aplikace, nejčastěji HTML kód. Vykresluje Model, od kterého získává informace o změnách. View se také stará o zachytávání událostí od uživatele, které předává Controlleru.

Controller Zodpovídá za chování aplikace. Pracuje se vstupy a událostmi uživatelů, na základě kterých volá vybrané části Modelu a mění jeho stav. Vstupem bývá HTTP GET nebo HTTP POST požadavek odeslaný z uživatelova prohlížeče.



Obrázek 3.4: Model-View-Controller schéma [25]

Všem webovým aplikacím nevyhovuje výše uvedený model smyčky $View \rightarrow Controler \rightarrow Model \rightarrow View \ldots$, proto je často modifikován tak, že Controller se nachází mezi Modelem a View a sám kontroluje celou aplikaci. I tak se jedná o základ u mnoha projektů.

3.2.4 Shrnutí

Protože výše uvedený model je dnes zcela běžným návrhovým vzorem pro vývoj webových aplikací, budu z něj vycházet v implementaci. Provedeme však několik změn, které jsou popsány v kapitole 5.

Kapitola 4

Návrh

Následující kapitola je rozdělena do dvou částí. První se zabývá uživatelským rozhraním a jeho návrhem. Ve druhé půlce se budeme zabývat návrhem backendu a výběrem vhodné architektury.

4.1 Návrh uživatelského rozhraní

Velká část této sekce vznikla v rámci týmové semestrální práce předmětu Návrh uživatelského rozhraní na Fakultě informačních technologií ČVUT. Cílem práce bylo udělat několik dílčích úkolů (např. vytvoření Lo-Fi[5] a Hi-Fi[6] prototypu), které plně pokrývají potřeby této diplomové práce pro návrh uživatelského rozhraní. Členy týmu, pracujícím na návrhu této diplomové práce, byli spolužáci Jančovičová Barbora, Larionova Veronika a Kozlov Andrej. Protože jsem přišel s nápadem na projekt a měl jsem jeho největší znalost, staral jsem se o vedení týmu já.

4.1.1 Rozdíl mezi Lo-Fi a Hi-Fi protypem

Častým požadavkem na nové rozhraní bývá to, aby aplikace byla přehledná. V kontextu návrhu uživatelského rozhraní nejde ani o to, že by aplikace měla mít nádhernou grafiku, ale spíše takové prostředí, ze kterého budou mít uživatelé co nejlepší dojem a bude se jim v něm dobře pracovat. Sebelepší grafický návrh totiž uživatelům nepomůže, pokud budou v aplikaci ztraceni nebo nebudou rozumět, proč se dějí některé akce. Obecně se jedná o komplexní problém, protože do něj přispívá mnoho faktorů – osobnost, inteligence a zkušenost uživatele, grafický návrh, barvy a další technické aspekty.

U větších aplikací je vhodné tento problém řešit ještě před samotným vývojem. Při testování s koncovými uživateli se totiž často objevují nedostatky,

[5] Low-Fidelity
[6] High-Fidelity

které opravit stojí mnoho času a peněz, pokud se dělají, až když je aplikace vytvořená.

Zmírněním tohoto rizika je tak vytvoření prototypu s omezenou funkčností, který se může otestovat na budoucích uživatelích a jednoduše upravovat na základě jejich připomínek. Takový prototyp má několik dalších výhod:

- ukáže, zda řešení je realizovatelné
- pomocí něj lze zmapovat a odhadnout rozsah budoucího projektu
- odhalí nejasné zadání v některých částech aplikace
- může sloužit jako referenční zadání pro dodavatele softwaru
- představuje pouze malou část nákladů na vývoji celé aplikace

Lo-Fi prototyp

Nejjednodušší prototyp, jenž může být vytvořen na papíře nebo pomocí online nástrojů. Výstupem jsou tzv. wireframy, které obsahují strohé obrazovky s obyčejnými barvami, schématickými návrhy formulářů, jednoduchá tlačítka a generované texty. Tento prototyp není zaměřen na detaily a jeho tvorba tak může být extrémně rychlá a levná. Další výhodou je taky fakt, že nestrhává nežádoucí pozornost ke grafice a barvám, jež by mohly zastínit diskuzi o smyslu aplikace a navržené informační architektuře.

Zpravidla se začíná právě od Lo-Fi prototypu, který se iteračně upravuje po oponenturách a testech. Později se přechází na Hi-Fi prototyp, který dovoluje testovat další elementy aplikace.

Hi-Fi prototyp

Prototyp Hi-Fi se svým chováním a vzhledem více blíží realitě, protože jeho prvky obsahují výrazně více detailů. Uživatelé si mohou vyzkoušet konkrétní ovládací prvky, interakce, validaci formulářů, animace apod. Tento prototyp je již obvykle tvořen pomocí standardních technologií (HTML, CSS, JavaScript ...), pokud jde o webovou aplikaci.

Při návrhu obou prototypů se také vycházelo z Nielsenovy heuristické analýzy [55], která je popsaná v následující části.

Nielsenova heuristická analýza

Tato heuristická analýza dokáže rychle odhalit hlavní nedostatky návrhu uživatelského rozhraní a zároveň umožňuje provést uživatelské testování bez uživatelů. Analýza se skládá z deseti pravidel, které vznikly z poznatků získaných dlouhodobým pozorováním. Už při návrhu prototypu je dobré na tyto základní pravidla pamatovat:

Viditelnost stavu systému Systém neustále informuje uživatele o tom, co se děje. Např. místo „zamrznutí“ zobrazí progress bar[7].

Shoda mezi systémem a realitou Aplikace mluví jazykem uživatelů. Ikony a texty odpovídají tomu, co by uživatel očekával i v reálném světě.

Uživatelská kontrola a svoboda Uživatel se může vrátit z určitého stavu zpět. Například pomocí tlačítek *storno* nebo *undo*. Před nevratnými akcemi se nachází varování apod.

Konzistence a standardizace Aplikace používá standardní systémové ovládací prvky a nevyskytuje se v ní více různých pojmů pro jednu akci.

Prevence chyb Systém se snaží zabránit chybám, např. zvýrazněním povinných položek ve formuláři, potvrzovacími dialogy atd. Uživateli se doslova brání v tom, aby zadal špatnou hodnotu.

Rozpoznání místo vzpomínání Uživatel dokáže ihned rozeznat, kde v aplikaci se nachází. Dále se mu zobrazují pouze relevantní akce a informace.

Flexibilita a efektivita používání Pokročilým uživatelům systém poskytuje možnosti, které jejich práci zrychlí, např. klávesové zkratky nebo šablony pro vytváření opakovaných dat.

Estetický a minimalistický design Nezobrazují se zbytečné volby a informace, aby byla práce se systémem jednoduchá, přehledná a rychlá. Méně používané prvky je lepší umístit na méně důležité obrazovky.

Pomoc uživatelům poznat, pochopit a vzpamatovat se z chyb Primárně by se měla aplikace snažit chybovým hlášením vyhnout, tzn. nedovolit uživateli se dostat do chybového stavu. V opačném případě by chybové hlášení mělo jasně a zřetelně popsat, co se stalo. Uživateli tak pomůžeme pochopit co má příště udělat jinak.

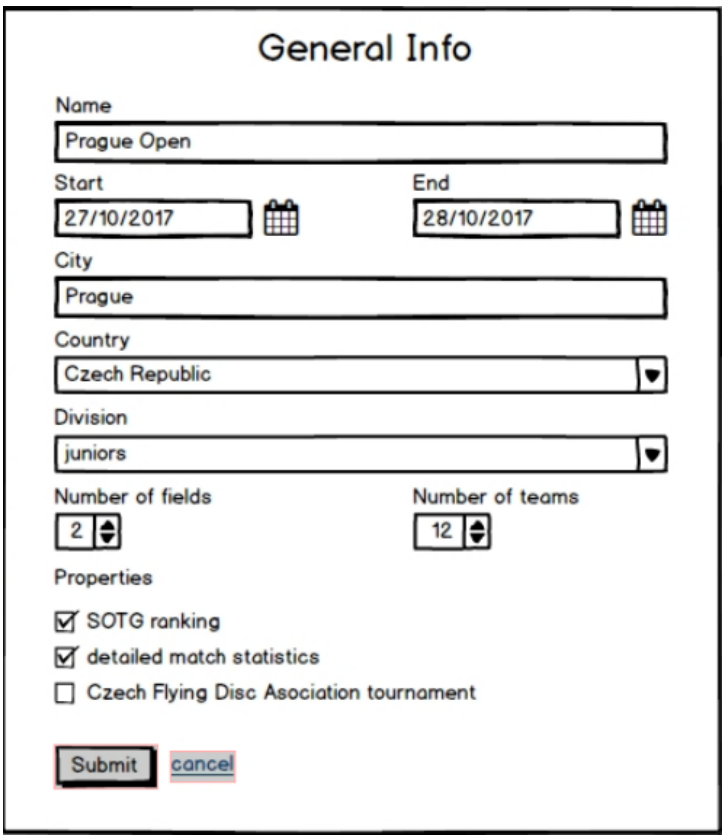
Help a dokumentace Pokud je uživatel v některých krocích ztracený, měl by mít možnost se podívat na nápovědu. Ta může být řešena kontextově, přímo u daného prvku, nebo globálně.

Těchto deset základních pravidel v našem Hi-Fi prototypu odhalilo několik nedostatků. Mezi ně patřila chybějící dokumentace, nedostatečná validace formulářů, nevhodné chybové hlášky nebo špatné rozeznání místa v aplikaci, kde se uživatel zrovna nachází. Všechny tyto problémy byly ještě v rámci předmětu MI-NUR z velké části napraveny.

[7] Indikátor průběhu

4.1.2 Tvorba prototypů a jejich testování

Před vytvořením Lo-Fi prototypu jsme sestavili seznam jednotlivých akcí a zobrazení seskupených podle toho, jak spolu racionálně souvisí. To nám následně pomohlo k uspořádání obrazovek a vytvoření prototypu, který se ještě několikrát měnil podle připomínek koncových uživatelů.



Obrázek 4.1: Popup okno pro editaci turnaje v Lo-Fi prototypu

Po několika těchto iteracích jsme dospěli k návrhu, který jsme transformovali do Hi-Fi prototypu. Ten kromě Lo-Fi prototypu vycházel i z poznatků z rešerší, které jsme také prováděli v rámci semestrální práce. Toto funkční uživatelské rozhraní bez datové části, které reagovalo na vstupy uživatelů a chovalo se již podle očekávání, se stalo předmětem testování – kognitivního průchodu[8] s několika vybranými uživateli: Martin Žid, Michal Kluzáček, Jaroslav Veselý, Adam Kugler, Jan Zípek a Michal Kváček (studenti FIT ČVUT). Hotový a opravený Hi-Fi prototyp se na základě testování stal podkladem pro tuto diplomovou práci.

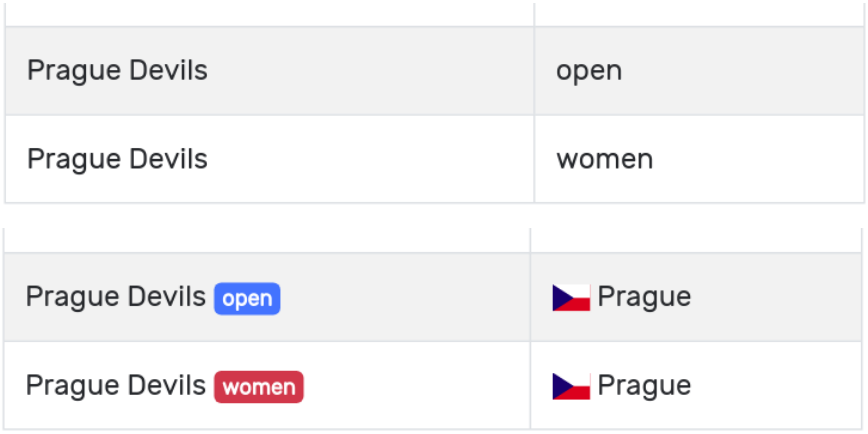
[8]Metoda, která je založena na tom, že průchod aplikace si ještě před reálnými uživateli vyzkouší testeři.

Nejčastější problémy v testování Hi-Fi prototypu uživateli

Testování na uživatelích pomohlo identifikovat několik drobných problémů, které se taky řešily ještě na úrovni Hi-Fi modelu.

Problém	Popis
Nutnost editovat zápas na více místech aplikace.	Uživatel vždy najde pouze první editaci a získá dojem, že druhá již neexistuje.
Nedostatečné rozlišení týmů se stejným jménem v různých divizích.	Nestačilo uvedení divize na stejném řádku tabulky.
Nevhodná barva tlačítek.	Tlačítko se stejnou barvou jakou má pozadí stránky ale inverzní barvou na okrajích je nedostatečné. Je tudíž žádané odlišit barevně všechny tlačítka.
Nevhodně pojmenované názvy položek v menu a v některých formulářích.	Uživatelé často hledali na špatných místech aplikace.

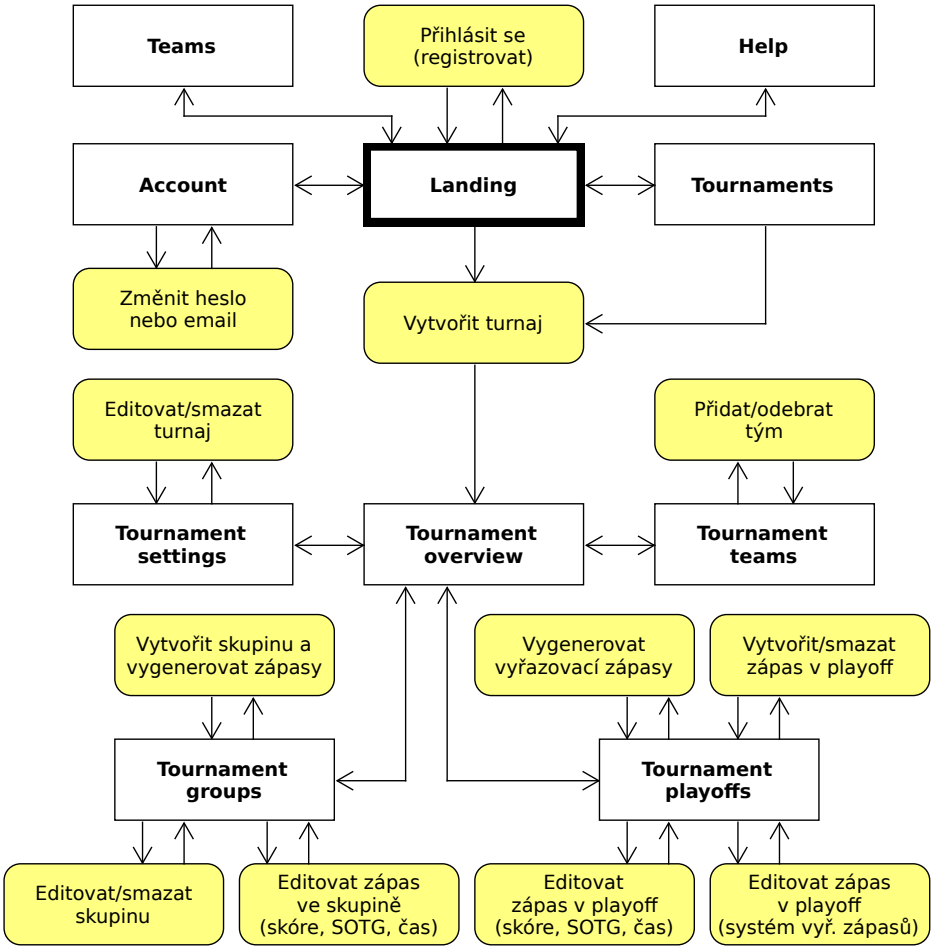
Tabulka 4.1: Tabulka nedostatků zjištěných při testování Hi-Fi prototypu



Obrázek 4.2: Zobrazení divize u týmů se stejným názvem před úpravou a po úpravě v Hi-Fi modelu. Druhá varianta odstranila situace, kdy si uživatel nevědomky vybral špatný tým.

4.1.3 Task graph

Návrh uživatelského rozhraní je vhodné doplnit tzv. task grafem, jenž nemá přímo danou notaci, ale i tak zřetelně zobrazuje jednotlivé obrazovky a aktivity spolu s přechody mezi nimi.



Obrázek 4.3: Task graph

4.2 Návrh backendu

Pri návrhu webové aplikace vycházíme ze tří různých modelů:

1. doménový model
2. datový model
3. use case diagramy

První model popisuje klíčové entity a jejich vazby. Jedná se o vysokoúrovňový pohled na celou problematiku návrhu, zatímco pomocí datového modelu pouze dořešíme nezbytné detaily. Diagram aktivit nám pomůže definovat některé akce, které nemusí být zcela jasné při použití pouze prvních dvou modelů. Všechny tři diagramy jsou tak v této diplomové práci zpracovány.

4.2.1 Doménový model

Z předchozích kapitol, zejména ze seznamu požadavků, můžeme získat základní představu o systému a jeho funkcionalitě. To nám pomůže vymodelovat jednotlivé entity a vztahy mezi nimi, jež se stanou součástí doménového modelu.

Uživatel

Uživatel je identifikován přihlašovacím jménem a heslem a zároveň vlastní e-mailovou adresu unikátní napříč systémem. Ta je ověřována pomocí potvrzovacího e-mailu, viz obrázek 4.4. Eventuálně disponuje rolí *administrátor*.

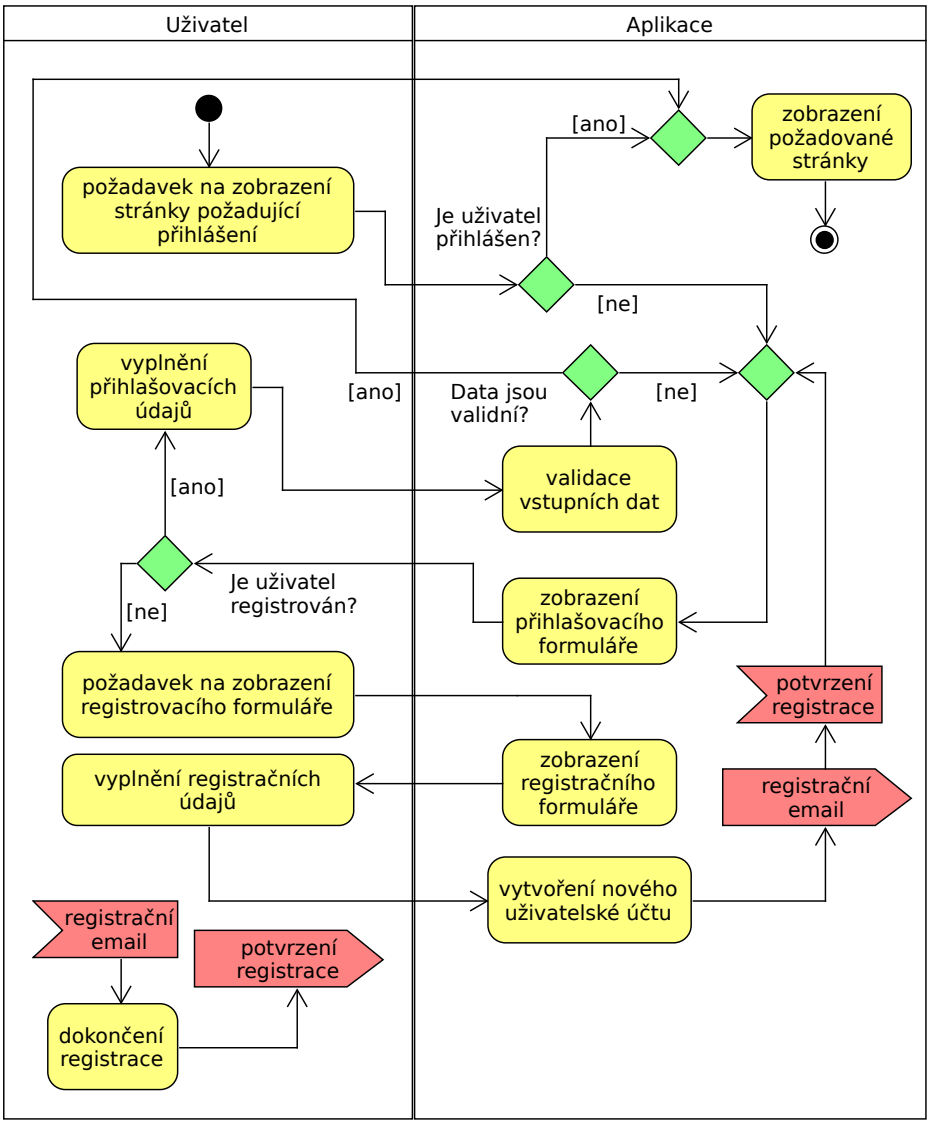
Divize

Jedná se o kategorii, ve které se pořádají turnaje nebo jsou složeny jednotlivé týmy. Příkladem může být ženský tým Prague Devils v divizi *women*. Nejběžnějšími divize jsou:

- open - bez omezení, ale nejčastější složení je pouze z mužů
- women - divize pouze pro ženy
- mixed - složení je v určitém poměru žen a mužů, které se může lišit turnaj od turnaje
- juniors - s omezením na horní hranici věku

Tým

Je reprezentován současně svým jménem a divizí, tzn. že dva týmy s identickým názvem ale odlišnou divizí jsou reprezentovány jako dva různé oddíly. Volitelnými atributy jsou město a země původu.



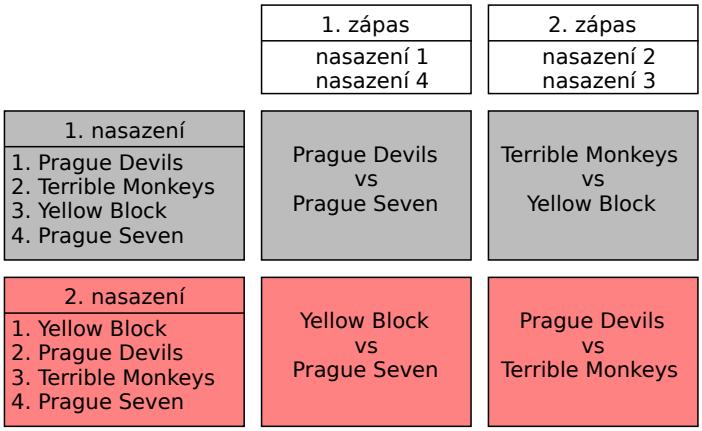
Obrázek 4.4: Diagram aktivit znázorňující přístup na stránku vyžadujicící příhlašení včetně procesu registrace.

Turnaj

Turnaj může být vytvořen libovolným uživatelem. Po vytvoření je automaticky nastaven jako *private*, tzn. že není ostatním uživatelům kromě administrátora viditelný (ten má stejná práva při správě turnaje jako jeho autor). V kterékoliv chvíli může správce turnaje odnastavit turnaji příznak *private* a zviditelnit ho

ostatním.

Všechny týmy, které se turnaje budou účastnit, se k turnaji přiřazují pomocí nasazení. To dovoluje měnit účastníky bez nutnosti opravovat všechny zápasy, ve kterých původní týmy byly nastaveny. Příkladem je obrázek 4.5.



Obrázek 4.5: Znázornění příkladu změny nasazení a jeho projevení ve dvou předem nastavených zápasech.

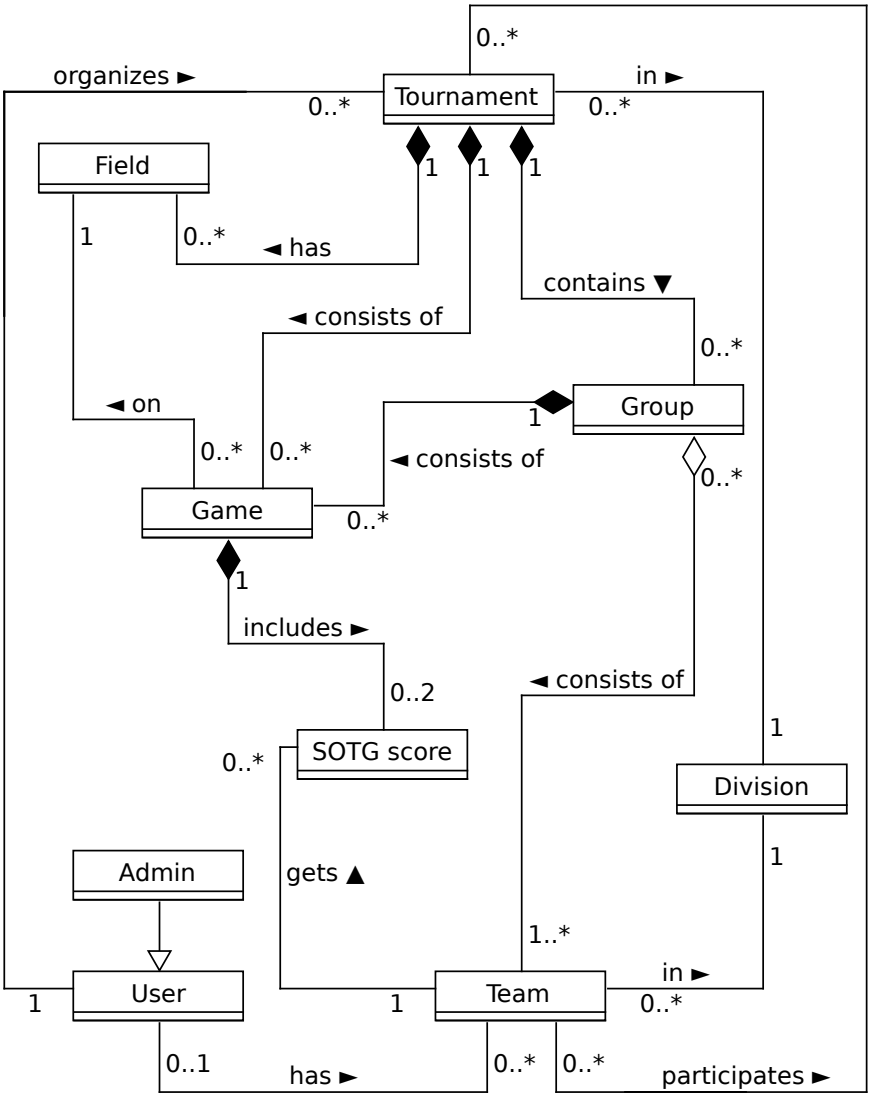
Na turnajích je také nutné vyplňovat hodnocení SOTG, které však nesmí být viditelné až do ukončení turnaje. Správce turnaje jej zveřejní nastavením příznaku *finished*, který označuje ukončení turnaje. U této změny nastavení musí proběhnout validace, zda aplikaci nechybí nějaká důležitá data, např. omylem zapomenutá hodnocení SOTG nebo výsledky zápasů.

Zápas

Existují celkem dva typy zápasů. Jeden ve vyřazovací části turnaje, kde domácí a hostující tým jsou doplňovány dynamicky na základě předchozích výsledků, a druhý ve skupinách, kde jeho zápasový výsledek je pouze klíčem k sestavení výsledného pořadí skupiny.

U vyřazovacích zápasů jsou tři volby pro nastavení domácího nebo hostujícího týmu:

- turnajové nasazení - tým je dosazen podle turnajového nasazení
- vítěz nebo poražený jiného vyřazovacího zápasu
- podle výsledného pořadí v dané skupině



Obrázek 4.6: Doménový model

Skupina

Jedná se o logický celek s vybranými týmy, které mezi sebou hrají libovolný počet zápasů. Za každou výhru a remízu získavají dva, respektive jeden bod. Po odehrání všech zápasů se určí pořadí na základě obdržených bodů a týmy jsou přiřazeny na další místa v turnaji, tzn. do vyřazovacích zápasů.

Následuje seznam pravidel pro určení pořadí ve skupině, pokud má více týmů stejný počet bodů. Vždy se uplatní nejvýše uvedené, pokud je to možné.

1. obdržené body ze vzájemných zápasů
2. rozdíl skóre ve vzájemných zápasech
3. rozdíl skóre ve všech zápasech
4. vyšší počet skórovaných bodů ve vzájemných zápasech
5. vyšší počet skórovaných bodů ve všech zápasech
6. náhodný výběr

Hřiště

Představuje místo, kde se budou hrát zápasy.

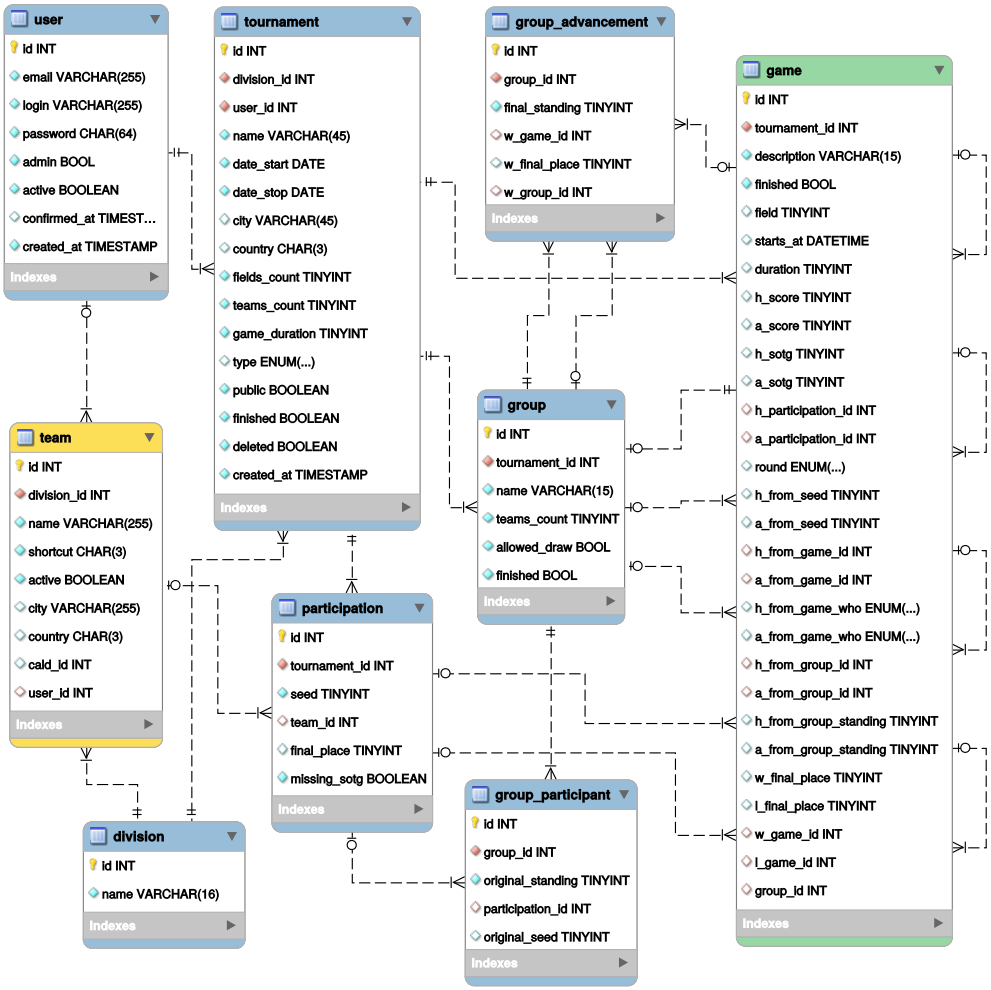
Hodnocení Spirit of the Game

Hodnocení SOTG se váže k zápasům, poněvadž po jeho skončení se oba soupeři ohodnotí v této kategorii na stupnici od 0 do 20 bodů. Hodnocení je viditelné pouze pro organizátora až do skončení turnaje, kdy je zveřejněno společně s pořadím týmů na základě průměrů hodnocení SOTG:

$$\frac{\text{suma obdržených bodů SOTG}}{\text{počet zápasů}}$$

4.2.2 Datový model

Doménový model nám pomohl ujasnit si jednotlivé entity a vztahy mezi nimi. Jeho hmatatelnou implementací je pak datový model. Kromě toho, že zahrnuje nové relační objekty včetně atributů, je také dekomponován, tzn. zbaven relací M:N.



Obrázek 4.7: Datový model

4.2.3 Use case diagramy

V návrhu této práce si vystačíme se seznamem případů užití a jejich diagramem, který je zpracován v sekci 2.3.

Kapitola 5

Implementace

5.1 Výběr technologií

Jeden z klíčových faktorů zajišťující úspěšné dokončení projektu je správný výběr technologií použitých k jeho vývoji. Špatná volba totiž může vést ke zbytečnému prodloužení vývoje, zvýšení nákladů nebo dokonce neúspěšnému ukončení.

Ať už se bude jednat o knihovny, frameworky či nástroje, je potřeba být ostražitý a pečlivě zvážit použití každé nové technologie. Pro odpovídající výběr je potřeba definovat náležitá kritéra, která by pomohla jasně porovnat všechny zvažované technologie.

Podpora webových služeb

Při vývoji webové aplikaci musíme zaměřit pozornost na technologie, které jsou vhodné pro použití na webu. Musí zvládat především HTTP komunikaci a práci s běžnými formáty používanými v internetové komunikaci.

Dokumentace a uživatelská podpora

Existence kvalitní a úplné dokumentace dokáže vývojáře seznámit s mnohými detaily, které jsou potřebné pro pochopení a správné používání vybrané technologie. Jestliže dokumentace není dostačující, může ji částečně nahradit široká komunita uživatelů, která bývá snižovatelem rizika, že zvolená technologie zastará.

Znalost technologie

Volba již dříve vyzkoušených technologií nám může ušetřit mnoho času, který bychom jinak potřebovali na složité zavádění nebo učení se jejich použití. Dále eliminuje riziko, že se narazí na neřešitelné problémy až během implementace.

Na druhou stranu nás může zavrhnutí neznámé technologie připravit o možnost použití nástroje, který je danému problému šitý na míru.

Výkon

I přesto, že úkolem není vyvinout vysokozátěžovou webovou aplikaci, není důvod používat pomalé nebo neefektivní technologie spotřebovávající velké množství prostředků

5.2 Zvolené technologie

5.2.1 Programovací jazyk Python

V roce 1991 Holanďan Guido van Rossum navrhl víceúčelový skriptovací jazyk Python [26], jenž je dnes vyvíjen jako open source projekt a bezplatně se nabízí jeho instalační balíky na všech velkých velkých platformách (Windows, Unix a Mac OS). Ve většině Linuxových distribucích je součástí základní instalace a protože pro Python existuje nespočet knihoven z různých oblastí, má rozsáhlé uplatnění. Používá se ve firmách jako Facebook, Google, Cisco, YouTube, Red Hat, Dropbox nebo Microsoft [27].

Díky tomu, že Python je často označován jako skriptovací jazyk, mnoho vývojářů netuší, že jeho návrh umožňuje implementaci rozsáhlých a plnohodnotných aplikací. Dokonce umožňuje v omezené míře používat i funkcionální programovací paradigma [28] vedle objektově orientovaného, procedurálního apod. Python je ale dynamicky interpretovaný jazyk, tudíž je třeba kalkulovat s tím, že jde o obecně pomalejší jazyk, než jazyky kompilované. Výkon Pythonu je však doháněn tím, že se části jeho knihoven implementují v jazyce C [31].

Python má díky své jednoduché syntaxi a čistému kódu krátkou učící křivku a je pokládán za jeden z nejvhodnějších programovacích jazyků pro výuku začátečníků. V porovnání s ostatními programovacími jazyky je jeho kód mnohdy kratší a lépe čitelný – hodí se tak i pro použití v praxi. PYPL index[9] jej řadí na první místo žebříčku celosvětově nejrychleji roustoucích programovacích jazyků co do oblíbenosti za posledních pět let [29]. Podle žebříčků trendů Stack Overflow je Python na druhém místě za JavaScriptem [30].

Python 2.x a 3.x

Uživatelé Pythonu mohou dnes používat dvě nekompatibilní verze 2.x a 3.x, kde 3.x odstraňuje velké množství nedostatků a chybných návrhů z předchozí verze. I přesto, že vývoj Pythonu 2.x se nikdy zcela nezastavil a přechod na novou verzi dodnes není ukončen v mnoha projektech, byl v roce 2017 oznámen konec jeho podpory, který začne platit v roce 2020.

[9] PopularitY of Programming Language Index je řazen zkoumáním toho, jak často jsou návody a tutoriály jednotlivých program. jazyků hledány na Googlu.

5.2.2 Webový framework Flask

V oblasti Python webových frameworků pravděpodobně nejčastěji narazíme na Django [32], jež je plnohodnotným nástrojem obsahujícím webový server pro testování, serializační a validační systém pro formuláře, vlastní ORM apod. Pro některé vývojáře se však jedná o „kanón na vrabce“ a pro ně je tu minimalistický webový framework napsaný v Pythonu – Flask [33].

Obrázek 5.1: Logo Flask frameworku [33]

Flask je postavený na knihovně Werkzeug [34] a šablonovacím systému Jinja2 [35]. Nedisponuje žádnou databázovou abstraktní vrstvou, validací formulářů atd. Na druhou stranu umožňuje programátorům doplnit chybějící funkcionalitu dalšími knihovnami, které se pro jejich případ použití zdají nejvhodnější. V základu framework obsahuje [33]:

- vestavěný vývojový server a debugger
- integraci pro podporu unit testování
- odbavování RESTful požadavků
- Jinja2 šablonování
- podporu pro zabezpečení cookies

Kromě toho je Flask v souladu s WSGI 1.0 [36], založen na unicode a rozsáhle zdokumentován narozdíl od jiných webových frameworků.

Co je to WSGI

Jedná se o skromnou a univerzální specifikaci rozhraní, pomocí které komunikuje webová aplikace (framework) v Pythonu a webový server. Dnes je vyžadovaným standardem u Python aplikací.

Flask příklad

Na následujícím příkladu použití Flasku je zobrazena funkce, doplněná o komentáře, která je volána v případě HTTP GET nebo POST požadavku na adresu `/register`.

```
import catcher as c
from flask import request, flash, render_template, redirect, url_for
...

@user.route('/register', methods=['GET', 'POST'])
def register():

    if request.method == 'GET':
        """
        Pokud se jedná o GET požadavek, pouze
        zobraz šablonu se vstupním parametrem title
        """
        return render_template('register.html', title='Register')

    """
    Pokud se jedná o POST požadavek, zkus vytvořit uživatele
    """
    try:
        user = c.User.create(**request.form.to_dict())
    except ValueError:
        """
        V případě chyby zobraz chybovou hlášku a vykresli šablonu
        """
        flash('Registration failed', 'danger')
        return render_template('register.html', title='Register')

    """
    Odešli potvrzovací email
    """
    send_confirm_email(user.email)

    """
    Zobraz hlášku o úspěšném provedení a přesměruj uživatele na /home
    """
    flash('A confirmation email has been sent via email.', 'success')
    return redirect(url_for("web.home"))
```

5.2.3 Databáze MySQL

Populární relační databáze, jež dosahuje poměrně vysokého výkonu, snadno se používá a jde o volně šiřitelný software (k dispozici je i komerční licence). MySQL [37] je součástí populární kombinace LAMP[10] – jedná se základní sestavu softwaru na mnoha serverech. Pro komunikaci s databází se používá jazyk SQL [38].

V případě správy databázového serveru je nutné použít některého z klientů. Nejčastěji se používá terminálový klient, avšak existují i pokročilejší nástroje, které umožňují například modelování databáze. Jedním z těchto nástrojů je MySQL Workbench [39]. Ten byl použit při návrhu databázového modelu této diplomové práce.

Databáze vznikla v roce 1995 [40] a během jejího vývoje vzniklo několik alternativních větví, tzv. forků – např. MariaDB [41] nebo Percona [42]

5.2.4 SQLAlchemy

Jedná se o open source soubor SQL nástrojů a ORM[11] vydaný pod MIT licencí pro programovací jazyk Python [43]. Podporuje databáze PostgreSQL [44], MySQL, SQLite [45] a další. První release[12] pochází z roku 2006 [46] a od té doby patří společně s Django ORM [47] mezi nejpopulárnější ORM nástroje v Python komunitě. Na následujících řádcích velmi zjednodušené použití:

```
session = Session()

class Tournament(Base):
    """
    Instance objektu reprezentuje jednu řádku
    tabulky 'tournament' v~databázi
    """
    id = Column(Integer(), primary_key=True)
    division_id = Column(Integer(), ForeignKey('division.id'))
    ...
    name = Column(String(255))

"""Získá tournaj s~id=5"""
tournament = Tournament.get(5)
"""Vytiskne na výstup do terminálu jméno turnaje"""
print(tournament.name)
"""Smaže turnaj"""
session.delete(tournament)
```

[10] Linux, Apache, MySQL, PHP
[11] Object-relational mapping
[12] jedna nebo více změn IT služby které jsou nasazeny najednou

Co je to ORM

Pokud chceme s daty v relačních tabulkách pracovat jako s objekty v OOP[13], je možné využít programovací techniku ORM, která data konvertuje do perzistentních objektů, se kterými programátor pracuje přímo a nemusí psát žádné SQL dotazy. Některé vyspělejší implementace ORM dokonce podporují využívání objektové dědičnosti, což relační databáze nepodporují.

Použití techniky ORM však přináší několik nevýhod, za které je často kritizována. Nejprve odnaučuje vývojáře psaní SQL dotazů a pak přináší zbytečnou režii spojenou s neefektivním překladem dotazů nebo vytvářením objektů i tam, kde to není vhodné. Příkladem může být aplikace vystavující REST API, která by s použitím ORM nejprve objekt deserializovala z databáze do objektu, aby o chvilku později stejná data serializovala do JSON[14] objektu.

5.2.5 Verzovací systém Git a služba GitLab

Linus Benedict Torvalds[15] je autorem nástroje Git [60], jenž je dnes populárním řešením pro vývoj v týmech a nejen to. Git se totiž stará o verzování libovolných digitálních informací, zejména zdrojových kódů, a zároveň napomáhá tomu, aby velké množství programátorů mohlo pracovat na jednom projektu. Dosaženo je to tím, že programátoři mají adresář s pojektem na vlastním počítači a všechny změny sdílí s centrálním repozitářem, k němuž přistupují i ostatní. Každý tak může pracovat se změnami, které provedli ostatní členové týmu.

Kromě toho, že se můžeme o hosting centrálního repozitáře starat sami, je tu ještě možnost využít již fungujících služeb: GitHub [49] nebo jiné jemu podobné. Tyto služby kromě hostování repozitáře a grafického rozhraní poskytují mnoho dalších funkcionalit, jako například možnost diskutovat nad zdrojovým kódem nebo zasílání notifikací o změnách. Kromě GitHubu, který funguje od roku 2008 [50] a je zdarma pro open source projekty, je velice populární i GitLab [51] s podobnými vlastnostmi. Ten ale jde ještě o krok dál a umožňuje aby byl jeho software nasazen i na serverech třetích stran. Tzn. že vývojáři si mohou repozitáře hostovat na své infrastruktuře.

Pro svoji práci jsem si vybral GitLab, protože poskytuje velice komplexní průběžnou integraci (CI), která mi výrazně pomáhala při nasazování a automatickém testování. O tom více v kapitole 7.

[13] Object-oriented programming
[14] JavaScript Object Notation
[15] tvůrce Linuxového jádra

5.3 Zvolená architektura

V případě vývoje malých webových aplikací je volba architektury něčím, co nemusí hrát velkou roli. U větších projektů ale nevhodný výběr nebo dokonce absence konceptu vývoje může vést k fatálnímu selhání v dokončení projektu.

Kdo se někdy podílel na vývoji rozsáhlejších webových aplikací, musel se setkat s pojmem MVC (popsáno v sekci 3.2.3). Ve velkém počtu Python webových frameworků se však MVC transformovalo do MTV (model, template, view):

Template Obsahuje HTML a logiku prezentace dat. Používá šablonovací jazyky, například populární Jinja2 u Flasku. Získává data z view a generuje výslednou webovou stránku.

View Někdy nazýván Controllerem. Je psán v Pythonu, využívá webový framework a stará se o spojení všech vrstev do fungujícího celku. Definuje URL adresy a mapuje na ně funkce nebo metody, které pracují s ostatními vrstvami tak, aby byly schopné na požadavek odpovědět. View by mělo být malé, protože jeho kód je velmi závislý na použitém frameworku.

Model Perzistentní vrstva, která je v mnoha projektech závislá na SQLAlchemy nebo jiném ORM nástroji. Model zná podobu dat v databázi a ví, jak je ukládat. Nepotřebuje vědět nic o webovém frameworku nebo HTTP komunikaci.

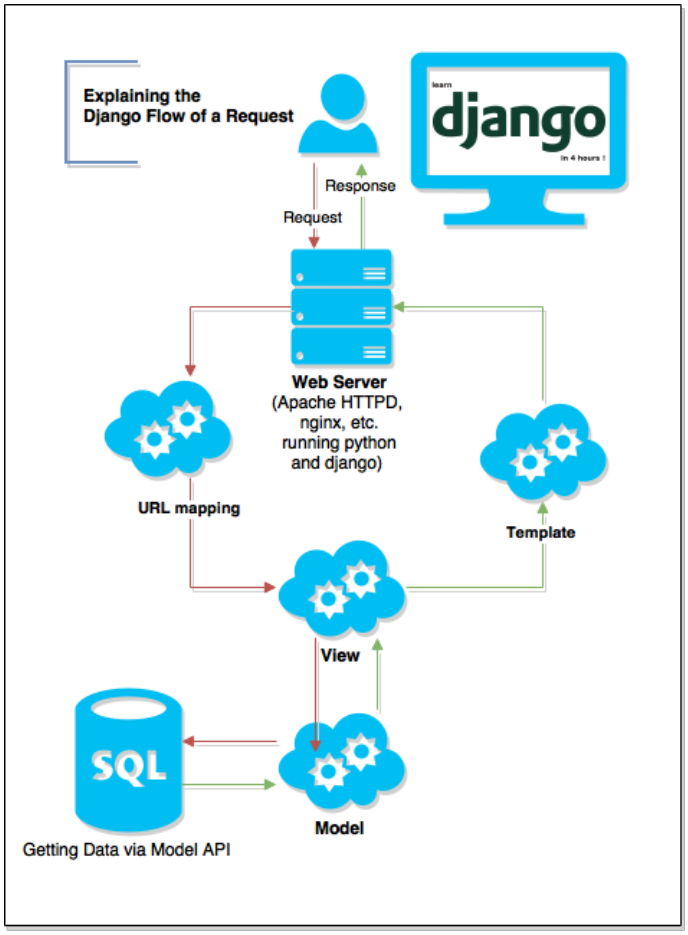
Problém ale nastává, pokud potřebujeme do projektu vložit nějakou složitější business logiku. Protože vrstva šablon se nepíše v Pythonu, zbývají tři možnosti kam ji vložit:

1. Vložení do **View** je nejhorší varianta, protože tím zamezíme izolování webového frameworku od aplikační logiky. Špatně se testuje a například v případě budoucího rozšíření aplikace o API je možné, že stejná aplikační logika se bude nacházet na více místech.

2. Preferovanější volbou je **Model**, protože jde o vícekrát použitelnou vrstvu. Ten by se ale opravdu měl starat pouze o perzistenci dat a ne o aplikační logiku.

3. Poslední možností je vytvořit **novou vrstvu**.

Koncept *Model-Template-View* stačí pro vytvoření jednoduchého webu, například blogu. Pokud ale budeme vytvářet komplexní systém, je potřeba přidat novou a opakovatelně použitelnou vrstvu. V této diplomové práci jsem ji pojmenoval názvem *Catcher*, ale lepší a obecnější pojmenování by pravděpodobně bylo *Service* apod.



Obrázek 5.2: Schéma vyřízení requestu v *Model-Template-View* (Django) [52]

Nová vrstva

Často se může stát, že jedna akce uživatele vyžaduje více činností v systému. Ukázkou může být registrace uživatele, která kromě vytvoření dat ve více databázových tabulkách vede taky k tomu, že odešle uživateli email atd. Pokud je uživatelova akce registrace popsána v nějaké funkci, v jaké vrstvě by tato funkce měla být?

Pokud by to bylo v Modelu, bylo by to velmi složité s importy. Kromě toho, že by tato vrstva obsahovala spoustu importů na externí služby (například odesílání emailu), hrozilo by také, že jednotlivé modely (třídy) by se potřebovaly importovat navzájem. Docházelo by tak k cyklické závislosti a to je jasným znamením, že je něco špatně.

Je důležité si také uvědomit, že Model obvykle mapuje třídy na tabulky v relační databázi. To ale nemusí odpovídat rozvržení logických entit v aplikaci. Pokud by tak měla jedna funkce pracovat s více modely zároveň, není

vůbec jasné, do kterého modelu by měla být vložena.

Dalším důvodem, proč je vhodné oddělit modely od aplikační logiky je to, že se tím ulehčí testování. Není totiž žádoucí nastavovat třídu pro posílání emailů nebo jiné externí služby, když chceme otestovat jen základní funcionalitu modelů.

Je však důležité zmínit, že není cílem odstranit veškerý kód z modelů. V modelech by měly zůstat pouze metody, které pracují s jejich daty a nevyužívají externí služby. Jinak by totiž návrh spěl k tzv. anemickému doménovému modelu[16].

5.4 Adresářová struktura

Všechny zdrojové kódy aplikace se nachází ve složce `/src`, která je přehledně rozdělena po jednotlivých vrstvách architektury. Soubory, které se týkají Docker kontejnerů budou vysvětleny v kapitole 7.

```
/
├── /mysql/fixtures ..... SQL skripty pro testovací databázi
├── /src
│   ├── /api ..... REST API pro budoucí rozšíření
│   ├── /catcher ..... business logika aplikace
│   ├── /models ..... mapování relační databáze na OOP
│   ├── /web ..... zdrojové kódy týkající se webového frameworku Flask
│   │   ├── /static ..... statické soubory (JS, CSS ...)
│   │   ├── /templates ..... Jinja2 šablony
│   │   └── /web.py ..... vrstva View
│   └── /wsgi.py ..... spouštěcí skript
├── /tests
│   ├── /catcher ..... testy business logiky
│   ├── /web ..... testy webového frameworku
│   └── /conftest.py ..... konfigurační soubor pro testy
├── /gitlab-ci.yml ..... konfigurační soubor pro CI
├── /Dockerfile ..... definice Docker kontejneru, ve kterém Catcher běží
├── /docker-compose.prod.yml ..... nastavení kontejnerů na produkci
├── /docker-compose.yml ..... nastavení kontejnerů na vývoji
└── /requirements.txt ..... seznam závislostí
```

Obrázek 5.3: Struktura webové aplikace.

[16] jeden z návrhových antivzorů

5.5 Bezpečnost

Dostat se do stavu, kdy můžeme o webové aplikaci prohlásit, že je zcela bezpečená, je takřka nemožné. Útočníci jsou dnes stále vynalézavější a bezpečnostní prvky, které před několika lety mohly být dostačující, dnes nemusí být vůbec vhodné. Nejlepší obranou je tak kombinace více ochranných prvků a dodržování zásad, které možnosti útočníků minimalizují.

Zabezpečená komunikace

Základním nástrojem k zabezpečení aplikace je použití protokolu HTTPS[17] využívající další protokoly HTTP, SSL[18] nebo TLS[19]. Výsledkem je důvěra v přenášená data a jejich integritu. Jedním z dalších důvodů k přechodu z HTTP na HTTPS může být to, že nejpoužívanější webový prohlížeč Chrome [53] bude od července 2018 označovat všechny weby s HTTP jako „nezabezpečené“ [54]. Tato práce používá pouze protokol HTTP, ale pro ostré nasazení se počítá s HTTPS.

Autentizace

Proces, jenž ověří identitu uživatele. O to se stará webový framework Flask pomocí ukládání uživatelova ID do session. Protože HTTP je ze svého principu bezstavový protokol, díky session lze získat kontext o uživatelích.

Autorizace

Autorizace je procesem, který schvaluje konkrétní uživatele k provádění vybranných akcí. Vše se děje na základě rolí, které jsou jim přiřazeny. Přihlášený uživatel může vytvořit turnaj a editovat jej, avšak cizí turnaj je pro něj viditelný stejně jako pro nepřihlášeného uživatele. To je docíleno tím, že turnaj si uchovává informaci o autorovi – uživateli, který jej vytvořil. Uživatel s rolí administrátor může provádět všechny možné operace v systému.

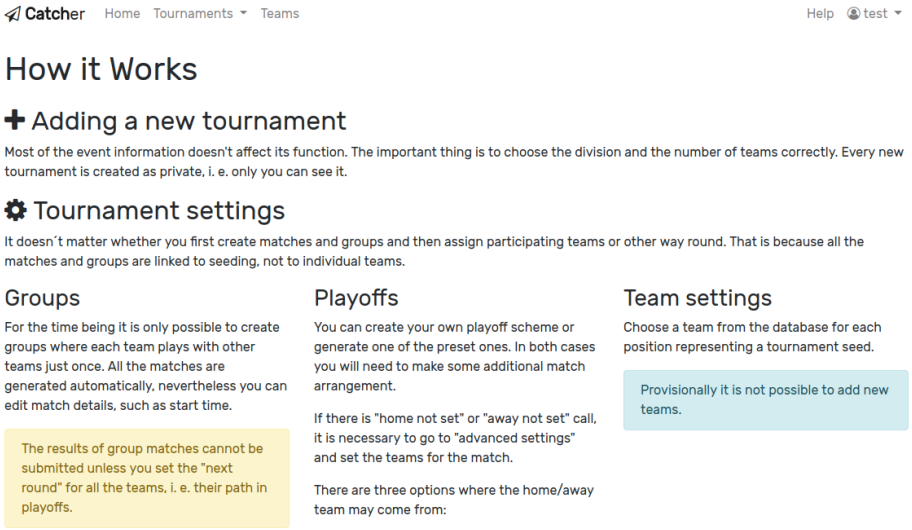
Šifrování dat v databázi

Jedním z rizik je únik dat, který není v IT světě výjimečnou událostí. Proti zneužití je vhodné šifrovat všechna citlivá data v databázi (například hesla).

[17] Hypertext Transfer Protocol Secure
[18] Secure Sockets Layer
[19] Transport Layer Security

5.6 Manuál pro uživatele

Protože některé akce v aplikaci nemusí být jasné každému při prvním použití, byl vytvořen manuál, kde jsou některé klíčové prvky systému popsány podrobněji.



Obrázek 5.4: Screenshot manuálu pro uživatele.

Kapitola 6

Testování

Jedním z dílčích cílů diplomové práce je navrhnout a provést vhodnou sadu aplikačních a uživatelských testů na finální aplikaci.

6.1 Aplikační testy

Testování obecně je jednou z velmi opomíjených součástí vývoje nového i údržby „starého“ kódu. Napomáhá zajistit, že výsledná aplikace neobsahuje chyby, splňuje požadované vlastnosti a obecně zvyšuje kvalitu kódu. Provádí se nejčastěji při změnách nebo nasazování nových verzí softwaru, aby se zamezilo chybám v produkčních prostředích. Jedním z prvků testování je pak především hlášení o nalezených chybách.

Manuální a automatizované testy

Testy se dělí na manuální a automatizované. Manuální testování nejde zcela nikdy nahradit automatizovaným, protože některé testy vyžadují rozdílné přístupy, lidský úsudek nebo je není nutné provádět opakovaně. Pokud ale velké množství manuálních testů probíhá opakovaně na stabilních testovacích případech, je možné uvažovat o jejich zautomatizování. Automatizované testy s sebou přináší zefektivnění a usnadnění testování softwaru, tzn. časovou úsporu a zlevnění testování. Výhodou automatizovaného testování je taky skutečnost, že stejné případy můžeme opakovat s velkým množstvím různých vstupních dat.

Blackbox, whitebox a graybox testování

Z hlediska terminologie testování se ještě uvádí dělení na tzv. whitebox a blackbox testy. Pro analýzu z pohledu uživatele se používá blackbox, protože při tomto testování se kontroluje pouze správnost výstupních dat na základě dat vstupních. O vnitřní podobě nemáme žádné informace narozdíl od testování

whiteboxů, které používáme při znalosti vnitřní implementace. To nám lépe pomáhá identifikovat místo chyby. Někde mezi je graybox testování, které nemusí znát přesnou vnitřní podobu, ale například pouze použitý algoritmus.

Mockování

Dalším zajímavým pojmem v testování je tzv. mockování. Jde o techniku, která pomáhá nahradit reálné objekty pouze testovací fasádou, která se chová podle našich požadavků. Pokud tedy v testovaných částech kódu potřebujeme volat další metody nebo funkce a ty nejsou předmětem testování, jednoduše jejich chování změníme. Při testech nad daty z databáze můžete například mockovat výstup z databáze. Pro testování tak není potřeba spouštět i databázový server, ale pouze předefinovat metodu, která data vrací.

6.1.1 Testování v Pythonu

Nejrozšířenějším frameworkem pro testování kódu v Pythonu je Unittest [56], který je inspirován frameworkem JUnit [57] pro jednotkové testy psané v Javě.

Unittest

Knihovna `unittest` pro účely vytvoření testovacího prostředí používá metody `setUp()` a `tearDown()`, které jsou volány před, respektive po každém testu. To pomáhá zajišťovat nezávislé chování, protože metody umožňují nastavit prostředí před testem a po testu. A právě v metodě po testu zpravidla probíhá čištění prostředí, například databáze, tak aby tento test neovlivňoval následující. Obecně je dobré se vyhnout tomu, aby test měl možnost být ovlivněn „zvenku“, protože by to mohlo způsobit nesprávné vyhodnocení výsledku testu jako neúspěšné, nebo v horším případě jako úspěšné. Takto může vypadat zjednodušená podoba testů napsaných za pomoci knihovny `unittest`:

```
import unittest
from db import configure_session

class TestClass(unittest.TestCase):

    def setUp(cls):
        """
        Volá se před každým testem. Může se zde
        například nainicializovat spojení
        s databází a naplnit ji testovacími daty.
        """
        cls.session = configure_session()

    def testGet(cls):
        """
        Testuje návratové hodnoty z databáze
        """
        object = cls.session.get_object(5)
        cls.assertEqual(object.id, 5)

    def tearDown(cls):
        """
        Volá se po každém testu. Zpravidla
        vyčistí databázi a uzavře spojení.
        """
        cls.session.clear_db()
```

Pytest

Kromě Unittestu je k dispozici ještě jeden poměrně propracovaný framework pro automatizované testování – pytest [58]. V posledních letech získává na oblibě především protože zjednodušuje psaní testů na úplné minimum, a zároveň přináší více funkcionality než Unittest. Opět velmi zjednodušeně představím podobu připojení do databáze, tentokrát ale pomocí pytestu.

```
import pytest
from db import configure_session

@pytest.yield_fixture(scope='function')
def db():
    """
    První část funkce se provede ještě před spuštěním testu.
    Příkaz yield testu předá objekt session a po testu funkce
    pokračuje za příkazem yield, takže může uklidit databázi.
    """
    session = configure_session
    yield session
    session.clear_db

def test_get(db):
    """
    Testuje návratové hodnoty z~databáze
    """
    object = db.get_object(5)
    assert object.id == 5
```

6.1.2 Moje řešení

I přesto, že Unittest je součástí standardní knihovny, vybral jsem si pro testování této diplomové práce pytest. V kombinaci s knihovnou flexmock [59], která zajišťuje mockování objektů, bylo napsáno 56 testů.

Testování probíhalo automaticky při každé změně v Git [60] repozitáři v rámci CI[20]. O tom více v kapitole 7.

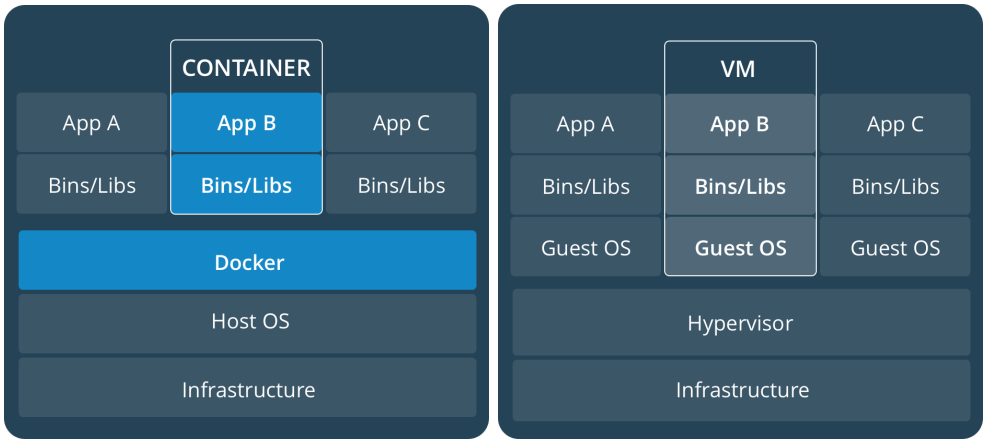
Pro lokální testování, tzn. testování při vývoji, byly testy upraveny tak, aby celý proces předcházelo spuštění testovací databáze v Docker [61] kontejneru.

Co je Docker

Docker je open source projekt, který umožňuje odhlehčenou virtualizaci. Izoluje jednotlivé aplikace nebo služby do kontejnerů, a k tomu poskytuje jednotné rozhraní. Výhodou Dockeru je to, že kontejner obsahuje pouze vybrané

[20] Průběžná integrace, angl. Continuous Integration

aplikace bez virtualizovaného operačního systému. Je tím dosažena menší velikost a větší flexibilita. Docker staví na virtualizačních a izolačních funkcích dostupných v Linux jádře [62].



Obrázek 6.1: Vizualizace rozdílů mezi virtualizací Docker kontejnerů a běžného virtualizováného stroje. Z obrázku je zřejmé, že kontejnery jsou „odlehčeny“ od virtualizovaného operačního systému. Výsledkem jsou tak kontejnery, které často zabírají pouze desítky MB. [63]

6.2 Uživatelské testy

Kromě toho, že uživatelské testování často pomáhá při návrhu uživatelského rozhraní, může i odhalit chyby aplikace, na které autoři aplikace do té doby nepřišli. Bylo tomu tak i v této diplomové práci.

Pro testování byly z velké části vybráni a požádáni uživatelé, kteří budou finální aplikaci používat. Ať již jako aktivní (správci turnajů) nebo pasivní (diváci, členové a hráči týmů) uživatelé.

Jediným podstatným rozdílem oproti reálnému použití bylo to, že uživatel dělal všechny testovací úkony při jednom sezení. Při běžném použití by totiž uživatel turnaj vytvořil několik dní před samotnou akcí a práci s výsledky by prováděl až na místě. Je tu tak riziko, že uživatel se bude s aplikací seznamovat dvakrát – jednou při vytváření a jednou při zadávání výsledků. Do budoucna se však počítá s dalším testováním přímo na turnajích.

6.2.1 Změny provedné po zahájení testování

Už při prvním testování se objevila závažná chyba v editaci zápasu playoff, kterou bylo potřeba opravit, jinak by mohla ovlivňovat další testované subjekty. Aplikace měla kontrolovat, zda se konkrétní hodnota u pole s nastave-

ním domácího a hostujícího týmu nevyskytuje již u ostatních zápasů. Bohužel kontrolovala i zrovna editovaný zápas, kde se hodnota již logicky vyskytovala, pokud uživatel editoval zápas vícekrát než jednou. Aplikace tak odmítla uživatelova data, i když je měla přijmout. Byla tak výrazně omezena funkcionalita aplikace, která byla pro následující testování stěžejní. Tento nedostatek byl po prvním testu opraven.

6.2.2 Testovací scénáře

Poněvadž vytváření turnaje je proces, který může být poměrně dlouhý, byly vytvořeny dva testovací scénáře – jeden náročnější zahrnující vytvoření turnaje podle předlohy a druhý jednodušší, jehož cílem bylo zjistit, jak se uživatel v aplikaci orientuje. Z časových důvodů jsme tak mohli testování uskutečnit u více uživatelů, protože některým z nich jsme zadávali pouze druhý scénář. Předlohami prvního scénáře se staly reálné turnaje v minulosti, ke kterým se evidují výsledky zápasů. Mohli jsme tak porovnávat, zda se aplikace chová očekávaně.

6.2.3 Uživatelské testování

Celkem se testování zúčastnilo 7 osob, kteří mají přímou zkušenost se sportem Ultimate Frisbee. Všichni mají vysokoškolské vzdělání nebo jsou studenty VŠ. Testování probíhalo na mobilních telefonech a noteboocích.

Většinu uživatelů nebylo třeba seznamovat s problematikou. Dostali jednoduché instrukce a byli sledováni, jak s aplikací pracují. Během této práce byli občas dotázáni, zda rozumí konkrétním prvkům nebo jaký mají dojem z konkrétní interakce. Testování se účastnili:

- Karolína Zubatá, hráč a příležitostný organizátor turnajů
- David Viktora, hráč
- Michal Kváček, student FIT ČVUT
- Dalimil Fišar, hráč (pouze druhý scénář)
- Pavel Milička, hráč (pouze druhý scénář)
- Adam Ogurcak, hráč (pouze druhý scénář)
- Vojtěch Fišer, hráč (pouze druhý scénář)

6.2.4 Výsledky testování

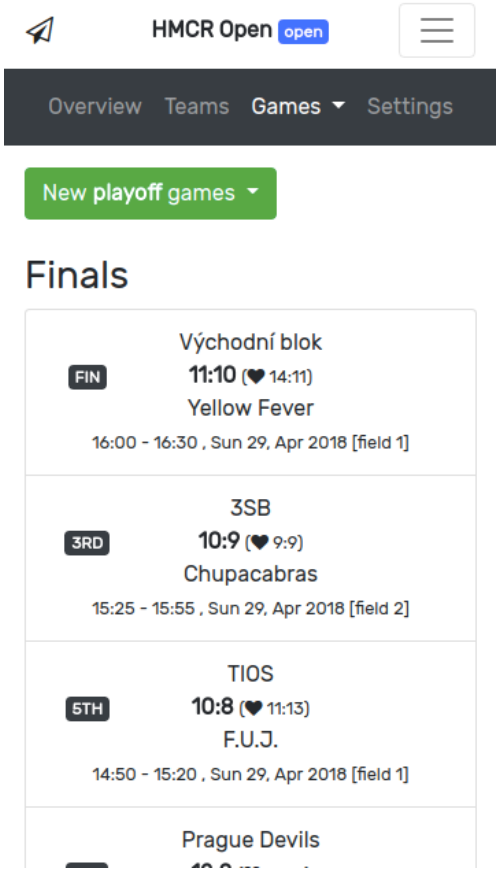
Z výše uvedených testů vznikl seznam zjištěných problémů a připomínek společně s jejich následným řešením nebo návrhem na něj:

Nedostatek nebo připomínka	**Návrhy na opravu**
Problém řešený v sekci 6.2.1.	Oprava aplikace již proběhla.
Tabulky a jejich řádky na stránce s přehledem turnaje (výsledné umístění týmů nebo pořadí v hodnocení SOTG) se zabarvují při najetí myši na řádek. Kliknutí na něj však neprovádí žádnou akci.	Nevyvolávat dojem, že uživatel může provést nějakou akci, pokud ji provést nemůže. Tabulky nechat imunní vůči pohybům uživatele.
Tabulka s výsledným pořadím obsahuje číslo v závorce za názvem týmu. Jde o nasazení týmu na turnaji, bylo však chybně zaměňováno za pořadí na turnaji.	Odstranit údaj z tabulky, protože na tomto místě není potřeba znát původní nasazení.
Na mobilních telefonech se zdá být seznam zápasů a jejich výsledky mírně nepřehledné, protože všechna data jsou v jednom sloupci.	Zjednodušit seznam zápasů tak, aby byly údaje spíše více do šířky než do výšky.
Aplikace odesílá link na resetování hesla i těm emailovým adresám, které nemá v databázi. Po kliknutí na link stránka vrátí HTTP status kód 404.	Jde o implementační chybu, která musí být opravena. Neregistrovaným uživatelům se nebudou posílat žádné emaily.
Uživatelé si často nevěděli rady s nastavením domácího a hostujícího týmu v zápasech playoff, kde bylo potřeba nastavení u dvou zápasů doslova prohodit. V obou případech byla totiž druhá hodnota již zadaná a systém vracel chybu, protože tím brání chybnému nastavení vyřazovacích zápasů.	Přidat možnost odebrat nastavení domácího nebo hostujícího týmu tak, aby se hodnota uvolnila pro nastavení druhého zápasu.
Akce vyvolávající posílání e-mailů trvají déle než ostatní a na okamžik se může zdát, že aplikace nic nedělá. Jde o porušení prvního pravidla Nielsenovy heuristické analýzy.	Zobrazit progress bar nebo implementovat posílání emailů asynchronně.

Tabulka 6.1: Tabulka nedostatků zjištěných při testování

Testování na mobilních telefonech prokázalo předpoklady návrhu. A tedy to, že pro prohlížení výsledků je uživatelské rozhraní naprosto dostačující. Pro vytváření turnaje je však pohodlnější použít stolní počítač nebo notebook. Uživatelé na telefonech však nenarazili na žádné omezení, které by se týkalo

rozdílů funkčnosti na mobilech a počítačích.



Obrázek 6.2: Zobrazení aplikace na mobilních telefonech.

Na úplném konci testování zazněly jednoduché otázky o tom, zda by účastníci testování použili aplikaci na správu turnaje, který by sami pořádali. Výsledkem bylo, že všech 7 respondentů odpovědělo kladně. Obecně taky panoval konsensus o tom, že aplikace vypadá dobře a příjemně se s ní pracuje.

Kapitola 7

Nasazení

K testování a vývoji mi byl zapůjčen jeden virtuální privátní server, ke kterému jsem přistupoval pomocí SSH[21]. Adresa, na které běží výsledná aplikace, je *www.catcher.zlutazimnice.cz*.

Virtuální privátní server

Jedná se o server, který běží na virtualizovaném hardwaru, označován zkratkou VPS. Jeden fyzický stroj je poskytován více uživatelům najednou tím, že každý z nich získá vymezený prostor s vlastní instancí OS, hostingovými službami, vlastní konfigurací apod. Je to běžný způsob hostování aplikací, protože pořízení VPS je výrazně levnější než získání fyzického serveru. Nevýhodou však může být to, že uživatelé musí sdílet stejné fyzické prostředky.

7.1 Softwarové požadavky

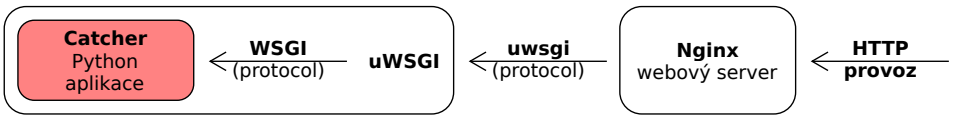
Webová aplikace běží na virtuálním privátním serveru s operačním systémem Ubuntu verze 16.04.3 LTS [64]. Je dostupná kterémukoliv zařízení s přístupem k internetu, protože běží na veřejné adrese (a portu 80). Požadavky vyřizuje vývojový Flask server, který je nutné před nasazením do ostrého provozu nahradit plnohodnotným webovým serverem, např. Nginxem [10]. O komunikaci mezi webovým serverem a aplikací se stará uWSGI [65]. O data se stará databázový server MySQL 5.7.22.

Aby aplikace mohla fungovat, bylo nutné stáhnout všechny potřebné závislosti pomocí pipu [66], jenž slouží pro správu nesystémových knihoven v Pythonu. Jejich výčet je uveden v souboru `/requirements.txt` v root adresáři projektu.

[21] Secure Shell

7.1.1 Co je uWSGI

Pro webové aplikace v Pythonu je dnes uWSGI nutností. Jedná se o malý program, jenž se stará o její chod. Bývá prostředníkem mezi aplikací a webovým serverem, se kterými komunikuje pomocí protokolů uwsgi a WSGI[22].



Obrázek 7.1: Komunikace mezi webovým serverem a aplikací v Pythonu [5]

7.2 Použití Dockeru

Protože tato práce používá Docker, je nutné si představit, jak funguje vytváření Docker kontejnerů a jak se používají, pokud je nutné jich pustit několik.

7.2.1 Vlastní kontejner v Dockeru

Jestliže potřebujeme rychle a jednoduše spustit databázový server MySQL v Dockeru, tak jednoduše zavoláme následující příkaz:

```
$ docker pull mysql && docker run mysql
```

Kromě databází lze podobně spustit mnoho dalších kontejnerů (webové servery, Linuxové distribuce, aplikace ...). Pokud ale chceme vytvořit vlastní kontejner, je potřeba jej definovat v jednom souboru, nejčastěji v `Dockerfile`, a sestavit jej (`docker build`).

Při vytváření kontejnerů je nutné zvolit výchozí image, ke kterému se pak přidává to, co je zrovna potřeba. Já jsem zvolil Alpine Linux [71], což je velmi miniaturní distribuce, která neobsahuje žádné nadbytečné balíky. Výsledkem je pak složený image, jehož velikost je pouze několik desítek MB. Podoba soboru `Dockerfile` v této diplomové práci:

[22] Web Server Gateway Interface

```
# nastavení výchozího image
FROM python:3.6.3-alpine

# nastavení systémové proměnné
ENV TZ Europe/Prague

# instalace systémových balíků
RUN apk add --no-cache tzdata mariadb-client-libs

# přesun do vybraného adresáře
WORKDIR /app/src

# přidání souboru do kontejneru z počítače,
# na kterém se sestavuje tento image
ADD requirements.txt ../

# instalace python balíků
ENV INSTALL_PACKAGES build-base linux-headers bash git openssh mysql-dev
RUN apk add --no-cache $INSTALL_PACKAGES && \
    pip install --no-cache-dir -r ../requirements.txt && \
    apk del $INSTALL_PACKAGES

# přídání zdrojových kódů do kontejneru
ADD ./src/ ./
```

Abychom sestavený kontejner (image) mohli stahovat na jiných počítačích pomocí příkazu `docker pull`, je potřeba jej nahrát do jednoho z mnoha repozitářů, tzv. registrů. Já jsem využil GitLab registry [73]. Takto vypadá postup publikování image do registrů:

1. sestavení Docker image

   ```
   $ docker build -t <jméno včetně tagu> .
   ```

2. přihlášení do registrů

   ```
   $ docker login -u <user> -p <heslo> <adresa registru>
   ```

3. odeslání image do registrů

   ```
   $ docker push <jméno včetně tagu>
   ```

7.2.2 Konfigurace

Kromě aplikace ve vlastním kontejneru je potřeba pustit i databázi, která s aplikací úzce spolupracuje. Jejich konfigurace se typicky nachází v souboru `docker-compose.yml`. V této práci jsou konfigurace celkem dvě, jedna pro vývoj a druhá pro produkční nasazení. Hesla a jiné citlivé údaje se pochopitelně v repozitáři nenachází.

Docker Compose [72] je nástroj sloužící pro definování běhu více Docker kontejnerů najednou. Definuje se pomocí YAML[23] souboru a umožňuje komplexnější správu, např. škálování kontejnerů. Na následujících řádcích je zjednodušená konfigurace produkčního prostředí, které je použito u mé aplikace. Zajímavostí může být nastavení `volumes`, které mapuje prostor na hostitelském počítači na prostor v kontejneru (prostor, kam MySQL ukládá data). Díky tomu je tak zajištěná perzistence dat, i přesto, že se smaže databázový kontejner.

```
version: '3'
services:
  app:
    # vlastní kontejner, v němž běží catcher
    image: registry.gitlab.com/dstlmrk/catcher:latest
    # kontejner se snaží restartovat, pokud dojde k chybě
    restart: always
    ports:
      - 80:80
    depends_on:
        - mysql
    # příkaz, který spouští aplikaci v kontejneru
    command: python ./wsgi.py
    # systémové proměnné
    environment:
      MAIL_SERVER: smtp.gmail.com
      MAIL_DEFAULT_SENDER: noreply.catcher@gmail.com
      LOG_LEVEL: INFO
      DB_NAME: catcher
      # místo IP adresy stačí použít hostname
      DB_HOST: mysql
      ...

  mysql: # název se používá taky jako hostname
    # kontejner, v němž běží MySQL databáze
    image: mysql:5.7
    # příkaz, který spouští databázi v kontejneru
    command: mysqld --character-set-server=utf8mb4
    # nastavení toho, aby se data místo do kontejneru
    # ukládala na hostitelský systém
    volumes:
     - ./data/db:/var/lib/mysql
    restart: always
    # systémové proměnné
    environment:
      MYSQL_ROOT_PASSWORD: <root heslo>
      MYSQL_DATABASE: catcher
      TZ: Europe/Prague
```

[23] Ain't Markup Language

7.3 Continuous Integration

Pokud se některý z procesů při nasazování a testování opakuje, je vhodné přemýšlet o průběžné integraci. Jde o souhrn technik a nástrojů, které zrychlují vývoj a slouží k nalezení chyb nebo nedostatků.

Já jsem pro průběžnou integraci využil GitLab CI. Při každém commitu do větve *master* v Git repozitáři proběhne celkem pět fází (každá v Docker kontejneru, který si zvolím), které mi zaručují bezpečné nasazení aplikace:

1. Prebuild testování: Okamžité testování na již vytvořeném Alpine Linux kontejneru, který obsahuje velké množství Python balíčků. Účelem je, aby testy mohly proběhnout co nejrychleji od commitu.

2. Build: Sestavení image podle souboru `Dockerfile`.

3. Testování: Stejné testování jako v prvním případě s tím rozdílem, že tentokrát se testuje v již sestaveném kontejneru, což zajišťuje totožné prostředí s produkčním. Ve webové aplikaci GitLab je možné si prohlédnout výstupy jednotlivých fází, např. testování:

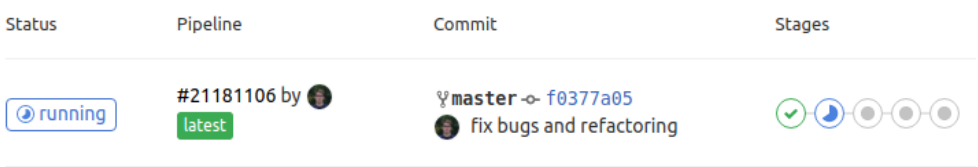
```
...
tests/web/test_user.py::test_register_post PASSED          [ 92%]
tests/web/test_user.py::test_reset_password_get PASSED     [ 94%]
tests/web/test_user.py::test_change_password_get PASSED    [ 96%]
tests/web/test_user.py::test_unconfirmed_get PASSED        [ 98%]
tests/web/test_user.py::test_confirm_email_get PASSED      [100%]

======== 49 passed, 4 skipped, 3 xfailed in 3.51 seconds ========
Job succeeded
```

4. Release: Image je odeslán do GitLab registru.

5. Nasazení: Pomocí SSH klienta dojde k přihlášení na produkční prostředí, stáhnutí nejnovějšího souboru `docker-compose.yml`, přihlášení do registru a spuštění `docker-compose pull && docker-compose up -d`. Docker už si sám stáhne nejnovější verze kontejnerů a pustí je na pozadí.

Pokud některá fáze skončí chybou, celý proces nasazení se ukončí a GitLab odešle notifikaci o neúspěchu. Pro představení toho, jak je jednoduché CI nastavit, následuje výtah z jeho konfigurace a popsání testovací fáze.

```
stages:
  - prebuild-test
  - build
  - test
  - release
  - deploy

variables:
  MYSQL_VERSION: 5.7.21
  TEST_DATABASE: catcher
  ...

test:
  stage: test

  # v jakém kontejneru se aplikace pustí
  image: $CONTAINER_TEST_IMAGE

  # jaké služby se mají v průběhu fáze pustit (zajišťuje GitLab)
  services:
    - mysql:$MYSQL_VERSION

  script:
    # nainstalování MySQL klienta v Alpine Linux
    - apk --update add mysql-client
    # vytvoření tabulek v databázi pro účely testování
    - mysql --user=root --password=<heslo> --host=mysql \
      "$MYSQL_DATABASE" < mysql/fixtures/1_schema.sql
    # instalace všech závislostí potřebných pro testování
    - pip install -r dev_requirements.txt
    # spuštění testů
    - py.test -v --color=yes tests

  variables:
    MYSQL_DATABASE: $TEST_DATABASE
    MYSQL_ROOT_PASSWORD: <heslo>
```



Obrázek 7.2: Screenshot z GitLab webové aplikace, na kterém je znázorněn aktuální průběh nasazení nové verze.

7.4 Údržba

Pokud je aplikace již v provozu a přistupují k ní uživatelé, je žádoucí, aby neměla žádné výpadky. A pokud nějaké má, tak abychom o nich věděli a mohli je začít včas řešit. K tomu je dobré aplikaci a v podstatě i celou infrastrukturu (např. volné místo na disku) sledovat. Já používám poměrně jednoduché řešení – další webovou aplikaci Uptime Robot [69], která v pravidelném intervalu přistupuje k aplikaci Catcher a testuje její dostupnost. V případě detekování výpadku posílá notifikaci emailem. Použití aplikace je pro omezené monitorování zdarma.

Dalším rizikem je ztáta dat. Proto je vhodné databázi jednou za čas zálohovat, např. programem mysqldump [70]. Zautomatizovat jeho volání je možné Cronem[24]. Samozřejmostí je mít zálohovaná data jinde, než samotnou aplikaci. Databáze aplikace Catcher se zatím zálohuje pouze manuálně.

[24]Softwarový démon, který obsluhuje pravidelné spouštění periodicky se opakujících procesů.

Kapitola 8

Návrh budoucích rozšíření

I přesto, že tento projekt splňuje zadání diplomové práce v celém rozsahu, během vývoje a testování se objevilo několik podnětů a připomínek, které vedou k následujícím návrhům rozšíření nebo vylepšení aplikace.

Legislativní ošetření

S přícházejícím nařízením GDPR o ochraně osobních údajů (legislativa Evropské unie), které vejde v platnost 25. května 2018 [74], je potřeba aplikaci upravit tak, aby tomuto nařízení vyhovovala. Problémovým místem v aplikaci může být uchovávání emailových adres uživatelů. Dalším legislativním opatřením je vyžádání souhlasu uživatele při využívání cookies[25].

Bezpečnost

Pro získání důvěry uživatelů je bezpodmínečně nutné přejít z protokolu HTTP na HTTPS.

Zvýšení výkonu

Aplikace stále běží na vývojovém Flask serveru určeném pro vývoj. Pro více spojení je však nutné použít plnohodnotnou verzi uWSGI a webového serveru, např. Nginx.

Uživatelské rozhraní

Všechny připomínky k uživatelském rozhraní jsou zpracovány v sekci 6.2.4.

[25] HTTP cookie - data uložená v počítači uživatele

Nová funkcionalita

Přidávání týmů do aplikace

Přidávat týmy není bez zásahu v databázi možné. Pro použití v ČR je to prozatím dostačující, protože většina týmů se již v DB nachází, ale pro budoucí použití je potřeba uživatelům poskytnout tuto funkcionalitu.

Delegovat zadávání SOTG z organizátora na jednotlivé týmy

To by znamenalo, že uživatel by kromě turnajů mohl spravovat i jednotlivé týmy a za ně by pak odevzdával hodnocení SOTG. Došlo by k ulehčení práce organizátora.

Podrobnější zadávání hodnocení SOTG

Poněvadž podle pravidel Ultimate Frisbee lze hodnocení zadávat i více komplexně, než pouze číslem od 0 do 20, bylo by vhodné hodnocení dostatečně upravit.

Zaznamenávání detailního průběhu zápasů

Možnost zaznamenávat průběžné skóre v době mezi začátkem a koncem zápasu.

Soupisky týmů a statistiky hráčů

Možnost vložit nebo importovat soupisky hráčů, u kterých lze evidovat statistiky, jako například chycené body či nahrávky.

Závěr

V rámci této diplomové práce jsem čtenáře uvedl do celkové problematiky, analyzoval již existující řešení, navrhl možná východiska a v neposlední řadě provedl realizaci webové aplikace. Ta je implementovaná za pomoci programovacího jazyka Python a jeho několika přidružených technologií určených pro vývoj webových aplikací – zejména frameworku Flask. Aplikace je úspěšně nasazena na virtuální privátní server a celý proces nasazování, včetně testování, zautomatizován pomocí CI.

Během vývoje jsem se neustále učil a dnes bych návrh a strukturu projektu provedl mírně jinak. Zejména bych se více zaměřil na důraznější oddělení jednotlivých vrstev architektury a umístění kódu v nich. V oblasti návrhu uživatelského rozhraní bych se v některých částech webu zaměřil více na jednoduchost než komplexnost.

Hlavním cílem bylo navrhnout a implementovat webovou aplikaci. V návrhu byl kladen důraz na uživatelské rozhraní a jeho testování. I přes moji naivní představu o velikosti rozsahu práce, jsem dokončil všechny hlavní i dílčí cíle. V části zabývající se analýzou konkurenčních řešení jsem se také utvrdil v tom, že podobně jednoduchá aplikace pro správu turnajů stále chybí. Proto její další vývoj dává smysl.

Literatura

[1] *The AUDL: American Ultimate Disc League* [online]. c2018 [cit. 2018-05-08]. Dostupné z: http://theaudl.com/

[2] What is Ultimate? *USA Ultimate: About Ultimate* [online]. c2015 [cit. 2018-05-08]. Dostupné z: http://www.usaultimate.org/about/

[3] Historie frisbee v ČR. *Česká asociace létajícího disku* [online]. [cit. 2018-05-08]. Dostupné z: http://www.cald.cz/historie-ultimate-v-cr

[4] ZÁPIS Z VALNÉ HROMADY 2017. In: *Česká asociace létajícího disku* [online]. [cit. 2018-05-08]. Dostupné z: http://www.cald.cz/sites/cald.cz/files/zapis-z-valne-hromady-cald-2017.pdf

[5] DOSTÁL, Marek. Systém pro skórování Ultimate Frisbee zápasů - backend: Nasazení. Praha, 2016. Bakalářská práce. České vysoké učení technické v Praze, Fakulta informačních technologií, Katedra softwarového inženýrství. Vedoucí práce Ing. Jiří Hunka.

[6] MAHDAL, Jakub. *Moderní rámce pro tvorbu webových aplikací: Architektura software a webové aplikace.* Brno, 2006. DIPLOMOVÁ PRÁCE. MASARYKOVA UNIVERZITA FAKULTA INFORMATIKY. Vedoucí práce Mgr. Jan Pavlovič.

[7] *JavaScript* [online]. c2016 [cit. 2018-05-03]. Dostupné z: https://www.javascript.com/

[8] GARRETT, Jesse James. Ajax: A New Approach to Web Applications. In: *Adaptive Path* [online]. c2018, 18.1.2005 [cit. 2018-05-03]. Dostupné z: http://adaptivepath.org/ideas/ajax-new-approach-web-applications/

[9] APACHE SOFTWARE FOUNDATION. *Apache HTTP Server Project* [online]. c1997-2016 [cit. 2018-05-03]. Dostupné z: https://httpd.apache.org/

[10] *Nginx* [online]. [cit. 2018-05-03]. Dostupné z: http://nginx.org/

[11] *Java* [online]. [cit. 2018-04-27]. Dostupné z: https://www.java.com/en/

[12] 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.

[13] *Python.org* [online]. c2001-2018 [cit. 2018-05-03]. Dostupné z: https://www.python.org/

[14] *Ruby Programming Language* [online]. [cit. 2018-05-03]. Dostupné z: https://www.ruby-lang.org/en/

[15] VOSEČEK, Jiří, Ondřej BURKERT a Petr KOTĚŠOVEC. Mobilní aplikace pro skórování na turnajích. In: *Česká Asociace Létjícího Disku* [online]. 2013 [cit. 2018-01-21]. Dostupné z: http://cald.cz/novinky/mobilni-aplikace-pro-skorovani-na-turnajich

[16] Catcher. *Google Play* [online]. c2016 [cit. 2018-01-21]. Dostupné z: https://play.google.com/store/apps/details?id=com.ulti.catcher

[17] Ultimate Organizer. *SourceForge* [online]. 2016 [cit. 2018-01-21]. Dostupné z: https://sourceforge.net/projects/ultiorganizer/

[18] *Ultimate Central: Find and manage ultimate frisbee events all over the world* [online]. [cit. 2018-01-21]. Dostupné z: https://ultimatecentral.com/

[19] *About Ultimate Central* [online]. [cit. 2018-01-21]. Dostupné z: https://ultimatecentral.com/en_us/about/

[20] *Leaguevine: Take stats for your team anywhere, anytime, with our new app.* [online]. c2010-2018 [cit. 2018-01-21]. Dostupné z: https://www.leaguevine.com/

[21] *Leverade: League Manager and Online Tournament organizer.* [online]. c2018 [cit. 2018-01-21]. Dostupné z: https://leverade.com/

[22] *Konkuri: Tournament and league manager* [online]. c2009-2018 [cit. 2018-01-21]. Dostupné z: http://www.konkuri.com/

[23] *Enjore.com: Tournament and league manager* [online]. c2012-2018 [cit. 2018-01-21]. Dostupné z: https://www.enjore.com/

[24] Model View Controller. KELLY, Sloan. *Python, PyGame, and Raspberry Pi game development.* New York, NY: Apress, 2016, s. 141-151. ISBN 9781484225165.

[25] Laravel Design Patterns and Best Practices. In: *Safari* [online]. [cit. 2018-05-03]. Dostupné z: https://www.safaribooksonline.com/library/view/laravel-design-patterns/9781783287987/ch01s02.html

[26] VAN ROSSUM, Guido. *A Brief Timeline of Python*. In: The History of Python [online]. 2009 [cit. 2018-05-04]. Dostupné z: http://python-history.blogspot.cz/2009/01/brief-timeline-of-python.html

[27] COCHRANE, Ken. *Best Python Companies to Work For*. In: DZone [online]. 2012 [cit. 2018-05-04]. Dostupné z: https://dzone.com/articles/best-python-companies-work

[28] KUCHLING, A. M. Functional Programming HOWTO: Introduction. PYTHON SOFTWARE FOUNDATION. *Python 2.7.11 documentation* [online]. c1990-2016 [cit. 2018-05-04]. Dostupné z: https://docs.python.org/2/howto/functional.html

[29] PYPL Index. *PYPL PopularitY of Programming Language* [online]. 2016 [cit. 2018-05-04]. Dostupné z: http://pypl.github.io/PYPL.html

[30] Stack Overflow Trends: Stack Overflow Insights. *Stack Overflow* [online]. c2018 [cit. 2018-05-04]. Dostupné z: https://insights.stackoverflow.com/trends

[31] RITCHIE, Dennis M. The Development of the C Language. *Second History of Programming Languages conference*. Cambridge, Mass., 1993 [cit. 2018-05-04]. Dostupné také z: https://www.bell-labs.com/usr/dmr/www/chist.pdf

[32] *Django: The web framework for perfectionists with deadlines.* [online]. c2005-2018 [cit. 2018-05-04]. Dostupné z: https://www.djangoproject.com/

[33] *Flask* [online]. c2010-2018 [cit. 2018-05-04]. Dostupné z: http://flask.pocoo.org/

[34] *Werkzeug: The Python WSGI Utility Library* [online]. c2014 [cit. 2018-05-04]. Dostupné z: http://werkzeug.pocoo.org/

[35] *Jinja2: Jinja2 is a templating engine for Python.* [online]. c2008 [cit. 2018-05-04]. Dostupné z: http://jinja.pocoo.org/docs/2.10/

[36] *WSGI* [online]. c2006-2015 [cit. 2018-05-04]. Dostupné z: http://wsgi.tutorial.codepoint.net

[37] *MySQL* [online]. c2018 [cit. 2018-05-04]. Dostupné z: https://www.mysql.com/

[38] SUMATHI, S. a S. ESAKKIRAJAN. Structured Query Language. *Fundamentals of relational database management systems.* Berlin: Springer, 2007, s. 111-212. ISBN 9783540483991.

[39] *MySQL: MySQL Workbench* [online]. c2018 [cit. 2018-05-04]. Dostupné z: https://www.mysql.com/products/workbench/

[40] DRIES, Buytaert. The history of MySQL AB. In: *Dries Buytaert: On digital experiences, Open Source, startups & the future* [online]. c1999-2018 [cit. 2018-05-04]. Dostupné z: http://buytaert.net/the-history-of-mysql-ab

[41] *MariaDB.org: The MariaDB Foundation* [online]. c2018 [cit. 2018-05-04]. Dostupné z: https://mariadb.org/

[42] *Percona: The Database Performance Experts* [online]. c2006-2018 [cit. 2018-05-04]. Dostupné z: https://www.percona.com/

[43] *SQLAlchemy: The Python SQL Toolkit and Object Relational Mapper* [online]. [cit. 2018-05-05]. Dostupné z: https://www.sqlalchemy.org/

[44] *PostgreSQL* [online]. c1996-2018 [cit. 2018-05-05]. Dostupné z: http://www.postgresql.org/

[45] *SQLite* [online]. [cit. 2018-05-05]. Dostupné z: https://www.sqlite.org/

[46] SQLAlchemy: Get SQLAlchemy. *SQLAlchemy* [online]. [cit. 2018-05-05]. Dostupné z: http://www.sqlalchemy.org/download.html

[47] Models and databases. *Django: The web framework for perfectionists with deadlines.* [online]. [cit. 2018-05-05]. Dostupné z: https://docs.djangoproject.com/en/2.0/topics/db/

[48] SOFTWARE FREEDOM CONSERVANCY. *Git* [online]. [cit. 2018-05-05]. Dostupné z: https://git-scm.com/

[49] *GitHub* [online]. c2018 [cit. 2018-05-05]. Dostupné z: https://github.com/

[50] DOLL, Brian. 10 Million Repositories. In: *GitHub: Blog* [online]. 2013 [cit. 2018-05-05]. Dostupné z: https://github.com/blog/1724-10-million-repositories

[51] *GitLab: Concurrent DevOps* [online]. [cit. 2018-05-05]. Dostupné z: https://about.gitlab.com/

[52] Learn Django – Chapter 3: Django Flow. In: *Slash4 Blog* [online]. c2018 [cit. 2018-05-06]. Dostupné z: http://slash4.net/blog/django-tutorial/learn-django-views-templates-and-urls.html

[53] Browser Statistics: The Most Popular Browsers. *W3schools.com* [online]. c1999-2018 [cit. 2018-05-06]. Dostupné z: https://www.w3schools.com/browsers/default.asp

[54] SCHECHTER, Emily. A secure web is here to stay. In: *Google Security Blog* [online]. 8. 2. 2018 [cit. 2018-05-06]. Dostupné z: https://security.googleblog.com/2018/02/a-secure-web-is-here-to-stay.html

[55] NIELSEN, Jakob. 10 Usability Heuristics for User Interface Design. *Nielsen Norman Group* [online]. 2018, 1.1.1995 [cit. 2018-04-26]. Dostupné z: https://www.nngroup.com/articles/ten-usability-heuristics/

[56] Unittest. PYTHON SOFTWARE FOUNDATION. *Python 2.7.11 documentation* [online]. c1990-2016 [cit. 2018-04-26]. Dostupné z: https://docs.python.org/2.7/library/unittest.html

[57] *JUnit 5* [online]. 2018 [cit. 2018-04-27]. Dostupné z: https://junit.org/junit5/

[58] *Pytest* [online]. 2017 [cit. 2018-04-27]. Dostupné z: https://pytest.org/

[59] Flexmock. *PyPI: the Python Package Index* [online]. c2018 [cit. 2018-04-27]. Dostupné z: https://pypi.org/project/flexmock/

[60] SOFTWARE FREEDOM CONSERVANCY. *Git* [online]. [cit. 2018-05-02]. Dostupné z: https://git-scm.com/

[61] *Docker: Build, Ship, and Run Any App, Anywhere* [online]. c2018 [cit. 2018-05-02]. Dostupné z: https://www.docker.com/

[62] SEO, Kyoung-Taek, Hyun-Seo HWANG, Il-Young MOON, Oh-Young KWON a Byeong-Jun KIM. *Performance Comparison Analysis of Linux Container and Virtual Machine for Building Cloud: Docker* [online]. c2014, , 107 [cit. 2018-05-02]. Dostupné z: http://onlinepresent.org/proceedings/vol66_2014/25.pdf

[63] WHAT IS A CONTAINER: A standardized unit of software. In: *Docker* [online]. c2018, c2018 [cit. 2018-05-02]. Dostupné z: https://www.docker.com/what-container

[64] CANONICAL LTD. *Ubuntu* [online]. c2018 [cit. 2018-05-07]. Dostupné z: http://www.ubuntu.com/

[65] uWSGI 2.0.17 PYTHON SOFTWARE FOUNDATION. *Python: Pypi* [online]. c2018 [cit. 2018-05-07]. Dostupné z: https://pypi.org/project/uWSGI/

[66] pip 10.0.1 PYTHON SOFTWARE FOUNDATION. *Python: Pypi* [online]. c2018 [cit. 2018-05-07]. Dostupné z: https://pypi.org/project/pip/

[67] PRAUS, Petr. Produkční nasazení Django aplikací na Cherokee pomocí WSGI. In: *Zdroják.cz* [online]. [cit. 2018-05-07]. Dostupné z: https://www.zdrojak.cz/clanky/produkcni-nasazeni-django-aplikaci-na-cherokee-pomoci-wsgi

[68] GNU Wget. FREE SOFTWARE FOUNDATION. *GNU Operating System* [online]. c2017, aktualizováno 15.9.2017 [cit. 2018-05-07]. Dostupné z: https://www.gnu.org/software/wget/

[69] *Uptime Robot* [online]. c2018 [cit. 2018-05-07]. Dostupné z: https://uptimerobot.com

[70] Mysqldump: A Database Backup Program. *MySQL* [online]. c2018 [cit. 2018-05-07]. Dostupné z: http://dev.mysql.com/doc/refman/5.7/en/mysqldump.html

[71] *Alpine Linux* [online]. c2017 [cit. 2018-05-07]. Dostupné z: https://alpinelinux.org/

[72] Docker Compose. *Docker Docs* [online]. c2018 [cit. 2018-05-07]. Dostupné z: https://docs.docker.com/compose/

[73] PUNDSACK, Mark. GitLab Container Registry. In: *GitLab* [online]. 23. 5. 2016 [cit. 2018-05-07]. Dostupné z: https://about.gitlab.com/2016/05/23/gitlab-container-registry/

[74] *GDPR: Obecné nařízení o ochraně osobních údajů* [online]. [cit. 2018-05-07]. Dostupné z: https://www.gdpr.cz/

Příloha **A**

Seznam použitých zkratek

API Application Programming Interface

CI Continuous Integration

CSS Cascading Style Sheets

ČALD Česká asociace létajícího disku

GUI Graphical user interface

Hi-Fi High-Fidelity

HTTP Hypertext Transfer Protocol

HTTPS Hypertext Transfer Protocol Secure

JSON JavaScript Object Notation

Lo-Fi Low-Fidelity

OOP Object-oriented programming

ORM Object-relational mapping

RAM Random Access Memory

REST Representational State Transfer

SOTG Spirit of the Game

SQL Structured Query Language

SSL Secure Sockets Layer

TLS Transport Layer Security

VPS Virtual private server

WSGI Web Server Gateway Interface

XML Extensible markup language

Příloha B

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