ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Vodvářka** Jméno: **Otto** Osobní číslo: **474502**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačů**

Studijní program: **Softwarové inženýrství a technologie**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Aplikace pro tenisové kluby - backendová část

Název bakalářské práce anglicky:

Tennis application for clubs - backend component

Pokyny pro vypracování:

Nastudujte metodologii návrhu a tvorby moderních webových aplikací, dělených na logicky oddělenou frontendovou a backendovou část. Tyto komponenty jsou spolu provázány jen pomocí jasně definovaného HTTP rozhraní (REST, RPC, GraphQL a podobně).
Následně navrhněte a vytvořte backendovou část webové aplikace, poskytující jednoduchý a přehledný způsob správy tenisových klubů přinášející benefity i samotným hráčům. Vycházejte primárně z dat, nabízených oficiálním webem 'cztenis.cz'. Analyzujte nabízená data, uvažte možnost jejich strojového zpracování a vytvořte nástroj, který bude tato data konzumovat.
Pro potřeby komunikace s klientskou stranou aplikace definujte a otestujte vhodné rozhraní, které pokryje všechny uživatelské scénáře.
Hlavní rysy aplikace budou:
1) Registrace tenisových klubů
2) Registrace hráčů
3) Správa kurtů
4) Harmonogramy a výsledky soutěží
Výslednou aplikaci otestujte z uživatelského (kvalitativní testování) i bezpečnostního hlediska. Popište limity té komponenty aplikace, která se pomocí tzv. webscrapingu stará o zpracovávání dat poskytovaných třetí stranou (cztenis.cz). Nezapomeňte, že nedílnou součástí aplikace je i dokumentace generovaná ze zdrojového kódu.

Seznam doporučené literatury:

Clarence Ho, Chris Schaefer, Rob Harrop, Iuliana Cosmina: Pro Spring 5, Apress 2017, ISBN 9781484228081
Casimir Saternos: Client-Server Web Apps with JavaScript and Java, O'Reilly Media 2014, ISBN 9781449369330
https://developer.mozilla.org/en-US/docs/Web

Jméno a pracoviště vedoucí(ho) bakalářské práce:

RNDr. Ondřej Žára, Katedra počítačové grafiky a interakce

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) bakalářské práce:

Datum zadání bakalářské práce: **13.02.2020** Termín odevzdání bakalářské práce: **22.05.2020**

Platnost zadání bakalářské práce: **30.09.2021**

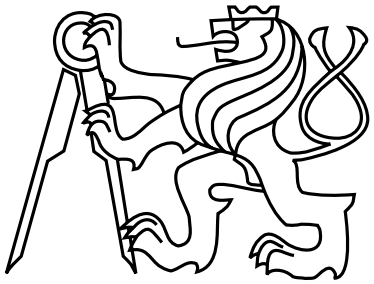
RNDr. Ondřej Žára		prof. Mgr. Petr Páta, Ph.D.
podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Student bere na vědomí, že je povinen vypracovat bakalářskou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v bakalářské práci.

.	
Datum převzetí zadání	Podpis studenta

České vysoké učení technické v Praze
Fakulta elektrotechnická
Katedra počítačů



Bakalářská práce

Aplikace pro tenisové kluby - backendová část

Otto Vodvářka

Vedoucí práce: RNDr. Ondřej Žára

Studijní program: Softwarové inženýrství a technologie

květen 2020

Poděkování

Chtěl bych zde poděkovat vedoucímu bakalářské práce RNDr. Ondřeji Žárovi za cenné rady, věcné připomínky a vstřícnost při konzultacích. Zároveň bych rád poděkoval kolegovi Ondřeji Marešovi za výbornou spolupráci při vývoji bakalářské práce.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze dne 20. 5. 2020 .....

Abstract

The thesis deals with the methodology of analysis and implementation of a modern web application with strictly separated frontend and backend sides, which are communicating via the defined HTTP REST interface. This thesis is focused primarily on the problematics of server-side development.
The final application provides a simple and intuitive way to manage tennis clubs, which brings benefits to both, managers and players. To improve user interaction and increase the informational value of the website, publicly accessible data from the official website of the Czech Tennis Association are used.

Keywords: Web application, Tennis, Backend, Spring Boot, REST, Software development, Web scraping, Relational database

Abstrakt

Práce se zabývá metodologií návrhu a tvorby moderní webové aplikace se striktně rozdělenou frontendovou a backendovou částí, které jsou navzájem propojeny pomocí definovaného HTTP REST rozhraní. V této práci je zpracována problematika vývoje pouze ze serverové strany.
Vytvořená aplikace poskytuje jednoduchý a přehledný způsob správy tenisových klubů, která přináší benefity jak správcům klubů, tak i samotným hráčům. Pro zlepšení uživatelské interakce a zvýšení informační hodnoty webových stránek jsou využívány veřejně přístupná data z oficiálního webu Českého tenisového svazu.

Klíčová slova: Webová aplikace, Tenis, Backend, Spring Boot, REST, Vývoj softwaru, Web scraping, Relační databáze

Obsah

Seznam obrázků

Seznam tabulek

Kapitola 1

Úvod

1.1 Popis

Tato práce se zabývá problematikou vývoje serverové části webové aplikace, která slouží tenisovým klubům v České republice. Mezi hlavní funkce patří správa tenisových klubů, rezervace tenisových kurtů a osobní detail závodních hráčů. Zajímavostí této aplikaci je provázání s oficiálním webem Českého tenisového svazu[1] pomocí tzv. web scrapingu neboli extrakce dat přímo z HTML stránek. Aplikaci se jmenuje TK21 - Tenisové kluby 21.století.[2]

Aplikace je vyvíjena ve dvoučlenném týmu společně s kolegou Ondřejem Marešem, který je studentem 3.ročníku oboru Softwarové inženýrství a technologie na ČVUT. Vývoj aplikace je striktně rozdělen na klientskou a serverovou část, kde se obě strany podílí na business logice aplikace a jejího testování.

1.2 Motivace

S kolegou hrajeme oba tenis již od dětství. Jsme dobře seznámeni se současnou situací používání moderních technologií v tenisových klubech, která je nedostatečná. Tato vytvořená aplikace by měla toto změnit. Chceme vytvořit aplikaci zcela zdarma, která je jednoduchá na používání. Obsahovala by nejen všechny základní informace o klubech, ale i o turnajích a soutěžích družstev. Především v mnoha malých klubech chybí možnost rezervovat kurt online. I toto by měla naše aplikace vyřešit ve formě velmi jednoduchého rezervačního systému dostupného všem klubům. Naší vizí je rozšíření aplikace po celé České republice do malých až středně velkých tenisových klubů.

[1] `http://www.cztenis.cz`
[2] `https://www.tk21.cz`

Kapitola 2

Analýza

2.1 Současná situace

V České republice je tenis velmi populární sport. Ve všech velkých městech a v drtivé většině menších měst se nachází několik tenisových klubů. Jen na Českém tenisovém svazu je registrováno přes 900 takovýchto klubů[15].

Většina malých a středně velkých klubů používá moderní technologie velmi zřídka, případně mají pouze vlastní statickou webovou stránku nebo využívají sociální síť Facebook.[1]. Příkladem může být Tenisový klub Písnice[2] či Tenisová akademie Březno u Loun[3]. Propracovanější systémy mají pouze kluby ve velkých městech, ovšem zdaleka ne všechny. Příkladem velkých klubů je TK Sparta Praha[4] nebo I. ČLTK Praha[5].

2.2 Konkurence

Konkurencí k této aplikaci by mohly být různé rezervační systémy pro sportovní kluby, jichž existuje velké množství, například ProKlub[6], které ovšem nenabízí rozhraní pro členy klubů. Dalším potenciálním konkurentem by mohly být online katalogy sportovních klubů, které ovšem kombinují několik sportů najednou. Výhoda tohoto systému oproti ostatním je zaměření pouze na tenisové kluby a jejich plnou podporu. Zároveň žádné ostatní systémy nepracují s daty z Českého tenisového svazu, takže nemají možnost ukazovat aktuální údaje bez dat zadaných přímo klubem.

2.3 Sběr požadavků

Na základě zadání a znalostí problematiky byly identifikovány následné základní vlastnosti aplikace, které jsou pro vyvíjenou aplikaci klíčové. Zatím byly identifikovány následující

[1] `https://www.facebook.com`
[2] `http://www.tkpisnice.cz`
[3] `http://www.tenisbrezno.cz`
[4] `http://www.tkspartapraha.cz`
[5] `https://cltk.cz`
[6] `https://www.proklub.cz`

role: *nepřihlášený uživatel*, *přihlášený uživatel*, *člen klubu*, *závodní hráč* a *administrátor klubu*. V případě zájmu je možné doplnit aplikaci o role *zaměstnanec* a *trenér*.

Vlastnost	**Popis**
Registrace uživatele	Možnost registrace nového uživatele pomocí unikátní emailové adresy
Registrace klubu	Přihlášený uživatel může zaregistrovat klub dle dat z Českého tenisového svazu nebo kompletně nový klub
Informace o klubech	Vyhledávání klubů a zobrazení všech důležitých informací týkajících se chodu klubu a jejich soutěží.
Členi klubu	Přidávání členů klubu, kterým jsou přiřazeny vyšší práva
Administrace klubu	Nastavování informací týkajících se klubu a správa členů
Rezervace kurtu	Možnost zarezervování tenisového kurtu uživatelem a správa rezervací administrátorem klubu
Příspěvky klubu	Možnost přidávání a správy příspěvků klubu včetně nahrávání obrázků
Přehled uživatele	Uživatelská stránka s příspěvky a podrobnými informacemi o jeho rezervacích, případně turnajích a soutěžích družstev

Tabulka 2.1: Základní vlastnosti

2.4 Získávání dat z Českého tenisového svazu

Jelikož jednou z klíčových funkcí je provázání aplikace s oficiálním webem Českého tenisového svazu, je nutné z něj získávat nějakým způsobem data. Ideálním řešením by bylo veřejné API[7], přes které by se informace velmi lehce získávaly. Žádné veřejné API ale Český tenisový svaz nemá. Je tedy nutné využít jinou metodu extrakce dat. Analýza HTML kódu webových stránek ukázala, že nejlepším řešením je použít tzv. web scraping[8]. Stránky jsou sice již zastaralé, ale relativně dobře organizované, většinou v tabulkách. Následující sekce podrobněji rozebírá způsob a použití vyextrahovaných dat.

2.4.1 Kluby

Umístění
Hlavní strana → Kluby

[7] API - Soubor funkcí či endpointů dostupných k využívání vývojáři

[8] Web scraping - Technika extrahování dat do lokálního úložiště pomocí automatizovaného kopírování dat z HTML stránek

URL(Přehled): *http://www.cztenis.cz/adresar-klubu*
URL(Detail klubu): *http://www.cztenis.cz/adresar-klubu/{id}*
id - Unikátní klubový identifikátor

Popis
Přehled klubů je rozdělen do několika sekcí podle oblastních svazů (Praha, Středočeský, Jihočeský atd.). V každé sekci jsou kluby vypsány v tabulce včetně odkazu na detailní přehled klubu, kde jsou zobrazeny podrobnější informace: např. adresa, telefon, web nebo kontaktní email.
Kluby jsou registrovány v aplikaci při splnění následujících podmínek:

- Název klubu obsahuje alespoň 3 znaky a zároveň jeho název nekončí řetězcem „zrušen“
- Adresa klubu je platná
- Existuje alespoň jeden kontaktní email

Použití
Získaná data jsou využívána především k zobrazování informací o klubu. Tyto kluby lze zaregistrovat pouze s účtem, který má identifikační email shodný s jedním z kontaktních emailů klubu. U takto registrovaného klubu není možné měnit údaje získané z Českého tenisového svazu.

2.4.2 Hráči

Umístění
Hlavní strana → Hledej hráče
Hlavní strana → {kategorie} → Žebříčky
URL(Detail hráče): *http://www.cztenis.cz/hrac/{id}*
URL(Žebříčky): *http://www.cztenis.cz/dospeli/zebricky{kategorie_url}*
id - Unikátní identifikátor hráče
kategorie - {Dospělí, Dorost, Starší žactvo, Mladší žactvo}
kategorie_url - {dospeli, dorost, starsi-zactvo, mladsi-zactvo}

Popis
Pro vyhledávání hráčů existují dva způsoby. Jeden přes vyhledávání hráčů podle jména a druhý přes žebříčky. Při analýze bylo zjištěno, že žebříčky neobsahují hráče s jedním bodem. To je značná část závodních hráčů. Metoda extrahování hráčů z žebříčků je tedy v našem případě nevhodná.
Při vyhledávání hráčů na základě jména je nutné brát v potaz následující vlastnost vyhledávání: jméno je nutné psát ve formátu příjmení a následně křestní jméno.
Tento způsob znemožňuje hromadné hledání všech hráčů, proto je tato extrakce použita pouze na vyžádání uživatele.

Použití
Hráči se na webových stránkách ČTS vyhledávají pouze v případě, že administrátor klubu chce přidat uživatele do klubu jako závodního hráče viz Obrázek 3.3. Takto propojení hráči mají na své zdi přehled svých turnajů a soutěží družstev. Hráč může být vždy registrován jako závodní hráč pouze u jednoho klubu.

2.4.3 Turnaje

Umístění
Hlavní strana → {kategorie} → Jednotlivci
URL(Seznam turnajů): *http://www.cztenis.cz/{kategorie_url}/jednotlivci*
URL(Detail turnaje): *http://www.cztenis.cz/turnaj/{id}/sezona/{sezona}/informace*
id - Unikátní identifikátor turnaje
kategorie - {Dospělí, Dorost, Starší žactvo, Mladší žactvo}
kategorie_url - {dospeli, dorost, starsi-zactvo, mladsi-zactvo}
sezona - kód pro sezónu, např. L20

Popis
Turnaje jsou rozděleny do 4 kategorií podle věku, kde v každé kategorii jsou turnaje dále rozděleny podle sezóny a pohlaví. V seznamu turnajů jsou zobrazeny základní údaje jako je druh turnaje, pořádající klub a datum konání ve tvaru *dd/MM-dd/MM* nebo *dd-dd/MM*

Použití
Získaná data se používají k zobrazení turnajů na stránce klubu a na úvodní stránce závodního hráče.

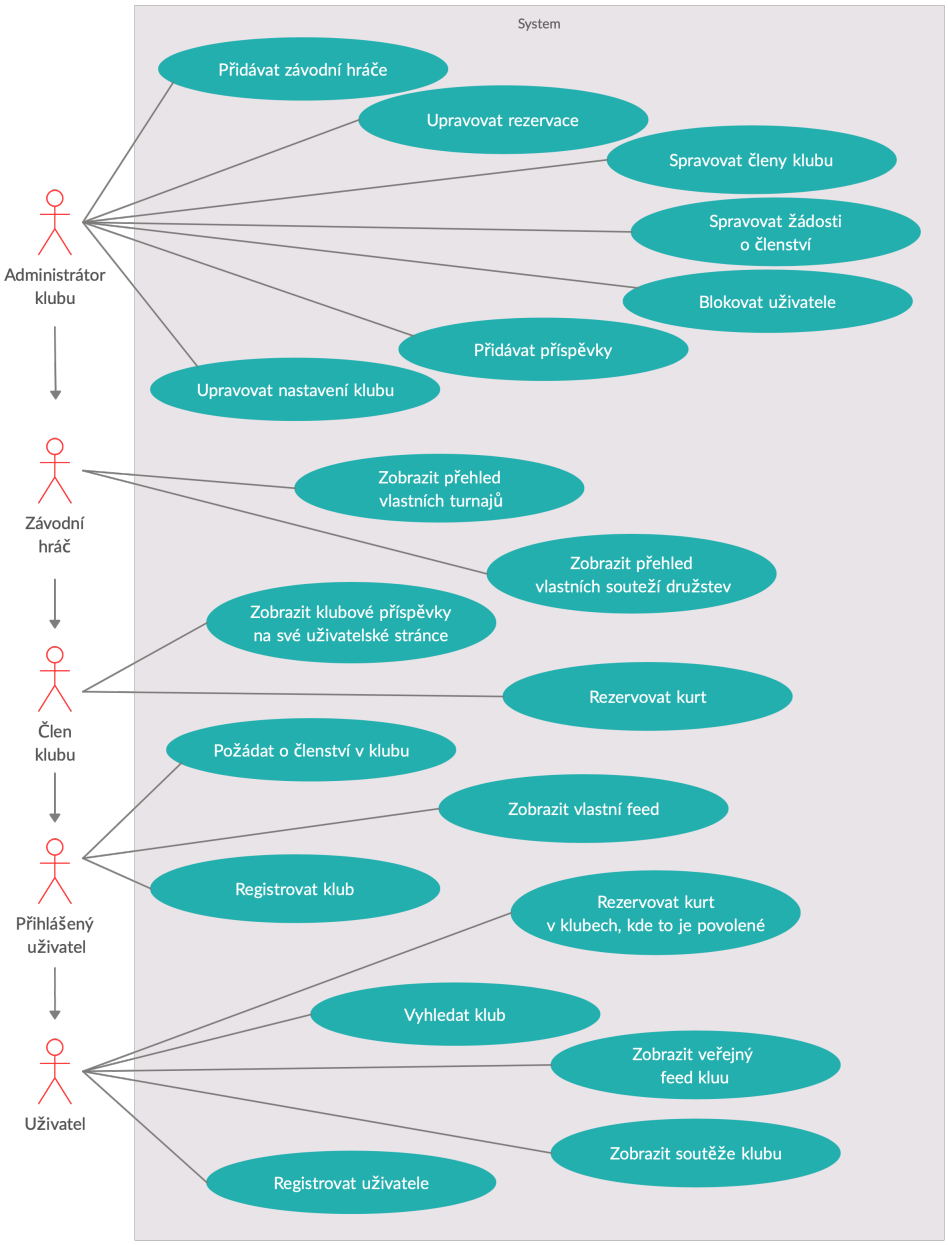
2.4.4 Soutěže družstev

Umístění
Hlavní strana → {kategorie} → Družstva
URL(Seznam soutěží): *http://www.cztenis.cz/{kategorie_url}/druzstva*
URL(Detail soutěže): *http://www.cztenis.cz/{kategorie_url}/druzstva/sezona/{rok}/soutez/{id}*
id - Unikátní identifikátor soutěže
kategorie - {Dospělí, Dorost, Starší žactvo, Mladší žactvo}
kategorie_url - {dospeli, dorost, starsi-zactvo, mladsi-zactvo}
rok - rok konání, např. 2020

Popis
Soutěže družstev jsou rozděleny do 4 kategorií podle věku, kde v každé kategorii jsou soutěže dále rozděleny podle oblastních svazů. V detailu soutěže se nacházejí týmy, termíny, výsledky zápasů a soupisky jednotlivých týmů. Termíny jednotlivých zápasů jsou problémovým místem, protože v současné situaci se na webových stránkách ČTS používá mnoho různých formátů dat.

Použití
Získaná data se používají k zobrazení soutěží družstev na stránce klubu a na úvodní stránce závodního hráče.

2.5 Případy užití



Obrázek 2.1: Případy užití

2.6 Technologie

2.6.1 Jazyk & Framework

Je celá řada jazyků, ve kterých lze psát backend webové aplikace např. PHP, Python, Java, Ruby atd. Pro mne byl výběr jednoznačný a tím je Java. Budu se tedy zabývat pouze technologiemi v Javě nebo s Javou kompatibilními.

Java EE
Java EE, nově známa také jako Jakarta, je standard rozšiřující Java SE. Java EE poskytuje API a prostředí pro vývoj velkých enterprise webových aplikací. Aplikace splňující kritéria Java EE nezávisí na konkrétní implementaci použitých API rozhraních a je tedy univerzálnější. Konkrétní implementace závisí na aplikačním serveru.[6]

Spring
Na rozdíl od Java EE, která je standardem, Spring je framework. Obě technologie využívají techniku zvanou Dependency Injection[6]. Přestože Spring není standardem, v popularitě Javu EE překonává[13]. Je již velmi rozvinutý, široce podporovaný a existuje mnoho materiálů o fungovaní tohoto frameworku. Framework poskytuje potřebné vzory a strukturu. Velmi populární je Spring Boot 2, který vývoj aplikací ještě více zjednodušuje a obsahuje v sobě aplikační server. Je tedy velmi jednoduché a hlavně velmi rychlé aplikaci spustit.

2.6.2 Databáze

Jednou z klíčových částí jakékoli webové aplikace je databáze. Existuje několik různých typů databází. Úplně nejzákladnější dělení je na relační a nerelační databáze. Mezi nerelační databáze patří Key-Value DB, Dokumentově orientovaná DB nebo grafové databáze atd.

2.6.2.1 Relační databáze

Relační databáze jsou reprezentovány předem definovaným relačním modelem. Ukládají data do tabulek a vztahy jsou reprezentovány tzv. cizími klíči.

Oracle
Oracle Databáze je velmi rychlá relační databáze. Je vhodná pro velké firmy, které potřebují získávat data velmi rychle a bez neočekávaných výpadků.

MySQL
MySQL je open-source relační databáze založená na jazyce SQL. Je vhodná spíše pro menší projekty, kde MySQL exceluje v rychlosti, ale s přibývajícími daty a tabulkami se postupně zpomaluje.

PotgreSQL
PostgreSQL je open-source objektově-relační databáze splňující standarty a obsahující plno možných rozšíření. Je vhodná pro větší data.

2.6.2.2 Key-value databáze

Key-value databáze neboli klíč-hodnota databáze je nejjednodušším typem NOSQL[9] databáze. Jak již název napovídá data jsou uloženy systémem klíč-hodnota. Klíč musí být vždy unikátní a jak klíč, tak hodnota mohou nabývat jakéhokoli typu. Tato databáze nemá žádný definovaný model, hodí se tedy spíše na věci jako je cache nebo IP tabulky.

2.6.2.3 Dokumentově orientovaná databáze

Tento typ databáze ukládá objekty do dokumentů. Můžeme si představit, že objekt je jeden řádek v relační databázi např. v tabulce Osoba. V relační databázi mají všechny související data cizí klíč odkazující na konkrétní řádek. V dokumentové databázi bude každá osoba reprezentována jedním dokumentem a všechny data související s konkrétní osobou budou uloženy ve stejném dokumentu nejčastěji ve formátu JSON[10]. S celým dokumentem tedy mohu pracovat podobně jak s JSON objektem.

2.6.2.4 Grafová databáze

Grafová databáze se specializuje na ukládání nikoli informací o entitách, ale o vztazích mezi entitami. Data jsou ukládány jako uzly v grafu a vztahy jako hrany mezi uzly. Je velmi jednoduché a efektivní dotazovat se na vztahy mezi entitami. Při použití modelu odpovídajícímu grafovým datům je vhodné použít tento typ databáze.

2.6.2.5 Relační vs dokumentová databáze

	Výhody	**Nevýhody**
Relační databáze	Jednoduchá struktura Rychlý update Atomicita Jednoduchý jazyk SQL	Rychlost závisí na velikosti databáze Pomalé JOIN příkazy Obtížná škálovatelnost Složité ukládání strukturovaných dat
Dokumentová databáze	Jednoduché ukládání strukturovaných dat Velmi rychlé dotazovaní Jednoduchá škálovatelnost	Pomalé updaty Nepodporuje atomicitu

Tabulka 2.2: Porovnání databází (Zdroj [5])

[9] NOSQL - Termín pro databáze, které neukládají data v relačních tabulkách
[10] JSON - JavaScript Object Notation

2.6.3 Komunikace mezi klientem a serverem

2.6.3.1 RPC

RPC neboli Remote procedure call dovoluje spouštět procesy na jiném počítači/serveru. Jedná se o klient-server architekturu s tím, že klient je volající a server vykonává jednotlivé požadavky. Obě strany fungují nad stejným rozhraním, což dává možnost klientovi volat metody serveru. V Javě se tomuto říká Java RMI.

2.6.3.2 SOAP

SOAP(Simple Object Access Protocol) je protokol založený na posílání XML[11] zpráv přes HTTP protokol. SOAP již byl dnes z velké části nahrazen rozhraním REST, není tedy příliš využívaný.[9] Právě oproti REST rozhraní je SOAP pomalejší kvůli posílání velkého množství podpůrných dat a je tedy zároveň složitější na pochopení.[4] Jedním z druhů komunikace je výše již zmíněné RPC.

2.6.3.3 REST

REST je na rozdíl od SOAP spíše architektonický styl, nikoli protokol. REST funguje přes HTTP, kde se posílají zprávy nejčastěji v JSON formátu, ale to záleží již na konkrétní implementaci. Samotná REST architektura nemá žádná striktně daná pravidla, pouze pět následujících doporučení[11]:

- Klient-server architektura
- Bezestavová komunikace
- Využití cache
- Jednotné rozhraní
- Vrstvená architektura

Aby mohlo být rozhraní považováno za REST API, není nutné implementovat všechny výše uvedené body, ale je nutné používat různé HTTP operace tímto způsobem:

- GET - vrátí požadovaný zdroj pokud existuje, nemá žádné vedlejší efekty
- POST - vytvoří nový zdroj na základě těla v dotazu
- PUT - aktualizuje požadovaný zdroj
- DELETE - smaže zadaný zdroj
- OPTIONS - server vrací operace, které je možné na vybraný zdroj použít

[11] XML - Extensible Markup Language

2.6.3.4 GraphQL

GraphQL je syntaxe, která popisuje, jaké data má server vrátit. Narozdíl od RESTu, kde je vše předem definované, zde si může klient diktovat, jaká data se mu mají vrátit. Toto umožňuje vrátit všechna potřebná data v jednom jediném HTTP dotazu. V současné době je tento způsob komunikace velmi populární.[7]

2.6.4 PaaS

Platform as a Service (PaaS) je služba cloudové infrastruktury, která zajišťuje platformu tak, aby se vývojáři starali pouze o samotný běh aplikace a nemuseli řešit problémy s hardwarem, virtualizací atd. V této oblasti je na výběr z mnoha možností.

2.6.4.1 Dokku

Dokku se často označuje jako „mini PaaS“. Je velmi jednoduché na používání. Je ideální pro začátečníky v PaaS problematice a i pro malé či domácí projekty. Nevýhodou Dokku je dostupnost. Dokku může běžet pouze na jednom serveru, nelze tedy zaručit vysokou dostupnost aplikace.

2.6.4.2 Flynn

Flynn je velmi podobný již zmíněnému Dokku. Může ovšem běžet najednou na více serverech, takže neexistuje místo, které se označuje jako „single point of failure“.[14] Vhodné pro menší projekty, u kterých je důležitá dostupnost.

2.6.4.3 Kubernetes

Pro střední a velké projekty je zde Kubernetes. Oproti předchozím PaaS implementacím je Kubernetes mnohem komplexnější a propracovanější. Je vhodný pro velké projekty, které si nesmí dovolit žádné výpadky.[14]

2.6.5 Web scraping

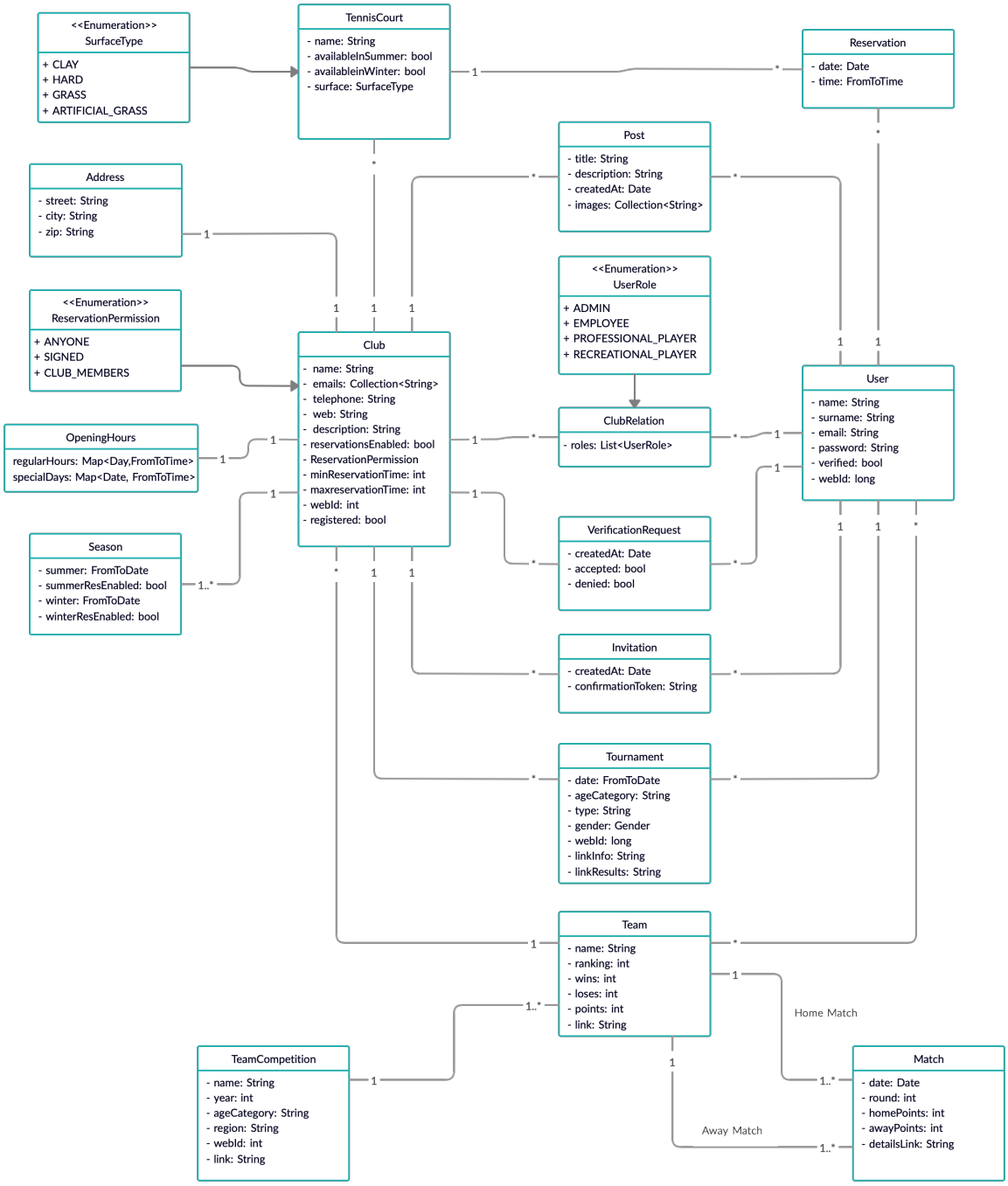
Pro web scraping existuje plno dostupných knihoven v jazyce Java pro usnadnění vývoje. Mezi nejpopulárnější patří Apache Nutch, Storm Crawler nebo JSoup. Apache Nutch a Storm Crawler jsou velké a propracované knihovny, schopné dobře fungovat i při velkých zátěžích a parsování složitých webových stránek. JSoup je na druhou stranu knihovna velmi lehká na použití, která se specializuje na parsování pouze HTML stránek pomocí CSS selektorů.[1]

Kapitola 3

Návrh

3.1 Diagram tříd

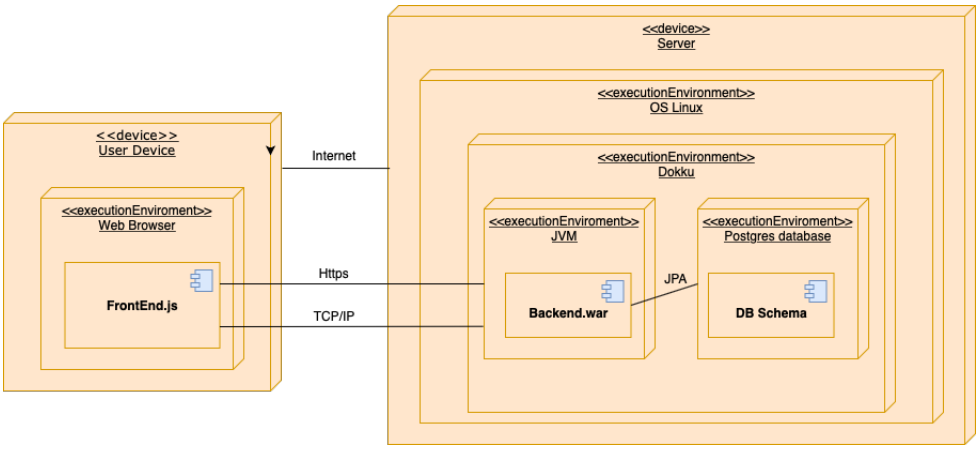
Pro lepší pochopení a rychlejší implementaci byl vytvořen diagram tříd v jazyce UML. Znázorňuje požadovanou strukturu modelu včetně atributů tříd a popsaných vztahů mezi entitami.



Obrázek 3.1: Diagram Tříd

3.2 Diagram nasazení

Následující diagram zobrazuje použité komponenty v nasazeném systému. Je zde znázorněno klasické použití HTTP protokolu přes webový prohlížeč. Databáze se nachází na stejném serveru jako aplikace především z finančních důvodů.



Obrázek 3.2: Diagram nasazení

3.3 Web scraping

3.3.1 Práce s nalezenými daty

Již často zmiňované extrahování dat ze stránek Českého tenisového svazu probíhá vždy ve dvou případech. Při nahrání nové produkční verze aplikace nebo každý den ve 03:00 ráno. Při pravidelném extrahování dat z jiné stránky vznikají nové business problémy, které je nutné řešit. Můžeme si představit například situaci, kdy klub registrovaný v databázi u ČTS je zároveň registrován i v naší aplikaci a každodenně ji používá. Co by se tedy mělo stát v případě, když klub je smazán z ČTS? Postup při takovýchto situacích je popsán v následujících tabulkách.

Hráči

Stav v TK21	Akce ČTS	Popis řešení
Registrován	Změna klubu	V klubu, ve kterém je označen jako závodní hráč, se stane rekreačním hráčem
Registrován	Změna jména	Jméno hráče se aktualizuje

Tabulka 3.1: Situace při scrapingu hráčů a jejich řešení

Kluby

Stav v TK21	Akce ČTS	Popis řešení
Registrován	Smazán/Zrušen	Klub zůstane v aplikaci, závodní hráči se stanou rekreačními a přestanou se každou noc aktualizovat
Registrován	Změna údajů	Všechny změněné údaje se aktualizují
Neregistrován	Smazán	Klub je smazán z databáze TK21
Neregistrován	Změna údajů	Všechny změněné údaje se aktualizují
Nenalezen	Přidán	Klub se automaticky přidá do aplikace TK21

Tabulka 3.2: Situace při scrapingu klubu a jejich řešení

Turnaje
Turnaje se hledají vždy v aktuální sezóně. V databázi se uchovávají pouze turnaje, které se konaly před méně než třemi roky. V opačném případě jsou z databáze smazány.

Stav v TK21	Akce ČTS	Popis řešení
Nalezen	Smazán/Zrušen	Turnaj se smaže z databáze TK21
Nalezen	Změna údajů	Všechny změněné údaje se aktualizují
Nalezen	Změna hráčů	Při každé iteraci se data přemažou novými
Nenalezen	Přidání	Turnaj se přidá do databáze TK21

Tabulka 3.3: Situace při scrapingu turnajů a jejich řešení

Soutěže družstev
Soutěže se hledají pouze v aktuálním roce. Při nalezení soutěží v novém roce, jsou data z minulého roku smazána.

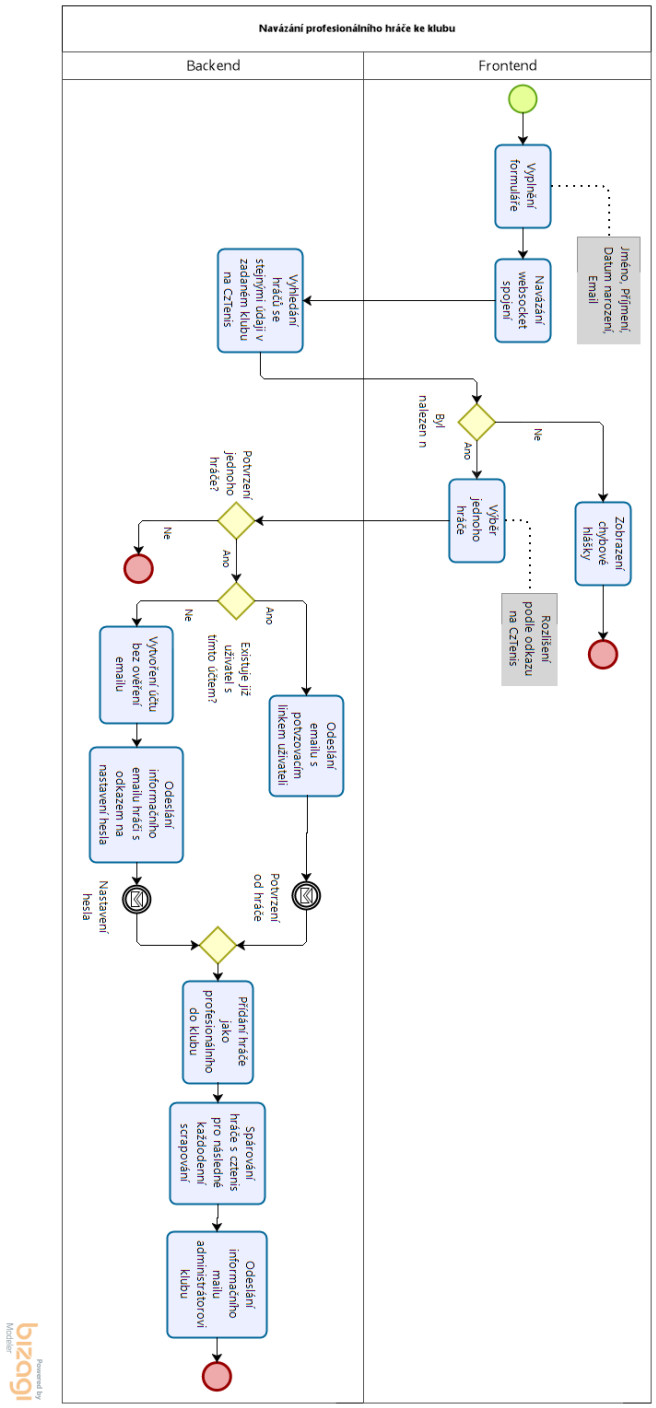
Stav v TK21	Akce ČTS	Popis řešení
Nalezen	Smazán/Zrušen	Soutěž se smaže z databáze TK21
Nalezen	Změna údajů	Všechny změněné údaje se aktualizují
Nenalezen	Přidání	Soutěž je přidána do TK21

Tabulka 3.4: Situace při scrapingu soutěží družstev a jejich řešení

Jednotlivé týmy, zápasy a hráči patřící do soupisek týmů se při každé iteraci nahradí nově nalezenými daty.

3.3.2 Navázání závodního hráče ke klubu

Jednou z velmi důležitých částí je propojení hráče s ČTS. Tento proces se zdá být komplikovaný. Pro správné fungování je důležité, aby obě strany vývoje (frontend a backend) správně pochopily tok procesu.



Obrázek 3.3: Proces propojení hráče s Českým tenisovým svazem

3.4 Nástroje

Pro návrh Případů užití a diagramu tříd byl použit webový program Creately[1] dostupný online a zdarma. Diagram nasazení byl vytvořen v programu DrawIO[2], který je volně dostupný v Google Drive. Pro návrh RESTového rozhraní jsme dali na doporučení vedoucího práce a použili Apiary[3].

3.4.1 Apiary

Jak již bylo zmíněno, tak pro návrh RESTového rozhraní byl použit software Apiary od firmy Oracle.

Apiary nabízí 2 značkovací jazyky: API Blueprint a Swagger. Pro tento projekt byl vybrán jazyk API Blueprint(viz Obrázek 3.4) z jednoho prostého důvodu, je přehlednější. Tento jazyk poskytuje veškeré potřebné nástroje, které by mohly být k dokumentaci HTTP dotazů a požadavků potřeba.

Apiary navíc generuje přehlednou, interaktivní a uživatelsky přívětivou dokumentaci, ve které je možné prokliknout každý endpoint a vidět jakýkoli dotaz/odpověď přehledně bez hledání v kódu se všemi atributy jako jsou HTTP kódy, potřebné hlavičky nebo GET parametry (viz Obrázek 3.5). Navíc toto neslouží pouze jako dokumentace, ale generuje i „fake endpointy“, jak pro klientskou část aplikace, tak i pro server. Je tedy možné různé endpointy testovat bez toho, aby byly implementovány na druhé straně.

Při používání tohoto programu nastalo několik problémů, které stojí za zmínku. Ze začátku se zdá vše přehledné, ale později, jak postupně přibývají další endpointy, kód začne být velmi nepřehledný. Nejspíše by pomohla možnost rozdělení do více souborů. Pomohla by i možnost psaní více stejných HTTP operací do stejné kolekce. Někdy je opravdu zbytečné vytvářet novou kolekci dotazů kvůli pouze jednomu endpointu. Důležitým bodem je absence automatického ukládání, což již není problém jazyku API Blueprint, ale Apiary. Je velmi frustrující smazat si hodinovou práci pouze tím, že zapomenete zmáčknout tlačítko v pravém horním rohu. Zároveň samotný interpreter někdy nefunguje optimálně. Někdy může být velmi pomalý a někdy se dokonce úplně ztrácí barvy označující syntaxi jazyka.

I přes nějaké problémy je Apiary dobrý nástroj. Je to velmi dobrá pomůcka pro dokumentaci RESTového rozhraní. Vývoj bez podobného softwaru by byl velice náročný.

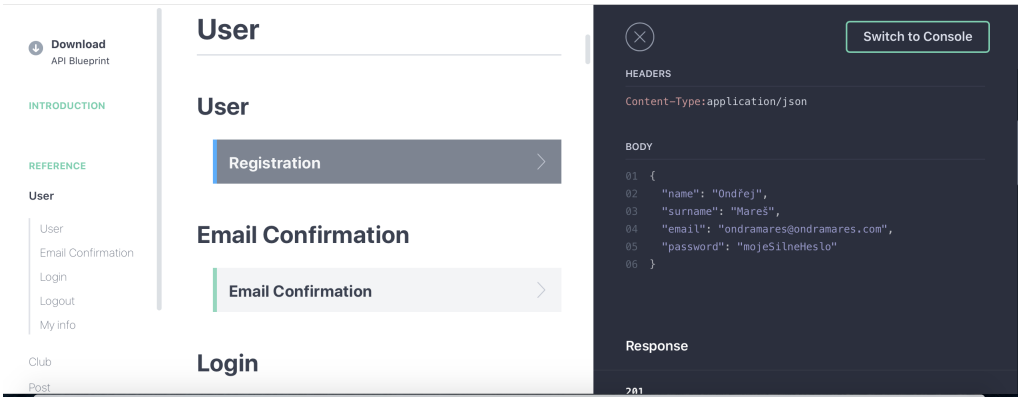
3.4.2 Bizagi Modeler

Bizagi Modeler je volně dostupný program pro mapování procesů v jazyce BPMN[4]. Program je velice intuitivní a lehký na používání. V tomto projektu byl použit na mapování složitějších procesů (viz Obrázek 3.3). Samotné namapování procesu není většinou složité. Velmi napomáhá pochopení toku informací a požadavků u vyvíjeného projektu.

[1] `https://creately.com/`
[2] `https://app.diagrams.net`
[3] `https://apiary.io`
[4] Business Process Model and Notation - jazyk pro modelování podnikových procesů

```
# Group User

## User [/api/user]

### Registration [POST]

+ Request (application/json)

        {
            "name": "Ondřej",
            "surname": "Mareš",
            "email": "ondramares@ondramares.com",
            "password": "mojeSilneHeslo"
        }

+ Response 201 (application/json)

        {
            "message": "Účet byl úspěšně vytvořen. Na uvedený email byl odeslán ověřovací link"
        }
```

Obrázek 3.4: API Blueprint



Obrázek 3.5: Apiary vygenerovaná dokumentace

Kapitola 4

Implementace

4.1 Použité technologie

Pro vývoj backendové části aplikace byla použita Java. Serverová část aplikace je vyvíjena ve frameworku Spring, konkrétněji Spring Boot 2, který byl představen v předmětu Enterprise aplikace ve třetím semestru a později i v několika semestrálních pracích.

Jako aplikační databáze je použita relační databáze PostgreSQL a to z několika důvodů. Prvním a velmi důležitým bodem je podpora ACID[1]. Spojování tabulek při dotazování by nemělo být příliš časté. Neměl by tedy nastat problém s rychlostí a zároveň neočekáváme příliš velký nárůst dat. Není tedy potřeba dobrá škálovatelnost, která je dostupná u NoSQL databází. Jelikož konkrétní implementace PostgreSQL a MySQL mají při takto malé aplikaci skoro stejné vlastnosti [8], při výběru rozhodovala pouze osobní preference.

Pro komunikaci se serverem bylo vybíráno mezi REST rozhraním a GraphQL. Při výběru byly vzaty v potaz zkušenosti s vývojem REST rozhraním, zatímco s GraphQL žádné zkušenosti nemáme. Zároveň jsme chtěli mít pevně dané rozhraní, kde strukturu určuje server. Rozhodli jsme se tedy využívat pevně dané REST rozhraní.

Volba PaaS byla vcelku jednoznačná. Nemám žádné zkušenosti s tímto typem služeb, bylo potřeba vybrat něco jednoduchého, ale přesto použitelného v produkci. Aplikace nevyžaduje 100% dostupnost, není tedy nutné mít ji spuštěnou na více serverech. Těmto požadavkům nejvíce vyhovuje Dokku.

Pro web scraping existuje plno možných knihoven v jazyce Java. Stránky ČTS obsahují pouze HTML a CSS bez přidaného Javascriptu. Bylo nutné, aby použitá knihovna uměla velmi dobře pracovat s HTML DOM stromem a podporovala všechny CSS[2] selektory. Knihovna JSoup je zde jasným vítězem, jelikož tyto požadavky splňuje a poskytuje vývojářům velmi jednoduché API.

[1] Atomicita(A), Konzistence(C), Izolovanost(I), Trvalost(D)
[2] Jazyk pro definování stylů v HTML stránce

4.2 Nástroje

4.2.1 IntelliJ IDEA

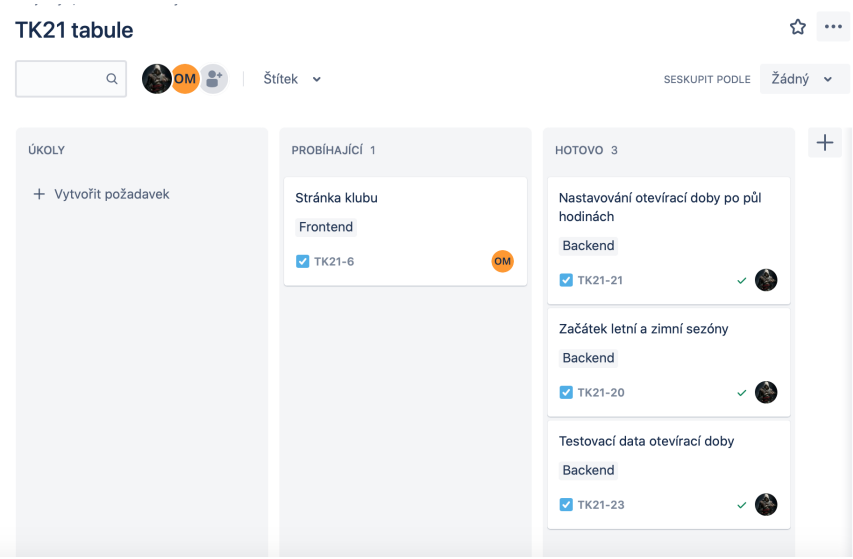
Jako vývojové prostředí byla použita IntelliJ IDEA, což je nejznámější produkt od české firmy JetBrains. Licenci zprostředkovává ČVUT. Není zde tedy potřeba platit žádné poplatky. IDEA obsahuje obrovské množství funkcí jako je podpora Springu, práci s databází, generování dokumentace a mnoho dalších užitečných funkcí.

4.2.2 Git

Jelikož aplikace je vyvíjena ve dvoučlenném týmu, je nutné mít aplikaci uloženou a zálohovanou v nějakém verzovacím systému. Není velkým překvapením, že jsme si vybrali Git. Konkrétněji je použit školní Gitlab jako vzdálený repozitář. Práce je rozdělena do dvou oddělených větví: backend a frontend. Tento postup se může zdát oproti dvěma rozdílným repozitářům nepraktický. Při každém nahrání na produkční server je vždy nutné zkopírovat obsah frontendové části do podsložky backendu zvané *webapp*. Kvůli tomuto postupu jsme se rozhodli pro sdílený repozitář. Jelikož pracujeme zcela odděleně, není žádný problém se slučováním s master větví, ve které jsou pouze hotové funkcionality.

4.2.3 JIRA

Přestože dvoučlenný tým není zrovna velký, i tak bylo po velmi krátké době zjištěno, že domlouvat se na všech úkolech v obyčejném chatu či osobně nestačí. Uchýlili jsme se tedy ke způsobu vývoje zvaném Kanban. Ve zkratce se jedná o způsob softwarového vývoje, kde jsou úkoly vyvěšené na veřejně přístupné tabuli rozdělené do tří kategorií: Úkoly, Probíhající a Hotovo. Pro tento účel využíváme volně dostupnou aplikaci od společnosti Atlassian jménem JIRA(viz Obrázek 4.1), který dle mého názoru splňuje všechny požadavky.



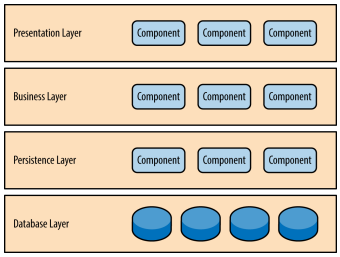
Obrázek 4.1: JIRA - Softwarový vývoj Kanban

4.2.4 Postman

Postman je aplikace pro testování REST rozhraní. Aplikace umožňuje jak manuální, tak automatické testování. HTTP dotazy se dají různě uspořádávat do kolekcí, což velmi zpřehledňuje práci u komplexnějších systémů. Samotné testování je velmi jednoduché. Design aplikace je velmi intuitivní a lze zde nastavovat vše, co si může člověk u HTTP dotazů jen přát.

4.3 Architektura

Architektura aplikace využívá tzv. vrstvy. Je logicky rozdělená do horizontálních vrstev, kde každá vrstva má svojí specifickou funkci. V tomto případě aplikace má čtyři vrstvy: prezentační, businessovou, persistenční a databázi. Prezentační vrstva je zodpovědná za přijímání a odesílání HTTP dotazů/odpovědí ve správném formátu. V business vrstvě probíhá veškerá logika spojená s businessovým modelem aplikace. Persistenční vrstva se stará o získávání a zapisování dat do databáze. Silnou stránkou tohoto stylu je rozdělení velkého celku do menších nezávislých částí. Je tedy lehčí aplikaci udržovat, testovat a vyvíjet.



Obrázek 4.2: Vrstvená architektura
Zdroj: *https://www.oreilly.com*

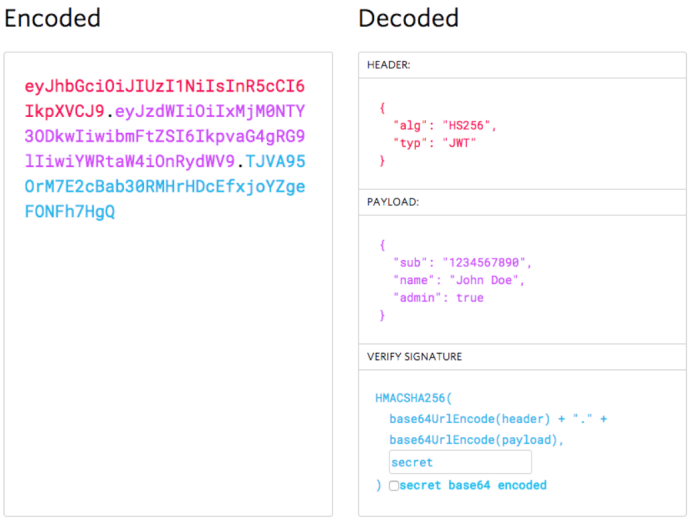
4.4 Přihlášení

Pro přihlášení uživatele jsou použity JWT neboli JSON Web Tokeny. JWT je standard, který definuje, jak posílat data mezi různými stranami bezpečně jako JSON objekt. Pro zabezpečení tohoto objektu lze použít 2 metody. Jednou je zašifrování a dešifrování pomocí skrytého tajemství na serveru pomocí algoritmu HMAC(tato varianta je aplikována v tomto projektu). Druhou možností je použití privátního a veřejného klíče.[2]

Samotný JSON objekt je možné si představit jako zašifrovaný řetězec rozdělený na tři části: hlavičku, obsah a podpis. Hlavička obsahuje použitý algoritmus pro šifrování a typ tokenu, který je v našem případě JWT. Část obsah obsahuje všechny důležité informace, které server potřebuje k autentizaci a autorizaci uživatele např. jméno, email atd. Poslední částí je podpis, který slouží pro kontrolu, jestli nebyl token při cestě nijak změněn.

Jak tedy ale samotné JWT používat? Při úspěšné autentizaci uživatele pošle server v odpovědi hlavičku Set-Cookie s tokenem a příznakem http-only. Tato hlavička automaticky

nastaví v prohlížeči cookie s příznakem http-only, který zamezí přístupu Javascriptu z bezpečnostních důvodů. Nastavená cookie se následně posílá s každým požadavkem na server, dokud se uživatel neodhlásí nebo nevyprší platnost tokenu.



Obrázek 4.3: Ukázka šifrovaného a dešifrovaného JWT
Zdroj: *https://jwt.io/introduction/*

4.5 Bezpečnost

Bezpečnost webových aplikací je v dnešní době velmi důležitou součástí každodenního života. Nejedná se pouze o soukromí uživatele, ale i možné ukradení identity nebo změnu vyžádané stránky během jejího přenosu.

4.5.1 HTTPS

V současné době se šifruje přenos dat na transportní vrstvě pomocí protokolu HTTPS využívající asymetrické a symetrické šifry. Pro správné fungování je nutné, aby server při navázání spojení poslal klientovi platný certifikát vydaný ověřenou certifikační autoritou.

Tato aplikace samozřejmě používá protokol HTTPS. V případě, že se klient dotáže na nezabezpečenou verzi, neboli v prohlížeči napíše *http://..* je automaticky přesměrován na zabezpečenou verzi aplikace. Certifikát je vydaný certifikační autoritou Let's Encrypt[3], která umožňuje rychlé, automatické a bezplatné získání certifikátu ke kterékoli doméně.

4.5.2 SQL Injection

SQL Injection je typ útoku, kdy uživatel do vstupu, můžeme si představit např. registraci, místo platných dat vyplní SQL příkaz. Pokud tento vstup od uživatele není správně ošetřen,

[3] `https://letsencrypt.org`

může dojít k fatální ztrátě či úpravě dat. V následujícím případě, pokud uživatel zadá jméno „*' ';DROP ALL TABLES;*“, tak se smažou všechny tabulky.

```
String jpql = "SELECT c FROM Club c WHERE c.name = " + name;
TypedQuery<Club> query = em.createQuery(jpql, Club.class);
```

V případě použití JPA existuje několik možností. V této práci je využívána třída PreparedStatement, která automaticky ošetří zadané parametry. Konkrétněji je použita metoda *createQuery()* v entity manageru, která třídu PreparedStatement využívá.

```
String jpql = "SELECT c FROM Club c WHERE c.name = :name";
TypedQuery<Club> q = em.createQuery(jpql, Club.class)
    .setParameter("name", name);
```

4.5.3 Cross-site scripting

Cross-site scripting(XSS) je útok, kdy uživatel do vstupního pole vyplní kód, nejčastěji <script>...</script>, a ten se následně provede na klientském zařízení. V této práci se o ochranu stará klientská část. Na serveru se tedy žádná kontrola neprovádí, pouze se ukládá text v databázi.

4.5.4 Cross-site request forgery

Cross-site request forgery(CSRF) je typ útoku, kde útočník spoléhá na to, že uživatel je přihlášen do aplikace pomocí cookie, kterou má uloženou v prohlížeči. V tom případě stačí, aby uživatel otevřel útočníkovu stránku, kde je např. v obrázku vložen odkaz na naší aplikaci. V takovém případě se dotaz pošle a bude vypadat, jako kdyby přišel od přihlášeného uživatele.

Tato vyvinutá aplikace, jelikož používá pro přihlášeni JWT uložený v cookie, je náchylná na tento typ útoku. Řešení spočívá v posílání speciálního tokenu vždy při přihlášení uživatele. K tomuto tokenu má potom tedy přístup pouze klientská část aplikace a útočník nemá možnost tento token zjistit. Dotaz na server se tedy provede pouze v případě, když se CSRF token poslaný v hlavičce shoduje s tokenem patřícímu danému uživateli.

4.6 Model

Pro vytvoření databázového modelu viz Obrázek 3.1 a pracováním s ním je používáno JPA, konkrétněji implementace EclipseLink. Generování tabulek je blíže specifikováno typem *create-or-extend-tables*, což znamená, že pokud databázové schéma neexistuje, automaticky se vytvoří. V opačném případě zůstane stejné nebo se ho JPA pokusí upravit tak, aby sedělo do nově definovaného modelu.

4.6.1 Řešené problémy

Při práci s JPA se vyskytly dva zajímavé problémy, které musely být řešené předěláním modelu. Oba problémy se chovaly podobným způsobem. Při načítání dat z interní cache vše fungovalo v pořádku. Ovšem v případě načítání požadovaných dat z databáze přestala aplikace fungovat kvůli špatnému formátu dat.

4.6.1.1 Serializace/Deserializace

Tato situace nastávala při ukládání a načítání objektu OpeningHours. Tato třída obsahovala dvě mapy popisující konkrétní otevírací dobu pro každý den. Jako klíč byly brány dny nebo datumy a jako hodnota byl vždy objekt FromToTime, který označoval časy od a do.

```
@Entity
public class OpeningHours extends AbstractEntity {

    @ElementCollection
    @Column(length = 10000)
    MapKeyEnumerated(EnumType.STRING)
    private Map<Day, FromToTime> openingHours;

    @ElementCollection
    @Column(length = 10000)
    private Map<LocalDate, FromToTime> specialDays;
}
public class FromToTime implements Serializable {
    private LocalTime from;
    private LocalTime to;
}
```

Takto definovaný model ukládal data jako serializovaný objekt do databáze ve sloupci typu Varchar. Bohužel JPA nebylo schopné u některých objektů, přestože správně serializovaných, je deserializovat a vyhazovalo chyby. Řešením bylo použití tzv. embedded objektů.

```
@Entity
public class OpeningHours extends AbstractEntity {

    @ElementCollection
    @CollectionTable(name = "OPENING_HOURS")
    @MapKeyEnumerated(EnumType.STRING)
    private Map<Days, FromToTime> openingHours;

    @ElementCollection
    @CollectionTable(name = "SPECIAL_DAYS")
    @MapKeyColumn(columnDefinition = "DATE")
    private Map<LocalDate, FromToTime> specialDays;
}

@Embeddable
public class FromToTime {
    @Column(name = "FROM_TIME", columnDefinition = "TIME")
    private LocalTime from;

    @Column(name = "TO_TIME", columnDefinition = "TIME")
    private LocalTime to;
}
```

4.6.1.2 Mapa s enumerovanými klíči

Při použití již výše zmíněného řešení předchozího problému, vznikl nový. Mapa se dny jako klíče označující regulérní otevírací dobu se do databáze neukládala správně. Vznikaly duplicitní klíče, někdy se mazaly bezdůvodně záznamy. JPA si ve zkratce dělalo, co chtělo. Po konzultaci s oponentem práce bylo zjištěno, že je to nejspíše chyba v aktuální verzi EclipseLink. Jako řešení se nabízí změna implementace JPA, nejspíše Hibernate. S tím ovšem přišlo plno dalších problémů, které se u EclipseLink neobjevovaly. Chyba nebyla nakonec vyřešena odstraněním EcliseLink, ale odstraněním mapy a vytvořením samostatného atributu pro každý den.

4.7 Websocketové spojení

Websocket je technologie umožňující obousměrnou komunikaci mezi klientem a serverem bez nutnosti čekání na dotaz od klienta. V této práci je tato technologie použita při implementaci rezervací a propojení závodního hráče s ČTS. U rezervací byl důvod použití především rychlost a automatická aktualizace rezervací. U druhého případu bylo websocketové spojení použito hlavně pro udržení stavu při komunikaci.

Pro samotnou implementaci websocketového spojení byl vytvořen potomek třídy TextWebsocketHandler, která poskytuje základní metody pro přijímání a posílání zpráv. Dle specifikace[10] by měl server sám posílat na transportní vrstvě tzv. heartbeat zprávy. Toto se bohužel nepovedlo zprovoznit. Řešení spočívalo v implementaci vlastních heartbeat zpráv na aplikační vrstvě. Klient v pevně daných intervalech pošle zprávu ve tvaru -*h*-, na kterou server stejným způsobem okamžitě odpoví.

4.8 AOP

Aspektově orientované programování(AOP) je styl programování, který umožňuje oddělit implementaci funkcionalit procházejících celým systémem. Zároveň umožňuje přidat funkcionalitu do již vytvořeného kódu. Takto můžeme přidat funkcionalitu do metod podle jejich jména, vstupních nebo návratových hodnot či pomocí anotací.[12] Samotný Spring z velké části funguje pomocí právě AOP.

V projektu TK21 se objevuje stále dokola jeden prvek a tím je kontrola práv uživatele, zda je správcem klubu. V případě nedostačujících práv je vyhozena výjimka. Tento případ je ideálním kandidátem na AOP. V následující ukázce je vytvořena vlastní anotace jménem ClubManagementOnly. Je použita pro označení metod, které mají být obohaceny o daný aspekt. Aspekt je prováděn vždy před vstupem do metody viz @Before a jako argument musí být vždy přítomna alespoň jedna instance třídy Club.

```
@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.METHOD)
public @interface ClubManagementOnly {
}

@Component
@Aspect
public class ClubManagementAspect {

    @Before("execution(@cz.cvut.fel.tk21.annotation.ClubManagementOnly * *(..,
        cz.cvut.fel.tk21.model.Club, ..)))")
    public void checkPermission(JoinPoint thisJoinPoint){
        for(Object arg : thisJoinPoint.getArgs()) {
            if (arg instanceof Club){
                if(!clubService.isCurrentUserAllowedToManageThisClub((Club) arg)) {
                    throw new UnauthorizedException("Pristup odepren");
                }
                break;
            }
        }
    }
}

@Service
public class ClubService extends BaseService<ClubDao, Club> {
    ...
    @Transactional
    @ClubManagementOnly
    public void removeCourt(Club club, TennisCourt tennisCourt){
        club.removeCourt(tennisCourt);
        this.update(club);
    }
    ...
}
```

4.9 Web scraping

Pro web scraping je používána knihovna JSoup. Tato knihovna exceluje v práci s HTML DOM stromem a ve využití CSS selektorů. Je tedy velmi jednoduché získat jakýkoli element ze stránky viz ukázka níže.

```
Document doc = Jsoup.connect(url).get();
Element clubTableBody = doc.select("table tbody").first();
```

Problémem při procházení mnoha stránek najednou může být omezení serveru na počet dotazů za sekundu. Při analýze bylo zjištěno, že toto omezení je u webových stránek ČTS aplikováno pouze při POST dotazech. Je tedy nutné při více POST dotazech pozastavit běh programu na alespoň jednu sekundu. Přestože limit není nastaven na GET, může se stát, že i při obyčejném GET dotazu server odmítne poslat validní odpověď. Z takového důvodu

je potřeba při neúspěšném dotazu pokus opakovat. V tomto případě byl zvolen maximální počet opakování třikrát. Při neúspěšném připojení je daná stránka přeskočena.

```
public Document connectGETWithRetries(String link){
    Document doc = null;
    int i = 0;
    while( i < 3){
        try {
            doc = Jsoup.connect(link).get();
            break;
        } catch (IOException ignored) {
        }
        //Wait for 500 ms
        try {
            Thread.sleep(500);
        } catch (InterruptedException ignored) {
        }
        i++;
    }
    return doc;
}
```

4.10 Emaily

Velmi důležitou součástí pro každou webovou aplikaci je posílání emailů. Spring v tomto ohledu umožňuje velmi jednoduché pracování s emaily. Pro správné fungování je nutné se připojit na SMTP server a poté pouze pomocí třídy *JavaMailSender* odesílat emaily. Emaily se mohou posílat v několika formátech. Zde je použit klasický formát HTML, kdy je pro vložení proměnných použit šablonovací jazyk Thymeleaf.[4]

4.11 Cron

V této aplikace je nutné použití časově řízených událostí. Spring opět toto velmi zjednodušuje. Obsahuje totiž balíček scheduling, kde je vše potřebné implementováno. Stačí tedy nad metodu, kterou chceme vykonávat pravidelně přidat anotaci *@Scheduled()*. Jako parametr přidáme tzv. cron výraz, specifikující, kdy přesně se má daná metoda provést. Příklad níže se vykoná každý rok, každý měsíc, každý den ve 3 hodiny 0 minut a 0 sekund.[3]

```
@Scheduled(cron = "0 0 3 * * *")
public void run(){
    //code here...
}
```

Nečekaným problémem při použití scheduled balíčku v kombinaci s websockety je *TaskScheduler*. Z neznámého důvodu tyto dvě funkce nejsou schopny sdílet defaultně vytvořený plánovač. Je nutné proto vytvořit vlastní plánovač.

[4] `https://www.thymeleaf.org`

```
@Bean
public TaskScheduler taskScheduler() {
    ThreadPoolTaskScheduler scheduler = new ThreadPoolTaskScheduler();
    scheduler.setPoolSize(5);
    scheduler.setThreadNamePrefix("scheduled-task-");
    scheduler.setDaemon(true);
    return scheduler;
}
```

4.12 Nasazení

Aplikace je dostupná na adrese `https://www.tk21.cz`. Jedná se o virtuální server koupený u společnosti Forpsi INTERNET CZ, a. s. s operačním systémem Linux - Ubuntu 18.

Jak již bylo zmíněno v předchozích kapitolách, je používáno Dokku. Jedná se o volně dostupnou platformu(PaaS) založenou na platformě Docker, která se stará o „postavení“ aplikace a její životní cyklus. Díky Dokku je nasazení na server velmi jednoduché. Po spuštění serveru je nutné nainstalovat samotné Dokku. Pro správné fungování je nutné, aby Dokku znalo váš veřejný klíč z důvodu bezpečného nahrávání aplikace na server. V Dokku poté stačí vytvořit instanci aplikace, doinstalovat jedním příkazem databázi, spojit ji s aplikací a vše je vytvořeno. Velkou výhodu při samotném vývoji je samotné nasazení. Nasazení trvá jen několik desítek sekund a to na způsob gitu. Dokku se chová jako vzdálený repozitář. Stačí tedy v našem případě Spring aplikace jen „pushnout“ na server a Dokku samo doinstaluje všechny potřebné balíčky, aplikaci postaví a spustí. Dle mého názoru je toto velmi jednoduchý a efektivní způsob nasazování softwaru hlavně při takto malých projektech.

4.13 Logování

Především pro produkci je logování nejenom chyb velmi důležité. Umožňuje vývojářům snadno odhalit příčinu chyby a následně ji opravit. Spring má v sobě již zabudovaný logovací systém, který je velmi účinný. Základní knihovna vestavěná ve Springu je Logback.[5] Přestože základní konfigurace je zpočátku dostačující, pro potřeby produkce je nutné ji upravit. Stačí vytvořit konfigurační soubor jménem *logback-spring.xml*. Kromě logování do konzole bylo přidáno logování do souborů podle dní ve tvaru *${logging_path}/archived/log_%d{dd-MM-yyyy}.log*. Lokace ukládaných souborů je definována v Enviroment proměnné[6]. Logovací soubory jsou archivovány po dobu 10 dnů a poté se automaticky smažou.

4.14 Datové úložiště

Jelikož u klubových příspěvků lze nahrávat obrázky typu JPG, PNG nebo GIF, je nutné mít na serveru implementované datové úložiště. S tím přichází mnoho problému, které ovšem

[5] `http://logback.qos.ch`
[6] Konfigurační proměnná definovaná mimo aplikaci

nejsou neřešitelné. Dokku obsahuje plugin pro jednoduché vytvoření datového úložiště. Vytvoří Docker kontainer, který následně propojí s požadovanou aplikací. Poté stačí pouze definovat Enviroment proměnnou, která určuje cestu k souborům. Jelikož je úložiště vytvořené jako Docker kontainer nebyl by v případě potřeby problém přesunout úložiště například na nějaké cloudové úložiště.

4.14.1 Komprese

Maximální dovolená velikost nahrávaného souboru jsou 2MB. Jako vývojáři chceme, aby obrázky zabíraly co nejméně možného místa na disku, je potřeba obrázky nadále komprimovat bez ztráty kvality. Pro tento účel je v aplikaci využita služba TinyPNG[7]. Tato firma zprostředkovává API pro nejenom komprese, ale i zmenšení či úpravy obrázků. Měsíčně poskytuje 500 kompresí zdarma. V případě, že je tento limit přesažen, aplikace nahraje obrázek bez komprese.

4.14.2 Zmenšení

Pro nahrání obrázků je definována minimální šířka obrázku 280 px. Zároveň musí platit, že poměr stran obrázku je mezi 0,1 a 10. Pokud nahraný obrázek má šířku více jak 2000 px je automaticky zmenšen na šířku 2000 px se zachováním poměru stran. Jelikož jsou obrázky zobrazovány na stěně klubu či uživatele, není praktické zobrazovat velké obrázky. Z tohoto důvodu se pro každý nahraný obrázek zároveň vytvoří i miniatura se šířkou 280 pixelů se zachovaným poměrem stran. Pro vytváření miniatur se ukázalo, že vestavěné Java knihovny velmi zhoršují kvalitu obrázku. Bylo tedy nutné využít jinou dostupnou knihovnu. Nejlépe vypadala volně dostupná knihovna Thumbnailator[8], která je zároveň velmi jednoduchá na používání.

4.14.3 Problém s formátem GIF

Při práci s obrázky nastal zajímavý problém, který se zdá být chybou v třídě ImageIO v balíčku javax viz odkaz[9]. Při načítání některých GIFů z paměti vyhodí zmíněná třída chybu *java.lang.ArrayIndexOutOfBoundsException: 4096*. U většiny nahrávaných GIFů tato chyba nenastává a zároveň tento typ obrázku není příliš populární. Bylo tedy rozhodnuto tuto chybu neřešit a nechat tento formát povolený.

[7] `https://tinypng.com`
[8] `https://github.com/coobird/thumbnailator`
[9] `https://bugs.openjdk.java.net/browse/JDK-7132728`

Kapitola 5

Testování

5.1 Jednotkové testování

Jednotkové testování by mělo být součástí každého softwarového projektu. Jedná se o metodu, kde jsou testovány výstupy jednotlivých funkcí při zadaném vstupu. Spring poskytuje podporu pro tento typ testování v balíčku *spring-test*. I pro testování je nutné používat databázi, nabízí se nám několik možností. Dobrou volbou je in-memory databáze H2, která se po provedení testu vždy resetuje. Každou testovací třídu využívající Spring je nutné označit anotací *@SpringBootTest*, která v sobě již obsahuje potřebnou anotaci *@ExtendWith(SpringExtension.class)*, která propojí Spring s JUnit. JUnit je velmi polulární knihovna pro jednotkové testování.[3] V tomto projektu je používána verze JUnit5. Samotnou testovací metodu je nutné označit pomocí anotace *@Test*.

```
@SpringBootTest
public class PostServiceTest {

    @Autowired
    private PostService service;
    ...
    @Test
    public void createPostWithoutName_Fail() {
        //test code here...
    }
    ...
}
```

5.2 Uživatelské testování

Uživatelské testovaní provádějí samotní uživatelé aplikace, kterým je poskytnut pouze testovací scénář s obecnými úkoly viz příloha B. Uživatel tedy musí na vše přijít sám. Tímto způsobem se rychle odhalí nepřehledné, neintuitivní ovládání nebo chybějící funkcionality.

Aplikace TK21 byla testována v Tenisové akademii Březno u Loun. Jedná se o menší klub se čtyřmi tenisovými kurty. Klub se každoročně účastní severočeské soutěže družstev.

Z důvodu vyhlášení nouzového stavu způsobeného nemocí COVID-19 bylo testování zahájeno začátkem května 2020. Turnaje i soutěže družstev byly zrušeny, proto tyto části aplikace nebylo možné otestovat v reálném provozu. Výsledky testování jsou tedy orientované spíše na správu klubu a rezervaci kurtů.

5.2.1 Nalezené chyby

Popis chyby	Nelze měnit název klubu
Kde	Serverová část aplikace
Detail	Při změně názvu klubu vrátí server stavový kód 500[1]. Chyba způsobená hodnotou null v proměnné jméno.
Závažnost	3/5*
Řešení	Oprava mapování DTO[2]objektu do entity

Popis chyby	Pád aplikace po změně otevírací doby
Kde	Serverová i klientská část aplikace
Detail	Při uložení nové otevírací doby do databáze vznikají nekorektní záznamy viz sekce 4.6.1.2. Nekorektní záznamy způsobí pád klientské části aplikace.
Závažnost	5/5*
Řešení	Předělání modelu otevíracích hodin a kontrola v klientské části

Popis chyby	Pád aplikace při zobrazení soutěže družstev
Kde	Klientská část aplikace
Detail	Není-li k dispozici datum konání alespoň u jednoho zápasu, dojde k pádu klientské části aplikace
Závažnost	4/5*
Řešení	Oprava na klientské straně

* 1-malá, 5-kritická

[1] Internal Server Error
[2] Data Transfer Object

5.2.2 Chybějící funkcionality

5.2.2.1 Telefon v rezervacích

Při rezervaci tenisových kurtů se ukázala emailová adresa jako nedostačující kontakt na zákazníka. Klub vyžaduje telefonní číslo pro rychlé oznámení nečekaných událostí.

Email musí být stále zachován pro zasílání rekapitulací rezervací. Řešením tedy není email nahradit, ale přidat telefonní číslo jako další prvek ke každé rezervaci. Administrátor klubu by měl mít možnost nastavit povinnost zadávat telefonní číslo.

5.2.2.2 Vytváření cizích rezervací administrátorem

Při testování klubu chyběla možnost rezervace kurtů pro jiného uživatele přes účet administrátora. V současné situaci administrátor může rezervovat kurt pouze na svůj účet. Není to ideální hlavně pro zjištění, kdo si tenisový kurt opravdu objednal. Aplikace by tedy měla umožnit administrátorovi klubu rezervaci na cizí jméno, email a telefon.

5.2.2.3 Detail příspěvku

Jednotlivé klubové příspěvky nemají vlastní stránku s detailem, což se ukázalo jako nedostačující. V současné situaci není možné sdílet jednotlivé příspěvky. Je proto nutné, je vždy najít v seznamu, který může být velmi dlouhý. Každý klubový příspěvek by tedy měl mít vlastní detail s unikátním URL.

Kapitola 6

Závěr

Tato práce od začátku nebyla zamýšlena pouze jako podklad k bakalářské práci, ale jako aplikace, která bude mít využití v reálném provozu. Po představení aplikace několika klubům jsme získali smíšenou reakci. Dva kluby měly na aplikaci kladný názor, jeden nechtěl aplikaci ani vyzkoušet. Tím se potvrzuje, že aplikace není dokonalá, ale využití v praxi je reálné.

Splnil jsem vše, co bylo definováno v požadavcích na začátku práce a navíc přibyly další vlastnosti aplikace, které nebyly plánované. Pokud by aplikace našla reálné využití, nejspíše by se objevily další nedostatky a chybějící funkcionality. Na této práci oceňuji především využití nových technologií a získání zkušeností ve vývoji backend aplikací.

Aplikace TK21 - Tenisový klub 21.století je dostupná na adrese `https://www.tk21.cz` plně funkční. Jakýkoli tenisový klub v České republice ji může bezplatně využívat.

Literatura

[1] *8 Most Popular Java Web Crawling & Scraping Libraries* [online]. 2020. [cit. 1. 5. 2020]. Dostupné z: https://mobilemonitoringsolutions.com/8-most-popular-java-web-crawling-scraping-libraries/.

[2] AUTH0. *JSON Web Token Introduction* [online]. 2020. [cit. 1. 5. 2020]. Dostupné z: https://jwt.io/introduction/.

[3] BAELDUNG. *Java, Spring and Web Development* [online]. 2020. [cit. 3. 5. 2020]. Dostupné z: https://www.baeldung.com.

[4] BREAKER, D. *SOAP vs. REST APIs – Which Reigns Supreme?* [online]. 2020. [cit. 30. 4. 2020]. Dostupné z: https://blog.dreamfactory.com/soap-vs-rest-apis-understand-the-key-differences/.

[5] BROSH, T. A. *How to Choose Right Database* [online]. 2020. [cit. 30. 4. 2020]. Dostupné z: https://towardsdatascience.com/how-to-choose-the-right-database-afcf95541741.

[6] EDUCBA. *Difference Between Java EE and Spring* [online]. 2020. [cit. 30. 4. 2020]. Dostupné z: https://www.educba.com/java-ee-vs-spring/.

[7] GRAPHQL. *GraphQL* [online]. 2020. [cit. 30. 4. 2020]. Dostupné z: https://graphql.org.

[8] HRISTOZOV, K. *MySQL vs PostgreSQL – Choose the Right Database for Your Project Libraries* [online]. 2020. [cit. 4. 5. 2020]. Dostupné z: https://developer.okta.com/blog/2019/07/19/mysql-vs-postgres.

[9] MOTROC, G. *The State of API Integration: SOAP vs. REST, public APIs and more* [online]. 2020. [cit. 30. 4. 2020]. Dostupné z: https://jaxenter.com/state-of-api-integration-report-136342.html.

[10] MOZZILA. *Wriring Websocket Servers* [online]. 2020. [cit. 1. 5. 2020]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API/Writing_WebSocket_servers.

[11] RESTFULAPI. *What is REST* [online]. 2020. [cit. 7. 5. 2020]. Dostupné z: https://restfulapi.net.

[12] SPRING. *Aspect Oriented Programming* [online]. 2020. [cit. 3. 5. 2020]. Dostupné z: `https://docs.spring.io/spring/docs/2.5.x/reference/aop.html`.

[13] VERMEER, B. *Spring dominates the Java ecosystem with 60% using it for their main applications* [online]. 2020. [cit. 14. 5. 2020]. Dostupné z: `https://snyk.io/blog/spring-dominates-the-java-ecosystem-with-60-using-it-for-their-main-applications/`.

[14] VIRAY, J. C. *PAAS Comparison* [online]. 2020. [cit. 30. 4. 2020]. Dostupné z: `http://www.jancarloviray.com/blog/paas-comparison-2017-dokku-flynn-deis-kubernetes-docker-swarm/`.

[15] ČTS. *Adresář tenisových klubů* [online]. 2020. [cit. 28. 4. 2020]. Dostupné z: `http://www.cztenis.cz/adresar-klubu`.

Příloha A

Seznam použitých zkratek

DOM Document Object Model

HTML Hypertext Markup Language

JPA Java Persistance API

JSON JavaScript Object Notation

XML Extensible Markup Language

ČTS Český tenisový svaz

Příloha B

Testovací scénář

Vyzkoušejte postupně každý z následujících bodů a připište k němu, zda a jak bylo složité splnit úkol. Nebudete-li si vůbec vědět rady, obraťte se na Ondřeje Mareše (mareson3@fel.cvut.cz).

1. **Registrujte Váš klub.**
   „Registrace klubu jednoduchá, srozumitelná.“

2. **Nastavte tenisové kurty ve Vašem klubu – název např. Centr, 1, 2, V lese apod. (nepište slovo kurt do názvu, používejte výhradně číslovky).**
   „Založeni tenisových kurtů srozumitelné.“

3. **Nastavte otevírací dobu Vašeho klubu.**
   „Běžná otevírací doba nastavena a plně vyhovuje.“

4. **Nastavte sezóny v klubu, v následující sezóně neumožněte rezervace.**
   „Výhoda pro antukové kurty.“

5. **Zarezervujte si jakoukoliv hodinu na jakémkoliv kurtě.**
   „Reservace jednoduchá a jasná.“

6. **Vytvořte opakující se rezervaci – opakovat se bude jednou za 7 dní, po dobu alespoň 28 dní, tj. čtyřikrát.**
   „Při prvním pokusu provedení trochu nesrozumitelné, následně opakující se reservace proběhla v pořádku.“

7. **Přidejte závodního hráče – Ondřej Mareš, email: ondramares@ondramares.com, 22. 5. 1997.**
   „Líbilo se mi, že je možné přetáhnout informace z tenisového svazu a je to vlastně jednoduché zadat závodního hráče.“

8. **Vytvořte v aplikaci další uživatelský účet – na libovolné údaje. Následně se na něj přihlaste do aplikace, vyhledejte Váš klub a požádejte o členství v klubu.**
   „Uživatelský účet vytvořen, klub vyhledán a členství bylo přijato.“

9. **Odhlaste se a přihlaste se znovu na administrátorský účet. Nyní potvrďte žádost o členství, z předchozího bodu.**
   „V pořádku.“

10. **Nového člena klubu udělejte také administrátorem.**
    „Jednoduché, přehledné.“

11. **Přidejte příspěvek s obrázkem – zkuste obsah příspěvku naformátovat, tučně, kurzívou, vytvořte seznam – následně přidejte libovolný obrázek a příspěvek vytvořte.**
    „Zadání příspěvku jednoduché, tlačítko Přidat, spíše bych volila uložit. Pochopila jsem jako přidání nového příspěvku, to jsem nechtěla, tedy nepřidala a zároveň neuložila a příspěvek se nezobrazil.“

12. **Přidaný příspěvek odstraňte.**
    „Odstraněno.“

Uživatelský scénář byl vytvořen Ondřejem Marešem a vyplněn Tenisovou akademií Březno u Loun.