Univerzita Pardubice

Fakulta elektrotechniky a informatiky

Informační systém pro vizualizaci sportovních statistik

Bc. Tomáš Vaniš

Diplomová práce

2024

Univerzita Pardubice
Fakulta elektrotechniky a informatiky
Akademický rok: 2023/2024

ZADÁNÍ DIPLOMOVÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Bc. Tomáš Vaniš**
Osobní číslo:	**I22173**
Studijní program:	**N0613A140007 Informační technologie**
Téma práce:	**Informační systém pro vizualizaci sportovních statistik**
Zadávající katedra:	**Katedra softwarových technologií**

Zásady pro vypracování

Cílem diplomové práce je navrhnout a implementovat webový informační systém s primárním zaměřením na data vybraného sportu (např. fotbal). Uvažovaná data budou uložena v interní databázi systému, bude možné nad nimi realizovat základní statistické výpočty (např. vyhodnocení minimálních, maximálních, klouzavých hodnot jak za vybrané období, tak např. za celou sezónu) a takto vypočtená data bude možné vizuálně prezentovat např. v podobě tabulek, grafů apod.
Teoretická část práce se bude zabývat analýzou a návrhem aplikace společně s rozborem zvolených technologií ve kterých bude aplikace vytvořena (předpokladem je využití frameworku Spring Boot pro back-end část a knihovny React nebo jiných vhodných knihoven pro front-end část).
Praktická část práce bude obsahovat samotnou tvorbu systému, jeho implementaci a nasazení. Součástí řešení budou také jednotkové a integrační testy pro back-end část a také vybrané testy front-end části.

Rozsah pracovní zprávy: **cca 60 stran**
Rozsah grafických prací:
Forma zpracování diplomové práce: **tištěná/elektronická**

Seznam doporučené literatury:

1. ARLOW, Jim a NEUSTADT, Ila. UML 2 a unifikovaný proces vývoje aplikací: objektově orientovaná analýza a návrh prakticky. 2., aktualiz. a dopl. vyd. Brno: Computer Press, 2007. ISBN 978-80-251--1503-9.
2. LEONARD, Anghel. Spring Boot persistence best practices: optimize Java persistencce performance in Spring Boot Applications. [New York, NY]: Apress, [2020]. ISBN 978-1-4842-5625-1.

Vedoucí diplomové práce: **doc. Ing. Michael Bažant, Ph.D.**
Katedra softwarových technologií

Datum zadání diplomové práce: **8. listopadu 2023**
Termín odevzdání diplomové práce: **17. května 2024**

L.S.

Ing. Zdeněk Němec, Ph.D. v.r.
děkan

prof. Ing. Antonín Kavička, Ph.D. v.r.
vedoucí katedry

V Pardubicích dne 30. listopadu 2023

Prohlašuji:

Tuto práci jsem vypracoval samostatně. Veškeré literární prameny a informace, které jsem v práci využil, jsou uvedeny v seznamu použité literatury.

Byl jsem seznámen s tím, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorský zákon, zejména se skutečností, že Univerzita Pardubice má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona, a s tím, že pokud dojde k užití této práce mnou nebo bude poskytnuta licence o užití jinému subjektu, je Univerzita Pardubice oprávněna ode mne požadovat přiměřený příspěvek na úhradu nákladů, které na vytvoření díla vynaložila, a to podle okolností až do jejich skutečné výše.

Beru na vědomí, že v souladu s § 47b zákona č. 111/1998 Sb., o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších předpisů, a směrnicí Univerzity Pardubice č. 9/2012, bude práce zveřejněna v Univerzitní knihovně a prostřednictvím Digitální knihovny Univerzity Pardubice.

V Pardubicích dne 1. 8. 2024

Bc. Tomáš Vaniš

Poděkování

Poděkování patří panu doc. Ing. Michaelovi Bažantovi, Ph.D. za skvělou pomoc při tvorbě této diplomové práce. Poděkování dále patří všem, kteří mě při studiu podporovali.

ANOTACE

Cílem této práce je návrh a implementace webového informačního systému se zaměřením na sledování a zpracování statistických dat fotbalových soutěží. Aplikace bude automaticky načítat reálná data, která budou ukládána pro další zpracování. Důraz bude kladen zejména na data časových řad a možnosti jejich vzájemného porovnávání na základě uživatelsky zvoleného období. Teoretická část práce nabídne detailní analýzu a návrh celé aplikace včetně rozboru použitých technologií a knihoven, které budou při implementaci použity. Součástí bude také podrobný popis průběhu celé implementace. V praktické části bude následně celý systém naimplementován.

KLÍČOVÁ SLOVA

informační systém, webová aplikace, fotbal, statistiky, časové řady, API, Java, Spring Boot, JavaScript, React, MySQL

TITLE

Information system for visualization of sports statistics

ANNOTATION

The aim of this thesis is the design and implementation of a web information system focused on monitoring and processing statistical data of football competitions. The application will automatically load real data, which will be stored for further processing. Emphasis will be placed particularly on time series data and the ability to compare them based on a user-selected period. The theoretical part of the thesis will offer a detailed analysis and design of the entire application, including an examination of the technologies and libraries used in the implementation. It will also include a detailed description of the entire implementation process. In the practical part, the entire system will then be implemented.

KEYWORDS

information system, web application, football, statistics, time series, API, Java, Spring Boot, JavaScript, React, MySQL

OBSAH

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

SEZNAM ZKRATEK

API	Application Programming Interface
UML	Unified Modeling Language
ORM	Object Relational Mapping
ER	Entity Relationship
JSON	JavaScript Object Notation
JWT	JSON Web Token
UEFA	Union of European Football Associations
REST	Representational State Transfer
OOP	Object-oriented programming
XML	Extensible Markup Language
CSV	Comma-separated values
SQL	Structured Query Language
HTML	HyperText Markup Language
DOM	Document Object Model
SVG	Scalable Vector Graphics
YAML	YAML Ain't Markup Language
JPA	Java Persistence API
DTO	Data transfer object
CORS	Cross-Origin Resource Sharing

ÚVOD

Sportovní statistiky jsou bezpochyby klíčovým prvkem při hodnocení výkonů jak jednotlivých sportovců či týmů, tak i celých soutěží nebo zemí. Díky narůstajícímu množství dostupných dat a moderním technologiím, které umožňují jejich sběr a analýzu, se zvyšuje potřeba efektivních nástrojů pro jejich vizualizaci a interpretaci. V důsledku nárůstu těchto dat se zvyšuje potřeba efektivních nástrojů, které nám umožní tyto data co nejlépe analyzovat a následně efektivně vizualizovat. Tato diplomová práce je zaměřena na vytvoření informačního systému pro vizualizaci fotbalových statistik, který usnadní zpracování a zobrazení těchto dat tak, aby byla lépe srozumitelná pro všechny fanoušky tohoto všem dobře známého sportu.

Cílem této práce je navrhnout a implementovat informační systém, který dokáže sbírat, zpracovávat a následně vizualizovat sportovní statistiky z veřejně dostupných zdrojů. Systém bude zahrnovat nejen základní statistické údaje, ale i pokročilé metriky, které poskytují lepší nadhled o výkonu jednotlivých týmů. Hlavní důraz bude kladen na jednoduché a intuitivní uživatelské prostředí, které umožní snadnou orientaci v těchto datech. Data budou reprezentována jak na základě samotných hodnot a statistických výpočtů, tak i na základě časových řad v podobě grafů, které nám umožní tyto data jednoduše srovnávat s jinými týmy.

Práce je rozdělena do několika částí. Úvodní část se věnuje analýze požadavků, návrhu systému nebo také rozboru již existujících konkurenčních řešení, který nám umožní najít jejich nedostatky, které budou následně v tomto systému implementovány. Další kapitoly popisují metody a technologie použité při vývoji systému, včetně výběru vhodných vizualizačních nástrojů a datových zdrojů. Nebude chybět ani kapitola popisující proces implementace celého systému. Závěrečná část je zaměřena na testování systému, hodnocení dosažených výsledků a návrhy na jeho další rozvoj a zlepšení.

Tato práce usiluje nejen o přínos k rozvoji nástrojů pro analýzu sportovních statistik, ale také o nabídnutí praktického příkladu využití moderních technologií pro efektivní zpracování a prezentaci dat. Výsledný systém by měl usnadnit práci s daty v oblasti sportu a přispět k lepšímu porozumění a interpretaci sportovních výkonů.

Autor si zvolil toto téma na základě společného zájmu o sport a webové technologie. Oba tyto faktory se neustále rozšiřují a ovlivňují naše životy.

1 NÁVRH A ANALÝZA

Tato kapitola podrobně seznámí s procesem analýzy požadavků a celkového návrhu projektu. Analýza požadavků a návrh softwaru jsou klíčovými fázemi v životním cyklu vývoje softwaru, které se často souhrnně označují jako softwarové inženýrství. Tento krok je ve vývoji softwaru zásadním krokem k vytvoření kvalitních, spolehlivých, udržitelných a efektivních softwarových systémů a aplikací. Jde o proces, který zahrnuje nejen technické prvky, ale i metodologické, manažerské a marketingové postupy, které se vzájemně doplňují a jejich integrace dohromady zajišťuje konečný úspěch celého projektu.

Zaměříme se na základní principy a metody softwarového inženýrství, které tvoří základ efektivního návrhu softwaru. Budeme se věnovat různým modelům a analytickým technikám, jako je například detailní rozbor požadavků, diagram případů užití nebo také databázový model. Tyto položky nám umožní lépe a detailně pochopit požadavky a vlastnosti požadovaného systému. Jednak si obecně rozvedeme jednotlivé analytické položky a následně se zaměříme na konkrétní části projektu implementovaného v této práci.

1.1 UML

Na úvod by bylo dobré říct, co je to vlastně UML. Celým názvem Unified Modeling Language, česky Jednotný modelovací jazyk, je standardizovaný modelovací jazyk zahrnující soubor integrovaných diagramů. Byl vyvinut, aby pomáhal systémovým a softwarovým vývojářům při specifikaci, vizualizaci, vytváření a dokumentaci softwarových systémů a rovněž při modelování podnikových procesů a jiných systémů. UML zahrnuje osvědčené inženýrské postupy, které se osvědčily při modelování rozsáhlých a složitých systémů. Je klíčovým prvkem při vývoji objektově orientovaného softwaru a celého procesu vývoje softwaru. UML převážně využívá grafické notace k vyjádření návrhů softwarových projektů. Používání UML usnadňuje projektovým týmům komunikaci, zkoumání možných návrhů a ověřování architektonického návrhu softwaru.

1.2 Analýza požadavků

Jak již z názvu napovídá, tak se jedná o analýzu konkrétních požadavků na systém či aplikaci. Tyto požadavky obvykle definuje klient či zadavatel projektu na základě vlastních

potřeb. Analýza požadavků je proces, během kterého jsou shromažďovány a analyzovány všechny požadavky na systém od všech zúčastněných stran. Jde o klíčový krok pro zajištění toho, že vyvíjený systém bude plně naplňovat potřeby a očekávání uživatelů. Požadavky mohou vycházet například z textového zadání projektu nebo z různých schůzek s klienty, pro které je daný softwarový systém vyvíjen. Klíčové je plné porozumění požadavkům a potřebám klienta, abychom předešli jednak reklamacím ze strany klienta, tak zbytečnému vývoji nechtěných funkcionalit, které by celkový proces vývoje mohly opozdit a zdražit.

Požadavky dělíme na dvě skupiny. První skupinou jsou požadavky funkční, které určují, co všechno bude daný systém uživateli umožňovat a jaké bude obsahovat funkcionality. Druhou skupinou jsou pak požadavky nefunkční. Ty nám určují pasivní vlastnosti systému, kterými mohou být například zabezpečení systému, výkon, provozní kapacita, použité standardy, vzhled aplikace nebo také použité technologie, ve kterých bude daný systém vyvíjen.

Zápis požadavků je možný například do speciálních programů určených k návrhu systému, mezi které může patřit například program Enterprise Architect, ale lze je také zapsat jednoduše pomocí tabulky. Právě takovýto zápis bude použit v této práci.

Tabulka požadavků obsahuje vždy tři sloupce. Tyto sloupce tvoří ID, které slouží jako unikátní identifikátor každého požadavku, dále pak samotný název požadavku a nakonec akceptační kritérium, které obsahuje detailnější popis toho, co musí být splněno, aby byl daný požadavek naplněn.

1.2.1 Funkční požadavky

Funkční požadavky jsou rozděleny na dvě části, prioritní a rozšiřující. Prioritní požadavky jsou takové, které jsou naprosto nutné ke splnění projektu. Oproti tomu rozšiřující požadavky nejsou klíčové, ale slouží ke zlepšení celého systému a uživatelského prožitku.

a) Prioritní:

Tabulka 1: Prioritní funkční požadavky

ID	Název požadavku	Akceptační kritérium

1	Systém - Automatický sběr dat	Aplikace bude automaticky načítat aktuální data z odehraných zápasů z veřejného API. Data budou načítány vždy jednou za den a budou zpracovávány a ukládány do interní databáze systému.
2	Administrátor - Manuální editace zdrojových dat	Aplikace umožní administrátorovi systému upravovat veškerá zdrojová data v interní databázi systému. Bude mít možnost například přidávat a odebírat týmy, fotbalové ligy anebo přepínat sezóny po jejich skončení.
3	Zobrazení výsledků zápasů	Aplikace umožní každému uživateli zobrazit výsledky všech odehraných zápasů.
4	Zobrazení bodové tabulky	Aplikace umožní každému uživateli zobrazit bodovou tabulku dané soutěže.
5	Zobrazení detailních statistik ke každému zápasu	Aplikace umožní přihlášenému uživateli zobrazit detailní statistiky každého zápasu. Například střely na bránu, udělené karty, držení míče atd.
6	Zobrazení statistik za libovolné období včetně základních statistických ukazatelů	Aplikace umožní přihlášenému uživateli zobrazit detailní statistiky za libovolné období a provést nad nimi základní statistické výpočty.
7	Zobrazení v grafické podobě (grafy)	Aplikace umožní vypočtená data zobrazovat v grafické podobě pomocí grafů.
8	Registrace a přihlášení uživatele	Aplikace umožní registraci a přihlášení do systému.

Dalo by se říct, že prioritní požadavky odráží body ze zadání projektu. Mezi hlavní požadavky tedy spadá například sběr zdrojových dat, jejich následné uložení, zpracování a zobrazení uživateli v aplikaci v různých podobách. Mezi další prioritní požadavky patří ale také i body, které přímo nevychází ze zadání, ale i tak jsou potřebné pro funkčnost celého systému. Těmito body máme na mysli například možnost registrace a přihlášení uživatele do systému.

b) Rozšiřující:

Tabulka 2: Rozšiřující funkční požadavky

ID	**Název požadavku**	**Akceptační kritérium**
9	Obnovení zapomenutého hesla	Aplikace umožní registrovaným uživatelům obnovit zapomenuté heslo pomocí kódu zaslaného na daný email.

10	Možnost porovnání statistik dvou vybraných mužstev	Aplikace umožní přihlášenému uživateli porovnávat statistiky dvou vybraných mužstev.
11	Komentování jednotlivých zápasů v diskusní sekci	Aplikace umožní přihlášenému uživateli přidávat komentáře k odehraným zápasům.
12	Systém - Zasílání vybraných notifikací (např. emailem)	Aplikace bude umožňovat nastavení zasílání emailových notifikací. Například o odehraném zápasu sledovaného mužstva.
13	Administrátor - Možnost zablokování uživatele s nevhodnými komentáři	Administrátor aplikace bude mít možnost zablokovat uživatele s nevhodnými komentáři. Takto zablokovaný uživatel se bude moci nadále přihlásit, ale nebude mu již umožněno účastnit se diskusní sekce.

Jak již bylo řečeno, tak rozšiřující požadavky nejsou klíčové k naplnění funkčnosti celého systému. Rozšiřují nám ovšem systém o další funkcionality, které vylepší celkový uživatelský komfort. Například možnost obnovení zapomenutého hesla, systém notifikací či přidávání komentářů. Velice zajímavou rozšiřující funkcionalitou je také možnost porovnávat statistiky dvou vybraných mužstev. Lze porovnávat jak celkové statistiky, tak i samotné časové řady.

1.2.2 Nefunkční požadavky

Krom funkčních požadavků se v systému nachází i několik nefunkčních, které nám dále rozšíří kvalitu celého systému.

Tabulka 3: Nefunkční požadavky

ID	**Název požadavku**	**Akceptační kritérium**
1	Vícejazyčnost (čeština/angličtina)	Aplikace bude umožňovat přepínání mezi českým a anglickým jazykem. Výchozí jazyk bude rozpoznám na základě prohlížeče.
2	Responzivita aplikace (PC/mobile)	Aplikace bude responzivní a uživatelsky přívětivá pro zobrazování jak na PC, tak na mobilních zařízeních.
3	Světlý/tmavý režim aplikace	Aplikace bude umožňovat přepínání mezi světlým a tmavým režimem.

4	Bezpečnost aplikace a uživatelských dat	Aplikace bude zabezpečena oproti základním útokům, aby byla zaručena bezpečnost uživatelských dat.
5	Technologie serverové části aplikace - Spring Boot	Serverová část aplikace bude napsána pomocí frameworku Spring Boot.
6	Technologie klientské části aplikace - React	Klientská část aplikace bude napsána pomocí knihovny React.

Samozřejmostí je v dnešní době zajištění bezpečnosti celého systému, zejména pak uživatelských dat. Můžeme s jistotou říct, že dnes při vývoji jakékoliv aplikace či systému musíme věnovat bezpečnosti čím dál větší pozornost, protože útoků a zneužití systémů každým dnem přibývá. V dnešní době, kdy stále přibývá mobilních zařízení, se také standardem stává responzivita aplikací, abychom s nimi mohli snadno pracovat jak na osobním počítači, tak na mobilu. Běžně také již moderní aplikace podporují přepínání jazyků či světlého a tmavého režimu. Proto tomu byla v této práci také věnována pozornost. Posledními nefunkčními požadavky jsou technologie, ve kterých je systém vyvíjen. V tomto případě byl tento požadavek součástí zadání projektu.

1.3 Případy užití

Další velice důležitou položkou v návrhu systému je analýza případů užití, často také nalezneme pod anglickým názvem Use Case. Případy užití jsou vyznačeny pomocí speciálního diagramu, ve kterém vystupují aktéři (uživatelé) systému a jsou propojeni s jednotlivými funkcemi téhož systému. Tento diagram nám pomůže detailně porozumět tomu, kým a jak bude systém užíván, anebo například jaké mají v systému vystupovat role a oprávnění. Use Case diagramy jsou součástí standardu UML.

V diagramu jsou obvykle aktéři znázornění jako postavy či ikony a jednotlivé funkce systému jsou reprezentovány jako ovály nebo obdélníky. Aktéři (uživatelé) systému jsou propojeni s funkcemi, které v systému využívají. Toto spojení ukazuje, jakým způsobem budou uživatelé interagovat se systémem.

Diagramy případů užití nám poskytují detailní pohled na to, jak bude systém používán. To nám umožňuje lépe porozumět požadavkům uživatelů a zajistit, aby systém splňoval jejich potřeby a očekávání. Dále nám pomáhají identifikovat různé role, které budou v systému existovat, a jaká oprávnění budou jednotlivé role mít. Tato analýza je nezbytná pro

správný návrh a implementaci systému, protože bez ní by mohlo dojít k nedorozuměním ohledně požadavků a funkcionalit systému.

Když to tedy shrneme, tak mezi hlavní přínosy případů užití patří:

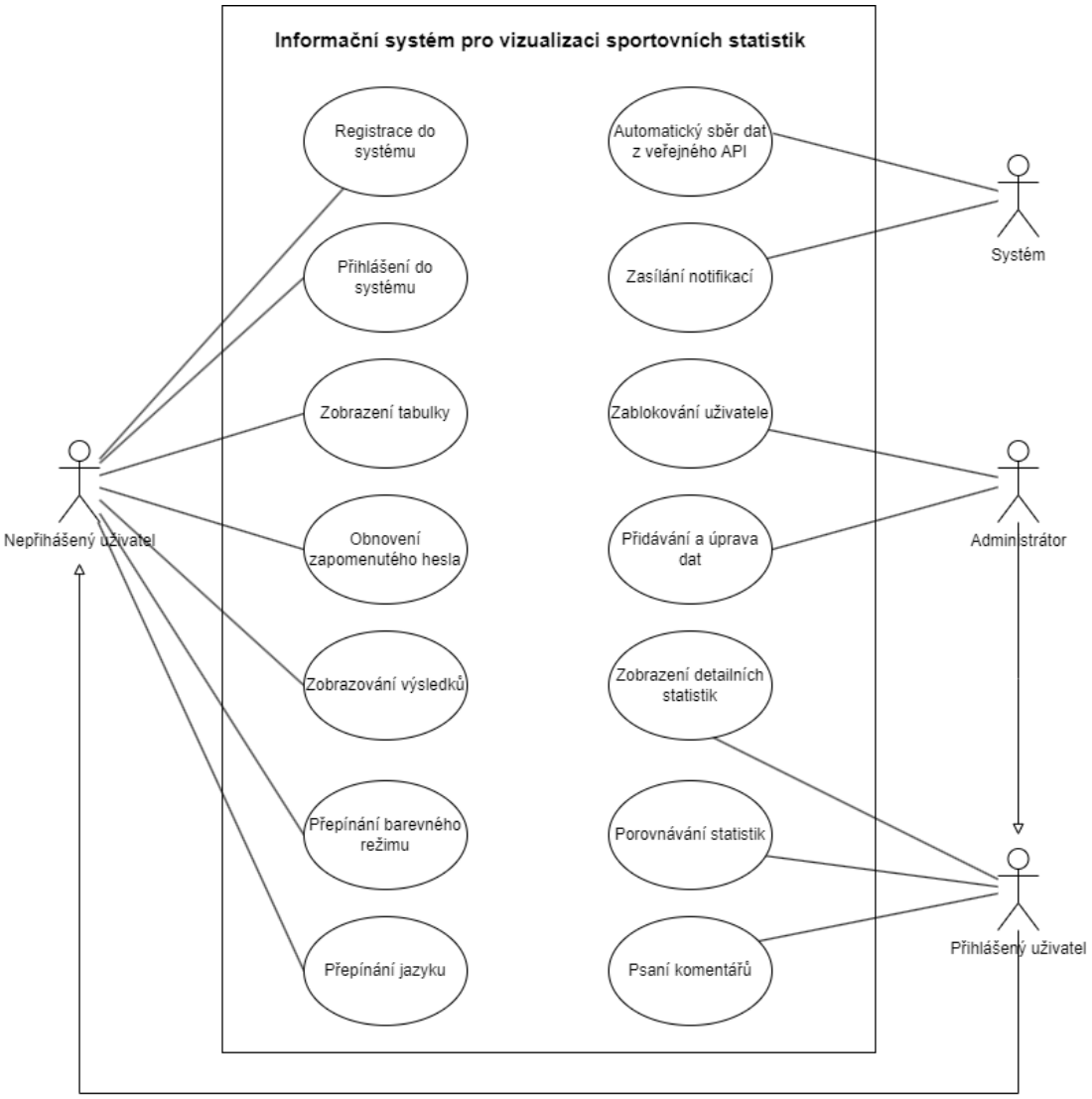
1. **Jasné definování požadavků:** Případy užití nám jasně specifikují, jaké funkce musí systém obsahovat.
2. **Lepší porozumění uživatelským potřebám:** Pomáhají vývojářům lépe pochopit, jakým způsobem uživatelé systém používají.
3. **Identifikace rolí a oprávnění:** Určují různé role v systému a jejich oprávnění, což je klíčové pro bezpečnost a správu systému.

Proces tvorby případů užití zahrnuje několik kroků. Prvním krokem je určení všech možných aktérů, kteří budou v systému figurovat. Druhým krokem následuje definice funkcionalit na základě požadavků. Jednotlivé funkcionality si vložíme do diagramu, kde každá funkcionalita bude reprezentována jako jeden ovál či obdélník. Třetím krokem je samotné přiřazení jednotlivých funkcionalit k patřičným aktérům. Mezi aktéry lze využít princip dědičnosti. Řekněme, že máme roli vedoucího a podřízeného. Vedoucí má stejné pravomoci jako podřízený plus něco navíc. V tomto případě nemusíme oba aktéry přiřazovat ke společným funkcionalitám, ale přiřadíme je pouze k podřízenému a u vedoucího šipkou vyznačíme, že dědí z možností podřízeného. Podobného principu lze využít i u jednotlivých funkcionalit. Jestliže jedna funkcionalita rozšiřuje druhou nebo je na ní přímo závislá, můžeme je vzájemně napojit a aktérovi pak přiřadit pouze základní funkcionalitu. Díky tomu můžeme opět zpřehlednit celý výsledný diagram. Dalším krokem v tvorbě diagramu je vyhledání možných výjimek, které mohou nastat. Například výjimky v již zmíněné dědičnosti. Tyto případy pak musíme vyřešit jednotlivě. Posledním krokem je vytvoření textové dokumentace ke každému případu užití.

1.3.1 Diagram případů užití

Nyní se podíváme na konkrétní diagram systému vyvíjeného v této práci.

Jak si lze všimnout, tak v systému figurují celkem 4 aktéři, nepřihlášený uživatel, přihlášený uživatel, administrátor systému a systém. Ovály s funkcionalitami pak přímo odráží požadavky stanovené na začátku projektu.



Obrázek 1: Diagram případů užití (Zdroj: Autor)

Jak již z názvu napovídá, tak nepřihlášený uživatel je uživatel, který zavítá do systému bez jakéhokoliv ověření. Tento uživatel má k dispozici pouze základní funkcionality, mezi které patří například zobrazení jednotlivých výsledků zápasů nebo bodové tabulky. Mezi další možnosti patří samozřejmě možnost registrace či následného přihlášení do systému. S tímto procesem souvisí i možnost obnovení zapomenutého hesla. Opomenout nelze ani možnost využívání funkčních tlačítek na přepínání barevného schématu aplikace či přepínání jazyku mezi angličtinou a češtinou.

Uživatel, který se do systému zaregistruje a následně přihlásí dostane přístup k rozšiřujícím a již mnohem zajímavějším funkcionalitám. Mezi hlavní patří samotné gró celého systému, a to přístup k detailním statistikám. Zobrazovat lze statistiky jednotlivě ke každému zápasu nebo jejich shrnutí na základě libovolně zvoleného období včetně časových

řad. Přihlášení uživatelé mají také přístup k porovnávači statistik, kde lze porovnávat statistiky dvou vybraných mužstev opět na základě libovolného období. Nelze opomenout možnost účasti v diskusní sekci pod každým zápasem. Tato sekce je pro nepřihlášené uživatele kompletně skryta. Poslední je možnost nastavování automatických notifikací. Uživatel si zvolí, který tým nebo týmy ho zajímají nejvíc a následně pokud nějaký z těchto týmu dohraje svůj zápas, tak systém zašle uživateli notifikaci a informuje ho o tom.

Posledním uživatelským aktérem je administrátor systému. Ten má na starosti úpravu zdrojových dat. Tím je na mysli to, že například na konci každé sezóny upraví týmy v dané soutěži na základě jejich podoby v nové sezóně. Například přidá týmy, které postoupili, a naopak odebere ty, které sestoupili. Dává také v aplikaci pokyn pro zahájení nové sezóny, aby systém mohl správně začít automaticky načítat data z veřejného API. Administrátor systému funguje také jako moderátor diskusní sekce. Může mazat nevhodné komentáře a popřípadě tyto uživatele zablokovat a znepřístupnit jim tak možnost psaní dalších komentářů.

Čtvrtým a celkově posledním aktérem je samotný systém. Ten se stará o automatické načítání zdrojových dat z veřejného API. Data načítá na základě zvoleného intervalu, aby data byla vždy co nejvíc aktuální. S tím souvisí i automatické zasílání notifikací. Pokaždé, když je načten nějaký zápas, tak systém zkontroluje, zda některý z hrajících týmů není sledovaný nějakým uživatelem. Pokud ano, systém automaticky zašle tomuto uživateli notifikaci.

1.4 Databázový model

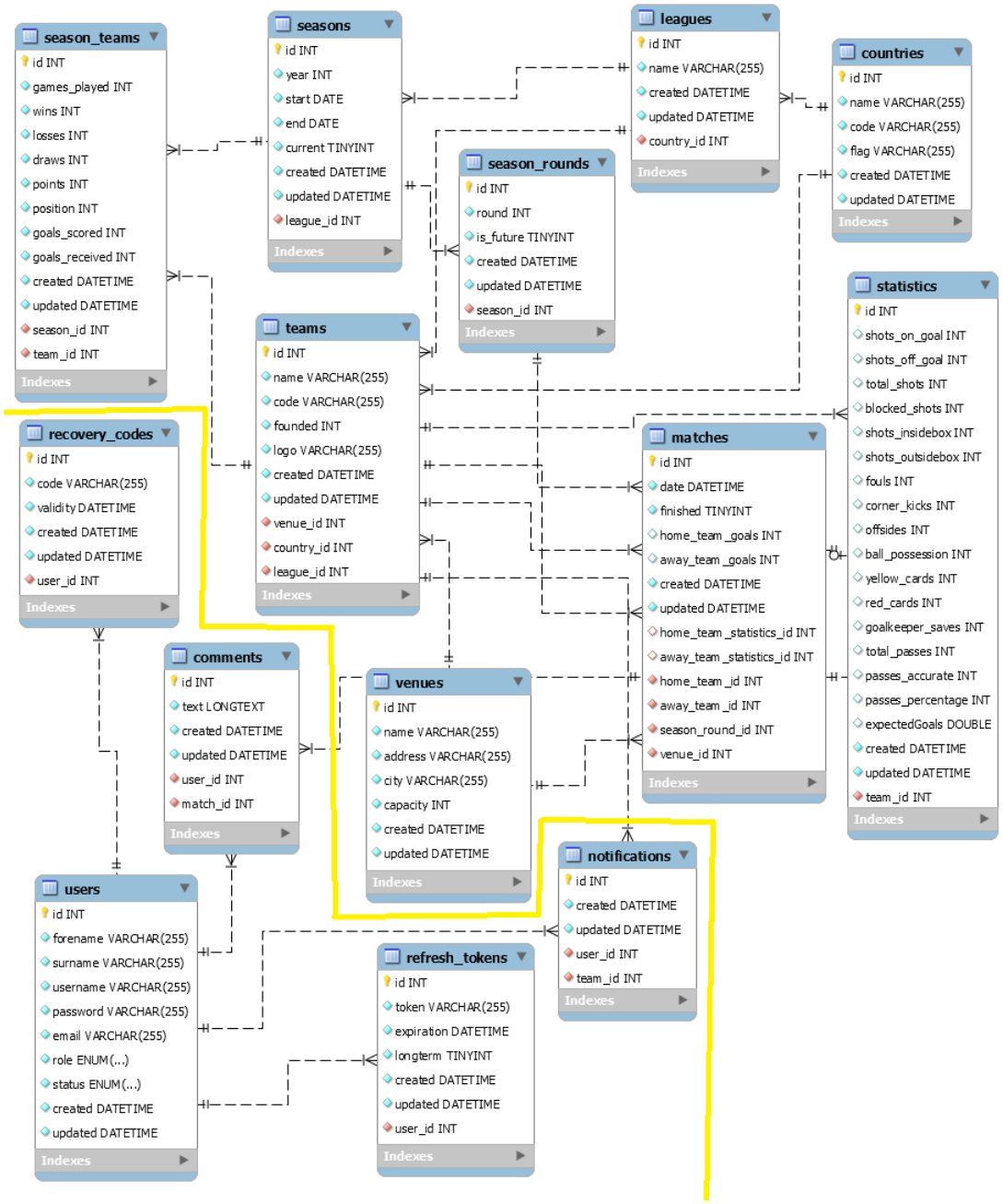
Databázový model, kterému se také říká ER diagram, je důležitý zejména pro vývojáře. Vyjadřuje nám, jaké entity se v systému vyskytují a jaký mezi sebou mají vzájemný vztah. Z vývojářského pohledu se jedná o jeden z nejdůležitějších modelů pro pochopení struktury celého systému. Nabízí zcela jiný pohled na systém oproti diagramu případů užití. Ten nám na systém nahlížel z hlediska funkcionalit a jejich přidělení jednotlivým aktérům, ale neřekl nám toho moc o technické struktuře projektu.

V dnešní době, kdy se hojně využívají modely ORM, kde nám jedna databázová entita přímo odpovídá jednomu modelu v aplikaci, můžeme zcela jednoduše na základě hotového databázového modelu sestavit modelovou vrstvu celého systému. Tento proces je pak velice jednoduchý a rychlý. Podobného principu je využito i v této diplomové práci, kdy jedna

entita v databázi odpovídá jedné entitě frameworku Spring Boot. O těchto entitách si blíže povíme v dalších kapitolách.

1.4.1 Ukázka databázového modelu

Databázový model vyvíjeného v rámci práce vypadá následovně.



Obrázek 2: Databázový model (Zdroj: Autor)

Jak si lze všimnout, tak model obsahuje celkem 14 databázových entit. Model lze rozdělit na dvě pomyslné části. Tyto části jsou v diagramu odděleny žlutou čarou. První

a větší část je zaměřena na zdrojová data týkající se právě fotbalu a jeho statistik. Data těchto entit jsou načítána automaticky z veřejného API nebo zadávána administrátorem systému. Druhá část je pak částí uživatelskou. Tím je na mysli, že se jedná o entity, jejíchž data definují samotní uživatelé systému.

Nyní si popíšeme jednotlivé entity.

a) Systémová část:

- **teams:** Obsahuje záznamy o jednotlivých týmech. Například jejich název, logo nebo rok založení.
- **countries:** Obsahuje záznamy o jednotlivých státech či zemích.
- **leagues:** Obsahuje záznamy o jednotlivých soutěžích.
- **seasons:** Obsahuje záznamy o jednotlivých sezónách dané soutěže. Například od kdy do kdy se sezóna hraje, či jestli je aktuální nebo již odehraná.
- **season_rounds:** Kola jednotlivých sezón.
- **matches:** Obsahuje záznamy o jednotlivých zápasech. Například datum odehrání, které týmy se utkaly nebo kolik gólů jaký tým vstřelil.
- **statistics:** Obsahuje podrobné statistiky k jednotlivým zápasům.
- **season_teams:** Uvádí, jak si v dané sezóně jednotlivé týmy vedou. Obsahuje součty výher, proher, remíz, vstřelených a obdržených gólů a také získané body. Na základě těchto dat je pak v aplikaci zobrazena bodová tabulka.
- **venues:** Obsahuje záznamy o stadionech jednotlivých týmů.

a) Uživatelská část:

- **users:** Obsahuje jednotlivé uživatele systémů. Například jméno, příjmení, email nebo také heslo a roli.
- **recovery_codes:** Obsahuje obnovovací kódy vygenerované pro uživatele během procesu obnovení zapomenutého hesla. Mimo samotný kód obsahuje také záznam o tom, jak dlouho je daný kód validní.
- **refresh_tokens:** Tabulka nutná pro proces autentizace. Obsahuje tokeny, které slouží pro obnovování samotných JWT tokenů, na základě kterých se uživatelé v systému ověřují. Tento proces bude podrobně popsán v dalších kapitolách.
- **comments:** Obsahuje komentáře k uživatelské jednotlivým zápasům.

- **notifications:** Obsahuje záznamy o nastavení notifikací uživatelů. Který uživatel dostává notifikace zvolených týmů.

2 REŠERŠE EXISTUJÍCÍCH ŘEŠENÍ

Tato kapitola se bude věnovat rozboru obdobných existujících řešení a systémů. Cílem této části je poskytnout přehled několika nejznámějších a nejpoužívanějších systémů zaměřených na toto téma. Tyto systémy budou zahrnovat jak domácí řešení, tak i významné zahraniční systémy. Důkladným rozborem těchto systémů budeme schopni lépe pochopit, jaké jsou jejich silné stránky a přínosy, které přinášejí svým uživatelům, a na druhé straně také identifikovat jejich nedostatky a slabiny. Tato analýza bude zásadní pro formulaci doporučení a návrhů na zlepšení, které by mohla tato práce nabídnout, aby se stala přínosem pro tento obor.

2.1 Livesport.cz

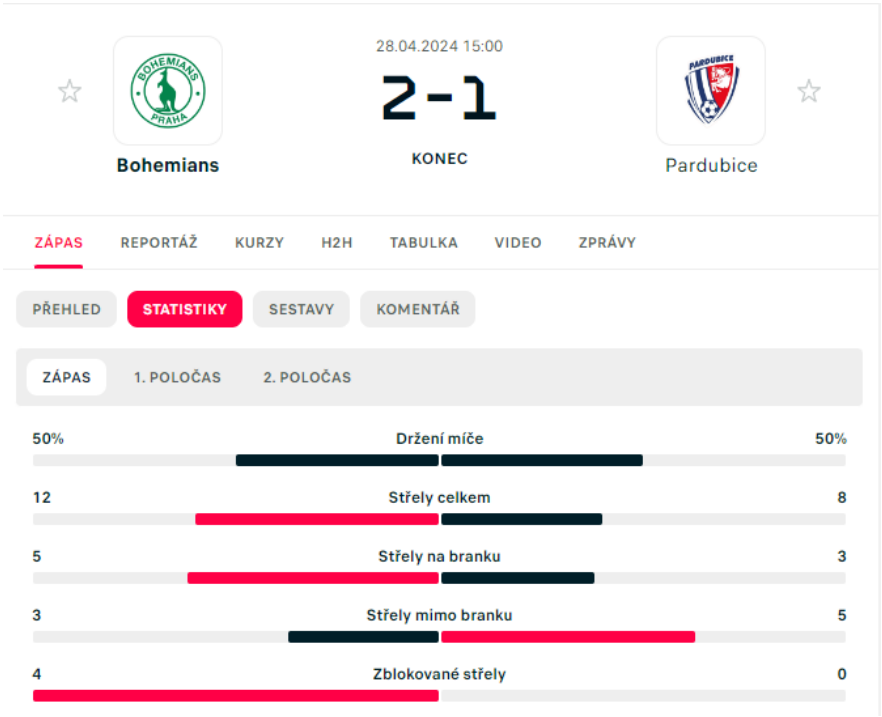
Livesport je celosvětově populární webová platforma poskytující výsledky sportovních událostí. Každý měsíc ji navštíví přes 100 milionů uživatelů. Fotbal byl jedním z prvních sportů, které se na této platformě objevily. Kromě výsledků nabízí také podrobnosti o klubech a hráčích, včetně jejich statistik a dalších informací. Livesport rovněž spolupracuje se sázkovými kancelářemi a zveřejňuje jejich kurzy na jednotlivé události. Chybí zde ovšem možnost grafického zobrazení celkových statistik například v podobě časových řad nebo vypočtené statistické ukazatele.

Výhody:

- Veřejně dostupný portál pro všechny
- Detailní statistiky jednotlivých zápasů
- Velké množství soutěží z celého světa
- Velké množství různých sportů

Nevýhody:

- Chybí zobrazení celkových statistik či časových řad
- Spousta reklam na stránce zhoršuje přehlednost



Obrázek 3: Ukázka serveru Livesport (Zdroj: `https://livesport.cz`)

Livesport nabízí vysoce kvalitní data pro širokou škálu soutěží. Nutno také podotknout, že systém není zaměřen pouze na fotbalová data. Najdeme zde opravdu širokou škálu sportů přes hokej, tenis, basketbal až po dostihy nebo dokonce e-sports. Jediný nedostatek je právě tedy nemožnost zobrazení celkových statistik v nějaké dobře čitelné podobě.

2.2 Fortunaliga.cz

Dalším příkladem jsou oficiální stránky nejvyšší české soutěže. Tento server je ale zaměřen právě pouze na nejvyšší soutěž, takže oproti serveru Livesport zde nenalezneme tak rozmanité množství soutěží a sportů. Nabízí ovšem velké množství dat. Dalo by se říct, že o této soutěži zde nalezneme opravdu skoro všechno. Je zde popsán průběh každého zápasu včetně i krátkého videa vystihujícího nejlepší okamžiky zápasu. Co se týká statistik, tak počet sledovaných ukazatelů je zde nižší v porovnání se serverem Livesport. Nalezneme zde mimo jiné taky detailní přehled jednotlivých týmů a jejich historické úspěchy. Velice zajímavý je taky přehled všech hráčů soutěže včetně informací o jejich přestupech.

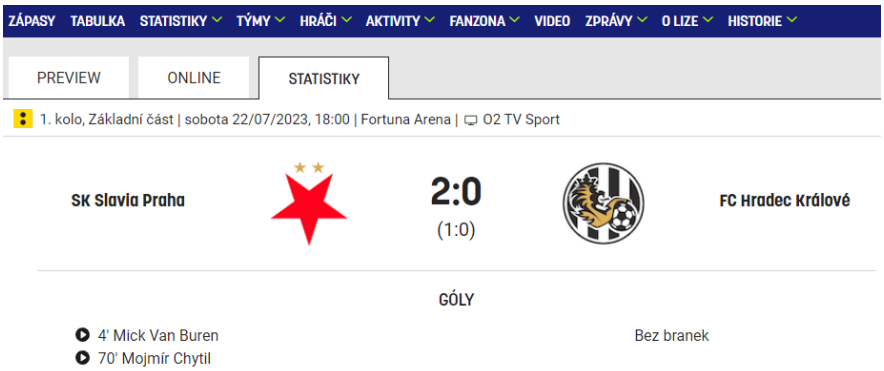
Pro přihlášené uživatele je zde také tzv. Fanzóna, kde mohou uživatelé například tipovat výsledky zápasů nebo hlasovat o nejlepšího hráče či trenéra měsíce.

Výhody:

- Veřejně dostupný portál pro všechny
- Velké množství informací o soutěži
- Detailní popis týmů a hráčů včetně jejich úspěchů a historie přestupů
- Relativně přehledná stránka bez reklam

Nevýhody:

- Pouze data o nejvyšší české soutěži
- Chybí grafické zobrazení celkových statistik a časových řad
- Méně statistických ukazatelů u jednotlivých zápasů



Obrázek 4: Ukázka serveru Fortunaliga (Zdroj: `https://fortunaliga.cz`)

Server Fortunaliga.cz nabízí opravdu dobré zázemí pro fanoušky nejvyšší české soutěže. Bohužel zde právě nenajdeme jakékoliv informace a data o jiných soutěžích. Dalším nedostatkem je pak omezenější počet sledovaných statistik a opět nemožnost zobrazení časových řad.

2.3 Sport.cz

Sport.cz je také bez pochyb další z velmi oblíbených a navštěvovaných sportovních serverů v České republice. Podobně jako Livesport je zaměřen na více různých sportů. Oproti němu ale nenabízí možnost prohlížet nižší, například krajské, fotbalové soutěže. Dále pak nenabízí jakoukoliv možnost prohlížet statistiky z odehraných zápasů, k dispozici jsou

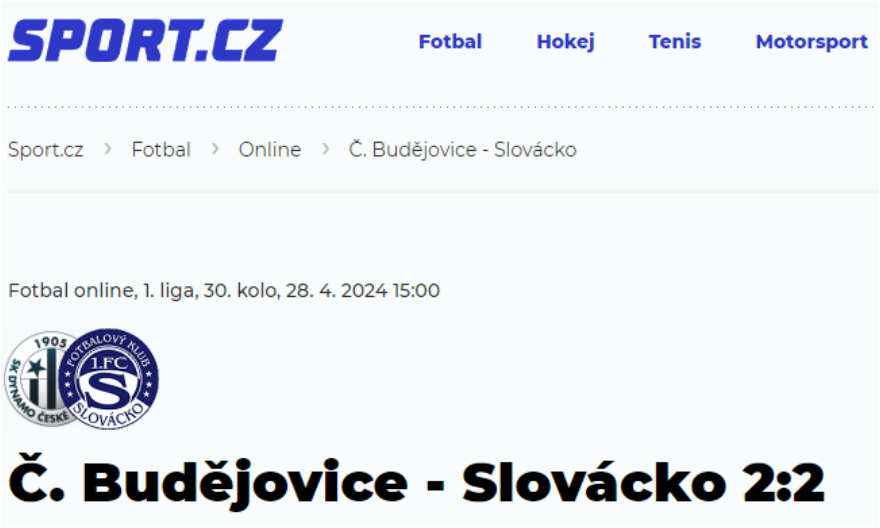
pouze góly a žluté či červené karty. Povedená je však online reportáž aktuálně hraných zápasů. V reálném čase jsou zde aktualizovány všechny důležité okamžiky daného zápasu. Za zmínku také určitě stojí u těchto reportáží možnost nastavení automatického čtení nových příspěvků, takže nemusíme mít stránku aktivně otevřenou a vše si sami číst, ale systém nám všechno automaticky poví. Je zde na výběr ze dvou různých hlasů a zdali fandíme domácím nebo hostům. Podle toho budou komentáře emocionálně zbarvené. Systém zároveň slouží jako sportovní noviny, takže zde nalezneme každý den nové články o tom, co se zrovna ve světě sportu událo.

Výhody:

- Veřejně dostupný portál pro všechny
- Přehledné a jednoduché uživatelské rozhraní
- Více možných sportů
- Online reportáže ze zápasů s možností automatického čtení
- Novinky a články ze světa sportu

Nevýhody:

- Pouze výsledky a žádné podrobnější statistiky
- Pouze nejvyšší a nejvýznamnější soutěže



Obrázek 5: Ukázka serveru Sport.cz (Zdroj: `https://sport.cz`)

Vcelku jde o jednoduchou a intuitivní stránku, které poslouží pro rychlé zobrazení nejdůležitějších informací a výsledků. Není ovšem vhodná a stavěná pro zkoumání podrobnějších statistik a ukazatelů. Primárně se jedná o takové sportovní noviny.

2.4 Vysledky.com

Jde o veřejnou webovou stránku, které zveřejňuje výsledky různých sportovních akcí. Opět se tedy nejedná pouze o fotbal. Největší zajímavostí je ale velké pokrytí právě tuzemských fotbalových soutěží. Můžeme říct, že zde nalezneme opravdu všechny, včetně těch nejnižších okresních. Zápasy bohužel obsahují pouze výsledky a žádné další doplňující informace natož statistiky. Na druhou stranu k velké části zápasů je přiložen videozáznam, který si lze volně přehrát. Veškeré informace na stránce ovšem nemusí být zcela pravdivé, jelikož je na stránce přímo uvedeno, že veškerý obsah je tvořen ve spolupráci s jednotlivými kluby a fanoušky, tudíž ho nelze považovat za oficiální. Stránka sice obsahuje odkazy na stránky s bližšími informacemi, jako například statistiky či přehled stadionů, ovšem většina těchto stránek je kompletně prázdná bez obsahu. Nelze si také nepovšimnout ohromného množství reklam, včetně otravných vyskakovacích oken.

Výhody:

- Veřejně dostupný portál pro všechny
- Více možných sportů
- Výsledky opravdu ze všech tuzemských soutěží, včetně těch nejnižších okresních
- Přehledné vyhledávání podle krajů a okresů
- U některých zápasů kompletní videozáznam
- Možnost přidávání komentářů

Nevýhody:

- Pouze výsledky a žádné podrobnější statistiky
- Informace nejsou oficiální a nemusí být pravdivé
- Spousta reklam na stránce
- Spousta odkazů na stránky bez obsahu

Obrázek 6: Ukázka serveru Vysledky.com (Zdroj: `https://vysledky.com`)

Na první pohled je vidět, že co se týče obsahu, tak jde převážně o komunitní projekt. Díky tomu zde lze ale nalézt data, která bychom v jiných systémech hledali velice těžko. Například pokud chceme vědět, jak odehrála zápas naše vesnice, tak nenalezneme lepší místo, kde to zjistit než právě na těchto stránkách. Krom výsledků a bodové tabulky zde ale žádné bližší informace nenaleznete.

2.5 365Scores.com

Posledním zkoumaným je mezinárodní systém 365Scores.com. Dle jejich vlastního popisu jde o nejrychlejší a nejpřesnější servis pro poskytování sportovních výsledků, který slouží více než 100 milionům uživatel po celém světě. Krom fotbalu systém obsahuje i více sportů, ovšem rozsah není takový, jako například u serveru Livesport. Najdeme zde pouze nejpopulárnější sporty jako například hokej, basketbal, tenis nebo volejbal. Zápasy zde jsou opravdu dobře popsány včetně detailních statistik. Vše je zde krásně graficky zpracované, například sestavy týmů nebo jednoduchá mapa střel na branku, ve které se můžeme podívat, odkud daný hráč střílel a do jakého místa. Informace o jednotlivých zápasech jsou zde opravdu vyčerpávající a nejlepší ze všech zkoumaných systémů. Najdeme zde také detailní profily jednotlivých hráčů včetně jejich statistik. Tyto statistiky lze ovšem zobrazit pouze za celou sezónu nebo za celou kariéru daného hráče. Celkové statistiky celého týmu zde ovšem nenajdeme. U jednotlivých soutěží zde není možnost přepnout

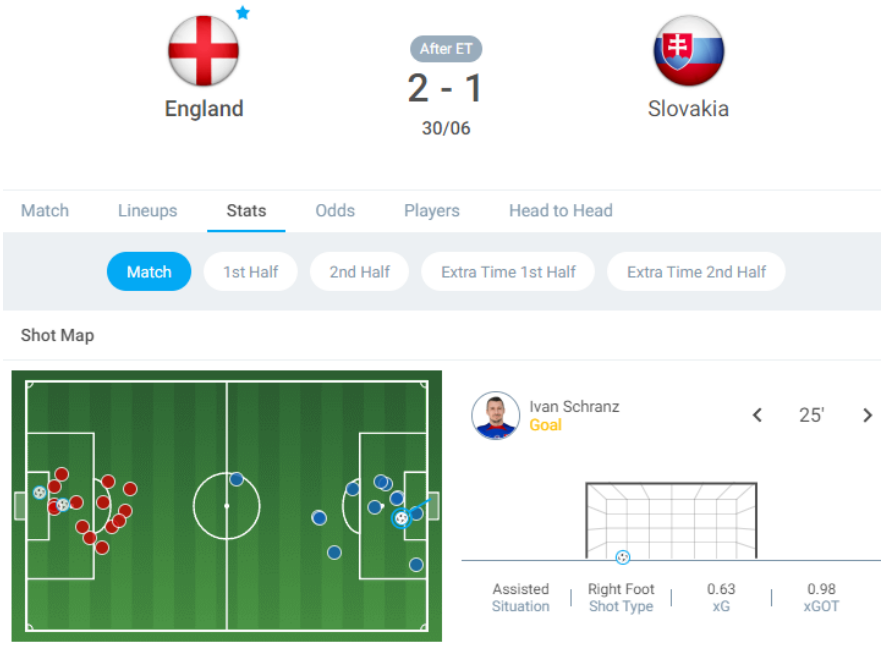
sezónu a snadno si tak zobrazit historická data. Pokrytí soutěžemi je pak omezeno pouze na ty nejvyšší.

Výhody:

- Veřejně dostupný portál pro všechny
- Více možných sportů
- Detailní informace a statistiky k jednotlivým zápasům
- Detailní informace a statistiky jednotlivých hráčů
- Graficky zpracované sestavy týmu a mapa střel na branku

Nevýhody:

- Pouze nejvyšší soutěže
- Nemožnost přepnutí sezóny
- Chybí celkové statistiky jednoho týmu za sezónu



Obrázek 7: Ukázka serveru 365Scores (Zdroj: `https://365scores.com`)

Jde o opravdu propracovaný systém, který bez pochyb navštěvuje spousta lidí po celém světě. Zobrazení jednotlivých zápasů je opravdu velice propracované a nalezneme zde vše, co by nás mohlo zajímat. Velká škoda je, že informace a souhrn statistik jednotlivých hráčů není aplikovaný na celý tým, což nám odebírá možnost týmy vzájemně porovnávat dle jednotlivých ukazatelů.

2.6 Shrnutí existujících řešení

Následující tabulka shrnuje všechna rozebraná konkurenční řešení na základě stanovených požadavků.

Tabulka 4: Shrnutí existujících řešení

	Livesport	**Fortunaliga**	**Sport.cz**	**Vysledky.com**	**365Scores**
Zobrazení výsledků	ano	ano	ano	ano	ano
Zobrazení bodové tabulky	ano	ano	ano	ano	ano
Více možných soutěží	ano	ne	ano	ano	ano
Nejnižší soutěže	ne	ne	ne	ano	ne
Statistiky jednotlivých zápasů	ano	ano	ne	ne	ano
Celkové statistiky celého týmu	ne	ne	ne	ne	ne
Zobrazení časových řad	ne	ne	ne	ne	ne
Základní statistické ukazatele	ne	ne	ne	ne	ne
Možnost přepnutí jazyka	ne	ne	ne	ne	ano
Možnost přepnutí barevného schématu	ano	ne	ne	ne	ano

Z tabulky vychází, že nejvíce požadavků splňují servery Livesport a 365Scores. Detaily jednotlivých zápasů u nich obsahují nejvíce informací, včetně detailních statistik. Tyto statistiky obsahuje i server Fortunaliga.cz, ale sleduje sotva polovinu ukazatelů oproti serverům Livesport a 365Scores. Jeho největší nevýhoda pak ale spočívá v tom, že obsahuje pouze data o jedné soutěži.

Žádný z rozebíraných serverů ovšem neobsahuje celkové statistiky jednotlivých týmů. A to jak v podobě časových řad, tak v podobě tabulky s vypočtenými statistickými ukazateli. Proto žádný z těchto serverů nesplňuje základní požadavek této práce.

Zajímavostí je také například to, že žádný z českých serverů neposkytuje možnost přepnutí do anglického jazyka.

3 VÝBĚR DATOVÉHO ZDROJE

Tato kapitola se zaměřuje na jeden z klíčových prvků systému, kterým je spolehlivý a dostatečný datový zdroj. Správný výběr datového zdroje je zásadní pro fungování celého systému, protože kvalita a dostupnost dat přímo ovlivňuje kvalitu celého systému.

Budou představeny různé možnosti, které by mohly posloužit jako datový zdroj pro vyvíjený systém. Stejně jako v předchozí kapitole budou všechny možnosti detailně popsány včetně jejich výhod a nevýhod. Budou hodnoceny na základě kritérií jako například množství a kvalita dat, dostupnost, formát dat, ale také náklady na implementaci a provoz. Díky tomuto rozboru, bude pro implementovaný systém zvolen ten nejlepší a nejspolehlivější zdroj. Tim se zajistí, že systém bude moci efektivně fungovat a plnit své úkoly na základě kvalitních a spolehlivých dat.

Veřejných API, které by mohly posloužit jako datový zdroj pro vyvíjený systém je opravdu mnoho. Byly proto vybrány čtyři systémy na základě doporučení a relevance vyhledávání.

3.1 Výběrová kritéria

Nejprve je třeba stanovit kritéria na základě kterých bude vybrán výsledný systém, který bude sloužit jako datový zdroj.

1. Platební plán zdarma zahrnující nejvyšší českou soutěž
2. Platební plán zdarma zahrnující podrobné statistiky
3. Dostatečné množství dostupných dotazů za den
4. Data ve formátu JSON

3.2 Apifootball.com

Prvním rozebíraným systémem je server Apifootball.com. Dle jejich vlastního popisu jde o systém poskytující rychlé, spolehlivé a přesné informace ze soutěží z celého světa. Servis by měl být cenově dostupný a snadný na implementaci. Krom API nabízí například widgety pro přímou implementaci do webové stránky nebo plugin do redakčního systému WordPress.

Nabízí celkem čtyři platební plány, kterými jsou Free, European, Worldwide a Premium. Všechny platební plány nabízí přístup ke stejným datům, liší se pouze v množství nabízených soutěží a počtu možných dotazů na API za jednu hodinu.

Platební plán Free, který je zcela zdarma ovšem nabízí pouze dvě souže, kterými jsou druhá anglický a druhá francouzská nejvyšší soutěž. Tento plán slouží převážně pro volné testování všech možností API. Počet dotazů je u tohoto plánu omezen na 180 za hodinu.

Oproti tomu platební plán European je již použitelný pro reálný systém. Nabízí více než 60 evropských soutěží včetně Ligy mistrů a Evropské ligy UEFA. Zároveň i počet možných dotazů za hodinu se zvýšil na 100. Z českých soutěží nabízí tento plán pouze nejvyšší soutěž. Za tyto možnosti si ovšem zaplatíme 25 amerických dolarů měsíčně.

Za dalších 25 dolarů můžeme mít platební plán Worldwide. Jak již z názvu napovídá, tak tento plán nám nabídne soutěže z celého světa. Konkrétně více než 800 soutěží, včetně značného rozšíření těch českých. Nyní jich máme k dispozici až 8, včetně ženské nebo dorostenecké ligy. Počet možných dotazů za hodinu zůstává stejný jako u předchozího plánu. Tento plán je na webových stránkách označen jako nejpopulárnější.

Posledním plánem je plán Premium. Ten nabízí stejné možnosti jako plán Worldwide, ovšem nyní máme k dispozici neomezený počet dotazů na API a přístup k predikcím výsledků zápasů. Měsíční cena stoupne o dalších 25 dolarů.

Výsledné hodnocení na základě výběrových kritérií je následující:

1. Platební plán zdarma pouze pro dvě soutěže
2. Velké pokrytí dat, včetně detailních statistik v platebním plánu zdarma
3. Dostatečný počet dotazů na API za jeden den
4. Data ve formátu JSON

Jelikož systém v platebním plánu zdarma obsahuje pouze dvě soutěže, tak je pro tuto práci nepoužitelný.

3.3 Api-football.com

Dalším v pořadí je server Api-football.com. Jde o propracovanou REST API, která stejně jako předešlý server nabízí mimo jiné taky widgety pro přímou implementaci. Tuto možnost jistě ocení zájemci o službu, kteří nemají zkušenosti s vývojem webových aplikací.

Služba nabízí celkem 1130 různých soutěží a pohárů, což je více než předešlý server a navíc všechny soutěže jsou dostupné ve všech platebních plánech.

Tyto plány zde najdeme opět čtyři a krom všech dostupných soutěží každý z těchto plánů nabízí všechny dostupné endpointy. Jednotlivé plány se od sebe liší pouze počtem možných dotazů na API, které zde nejsou limitovány na hodinové bázi, ale na denní. Platební plán zdarma pod názvem Free nabízí uživateli celkem 100 možných dotazů za den. Platební plán Pro, který nás přijde na 19 dolarů měsíčně tento počet rozšíří na celých 7 500 dotazů denně. Za dalších 10 dolarů můžeme mít plán Ultra, který tento počet dokonce zdesetinásobí. Dostaneme se tedy na limit 75 000 dotazů za den. Posledním plánem je pak plán Mega, se kterým se za příplatek dalších 10 dolarů dostaneme na limit 150 000 dotazů.

Tato služba je tedy na první pohled mnohem levnější a výhodnější než předešlý server. Poskytuje nejen stejné, ale dokonce větší množství dat a za mnohem příznivějších podmínek. Jelikož jde o REST API, tak veškeré výsledky vrácené ze služby jsou ve formátu JSON.

Větší pokrytí nastává i v otázce českých soutěží. Služba Api-football.com jich nabízí dokonce hned 16, včetně nižších divizních soutěží.

Výsledné hodnocení na základě výběrových kritérií je následující:

1. Platební plán zdarma obsahuje všechny soutěže
2. Velké pokrytí dat, včetně detailních statistik v platebním plánu zdarma
3. Dostatečný počet dotazů na API za jeden den
4. Data ve formátu JSON

Systém obsahuje vše potřebné v platebním plánu zdarma a je tak vhodný datový zdroj pro tuto práci.

3.4 Football-data.org

Následuje server Football-data.org, který hned na první pohled vypadá oproti předchozím řešením mnohém méně rozsáhlejší. Opět jde o REST API, které veškerá data vrací ve formátu JSON. Nabídka různých soutěží je zde znatelně omezená. Najdeme jich zde pouze 100 a to pouze v nejdražším platebním plánu. Bohužel z českých soutěží zde nenalezneme žádnou.

Platebních plánů zde najdeme celkem pět. Těmito plány jsou Free, Free + Three, Standard, Advanced a Pro. Dále je v nabídce balíček, který za cenu 15 euro měsíčně přidá přístup k odhadům budoucích výsledků. Plán Free obsahuje pouze velice omezená data. K dispozici máme pouze 12 soutěží, a to bez podrobnějších statistik. Počet možných dotazů na API je zde na minutové bázi a platebním plánu zdarma jich máme k dispozici 10. Podrobnější data nalezneme až v plánu Free + Three, který nám, jak již z názvu vypovídá, přidá i možnost dalších třech soutěží. Zvýšil se i počet možných dotazů a to na 30 za minutu. Zaplatíme ovšem 29 euro měsíčně. Následují další tři platební plány, které nám již pouze rozšiřují počet přístupných soutěží a počet možných dotazů na API za jednu minutu. Jak již bylo řečeno, tak v nejdražším plánu Pro ale nalezneme pouze 100 možných soutěží, a to vyjma jakékoli české. Tohle málo nás ovšem přijde na celých 199 euro měsíčně.

Výsledné hodnocení na základě výběrových kritérií je následující:

1. Platební plán zdarma neobsahuje žádné české soutěže
2. Podrobnější statistiky pouze v placených plánech
3. Dostatečný počet dotazů na API za jeden den
4. Data ve formátu JSON

Systém neobsahuje žádnou z českých soutěží a zároveň ani podrobnější statistiky nejsou k dispozici zdarma. Obecně nabízí malé množství dat za obrovské ceny. Zatím se jedná o nejhorší službu a pro tuto práci je jako datový zdroj nepoužitelná.

3.5 Sportmonks.com

Poslední je server Sportmonks.com, který nám toho oproti předešlému serveru nabídne mnohem více. V nabídce nalezneme dokonce vice než 2 200 různých soutěží z celého světa. Stejně jako u předchozích řešení jde o REST API, takže datový formát je opět ve formátu JSON. Služba nabízí také možnost widgetů pro snadnou integraci s existující webovou aplikací.

V nabídce platebních plánů nalezneme celkem čtyři. Každý z těchto plánu je navíc dále rozdělen do třech možných pod-plánů, které nám určují, jaké množství dat můžeme z API získat. Výjimkou je pouze plán Free, který nabízí pouze základní pod-plán. V základním pod-plánu máme k dispozici pouze výsledky, bez žádných podrobnějších statistik. Tyto

statistiky dostaneme až v následujícím, ovšem opět si za ně připlatíme. V nejdražším pod-plánu pak máme k dispozici například ještě podrobnější statistiky nebo analýzu různých trendů.

Bohužel jak již ze samotného popisku plánu Free napovídá, tak slouží pouze pro testovací účely a nabízí pouze dvě možné soutěže, a to dánskou a skotskou nejvyšší soutěž. Jak již bylo řečeno, tak nemáme ani možnost získat podrobnější statistiky. Počet možných dotazů na API je zde omezen na 180 za hodinu. Následují plány European a Worldwide. Plán European nabízí celkem 27 evropských soutěží a počet možných dotazů se zvýšil na 3 000 za hodinu. Za tento plán si se základním pod-plánem zaplatíme 34 euro měsíčně. Pokud bychom chtěli podrobnější statistiky, tak nás to vyjde na 51 euro. Plán Worldwide nabízí soutěže z celého světa a k dispozici jich máme celkem 111. Základní cena je 112 euro a s možností podrobnějších statistik pak 165 euro měsíčně. Posledním plánem je plán Enterprise, který slouží pro profesionální řešení. Možnosti tohoto plánu jsou čistě individuální a záleží na domluvě s poskytovatelem služby. Můžeme se ale dostat až na 2 20 různých soutěží a na počet dotazů za hodinu až 250 000.

Výsledné hodnocení na základě výběrových kritérií je následující:

1. Platební plán zdarma pouze pro dvě soutěže
2. Podrobnější statistiky pouze v placených plánech s vyšším pod-plánem
3. Dostatečný počet dotazů na API za jeden den
4. Data ve formátu JSON

Jelikož systém v platebním plánu zdarma obsahuje pouze dvě soutěže a navíc bez žádných podrobnějších statistik, tak je pro tuto práci nepoužitelný.

3.6 Shrnutí možných datových zdrojů

Následující tabulka shrnuje všechny rozebrané datové na základě stanovených výběrových kritérií.

Tabulka 5: Shrnutí možných datových zdrojů

	Apifootball	**Api-football**	**Football-data**	**Sportmonks**
Platební plán zdarma zahrnující nejvyšší českou soutěž	ne	ano	ne	ne
Platební plán zdarma zahrnující podrobné statistiky	ano	ano	ne	ne
Dostatečné množství dostupných dotazů za den	ano	ano	ano	ano
Data ve formátu JSON	ano	ano	ano	ano

Všechna výběrová kritéria splnil pouze server Api-football.com a byl proto vybrán jako vhodný datový zdroj pro tuto práci.

4 POUŽITÉ TECHNOLOGIE

Tato kapitola představí veškeré technologie a knihovny, které byly při vývoji systému použity. Kapitola bude rozdělena na tři hlavní části. První částí bude rozbor technologií použitých pro tvorbu serverové části aplikace. Druhá část se naopak zaměří na technologie použité pro tvorbu klientské části aplikace. Poslední část představí další technologie, které byly při vývoji použity, jako například vývojové prostředí či verzovací systém.

4.1 Serverová část

Serverová část aplikace, označována také jako backend, je taková část, která se stará o zpracování dat, logiku a komunikaci s databázemi. Backend běží na serverech a uživatelé tak k němu nemají přístup. Z toho důvodu by na této vrstvě měla být vyřešena veškerá bezpečnost, protože tento kód již nelze podvrhnout. Například do databáze by se pak měla dostat jen a pouze čistá a ošetřená data. Komunikace se serverovou částí obvykle bývá prostřednictvím API.

Mezi základní úkoly serverové části tedy patří:

- **Business logika:** Určuje pravidla, podle kterých aplikace funguje.
- **Validace a ošetření dat:** Stará se o finální validaci vstupních dat.
- **Autentizace a autorizace:** Ověřování uživatelů a řízení jejich přístupu k různým částem aplikace.
- **Komunikace s databází:** Stará se o načítání, ukládání a obecně manipulaci dat uložených v databázi.
- **Komunikace s externími službami:** Integrace s externími systémy pomocí API.

4.1.1 Java

Java je objektově orientovaný jazyk, který vyvinula společnost Sun Microsystems v roce 1995. Od roku 2010 je již pod správou společnosti Oracle. V roce 2007 byly také uvolněny zdrojové kódy Javy a od té doby je vyvíjena jako open source. Cílem bylo vyvinout prostředí, které by bylo nezávislé na platformě. Hlavní myšlenka tady spočívala v tom, aby bylo možné napsat kód pouze jednou a následně ho spouštět kdekoliv. Toho bylo dosaženo pomocí Java Virtual Machine. Úkolem tohoto modulu je zpracovat tzv. mezikód, který je

označován jako Java bytecode. Tento mezikód je vytvořen ze zdrojových souborů jazyka Java. V dnešní době je jazyk Java populární hlavně díky velkému rozšíření na serverových aplikacích či mobilnímu operačnímu systému Android.

Java je silně staticky typový jazyk. To znamená, že datové typy všech proměnných jsou známy již během kompilace. Zároveň se jedná o čistě objektově orientovaný jazyk, takže všechno v Javě je objekt. Plně podporuje základní principy OOP, kterými jsou dědičnost, zapouzdření a polymorfismus. Důraz je v Javě také kladen na bezpečnost a robustnost. Má přísnou správu paměti a další bezpečností mechanismy, jako je například kontrola přístupu. Podporuje například také výjimky nebo multithreading.

Java rozlišuje několik různých edic. Tyto edice jsou navrženy pro různé typy aplikací. Základní edicí je Java Standard Edition (Java SE), která je určená pro desktopové či jednoduché serverové aplikace. Následuje edice Enterprise (Java EE), která je určená pro podnikové aplikace, rozšiřuje Javu o technologie jako servlety a další webové služby. Poslední edicí je edice Micro (Java ME). Ta je určená pro mobilní zařízení a vestavěné systémy.

```java
@Transactional
public JwtResponse refreshToken(RefreshTokenRequest request) {
  RefreshToken refreshToken = refreshTokenService.findByToken(request.getToken());
  User user = refreshToken.getUser();
  String jwt = jwtService.generateToken(user);
  RefreshToken newRefreshToken = refreshTokenService.create(user, refreshToken.getLongterm());
  return JwtResponse.builder().user(user.toDto()).jwt(jwt).refreshToken(newRefreshToken.getToken()).build();
}
```

Obrázek 8: Ukázka kódu v jazyce Java (Zdroj: Autor)

Díky své stabilitě, bezpečnosti, platformové nezávislosti a širokému ekosystému různých knihoven se Java stala jedním z nejpopulárnějších programovacích jazyků na světě a stále zůstává velice oblíbenou volbou pro vývojáře.

4.1.2 Spring Boot

Spring Boot je framework, který umožňuje snadný a rychlý vývoj aplikací na platformě Java. Jde o nadstavbu staršího frameworku Spring a jeho hlavním cílem je oproti němu zjednodušit konfiguraci aplikací. Odpadá nám nutnost nastavování složitých XML konfiguračních souborů a můžeme se tak více zaměřit na tvorbu samotné aplikace než na její

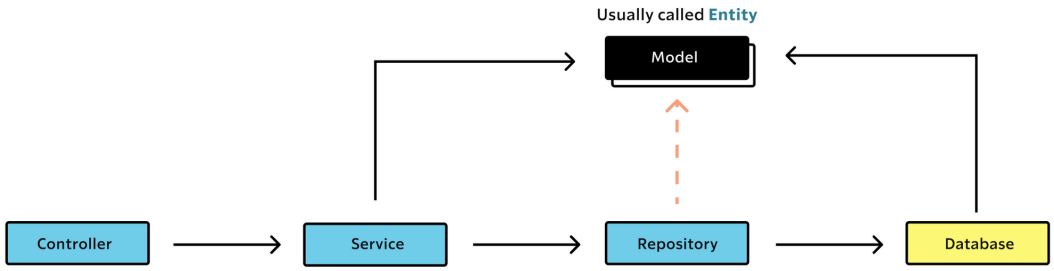
konfiguraci. Spring Boot byl poprvé představen v roce 2014 společností Pivotal Software. Od té doby se z něj stal jeden z nejpoužívanějších nástrojů pro vývoj moderních aplikací.

Mezi klíčové vlastnosti tohoto frameworku patří:

- **Automatická konfigurace:** Mnoho běžných nastavení je automaticky nakonfigurováno bez nutnosti ručního zásahu.
- **Integrovaný server:** Obsahuje integrovaný aplikační server jako například Tomcat. To nám umožňuje snadné spuštění aplikací pomocí jar souborů.
- **Spustitelné jar soubory:** Aplikace vytvořené frameworkem jsou distribuovány jako spustitelné jar soubory, které obsahují všechny nezbytné závislosti a konfigurace. Díky tomu je lze snadno nasadit a spouštět.
- **Spring Initializr:** Díky této webové aplikaci můžeme snadno a rychle vygenerovat základní strukturu projektu, a to včetně zvolených závislostí. Následně stačí pouze stáhnout hotový projekt a začít tvořit jeho obsah.

Spring Boot také nabízí integraci se Spring Security, což nám umožní snadné nastavení autorizace a autentizace v naší aplikaci.

Cyklus frameworku Spring Boot tvoří několik vrstev. První vrstvou je jsou kontrolery, které definují samotné endpointy aplikace. Následují servisní vrstvy, které obsahují veškerou logiku. Servisní vrstva dále volá metody z vrstvy repozitářů, které slouží ke komunikaci s databází. Vše je zobrazeno na následujícím diagramu.



Obrázek 9: Diagram architektury Spring Boot (Zdroj: `https://1kevinson.com/understand-how-spring-boot-architecture-is-designed/`)

Spring Boot je vcelku velice mocný nástroj pro vývoj moderních webových aplikací. Snadná konfigurace, škálovatelnost a bohatá sada nástrojů jej činí ideální volbou pro mnoho projektů.

4.1.3 MySQL

MySQL je jednou z nejpopulárnějších open-source relačních databází. První verze byla vydána v roce 1995 švédskou společností MySQL AB, později byla v roce 2010 odkoupena společností Oracle, která ji spravuje dodnes. Byla navržena jakožto spolehlivý, robustní, rychlý, ale zároveň jednoduchý databázový systém. Jde o plně multiplatformní databázi, se kterou lze komunikovat pomocí strukturovaného dotazovacího jazyka SQL. Díky svým vlastnostem a zároveň díky tomu, že jde o volně šiřitelný software, má velmi vysoký podíl v současně používaných systémech. Mimo komunitní verzi MySQL nabízí také další komerční verze, které nabízí další funkce a podporu. Komunitní open-source verze je ovšem velice dostačující pro většinu malých, ale i větších projektů. Často bývá použita v kombinaci s jazykem PHP, tato kombinace je natolik oblíbená, že bývá distribuována v balíčkách obsahujících právě databázi MySQL, jazyk PHP a webserver Apache. Těmito balíčky jsou například XAMPP, LAMP nebo WAMP.

Databáze má podporu různých uživatelských účtů a oprávnění. Mimo klasické tabulky lze vytvářet také pohledy, procedury nebo triggery. Největší nevýhodou oproti ostatním databázím je například pouze jednoduchý způsob zálohování dat. Datové přenosy jsou šifrovány pomocí SSL. Podporuje hned několik možných úložných enginů mezi které patří například výchozí InnoDB nebo MyISAM. Dokáže ale data ukládat například také v textové podobě ve formátu CSV. Připojení k databází je možné pomocí konektorů, které jsou podporované ve většině nejpoužívanějších jazycích jako například již zmiňované PHP nebo také Java či Python. Konfigurace databáze je možná pomocí speciálních konfiguračních souborů, ve kterých lze snadno nastavit například port na kterém databázový server poběží, maximální počet připojení nebo velikost bufferů.

Díky své jednoduchosti, rychlosti, robustnosti a aktivní komunitě se databáze MySQL stala základem pro mnoho aplikací po celém světě od malých webových stránek po velké podnikové aplikace.

4.2 Klientská část

Klientská část aplikace, která se také označuje jako frontend, je část, se kterou přímo interagují uživatelé systému. Oproti serverové část je kód spouštěn na straně uživatele, například ve webovém prohlížeči. Veškeré zdrojové kódy jsou tedy volně přístupné uživateli

a dají se snadno změnit a podvrhnout. Z toho důvodu bychom rozhodně neměli spoléhat na bezpečnost a validaci vstupních dat provedenou právě touto částí.

Mezi základní úkoly klientské části tedy patří:

- **Uživatelské rozhraní:** Vykresluje uživateli data a ovládací prvky, kterými jsou například tlačítka, formuláře a menu.
- **Interaktivita:** Implementace funkcí, které reagují na uživatelské akce, jako například kliknutí na tlačítko. Čím interaktivnější systém, tím lepší uživatelské zkušenost.
- **Komunikace se serverovou částí:** Zasílání a načítání dat pomocí HTTP dotazů na API.

4.2.1 HTML

HTML je značkovací jazyk používaný pro vytváření webových stránek. Pro webové stránky jde o standardní technologii a základní kámen pro vytváření jejich obsahu. HTML nám umožňuje definovat strukturu celého webu, například texty, odstavce, tabulky, formuláře, odkazy nebo obrázky. HTML bylo poprvé představeno v roce 1991 Timem Berners-Leem, jakožto způsob snadné publikace dokumentů na internetu. První verze byla velice jednoduchá a obsahovala pouze několik základních značek. Rozvoj HTML se zvyšoval s rostoucími požadavky na interaktivitu a multimediální obsah. Poslední vydanou verzí je HTML5, které vyšlo v roce 2014. HTML5 přináší mnoho novinek a vylepšení, které nám usnadní tvorbu struktury webu. Zavádí například nové sémantické značky jako *header*, *footer*, *nav*, *section* a *article*, které nám zlepšují strukturu a přehlednost kódu.

Jazyk HTML je charakterizován množinou značek a jejich atributů. Obsahuje dva typy různých značek, párové a nepárové. Párové značky jsou takové, které uzavírají nějaký obsah v sobě a jsou zakončeny stejnou značkou jako je značka počáteční. Mezi párové značky patří například značka pro odstavec textu, nadpis, anebo třeba tabulka či formulář. Oproti tomu nepárové značky žádný obsah nemají a nejsou tedy zakončeny koncovou značkou. Nepárovými značkami jsou například obrázek či textové pole.

Základ HTML struktury je pevně definován. Dokument musí vždy začínat deklarací typu dokumentu, aby prohlížeč věděl, že otevřel právě dokument typu HTML. Následuje značka *html*, která slouží jako kořenový element celého dokumentu. Tato značka má v sobě další dvě povinné značky, kterými jsou *head* a *body*. Prvek *head* obsahuje definici metadat celého dokumentu. Definuje například titulek dokumentu, kódování, klíčová slova pro

vyhledávač, kaskádové styly nebo také načtení externích JavaScript souborů. Prvek *body* pak zahrnuje vlastní obsah dokumentu.

```html
<!DOCTYPE html>
<html>
 <head>
  <meta charset="utf-8">
  <link href='https://fonts.googleapis.com/css?family=Roboto' rel='stylesheet'>
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <link rel="icon" type="image/x-icon" href="/favicon.png">
  <title>Titulek</title>
  <script type="module" src="/src/main.tsx"></script>
 </head>
 <body>
  <h1>Nadpis dokumentu</h1>
  <p>Tohle je nějaký odstavec</p>
  <img src="/img/obrazek.png" alt="Obrázek">
 </body>
</html>
```

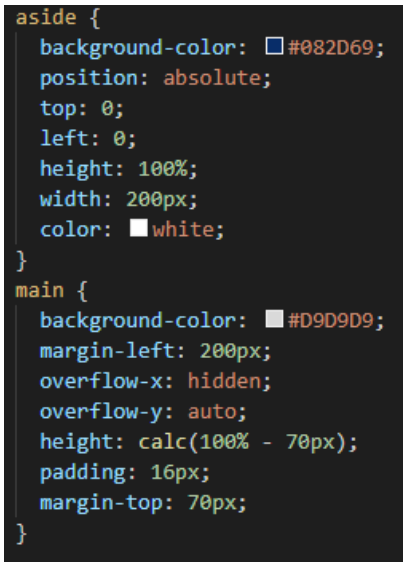
Obrázek 10: Ukázka kódu v jazyce HTML (Zdroj: Autor)

HTML je základním stavebním kamenem webových stránek a aplikací. Neustále se rozvíjí a zůstává tak stále relevantní a efektivní nástroj pro vytváření moderních webových aplikací. Jeho znalost je nutnou podmínkou pro jakéhokoliv vývojáře se zaměřením na webové systémy.

4.2.2 CSS

CSS je jazyk pro definování vzhledu webových aplikací napsaných v jazyce HTML. Poskytuje mnoho možností pro definování vizuálních vlastností HTML prvků, od různých barev až po složitější pozicování. CSS jsou psány oddělené od samotného obsahu, což zjednodušuje organizaci a údržbu projektu. První specifikace CSS1 byla vydána v roce 1996 Håkonem Wiumem Lie. V roce 1998 následovalo CSS2, které přineslo podporu media queries, což umožnilo tvorbu responzivních webových stránek. Aktuální verzí je CSS3, která byla vydána v roce 1999. Nejnovější verze přinesla mnoho nových funkcí, včetně animací a transformací.

Základní syntaxe CSS se skládá ze třech prvků. Prvním je selektor, který určuje, na jaké HTML prvky se daný styl bude aplikovat. Následují vlastnosti a jejich hodnoty.



Obrázek 11: Ukázka kódu CSS (Zdroj: Autor)

Českým názvem pro CSS jsou kaskádové styly což odpovídá tomu, jak se styly uplatňují. Vždy bude uplatněn styl, který se v souboru s CSS nachází níže. Pokud tedy nadefinujeme jednu vlastnost pro určitý prvek dvakrát, bude uplatněna až spodnější definice. Prioritu uplatnění také ovlivňuje zvolený typ selektoru. Například pokud budeme mít selektor na základě HTML tagu, atributu class a atributu id, tak nejvyšší prioritu budou mít styly v selektoru s atributem id, následně s atributem class a až nakonec styly na základě HTML tagu. Pokud chceme nějakému stylu vnutit nejvyšší možnou prioritu, můžeme použít speciální klíčové slovo *!important*.

CSS jsou klíčovým nástrojem pro tvorbu vizuálně přitažlivých webových aplikací. Bez CSS bychom nebyli schopni vytvořit tak dokonalý uživatelský zážitek a všechny webové aplikace a stránky by vypadaly ošklivě a podobně.

4.2.3 JavaScript

JavaScript je vysoce populárním jazykem pro vytváření interaktivních a dynamických webových aplikací. Spolu s HTML a CSS tvoří základní stavební kameny moderního webu. Dalo by se říct, že ve svém oboru nemá JavaScript ve světě pořádného konkurenta a bez něj by neexistovala většina webových aplikací v podobě, jaké je známe. Jmenovitě například giganti jako Facebook. JavaScript byl vytvořen v roce 1995 během pouhých deseti dnů. V roce 1997 byl standardizován pod názvem ECMAScript. Od té doby je neu-

stále aktualizován a vylepšován o nové funkcionality a zlepšení výkonu. Kód JavaScriptu lze psát přímo do HTML struktury do tagu *script* nebo do samostatného souboru.

JavaScript je dynamicky typový jazyk, což znamená, že proměnným nespecifikujeme žádné datové typy, ale jsou jim automaticky přiřazeny na základě hodnoty, kterou daná proměnná obsahuje. Právě absence typové kontroly je velkým zdrojem chyb, které se projeví až za běhu aplikace a často tak může být složité je odhalit již během vývoje. Musíme například neustále hlídat, zdali porovnáváme proměnné stejného typu nebo zdali nějaká objekt obsahuje vlastnost, ke které chceme přistupovat. Velkým problémem také může být kompatibilita různých funkcí napříč různými prohlížeči. Pokud budeme chtít použít nějakou novou funkci, musíme zkontrolovat, zdali je podporována všemi nejpoužívanějšími prohlížeči. Mohlo by se nám také stát, že nějakému uživateli na jiném prohlížeči nebude aplikace fungovat tak, jak má. Většinu těchto nedostatků řeší nadstavba TypeScript, které bude představena v následující podkapitole.

```
const validateEmail = (email) => {
  const re = /\S+@\S+\.\S+/;
  return re.test(email);
};

const input = document.getElementById('emailField');
input.addEventListener('change', function() {
  if (!validateEmail(this.value)) {
    alert('Input is not valid email!');
  }
});
```

Obrázek 12: Ukázka kódu v jazyce JavaScript (Zdroj: Autor)

JavaScript je klíčovým nástrojem pro tvorbu moderních interaktivních webových aplikací. Navzdory tomu, že má spoustu odpůrců, například kvůli absenci typové kontroly a velké chybovosti, tak stále zůstává světovou špičkou pro tvorbu webových aplikací. Tento fakt dále umocnil příchod moderních frameworků a knihoven, kterými jsou například React, Vue.js a Angular.

4.2.4 TypeScript

TypeScript je nadstavba jazyka JavaScript, která mimo jiné přidává volitelnou statickou typovou kontrolu. Byl vyvinut společností Microsoft a poprvé představen veřejnosti v roce 2021. Hlavním cílem bylo navrhnout jazyk, který by pomohl vývojářům psát robustnější

a udržovatelnější kód. TypeScript se kompiluje právě do čistého JavaScriptu, což zajišťuje, že je kompatibilní se všemi prohlížeči či dalšími běhovými prostředími. Populárním se stal zejména ve velkých projektech, kde řeší problémy s rostoucí složitostí aplikací vyvíjených v JavaScriptu. Toho bylo dosaženo díky jeho nástrojům pro lepši detekci chyb a lepší dokumentaci kódu. Díky TypeScriptu je mnoho chyb odhaleno už během kompilace a je tak zabráněno nečekaným chybám již nasazeného systému. Při kompilaci máme možnost výběru cílové verze ECMAScript, do které bude kód napsaný v TypeScriptu přeložen. Díky tomu můžeme při vývoji používat nejmodernější funkce, například funkce přidané v ES6, a následně kód přeložit do verze ES3. To nám zajistí nejlepší vývojářský zážitek a zároveň stoprocentní kompatibilitu se starými prohlížeči.

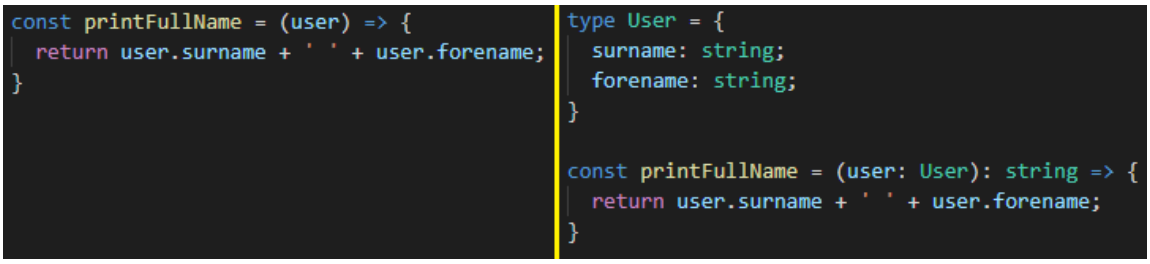
Mezi klíčové výhody TypeScriptu patří:

- **Statická typová kontrola:** Umožňuje definovat datové typy proměnných, parametrů a návratových hodnot funkcí.
- **Podpora nejmodernějších funkcí:** Podporuje všechny moderní funkce dostupné v ES6 a novějších.
- **Kompatibilita s JavaScriptem:** Každý validní kód v JavaScriptu je automaticky validní kód v TypeScriptu.
- **Možnost výběru cílového kódu:** Při kompilaci možnost výběru cílové verze ECMAScript.
- **Silné nástroje pro vývojáře:** Díky typům nabízí lepší podporu a automatické napovídání pro IDE.

Krom statických typů TypyScript přináší také další nové pokročilé funkce, které v klasickém JavaScriptu nenašli. Jmenovitě například genericitu, výčtový typ enum či rozhraní nebo typové aliasy pro přesné definování struktury objektů. Stal se natolik populární, že moderní frameworky React, Vue.js a Angular nabízí jeho plnou podporu. Angular je dokonce vyvinutý přímo s podporou TypeScriptu a v nejnovějších verzí je jeho použití vynucené.

Následující obrázek porovnává dvě stejné funkce, přičemž první je napsaná v čistém JavaScriptu a druhá pomocí TypeScriptu. TypeScript nám zajistí, že do funkce opravdu vstoupí objekt, který bude mít požadované vlastnosti. V JavaScriptu by se mohlo stát, že

omylem do funkce jako parametr vstoupí například textový řetězec a program se ukončí chybovou hláškou o čtení neexistující vlastnosti daného textového řetězce.



Obrázek 13: Porovnání jazyků JavaScript a TypeScript (Zdroj: Autor)

TypeScript přináší do světa JavaScriptu spoustu výhod, a hlavně mnohem lepší detekci chyb. Díky jeho schopnostem zlepšit kvalitu kódu a celkový vývojářský prožitek a produktivitu, se stal dobrou volbou pro vývoj moderních a robustních webových aplikací.

4.2.5 React

React je populární JavaScriptová knihovna pro tvorbu dynamických webových aplikací. Byl vyvinut firmou Facebook a oficiálně představen v roce 2013. Primárním cílem bylo vytvořit nástroj pro snadnější vývoj komplexních interaktivních webových aplikací na základě komponentového přístupu a efektivního renderování. Vývoj byl motivován potřebou efektivního nástroje pro vývoj složitého uživatelského rozhraní pro platformu Facebook. Od té doby tato knihovna ušla dlouhou cestu a nyní je spravována jak Facebookem, tak i otevřenou komunitou vývojářů.

Podobně jako ostatní moderní JavaScriptové frameworky je React založen na komponentovém přístupu. Vývojářům tedy umožňuje vytvářet aplikace na základě samostatných, znovu použitelných komponent. Jednotlivé komponenty obsahují jak vzhled, tak i logiku. Díky tomu máme vše potřebné pro danou funkcionalitu přehledně na jednom místě. React využívá JSX, což je rozšíření, které umožňuje psát HTML kód přímo do kódu JavaScriptu. To nám velice usnadní manipulaci s DOM strukturou. JSX by se dal nazvat takovou šablonou do které můžeme přímo vkládat například proměnné, jejíchž hodnota bude automaticky vložena do vyrenderovaného obsahu. Veškerý takto dynamický obsah vkládáme do JSX pomocí složených závorek. Jednotlivé komponenty si mohou předávat data v podobě tzv. props, které se ke komponentě přiřazují stejně, jako atributy k běžným HTML prvkům.

Renderování je v Reactu řešeno pomocí virtuálního DOM, který slouží jako takový odraz skutečného DOM. Tento virtuální DOM si React uchovává ve své paměti. Pokaždé když se se změní stav aplikace, tak React vytvoří nový virtuální DOM, ten následně porovná s předchozím, a nakonec zapíše změny do reálného DOM. Tím je zajištěno, že bude prováděn pouze minimální počet změn, což vede ke zlepšení výkonu.

Mimo stavové proměnné a metody mohou komponenty v Reactu obsahovat také speciální funkce, které reagují na určité změny. Můžeme například provést nějakou akci pokaždé, když se nám změní hodnota sledované proměnné. Těmto funkcím se v Reactu říká vedlejší efekty. Jsou obzvláště užitečné pro načítání dat z API, protože pomocí nich zajistíme, že se dotaz na API odešle pouze při prvním načtení komponenty.

Oproti rozsáhlému frameworku typu Angular obsahuje React pouze základní funkce. Pokud chceme použít například router musíme jej doinstalovat samostatně. Stejně tak je tomu u mnoha dalších standardních knihoven, které v Reactu běžně použijeme. Ekosystém okolo Reactu je ovšem rozsáhlý a máme na výběr z opravdu velkého množství různých knihoven, které můžeme do aplikace přidat. Tento modulární přístup nám zajistí, že aplikace bude minimální a bude obsahovat pouze funkce, které opravdu potřebujeme.

```
import { useState } from "react";

function Counter() {
  const [count, setCount] = useState<number>(0);

  const incrementCounter = () => {
    setCount(count + 1);
  };

  return (
    <div>
      <p>You clicked {count} times</p>
      <button onClick={incrementCounter}>Click me</button>
    </div>
  );
}

export default Counter;
```

Obrázek 14: Ukázka komponenty v knihovně React (Zdroj: Autor)

React se stal jedním z nejpopulárnějších a nejefektivnějších nástrojů pro tvorbu moderních webových aplikací. Jeho komponentový přístup, flexibilita, rozsáhlý ekosystém

a silná aktivní komunita umožňují vytvářet robustní, škálovatelné a uživatelsky přívětivé aplikace.

4.2.6 Material UI

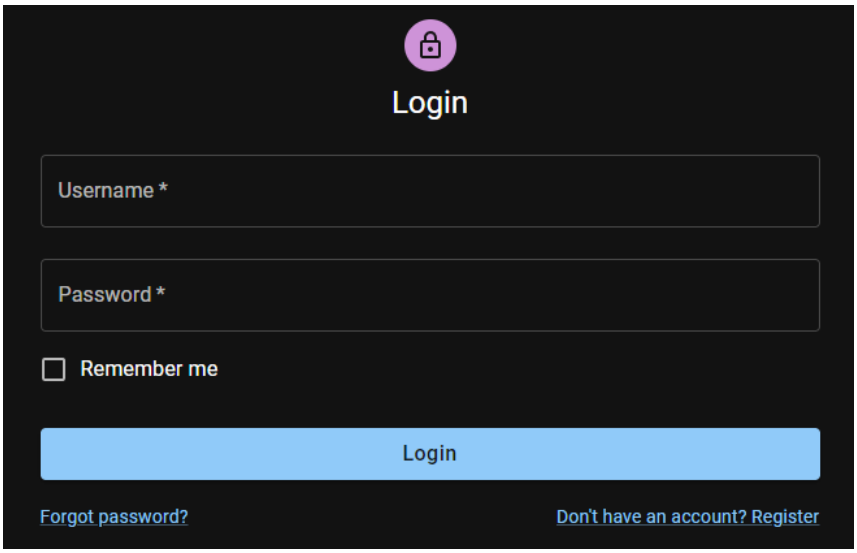
Material UI je populární knihovna pro React obsahující mnoho předpřipravených komponent pro snadnější vývoj webových aplikací. Všechny komponenty implementují Material Design, což je designový a grafický standard od společnosti Google. Knihovna byla vydána v roce 2014 jako open source projekt. Od té doby je aktivně spravován a vylepšován širokou komunitou vývojářů. Použití této knihovny nám poskytne snadný vývoj esteticky příjemného vzhledu naší aplikace. Mimo designových komponent nám knihovna nabídne i mnoho dalších zajímavých komponent, které nám usnadní vývoj. Nalezneme zde například komponenty pro vytvoření našeptávače, dialogového okna nebo dynamických přechodů. To by bylo obvykle řešeno instalací samostatných komponent, ale díky Material UI je k dispozici vše na jednom místě, a navíc se standardizovaným designem.

Mezi základní přednosti knihovny patří:

- **Předpřipravené komponenty:** Obsahuje velké množství funkčních komponent, kterými jsou například tlačítka, formulářové prvky a mnoho dalšího.
- **Responsivní design:** Veškeré komponenty jsou plně responsivní a zajišťují tak dobré fungování na všech možných zařízeních.
- **Přizpůsobitelnost:** Lze snadno přizpůsobit vzhled jak globálně pomocí různých témat, tak vzhled jednotlivých komponent přetížením jejich výchozího nastavení.
- **Ikony:** Material UI nabízí širokou škálu zdarma přístupných ikonek ve vektorovém formátu SVG, který je snadno použitelný ve webových aplikacích.
- **Podpora TypeScriptu:** Material UI plně podporuje TypeScript, což díky typové kontrole a automatickému napovídání usnadní vývoj aplikace.

Použití knihovny je velice jednoduché. Po nainstalování stačí pouze importovat danou komponentu a použít ji stejné jako ostatní React komponenty v JSX šabloně. Komponenty mají spoustu možných atributů, kterými můžeme jejich ovlivnit vzhled a chování. Podrobný rozpis těchto dostupných atributů nalezneme v dokumentaci knihovny, která je velice dobře zpracovaná a přehledná. Knihovna podporuje již v základu několik možných témat, takže například implementace přepínání mezi světlým a tmavým režimem je díky

Material UI velice jednoduchá. Material UI také podporuje možnou integraci s různými standardními knihovnami, mezi které patří například React Router. Díky tomu můžeme vytvořit funkční React Router link, který bude zároveň automaticky nastylován pomocí Material UI.



Obrázek 15: Ukázka přihlašovacího formuláře vytvořeného pomocí Material UI (Zdroj: Autor)

Material UI je obsáhlá a flexibilní knihovna komponent, díky které můžeme snadno a rychle vytvářet moderní webové aplikace. Stává se stále častější volbou pro vývoj jak jednoduchých, tak komplexních aplikací psaných pomocí knihovny React.

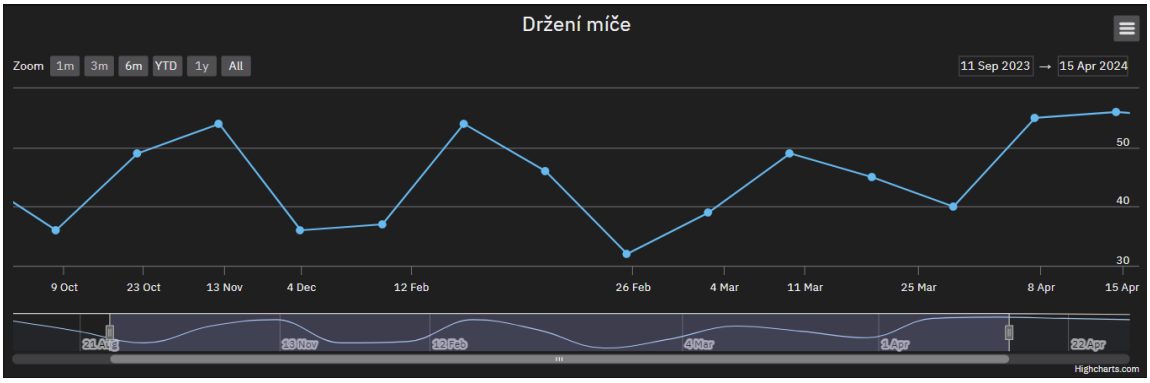
4.2.7 Highcharts

Highcharts je populární knihovna na vytváření interaktivních grafů na webových stránkách. Dle jejich vlastního popisku na toto řešení spoléhá 80 ze 100 největších světových společností, kterými jsou například Google, Samsung, Netflix či Visa. Nabízí desítky různých typů grafu, od těch základních až po složitější grafy časových řad nebo interaktivní geografické mapy. V nabídce jsou také kompletní vytvořené dashboardy, které nám umožní monitorovat spoustu dat na jednom místě. Highcharts lze integrovat do široké škály aplikací. Můžeme vložit přímo do webové stránky pomocí JavaScriptu, nebo využít speciální knihovny pro integraci s frameworky jako Angular, Vue.js, React a Svelte. Také lze využít integraci s aplikacemi napsanými v jazycích Python, PHP, na platformě .NET, nebo do mobilních operačních systémů Android a iOS. Veškeré nabízené grafy jsou plně responsivní a zvládnou opravdu velké množství dat. Použití knihovny v komerčních systémech

je ovšem podmíněno zakoupením licence, jejíž cena se odvíjí od potřebné škály grafů. Například grafy pro zobrazení časových řad či interaktivní geografické mapy vyjdou dráž než klasické sloupcové či koláčové grafy.

Mezi klíčové vlastnosti patří:

- **Široká škála grafů:** Obsahuje desítky možných typů grafů.
- **Interaktivita:** Všechny grafy jsou vysoce interaktivní. Máme například možnost různě přibližovat nebo se v grafu posouvat.
- **Responsivní design:** Všechny grafy jsou plně responsivní a optimalizované pro zobrazení na všech možných zařízeních.
- **Možnost exportu:** Nabízí možnost snadno exportovat grafy do souborů typu PNG, JPEG, SVG a PDF.
- **Podpora pro velká data:** Knihovna je optimalizována pro práci s opravdu velkým množstvím dat.
- **Jednoduchá integrace:** Snadná integrace s různými programovacími jazyky či frameworky.



Obrázek 16: Ukázka grafu časové řady vytvořeného knihovnou Highcharts (Zdroj: Autor)

Highcharts je opravdu propracovaná knihovna pro vytváření grafů a vizualizací dat. Díky své robustnosti a všem nabízeným možnostem je ideální volbou pro aplikace, které vyžadují dynamické a atraktivní vizualizace dat.

5 IMPLEMENTACE APLIKACE

Tato kapitola představí klíčové moduly aplikace a vysvětlí způsob, kterým byly naimplementovány. Kapitola bude rozdělena na dvě základní části. První část se bude věnovat implementaci serverové části aplikace a druhá část se bude věnovat klientské části aplikace. Bude také popsán způsob, jakým mezi sebou tyto dvě části komunikují.

5.1 Serverová část

Serverová část aplikace byla naimplementována jako RESTful API. Jak již bylo řečeno, tak pro implementaci byl zvolen framework Spring Boot. Prvním krokem bylo vytvoření prázdného projektu pomocí služby Spring Initializr. Při vytváření bylo do projektu přidáno několik tzv. starterů. Těmi máme na mysli speciální balíčky obsahující rozšiřující funkce určitého charakteru. Byl přidán například balíček pro připojení k MySQL databázi, pro práci s JWT, pro testovaní pomocí JUnit, pro zasílání emailů, pro autentizaci a autorizaci nebo pro validaci vstupních dat.

5.1.1 Základní nastavení

Po vytvoření projektu následovalo vytvoření struktury a základní konfigurace. Konfigurace základních systémových proměnných ve frameworku Spring Boot probíhá pomocí speciálního souboru *application.yaml*, který se nachází ve složce *resources*. Zde nadefinujeme například na jakém portu a URL aplikace poběží, hodnoty pro připojení k databázi či nastavení pro emailového klienta. Pro oddělení samotných citlivých hodnot, kterými mohou být například klíče a hesla byl vytvořen speciální soubor *.env*, který se neukládá do repozitáře Git a díky tomu se nám citlivá data nedostanou z lokálního počítače či serveru a každá stanice může mít nadefinované své vlastní hodnoty. Hodnoty ze souboru *.env* jsou následně injektovány do souboru *application.yaml*.

Následující dva obrázky znázorňují příklad nastavení těchto dvou souborů a injektaci hodnot. Pro ukázku je zobrazena pouze část definovaných hodnot, kompletní příklad konfiguračního souboru je k nalezení v příloze.

```
DB_HOST = localhost
DB_PORT = 3306
DB_NAME = fotstat
DB_USER = root
DB_PASSWORD =

DDL_MODE = validate

MAIL_USERNAME = test@email.com
MAIL_PASSWORD = abcabcabc
```

Obrázek 17: Ukázka souboru .env (Zdroj: Autor)

Jak je vidno z ukázky, tak soubor *.env* má velice jednoduchou strukturu, definujeme pouze klíče a jejich hodnoty. Názvy klíčů jsou libovolné a můžeme tak zvolit jakýkoliv název, který pak ale musíme následně správně referencovat v souboru *application.yaml*.

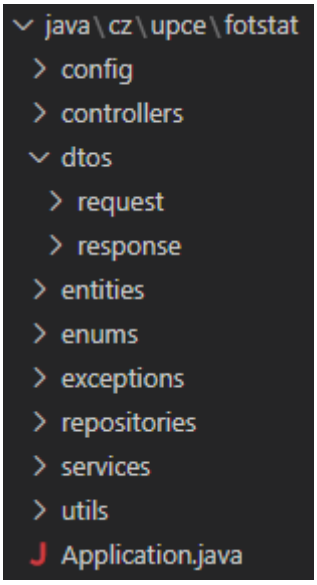
```yaml
server:
  port: 9000
  servlet:
    context-path: /api
spring:
  datasource:
    url: jdbc:mysql://${DB_HOST}:${DB_PORT}/${DB_NAME}
    username: ${DB_USER}
    password: ${DB_PASSWORD}
  jpa:
    hibernate:
      ddl-auto: ${DDL_MODE}
  mail:
    host: smtp.gmail.com
    port: 587
    username: ${MAIL_USERNAME}
    password: ${MAIL_PASSWORD}
```

Obrázek 18: Ukázka souboru application.yaml (Zdroj: Autor)

U souboru *application.yaml* je nutno dodržet jak strukturu souborů YAML, tak správně vyplnit předepsané hodnoty pro Spring Boot. Lze nadefinovat i hodnoty vlastní, které můžeme následně použít přímo v kódu. Lze si také všimnout injektnutí hodnot ze souboru *.env*. Správné hodnoty tohoto konfiguračního souboru nalezneme v dokumentaci ke Spring Boot nebo v dokumentacích jednotlivých přidaných knihoven. Nastavení správných hodnot je klíčové, jinak konfigurace bude chybná a aplikace nebude fungovat správně. Alter-

nativně nemusíme použít soubor typu YAML, ale soubor s příponou *.properties*, u kterého ale musíme všechny hodnoty vypisovat na samostatné řádky včetně jejich rodičovských klíčů. Použití souboru YAML nám tedy výrazně zpřehlední celý konfigurační soubor, protože jednotlivé podklíče můžeme shlukovat.

Po vytvoření a nastavení konfiguračních souborů následuje vytvoření adresářové struktury projektu. Výsledná adresářová struktura je zobrazena na následujícím obrázku a význam jednotlivých adresářů je následně podrobně vysvětlen.



Obrázek 19: Ukázka adresářové struktury serverové části (Zdroj: Autor)

- **config:** Obsahuje konfigurační třídy, například nastavení přístupových práv pro různé routy, JWT filter či Exception handler.
- **controllers:** Obsahuje kontrolery, které definují endpointy API, přijímají HTTP dotazy a vrací požadovaná data v HTTP odpovědi.
- **dtos:** Obsahuje pomocné datové třídy, které nám definují strukturu požadovaných dat v těle HTTP dotazu (podadresář *request*) či odpovědi (podadresář *response*).
- **entities:** Obsahuje datové entity JPA. Každá tabulka v databázi má zde odpovídající třídu.
- **enums:** Obsahuje výčtové typy.
- **exceptions:** Obsahuje vlastní definované výjimky.
- **repositories:** Obsahuje tzv. JPA repozitáře, které slouží pro komunikaci s databází.

- **services:** Obsahuje servisní třídy. Tyto třídy obsahují většinu logiky celé aplikace. Servisní vrstva slouží jako mezivrstva mezi kontrolery a repozitářemi.
- **utils:** Obsahuje pomocné třídy všeho druhu.

Soubor *Application.java* obsahuje metodu main a slouží jako spouštěč celé aplikace.

Posledním krokem před samotnou implementací je vytvoření patřičných konfiguračních tříd ve složce *config*, které nám zajistí správné fungovaní aplikace. Těmito třídami máme na mysli právě již zmíněný JWT filter, Exception handler, nastavení práv pro routy nebo konfigurace CORS.

CORS je speciální bezpečnostní mechanismus, který nám zajistí, že naše API bude schopna komunikovat s naší webovou aplikací. Ve výchozím stavu prohlížeč blokuje přístup k datům z jiné domény, než na které běží samotná aplikace. Jelikož naše API běží na samostatné doméně, je nutno tento krok provést.

Nyní je již vše připravené pro samotnou implementaci.

5.1.2 Autentizační a autorizační část

Základem většiny aplikací je autentizační a autorizační systém pro uživatele. Stejně tak je na tuto část kladen důraz i v této aplikaci. Systém rozlišuje celkem tři typy uživatel, nepřihlášený, přihlášený a přihlášený administrátor. Rozdíl mezi přihlášeným uživatelem a přihlášeným administrátorem je řešen na základě přiřazeného výčtového typu, který může nabývat právě dvou hodnot, *USER* a *ADMIN*. Administrátorovi je jeho role manuálně přiřazena a všichni nově registrovaní uživatelé automaticky obdrží roli *USER*. Jednotlivé routy jsou pak následně zabezpečeny právě kontrolou těchto rolí. Jednotlivé routy a jejich přiřazení k rolím jsou k nalezení v příloze.

Samotné ověřování uživatel je implementováno na základě JWT tokenu. Ten je uživateli vygenerován při každém přihlášení a při následně při každém dotazu na API je tento token zaslán v *Authorization* hlavičce. API nejprve tento token validuje a až následně vyhoví požadavku. Jestliže token není v dotazu přiřazen, anebo není validní, API odpoví HTTP kódem 401 *Unauthorized*. V rámci bezpečnosti je životnost JWT tokenu nastavena pouze na deset minut.

Aby se uživatel nemusel každých deset minut opětovně přihlašovat, tak je implementován systém tzv. refresh tokenů. Tyto tokeny slouží pro obnovení JWT. Jsou vygenerovány společně s JWT, ale na rozdíl od JWT jsou uloženy do databáze. Životnost refresh tokenů

je nastavena na základě uživatelského požadavku na zapamatování přihlášení. Jestliže uživatel při přihlášení zvolí možnost *Zapamatovat mě*, tak je životnost refresh tokenu nastavena na jeden rok. Jestliže tuto možnost nezvolí, je životnost nastavena na jeden den. Samotný mechanismus pro obnovení JWT je implementován tak, že při první odpovědi ze serveru se statusem 401 *Unauthorized* (uživateli vypršela platnost JWT tokenu) webová aplikace odešle dotaz na speciální endpoint pro obnovení JWT tokenu. K dotazu je přiložen právě refresh token, který je následně na serveru ověřen a na základě toho je vygenerován nový JWT token. Z bezpečnostních důvodů je vygenerován i nový refresh token, kterému je opět nastavena plná životnost. To znamená, že dokud bude uživatel aplikaci aktivně používat, nebude nucen se znovu přihlašovat, protože životnost refresh tokenu se bude automaticky prodlužovat. K automatickému odhlášení dojde až po vypršení životnosti tokenu bez toho, aniž by během té doby uživatel provedl jediný dotaz na API.

```
@Transactional
public JwtResponse refreshToken(RefreshTokenRequest request) {
  RefreshToken refreshToken = refreshTokenService.findByToken(request.getToken());
  User user = refreshToken.getUser();
  String jwt = jwtService.generateToken(user);
  RefreshToken newRefreshToken = refreshTokenService.create(user, refreshToken.getLongterm());
  return JwtResponse.builder().user(user.toDto()).jwt(jwt).refreshToken(newRefreshToken.getToken()).build();
}
```

Obrázek 20: Ukázka implementace metody pro obnovení JWT tokenu (Zdroj: Autor)

Pro uživatele jsou implementovány možnosti registrace, přihlášení do systému, změna hesla a obnovení zapomenutého hesla. Za podrobnější zmínku stojí funkcionalita pro obnovení zapomenutého hesla. Pro obnovení zapomenutého hesla uživatel zašle svoje přihlašovací jméno, API následně zkontroluje, zdali uživatel s tímto přihlašovacím jménem v systému existuje a pokud ano, tak na jeho email zašle náhodně vygenerovaný token. Tento token je uložen v databázi a je mu nastavena životnost pět minut. V dalším kroku uživatel zašle nově zvolené heslo společně s tímto tokenem. Jestliže je token validní, nové heslo je uživateli nastaveno.

5.1.3 Automatický sběr dat

Klíčovou částí systému je automatický sběr zdrojových dat z veřejného API. Jedná se o nejsložitější část v implementaci. Nejprve je nutno zajistit zasílání správných dotazů na

veřejné API a následně získaná data zpracovat a přetransformovat do podoby, která je uložena do databáze.

Se sběrem dat souvisí také několik vedlejších efektů, kterými je například zasílání automatických notifikací uživatelům, kteří mají tuto funkcionalitu nastavenou. Po načtení výsledků nových zápasů se zkontroluje, zdali pro tým, který daný zápas odehrál, nemá nějaký uživatel nastavenou možnost zasílání notifikací. Jestliže ano, je mu následně notifikace zaslána.

Samotné načítání jednotlivých zápasů je rozděleno do dvou částí. Nejprve se na začátku každé sezóny načtou všechny budoucí zápasy dané sezóny, které se následně uloží do databáze. V druhém kroku se v určitém intervalu projíždí jednotlivé zápasy a pokud již nastal jejich datum zahájení, tak jsou načítány výsledky a statistiky.

Načítání jednotlivých zápasů probíhá pomocí HTTP dotazu, kde do parametru *id* v URL adrese zadáme id daného zápasu. Pro načítání dat z veřejného API byla vytvořena pomocná funkce pro zaslání dotazu a navrácení získané hodnoty v podobě *JsonNode* objektu. Tento objekt nám výrazně ulehčí práci se složitějšími JSON objekty, které vrací veřejné API. Veškerá zpracovaná data jsou poté převedena na datové entity a uložena do databáze.

Na následujícím obrázku je zobrazena pomocná metoda pro načítání dat z veřejného API.

```
public JsonNode fetchData(String url) throws IOException, InterruptedException {
  HttpRequest request = HttpRequest.newBuilder()
      .uri(URI.create(rapidapiUrl + url))
      .header("X-RapidAPI-Host", rapidapiHost)
      .header("X-RapidAPI-Key", rapidapiKey)
      .method("GET", HttpRequest.BodyPublishers.noBody())
      .build();
  HttpResponse<String> response = HttpClient.newHttpClient().send(request, HttpResponse.BodyHandlers.ofString());
  return new ObjectMapper().readTree(response.body()).get("response");
}
```

Obrázek 21: Ukázka metody pro načítání dat z veřejného API (Zdroj: Autor)

Jak si lze z obrázku všimnout, tak do každého dotazu je nutno přidat dvě speciální hlavičky, díky kterým nás služba identifikuje a následně může vyhovět našemu požadavku.

5.1.4 Obecný popis implementace

Celé API je psáno ve stylu RESTful API, což znamená, že pro každou entitu existují endpointy pro její základní manipulaci. Konkrétně je na mysli pět základních endpointů, kterými jsou získání všech entit, získání jednotlivé entity na základě jejího unikátního identifikátoru, vytvoření nové entity, editace stávající entity na základě jejího unikátního identifikátoru a smazání dané entity. Tomuto přístupu se také říká *CRUD*, což je zkratka pro anglické Create (vytvořit), Read (číst), Update (editovat), Delete (smazat). Krom těchto základních endpointů je zde i několik doplňujících, které slouží například pro získání všech entit na základě cizího klíče. Mezi konkrétní příklady patří získání všech týmu z vybrané soutěže či všech zápasů odehraných jedním konkrétním týmem.

Nutno také podotknout, že většina endpointů, které manipulují se zdrojovými daty, čímž máme namysli právě vytváření, editaci a mazání, jsou dovoleny pouze administrátorovi systému. Běžní uživatele mohou tyto data pouze číst. Je to z toho důvodu, aby byla zdrojová data věrohodná a nemohl s nimi nikdo manipulovat a zaměňovat za chybné či dokonce mazat. Výjimkou jsou pouze endpointy pro manipulaci s daty v uživatelské části. Těmi jsou například uživatelské komentáře, editace vlastních profilu či nastavování notifikací.

5.2 Klientská část

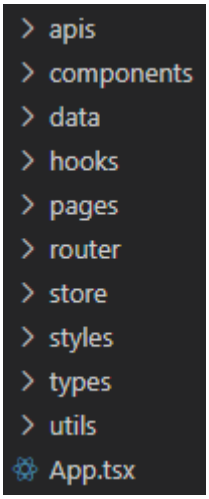
Implementace klientské části byla formou webové aplikace napsané pomocí knihovny React. Pro vizuální stránku aplikace byla použita knihovna Material UI. Samotná aplikace byla vytvořena pomocí nástroje Vite, což je moderní nástroj pro vytváření a sestavování projektů založených na jazyce JavaScript. Prvním krokem po vytvoření prázdného projektu bylo přidání jednotlivých závislostí a pomocných knihoven, kterými byly již zmíněné Material UI, Highcharts a další jako například knihovna pro routování, statistické výpočty či zasílání HTTP dotazů.

5.2.1 Základní nastavení

Stejně jako při implementaci serverové části bylo nejprve nutné provést základní konfiguraci projektu. Na rozdíl od serverové části zde není nutno konfigurovat tolik položek. Ze

základního nastavení v souboru *vite.config.ts* byl změněn pouze port, na kterém aplikace poběží. Ostatní nastavení, včetně kompilátoru pro TypeScript zůstalo výchozí.

Adresářová struktura projektu projektu je následující:



Obrázek 22: Ukázka adresářové struktury klientské části (Zdroj: Autor)

- **apis:** Obsahuje pomocné soubory pro zasílání HTTP dotazů.
- **components:** Obsahuje znovupoužitelné komponenty.
- **data:** Obsahuje opakující se pomocná data, například výčty.
- **hooks:** Obsahuje definici tzv. hooků, což jsou v knihovně React soubory se znovupoužitelnými funkcemi.
- **pages:** Obsahuje soubory definující jednotlivé stránky v aplikaci.
- **router:** Obsahuje definici routeru a všech jeho možných stránek.
- **store:** Obsahuje soubory pro knihovnu Redux, která slouží jako centrální úložiště pro sdílení dat mezi komponentami.
- **styles:** Obsahuje CSS soubory, ikony a další soubory definující vzhled aplikace.
- **types:** Obsahuje definované datové typy pro TypeScript.
- **utils:** Obsahuje soubory s pomocnými funkcemi.

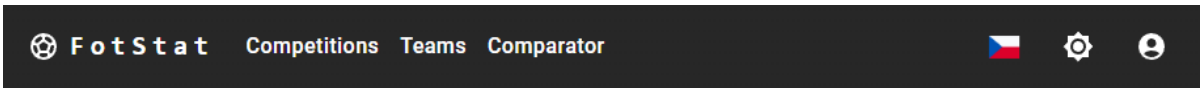
Soubor *App.tsx* slouží jako kořenová komponenta celé aplikace.

Posledním krokem v nastavení byla konfigurace jednotlivých pomocných knihoven. Například již zmíněné knihovny Redux, nastavení knihovny i18next pro multijazyčnost aplikace či základní definice stylu pro komponenty z knihovny Material UI.

5.2.2 Vzhled aplikace

Aplikace má velice jednoduchý a moderní design. Dělí se na dvě základní části. Na vrcholu stránky se nachází fixní hlavička, která obsahuje navigační menu a funkční tlačítka. Zbytek stránky tvoří samotný obsah, který je dynamicky vkládán a měněn pomocí React routeru. Aplikace je plně responzivní, takže je snadno čitelná na desktopovém i mobilním zařízení.

Fixní hlavička aplikace v desktopovém zobrazení vypadá následovně:

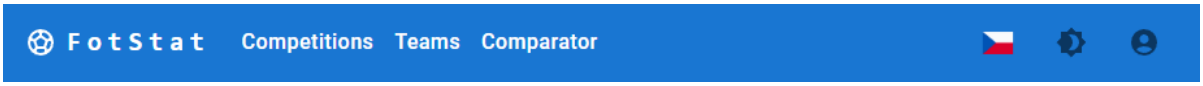


Obrázek 23: Ukázka hlavičky aplikace v desktopovém zobrazení (Zdroj: Autor)

Při zobrazení na mobilním zařízení je navigační menu schováno do rozbalovacího *Hamburger menu* a výsledná hlavička vypadá následovně:

Obrázek 24: Ukázka hlavičky aplikace v mobilním zobrazení (Zdroj: Autor)

Na pravé straně hlavičky se nachází tři tlačítka. První slouží pro přepínání jazyku aplikace a vždy lze měnit mezi češtinou a angličtinou. Druhé tlačítko přepíná barevný režim aplikace mezi světlým a tmavým zobrazením. Poslední tlačítko vyvolá rozbalovací menu uživatelské části, jako například odkaz na uživatelský profil, registraci, přihlášení a odhlášení.



Obrázek 25: Ukázka hlavičky aplikace ve světlém režimu aplikace (Zdroj: Autor)

5.2.3 Uživatelská část

V rámci uživatelské části je v aplikaci vytvořeno hned několik stránek. Pro nepřihlášené uživatele jsou zde samostatné stránky pro registraci, přihlášení a obnovení zapomenutého hesla. Pro přihlášené uživatele je zde možnost zobrazení vlastního profilu, editace profilu, změna hesla, nastavení notifikací a tlačítko pro odhlášení ze systému.

Na stránce s přihlášením do systému je zde možnost zaškrtnutí tlačítka *Zapamatovat mě*, díky kterému si aplikace bude pamatovat naše přihlášení a nebude nutné se po delší nečinnosti znovu přihlašovat.

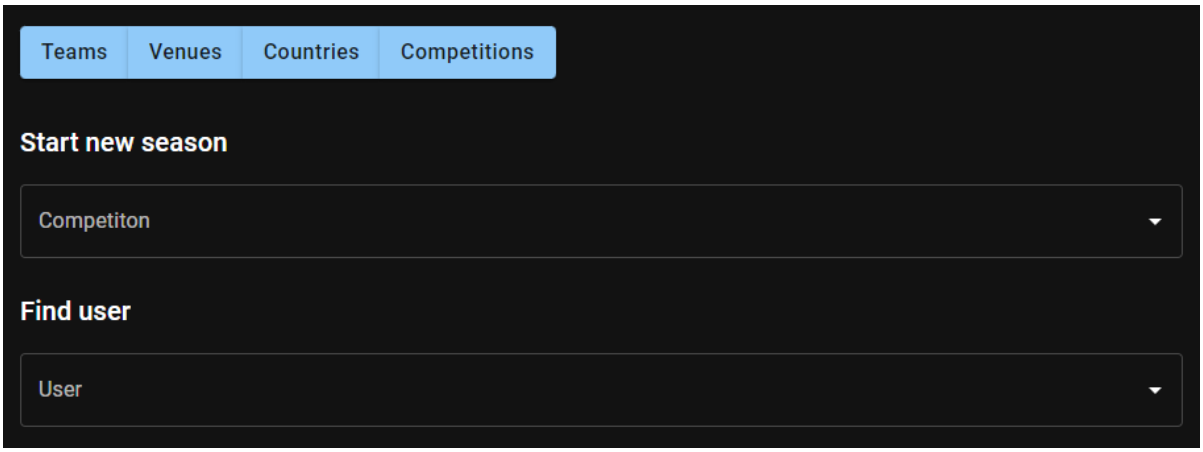
Celá ukázka přihlašovacího formuláře je zobrazena na obrázku 15.

5.2.4 Administrátorká část

Pro administrátora systému aplikace obsahuje speciální stránku s administrátorskými funkcemi. Jsou zde formuláře pro CRUD jednotlivých datových entit, které administrátorovi umožní manipulaci se zdrojovými daty. Příkladem může být přiřazování týmů k jednotlivým soutěžím, například pokud onen tým postoupí do vyšší soutěže nebo z ní naopak sestoupí.

Klíčovou funkcionalitou je tlačítko pro zahájení nové sezóny vybrané soutěže. To nám mimo jiné zajistí, že budou z veřejného API načteny všechny budoucí zápasy, které následně budou načítány v den jejich odehrání. Dojde také k vynulování bodové tabulky.

Poslední administrátorskou funkcionalitou je možnost vyhledat a zablokovat uživatele s nevhodnými komentáři. Vyhledat uživatele lze snadno pomocí našeptávače.



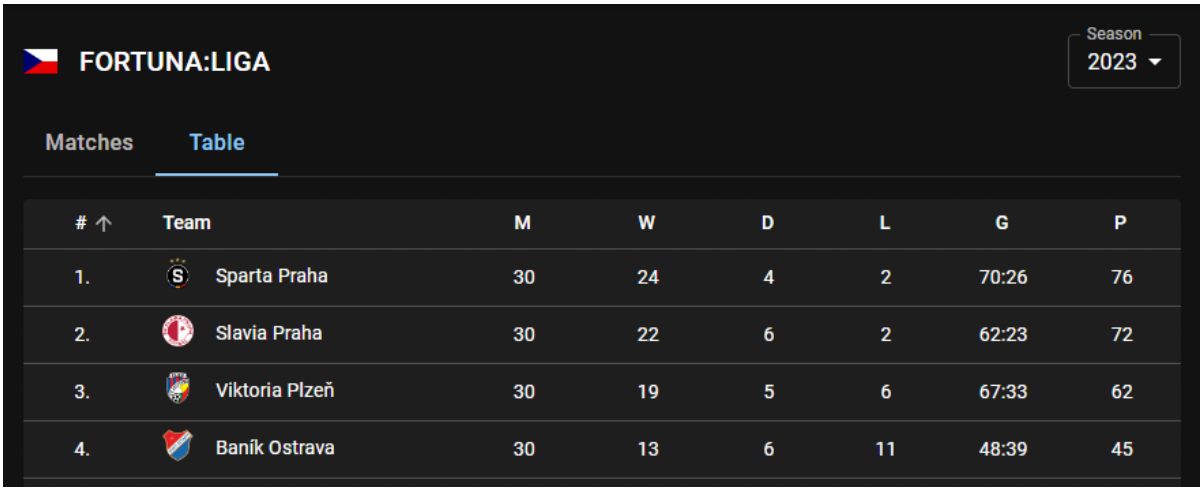
Obrázek 26: Ukázka stránky pro administraci (Zdroj: Autor)

5.2.5 Výsledky zápasů a bodová tabulka

Nyní již k hlavním prvkům celého systému. Hlavní stránka aplikace obsahuje výpis všech soutěží seskupených dle jednotlivých států. Detail každé soutěže obsahuje kompletní výpis

všech zápasů včetně jejich výsledků. Výpis zápasů je stránkován dle jednotlivých kol. Je zde možnost přepnout sezónu pro zobrazení historických dat.

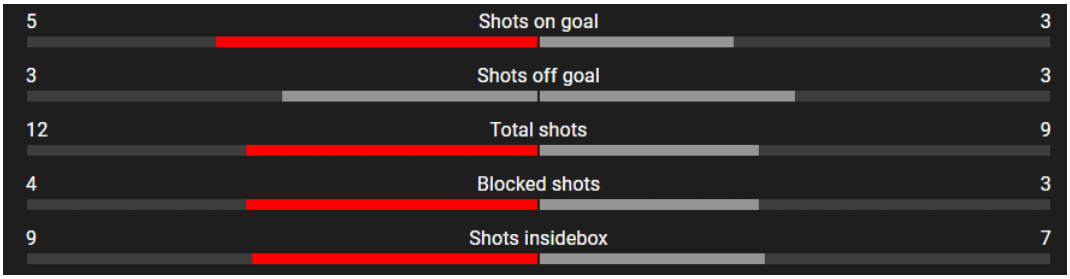
Na druhé kartě na detailu soutěže je k dispozici bodová tabulka obsahující všechny základní prvky, kterými jsou počet odehraných zápasů, počet výher, remíz a proher, vstřelené a obdržené góly a počet získaných bodů. Ve výchozím zobrazení je tabulka řazena na základě získaných bodů, ale aplikace umožňuje tabulku seřadit dle libovolných položek jak vzestupně, tak i sestupně.



#	Team	M	W	D	L	G	P
1.	Sparta Praha	30	24	4	2	70:26	76
2.	Slavia Praha	30	22	6	2	62:23	72
3.	Viktoria Plzeň	30	19	5	6	67:33	62
4.	Baník Ostrava	30	13	6	11	48:39	45

Obrázek 27: Ukázka bodové tabulky (Zdroj: Autor)

Po rozkliknutí jednotlivých zápasů se zobrazí detail zápasu, který obsahuje detailní statistiky ze zápasu a sekci s uživatelskými komentáři.



Obrázek 28: Ukázka zobrazení detailních statistik zápasu (Zdroj: Autor)

Zobrazení statistik je inspirováno zobrazením na jiných serverech, například na serveru Livesport. Pomocí barevných čar je vyobrazeno, jakou část z celkového počtu daný tým obsadil. Jestliže z deseti padlých gólů v zápase vstřelí první tým devět, tak čára na jeho straně bude vyplňovat 90 % celkové šířky. Oproti tomu u druhého týmu, který vstřelil

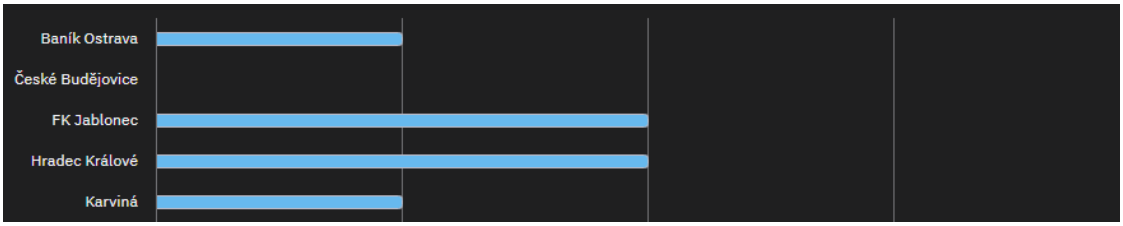
pouze jednu branku, bude čára vyplňovat pouze 10 %. Pro rychlejší vyznačení toho, který tým byl v určité statistice dominantnější, je čára na jeho straně označena červenou barvou.

5.2.6 Detailní statistiky jednotlivých týmů

Druhým hlavním prvkem systému je zobrazení celkových statistik jednotlivých týmů. Na stánce s detailem týmu jsou k nalezení dvě karty. První obsahuje výpis všech zápasů daného týmu a druhá podrobné zobrazení statistik. To se skládá ze několika hlavních částí. Na vrcholu je vždy přepínač s výběrem statistiky, kterou chceme sledovat, například počet vstřelených gólů nebo počet žlutých karet.

Následuje zobrazení časové řady, která přehledně zobrazuje, jakých hodnot tým dosahoval v jednotlivých po sobě jdoucích zápasech. Při najetí na libovolný bod v časové řadě je zobrazen proti jakému týmu byl daný zápas odehrán. Ukázka časové řady je zobrazena na obrázku 16.

Poté jsou zde dva sloupcové grafy, které přehledně zobrazují, jakých hodnot tým dosáhl proti jednotlivým soupeřům. To nám umožní získat lepší přehled o tom, proti kterému soupeři se zobrazovanému týmu dařilo více a proti kterému méně. První graf zobrazuje výsledky z domácích zápasů a druhý z venkovních.



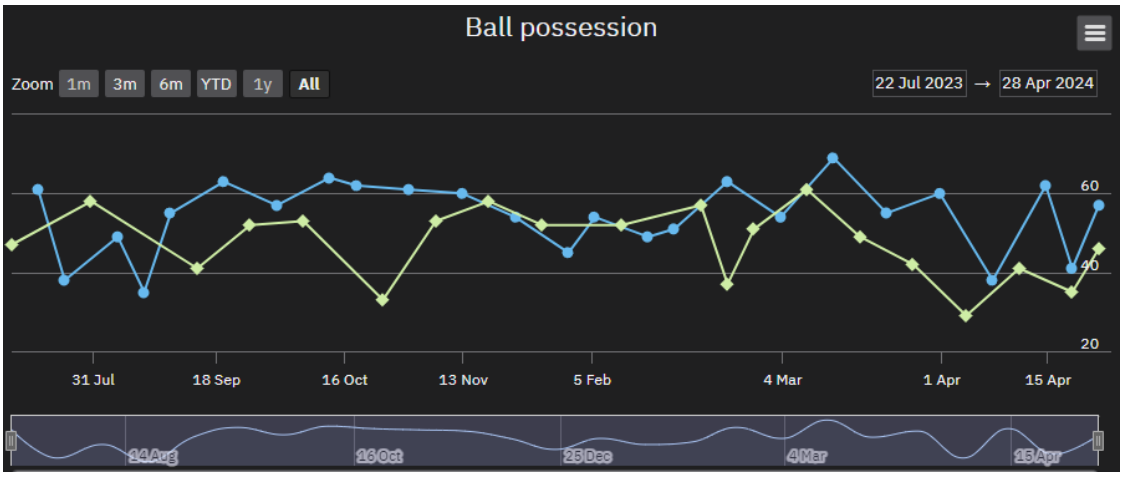
Obrázek 29: Ukázka sloupcového grafu zobrazujícího počet vstřelených gólů proti jednotlivým týmům (Zdroj: Autor)

Poslední položkou je tabulka s celkovými součty všech statistických dat. Ke každé statistické položce jsou zde zobrazeny minimální a maximální hodnoty, celkový součet, průměr, medián, modus a odchylka. Pro výpočet statistických ukazatel se využívá knihovna Simple Statistics, která je volně k dispozici na serveru NPM a obsahuje velké množství funkcí pro statistické výpočty. Tyto vypočtené statistiky lze zobrazit buď za celou sezónu, za poslední měsíc, za poslední týden, anebo za libovolně vybrané období.

5.2.7 Porovnávač statistik

Nyní k doplňujícím funkcionalitám. První z nich je systém na porovnávání statistik dvou vybraných týmu. Systém využívá stejných komponent, jaké jsou na detailním zobrazení statistik u jednotlivých týmů. Nalezneme zde tedy graf časové řady, sloupcové grafy určující výkon proti jednotlivým týmům a tabulku se statistickými výpočty. Hlavní rozdíl spočívá v tom, že grafy a tabulka obsahuje hodnoty dvou vybraných týmů. Díky tomu můžeme jejich výkony snadno porovnávat. Jednotlivé týmy jsou v grafech odděleny různými barvami a v tabulce se statistickými výpočty je barevně odděleno, který tým dosahuje lepších výsledků. Zelenou barvou jsou vyznačeny lepší hodnoty a červenou barvou ty horší.

Následující obrázek ukazuje graf časové řady porovnávající dva vybrané týmy v procentuálním držení míče.



Obrázek 30: Ukázka časové řady porovnávající výkon dvou týmů (Zdroj: Autor)

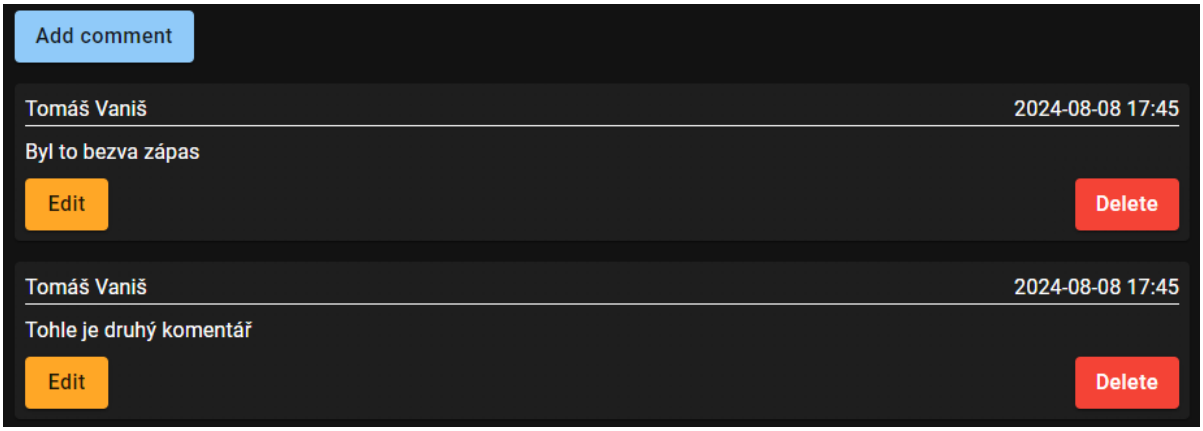
Následující obrázek ukazuje část tabulky s vypočtenými statistickými výpočty. Lze si všimnout barevného označení lepších a horších hodnot.

Indicator		Min	Max	Sum	Average	Median	Mode	Variance
Goals scored	Baník Ostrava	0	5	46	1.59	1	1	1.76
	Bohemians 1905	0	3	27	0.93	1	1	0.68
Goals received	Baník Ostrava	0	5	46	1.59	1	1	1.76
	Bohemians 1905	0	3	27	0.93	1	1	0.68
Shots on goal	Baník Ostrava	1	12	150	5.36	5	5	9.16
	Bohemians 1905	1	8	113	4.04	4	2	4.03

Obrázek 31: Ukázka tabulky porovnávající statistická data dvou týmů (Zdroj: Autor)

5.2.8 Komentářová sekce

Poslední částí, která stojí za zmínku je sekce s uživatelskými komentáři. Tuto sekci nalezneme pod každým jednotlivým zápasem. Komentářová sekce je přístupná pouze přihlášeným uživatelům. Uživatelé mohou komentáře libovolně přidávat, editovat i mazat. Editovat a mazat mohou pouze svoje vlastní komentáře, jedinou výjimkou je administrátor systému, který může mazat nevhodné komentáře všem uživatelům. Pokud by některý uživatel psal nevhodné komentáře opakovaně, má administrátor možnost daného uživatele zablokovat a zamezit mu tak možnost přidávání nových komentářů. Všechny komentáře jsou řazeny sestupně dne data přidání, takže nejnovější jsou vždy na prvním místě. Pro komentáře je také implementováno stránkování.



Obrázek 32: Ukázka komentářové sekce (Zdroj: Autor)

6 TESTOVÁNÍ APLIKACE

Předposlední kapitola je zaměřena na testování vyvíjené aplikace. Popíše jednak konkrétní případy testování aplikace, tak i obecně představí jednotlivé druhy testů a jejich využití.

Testování je kritickou částí procesu vývoje softwaru, která nám zajišťuje, že výsledný produkt neobsahuje žádné chyby a splňuje všechny požadavky a očekávání cílových uživatelů. Každý systém by měl být před nasazením řádně otestován, a to hned několika různými způsoby. V zásadě lze rozlišit manuální a automatické testování. Manuální testovaní znamená, že buď sám vývojář, anebo specializovaný tester manuálně projde celý systém či určitou testovanou část a otestuje, zdali se vše chová tak, jak by mělo. Oproti tomu automatické testovaní zahrnuje napsání automatických testů, které otestují určitou funkcionalitu a porovnají, zdali se výsledné hodnoty rovnají očekávaným.

6.1 Manuální testování

Manuální testování serverové části aplikace probíhalo pomocí nástroje Swagger. Jedná se o open source framework pro dokumentaci a testování RESTful API, který využívá specifikace OpenAPI.

Konfigurace nástroje Swagger je vcelku jednoduchá. Po přidání patřičného starteru nám stačí pouze vytvořit jeden konfigurační soubor typu Java, ve kterém nadefinujeme například URL na které má být nástroj dostupný. Při použití Spring Security nebo jiného filtru je nutno URL pro Swagger označit jako veřejné, jinak by nástroj nebyl přístupný. Výchozí URL pro Swagger lze přepsat v souboru *application.yaml*. Nadefinovat lze také autentizační schéma, které bude v nástroji použito. V tomto případě jde o použití tokenu JWT. Další možnosti konfigurace lze nalézt v dokumentaci nástroje Swagger.

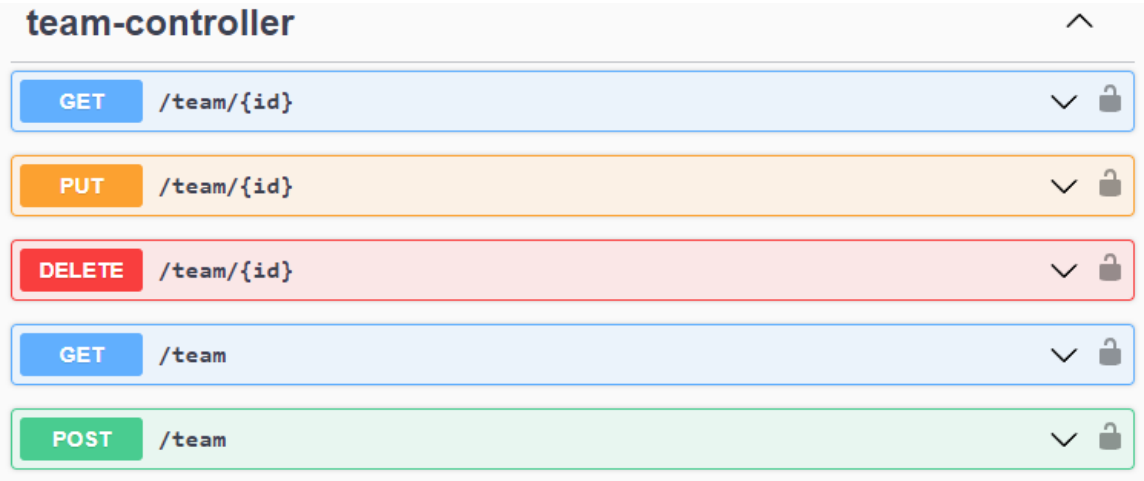
Konkrétní konfigurace v tomto projektu je zobrazena na následujícím obrázku.

```java
@Configuration
public class SwaggerConfiguration {
  @Bean
  OpenAPI openApi() {
    Server localServer = new Server().url("http://localhost:9000/api").description("Localhost Server URL");
    Info info = new Info().title("FotStat").version("V1.0.0");
    SecurityScheme securityScheme = new SecurityScheme()
        .type(SecurityScheme.Type.HTTP).bearerFormat("JWT").scheme("bearer");
    return new OpenAPI()
        .addSecurityItem(new SecurityRequirement().addList("Bearer Authentication"))
        .components(new Components().addSecuritySchemes("Bearer Authentication", securityScheme))
        .info(info).addServersItem(localServer);
  }
}
```

Obrázek 33: Ukázka konfigurace nástroje Swagger (Zdroj: Autor)

Práce s nástrojem je velice intuitivní. Swagger automaticky mapuje veškeré endpointy API a zobrazuje je po skupinách dle jednotlivých kontrolerů. Díky tomu je procházení jednotlivých endpointů velice přehledné. Mimo jednotlivých endpointů zde nalezneme kompletní výpis schémat, které do API vstupují nebo vystupují. Jedná se jednak o samotné entity, tak i o všechny objekty typu DTO. V hlavičce nástroje je k nalezení také tlačítko *Authorize*, které nám otevře jednoduché dialogové okno, do kterého lze zadat platný token JWT a všechny dotazy odeslané nástrojem budou obsahovat právě tento vložený token. Díky tomu lze otestovat i bezpečnost a endpointy, které vyžadují ověření uživatele.

Po rozkliknutí jednotlivých endpointů nástroj zobrazí kompletní strukturu vstupních i výstupních parametrů, a to včetně datových typů jednotlivých položek. Díky tomu lze jednoduše testovat i endpointy se složitější vstupní strukturou, protože máme vše dopředu předepsané a stačí pouze zaměnit za reálné hodnoty.



Obrázek 34: Ukázka použití nástroje Swagger (Zdroj: Autor)

Použití nástroje Swagger je velice jednoduché a značně ulehčuje vývoj celého RESTful API. Výsledná API je následně automaticky přehledně zdokumentovaná. Použití tohoto nástroje je při vývoji API výrazně doporučeno.

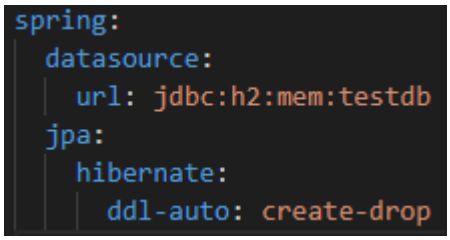
6.2 Automatické testování

Pro API bylo napsáno i několik automatických testů, které byly rozděleny do třech kategorií, dle zaměření daných testů. Těmito kategoriemi jsou jednotkové testy, integrační testy a repozitářové testy. V následujících podkapitolách budou jednotlivé kategorie testů vysvětleny a uvedeny konkrétní příklady implementací testů.

6.2.1 Základní nastavení

Před použitím automatických testů je nutno nejprve připravit testovací prostředí. Prvním krokem je přidání potřebných knihoven. Všechny potřebně knihovny lze zajistit přidáním jediného starteru pro testování, který obsahuje mimo jiné framework JUnit, který je určený právě pro psaní automatických testů v jazyce Java. Po přidání tohoto starteru jsou k dispozici veškeré prostředky potřebné pro všechny tři, již zmíněné, typy automatických testů.

Druhým krokem je přidání a nastavení testovací databáze. Pro jednotkové testy tento krok není nutný, ale pro integrační a repozitářové testy ano. Je to z toho důvodu, aby testy, které využívají právě databázi, pracovaly odděleně oproti hlavní produkční databázi. Pro tento účel jsou obvykle využívány jednoduché databáze, kterými jsou například tzv. in-memory databáze, které ukládají data pouze v paměti daného procesu, anebo například databázový systém SQLite. V tomto případě byla využita databáze H2, která je právě zástupce in-memory databází a Spring Boot s ní umí nativně pracovat. Pro její použití stačí pouze přidat potřebnou závislost a v konfiguračním souboru nastavit připojení. Konfigurační soubor je velice podobný hlavnímu konfiguračnímu souboru *application.yaml*. V kořenové složce s testy je nutno vytvořit opět složku *resources* a v ní patřičný konfigurační soubor pojmenovaný *application-test.yaml*. Následně již stačí všechny testy třídy s testy označit anotací *ActiveProfiles* a do ní vložit hodnotu *test*. Díky tomu bude framework využívat právě tento konfigurační soubor, a ne ten hlavní. Celý konfigurační soubor *application-test.java* je vyobrazen na následujícím obrázku.



Obrázek 35: Ukázka konfiguračního souboru pro automatické testy (Zdroj: Autor)

Jak si lze všimnout, tak konfigurace není nijak složitá. První věcí, kterou je nutno nastavit je samotné připojení k H2 databázi a druhou věcí je nastavení JPA schématu. Hodnota *create-drop* znamená to, že při spuštění aplikace bude vytvořeno databázové schéma a při jejím ukončení bude naopak odstraněno. Díky tomu budou všechny testy začínat s čistou, nově vytvořenou databází.

Po splnění těchto dvou kroků je již prostředí plně připraveno pro psaní a spouštění automatických testů.

6.2.2 Jednotkové testy

Jednotkové testy jsou základními testy, které testují vždy pouze jednu danou funkcionalitu (jednotku). Obvykle je tomu například výsledek nějaké metody, který je následně porovnán s očekávanou hodnotou. Jestliže se výsledná hodnota rovná očekávané, tak test proběhl úspěšně.

V rámci vývoje byly napsány celkem tři jednotkové testy. V aplikaci není využito moc složitých výpočtů, takže tyto jednotkové testy testují zejména pomocné metody.

Jednotlivé testy jsou následující:

- Test pro generování a validaci JWT tokenu.
- Test pro pomocnou metodu na převod časového razítka na datum.
- Test pro pomocnou metodu na extrahování číslic z řetězce pomocí regulárních výrazů.

Na následujícím obrázku je zobrazen příklad právě testu metody pro extrahování čísel z textového řetězce. Lze si také všimnout použití anotace *ActiveProfiles* pro načtení testové konfigurace.

```java
@SpringBootTest
@ActiveProfiles("test")
public class NumberExtractorTest {
  @Test
  void testExtract() {
    assertEquals(NumberExtractor.extract("avd2mhn1dv"), 21);
    assertEquals(NumberExtractor.extract("7"), 7);
    assertEquals(NumberExtractor.extract("acsD"), null);
    assertEquals(NumberExtractor.extract(""), null);
  }
}
```

Obrázek 36: Ukázka jednotkového testu (Zdroj: Autor)

6.2.3 Integrační testy

Integrační testy oproti jednotkovým mohou testovat hned několik prvků současně, které spolu obvykle spolupracují. Mohou testovat například celé endpointy aplikace. Často například pracují i s databází, proto bylo potřeba nejprve právě testovací databázi nastavit.

Integrační testy byly napsány celkem čtyři. Všechny testují celé endpointy aplikace pomocí modulu *MockMvc*.

Integrační testy jsou následující:

- Test pro přihlašování (endpoint */login*).
- Test pro registraci (endpoint */register*).
- Test pro refresh token (endpoint */refresh-token*).
- Test pro obnovení hesla (endpoint */user/new-password*).

Při používání testů je často nutné předpřipravit testovací data. K tomu slouží speciální metody, které jsou provedeny před samotnými testy. Tyto metody jsou označeny anotacemi, které udávají, kdy se má daná metoda zavolat. Existují čtyři základní anotace. Jsou jimi *BeforeAll*, *BeforeEach*, *AfterAll*, *AfterEach*. Metoda označená anotací *BeforeAll* se provede pouze jednou před spuštěním jednotlivých testů. Oproti tomu metoda označená anotací *BeforeEach* se provede před spuštěním každého testu. Podobně fungují metody s anotací *AfterAll* a *AfterEach*, které se ovšem provádějí až po skončení testů. Ty mohou sloužit například k vyčištění dat v testovací databázi. V tomto případě byly využity metody *BeforeEach* pro jednoduché vložení testovacích dat do databáze.

Na následujícím obrázku je zobrazen příklad integračního testu, který testuje endpoint */login* pro přihlášení do systému.

```
@Test
void testLogin() throws Exception {
  mockMvc.perform(post("/login")
      .content(ObjectConverter.asJsonString(new LoginRequest("tomas", "heslo1234", false)))
      .contentType(MediaType.APPLICATION_JSON)
      .accept(MediaType.APPLICATION_JSON))
      .andExpect(status().isOk())
      .andExpect(content().contentType(MediaType.APPLICATION_JSON_VALUE))
      .andExpect(jsonPath("$", hasKey("user")))
      .andExpect(jsonPath("$", hasKey("jwt")))
      .andExpect(jsonPath("$", hasKey("refreshToken")));

  mockMvc.perform(post("/login")
      .content(ObjectConverter.asJsonString(new LoginRequest("tomas", "heslo12345", false)))
      .contentType(MediaType.APPLICATION_JSON)
      .accept(MediaType.APPLICATION_JSON))
      .andExpect(status().isUnauthorized());
}
```

Obrázek 37: Ukázka integračního testu (Zdroj: Autor)

Při vyhodnocování integračního testu můžeme sledovat hned několik aspektů. Konkrétně je tomu status HTTP odpovědi, typ vráceného obsahu a kontrola, zdali vrácený objekt obsahuje všechna očekávaná data.

V rámci testu jsou provedeny celkem dva dotazy na stejný endpoint. Liší se v tom, že první dotaz obsahuje správné heslo a druhý nesprávné. U prvního dotazu se následně kontroluje, zdali dotaz prošel správně a byla vrácena správná data a u druhého dotazu, zdali API odpovědělo správně chybným HTTP kódem, že se ověření nezdařilo.

6.2.4 Repozitářové testy

Posledním typem použitých testů jsou repozitářové testy. Jedná se o speciální druh testu ve frameworku Spring Boot, které slouží pro ověření, zdali manipulace s databází prostřednictvím JPA funguje správně. Tyto testy se označují speciální anotací *DataJpaTest*, díky které se při spuštění testu načítá pouze konfigurace pro JPA, a ne pro celý projekt, jako je tomu u použití standardní anotace *SpringBootTest*.

V ramci projektu byly napsány dva testy tohoto typu, které slouží převážně jako ukázka, protože psaní testů pro všechny repozitáře by bylo velice repetitivní.

Repozitářové testy jsou následující:

- Test pro uložení uživatele a jeho načtení na základě přihlašovacího jména.
- Test pro uložení státu a jeho načtení na základě unikátního identifikátoru ID.

Na následujícím obrázku je zobrazen příklad repozitářového testu, který testuje ukládání uživatel do databáze a jejich opětovné načítání.

```
@Test
void testSaveAndFindByUsername() {
    User user = User.builder()
            .forename("Tomáš")
            .surname("Vaniš")
            .username("tomas")
            .email("tomas@test.com")
            .password("heslo1234")
            .role(Role.USER)
            .status(UserStatus.ACTIVE)
            .build();
    userRepository.save(user);

    Optional<User> result = userRepository.findByUsername("tomas");
    assertTrue(result.isPresent());
    User foundUser = result.get();
    assertEquals(foundUser.getForename(), "Tomáš");
}
```

Obrázek 38: Ukázka repozitářového testu (Zdroj: Autor)

7 POTENCIÁLNÍ ROZŠÍŘENÍ

Poslední krátká kapitola představí potenciální budoucí rozšíření systému. Funkcionality, které během tvorby práce nebyly implementovány, ale mohly by být zajímavé a užitečné pro budoucí použití systému.

7.1 Více soutěží

Prvním potenciálním rozšířením, které přichází v úvahu je přidání více soutěží. Systém je pro tento krok plně připraven. Díky více soutěžím by byl systém konkurence schopný oproti velkým systémům typu Livesport. Teoreticky by bylo možno přidat tolik soutěží, kolik jich poskytuje námi zvolené veřejné API. Problém by ovšem nastal v počtu prováděných dotazů na veřejné API. Platební model zdarma nám umožňuje pouze 100 dotazů denně. Pro pokrytí více soutěží bychom byli nuceni přistoupit k placenému plánu. Z toho důvodu by bylo nutné zavést systém předplatného i do našeho systému, abychom pokryli náklady na provoz.

7.2 Systém předplatného

Jak již bylo avizováno, tak rozšíření systému o další soutěže by vyžadovalo přidání systému předplatného. Registrovaní uživatelé, kteří by chtěli přístup ke statistikám u více možných soutěží by museli mít zakoupený měsíční odběr. Potenciálně by mohla být přidána i možnost ročního předplatného. Předplatné by bylo formou online plateb kartou přes platební bránu. Neplatící uživatelé by měli přístup pouze k jedné soutěži.

7.3 Automatická predikce

Poslední zajímavou rozšiřující funkcionalitou by mohla být automatická predikce budoucích výsledků a statistických dat. Predikce by probíhala na základě analýzy a vyhodnocování historických dat. Jednalo by se o prémiovou funkcionalitu, která by byla podmíněna placeným platebním plánem. Funkcionalita by poskytla lepší nadhled nad budoucími zápasy a mohla by se stát také oporou při sázení.

ZÁVĚR

Tato diplomová práce se zaměřila na vývoj informačního systému určeného pro zpracování a vizualizaci sportovních statistik. Hlavním cílem práce bylo vytvořit systém, který by automaticky sbíral rozsáhlá sportovní data, provedl jejich analýzu a následně uživatelům poskytl jejich efektivní a přehledné zobrazení.

Prvním krokem při realizaci tohoto cíle byla důkladná analýza požadavků na systém. Na základě této analýzy byl proveden návrh celého systému, který zahrnoval hned několik důležitých kroků, od určení jednotlivých aktérů a jejich možností až po kompletní návrh datového modelu. Takto vypracovaný návrh se stal pevnou oporou v průběhu samotné implementace. Díky tomu bylo zajištěno splnění všech stanovených požadavků.

Důležitým krokem byla také analýza existujících konkurenčních řešení. Během analýzy bylo zjištěno, že žádný z aktuálně nejpoužívanějších systému neposkytuje hlavní požadované funkce stanovené v této diplomové práci. Díky tomu se po dosažení stanovených cílů práce stala potenciálním přínosem do tohoto oboru.

Před samotnou implementací bylo nutné vybrat vhodný datový zdroj. Práce tedy poskytla rešerši několika možných veřejných API poskytujících reálná sportovní data. Na základě této rešerše byl zvolen vhodný datový zdroj, který vyhovoval všem předem stanoveným výběrovým kritériím.

Během implementace byl kladen důraz nejen na splnění všech stanovených požadavků, ale i na to, aby byl výsledný systém intuitivní a snadno použitelný i pro méně znalé uživatele. Zároveň byl systém vyvíjen tak, aby byl, co nejvíce modulární a škálovatelný, což umožňuje jeho snadnou údržbu a možnost budoucího rozšíření. Samozřejmostí při vývoji bylo dodržení nejlepších praktik a zásad pro tvorbu přehledného a čistého kódu. Výsledkem je systém, který nabízí přehledné interaktivní prostředí pro vizualizaci a porovnávání sportovních statistik.

Po dokončení vývoje byl systém řádně otestován, a to jak manuálně, tak použitím automatických testů. Bylo vytvořeno několik testovacích scénářů různých druhů, od jednoduchých jednotkových testů až po složitější integrační testy. Práce také vysvětlila jak důležitou roli hraje testování v celém procesu vývoje systému. Jde o zásadní krok, díky kterému se výsledný systém dostane do rukou cílových uživatelů v požadovaném stavu bez nežádoucích chyb.

K závěru práce nabídla krátké shrnutí možných rozšíření systému do budoucna, díky kterým by se mohl stát ještě více konkurence schopný velkým systémům, které se aktuálně v této oblasti působnosti vyskytují.

Na závěr lze konstatovat, že práce splnila veškeré své stanovené cíle a požadavky. Výsledný software nabídl zcela nové možnosti a pohled, kterým lze na sportovní statistiky nahlížet. Systém jistě ocení uživatelé s bližším zájmem o sportovní statistky a jejich vizualizaci.

Zároveň se práce stala přínosem pro samotného autora, kterému poskytla cenné zkušenosti s vývojem moderních webových aplikací.

POUŽITÁ LITERATURA

[1] ARLOW, Jim a NEUSTADT, Ila. *UML 2 a unifikovaný proces vývoje aplikací: objektově orientovaná analýza a návrh prakticky.* 2., aktualiz. a dopl. vyd. Brno: Computer Press, 2007. ISBN 978-80-251-1503-9.

[2] LEONARD, Anghel. *Spring Boot persistence best practices: optimize Java persistence performance in Spring Boot Applications.* [New York, NY]: Apress, [2020]. ISBN 978-1-4842-5625-1.

[3] VisualParadigm. *What is UML?* [online]. [cit. 2024-07-02]. Dostupné z: `https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-uml/`

[4] Amazon Web Services, Inc. *What is Java? - Java Programming Language Explained.* [online]. [cit. 2024-07-22]. Dostupné z: `https://aws.amazon.com/what-is/java/`

[5] Microsoft. *Co je Java Spring Boot?* [online]. [cit. 2024-07-22]. Dostupné z: `https://azure.microsoft.com/cs-cz/resources/cloud-computing-dictionary/what-is-java-spring-boot/`

[6] Oracle. *What is MySQL?* [online]. [cit. 2024-07-22]. Dostupné z: `https://www.oracle.com/mysql/what-is-mysql/`

[7] Future Learn. *What are HTML and CSS used for? The basics of coding for the web* [online]. 2022-06-03 [cit. 2024-07-22]. Dostupné z: `https://www.futurelearn.com/info/blog/what-are-html-css-basics-of-coding`

[8] GeeksforGeeks. *Introduction to JavaScript.* [online]. 2024-02-02 [cit. 2024-07-22]. Dostupné z: `https://www.geeksforgeeks.org/introduction-to-javascript/`

[9] TypeScript Tutorial. *What is TypeScript?* [online]. [cit. 2024-07-22]. Dostupné z: `https://www.typescripttutorial.net/typescript-tutorial/what-is-typescript/`

[10] GeeksforGeeks. *React Introduction.* [online]. 2024-03-11 [cit. 2024-07-23]. Dostupné z: `https://www.geeksforgeeks.org/reactjs-introduction/`

[11] Material UI. *Material UI - Overview.* [online]. [cit. 2024-07-23]. Dostupné z: https://mui.com/material-ui/getting-started/

[12] Highcharts. *Highcharts - Interactive Charting Library for Developers.* [online]. [cit. 2024-07-23]. Dostupné z: https://www.highcharts.com/

[13] Baeldung. *Testing in Spring Boot.* [online]. 2024-01-08 [cit. 2024-07-23]. Dostupné z: https://www.baeldung.com/spring-boot-testing

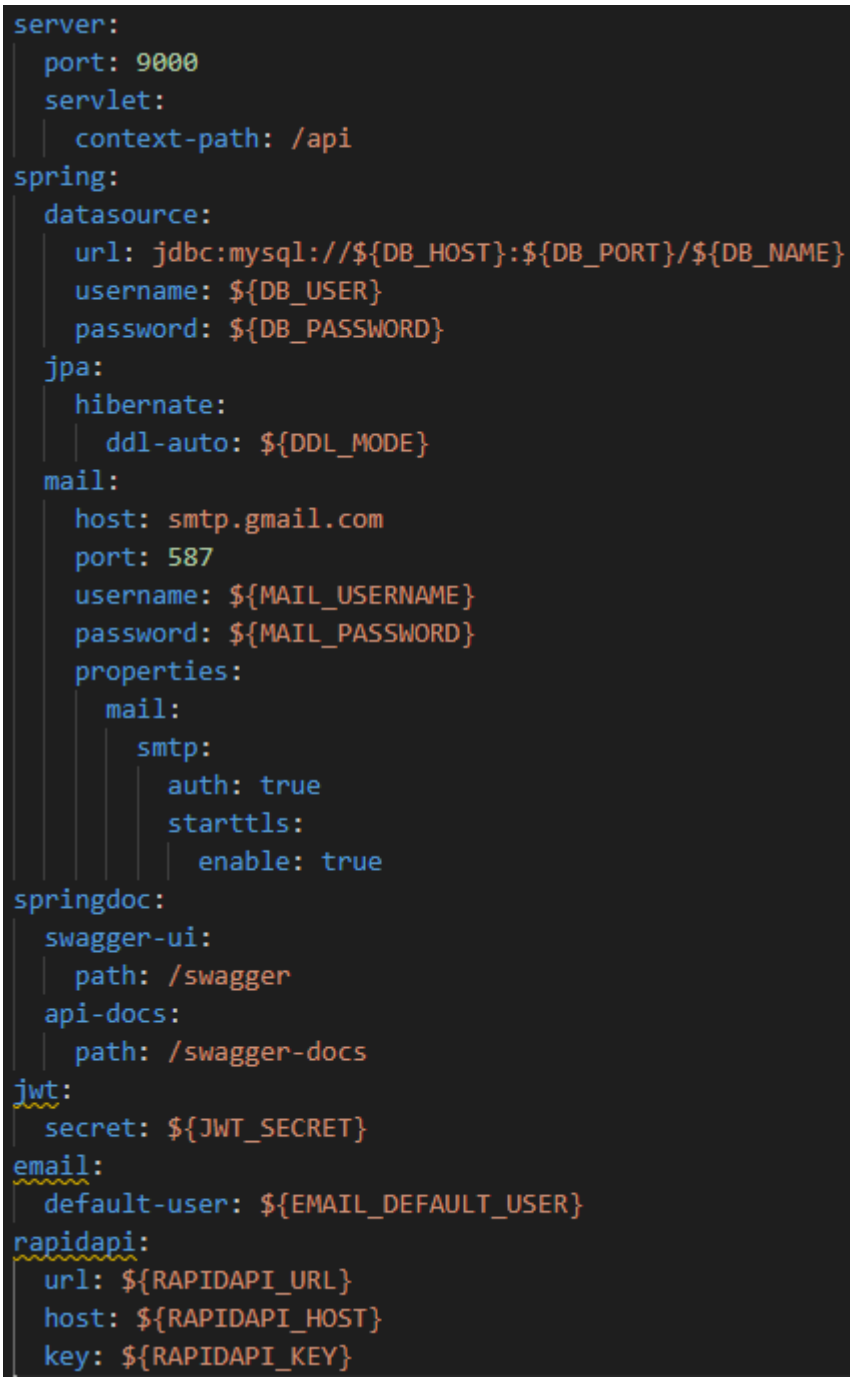
[14] Teric Cabrel Tchepannou. *Implement JWT Authentication in a Spring Boot 3 Application.* Medium [online]. 2024-03-19 [cit. 2024-07-23]. Dostupné z: https://medium.com/@tericcabrel/implement-jwt-authentication-in-a-spring-boot-3-application-5839e4fd8fac

[15] BezKoder. *Spring Boot Refresh Token with JWT example.* [online]. 2023-10-03 [cit. 2024-07-23]. Dostupné z: https://medium.com/@tericcabrel/implement-jwt-authentication-in-a-spring-boot-3-application-5839e4fd8fac

SEZNAM PŘÍLOH

KONFIGURAČNÍ SOUBOR SERVEROVÉ ČÁSTI APLIKACE

Příloha zobrazuje kompletní konfigurační soubor serverové části aplikace *application.yaml*.



Obrázek 39: Ukázka kompletního konfiguračního *application.yaml* (Zdroj: Autor)

NASTAVENÍ ENDPOINTŮ API (1. ČÁST)

První část výpisu nastavení jednotlivých endpointů API a jejich přístupových oprávnění.

```
.requestMatchers(getSwaggerRoutes()).permitAll()

.requestMatchers(HttpMethod.POST, "/login").permitAll()
.requestMatchers(HttpMethod.POST, "/register").permitAll()
.requestMatchers(HttpMethod.POST, "/refresh-token").permitAll()

.requestMatchers(HttpMethod.GET, "/user/**").hasAuthority("ADMIN")
.requestMatchers(HttpMethod.PUT, "/user").authenticated()
.requestMatchers(HttpMethod.POST, "/user/restore-password").permitAll()
.requestMatchers(HttpMethod.POST, "/user/new-password").permitAll()
.requestMatchers(HttpMethod.POST, "/user/change-password").authenticated()
.requestMatchers(HttpMethod.POST, "/user/change-status/**").hasAuthority("ADMIN")

.requestMatchers(HttpMethod.GET, "/venue/**").permitAll()
.requestMatchers(HttpMethod.POST, "/venue/**").hasAuthority("ADMIN")
.requestMatchers(HttpMethod.PUT, "/venue/**").hasAuthority("ADMIN")
.requestMatchers(HttpMethod.DELETE, "/venue/**").hasAuthority("ADMIN")

.requestMatchers(HttpMethod.GET, "/team/**").permitAll()
.requestMatchers(HttpMethod.POST, "/team/**").hasAuthority("ADMIN")
.requestMatchers(HttpMethod.PUT, "/team/**").hasAuthority("ADMIN")
.requestMatchers(HttpMethod.DELETE, "/team/**").hasAuthority("ADMIN")

.requestMatchers(HttpMethod.GET, "/statistics/**").authenticated()
.requestMatchers(HttpMethod.POST, "/statistics/**").hasAuthority("ADMIN")
.requestMatchers(HttpMethod.PUT, "/statistics/**").hasAuthority("ADMIN")
.requestMatchers(HttpMethod.DELETE, "/statistics/**").hasAuthority("ADMIN")

.requestMatchers(HttpMethod.GET, "/season/**").permitAll()
.requestMatchers(HttpMethod.POST, "/season/**").hasAuthority("ADMIN")
.requestMatchers(HttpMethod.PUT, "/season/**").hasAuthority("ADMIN")
.requestMatchers(HttpMethod.DELETE, "/season/**").hasAuthority("ADMIN")

.requestMatchers(HttpMethod.GET, "/season-team/**").permitAll()
.requestMatchers(HttpMethod.POST, "/season-team/**").hasAuthority("ADMIN")
.requestMatchers(HttpMethod.PUT, "/season-team/**").hasAuthority("ADMIN")
.requestMatchers(HttpMethod.DELETE, "/season-team/**").hasAuthority("ADMIN")
```

Obrázek 40: Ukázka nastavení endpointů API (1. část)

NASTAVENÍ ENDPOINTŮ API (2. ČÁST)

Druhá část výpisu nastavení jednotlivých endpointů API a jejich přístupových oprávnění.

```
.requestMatchers(HttpMethod.GET, "/season-round/**").permitAll()
.requestMatchers(HttpMethod.POST, "/season-round/**").hasAuthority("ADMIN")
.requestMatchers(HttpMethod.PUT, "/season-round/**").hasAuthority("ADMIN")
.requestMatchers(HttpMethod.DELETE, "/season-round/**").hasAuthority("ADMIN")

.requestMatchers(HttpMethod.GET, "/match/*/statistics").authenticated()
.requestMatchers(HttpMethod.GET, "/match/statistics/**").authenticated()
.requestMatchers(HttpMethod.GET, "/match/**").permitAll()
.requestMatchers(HttpMethod.POST, "/match/**").hasAuthority("ADMIN")
.requestMatchers(HttpMethod.PUT, "/match/**").hasAuthority("ADMIN")
.requestMatchers(HttpMethod.DELETE, "/match/**").hasAuthority("ADMIN")

.requestMatchers(HttpMethod.GET, "/league/**").permitAll()
.requestMatchers(HttpMethod.POST, "/league/**").hasAuthority("ADMIN")
.requestMatchers(HttpMethod.PUT, "/league/**").hasAuthority("ADMIN")
.requestMatchers(HttpMethod.DELETE, "/league/**").hasAuthority("ADMIN")

.requestMatchers(HttpMethod.GET, "/country/**").permitAll()
.requestMatchers(HttpMethod.POST, "/country/**").hasAuthority("ADMIN")
.requestMatchers(HttpMethod.PUT, "/country/**").hasAuthority("ADMIN")
.requestMatchers(HttpMethod.DELETE, "/country/**").hasAuthority("ADMIN")

.requestMatchers(HttpMethod.GET, "/notification/**").authenticated()
.requestMatchers(HttpMethod.POST, "/notification/**").authenticated()
.requestMatchers(HttpMethod.PUT, "/notification/**").authenticated()
.requestMatchers(HttpMethod.DELETE, "/notification/**").authenticated()

.requestMatchers(HttpMethod.GET, "/comment/**").authenticated()
.requestMatchers(HttpMethod.POST, "/comment/**").authenticated()
.requestMatchers(HttpMethod.PUT, "/comment/**").authenticated()
.requestMatchers(HttpMethod.DELETE, "/comment/**").authenticated()

.anyRequest().authenticated());
```

Obrázek 41: Ukázka nastavení endpointů API (2. část)