FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

# Zadání bakalářské práce

| | |
|---|---|
| **Název:** | Informační systém pro házenkářské sportovní kluby |
| **Student:** | Stela Augustínová |
| **Vedoucí:** | Ing. Filip Glazar |
| **Studijní program:** | Informatika |
| **Obor / specializace:** | Webové a softwarové inženýrství, zaměření Webové inženýrství |
| **Katedra:** | Katedra softwarového inženýrství |
| **Platnost zadání:** | do konce letního semestru 2023/2024 |

## Pokyny pro vypracování

Cílem práce je navrhnout a implementovat webovou aplikaci, která bude sloužit jako informační systém pro házenkářské kluby. Mezi hlavní funkční požadavky systému patří evidence zápasu a tréninků. Dále je možné evidovat pokročilé statistiky o hráčích i samotných zápasech. Postupujte ideálně dle následujících kroků.

1) Proveďte sběr požadavků na vlastnosti a funkční požadavky systému
2) Specifikujte funkční požadavky systému
3) Zvolte vhodnou architekturu aplikace a s ní související aplikace
4) Navrhněte a implementujte prototyp aplikace
5) Podrobte aplikaci vámi vybranou formou testu
6) Na základě výsledků testování navrhněte potřebné úpravy aplikace a proveďte zhodnocení výsledků

---

*Elektronicky schválil/a Ing. Jaroslav Kuchař, Ph.D. dne 9. února 2023 v Praze.*

Bakalárska práca

# INFORMAČNÍ SYSTÉM PRO HÁZENKÁŘSKÉ SPORTOVNÍ KLUBY

**Stela Augustínová**

Fakulta informačních technologií
Katedra softwarového inženýrství
Vedúci: Ing. Filip Glazar
11. mája 2023

Odkaz na túto prácu: Augustínová Stela. *Informační systém pro házenkářské sportovní kluby.* Bakalárska práca. České vysoké učení technické v Praze, Fakulta informačních technologií, 2021.

# Obsah

# Zoznam obrázkov

# Zoznam tabuliek



# Zoznam výpisov kódu

*Chcela by som poďakovať vedúcemu práce Ing. Filipovi Glazarovi za ochotu, cenné informácie a priateľský prístup pri vedení tejto práce. Taktiež ďakujem môjmu priateľovi za podporu a inšpiráciu pri tvorbe práce. Veľké poďakovanie patrí aj mojej rodine za povzbudenie a oporu počas celého štúdia.*

# Vyhlásenie

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací. Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů, zejména skutečnost, že České vysoké učení technické v Praze má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 citovaného zákona.

V Praze dňa 11. mája 2023 .....

# Abstrakt

Táto bakalárska práca popisuje návrh a implementáciu webovej aplikácie, ktorá slúži ako informačný systém pre hádzanárske kluby, ktorý zjednodušuje organizáciu zápasov a tréningov. Táto aplikácika taktiež poskytuje štatistiky, či už pre tímy alebo pre hráčov. Prvá časť sa zaoberá definovaním funkcionalít systému, pričom sa porovná aj s existujúcimi riešeniami. Nasleduje návrh samotnej webovej aplikácie a jej architektúry klient-server. Táto časť definuje aj technológie využité pre implementáciu zvolenej architektúry. Na návrh nadväzuje implementácia aplikácie. Tá využiva technológie z predchádzajúcej časti. Server je implementovaný v jazyku Java s využitím Spring Frameworku, zatiaľ čo klient je implementovaný pomocou TypeScriptu a Angularu. Na záver je popísané užívateľské testovanie a jeho výsledky, na čo nadväzujú budúce rozšírenia aplikácie.

**Klíčová slova** webová aplikácia, informačný systém, hádzaná, TypeScript, Angular, Java, Spring Framework

# Abstract

This bachelor thesis describes the design and implementation of a web application intended to function as information system for handball clubs, which simplifies organization of matches and trainings. This application also provides statistics, both for teams and individual players. The first part deals with defining the functionalities of the system, while also comparing it to existing solutions. This is followed by the design of the web application and its client-server architecture. This section also defines the technologies used to implement selected architecture. The design part is followed by the implementation of the application, utilizing the technologies from the previous section. The server is implemented in Java using the Spring Framework, while the client is implemented using TypeScript and Angular. In the end, user testing and its results are described, which is followed by future extensions of the application.

**Keywords** web application, information system, handball, TypeScript, Angular, Java, Spring Framework

# Seznam zkratek

| | |
|---|---|
| AJAX | Asynchronous JavaScript and XML |
| API | Application programming interface |
| CLI | Command-line interface |
| CORS | Cross-Origin Resource Sharing |
| CRUD | Create Read Update Delete |
| CSS | Cascading Style Sheets |
| DI | Dependency Injection |
| DOM | Document Object Model |
| DTO | Data Transfer Object |
| HTML | HyperText Markup Language |
| HTTP | Hypertext Transfer Protocol |
| JPA | Java Persistence API |
| JSON | JavaScript Object Notation |
| JVM | Java Virtual Machine |
| MVC | Model-View-Controller |
| ORM | Object-relational Mapping |
| REST | Representational state transfer |
| SMTP | Simple Mail Transfer Protocol |
| SOAP | Simple Object Access Protocol |
| SPA | Single-page application |
| SQL | Structured Query Language |
| TCP | Transmission Control Protocol |
| UI | User Interface |
| URI | Uniform Resource Identifier |
| URL | Uniform Resource Locator |
| XML | Extensible Markup Language |

# Kapitola 1
# Úvod

V posledných rokoch som mala možnosť nahliadnuť do fungovania väčšieho hádzanárskeho klubu, ktorý sa zúčastňuje aj súťaží na európskej úrovni. Organizácia mi niekedy pripadala mierne chaotická, keďže hráči dostávali informácie o tréningoch a zápasoch prostredníctvom troch rôznych aplikácií. Preto som sa rozhodla všetko spojiť a vytvoriť aplikáciu pre zjednodušenie organizácie hádzanárskych klubov.

Cieľom tejto bakalárskej práce je navrhnnutie a implementácia webovej aplikácie, ktorá bude slúžiť ako informačný systém pre hádzanárske kluby. Hlavnou úlohou tohto systému je uľahčenie organizácie tréningov a zápasov, čo je dôležité práve pri väčších kluboch, ktoré sa zúčastňujú viacerých súťaží. Okrem toho systém ponúka aj štatistiky vytvorené počas zápasov pre jednotlivých hráčov ale aj spoločne pre tím.

Prvým cieľom pri tvorbe tejto práce je analýza funkčných a nefunkčných požiadaviek, a taktiež rolí pre presnú predstavu ako má systém fungovať. Je potrebná aj analýza už existujúcich riešení. Touto analýzou sa zaoberá kapitola 2.

Druhým cieľom je návrh samotnej aplikácie. Je potrebné určiť typ aplikácie, jej architektúru a taktiež technológie, pomocou ktorých bude aplikácia vytváraná. Informačný systém bude webová aplikácia napísaná v jazykoch Java a TypeScript s použitím vhodných frameworkov. Celý návrh aplikácie je popísaný v kapitole 3.

Posledným cieľom je implementácia systému aplikovaním vybraných technológií. Podrobný popis sa nachádza v kapitole 4. Okrem toho je potrebné aplikáciu aj vhodne otestovať pre zaistenie správnej funkčnosti.

# Kapitola 2
# Analýza

*Táto kapitola sa zaoberá analýzou aplikácie. Najprv sa uvedú všetky požiadavky na danú aplikáciu a typy užívateľov, ktorí túto aplikáciu budú využívať. V závere kapitoly je vykonaná analýza existujúcich riešení podobných systémov.*

## 2.1 Funkčné požiadavky

Funkčné požiadavky sú špecifikáciou o tom, ako sa systém bude správať. Definujú celú funkcionalitu systému pre splnenie potrieb používateľov. Môžu byť vnímané aj ako funkcie, ktoré sú zaznamenávané používateľom. [1]

**F1 evidencia a správa užívateľov**

1. Užívateľ sa môže zaregistrovať. Pri registrácii je potrebné uviesť:
   - užívateľské meno
   - e-mail
   - heslo
2. Užívateľ sa môže prihlásiť zadaním užívateľského mena a hesla
3. Registrovaný užívateľ môže byť pridaný do tímu
4. Užívateľ môže byť odstránený z tímu
5. Prihlásený užívateľ môže upravovať svoje prihlasovacie údaje
6. Prihlásený užívateľ môže upravovať údaje vo svojom profile

**F2 evidencia a správa tréningov**

1. Prihlásený užívateľ má možnosť prezerať detaily tréningov.
2. Prihlásený tréner má možnosť vytvárať tréningy. Pri vytváraní tréningu sa uvádza:
   - dátum
   - čas
   - miesto
   - poznámka (voliteľné)
3. Prihlásený tréner má možnosť upravovať tréningy
4. Prihlásený tréner má možnosť vymazať tréningy.

**F3 evidencia dochádzky**

1. Prihlásený tréner alebo manažér má možnosť pri existujúcom tréningu zadávať dochádzku

**F4 evidencia a správa zápasov**

1. Prihlásený užívateľ má možnosť prezerať detaily zápasov.
2. Prihlásený manažér má možnosť vytvárať zápasy. Pri vytváraní zápasu je potrebné zadať:
   - dátum
   - čas
   - miesto
   - súťaž
   - domáci tím
   - hosťujúci tím
   - poznámka (voliteľné)
3. Prihlásený tréner alebo manažér má možnosť upravovať zápasy.
4. Po ukončení zápasu prihlásený manažér má možnosť zadať:

- skóre domáci
- skóre hostia

5. Prihlásený tréner má možnosť priradiť k zápasu súpisku.
6. Prihlásený manažér má možnosť vymazať zápasy.

**F5 evidencia a správa súpisky**

1. Prihlásený tréner má možnosť vytvoriť súpisku. Pri vytváraní súpisky je potrebné zadať:
   - názov súpisky
   - hráčov (voliteľné)
2. Prihlásený tréner má možnosť upravovať súpisku.
3. Prihlásený tréner má možnosť vymazať súpisku.

**F6 správa štatistík**

1. Prihlásený manažér má možnosť pridať vytvorenú štatistiku k zápasu.
2. Prihlásený užívateľ má možnosť prezerať štatistiky tímu k zápasu.
3. Prihlásený užívateľ ma možnosť prezerať štatistiku daného hráča.
4. Prihlásený manažér má možnosť vymazať štatistiku.

**F7 evidencia a správa športovísk**

1. Prihlásený manažér má možnosť vytvoriť športovisko pre svoj tím. Pri vytváraní športoviska je potrebné určiť:
   - názov športoviska
   - adresu
   - či je možné v ňom odohrať zápas
2. Prihlásený manažéer má možnosť upravovať športoviská.
3. Prihlásený manažér má možnosť odstrániť športoviská.

**F8 prezeranie udalostí**

1. Prihlásený užívateľ má možnosť prezerať udalosti v zozname udalostí.

2. Prihlásený užívateľ má možnosť filtrovať udalosti v zozname.
3. Prihlásený úžívateľ má možnosť zobraziť udalosti v kalendári.

**F9 prezeranie udalostí**

1. Prihlásený užívateľ má možnosť prezerať tímy.
2. Prihlásený manažér má možnosť upravovať svoj tím.

## 2.2 Nefunkčné požiadavky

**N1 Webová aplikácia**

Aplikácia je navrhnutá ako webová aplikácia a dostupná v prehliadači.

**N2 REST API**

Klientská a serverová časť budú komunikovať pomocou REST API.

**N3 Klient**

Klientská časť bude implementovaná ako SPA pomocou Angularu.

**N4 Server**

Serverová časť bude implementovaná pomocou Spring Framework.

## 2.3 Role

**R1 hráč**

- základná rola
- možnosť spravovať účet
- prezeranie tímových udalostí

**R2 tréner**

- možnosť spravovať účet
- prezeranie tímových udalostí
- pridávanie a správa tréningov
- kontrolovanie dochádzky na tréningoch
- vytváranie súpisok a ich pridávanie na zápasy

**R3 manažér**

- možnosť spravovať účet
- prezeranie tímových udalostí
- pridávanie a správa športovísk
- pridávanie a správa zápasov
- kontrolovanie dochádzky na tréningoch
- pridávanie štatistiky

– správa tímu

– pridávanie hráčov do tímu

**R3 administrátor**

– priradenie manažéra k tímu

– vytvorenie neového tímu pre manažéra

– priradenie trénera k tímu

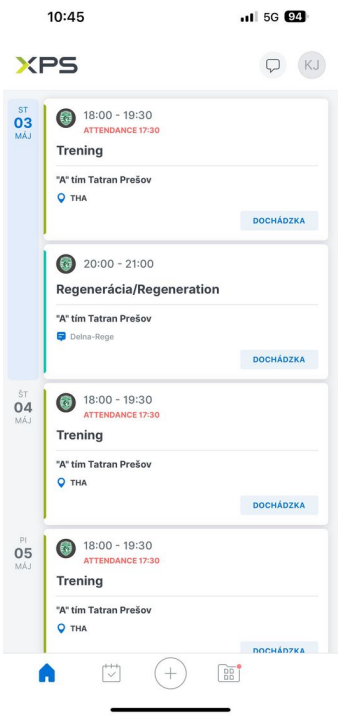
– vytvorenie novej súťaže

– prihlásenie tímu do súťaže

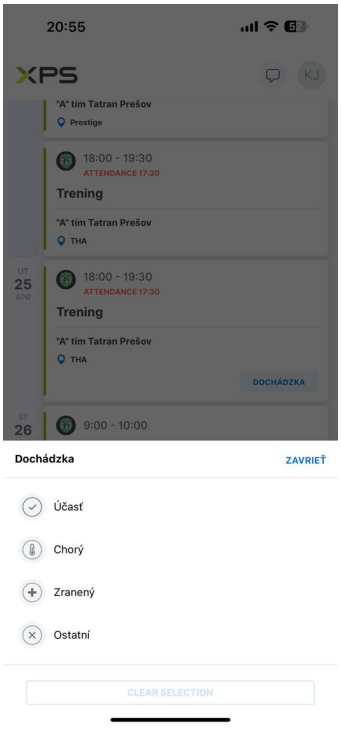
## 2.4 Existujúce riešenia

### 2.4.1 XPS Network

XPS Network poskytuje webovú aj mobilnú aplikáciu pre športové kluby. Ponúka pokročilé plánovanie tréningov, zápasov a taktiež ich analýzy. Tréneri v ňom môžu vytvárať situácie zo zápasov a tréningov pomocou animácií a diagramov v Playbooku. Tréneri a hráči v ňom môžu komunikovať, či už prostredníctvom skupinových alebo individuálnych správ. Hráči majú možnosť nahlásiť zranenia a zdravotný tím im môže pripraviť plán rehabilitácií a liečby. Nevýhodou tejto aplikácie je to, že je platená. Cena tejto aplikácie sa odvíja od zakúpených funkcionalít. Tiež si myslím, že hráči by nemali mať možnosť zadávať vlastnú dochádzku.




**Obr. 2.1** XPS Network




**Obr. 2.2** XPS Network dochádzka

### 2.4.2 Týmuj

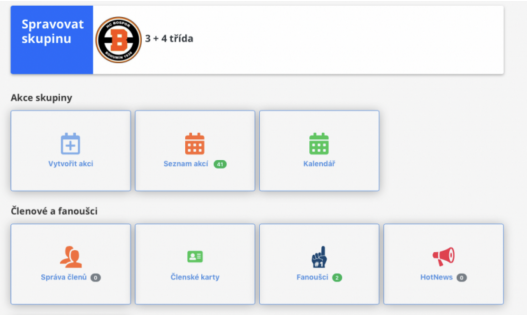
Týmuj rovnako ako XPS Network poskytuje aj webovú aj mobilnú aplikáciu. V každom tíme môžu mať členovia 3 role a to Majiteľ, Správca a Hráč. Podľa môjho názoru je rola Tréner podstatná pre profesionálny klub a tá tam chýba. Taktiež poskytuje tvorbu a organizáciu udalostí s dochádzkou, ktorá môže byť opäť upravovaná konkrétnym užívateľom ale len na webe. V neplatenej verzii je limitovaný počet účastníkov. Tento systém je podľa mňa využiteľný skôr pre amatérske tímy a pre profesionálne by nebol dostačujúci.


**Obr. 2.3** Týmuj


**Obr. 2.4** Týmuj udalosti

### 2.4.3 Sportnect

Sportnect je športová sociálna sieť spájajúca športovcov, trénerov, fanúšikov a rodičov športujúcich detí. Venuje sa všetkým úrovniam športu od rekreačných športovcov, cez detské športy až po profesionálov. Pre profesionálov sú využiteľné funkcie ako tréningové plány, nominácie na zápasy a komunikácia medzi trénermi a hráčmi. Túto webovú aplikáciu využívajú aj niektoré české hádzanárske kluby ako HK Lovci Lovosice a Akademie Dukla Praha. Chýbajú ale nejaké štatistiky ohľadom zápasov a hráčov, ktoré môžu byť dôležitou pomôckou pre trénerov. Výhodou tohto systému je, že je bezplatný. V prípade potreby je možné si zaplatiť premium verziu, ktorá za ďalšie poplatky sprístupní aj vlastnú mobilnú aplikáciu a web.




**Obr. 2.5** Sportnect


**Obr. 2.6** Sportnect udalosti

# Kapitola 3
# Návrh

*Táto kapitola sa zaoberá celkovým návrhom informačného systému. Popisuje návrh zvolenej architektúry pre implementáciu systému. Zároveň vyobrazuje návrh databázy s charakteristikou jednotlivých tabuliek. V poslednej sekcii sú uvedené technológie využité pre implementáciu zvolenej architektúry.*

Aplikácie, ktoré sú dostupné cez prehliadač, pričom sú uložené na serveri, sa nazývajú webové aplikácie. Príkladmi takých aplikácií sú e-shopy alebo e-mailové služby. Väčšina webových aplikácií je dostupná bez ohľadu na to, aký prehliadač práve používate. [2] Keďže je jedna časť uložená na vzdialenom serveri a druhá klientská časť je prístupná cez prehliadač, ide o architektúru klient-server.

Okrem webových aplikácií existujú ešte natívne a hybridné aplikácie. Natívne aplikácie sú vyvíjané pre konkrétne zariadenie alebo platformu, a pre ich použitie je nutná inštalácia na dané zariadenie. Pre niektoré natívne aplikácie je výhodou, že môžu fungovať aj offline. Hybridné aplikácie fungujú podobne ako tie webové, a zároveň je potrebná ich inštalácia na zariadenie. Výhodou hybridných aplikácií je možnosť využívať prostriedky špecifické pre zariadenie využitím interných API. [2]

Medzi najväčšie výhody webových aplikácií patrí schopnosť bežať na rozličných platformách bez ohľadu na práve využívané zariadenie alebo operačný systém, a prístup k rovnakej verzii pre každého používateľa. Informačný systém som sa rozhodla implementovať ako webovú aplikáciu práve kvôli týmto výhodám. Okrem toho tieto aplikácie nezaberajú žiadne miesto, keďže nie je potrebné ich inštalovať na pevný disk. [3]

## 3.1 Architektúra klient-server

Architektúra aplikácie popisuje zaužívané techniky a vzorce pri návrhu aplikácie, a poskytuje osvedčené postupy pre získanie dobre štruktúrovanej aplikácie. [4]

Klient-server je architektúra pozostávajúca z dvoch častí, a to klient a server, ktoré si medzi sebou vymieňajú rôzne dáta a vykonávajú funkcie, čím umožňujú interprocesovú komunikáciu. Klient posiela žiadosti, na ktoré následne server odpovedá. Medzi výhody tohto modelu patrí rozdelenie záťaže a jednoduché zdieľanie zdrojov medzi klientmi a servermi. Klient-server sa rozdeľuje na dve hlavné architektúry 2-vrstvové a 3-vrstvové. [5]

2-vrstvová architektúra sa ďalej rozdeľuje na thin-client / smart-server a thick-client / dumb-server. Tradičným návrhom je thin-client, kde je logika na strane servera. Naopak, pri thick-client ide o moderný prístup, v ktorom je logika na strane klienta a server je API.

Pri 3-vrstvovej architektúre je celá aplikácia rozdelená do prezentačnej, aplikačnej a dátovej

vrstvy. Prezentačná vrstva je užívateľské rozhranie, v aplikačnej sa spracovávajú dáta a do dátovej vrstvy sa tieto dáta ukladajú. Hlavnou výhodou trojvrstvovej architektúry je oddelený vývoj každej vrstvy, pričom sa ostatné vrstvy nijak neovplyvňujú. [6] Takou architektúrou je napríklad MVC, ktorú vzhľadom na využité technológie aplikujem.

Klient a server spolu komunikujú pomocou API. API (application programming interface) definuje pravidlá, umožňujúce vzájomnú komunikáciu medzi rôznymi aplikáciami. Predstavuje sprostredkovateľa, ktorý zabezpečuje prenos dát medzi klientom a serverom. [7]

Najznámejšími prístupmi, ktoré definujú ako vytvoriť API sú REST a SOAP. Najväčším rozdielom medzi nimi je, že SOAP je protokol, zatiaľ čo REST je sada architektonických princípov. Pri používaní SOAP môže byť požiadavka spracovaná cez akýkoľvek protokol, ako napríklad HTTP, TCP, SMTP a iné, pričom vrátené dáta sú vo formáte XML. Naopak pri REST je najbežnejším formátom JSON, ale môže sa použiť aj HTML alebo XML. Požiadavky sa zvyčajne posielajú cez HTTP protokol. [8] Pre sprostredkovanie komunikácie medzi klientom a serverom v mojej aplikácii som sa rozhodla pre REST.

### 3.1.1 HTTP protokol

HTTP je klient-serverový protokol pre získavanie zdrojov. Komunikácia medzi klientom a serverom prebieha prostredníctvom výmeny správ, ktoré sa u klienta nazývajú požiadavky, a pre server sú to odpovede. [9]

Klient odošle požiadavku na server, ktorý ho následne spracuje a odošle odpoveď. Komunikácia klienta a servera je vždy iniciovaná klientom, nikdy nie naopak. Tieto správy sú ľudsky čitateľné, pričom musia mať svoj presný formát. [9]

#### 3.1.1.1 HTTP požiadavky

Požiadavky sa skladajú z HTTP metódy, URL, verzie HTTP protokolu, voliteľných hlavičiek a tela. [9]

HTTP definuje niekoľko HTTP metód, ktoré určujú aká akcia sa má pre daný zdroj vykonať. Tieto metódy sa tiež označujú ako HTTP slovesá, aj keď niekedy sú podstatnými menami. Pri týchto metódach sa určujú aj základné vlastnosti ako bezpečnosť, cacheovateľnosť alebo idempotencia. Idempotencia znamená, že vykonanie jednej požiadavky a vykonanie niekoľko rovnakých požiadaviek má stále rovnaký efekt. [10, 11]

| Metóda | Akcia | Vlastnosti |
|---|---|---|
| GET | získanie zdroja | bezpečná, idempotentná, cacheovateľná |
| POST | odoslanie entity na určitý zdroj | cacheovateľná |
| PUT | nahradenie zdroja | idempotentná |
| PATCH | čiastočná úprava zdroja | — |
| DELETE | odstránenie zdroja | idempotentná |

**Tabuľka 3.1** Tabuľka najpoužívanejších HTTP metód.

Okrem najpoužívanejších metód, ktoré sú zobrazené v tabuľke3.1, existujú ešte metódy HEAD, CONNECT, OPTIONS a TRACE. [10]

URL reprezentuje cestu k získavanému zdroju. Skladá sa z viacerých častí, z ktorých nie všetky sú povinné. Najdôležitejšie časti URL sú [12]:

- **Schéma** je prvou časťou URL. Označuje protokol, ktorý je nutné použiť v požiadavke.
- **Autorita** je oddelená od schémy pomocou „://“ . Ak sú prítomné, zahŕňa aj doménu aj port, ktoré oddeľuje dvojbodka. Doména je webový server, väčšinou sa udáva jej názov, ale

je možné namiesto neho uviesť IP adresu. Port je povinný len v prípade, že sa nevyužívajú štandartné porty protokolu HTTP.

- **Cesta k zdroju** na webovom serveri je abstrakciou, ktorú server spracováva. V minulosti predstavovala umiestnenie súboru na serveri.
- **Parametre** sú zoznamom kľúčov a ich hodnôt, ktoré sú navzájom oddelené symbolom &. Server použije tieto parametre pre ďalšie spracovanie dát.
- **Kotva** označuje konkrétnu časť zdroja, čím dáva inštrukcie pre zobrazenie obsahu na presne danej pozícii.


**Obr. 3.1** Časti URL

Verzia HTTP protokolu udáva aktuálne používanú verziu. Tie sa počas rokov vyvíjali následovne [13]:

- **HTTP/0.9** je úplne prvou verziou HTTP. Požiadavky mali len jeden riadok a podporovaná bola iba metóda GET. Odpoveďou bol len samotný súbor.
- **HTTP/1.0** do požiadavky bola pridaná verzia protokolu. Na začiatku odpovede sa posielal stavový kód, a boli pridané hlavičky pre požiadavky aj odpovede.
- **HTTP/1.1** priniesol niekoľko vylepšení ako napríklad opakované využitie spojenia, podpora posielania odpovede po častiach, zavedenie mechanizmov pre kontrolu cache, pridanie hlavičky Host pre umožnenie zdieľania servera, a iné.
- **HTTP/2.0** je binárny protokol, takže nie je ľudsky čitateľný ani sa nedá manuálne vytvoriť, napriek tomu ale vylepšuje optimalizačné techniky. Navyše je multiplexovaný, čo znamená, že požiadavky môžu byt zaslané paralelne. Okrem toho umožňuje aj kompresiu hlavičiek, čo znižuje objem prenášaných dát.

Ďalšou súčasťou HTTP požiadavku sú hlavičky, ktoré umožňujú klientovi a serveru prenos dodatočných informácií. Môžu to byť napríklad hlavičky, ktoré obsahujú informácie o zdrojoch, o tele odpovede alebo aj dĺžke obsahu, jazyku a kódovania. [14]

Poslednou časťou je telo požiadavky. To sa používa pri metódach ako POST a PUT, ktoré vytvárajú alebo upravujú zdroje. Telo HTTP požiadavky obsahuje daný zdroj, ktorý posiela. [9]

### 3.1.1.2 HTTP odpovede

HTTP odpovede sú zložené z verzie HTTP protokolu, stavového kódu a správy, HTTP hlavičiek a môže obsahovať aj telo získaného zdroja. [9] Verzia HTTP protokolu a HTTP hlavičky sú popísané v sekcii 3.1.1.1.

Pri HTTP odpovediach stavové kódy a ich správy udávajú, či boli požiadavky úspešne vykonané. Stavové kódy rozdeľujeme do 5 skupín. [15]

| Kód | Význam |
|---|---|
| 1xx | informačné |
| 2xx | úspech |
| 3xx | presmerovanie |
| 4xx | chyba na strane klienta |
| 5xx | chyba na strane servera |

**Tabuľka 3.2** Tabuľka stavových kódov. [15]

Každému stavovému kódu prináleží správa, ktorá poskytuje ďalšie informácie o vykonaní požiadavku. [9] V tabuľke 3.3 sú uvedené niektoré stavové správy patriace k daným kódom.

| Kód | Správa |
|---|---|
| 100 | Continue |
| 200 | OK |
| 201 | Created |
| 302 | Found |
| 400 | Bad Request |
| 401 | Unauthorized |
| 404 | Not Found |
| 500 | Internal Server Error |
| 502 | Bad Gateway |

**Tabuľka 3.3** Stavové kódy a správy. [15]

Voliteľnou súčasťou HTTP odpovede je ako aj pri požiadavkách, telo. V ňom server posiela údaje o získanom zdroji. [9]

### 3.1.2 REST API

REST je architektúra, kde sú jednotlivé zdroje reprezentované pomocou URL alebo URI, a najčastejšie využíva HTTP protokol (sekcia 3.1.1). V REST API je potrebné dodržiavať nasledujúce základné pravidlá.

**Jednotné rozhranie** definuje komunikáciu medzi klientom a serverom, a zároveň pomáha rozdeliť architektúru. Klient a server môžu byť vyvíjané nezávisle, kým sa riadia daným rozhraním. [16]

Obmedzenie **klient-server** pomáha vybudovať ľahko škálovateľnú a jednoduchšiu architektúru. [16]

**Bezstavovosť** je kľúčovým pravidlom RESTu, ktoré stanovuje, že server si nepamätá stav, v ktorom sa klient nachádza. Všetky informácie klient zasiela v požiadavkách na server, napríklad ako parameter dotazu, parameter hlavičky, v tele, alebo ako súčasť URI. [16]

**Cacheovateľnosť** prispieva k zlepšeniu škálovateľnosti a celkovému výkonu servera. Môže sa nachádzať na serveri alebo aj v klientskej aplikácii. Správne riadené cacheovanie môže odstrániť niektoré interakcie medzi klientom a serverom, čo ďalej zvýši výkon a škálovateľnosť. [16]

**Vrstvený systém** znamená, že volania a odpovede môžu prechádzať niekoľkými rôznymi sprostredkovateľmi. Rozhrania majú byť navrhnuté tak, aby klient ani server nevedeli určiť, či komunikujú so sprostredkovateľom alebo s koncovou aplikáciou. [16]

**Code on demand** (voliteľné) umožňuje prenášanie spustiteľného kódu zo servera na požiadanie klientovi. [16]

Aplikácie komunikujú prostredníctvom HTTP požiadavky, ktoré obsahujú URL adresu API, ale taktiež hlavičky a parametre, zahŕňajúce dôležité údaje o identifikátoroch, ako sú URI, metadáta, autorizácie, cookies a iné (sekcia 3.1.1.1). CRUD (create, read, update, delete) operácie

sa v RESTe vykonávajú pomocou HTTP metód, ktoré su súčasťou požiadavku. Operácie, ktoré prináležia daným metódam sú popísané v tabuľke 3.4

| Operácia | Metóda |
|---|---|
| Create | POST |
| Read | GET |
| Update | PUT, prípadne PATCH |
| Delete | DELETE |

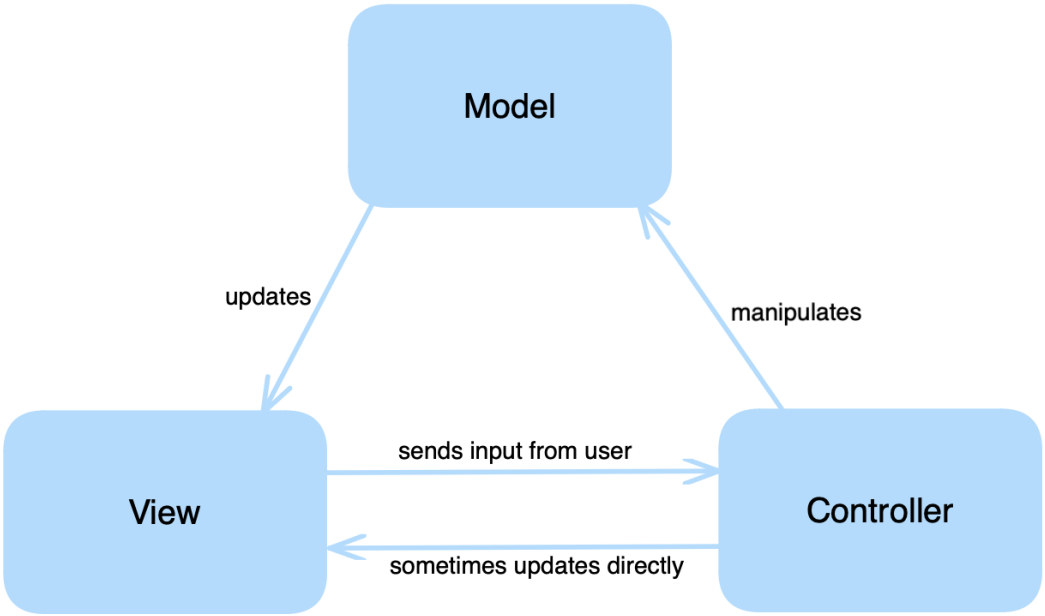
**Tabuľka 3.4** CRUD operácie pomocou HTTP metód. [16]

Server vracia informácie v podobe HTTP odpovedí (popísané v sekcii 3.1.1.2), ktorými informuje, či dané volania boli úspešné. Tieto informácie sú najčastejšie vo formáte JSON. [16]

### 3.1.3 MVC

MVC (Model-View-Controller) sa používa pre implementáciu užívateľských rozhraní, dát a logiky. Základnou myšlienkou tohto vzoru je oddelenie logiky od výstupu. MVC rozdeľuje aplikáciu na 3 hlavné komponenty[17]:

1. Model
2. View
3. Controller




**Obr. 3.2** MVC Architektúra

**Model** definuje dáta a obsahuje logiku aplikácie, ako napríklad výpočty, validácie a iné funkcie. Jeho úlohou je prijímanie vstupných parametrov, a na ich základe vydá dáta na výstup. V prípade využívania ORM (Objektovo-Relačné mapovanie) sa jednotlivé modely zhodujú so štruktúrou tabuliek v databáze. [18]

**View** (Pohľad) zobrazuje užívateľovi výstup, s ktorým následne interaguje. Najčastejšie ide o šablónu, ktorá obsahuje značkovací jazyk HTML, vďaka ktorému je do nej možné vkladať dáta z modelu. Kedže hlavnou funkciou je zobrazovanie výstupu, View obsahuje len logiku potrebnú pre vyobrazenie dát. [19]

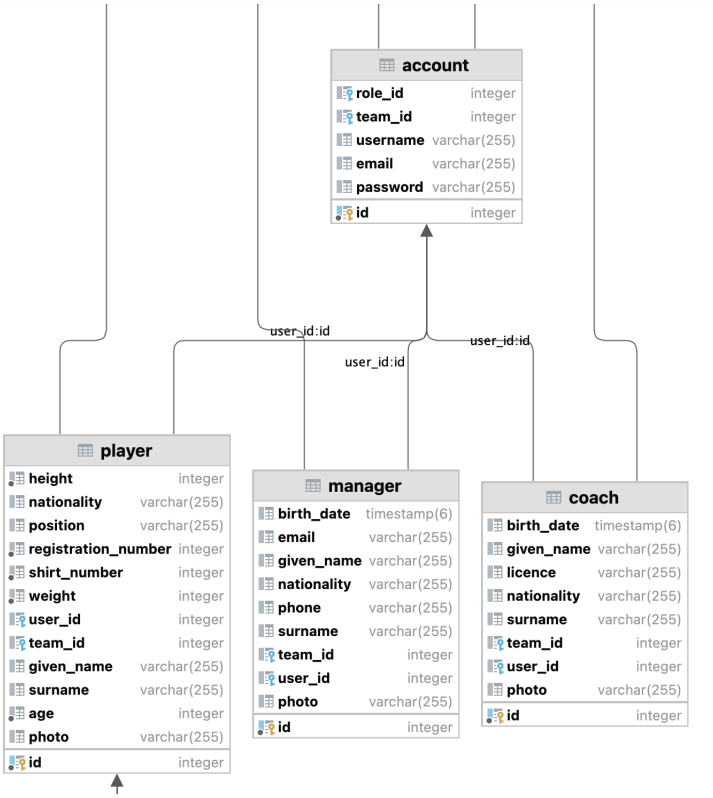
**Controller** funguje ako sprostredkovateľ medzi modelom, view a užívateľom. Prijíma údaje z modelu, následne ich spracuje, a všetky odošle do View. Najčastejšie každej entite prináleží jeden controller. [19]

## 3.2 Databázový Model

Kvôli dôležitým väzbám medzi jednotlivými tabuľkami a presnú štruktúru dát som sa rozhodla zvoliť relačnú databázu. V tejto sekcii je popísaný návrh databázy, zobrazený na obrázku A.1.

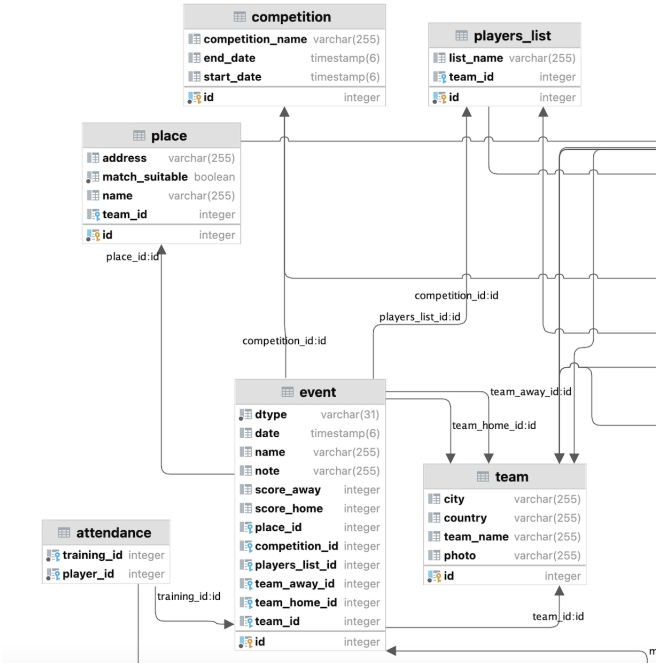
**Relačnú databázu** tvoria relácie, ktoré predstavujú jednotlivé tabuľky. Tieto tabuľky sú zložené z atribútov (stĺpce) a záznamov (riadky). [20]

V databáze sa nachádza celkovo 15 tabuliek. Pre užívateľov sa využíva 5 tabuliek. Tabuľka **Role** obsahuje možné role užívateľov. V tabuľke **Account** sú evidované prihlasovacie údaje užívateľov. Na tabuľku account sú naviazané tabuľky predstavujúce profily v závislosti od role. Ide o tabuľky **Player**, **Coach** a **Manager**.




**Obr. 3.3** Tabuľky definujúce uživateľov

Ďalej sa v databáze nachádza tabuľka **Event**, ktorá reprezentuje jednotlivé udalosti typu Tréning a Zápas. Táto tabuľka je naviazaná na tabuľku **Team**, aby bolo jednoznačné, pre ktorý tím/tímy je udalosť vytvorená. Ďalej je naviazaná na tabuľku **Place** pre určenie miesta udalosti. V prípade, že ide o zápas aj určená aj **Competition** a **PlayersList**. Naopak, v prípade tréningu ide o tabuľku **Attendance**, vďaka ktorej je zaznamenaná dochádzka na tréningy.



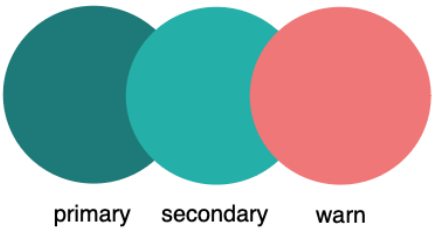

**Obr. 3.4** Tabuľka pre udalosti

Okrem toho sa v databáze nachádzajú ešte tabuľky **Standings** pre zaznamenávanie poradia tímov v súťažiach, a pre zaznamenávanie štatistík **StatMatch** a **StatPlayer**. Celá schéma databázy sa nachádza v prílohe A.

## 3.3 Návrh užívateľského rozhrania

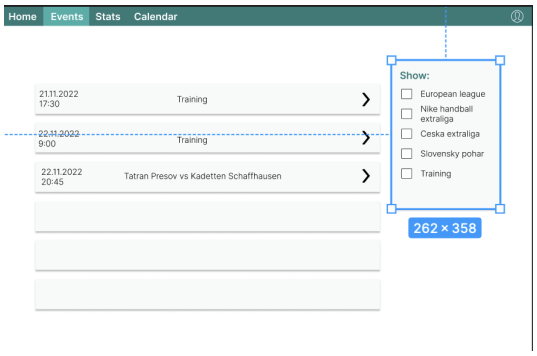
UI alebo užívateľské rozhranie je bodom komunikácie a interakcie medzi človekom a zariadením. Taktiež je spôsobom, ktorým užívateľ interaguje s aplikáciou alebo webovou stránkou. UI sa často spomína v súvislosti s UX, ktorý zahŕňa estetický vzhľad zariadenia, čas odozvy a obsah, ktorý sa prezentuje vrámci užívateľského rozhrania. Oba termíny spadajú pod interakciu človeka s počítačom. Jazyky ako HTML a CSS sa zameriavajú na uľahčenie vytvárania užívateľského rozhrania. [21]

Prvým krokom pri návrhu užívateľského rozhrania bolo zvoliť farby ktoré sa budú používať pre jednotlivé komponenty. Zvolila som tri farby, a to primárnu, sekundárnu a farbu pre varovania.




**Obr. 3.5** Zvolené farby pre užívateľské rozhranie

Následne boli vytvorené jednotlivé wireframe-y pre predstavu, ako má užívateľské rozhranie vyzerať. Tieto wireframe-y boli vytvorené pomocou nástroja Figma (sekcia 3.4.2.1). Tento nástroj ponúka veľké množstvo funkcionalít, ako napríklad vytvorenie znovupoužiteľných komponentov, ktoré som využila pre urýchlenie vytvárania návrhu. Tiež je dostupných mnoho pluginov, využila som napríklad plugin Icons8 pre vkladanie ikoniek do návrhu. Vo výslednej aplikácii boli nejaké zmeny, ako napríklad presunutie navigácie doprava.




**Obr. 3.6** Návrh prezerania udalostí


**Obr. 3.7** Návrh detailu udalosti

Zvyšné návrhy sa nachádzajú v dodatku B.

## 3.4 Zvolené technológie

V tejto sekcii sú popísané jednotlivé technológie, ktoré som zvolila pre implementáciu klientskej a serverovej časti. Tieto technológie boli volené podľa architektúry aplikácie.

### 3.4.1 Serverová časť back-end

Pre vývoj serverových častí webových aplikácií je dostupné množstvo rôznych frameworkov. Medzi najznámajšie frameworky patria napríklad Spring, Laravel alebo ASP.NET. Všetky tieto frameworky umožňujú implementáciu MVC architektúry, no každá v inom jazyku. Rozhodla som sa pre Spring Framework, ktorý ponúka veľké množstvo modulov pre zjednodušenie implementácie. Ostatné frameworky taktiež ponúkajú množstvo výhod, v tomto prípade ide skôr o osobnú preferenciu.

Vo všetkých týchto frameworkoch sa dá využiť **ORM**, ktoré je podstatné pre prácu s databázou. ORM predstavuje vrstvu spájajúcu objektovo orientované programovanie s relačnými

databázami. Umožňuje vykonávanie CRUD operácií pri interakcii s databázou pomocou objektovo orientovaných jazykov. Pri použití ORM pre každú tabuľku v databáze existuje trieda reprezentujúca danú entitu. [22]

#### 3.4.1.1 Spring Framework

Spring je open source framework, ktorý umožňuje jednoduché vytváranie Java enterprise aplikácií poskytovaním všetkého potrebného pre používanie Javy v enterprise prostredí, vrátane podpory pre Kotlin a Groovy ako alternatívou v JVM. Navyše, vďaka flexibilite je možné vytvárať rôzne druhy architektúr v závislosti na požiadavkách aplikácie. [23]

Spring framework je rozdelený na moduly, ktoré si aplikácie volia podľa svojich potrieb. Jadro tvoria konfiguračné moduly a moduly s dependency injection. Spring taktiež poskytuje základnú podporu pre rozličné architektúry aplikácií, ako napríklad web, správy a iné. [23] Pre jednoduché vytvorenie aplikácie využívajucej moduly Spring Frameworku sa používa Spring Boot.

#### 3.4.1.2 Spring Boot

Spring Boot poskytuje jednoduché vytváranie spustiteľných aplikácií založených na Springu. Väčšina týchto aplikácií potrebuje minimálnu konfiguráciu. Ponúka rýchle a prístupné spustenie pre všetkých vývojárov, funkcionality ako napríklad vložené servery alebo externalizovaná konfigurácia. Taktiež nevyžaduje generovanie kódu ani požiadavky na XML konfiguráciu. [24]

#### 3.4.1.3 Spring Web

Spring Boot je vhodný aj na vývoj webových aplikácií. Vďaka nemu je možné vytvoriť samostatný HTTP server pomocou Jetty, Undertow, Tomcat alebo Netty. Vo väčšine webových aplikácií sa používa modul spring-boot-starter-web, prípadne pre vytvorenie reaktívnych aplikácií je to modul spring-boot-starter-webflux. [25] Tento modul využívam pre aplikovanie MVC architektúry.

#### 3.4.1.4 Spring Data

Spring Data sa zameriava na poskytovanie konzistentnej architektúry pre prácu s dátami založenej na Springu, pričom stále dodržiava špeciálne vlastnosti dátových úložísk. Vďaka Spring Data je možné jednoducho používať technológie na prístup k relačným aj nerelačným databázam, cloudovým službám a iným dátam. Vrámci tohto projektu je veľké množstvo subprojektov špecifických pre konkrétne databázy, ktoré su vyvíjané v spolupráci s mnohými spoločnosťami. [26]

Medzi jeho funkcie patria napríklad silné abstrakcie repozitára, vlastné mapovanie objektov a podpora pre auditáciu dát. Taktiež je pomocou neho možné dynamické odvodzovanie dotazov podľa názvov metód repozitára alebo vloženie svojho kódu do repozitára. Ľahko sa integruje s prostredím Spring pomocou JavaConfig a poskytuje aj pokročilú integráciu s MVC controllermi. Moduly Spring Data poskytujú technológie pre ORM, ktoré aplikujem. [26, 27]

#### 3.4.1.5 Jsoup

Jsoup je open source Java knižnica umožňujúca prácu s HTML, primárne získavanie dát z HTML súborov. [28] Túto knižnicu som si zvolila pre prácu so štatistikami.

#### 3.4.1.6 PostgreSQL

**Relačný databázový systém** je podkladovým databázovým softvérom, ktorý umožňuje prácu s relačnými databázami (sekcia 3.2). Vďaka nemu je možné v relačnej databáze vykonávať CRUD

operácie a zároveň poskytuje štruktúru dát, sieťový prístup, alebo kontrolu oprávnení. Všetky dáta ukladá do tabuliek. [29] Pre databázovú časť som sa rozhodla použiť PostgreSQL.

**SQL** je jazyk používaný na interakciu s relačnými databázovými systémami. Umožňuje jednoduché používanie CRUD operácií iba niekoľkými riadkami kódu. Vzhľadom na závislosť medzi týmto jazykom a relačnými databázovými systémami, sú niekedy relačné databázy nazývané aj SQL databázy. [29]

PostgreSQL je open source výkonný relačný databázový systém. Používa jazyk SQL, ktorý zároveň rozširuje o množstvo funkcií pre bezpečné ukladanie a spracovávanie náročných dátových záťaží. [30]

Tento systém je známy svojou overenou architektúrou, dátovou integritou, rozšíriteľnosťou, spoľahlivosťou a množstvom funkcií. Open source komunita neustále poskytuje inovatívne a výkonné riešenia, a aj preto je PostgreSQL voľbou mnohých organizácií pri výbere relačnej databázy. [30]

#### 3.4.1.7 OpenAPI

OpenAPI je špecifikácia, ktorá definuje štandardné a jazykovo nezávislé rozhranie pre HTTP API. Umožňuje ľuďom aj počítačom pochopiť schopnosti služby bez nutnosti prístupu zdrojovému kódu alebo dokumentácii. [31] Tento nástroj bol využitý pre vygenerovanie dokumentácie využívaného REST API.

### 3.4.2 Klientská časť front-end

Pri vývoji webových aplikácií je možné zvoliť tradičný prístup alebo SPA. Tradičným prístupom sú Multi-page applications, pri ktorých každé zobrazenie alebo odoslanie dát vyžaduje prerendrovanie novej stránky. AJAX umožňuje aktualizáciu len konkrétnej časti, ale to spôsobuje väčšiu zložitosť a ťažší vývoj v porovnaní so SPA. [32]

SPA alebo jednostránková aplikácia nevyžaduje pri používaní obnovenie stránky. Jedná sa len o jednu stránku, v ktorej sa obsah načítava pomocou JavaScriptu. To umožňuje rýchlejšie načítavanie obsahu a aj užívateľské akcie sú rýchlejšie, keďže vo väčšine prípadov nie je potrebné načítanie celej stránky. Tiež môže byť aplikácia používaná aj offline keďže dáta už boli uložené, a neskôr sa zosynchronizuje so serverom. Vzhľadom na rozsiahlejšiu funkčnosť a rýchlosť som sa rozhodla pre SPA prístup. [32, 33]

Pre vývoj front-endu webových aplikácií sa používajú hlavne programovacie jazyky JavaScript a TypeScript. Rozhodla som sa použiť TypeScript (popísaný v sekcii 3.4.2.4), ktorý je vlastne nadstavbou jazyka JavaScript, pretože poskytuje väčsie množstvo funkcií, a zároveň umožňuje jednoduchšie ladenie a odhaľovanie chýb.

Taktiež je dostupné veľké množstvo frameworkov, ktoré uľahčujú vývoj klientskej časti webových aplikácií. Rozhodovala som sa medzi tromi najznámejšími JavaScriptovými frameworkami:

1. React
2. Angular
3. Vue.js

Okrem toho sú celkom využívané napríklad JQuery, EmberJs, Django alebo Symfony.

Vo všetkých frameworkoch, medzi ktorými som sa rozhodovala, sa webová aplikácia implementuje ako SPA. Rozhodla som sa použiť Angular, ktorý narozdiel od Reactu využíva MVC model, vkladanie závislostí a pre prácu s dátami ponúka Two-way Data Binding.[34] Angular má oproti Vue lepšiu škálovateľnosť a ponúka Dependency Injection. Taktiež je Vue najmladší framework, takže má menej knižníc špecifických pre framework ako Angular. Čo sa týka celkovej

štruktúry, Angular na rozdiel od Reactu a Vue, vyžaduje presnú štruktúru projektu. [35] Pre mňa je presná štruktúra výhodou, ale to je osobnou preferenciou každého vývojára. Angular je detailnejšie popísaný vrámci sekcie 3.4.2.5.

Pre vytvorenie klientskej časti používam okrem TypeScriptu a Angularu HTML, ktorým vytváram šablóny pre jednotlivé komponenty a CSS pre definovanie vzhľadu týchto šablón.

### 3.4.2.1 Figma

Figma je cloudový nástroj pre návrh rozhraní umožňujúci tvorbu interaktívnych prototypov. Už od roku 2016 je veľmi obľúbeným nástrojom nie len pri webovom dizajne, ale aj v komunitách. Používatelia môžu medzi sebou zdieľať množstvo návrhov a šablón. [36]

Vďaka nástrojom na vektorovú grafiku umožňuje vytváranie zložitých optimalizovateľných wireframe-ov pre webové stránky. Navyše ponúka možnosť pridávania interaktívnych prvkov ako posúvanie myšou, vďaka čomu webové stránky pôsobia moderne. [36]

**Wireframing** je spôsob návrhu štruktúry webovej služby. Kladie dôraz na užívateľské potreby a podľa toho usporiadáva obsah stránky. **Wireframe** je vlastne rozloženie webovej stránky, ktoré určuje pozície prvkov nachádzajúcich sa na kľúčových stránkach. Jeho cieľom je poskytnúť vizualizácie pred začatím samotného vývoja. [37]

Pomocou tohto nástroja som vytvárala prvotný návrh dizajnu stránok.

### 3.4.2.2 HTML

HTML je základným stavebným prvkom webu, ktorý definuje štruktúru a zároveň význam celého obsahu webovej stránky. Pre definovanie vzhľadu stránky sa k HTML použije aj CSS a JavaScriptom, alebo v prípade tejto webovej aplikácie TypeScriptom, sa pridá stránke funkcionalita. [38]

Ide o značkovací jazyk, ktorý tvoria značky, tzv. tagy. Tieto tagy dodávajú význam každej oblasti stránky. Tagy môžeme rozlišovať na párové a nepárové. Párové tagy celý obsah obalia medzi začiatočnú (<>) a koncovú značku (</>). Naopak nepárová značka sa píše len raz a nie je potrebné ju uzavrieť. [39]

Okrem tagov existujú ešte elementy, ktorými označujeme prvky vložené na stránku pomocou tagov. Tagy sú teda značky, ktoré definujú elementy. [39]

HTML dokument musí mať určitú štruktúru. Na začiatok dokumentu musí byť vložený <!DOCTYPE>, ktorý udáva že ide o HTML súbor. Okrem toho je potrebné rozdeliť dokument na dve časti, a to hlavičku obsahujúcu informácie o dokumente a telo, v ktorom sa nachádza samotný obsah. [39]

| Element | Význam |
|---|---|
| <h1> | nadpis prvej úrovne |
| <nav> | reprezentuje oblasť stránky s navigačnými linkami |
| <section> | reprezentuje osobitnú sekciu dokumentu |
| <li> | reprezentuje položku v zozname |
| <ol> | reprezentuje zoradený zoznam |
| <p> | reprezentuje paragraf textu |
| <a> | s atribútom href vytvára hyperlink |
| <img> | vloží obrázok |
| <tab> | reprezentuje tabuľku |
| <button> | interaktívny element aktivovaný myšou |

**Tabuľka 3.5** Príklady niektorých HTML elementov a ich význam. [40]

#### 3.4.2.3 CSS

CSS je deklaratívny jazyk ovládajúci vzhľad webových stránok úpravou prvkov HTML dokumentov. Okrem HTML môže byť použitý aj pre iné značkovacie jazyky ako XML alebo SVG.[41]

CSS poskytuje veľké množstvo úprav, ktorými sa mení vzhľad prvkov daného dokumentu. Pomocou neho je možné meniť farbu a veľkosť textov. Taktiež umožňuje defnovanie rozloženia prvkov na stránke a môže sa použiť aj pre vytváranie animácií. [42]

Pre definovanie vzhľadu je potrebné uviesť selektor, ktorý reprezentuje daný HTML element. Za ním nasledujú zložené zátvorky, v ktorých sa nachádzajú jednotlivé deklarácie. Deklarácia štýlu daného prvku pozostáva z vlastnosti a jej hodnoty. [41, 42]

#### 3.4.2.4 TypeScript

TypeScript je open source jazyk vyvíjaný spoločnosťou Microsoft, ktorého cieľom je posunúť vývoj JavaScriptu na vyššiu úroveň a umožniť vývoj lepšie štruktúrovaných a udržateľnejších aplikácií. Je to vlastne nadmnožina jazyka JavaScript, ktorá je prekladaná na bežný JavaScript kód. TypeScript zavádza niekoľko koncepcií z objektovo orientovaných jazykov ako napríklad statické typovanie, rozhrania a iné. [43]

Voliteľným statickým typovaním transformuje JavaScript na silne typovaný programovací jazyk. Kompilátory a vývojové nástroje poskytujú lepšiu kontrolu a pomoc, a to práve vďaka statickému typovaniu, ktoré slúži ako obmedzenie funkcií, vlastností a premenných. Typová analýza sa vykonáva výlučne počas kompilácie, a teda nepriťažuje beh programu. [44]

#### 3.4.2.5 Angular

Angular je vývojová platforma založená na TypeScripte. Jeho súčasťou je framework založený na komponentoch, vďaka ktorému je možné tvoriť škálovateľné webové aplikácie. Taktiež ponúka integrované knižnice pokrývajúce mnoho funkcionalít vrátane formulárov, routovania, komunikácie klient-server, a ďalšie nástroje pre testovanie, vývoj a aktualizáciu kódu. [45]

V angulari sú **komponenty** stavebnými blokmi tvoriacimi celú aplikáciu. Súčasťou komponentu je trieda v Typescripte, ktorá má dekorátor @Component, šablóny HTML a CSS štýly. @Component definuje pre Angular špecifické informácie ako HTML šablóna určujúca zobrazenie v aplikácii a voliteľné CSS štýly definujúce vzhľad elementov. Tiež určuje CSS selektor stanovujúci použitie komponentu v šablóne, ktorej zodpovedajúce elementy sa stanú inštanciami daného komponentu. [46]

Angular rozširuje HTML v **šablónach** pomocou špeciálnej syntaxe, napríklad umožňuje dynamicky nastavovať a získavať hodnoty DOM (document object model) s funkciami, premennými a viazaním dát. Nie je potrebné ani zahrnutie prvkov ako <html>, <body>alebo <base>, keďže šablóna nepredstavuje celú stránku, ale len jej časť. [45]

Jedným z hlavných konceptov Angularu je **Dependency Injecton**, alebo inak DI, ktorý umožňuje konfiguráciu závislostí potrebnú pre jednotlivé komponenty. Spotrebiteľ a poskytovateľ závislostí sú hlavnými rolami DI, ktorých interakciu uľahčuje abstrakcia nazývaná Injector. Ten zisťuje či sa pri požiadavke závislosti inštancia nenachádza v jeho registri. Ak ju vo svojom registri nenájde, vytvorí novú a uloží ju do neho. [47]

Angular ponúka Two-way data binding, čo môže byť preložené ako dvojsmerná väzba dát. To umožňuje jednotlivým komponentom danej webovej aplikácie zdieľanie dát. Umožňuje použitie počúvania udalostí, čím je možné súčasne aktualizovať dáta medzi rodičovským komponentom a jeho potomkom. [48]

#### 3.4.2.6 Angular Material

Google v roku 2014 predstavil designový jazyk Material, ktorý o dva roky neskôr integroval do Angularu ako Angular Material. Angular Material je modul užívateľského rozhrania, ktorý pre

frameworky ako sú Angular, React a Vue.js poskytuje zbierku predpripravených komponentov, napríklad navigácia na webovej stránke, tabuľky, zaškrtávacie políčka a iné. Vďaka nemu je možné jednoducho vytvárať responzívne webové aplikácie. [49]

#### 3.4.2.7 FullCalendar

V informačnom systéme si okrem obyčajného zobrazenia všetkých udalostí môžu hráči tieto udalosti prezerať aj v kalendári. Pre toto zobrazenie som sa rozhodla použiť JavaScriptovú knižnicu FullCalendar, ktorá má aj platenú verziu, no pre potreby tejto webovej aplikácie úplne postačuje aj demo verzia.

FullCalendar sa ľahko integruje do rôznych frameworkov ako React, Vue.js a taktiež aj Angular. Všetkým poskytuje komponent odpovedajúci funkcionalite štandardného API FullCalendaru. [50]

# Kapitola 4

# Implementácia

## 4.1 Serverová časť

V tejto sekcii je popísaná implementácia serveru využitím technológií z predchádzajúcej kapitoly.

### 4.1.1 Databáza

Prvým krokom pri implementácii serveru bola konfigurácia databázy. Najprv bolo potrebné uviesť adresu, port a názov databázy pre úspešné pripojenie. Taktiež je potrebné uviesť užívateľské meno a heslo. Do konfigurácie som pridala aj ddl-auto=update, čo umožňuje automatické pridávanie tabuliek, stĺpcov a kľúčov. V konfigurácii pri ddl-auto je viac možností ako napríklad create (pri spustení aplikácie najprv všetko vymaže a potom vytvorí nanovo), create-drop (pri spustení sa všetko nanovo vytvorí a pri ukončení zmaže), none (je potrebná manuálna správa databázy). Pre update som sa rozhodla kvôli zachovaniu dát a automatickému generovaniu tabuliek.

```
spring.datasource.url=jdbc:postgresql://localhost:5432/bi-bap
spring.datasource.username=stelaaugustinova
spring.datasource.password=
spring.jpa.hibernate.ddl-auto=update
```

**Výpis kódu 1** Konfigurácia databázy

### 4.1.2 Entity

Po konfigurácii databázy nasledovalo vytvorenie entitných tried, ktoré reprezentujú jednotlivé tabuľky. V nasledujúcom kóde môžete vidieť základ entity Standings.

```
@Entity
public class Standings {

    @GeneratedValue(generator = "sequence-generator")
    @GenericGenerator(
            name = "sequence-generator",
            strategy = "org.hibernate.id.enhanced.SequenceStyleGenerator",
            parameters = {
                    @Parameter(name = "initial_value", value = "1")
            }
    )
    @Id
    private int id;

    @ManyToOne
    @JoinColumn( name = "competition_id" )
    private Competition competition;

    @ManyToOne
    @JoinColumn( name = "team_id")
    private Team team;

    @Nonnull
    private int points;

    public Standings() {
    }

    public Standings(Competition competition, Team team, int points) {
        this.competition = competition;
        this.team = team;
        this.points = points;
    }
}
```

**Výpis kódu 2** Reprezentácia entity Standings

Prvou dôležitou časťou pri vytváraní entity je pre danú triedu použiť anotáciu @Entity. Táto anotácia špecifikuje, že daná trieda je entita a podľa toho sú automaticky vytvorené dané tabuľky v databáze.

Pri definovaní triedy ako entity je potrebné zvoliť atribút, ktorý bude predstavovať identifikátor záznamu, a ten označiť anotáciou @Id. Tento atribút bude v tabuľke predstavovať primárny kľúč.

Pre Id som použila aj anotáciu @GeneratedValue, kedže by sa táto hodnota mala generovať. Vrámci tejto anotácie je možné definovať aj generátor tejto hodnoty anotáciou @GenericGenerator, ktorú som využila pre generovanie postupnosti hodnôt od 1.

Pre vyjadrenie vzťahov medzi jednotlivými entitami sú dostupné anotácie @OneToOne, @ManyToOne, @OneToMany a @ManyToMany. K týmto anotáciám je potrebné použiť aj anotáciu @JoinColumn, v ktorom je uvedený názov stĺpca, ktorý bude použitý ako cudzí kľúč. Pri použití @ManyToMany sa generuje pomocná tabuľka reprezentujúca tento vzťah, a preto sa namiesto @JoinColumn používa anotácia @JoinTable, vrámci ktorej sa uvádza názov novej tabuľky a

stĺpce, ktoré budú použité ako cudzie kľúče. Použitie anotácie @ManyToMany je zobrazené v nasledujúcom kóde, kde bola použitá pre vytvorenie pomocnej tabuľky pre dochádzku na tréningy.

```
@ManyToMany
@JoinTable(name = "Attendance",
            joinColumns = @JoinColumn(name = "training_id"),
            inverseJoinColumns = @JoinColumn(name = "player_id")
)
private List<Player> players;
```

**Výpis kódu 3** Použitie anotácie @ManyToMany

Ďalej sa v tejto triede nachádzajú konštruktory. Jeden bez parametrov, keďže to @Entity vyžaduje, a druhý s parametrami, ktorý využívam pri ukladaní dát, ten ale nie je povinný.

Vo výpise kódu nie sú zobrazené, ale v entite sa ešte nachádzajú gettery a settery pre získanie a nastavenie privátnych hodnôt. Gettery sú vytvorené pre každú premennú a settery pre každú premennú okrem id, s ktorým by sa po vytvorení nemalo ďalej manipulovať.

### 4.1.3 Repository

Spring poskytuje rozhrania JPA repozitárov, pomocou ktorých pracujem s databázou. JPA repozitár obsahuje predgenerované metódy ako findById, findAll, save, prípadne aj delete. Ak tieto metódy nie sú postačujúce, ľahko sa dogenerujú ďalšie, alebo sa použije @Query, v ktorom je uvedený dotaz, ktorý sa má nad tabuľkami vykonať.

```
public interface PlayerRepository extends JpaRepository<Player, Integer> {

    List<Player> findAllByTeam(Team team);

    @Query("SELECT players FROM PlayersList WHERE id = :list")
    List<Player> findAllByList(int list);

}
```

**Výpis kódu 4** Repozitár s dogenerovanými funkciami

### 4.1.4 DTO

DTO sú objekty využívané pre prenos dát medzi procesmi s cieľom minimalizovať počet volaných metód. Tento návrhový vzor zaobaľuje viaceré parametre do jedného volania, čo znižuje záťaž v prípadoch vzdialených operácií. Ďalšou výhodou je zapuzdrenie serializačnej logiky, ktorá prekladá objekt a dáta do špecifického formátu, ktorý sa môže ukladať a prenášať. [51]

Pre posielanie dát cez HTTP požiadavky som využila práve DTO. Pri DTO, narozdiel od entít nie je vzťah s inou tabuľkou reprezentovaný danou entitou, ale len jej id. Je to z dôvodu, že pri prenášaní celých entít by sa prenášalo veľké množstvo dát, kedže aj tieto entity môžu mať vzťahy s inými tabuľkami. Takto sa prenesie len id entity, ktorá ma vzťah s danou entitou, a v prípade, že sú potrebné aj hodnoty tej entity, tak sa na ňu pošle osobitná HTTP požiadavka.

Pre každú entitu som vytvorila dve takéto triedy, a to DTO a CreateDTO. DTO využívam pre posielanie dát zo servera, a naopak CreateDTO server prijíma pri požiadavke na vytvorenie

záznamu alebo jeho úpravu. Tieto dve triedy sa líšia len parametrom id, ktorý sa pre CreateDTO nevyužíva, kvôli automatickému generovaniu v entitách.

```
public record CompetitionDTO(int id, String competitionName,
                                Date startDate, Date endDate) {}

public record CompetitionCreateDTO(String competitionName,
                                    Date startDate, Date endDate) {}
```

**Výpis kódu 5** DTO a CreateDTO pre entitu Competition

Všetky tieto premenné vrámci DTO sú immutable, čo znamená, že sa nemôžu meniť. Preto namiesto vytvorenia triedy, ktorá by mala každú premennú final, používam record, čo je vlastne špeciálna trieda pre immutable premenné. Record automaticky vygeneruje konštruktor a gettery, ktoré sú potrebné na získavanie dát z DTO.

### 4.1.5 Service

Ďalšou časťou sú services, alebo služby, ktoré predstavujú logiku servera. Pomocou nich sa spracovájú požiadavky a vracajú vhodné dáta. Príklad Service uvediem pre súpisky a osobitne popíšem jej časti.

```
@Service
public class PlayersListService {

    private final PlayersListRepository playersListRepository;

    private final TeamService teamService;
    private final PlayerService playerService;

    @Autowired
    public PlayersListService(PlayersListRepository playersListRepository,
        TeamService teamService, PlayerService playerService) {
        this.playersListRepository = playersListRepository;
        this.teamService = teamService;
        this.playerService = playerService;
    }
```

**Výpis kódu 6** Service pre súpisky

Spring ponúka anotáciu @Service pre triedy reprezentujúce služby konkrétnych entít. V týchto triedach vždy využívam repozitár zhodnej entity ako je v službe, keďže jeho prostredníctvom získam dáta alebo vykonám zmeny v databáze. Okrem toho v prípade potreby pridávam aj služby iných entít, s ktorými má hlavná entita vzťah. V tomto prípade ide o PlayersListService, a keďže PlayersList má väzby na tím aj na hráčov, pridávam aj služby daných entít.

Vďaka DI je možné nastavenie závislostí objektu a pre vkladanie objektov do iných objektov nie je potrebná konkrétna inštancia objektu. V Springu sú zavedené anotácie pre DI zabezpečujúce vkladanie spolupracujúcich objektov alebo tried.[52, 53] V konštruktore používam

anotáciu @Autowired a vďaka nej Spring zabezpečí inštanciu daného repozitára, prípadne doplnkových služieb.

Začínajúc s verziou Spring 2.5, framework zaviedol Dependency Injection riadený anotáciami. Hlavnou anotáciou tejto funkcie je @Autowired. Tá umožňuje Springu vyriešiť a vložiť spolupracujúce beany do nášho beana.

```
private PlayersListDTO toDTO(PlayersList playersList){
        return new PlayersListDTO(playersList.getId(),
                playersList.getListName(),
                playersList.getTeam().getId(),
                playersList.getPlayers()
                            .stream()
                            .map(Player::getId)
                            .collect(Collectors.toList()));
    }

private Optional<PlayersListDTO> toDTO(Optional<PlayersList> playersList){
    if (playersList.isEmpty())
        return Optional.empty();
    return Optional.of(toDTO(playersList.get()));
}
```

**Výpis kódu 7** Funkcie pre vytvorenie DTO

Dôležitou súčasťou service sú funkcie pre vytvorenie DTO zo získanej entity. Repozitár vráti entitu, ale pre jej odoslanie (sekcia 4.1.4) je potrebné získané dáta vracať ako DTO. Vytvorená musí byť aj funkcia vracajúca Optional DTO, keďže funkcie volané z repozitára, ako napríklad findById(), taktiež vracajú Optional danej entity.

```
public List<PlayersListDTO> findAllByTeam(int teamId) throws Exception{

    Optional<Team> optionalTeam = teamService.findById(teamId);
    if (optionalTeam.isEmpty())
        throw new ResponseStatusException(HttpStatus.BAD_REQUEST);

    return playersListRepository.findAllByTeam(optionalTeam.get())
            .stream()
            .map(this::toDTO)
            .collect(Collectors.toList());
}

public Optional<PlayersList> findById(int id){
    return playersListRepository.findById(id);
}
```

**Výpis kódu 8** Príklady vyhľadávania v service

Súčasťou logiky je vyhľadávanie potrebných záznamov. V tomto príklade uvádzam vyhľadávanie súpisiek pre daný tím. V prvom rade je potrebné skontrolovať, či vôbec daný tím existuje, bez

neho sa žiadne súpisky nenájdu. Ak tím existuje, využijem vygenerovanú funkciu v repozitári, pomocou ktorej vrátim zoznam daných súpisok. Ďalším príkladom je využitie predgenerovanej metódy z repozitára.

```
@Transactional
public PlayersListDTO create(
PlayersListCreateDTO playersListCreateDTO) throws Exception{

    Optional<Team> optionalTeam = teamService.findById(playersListCreateDTO
        .team()
        );
    if (optionalTeam.isEmpty())
        throw new ResponseStatusException(HttpStatus.BAD_REQUEST);

    List<Player> players = playerService.findByIds(playersListCreateDTO
        .players()
        );

    if (players.size() != playersListCreateDTO.players().size())
        throw new ResponseStatusException(HttpStatus.BAD_REQUEST);

    return toDTO(playersListRepository.save(
            new PlayersList(playersListCreateDTO.listName(),
                    optionalTeam.get(),
                    players
            )
    ));
}
```

**Výpis kódu 9** Vytvorenie záznamu

V service sa taktiež vytvárajú, update-ujú a odstraňujú záznamy. V prípade, že sa pri vytváraní alebo aj úprave môžu vyskytnúť nejaké chyby, je potrebné funkciu oanotovať pomocou @Transactional. Vďaka tejto anotácii sa vytvorí transakcia a po skončení funkcie sa vykoná commit. V prípade, že nastane nejaká chyba sa záznam neuloží. V tejto funkcii pre vytvorenie záznamu môže chyba nastať, ak tím, pre ktorý sa súpiska vytvára, neexistuje. Preto je potrebné pomocou PlayerService skontrolovať, či vôbec taký tím existuje. To isté platí aj pre zoznam hráčov.

Úprava záznamov nie je výrazne odlišná od ich vytvárania. Parametrom funckie pre úpravu je okrem CreateDTO aj id upravovaného záznamu. V úprave je navyše potrebné skontrolovať, či už daný záznam existuje a následne len upraviť jeho atribúty.

Pre odstránenie záznamov je potrebné funkcii dodať len id a následne prebehne kontrola existencie záznamu. Potom sa daný záznam vymaže použitím predgenerovanej funkcie repozitára.

### 4.1.6 Controller

Poslednou časťou pre aplikovanie MVC architektúry sú Controllery, ktoré umožnia odosielanie a prijímanie dát cez HTTP. V nasledujúcej ukážke sú zobrazené len niektoré funkcie Controlleru pre tímy.

```java
@RestController
@RequestMapping("/api")
public class TeamController {

    private final TeamService teamService;

    @Autowired
    public TeamController(TeamService teamService) {
        this.teamService = teamService;
    }

    @GetMapping("/team")
    List<TeamDTO> all(){
        return teamService.findAll();
    }

    @PostMapping("/team")
    TeamDTO save(@RequestBody TeamCreateDTO teamCreateDTO){
        return teamService.create(teamCreateDTO);
    }

    @PutMapping("/team/{id}")
    TeamDTO update(@PathVariable int id,
    @RequestBody TeamCreateDTO teamCreateDTO) throws Exception{
        return teamService.update(id, teamCreateDTO);
    }

    @DeleteMapping("team/{id}")
    void delete(@PathVariable int id) throws Exception{
        teamService.delete(id);
    }
}
```

**Výpis kódu 10** Vytvorenie záznamu

Spring ponúka pre tvorbu controllerov anotácie @Controller a @RestController. Použila som @RestController, keďže pre komunikáciu medzi klientom a serverom používam REST API. Anotáciu @RequestMapping používam pre namapovanie požiadaviek na metódy controlleru. Pre získanie všetkých tímov url bude .../api/team.

Okrem toho, anotáciou @PathVariable získavam premenné nachádzajúce sa v danej URL, ktoré sú v danom Mappingu uzavreté v {}. Použitím anotácie @RequestBody sa v HTTP požiadavke vyžaduje telo, ktoré bude reprezentovať dané CreateDTO pre vytvorenie alebo úpravu záznamu.

Pre vykonanie CRUD operácií som použila anotácie @PostMapping(create), @GetMapping(read), @PutMapping(update), @DeleteMapping(delete).

### 4.1.7 WebConfig

Poslednou a veľmi dôležitou časťou pri implementácii klienta bolo nastavenie CORS pre umožnenie komunikácie medzi serverom a klientom prostredníctvom REST API. CORS je vlastne mechanizmus založený na HTTP hlavičkách umožňujúci serveru indikovať rôzne domény, schémy alebo

porty (okrem vlastných), ktoré sú povolené pre načítavanie zdrojov. [54] V allowedOrigins som zadala odkiaľ môže server prijímať požiadavky a zároveň som v allowedMethods určila, ktoré HTTP metódy môže použiť.

```
@Configuration
@EnableWebMvc
public class WebConfig implements WebMvcConfigurer {

    @Override
    public void addCorsMappings(CorsRegistry registry){
        registry.addMapping("/**")
                .allowedOrigins("http://localhost:4200/")
                .allowedMethods("GET", "POST", "PUT", "DELETE")
                .allowedHeaders("Authorization", "Content-Type");
    }
}
```

**Výpis kódu 11** Konfigurácia Web MVC

### 4.1.8 Spracovanie štatistiky

Boli mi sprístupnené štatistiky tímu Tatran Prešov vytvorené počas zápasov, ktoré boli vo formáte HTML. Kedže je väčšina logiky v serveri, rozhodla som sa ich spracovávať vrámci serverovej časti. Pre spracovávanie HTML súborov existuje Java knižnica Jsoup.

Pre štatistiky bolo vytvorené špeciálne CreateDTO, ktoré pre štatistiky zápasov obsahuje id zápasu, pre ktorý bola vytvorená, a pre štatistiky hráčov id na štatistiku zápasu, ku ktorej patria. Okrem toho tieto CreateDTO obsahujú URL na súbor danej štatistiky. DTO sú už vytvorené klasicky ako všetky iné, keďže sú potrebné pre zobrazenie daných štatistík.

Po poslaní požiadavky s danou URL sa najprv overí, či daná štatistika už nebola uložená. Keďže je to súbor vytvorený počas zápasu a ďalej sa už neupravuje, je možné ho spracovať iba raz. Po uložení sa už štatistika meniť nedá, a ak by bola potrebná zmena, musí sa vymazať a pridať nanovo.

Nateraz sa vyžaduje, aby súbor so štatistikou bol uložený na Google Drive a mal povolený prístup pre každého, kto má jeho adresu. Google Drive som zvolila kvôli jeho popularite, takže predpokladám, že s ním vie pracovať takmer každý. Tomu je prisposôbené aj samotné sťahovanie súboru.

```
public String downloadStatFile(String fileUrl){

        String ret = "";
        try {
            String fileId = getFileId(fileUrl);
            URL url = new URL("https://drive.google.com/uc?export=download&id="
                + fileId);
            InputStream inputStream = url.openStream();
            OutputStream outputStream = new FileOutputStream("statP");
            byte[] buffer = new byte[1024];
            int bytesRead;
            while ((bytesRead = inputStream.read(buffer)) != -1) {
                outputStream.write(buffer, 0, bytesRead);
            }
            outputStream.close();
            inputStream.close();
            ret = "statP";
        } catch (IOException e) {
            e.printStackTrace();
        }

        return ret;
    }

    public String getFileId(String fileUrl){
        String[] urlParts = fileUrl.split("/");

        String fileId = "";

        for (int i = 0; i < urlParts.length; i++) {
            if (urlParts[i].equals("d")) {
                fileId = urlParts[i + 1];
                break;
            }
        }

        return fileId;
    }
```

**Výpis kódu 12** Stiahnutie súboru z Google Drive

Pre sťahovanie súborov z Google Drive sa využíva iná URL, ktorá je poskytnutá pre zdieľanie. Preto je prvým krokom získanie id súboru, ktoré je potrebné na jeho stiahnutie. URL poskytnutá pre zdieľanie vždy obsahuje „.../d/fileId/...“, táto časť url je vždy extrahovaná vo funkcii getFileId() (výpis kódu 12). Následne sa vo funkcii downloadStatFile() pomocou novej URL súbor stiahne.

Po stiahnutí súborov so štatistikami sa použitím knižnice Jsoup rozparsuje daný HTML súbor. Vzhľadom na presnú štruktúru dát nebolo nutné zložité prechádzanie HTML súboru, postačovalo si len zistiť poradie stĺpca, v ktorom sa daný údaj nachádza.

**Štart Nové Zámky - Štart Nové Zámky - Tatran Prešov 28:35**

**Hráči v poli**

| | Hráč | Góly | Strely | Úspešnosť streľby | 7m | 6m | Brejk | Krídlo | 9m | Protiútok | +7 | -7 | Asistencia | Získaná lopta | Stratená lopta | 2 min | Červená karta | Žltá karta | Plus obrana | Mínus obrana | Prešľap, kroky | Vylúčený súper | Blok | Pr |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 2 | Oliver Rábek | 4 | 6 | 66 % | 0/1 | 0/0 | 0/0 | 0/0 | 4/5 | 0/0 | | 1 | 2 | | 1 | | | | 3 | 2 | | | | |
| 5 | Tomáš Fech | 10 | 11 | 90 % | 0/0 | 1/1 | 0/0 | 4/5 | 0/0 | 5/5 | 1 | | 1 | 3 | | | | | 2 | | | | | |
| 9 | Dominik Krok | 1 | 1 | 100 % | 0/0 | 1/1 | 0/0 | 0/0 | 0/0 | 0/0 | | | | | | | | | | | | | | |
| 11 | Erik Fenár | 0 | 0 | 0 % | 0/0 | 0/0 | 0/0 | 0/0 | 0/0 | 0/0 | | | | | | | | | | 1 | | | | |
| 21 | Nikola Ivanovič | 6 | 9 | 66 % | 0/0 | 2/2 | 0/0 | 0/0 | 2/5 | 2/2 | | 1 | 5 | 1 | 1 | | | | 1 | 7 | 1 | | | |
| 22 | Viacheslav Kasatkin | 0 | 0 | 0 % | 0/0 | 0/0 | 0/0 | 0/0 | 0/0 | 0/0 | | | | | | | | | | | 1 | | | |
| 28 | Sergio Lopez Garcia | 1 | 2 | 50 % | 0/0 | 0/0 | 0/0 | 0/1 | 0/0 | 1/1 | | | | 1 | 1 | | | | | 1 | | | 1 | |
| 33 | Dávid Michalka | 1 | 2 | 50 % | 0/0 | 0/0 | 0/0 | 1/1 | 0/0 | 0/1 | 1 | | | | | | | | 1 | 1 | | 1 | | |
| 34 | Marko Davidovič | 2 | 3 | 66 % | 0/0 | 0/1 | 0/0 | 0/0 | 1/1 | 1/1 | | | 1 | | 2 | 1 | | | | | | | | |
| 44 | Djordje Golubovič | 2 | 2 | 100 % | 0/0 | 1/1 | 0/0 | 0/0 | 1/1 | 0/0 | | | 1 | | | | | | | | | | | |
| 47 | Lukáš Urban | 3 | 4 | 75 % | 0/0 | 1/1 | 0/0 | 0/0 | 1/1 | 1/2 | 1 | | 4 | 2 | 1 | | | | 4 | 2 | | 1 | 2 | |
| 66 | Pavel Caballero Hernandez | 3 | 5 | 60 % | 1/2 | 2/3 | 0/0 | 0/0 | 0/0 | 0/0 | | | 5 | 3 | 4 | 1 | | | 1 | 2 | | | | |
| 77 | Roman Carapkin | 2 | 2 | 100 % | 0/0 | 2/2 | 0/0 | 0/0 | 0/0 | 0/0 | 1 | 2 | | | | 2 | | | 3 | | 1 | 1 | | |

**Brankári**

| | Hráč | Chytené strely | Strely | Úspešnosť brankára | 7m | 6m | Krídlo | 9m | Protiútok | Góly | +7 | -7 | Asistencia | Získaná lopta | Stratená lopta | 2 min | Červená karta | Žltá karta | Plus obrana | Mínus obrana | Prešľap, kroky | Vylúčený súper | Blok | Preráža |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 12 | Tilen Leben | 2 | 2 | 100 % | 1/1 | 1/1 | 0/0 | 0/0 | 0/0 | 0/0 | | | | | | | | | | | | | | |
| 73 | Igor Chupryna | 10 | 29 | 34 % | 0/2 | 3/11 | 0/0 | 7/12 | 0/4 | 0/0 | | | | | | | | | | | | | | |
| 86 | Marcos Colodeti | 5 | 14 | 35 % | 0/1 | 1/4 | 1/2 | 3/3 | 0/4 | 0/0 | | | 2 | | | | | | | | | | | |

**Obr. 4.1** Štruktúra štatistiky pre jednotlivých hráčov

V nasledujúcich ukážkach nie je zobrazené získanie všetkých atribútov, zobrazené sú len tie, ktoré sa získavali odlišným spôsobom.

Pomocou knižnice Jsoup bol stiahnutý súbor prečítaný a uložený ako Document, následne sa tento súbor odstráni, keďže už nie je potrebný, všetky údaje už sú načítané v Document.

Pomocou funkcie select, v ktorej je uvedený názov získavaného elementu boli získané telá tabuliek s dátami. V prvej tabuľke sa vždy nachádzajú hráči a v druhej brankári. Preto pre získane dát o hráčoch je zvolená prvá tabuľka, z ktorej sa následne vyberú všetky jej riadky pomocou funkcie children().

```
String downloadedFileName = downloadStatFile(statUrl);

File file = new File(downloadedFileName);

Document stats = Jsoup.parse(file);
file.delete();

Elements playersTable = stats.select("TBODY");
Elements players = Objects.requireNonNull(playersTable.first()).children();
```

**Výpis kódu 13** Získanie potrebných elementov zo súboru HTML

Následne je potrebné pre každého hráča zistiť, či vôbec existuje. Zvolila som vyhľadávanie podľa čísla dresu a tímu, keďže pri mene by mohla ľahko nastať nejaká chyba, a číslo dresu sa väčšinou počas sezóny nemení.

Ako je možné vidieť na obrázku 4.1, v stĺpcoch sa vyskytuje len pár typov údajov. Góly a strely sú číslo, ktoré je zadané vždy aj v prípade, že je to nula. Ďalej sa nachádza údaj v % predstavujúci úspešnosť streľby, strely z určitých pozícií sú zadané ako góly/strely, a ostatné údaje sú číslo, ktoré je zadané len v prípade, že je väčšie ako nula.

Údaje boli získavané zadaním čísla poradia stĺpca, na ktoré bola zavolaná funkcia get(), ktorá umožňuje získanie kompletného textu nachádzajúceho sa vrámci daného elementu.

Jednotlivé typy údajov boli získané nasledovne:

- Pre góly a streľby, ktoré sú zadané vždy, nebola potrebná žiadna špeciálna úprava, postačujúca bola iba konverzia na int.

```
int goals = Integer.parseInt(cols.get(2).text());
int shots = Integer.parseInt(cols.get(3).text());
```

**Výpis kódu 14** Získanie dát pre góly a streľby

- Pre percentuálnu úspešnosť streľby bol využitý regulárny výraz, pomocou ktorého bolo získané len číslo z textu.

```
String efficiencyPattern = "\\d+";
Pattern pattern = Pattern.compile(efficiencyPattern);
Matcher matcher = pattern.matcher(cols.get(4).text());
int efficiency = 0;
if (matcher.find()) {
    String efficiencyString = matcher.group();
    efficiency = Integer.parseInt(efficiencyString);
}
```

**Výpis kódu 15** Získanie percentuálnej úspešnosti streľby

- Pri streľbách z určitých pozícií bol text rozdelený podľa znaku /, následne sa prvá časť uložila ako góly z pozície a druhá ako strely z pozície.

```
String[] wing = cols.get(8).text().split("/");
int goalWing = Integer.parseInt(wing[0]);
int shotsWing = Integer.parseInt(wing[1]);
```

**Výpis kódu 16** Spracovanie údajov pre strely z konkrétnych pozícií

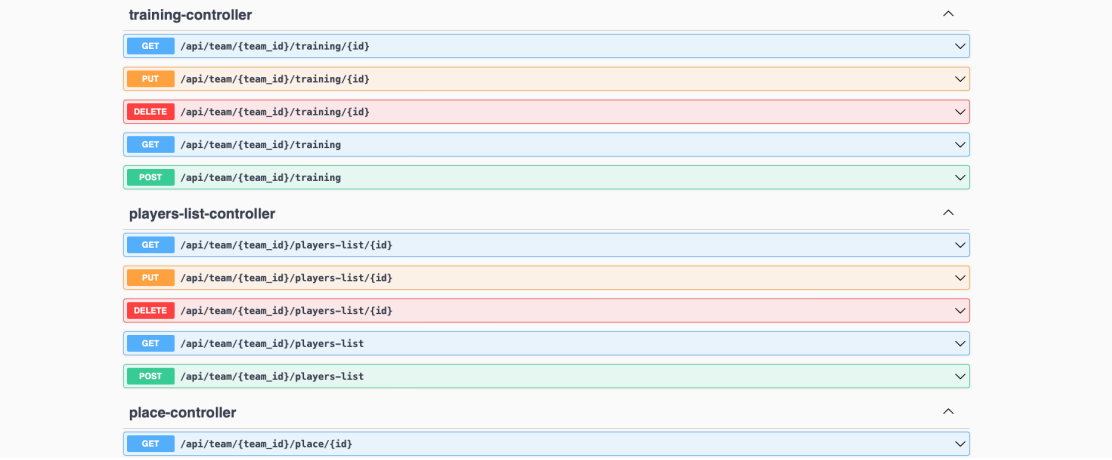
- V prípade ostatných údajov bola pre daný údaj predvolene nastavená 0, a v prípade, že sa daný údaj v štatistike nachádzal, bola jeho hodnota pripočítaná. Pre zjednodušenie štatistiky som sa rozhodla všetky chyby hráča uložiť ako turnover, keďže to nie je potrebné uvádzať až do detailov.

```
int turnover = 0;
if (cols.get(15).text().length() != 0)
    turnover += Integer.parseInt(cols.get(15).text());

if (cols.get(21).text().length() != 0)
    turnover += Integer.parseInt(cols.get(21).text());

if (cols.get(20).text().length() != 0)
    turnover += Integer.parseInt(cols.get(20).text());

if (cols.get(24).text().length() != 0)
    turnover += Integer.parseInt(cols.get(24).text());
```

**Výpis kódu 17** Spracovanie chýb hráča

### 4.1.9 Dokumentácia API

Pomocu nástroja OpenAPI (sekcia 3.4.1.7) bola vygenerovaná dokumentácia implementovaného REST API. Táto dokumentácia je prístupná po spustení servera na adrese http://localhost:8080/api-docs. Kompletná dokumentácia je zobrazená v prílohe C.




**Obr. 4.2** Dokumentácia REST API

## 4.2 Klientská časť

V tejto časti je popísaná implementácia front-endu webovej aplikácie.

### 4.2.1 Inicializácia

Pre JavaScrip, resp. TypeScript existuje správca balíčkov npm, ktorý umožňuje inštaláciu, update a odstraňovanie balíčkov pre aplikáciu. Pred vytvorením projektu som si pomocou npm nainštalovala Angular CLI, ktoré uľahčuje a zároveň urýchľuje vývoj Angular aplikácií.

```
npm install -g @angular/cli
```

**Výpis kódu 18** Inštalácia Angular CLI

Po inštalácii Angular CLI som mohla využívať ng pre uľahčenie vývoja. Pomocou ng v príkazovom riadku som vytvorila nový projekt s nainicializovanou aplikáciou.

```
ng new project-name
```

**Výpis kódu 19** Vytvorenie projektu

Po inicializácii projektu bolo v zložke src vytvorených niekoľko súborov. Prvým dôležitým súborom je app.module.ts. V tomto súbore sa nachádza deklarácia všetkých komponentov patriacich danej aplikácii a importy všetkých modulov, ktoré využívam. Príkladom môže byť MatSidenavModule z balíčku @angular/material, ktorý som využila pre vytvorenie navigácie.

Okrem toho boli ešte vytvorené súbory app.component.ts, app.component.html a app.component.css. Tieto súbory predstavujú základný komponent aplikácie.

### 4.2.2 Entity

Aj v klientskej časti bolo potrebné vytvoriť entity. Tieto ale nefungujú ako @Entity v serverovej časti, predstavujú iba reprezentáciu entít a ich atribútov. Sú dôležitou súčasťou pre prijímanie a odosielanie dát, a taktiež pre ich manipuláciu v jednotlivých komponentoch.

```
export interface Player {

  id: number;
  givenName: string;
  surname: string;
  age: number;
  nationality: string;
  position: string;
  team: number;
  photo: string;
  height: number;
  weight: number;
  shirtNumber: number;
  registrationNumber: number;
  user: number;
}
```

**Výpis kódu 20** Reprezentácia entít

### 4.2.3 Service

Pre komunikáciu serverom prostredníctvom REST API bolo potrebné vytvoriť služby, ktoré s ním budú komunikovať. Služba sa jednoducho vytvorí pomocou Angular CLI.

```
ng generate service player
```

**Výpis kódu 21** Vytvorenie služby s názvom player

Týmto príkazom sa vytvorí základ triedy PlayerService.

V prvom rade je nutné túto triedu oanotovať pomocou dekorátoru @Injectable. To zabezpečí využitie Dependency Injection v komponentoch.

Pre umožnenie komunikácie pomocou HTTP protokolu bol do app.module.ts pridaný HttpClientModule. Vďaka nemu je možné do každej service v konštruktore vložit HttpClient, ktorý umožňuje odosielanie požiadavok na server.

Pre posielanie HTTP požiadaviek som zadefinovala aj URL potrebné pre získanie požadovaných dát. V prípade hráča to boli URL samostatne pre hráčov, pre tím, ktorá sa využíva pre získanie hráčov daného tímu a URL pre súpisky na získanie hráčov v danej súpiske.

HTTP client obsahuje funkcie pre vykonávanie HTTP metód, využívam konkrétne GET, POST, PUT a DELETE. Štandardne sa pri metódach get, post a put vracia netypovaný JSON objekt. Pridaním voliteľnej špecifikácie sa tento netypovaný JSON objekt pri prijímaní HTTP odpovede automaticky prekonvertuje na danú entitu, resp. pole týchto entít. Tieto entity následne využívajú jednotlivé komponenty. Pri odosielaní požiadavky na server sa v parametroch danej metódy zadáva URL, prípadne aj inštancia entity, z ktorej sa taktiež vytvorí JSON objekt.

Okrem toho HTTP client poskyuje aj HttpParams. Pomocou metódy set sa nastavia dané parametre, ktoré sa potom odošlú v danej URL.

Všetky volania HTTP metód su asynchronné, teda neblokujú program kým API vráti odpoveď s dátami. Preto tieto metódy, ktoré poskytuje HttpClient, vždy vracajú Observable<>danej entity. V angulari sa observable využívajú na riešenie bežných asynchronných operácií.

Požiadavky klienta na server sa odosielajú následovne:

- Metóda **GET** s využitím parametrov

```
filterEvents(team_id: number, training: boolean, competitions: number[]):
    Observable<Event[]>
    {
        let params = new HttpParams().set('training', training)
        competitions.forEach( competition =>
            params = params.append('competition', competition)
        )
        return this.http.get<Event[]>(`${this.team_url}/${team_id}/event`,
            {params}
            )
    }
```

**Výpis kódu 22** Získanie udalostí s použitím parametrov

- Metóda **POST**, v ktorej sa posiela aj entita hráča

```
savePlayer(player: Player): Observable<Player>{
    return this.http.post<Player>(this.player_url, player);
  }
```

**Výpis kódu 23** Uloženie nového hráča

- Metóda **PUT**, ktorá pre dané id hráča aktualizuje jeho údaje podľa danej entity

```
updatePlayer(player : Player): Observable<Player>{
    return this.http.put<Player>(`${this.player_url}/${player.id}`, player)
  }
```

**Výpis kódu 24** Update existujúceho hráča

- Metóda **DELETE**, ktorá vymaže pre určitý tím zápas s daným id

```
deleteMatch(id: number, team_id: number): Observable<Match>{
    return this.http.delete<Match>(`${this.team_url}/${team_id}/match/${id}`)
  }
```

**Výpis kódu 25** Odstránenie zápasu

### 4.2.4 Router

Pre prechádzanie medzi jednotlivými komponentmi aplikácie sa využíva routovanie. Kvôli tomu je potrebné vytvoriť router, ktorý sa má zadefinovať v súbore app-routing.module.ts Ten sa opäť vygeneruje pomocou ng. V nasledujúcej ukážke sa –flat využije pre vytvorenie súboru v zložke src a –module=app pre automatické importovanie do AppModule.

```
ng generate module app-routing --flat --module=app
```

**Výpis kódu 26** Vytvorenie modulu pre routovanie

V tomto module je potrebné naimportovať všetky komponenty, na ktoré budú definované cesty viesť. Jednotlivé cesty sa uvedú ako pole Route. Každá Route musí obsahovať cestu a komponent, na ktorý vedie. Okrem toho bola nastavená aj predvolená Route, ktorá prázdnu cestu presmeruje na domovskú obrazovku.

```
const routes: Routes = [
  {path: '', redirectTo: '/home', pathMatch: "full"},
  {path: 'home', component: HomescreenComponent},
  {path: 'events', component: EventsComponent},
  {path: 'events/new_event', component: EventEditComponent},
  {path: 'training/:id', component: TrainingComponent},
];

@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule { }
```

**Výpis kódu 27** AppRoutingModule

Navyše router dodáva element pre HTML <router-outlet>, ktorý zobrazí žiadaný komponent. A taktiež routerLink, ktorý pri aplikovaní na element v šablóne vytvorí odkaz, ktorý naviguje na zadanú cestu.

### 4.2.5 Komponenty

Ako som popisovala v sekcii 3.4.2.5, komponenty sú blokmi, ktoré tvoria celú aplikáciu. Každý komponent tvoria súbory so šablónou komponentu, jeho štýly a trieda komponentu.

#### 4.2.5.1 Angular Material Komponenty

V šablónach sú často využité komponenty, ktoré poskytuje Angular Material. Tieto komponenty majú už preddefinovaný vzhľad, aj často využívané metódy pre daný komponent.

**Navigácia**

Pre implementáciu navigácie bol využitý komponent mat-sidenav-container, ktorý predstavuje kontajner pre celkový obsah a navigáciu. V tomto komponente sa uvádzajú ďalšie dva komponenty, a to mat-sidenav-content, v ktorom sa definuje obsah, a mat-sidenav pre samotnú navigáciu. Vrámci mat-sidenav som využila komponent mat-nav-list, v ktorom sú umiestnené jednotlivé položky navigácie.

**Expansion Panel**

Angular Material poskytuje komponent mat-expansion-panel, ktorý sa po kliknutí rozbalí. Tento komponent bol využitý vrámci navigácie, napríklad pre štatistiky, ktoré po rozkliknutí zobrazia možnosti štatistík hráča a zápasov.

**Toolbar**

Kontajner mat-toolbar môže obsahovať rôzne nadpisy a akcie. Nachádza sa v hornej časti aplikácie a vrámci neho je umiestnené tlačidlo pre zobrazenie/skrytie navigácie.

**Tabuľky**

Pre zobrazenie tabuliek bol využitý mat-table, ktorý poskytuje štýly pre tabuľku podľa Material Design. Pri tabuľkách sú zadefinované aj zobrazené stĺpce a vzhľad hlavičky tabuľky.

**Formulárové prvky**

Komponent mat-form-field uľahčuje vytváranie formulárových prvkov. Tento komponent obaľuje iné komponenty tvoriace daný formulárový prvok. V tomto komponente je možné zadefinovať aj jeho vzhľad pomocou atribútu appearance, ktorý môže mať jednu z hodnôt fill alebo outline.

Vrámci každého formulárového prvku bol využitý mat-label, v ktorom sa uvádza, údaj, ktorý je potrebné zadať.

Využité boli viaceré typy formulárových prvkov. Najčastejšie som používala základný pre zadanie textu alebo čísla. Okrem toho boli použité aj formulárové prvky s výberom poskytnutých možností. Pri zadávaní dátumu bol vrámci mat-form-field pridaný aj komponent mat-date-picker, ktorý umožní okrem zadávania dátumu aj jeho výber z kalendára

```
<mat-form-field appearance="fill">
        <mat-label>Competition</mat-label>
        <input matInput>
        <mat-select [(ngModel)]="match.competition"
            (selectionChange)="competitionChange()">
          <mat-option *ngFor="let competition of competitions"
            [value]=competition.id>{{competition.competitionName}}
          </mat-option>
        </mat-select>
      </mat-form-field>
```

**Výpis kódu 28** Formulárový prvok s výberom

**Zoznam s výberom**

Pre filtrovanie udalostí a evidenciu dochádzky som sa rozhodla použiť komponent mat-selection-list. Ten obaľuje dané mat-list-options, ktoré predstavujú všetky možnosti výberu.

Pri filtrácii udalostí bola zadefinovaná aj funkcia, ktorá sa vykoná pri každej zmene výberu v tomu zozname. Konkrétne v tomto prípade ide o znovunačítanie udalostí podľa filtra.

```
<mat-selection-list [(ngModel)]="filter"
    (ngModelChange)="onFilterChange()">
  <mat-list-option *ngFor="let competition of competitions"
        [value]="competition.id">
    {{competition.competitionName}}
  </mat-list-option>
  <mat-list-option [value]="0" >Training</mat-list-option>
</mat-selection-list>
```

**Výpis kódu 29** Filter pomocou mat-selection-list

**Card**

Komponent mat-card predstavuje kontajner pre obsah, ako napríklad text a obrázky. Pre mat-card je možné využiť niekoľko rôznych častí. Pri implementácii som využila konkrétne mat-card-header, čo je úsek nachádzajúci sa v hornej časti mat-card. V ňom je uvedený názov danej mat-card. Obsah je zadefinovaný v komponente mat-card-content. Ako posledné som použila mat-card-actions, v ktorom som väčšinou definovala tlačidlá pre úpravu alebo odstránenie záznamu reprezentovaného danou mat-card.

### 4.2.5.2 Vlastné Komponenty

Celú aplikáciu tvoria komponenty vygenerované pomocou Angular CLI. Použitím nasledujúceho príkazu sa vygenerujú súbory pre nový komponent, a to name.component.ts s triedou daného komponentu, name.component.html obsahujúci šablónu, name.component.css s definíciou jeho vzhľadu, a compontn.spec.ts, kde sa nachádzajú testy.

```
ng generate component name
```

**Výpis kódu 30** Vytvorenie nového komponentu

Po vytvorení komponentu som najprv definovala jeho šablónu v HTML súbore. V šablónach sú využité aj komponenty z Angular Material. Pre vloženie premenných definovaných v triede sa daná premenná uvedie ako {{ názov }}. Pre zobrazovanie len vybraných sekcií alebo iných elementov, je využitý atribút pre daný element *ngIf. V direktíve *ngIf je uvedená podmienka, ktorá v prípade vyhodnotenia ako pravdivá, zobrazí danú časť šablóny. V ukážke kódu 31 sa časť s tabuľkou zobrazí, len ak sú definovaní hráči. Ďalšou direktívou, ktorá je často využívaná je ngModel. Tá sa využíva pri formulárových prvkoch a zobrazená je vo výpise kódu 28. Uvedenie ngModel vrámci [()] umožňuje synchronizáciu dát medzi užívateľským rozhraním a daným komponentom. Ak sa hodnota aktualizuje v užívateľskom rozhraní, aktualizuje sa aj v komponente, a naopak.

```
<img class="team_photo" src="{{team.photo}}" alt="team">

<section class="team">
<h2>{{team.teamName}}</h2>

<section class="team_info">
  <section class="info">
    <dl>
      <dt>City</dt>
      <dd>{{team.city}}, {{team.country}}</dd>
      ...
    </dl>
    <button *ngIf="role == 3" mat-button
        routerLink="/team/{{team.id}}/edit">edit
    </button>
  </section>

  <section class="table mat-elevation-z8" *ngIf="players">
    <mat-table [dataSource]="players">
        ...
    </mat-table>
  </section>
</section>
</section>
```

**Výpis kódu 31** Príklad šablóny komponentu

Následne som implementovala triedu daného komponentu. Bolo nutné pridať požadované služby pre získavanie dát pre daný komponent, resp. pre ich aktualizáciu. Keďže sú služby oanotované ako @Injectable(), s využitím DI sa jednoducho vložia do triedy komponentu, pridaním v jej konštruktore. Okrem toho je do tried vložený aj Storage. Ten je využitý pre ukladanie často používaných dát, aby nebolo nutné v každej triede posielať požiadavky na server. Sú v ňom uložené dáta ako užívateľ, jeho profil a tím. V prípade potreby je vložený aj Router.

Keďže by sa v konštruktore mala nachádzať minimálna inicializácia a nemali by sa v ňom volať funkcie posielajúce HTTP požiadavky, je definovaná funkcia ngOnInit(). Vo funkcii ngOnInit sa vykonajú všetky funkcie pre inicializáciu dát, ktoré využívajú vložené služby. Angular túto funkciu vykonáva po vytvorení inštancie daného komponentu.

Všetky funkcie volané v ngOnInit volajú funkcie zo služieb, ktoré odosielajú požiadavky na server. Po zavolaní funkcie sa použije metóda subscribe, ktorá čaká na vrátenie odpovede, keďže odpoveď zo servera nemusí prísť hneď. Po prijatí odpovede sa premenné komponentu nainicializujú dátami z odpovede.

```
getPlayers() : void {
    if (this.team)
      this.playerService.getTeamPlayers(this.team.id)
        .subscribe(players => this.players = players)
}
```

**Výpis kódu 32** Získanie dát zo služby

Funkcie získavajúce dáta sa nevyužívajú len v ngOnInit, ale aj pri zmene nejakého komponentu, napríklad pri zobrazení udalostí sa pri zmene filtra zakaždým odošle požiadavka na získanie udalostí podľa vybraných kritérií. Okrem získavania dát boli vytvorené aj funkcie pre vytvorenie nových záznamov, ich úpravu a odstránenie. Tieto funkcie ale nie sú volané v ngOnInit, ale sú naviazané na tlačidlá, a teda sa vykonávajú po kliknutí na príslušné tlačidlo.

### 4.2.6 Kalendár

Pre implementáciu prezerania udalostí bol vytvorený komponent Calendar, v ktorom sa využíva JavaScriptová knižnica FullCalendar (sekcia 3.4.2.7), ktorá poskytuje komponent aj pre Angular.

Tento komponent sa integruje do šablóny pomocou elementu <fullcalendar>, pričom vlastnosti tohto kalendára sa nastavia vrámci elementu pomocou vlastnosti [options]. Do [options] sa vložia CalendarOptions, ktoré ponúkajú možnosti pre definovanie vzhľadu kalendáru.

Ako prvé bol nastavený celkový vzhľad, rozhodla som sa pre len mesačné zobrazenie. Je možné pridať aj zobrazenie po dňoch, týždňoch alebo viacerých mesiacoch. Zvolila som len mesačné, keďže sa udalosti môžu prezerať ešte v zozname alebo ako udalosti len aktuálneho dňa, tak by iné zobrazenia neboli veľmi využívané.

Nasledovalo nastavenie vzhľadu jednotlivých udalostí, pre nich boli využité vlastnosti eventDisplay a eventTimeFormat. EventDisplay som zvolila ako block, čo v mesačnej mriežke zobrazuje udalosti ako obdĺžnik. V eventTimeFormat je nastavené, aby sa čas v jednotlivých udalostiach vypísal ako hodina:minúta am/pm.

Pre definíciu hlavičky existuje vlastnosť headerToolbar, v ktorej sa definuje, čo sa nachádza na začiatku hlavičky, v jej strede a na konci. Na začiatok som umiestnila tlačidlo pre prepnutie na predchádzajúci mesiac a na koniec tlačidlo pre zobrazenie nasledujúceho mesiaca. V strede hlavičky je vypísaný aktuálny mesiac a rok.

Poslednou a najdôležitejšou súčasťou calendarOptions, ktorú som využila, je pridanie daných udalostí. Pre jednotlivé udalosti je možné definovať niekoľko vlastností, ako napríklad id, začiatok, koniec, či trvá celý deň, názov, url, farba pozadia, farba rámčeka, a iné. Pre každú udalosť som zvolila jej id, názov, dátum a čas jej začiatku, a farbu. Pre prehľadné rozlíšenie udalostí som zvolila viaceré farby. Tie sú priradené podľa toho, či ide o tréning alebo zápas. Zápasy sa rozlišujú ešte podľa konkrétnej súťaže. K dispozícii je 5 farieb, jedna pre tréning a zvyšné štyri pre zápasy. Počet farieb pre zápasy som volila podľa toho, že nie je veľmi pravdepodobné, že jeden tím sa zúčastňuje naraz viac ako 4 rôznych súťaží. V prípade, že to nastane, bude viacerým zápasom rôznych súťaží priradená rovnaká farba.


**Obr. 4.3** Zvolené farby pre udalosti v kalendári


**Obr. 4.4** Udalosti v kalendári

# Kapitola 5
# Testovanie

Testovanie je dôležitou súčaťou vývoja všetkých aplikácií. Testovanie softvéru je vlastne proces overovania a hodnotenia, či aplikácia robí to, čo má robiť. Vďaka testovaniu je možné predísť chybám, taktiež zlepšiť celkový výkon aplikácie. [55]. Testovať sa môže všetko od serveru, užívateľského rozhrania, až po samotnú výkonnosť systému. Rozhodla som sa zamerať na užívateľské testovanie.

## 5.1 Užívateľské testovanie

Užívateľské testovanie by malo byť súčasťou tvorby webu alebo mobilných aplikácií. Je to vlastne analytická metóda zameriavajúca sa na interakciu skutočných užívateľov, a teda poskytuje relevantné dáta. Vďaka tomu majú designéri a programátori potrebnú spätnú väzbu od cieľovej skupiny. [56]

Jednotliví účastnící testovania predstavujú jednotlivé cieľové skupiny. Chápanie prvkov webu a správna orientácia užívateľov sú posudzované testovacím tímom. Tieto informácie vyjadrujú prehľad a jednoduchosť stránok. Cieľom je vytvárať designy, ktoré uľahčujú užívateľom prácu s webovými stránkami alebo aplikáciami, bez zložitého premýšľania, ako danú aplikáciu používať.[56]

Prvým krokom užívateľského testovania je výber prvkov, na ktoré bude testovanie zamerané. Celý proces môže byť zhrnutý do nasledujúcich krokov [56]:

1. výber testovaných prvkov
2. výber vhodných testerov
3. vytvorenie scenárov
4. vyhodnotenie
5. prípadná optimalizácia

### 5.1.1 Voľba vhodných účastníkov

Pri výbere testerov je dôležité brať do úvahy charakteristiku cieľovej skupiny. Ideálne je zvoliť 5 testerov, keďže už tento počet dokáže odhaliť množstvo chýb. [56]

Pre testovanie prototypu aplikácie som zvolila xy užívateľov. Užívatelia sa aktívne venujú hádzanej, prípadne iným športom, čo z nich robí vhodných adeptov cieľovej skupiny.

**Detaily testerov**

- Tomáš F.

  21 rokov, aktívny hráč hádzanarského klubu Tatran Prešov, študent vysokej školy

- Daniel O.

  20 rokov, aktívny hráč hádzanarského klubu Tatran Prešov kategória dorast, študent vysokej školy

- Peter A.

  44 rokov, vysokoškolsky vzdelaný, vo voľnom čase sa aktívne venuje športu a trénovaniu

- Richard F.

  49 rokov, vysokoškolsky vzdelaný, vytváranie štatistík a vedenie mládeže v hádzanárskom klube Tatran Prešov

- Martin K.

  31 rokov, vysokoškolsky vzdelaný, manažér, aktívne sa venuje športu

- Barbora K.

  26 rokov, vysokoškolsky vzdelaná, bývalá hráčka a trénerka mládeže hádzanarského klubu ŠŠK Prešov

- Tomáš A.

  37 rokov, vysokoškolsky vzdelaný, vo voľnom čase sa aktívne venuje športu

### 5.1.2 Testovacie scenáre

Pri testovaní budú zvolené osoby testovať systém pre všetky užívateľské role, a to hráč, tréner a manažér. V aplikácii už budú testovacie dáta, aby bolo možné simulovať reálne používanie aplikácie.

### 5.1.3 Testovanie hráčov

1. Prihlásiť sa do systému
2. Upraviť svoj profil

   Overuje sa, či je pre užívateľa jednoduché sa orientovať v aplikácii, keďže musí nájsť svoj profil a následne prejsť na editáciu. Ďalej sa overí, či je pre užívateľa jednoduché a jednoznačné ako svoj profil upraviť a potvrdiť úpravu.
3. Prezrieť informácie o spoluhráčoch

   Taktiež sa zisťuje jednoduchosť orientácie v aplikácii. Zameriava sa na to, či užívateľ bude spoluhráčov hľadať cez tím, alebo sa to bude snažiť nájsť niekde inde. Tiež sa zisťuje, či užívateľovi napadne kliknúť na hráča v tabuľke, alebo nebude vedieť nájsť jeho profil.
4. Zistiť detaily najbližšieho tréningu

   V tomto scenári sa zisťuje, či hráč bude detail udalosti hľadať medzi udalosťami, alebo sa bude snažiť k udalosti dostať cez kalendár a bude chcieť kliknúť na danú udalosť. Okrem toho sa overí, či je prezeranie udalostí intuitívne a užívateľ klikne na udalosť v zozname pre zobrazenie detailov.

5. Prezerať svoje štatistiky

   Tento test skúma, či pri prezeraní štatistík bude užívateľovi jasné, že musí pred zobrazením zvoliť hráča.

6. Návrat na domovskú obrazovku

   Zisťuje sa, či užívateľ uprednostní kliknutie na názov systému, ktorý ho presmeruje na domovskú obrazovku, alebo túto položku vyhľadá v navigácii.

7. Zistiť svoje identifikačné údaje

   Test overuje jednoduchosť vyhľadania užívateľových identifikačných údajov.

8. Odhlásiť sa zo systému

   Overenie, či si užívateľ všimol v navigácii položku pre odhlásenie.

### 5.1.4 Testovanie trénerov

1. Prihlásiť sa do systému

2. Pridať nový tréning

   Tento test zisťuje, či užívateľ pôjde hneď na udalosti, kde sa nachádza tlačidlo pre vytvorenie tréningu, alebo sa o to bude pokúšať pri zobrazení kalendára. Taktiež overuje, či je vytváranie tréningu intuitívne, a užívateľ bude vedieť rýchlo tréning vytvoriť a uložiť.

3. Vytvoriť súpisku a priradiť ju k najbližšiemu zápasu určitej súťaže

   V tomto scenári sa overuje, že užívateľ nájde položku pre súpisky v menu, a bude pre neho jednoduché ju vytvoriť. Následne užívateľ musí nájsť danú udalosť. Skúma sa ako ju uživateľ bude hľadať, či bude prechádzať všetky udalosti, alebo využije možnosť filtrovania. Následne musí upraviť daný zápas priradením súpisky a uložiť ho.

4. Upraviť svoj profil

   Overuje sa, či je pre užívateľa jednoduché sa orientovať v aplikácii, keďže musí nájsť svoj profil a následne prejsť na editáciu. Ďalej sa overí, či je pre užívateľa jednoduché a jednoznačné ako svoj profil upraviť a potvrdiť úpravu.

5. Zaznamenať dochádzku

   Zisťuje sa, či užívateľ ľahko nájde požadovaný tréning, a či dokáže zadať dochádzku. Taktiež sa overuje, či bude užívateľovi jasné, že sa dochádzka uloží automaticky, alebo bude hľadať ako ju uložiť.

6. Odhlásiť sa zo systému

   Overenie, či si užívateľ všimol v navigácii položku pre odhlásenie.

### 5.1.5 Testovanie manažérov

1. Prihlásiť sa do systému

2. Upraviť svoj profil

   Overuje sa, či je pre užívateľa jednoduché sa orientovať v aplikácii, keďže musí nájsť svoj profil a následne prejsť na editáciu. Ďalej sa overí, či je pre užívateľa jednoduché a jednoznačné ako svoj profil upraviť a potvrdiť úpravu.

3. Vytvoriť nové športovisko pre zápasy

   Tento test zisťuje, ako jednoducho používateľ nájde prezeranie športovísk, kde je možnosť vytvoriť nové. Tiež je potrebné zistiť ako reaguje na rozloženie, a či pre neho bude jednoduché prejsť na vytváranie nového záznamu. Skúma sa aj, či je pre užívateľa jasné ako vytvoriť športovisko pre zápasy.

4. Priradiť vytvorené športovisko k zápasu.

   V tomto teste musí užívateľ nájsť zápas, ktorý chce upraviť. Rovnako ako aj pri trénerovi sa skúma, kde bude chcieť udalosť upraviť. Ak v predchádzajúcom teste vytvoril športovisko, ktoré nie je vhodné na zápasy, je potrebné zistiť ako bude ďalej postupovať, a či sa mu to podarí upraviť.

5. Pridať užívateľa do tímu a zároveň mu vytvoriť hráčsky profil alebo odstrániť hráča z tímu

   V teste sa opäť skúma jednoduchosť orientácie v aplikácii, keďže sa uživateľ najprv musí dostať na profil tímu a pri jeho úprave pridať hráča. Je potrebné skúmať ako dlho to užívateľovi bude trvať, keďže pri pridávaní hráča má viac možností.

6. Vytvoriť nový zápas

   V tomto scenári sa skúma schopnosť užívateľa správne vytvoriť zápas. Dôležité je pozorovať, akým spôsobom bude zápas vytvárať, keďže zadávanie položiek v poradí, v akom sú zobrazené, mu môže vytváranie urýchliť.

7. Návrat na domovskú obrazovku

   Zisťuje sa, či užívateľ uprednostní kliknutie na názov systému, ktorý ho presmeruje na domovskú obrazovku, alebo túto položku vyhľadá v navigácii.

8. Zobraziť zápasy najbližšieho mesiaca

   Tento test umožňuje zistiť, ako by si bežný užívateľ tohto systému prezeral budúce udalosti. Zápasy boli zvolené z toho dôvodu, že ich je menej, takže sa skúma, či užívateľ zvolí možnosť kalendára, využije filter udalostí, alebo zvolí úplne iný spôsob.

9. Zaznamenať dochádzku

   Zisťuje sa, či užívateľ ľahko nájde požadovaný tréning, a či dokáže zadať dochádzku. Taktiež sa overuje, či bude užívateľovi jasné, že sa dochádzka uloží automaticky, alebo bude hľadať ako ju uložiť.

10. Pridať k ukončenému zápasu štatistiku

    Užívateľovi bude sprístupnená adresa pre súbory štatistiky zápasu a štatistiky hráčov. Skúma sa ako ich bude zadávať, keďže je potrebné ich zadať v určitom poradí.

11. Odhlásiť sa zo systému

    Overenie, či si užívateľ všimol v navigácii položku pre odhlásenie.

## 5.2 Zhrnutie testovania

Väčšina osôb nemala s pohybom vrámci aplikácie problém a až na menšie zaváhania sa v aplikácii pohybovali plynule. Jedna osoba sa so systémom stretla aj počas vývoja, a tá nemala žiadne problémy pri vykonávaní určených scenárov. Testovanie hráča prebehlo celkovo 5-krát, trénera 4-krát a manažéra 3-krát.

Pri testovaní hráča sa skúmala najmä schopnosť pohybu v apkliácii, keďže rola hráč neponúka pokročilejšie funkcionality, ide skôr o prezeranie rôznych častí aplikácie. Pre niektorých bol na začiatku testu problém sa zorientovať, keďže po prihlásení menu ešte nie je otvorené, ale treba ho

otvoriť kliknutím na ikonu. Pri zobrazovaní detailov o tréningu sa zisťovalo, či osoba uprednostní zoznam udalostí alebo kalendár. Väčšina osôb išla cez zoznam udalostí, možnosť kalendára zvolila len jedna. Pri prezeraní cez udalosti bol pre niektorých problém zobraziť detail, keďže klikali na celú udalosť a nie na ikonu šípky, ktorá by ich presmerovala na detail udalosti. Pri návrate na domovskú obrazovku nikto nevyužil možnosť kliknutia na názov aplikácie, všetci išli cez možnosť Home v navigácii. Najväčším problémom pre testerov bolo zistiť svoje identifikačné údaje. Všetci až na jedného, ktorý bez zaváhania zamieril do settings, skúšali hľadať najprv v profile a prechádzali takmer celú aplikáciu. Prihlásenie, odhlásenie a prezeranie štatistík prebehlo bez problémov.

Počas testovania trénera bolo pre testerov jednoduché vytvoriť tréning. Pri vytváraní súpisky niekoľko z nich chcelo ísť cez zápas a hľadali to pri úprave daného zápasu. Najväčší problém bol pri zadávaní dochádzky. Testeri to našli ľahko ale mali problém s ukladaním. Pri dochádzke sa nenachádza žiadne tlačidlo pre uloženie, dochádzka sa ukladá automaticky. To bolo pre nich problémom, pretože sa všetci snažili nájsť tlačidlo pre uloženie. Okrem toho všetky ostatné body v scenári prebehli bez väčších problémov.

Pri testovaní manažéra bolo pre niektoré osoby ťažšie vytvoriť športovisko pre zápasy, keďže prehliadali checkbox, ktorý bol na to určený. Keď chceli dané športovisko priradiť k zápasu, museli sa potom vrátiť k športoviskám a upraviť ho. To im ale nikdy netrvalo dlho a rýchlo sa im podarilo športovisko zmeniť a priradiť. Keďže manažér môže vytvárať aj tréning aj zápas, je nutné sa pri vytváraní udalosti medzi týmito typmi prepnúť. To si ale väčšina užívateľov nevšimla, a namiesto zápasu vytvárali tréning, po chvíli si to ale všimli a vytvorili zápas. Pri pridávaní užívateľa do tímu problémy neboli, no pri odstránení sa zistilo, že odtraňovanie hráča nefunguje. To bolo hneď aj opravené. Pri pridávaní štatistiky k zápasu neboli žiadne komplikácie.

Užívateľské testovanie splnilo moje očakávania. Pri zadávaní dochádzky som predpokladala, že užívatelia sa budú snažiť nájsť tlačidlo pre uloženie, čo sa aj potvrdilo. Podľa môjho názoru im orientácia v aplikácii nerobila väčšie problémy a scenáre vykonávali celkom rýchlo. Problémy boli hlavne ak niečo prehliadli, čo v niektorých prípadoch bolo spôsobené tým, že sa to snažili vyplniť rýchlo. Hodnotenia systému od užívateľov boli kladné, vedeli by si predstaviť ho používať pre organizáciu udalostí v športovom klube. Návrhy pre vylepšenie od testerov sú popísané v nasledujúcej kapitole, ktorá sa zaoberá možnými rozšíreniami.

# Kapitola 6
# Možné rozšírenia aplikácie

Aplikácia bola zameraná len na tri typy užívateľov, a to hráč, tréner a manažér. Tieto tri typy osôb má každý tím, ale okrem nich sa vo veľkých kluboch nachádzajú aj iné, a preto by najbližšie rozšírenia mohli byť zamerané na nich. Ďalej by mohli byť pridané ďalšie funkcionality pre uľahčenie zdieľania informácií vrámci klubov.

## 6.1 Fyzioterapeut

Každý veľký klub má minimálne jedného fyzioterapeuta, ktorý pomáha zraneným hráčom ku čo najrýchlejšiemu návratu na palubovku. Okrem toho uľahčuje a urýchľuje regeneráciu hráčov medzi tréningami a zápasmi.

V tomto rozšírení by bol pridaný nový typ užívateľa, fyzioterapeut. Bol by pre neho dostupný zoznam zranených hráčov, s ktorými by si v osobitnom kalendári mohol naplánovať tréningy a procedúry pre rýchlejšie uzdravenie. Udalosti by už neboli zobrazované pre celý tím, ale pre jednotlivých užívateľov.

Ďalej by bol pridaný rezervačný systém, kde by si hráči po tréningu v čase vymedzenom fyzioterapeutom, mohli rezervovať masáž alebo iné procedúry umožňujúce rýchlejšiu regeneráciu.

## 6.2 Štatistik

Či už ide o menší alebo väčší klub, počas zápasov sa vytvára štatistika. Pre implementáciu mi boli sprístupnené štatistiky klubu Tatran Prešov, ale štatistika vždy takejto štruktúry nemusí byť pravidlom. Preto by bolo vhodné integrovať vytváranie štatistiky priamo do informačného systému. Vyriešilo by to aj možné problémy s nahrávaním súborov, aj s ich štruktúrou.

Pri tomto rozšírení by bol pridaný typ užívateľa štatistik. Ten by mal v menu medzi štatistikami aj možnosť vytvorenia novej štatistiky. Po spustení času v čase začiatku zápasu by bolo možné zvoliť hráča a nasledovne k nemu zvoliť požadovanú akciu. Vďaka takému zadávaniu štatistiky, by okrem štatistík, ktoré sú implementované teraz, bolo možné prezerať celý priebeh zápasu.

## 6.3 Posielanie správ

Pre uľahčenie komunikácie vrámci tímu by mohlo byť pridané odosielanie správ. Pri vytvorení tímu by bola automaticky vytvorená nová skupina so všetkými členmi tímu. Noví členovia by boli taktiež automaticky pridaní. Okrem toho by každý užívateľ mohol začat súkromnú konverzáciu s ktorýmkoľvek členom v tíme.

## 6.4 Rozšírenia navrhnuté po testovaní

Počas užívateľskéhop testovania boli testermi navrhnuté nasledujúce vylepšenia:

1. Zmena lokácie ID

   Pre užívateľov bolo komplikované nájsť svoje ID, tak navrhovali zmenu jeho miesta. V budúcnosti by ale ID nemuselo byť vyžadované, keďže teraz sa používa pre priraďovanie do tímu, a to by sa v budúcnosti mohlo zmeniť.

2. Pridanie tlačidla pre uloženie dochádzky

   Užívateľom nebolo jasné ako uložiť dochádzku, takže by uvítali tlačidlo pre jej uloženie. Momentálne sa dochádzka ukladá automaticky. Toto vylepšenie bolo navrhnuté takmer každým užívateľom, ktorý testoval trénera.

3. Pridanie ďalšieho jazyka

   Toto rozšírenie bolo navrhnuté jedným užívateľom, ktorý by možnosť zmeny jazyka uvítal.

4. Viac odlíšiť vytváranie tréningu a zápasu

   Keďže si niektorí testeri nevšimli hneď, že je potrebné prepnúť typ udalosti, tak bolo navrhnuté toto vylepšenie.

Niektoré návrhy sa týkali zmeny dizajnu, ako pridanie ikoniek, zmena rozloženia domovskej obrazovky a iné. Keďže ale ide o prototyp systému, na dizajn za zatiaľ nekladol až taký dôraz.

# Kapitola 7
# Záver

Cieľom tejto práce bolo analyzovať, navrhnúť, implementovať a otestovať prototyp sytému určený pre hádzanárske kluby. Všetky tieto ciele boli splnené.

V prvej kapitole bola uvedená analýza existujúcich riešení, kde boli uvedené aj ich výhody a nevýhody. Nasledovalo určenie funknčných požiadaviek systému pre určenie funkcionalít daného systému. Tieto boli určené aj pomocou konzultácií s osobami, pre ktoré by bol tento systém určený.

Po analýze systému nasledoval jeho samotný návrh. Tento návrh sa zaoberá typom a architektúrou aplikácie a vhodnými technológiami pre implementáciu zvolenej architektúry. Systém je implementovaný ako webová aplikácia a má trojvrtstvovú architektúru podľa návrhového vzoru MVC, takže aplikácia je rozdelená na dátovú, aplikačnú a prezentačnú vrstvu. Aplikácia sa taktiež dá rozdeliť na klientskú a serverovú časť, ktoré spolu komunikujú prostredníctvom REST API. V návrhu je uvedený aj databázový model, ktorý aplikácia využíva. V poslednej časti druhej kapitoly sú uvedené vybrané technológie pre implementáciu navrhnutej aplikácie.

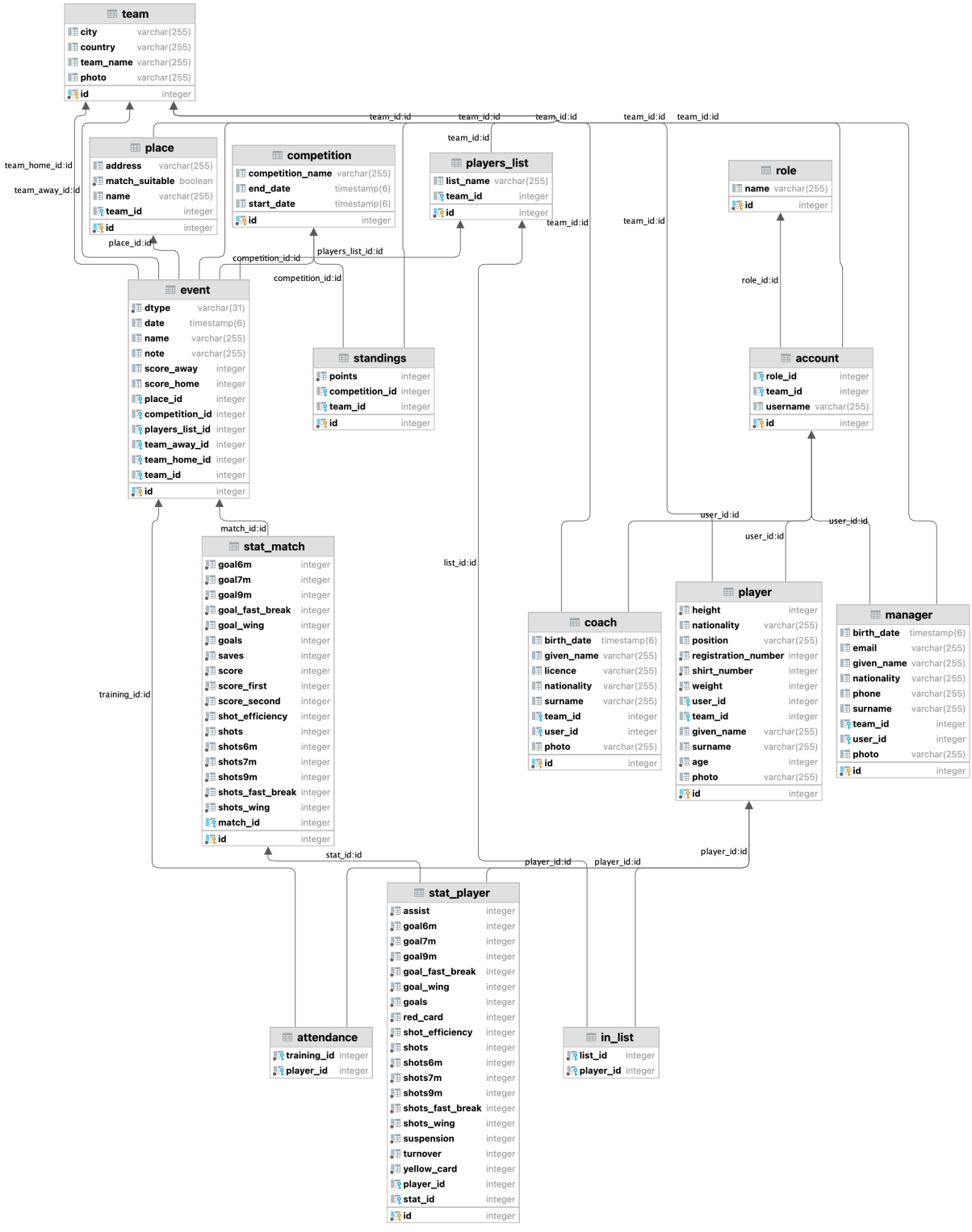
Tretia kapitola sa zaoberá samotnou implementáciou webovej aplikácie. Táto kapitola je rozdelená na dve časti. V prvej časti je uvedená implementácia back-endu, a v druhej je popísaná implementácia front-endu. Back-end je napísaný v jazyku Java a ako framework je využitý Spring Framework, ktorý umožňuje jednoduchú implementáciu MVC architektúry. Pre prácu s databázou je použitý relačný databázový systém PostgreSQL. Implementácia front-endu je pomocou jazyka TypeScript a využitý bol front-endový framework Angular. Pre uľahčenie implementácie front-endu boli využité knižnice Angular Material, z ktorého bolo využité množstvo preddefinovaných komponentov, a pre vytvorenie kalendára bola použitá JavaScriptová knižnica FullCalendar.

Nasledovalo testovanie prototypu systému. Využitím užívateľského testovania bola overená použiteľnosť a odhalené nedostatky aplikácie. Testy boli vykonané podľa uvedených scenárov, a vykonávali ich osoby, pre ktoré by bol systém určený.

Poslednou časťou tejto práce bolo uvedenie možných rozšírení. Pre konrétne rozšírenia som sa rozhodla z pozorovania, alebo boli navrhnuté užívateľmi, ktorí systém testovali.

Dodatok A

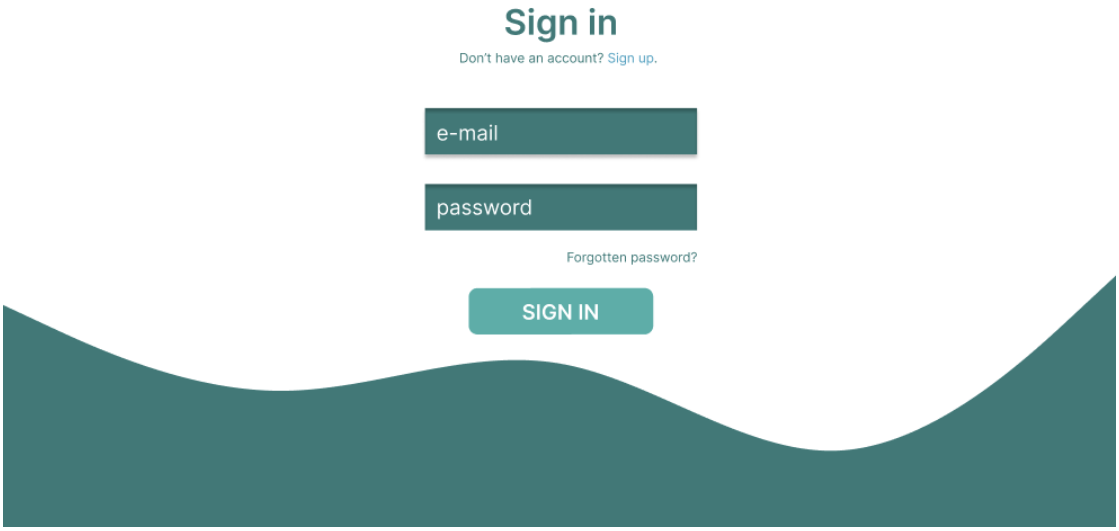
# Databázový model



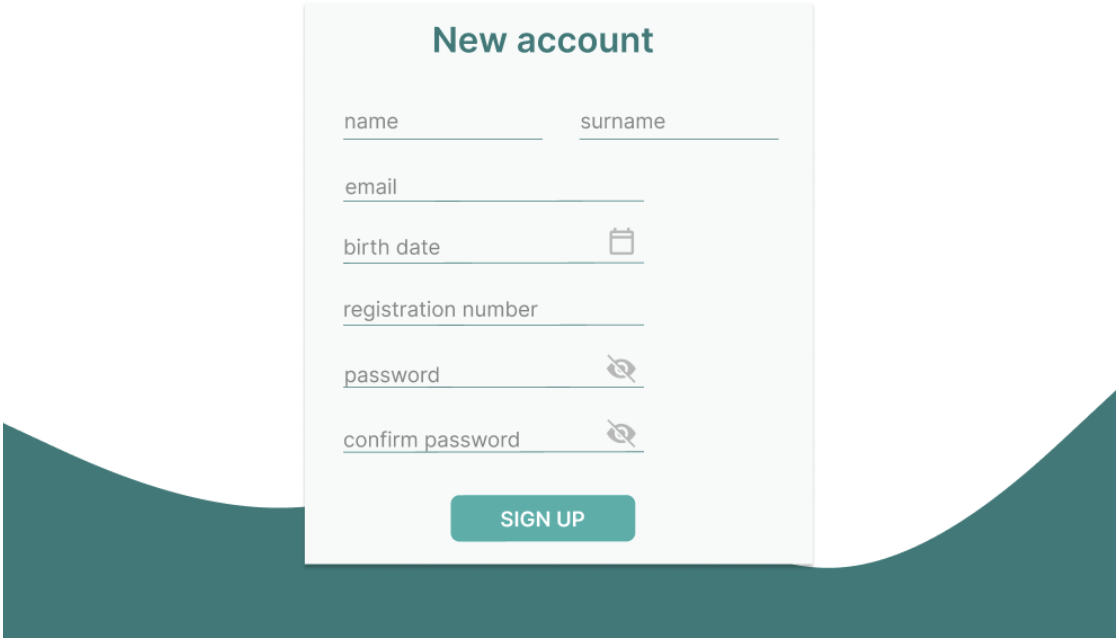

**Obr. A.1** Návrh databázy

Dodatok B

# Návrh užívateľského rozhrania

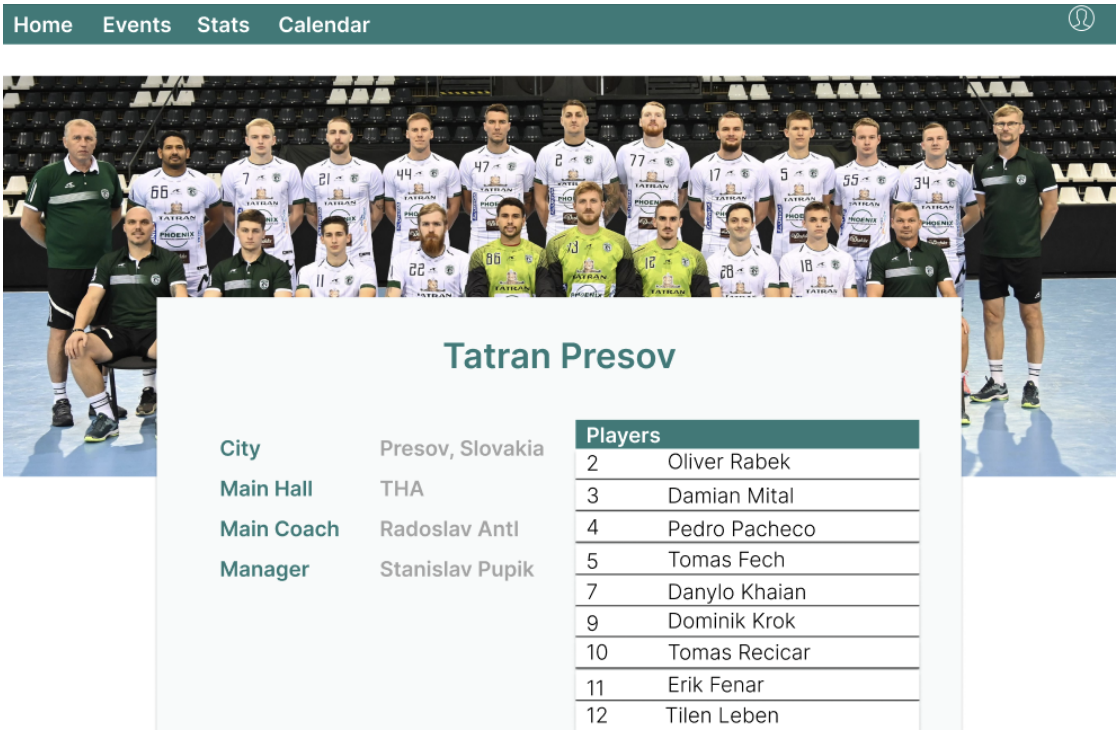



**Obr. B.1** Návrh prihlásenia

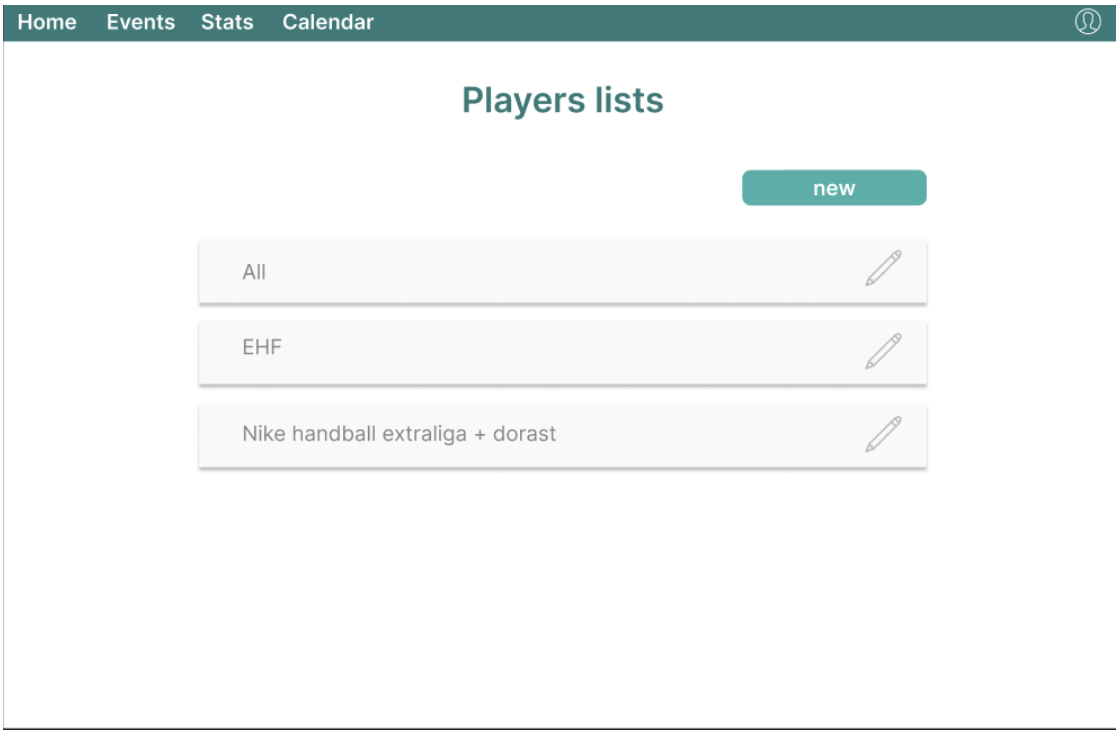



**Obr. B.2** Návrh registrácie

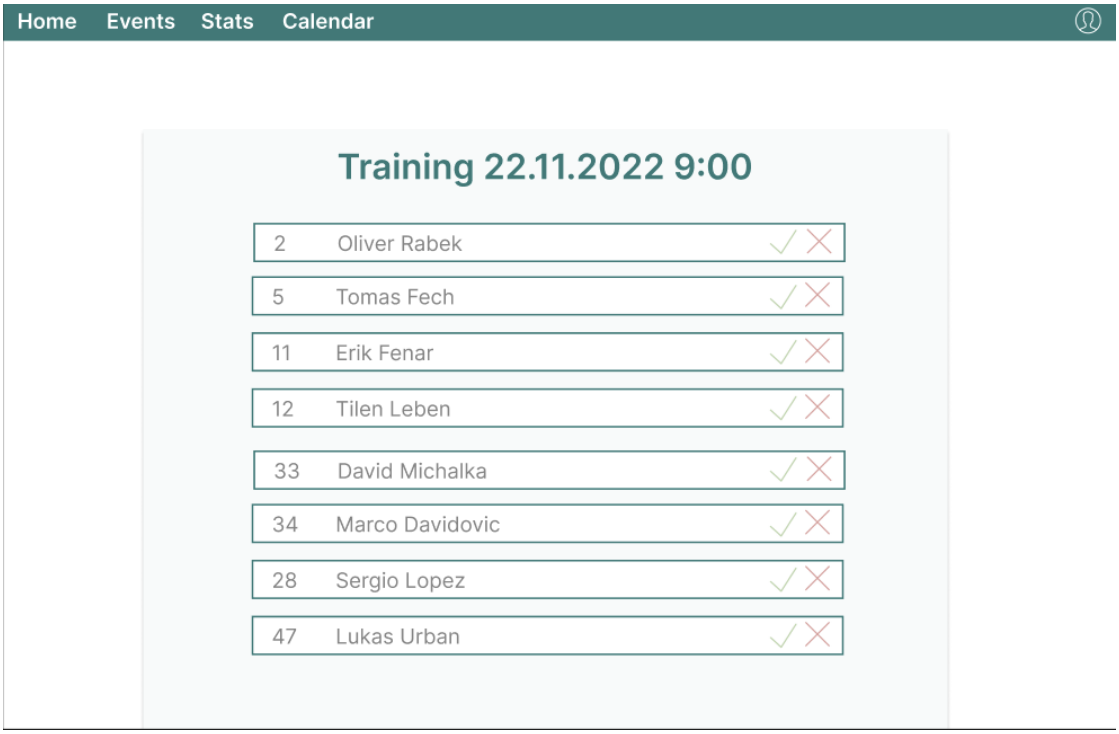

**Obr. B.3** Návrh profilu



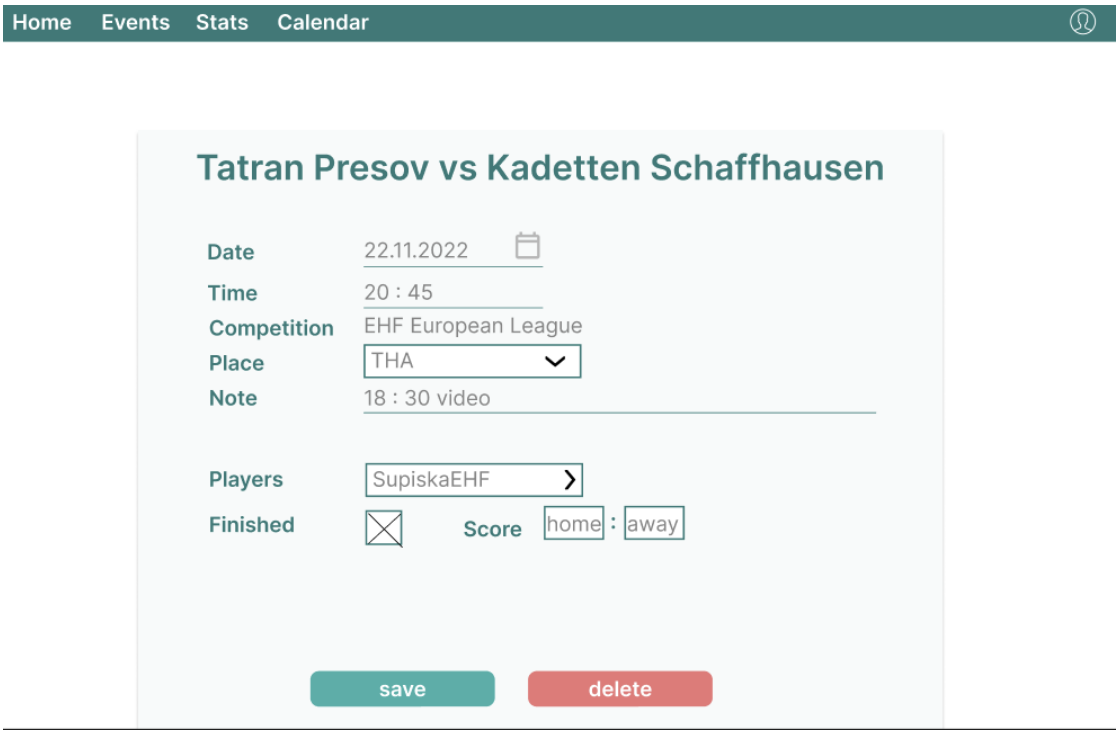

**Obr. B.4** Návrh tímu



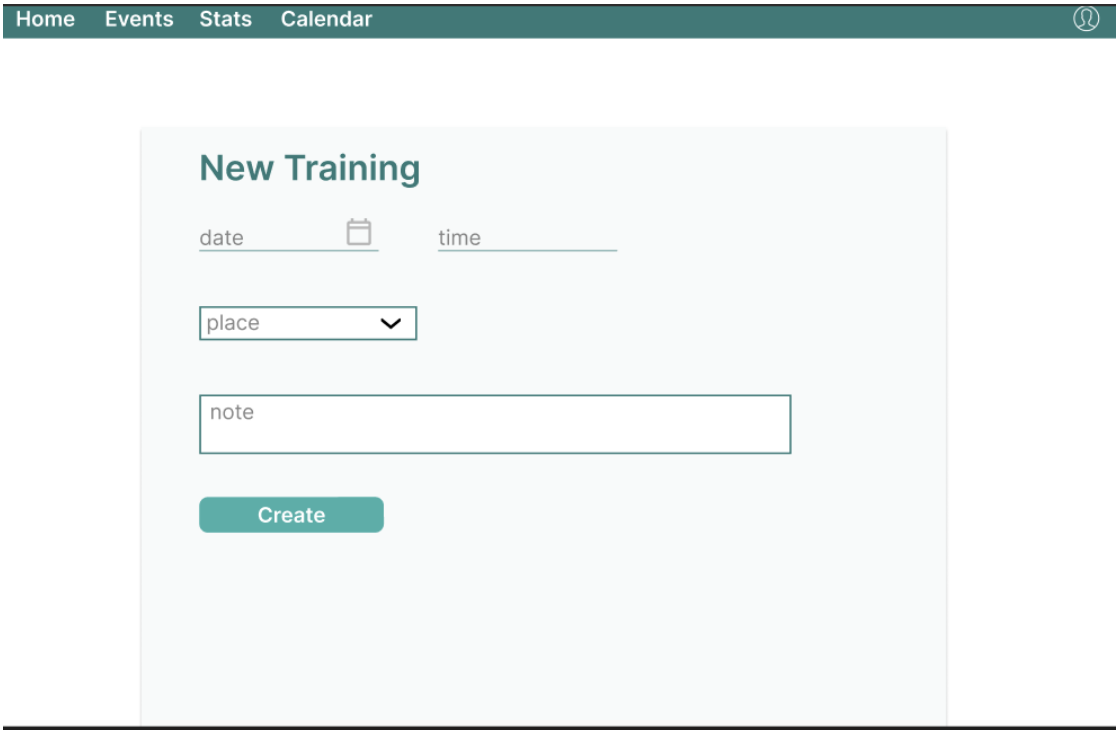

**Obr. B.5** Návrh súpisiek



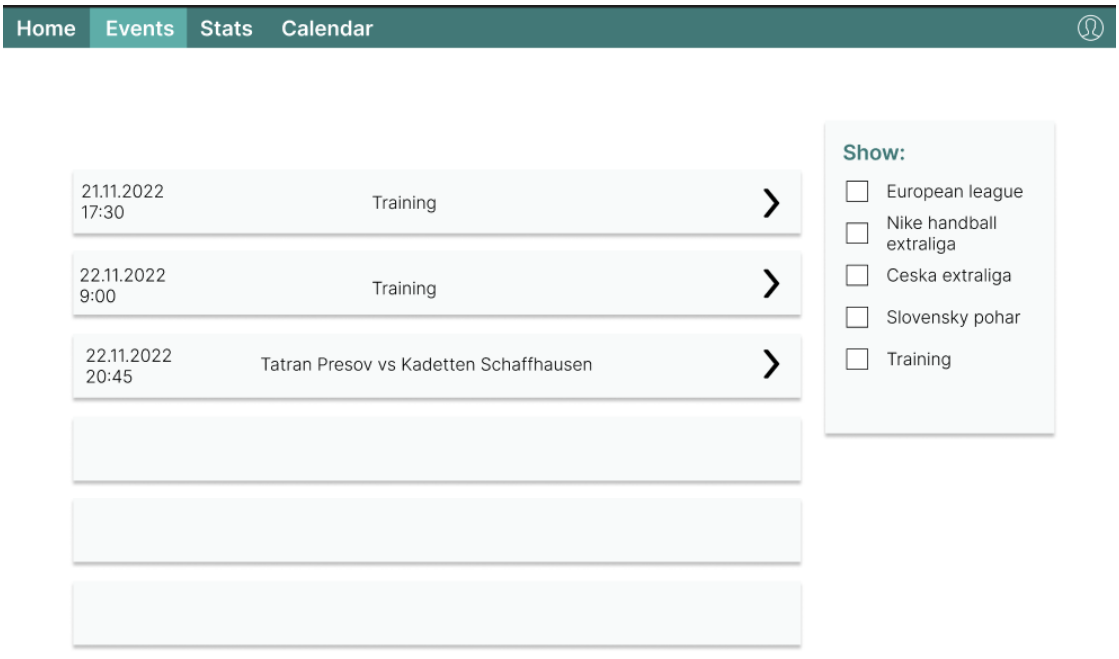

**Obr. B.6** Návrh dochádzky




**Obr. B.7** Návrh vytvorenia zápasu




**Obr. B.8** Návrh vytvorenia tréningu




**Obr. B.9** Návrh zoznamu udalostí


**Obr. B.10** Návrh detailu udalosti

# Dodatok C

# Dokumentácia API

| HTTP Metóda | URL | Popis |
|---|---|---|
| GET | api/team/{team_id}/training | získanie všetkých tréningov tímu |
| GET | api/team/{team_id}/training/{id} | získanie tréningu tímu |
| POST | api/team/{team_id}/training | vytvorenie tréningu pre tím |
| PUT | api/team/{team_id}/training/{id} | úprava tréningu |
| DELETE | api/team/{team_id}/training/{id} | odstránenie tréningu |

**Tabuľka C.1** Tréningy

| HTTP Metóda | URL | Popis |
|---|---|---|
| GET | api/team/{team_id}/players-list | získanie všetkých súpisiek tímu |
| GET | api/team/{team_id}/players-list/{id} | získanie súpisky tímu |
| POST | api/team/{team_id}/players-list | vytvorenie súpisky pre tím |
| PUT | api/team/{team_id}/players-list/{id} | úprava súpisky |
| DELETE | api/team/{team_id}/players-list/{id} | odstránenie súpisky |

**Tabuľka C.2** Súpisky

| HTTP Metóda | URL | Popis |
|---|---|---|
| GET | api/team/{team_id}/place | získanie všetkých športovísk tímu |
| GET | api/team/{team_id}/place/{id} | získanie športoviska tímu |
| POST | api/team/{team_id}/place | vytvorenie športoviska pre tím |
| PUT | api/team/{team_id}/place/{id} | úprava športoviska |
| DELETE | api/team/{team_id}/place/{id} | odstránenie športoviska |

**Tabuľka C.3** Športoviská

| HTTP Metóda | URL | Popis |
|---|---|---|
| GET | api/team/{team_id}/match | získanie všetkých zápasov tímu |
| GET | api/team/{team_id}/match/{id} | získanie zápasu tímu |
| POST | api/team/{team_id}/match | vytvorenie zápasu pre tím |
| PUT | api/team/{team_id}/match/{id} | úprava zápasu |
| DELETE | api/team/{team_id}/match/{id} | odstránenie zápasu |

**Tabuľka C.4** Zápas

| HTTP Metóda | URL | Popis |
|---|---|---|
| GET | api/team/{team_id}/event | získanie všetkých udalostí tímu |
| GET | api/team/{team_id}/event/{id} | získanie udalosti tímu |

**Tabuľka C.5** Udalosti

| HTTP Metóda | URL | Popis |
|---|---|---|
| GET | api/team | získanie všetkých tímov |
| GET | api/team/{team_id} | získanie tímu |
| POST | api/team | vytvorenie tímu |
| PUT | api/team/{team_id} | úprava tímu |
| DELETE | api/team/{team_id} | odstránenie tímu |
| GET | api/team/{team_id}/player/{player_id} | získanie hráčov tímu |

**Tabuľka C.6** Tím

| HTTP Metóda | URL | Popis |
|---|---|---|
| GET | api/player | získanie všetkých hráčov |
| GET | api/player/{player_id} | získanie hráča |
| POST | api/player | vytvorenie hráča |
| PUT | api/player/{player_id} | úprava hráča |
| DELETE | api/player/{player_id} | odstránenie hráča |

**Tabuľka C.7** Hráč

| HTTP Metóda | URL | Popis |
|---|---|---|
| GET | api/player | získanie všetkých manažérov |
| GET | api/manager/{manager_id} | získanie manažéra |
| POST | api/manager | vytvorenie manažéra |
| PUT | api/manager/{manager_id} | úprava manažéra |
| DELETE | api/manager/{manager_id} | odstránenie manažéra |

**Tabuľka C.8** Manažér

| HTTP Metóda | URL | Popis |
|---|---|---|
| GET | api/competition | získanie všetkých súťaží |
| GET | api/competition/{competition_id} | získanie súťaže |
| POST | api/competition | vytvorenie súťaže |
| PUT | api/competition/{competition_id} | úprava súťaže |
| DELETE | api/competition/{competition_id} | odstránenie súťaže |

**Tabuľka C.9** Súťaž

| HTTP Metóda | URL | Popis |
|---|---|---|
| GET | api/competition/{competition_id}/standings | získanie všetkých umiestnení súťaže |
| GET | api/competition/{competition_id}/standings/{standings_id} | získanie umiestnenia |
| POST | api/competition/{competition_id}/standings | vytvorenie umiestnenia |
| PUT | api/competition/{competition_id}/standings/{standings_id} | úprava umiestnenia |
| DELETE | api/competition/{competition_id}/standings/{standings_id} | odstránenie umiestenia |

**Tabuľka C.10** Umiestnenie

| HTTP Metóda | URL | Popis |
|---|---|---|
| GET | api/coach | získanie všetkých trénerov |
| GET | api/coach/{coach_id} | získanie trénera |
| POST | api/coach | vytvorenie trénera |
| PUT | api/coach/{coach_id} | úprava trénera |
| DELETE | api/coach/{coach_id} | odstránenie trénera |

**Tabuľka C.11** Tréner

| HTTP Metóda | URL | Popis |
|---|---|---|
| GET | api/stat-player | získanie všetkých štatistík |
| GET | api/stat-player/{stat_id} | získanie štatistiky |
| POST | api/stat-player | vytvorenie štatistiky |
| DELETE | api/stat-player/{stat_id} | odstránenie štatistiky |

**Tabuľka C.12** Štatistika hráča

| HTTP Metóda | URL | Popis |
|---|---|---|
| GET | api/stat-match | získanie všetkých štatistík |
| GET | api/stat-match/{stat_id} | získanie štatistiky |
| POST | api/stat-match | vytvorenie štatistiky |
| DELETE | api/stat-match/{stat_id} | odstránenie štatistiky |

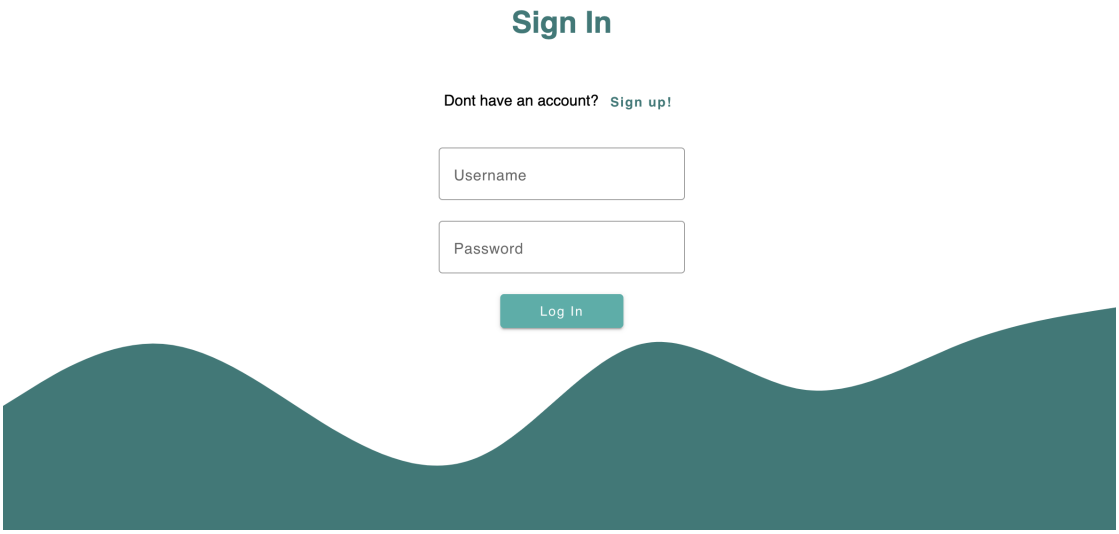
**Tabuľka C.13** Štatistika zápasov

| HTTP Metóda | URL | Popis |
|---|---|---|
| GET | api/user | získanie všetkých užívateľov |
| GET | api/user/{user_id} | získanie uživateľa |
| POST | api/user | vytvorenie užívateľa |
| PUT | api/user/{user_id} | úprava užívateľa |
| DELETE | api/user/{user_id} | odstránenie užívateľa |

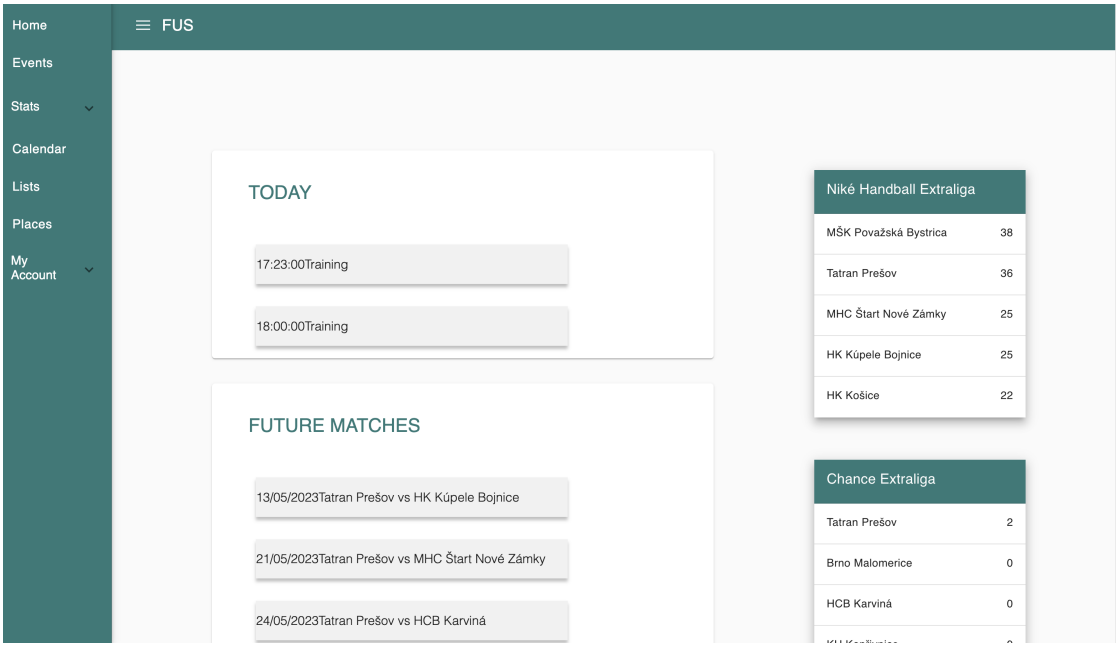
**Tabuľka C.14** Užívateľ

Dodatok D

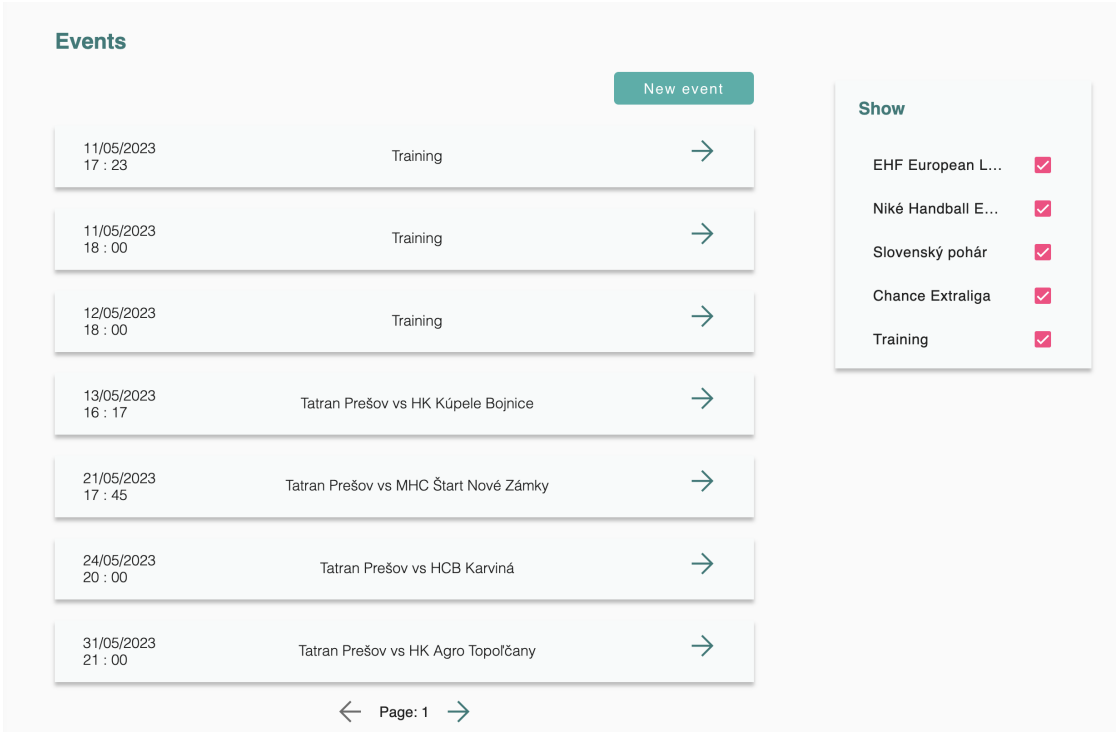
# Výsledná aplikácia



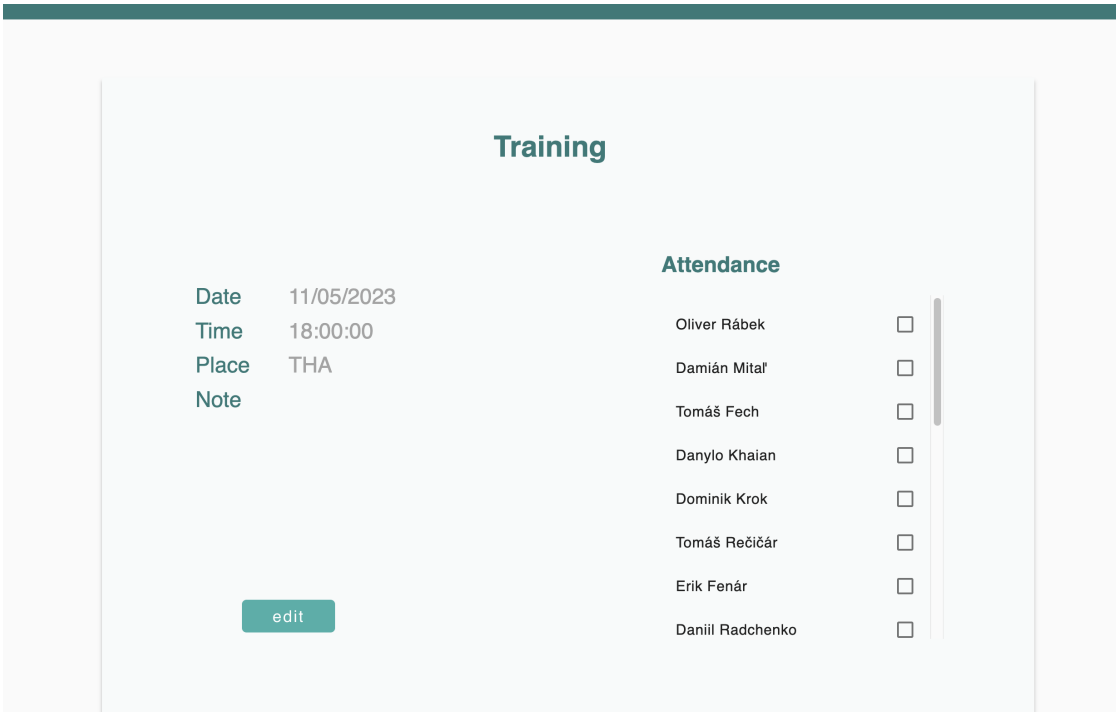

**Obr. D.1** Prihlásenie




**Obr. D.2** Úvodná obrazovka

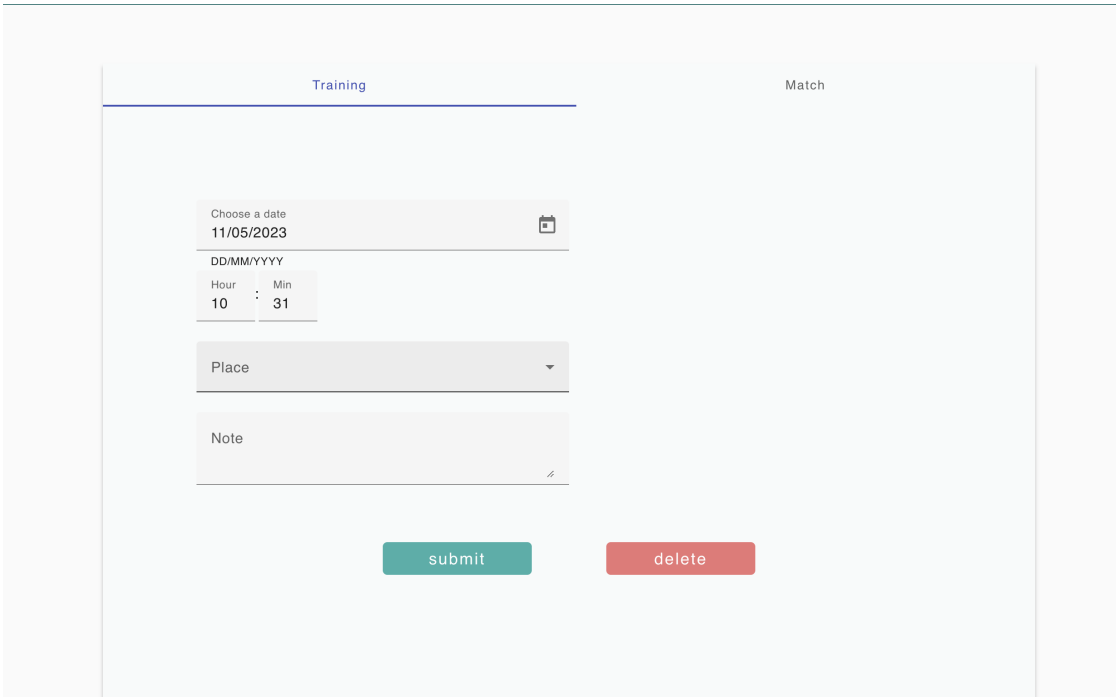



**Obr. D.3** Zobrazenie zoznamu udalostí

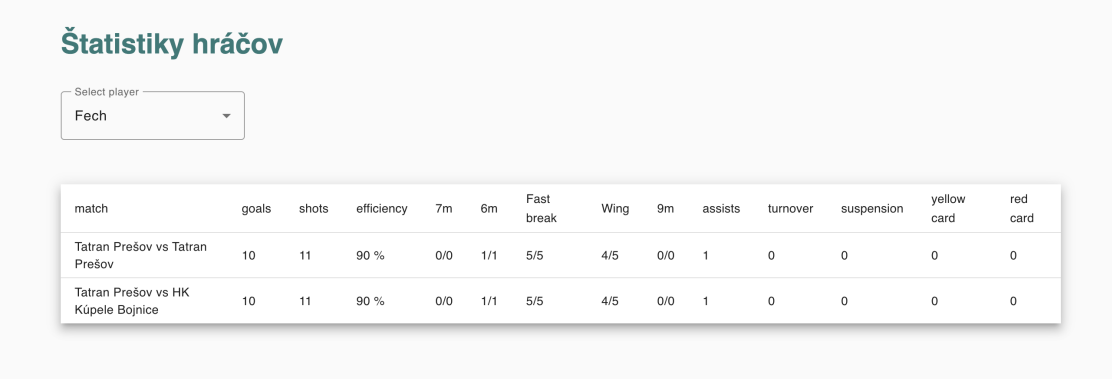



**Obr. D.4** Zobrazenie detailu tréningu

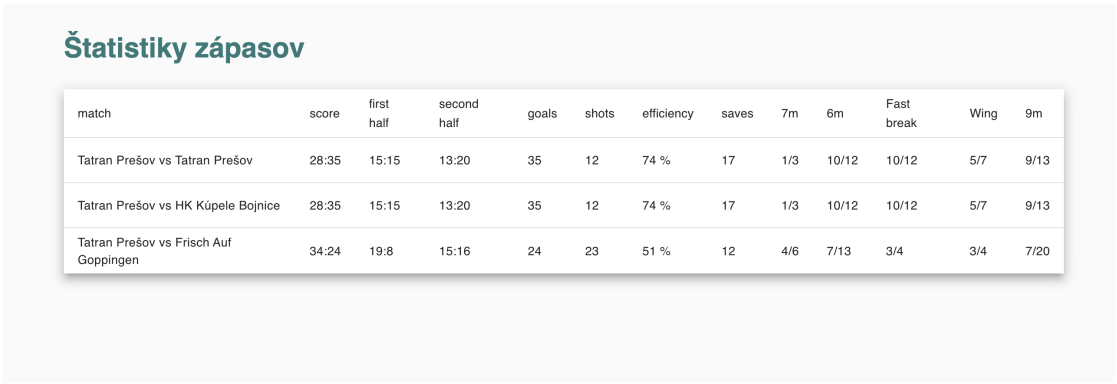

**Obr. D.5** Zobrazenie detailu zápasu




**Obr. D.6** Vytvorenie/úprava udalosti




**Obr. D.7** Zobrazenie štatistík hráča

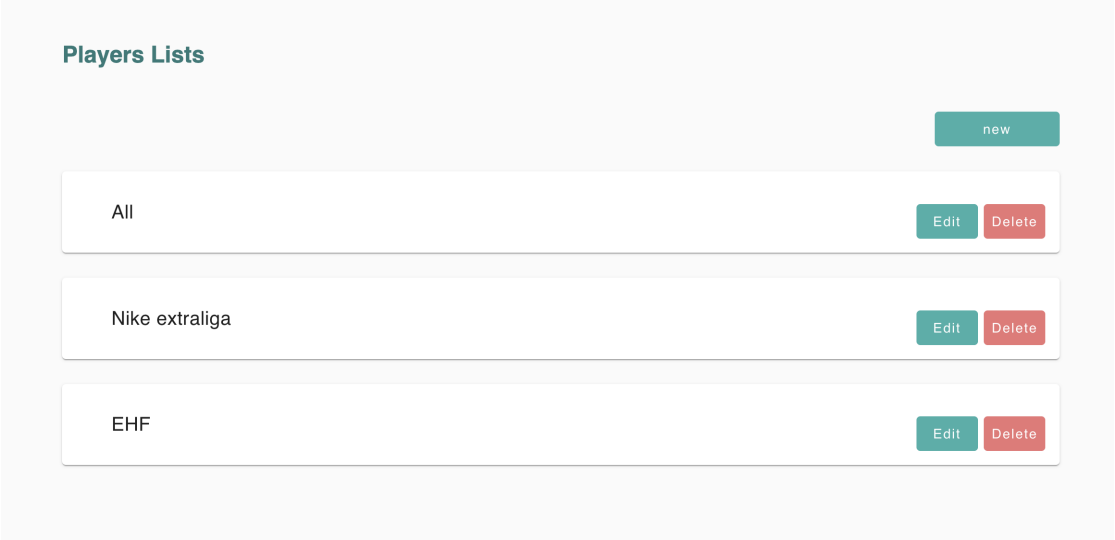


## Štatistiky zápasov

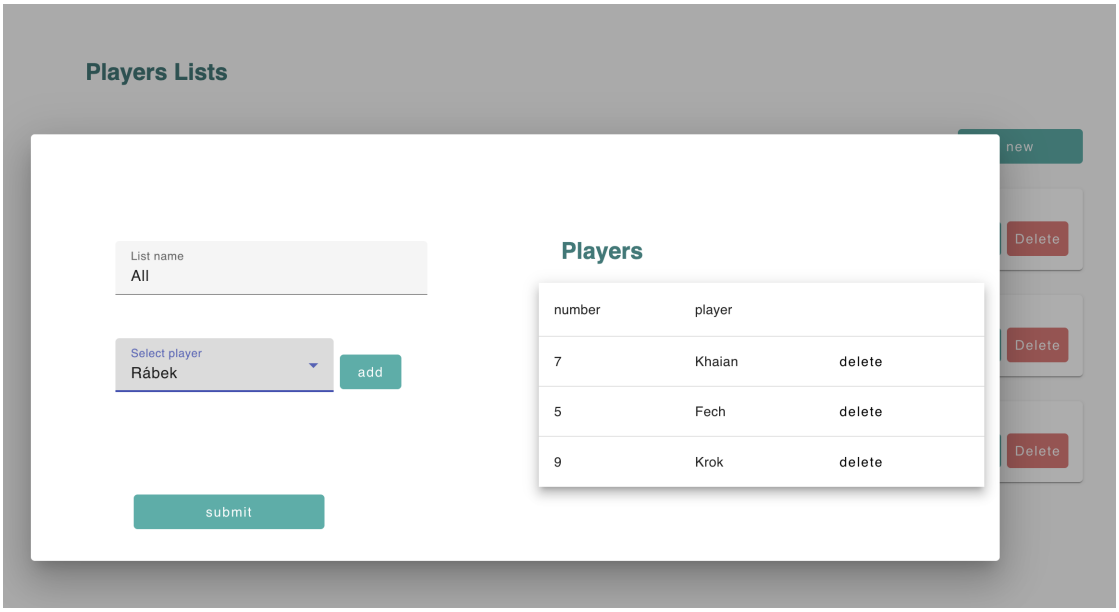
| match | score | first half | second half | goals | shots | efficiency | saves | 7m | 6m | Fast break | Wing | 9m |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Tatran Prešov vs Tatran Prešov | 28:35 | 15:15 | 13:20 | 35 | 12 | 74 % | 17 | 1/3 | 10/12 | 10/12 | 5/7 | 9/13 |
| Tatran Prešov vs HK Kúpele Bojnice | 28:35 | 15:15 | 13:20 | 35 | 12 | 74 % | 17 | 1/3 | 10/12 | 10/12 | 5/7 | 9/13 |
| Tatran Prešov vs Frisch Auf Goppingen | 34:24 | 19:8 | 15:16 | 24 | 23 | 51 % | 12 | 4/6 | 7/13 | 3/4 | 3/4 | 7/20 |

**Obr. D.8** Zobrazenie štatistík zápasov

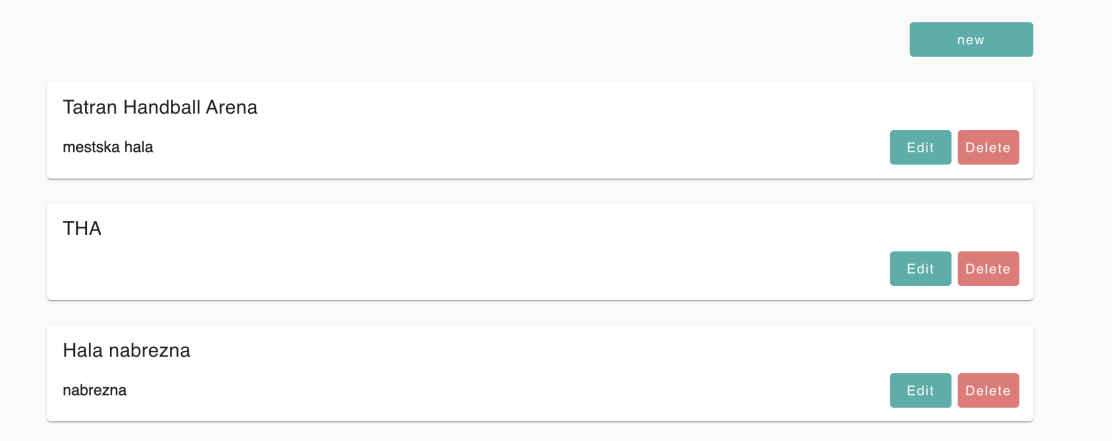

**Obr. D.9** Kalendár



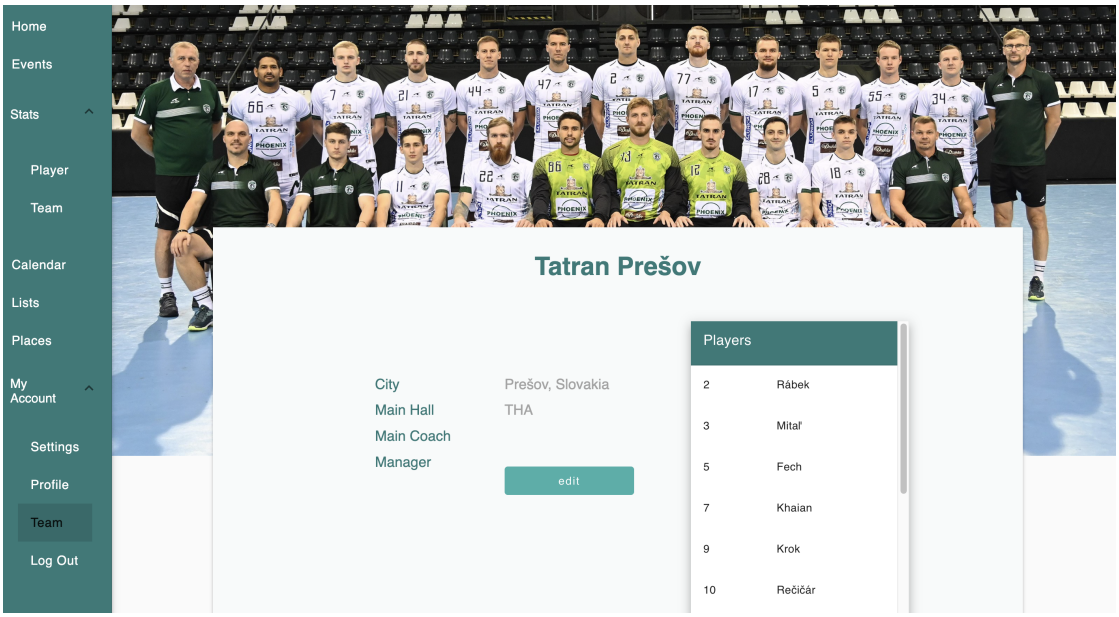

**Obr. D.10** Prezeranie súpisiek



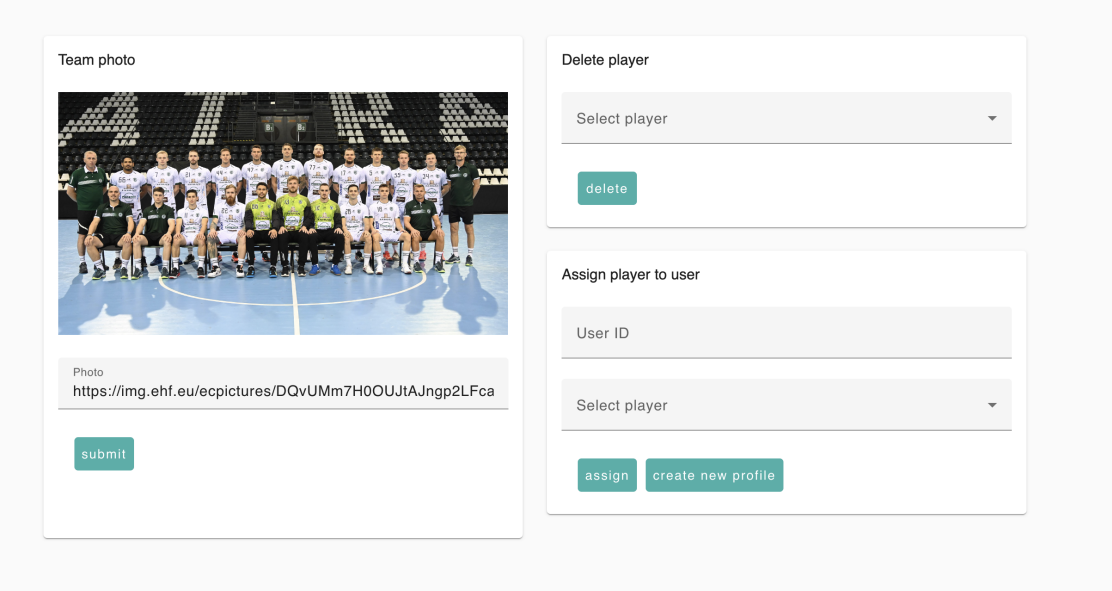

**Obr. D.11** Vytvorenie/úprava súpisky



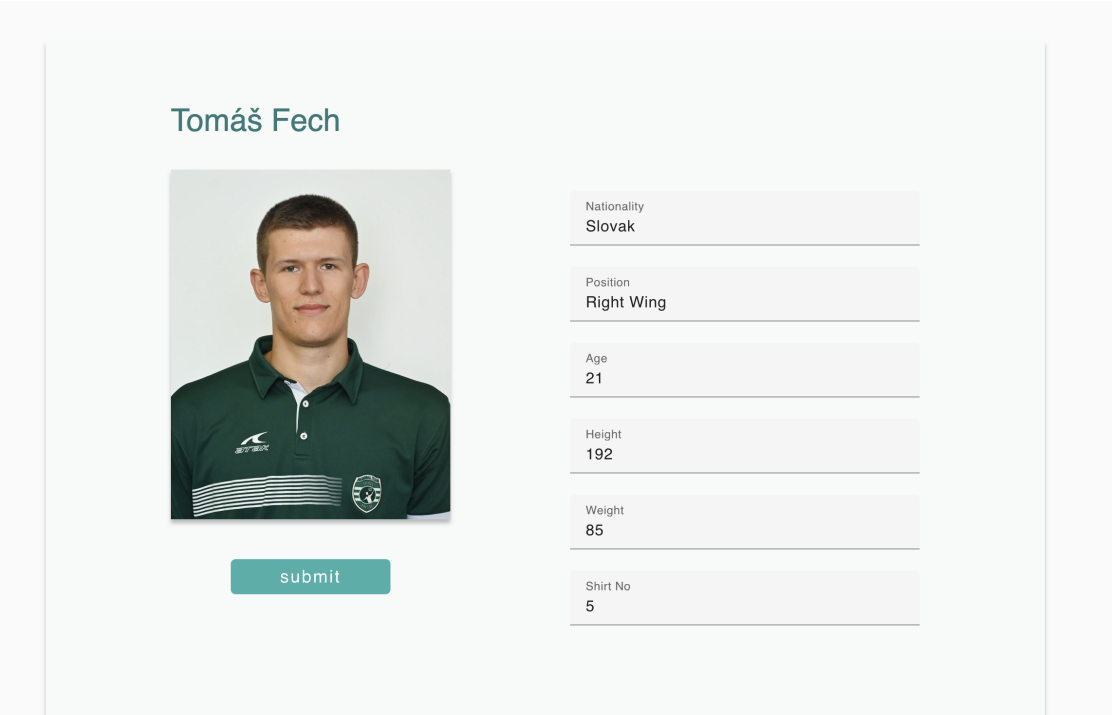

**Obr. D.12** Prezeranie športovísk




**Obr. D.13** Informácie o tíme




**Obr. D.14** Úprava tímu




**Obr. D.15** Úprava profilu

# Bibliografia


1. *What are functional requirements: Examples, definition, complete guide* [online]. 2023. [cit. 2023-04-15]. Dostupné z : https://visuresolutions.com/blog/functional-requirements/.
2. *What is web application (web apps) and its benefits* [online]. TechTarget, 2023 [cit. 2023-04-15]. Dostupné z : https://www.techtarget.com/searchsoftwarequality/definition/Web-application-Web-app.
3. *What Is A Web Application?* [Online]. 2023. [cit. 2023-04-15]. Dostupné z : https://www.stackpath.com/edge-academy/what-is-a-web-application.
4. HAT, Red. *What is an application architecture* [online]. 2020. [cit. 2023-04-15]. Dostupné z : https://www.redhat.com/en/topics/cloud-native-apps/what-is-an-application-architecture.
5. OLUWATOSIN, Haroon Shakirat. Client-server model. *IOSR Journal of Computer Engineering* [online]. 2014 [cit. 2023-04-15].
6. *What is Three-Tier Architecture — IBM* [online]. 2023. [cit. 2023-04-15]. Dostupné z : https://www.ibm.com/topics/three-tier-architecture.
7. IBM. *What is an Application Programming Interface (API)* [online]. 2023. [cit. 2023-04-16]. Dostupné z : https://www.ibm.com/topics/api.
8. HAT, Red. *REST vs. SOAP* [online]. 2019. [cit. 2023-04-16]. Dostupné z : https://www.redhat.com/en/topics/integration/whats-the-difference-between-soap-rest.
9. *An overview of HTTP - http: MDN* [online]. Mozilla, 2023 [cit. 2023-04-17]. Dostupné z : https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview.
10. *HTTP request methods - http: MDN* [online]. Mozilla, 2023 [cit. 2023-04-17]. Dostupné z : https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods.
11. *Idempotent - MDN Web Docs Glossary: Definitions of Web-related terms: MDN* [online]. Mozilla, 2023 [cit. 2023-04-17]. Dostupné z : https://developer.mozilla.org/en-US/docs/Glossary/Idempotent.
12. *What is a URL? - learn web development: MDN* [online]. 2023. [cit. 2023-04-17]. Dostupné z : https://developer.mozilla.org/en-US/docs/Learn/Common_questions/Web_mechanics/What_is_a_URL.
13. *Evolution of HTTP - http: MDN* [online]. 2023. [cit. 2023-04-17]. Dostupné z : https://developer.mozilla.org/en-US/docs/Web/HTTP/Basics_of_HTTP/Evolution_of_HTTP.
14. *HTTP headers - http: MDN* [online]. 2023. [cit. 2023-04-17]. Dostupné z : https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers.

15. *HTTP response status codes - HTTP: MDN* [online]. 2023. [cit. 2023-04-17]. Dostupné z : https://developer.mozilla.org/en-US/docs/Web/HTTP/Status.
16. *What is a rest api?* [Online]. 2023. [cit. 2023-04-17]. Dostupné z : https://www.ibm.com/topics/rest-apis.
17. *MVC - MDN Web Docs Glossary: Definitions of Web-related terms: MDN* [online]. Mozilla, 2023 [cit. 2023-04-16]. Dostupné z : https://developer.mozilla.org/en-US/docs/Glossary/MVC.
18. ČÁPKA, David. *MVC Architektura* [online]. 2023. [cit. 2023-04-16]. Dostupné z : https://www.itnetwork.cz/navrh/mvc-architektura-navrhovy-vzor.
19. SVIRCA, Zanfina. *Everything you need to know about MVC architecture* [online]. Towards Data Science, 2020 [cit. 2023-04-16]. Dostupné z : https://towardsdatascience.com/everything-you-need-to-know-about-mvc-architecture-3c827930b4c1.
20. MŮČKA, JAN. *Relační Databáze vs. Nerelační Databáze: čím Se Liší?* [Online]. 2022. [cit. 2023-04-17]. Dostupné z : https://www.master.cz/blog/relacni-databaze-nerelacni-databaze-jake-jsou-rozdily/.
21. CHURCHVILLE, Fred. *What is User Interface (UI)? definition from searchapparchitecture* [online]. TechTarget, 2021 [cit. 2023-05-09]. Dostupné z : https://www.techtarget.com/searchapparchitecture/definition/user-interface-UI.
22. ABBA, Ihechikara Vincent. *What is an ORM – the meaning of object relational mapping database tools* [online]. freeCodeCamp.org, 2022 [cit. 2023-04-18]. Dostupné z : https://www.freecodecamp.org/news/what-is-an-orm-the-meaning-of-object-relational-mapping-database-tools/.
23. *Spring Framework Overview* [online]. 2023. [cit. 2023-04-18]. Dostupné z : https://docs.spring.io/spring-framework/docs/current/reference/html/overview.html.
24. *Introducing Spring Boot* [online]. 2023. [cit. 2023-04-18]. Dostupné z : https://docs.spring.io/spring-boot/docs/current/reference/html/getting-started.html#getting-started.
25. *Web* [online]. 2023. [cit. 2023-04-18]. Dostupné z : https://docs.spring.io/spring-boot/docs/current/reference/html/web.html#web.
26. *Spring Data2022.0.5* [online]. 2023. [cit. 2023-04-18]. Dostupné z : https://spring.io/projects/spring-data.
27. *Data* [online]. 2023. [cit. 2023-04-18]. Dostupné z : https://docs.spring.io/spring-boot/docs/current/reference/html/data.html#data.
28. BAELDUNG, Written by: *Parsing HTML in Java with jsoup* [online]. 2023. [cit. 2023-04-25]. Dostupné z : https://www.baeldung.com/java-with-jsoup.
29. *What is a relational database?* [Online]. 2023. [cit. 2023-04-20]. Dostupné z : https://www.ibm.com/topics/relational-databases.
30. *About* [online]. 2023. [cit. 2023-04-20]. Dostupné z : https://www.postgresql.org/about/.
31. *OpenAPI specification* [online]. 2023. [cit. 2023-05-02]. Dostupné z : https://swagger.io/specification/.
32. NEOTERIC. *Single-page application vs. multiple-page application* [online]. Medium, 2018 [cit. 2023-04-22]. Dostupné z : https://medium.com/@NeotericEU/single-page-application-vs-multiple-page-application-2591588efe58.
33. ARDALIS. *Choose between traditional web apps and Single Page Apps* [online]. 2023. [cit. 2023-04-22]. Dostupné z : https://learn.microsoft.com/en-us/dotnet/architecture/modern-web-apps-azure/choose-between-traditional-web-and-single-page-apps#when-to-choose-spas.

34. NIHAR-RAVAL. *React vs angular: Which one is best for your next front-end project?* [Online]. Radixweb, 2023 [cit. 2023-04-22]. Dostupné z : https://radixweb.com/blog/react-vs-angular#react.
35. SYSTANGO. *Angular vs. Vue.js- a handy comparison guide* [online]. Medium, 2018 [cit. 2023-04-20]. Dostupné z : https://systango.medium.com/angular-vs-vue-js-a-handy-comparison-guide-8164ce1773e0.
36. *Learn hub* [online]. 2023. [cit. 2023-04-20]. Dostupné z : https://www.nobledesktop.com/learn/figma/what-is-figma.
37. REES, Damian. *What is wireframing* [online]. 2023. [cit. 2023-04-20]. Dostupné z : https://www.experienceux.co.uk/faqs/what-is-wireframing/.
38. *HTML: Hypertext markup language* [online]. 2023. [cit. 2023-04-22]. Dostupné z : https://developer.mozilla.org/en-US/docs/Web/HTML.
39. ČÁPKA, David. *Lekce 1 - Úvod DO HTML a váš první web* [online]. 2023. [cit. 2023-04-22]. Dostupné z : https://www.itnetwork.cz/html-css/webove-stranky/jak-psat-moderni-web-html-tutorial-uvod-do-html.
40. *HTML elements reference - HTML: Hypertext markup language: MDN* [online]. 2023. [cit. 2023-04-22]. Dostupné z : https://developer.mozilla.org/en-US/docs/Web/HTML/Element.
41. *CSS - MDN Web Docs Glossary: Definitions of Web-related terms: MDN* [online]. 2023. [cit. 2023-04-20]. Dostupné z : https://developer.mozilla.org/en-US/docs/Glossary/CSS.
42. *What is CSS? - learn web development: MDN* [online]. 2023. [cit. 2023-04-20]. Dostupné z : https://developer.mozilla.org/en-US/docs/Learn/CSS/First_steps/What_is_CSS.
43. NANCE, Christopher. *Typescript essentials*. Packt Publishing Ltd., 2014.
44. JANSEN, Remo H. *Learning typescript: Exploit the features of typescript to develop and maintain captivating web applications with ease*. Packt Publishing, 2015.
45. *What is Angular?* [Online]. 2022. [cit. 2023-04-20]. Dostupné z : https://angular.io/guide/what-is-angular.
46. *Angular components overview* [online]. 2022. [cit. 2023-04-20]. Dostupné z : https://angular.io/guide/component-overview.
47. *Understanding dependency injection* [online]. 2022. [cit. 2023-04-20]. Dostupné z : https://angular.io/guide/dependency-injection.
48. *Two-way binding* [online]. 2023. [cit. 2023-04-20]. Dostupné z : https://angular.io/guide/two-way-binding.
49. DUA, Ashwin. *Bootstrap vs angular: Which is your front-end library?: Turing* [online]. 2022. [cit. 2023-04-20]. Dostupné z : https://www.turing.com/blog/bootstrap-vs-angular-material-remote-software-developers/.
50. *Angular component* [online]. 2023. Dostupné tiež z: https://fullcalendar.io/docs/angular.
51. BAELDUNG, Written by: *The DTO pattern (data transfer object)* [online]. 2022. [cit. 2023-05-09]. Dostupné z : https://www.baeldung.com/java-dto-pattern.
52. CRUSOVEANU, Written by: Loredana. *Inversion of control and dependency injection with Spring* [online]. 2023. [cit. 2023-05-09]. Dostupné z : https://www.baeldung.com/inversion-control-and-dependency-injection-in-spring.
53. BAELDUNG, Written by: *Guide to spring @autowired* [online]. 2023. [cit. 2023-05-09]. Dostupné z : https://www.baeldung.com/spring-autowire.

54. MOZDEVNET. *Cross-origin resource sharing (CORS) - http: MDN* [online]. 2023. [cit. 2023-05-09]. Dostupné z : `https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS`.

55. *What is software testing and how does it work?* [Online]. 2023. [cit. 2023-05-02]. Dostupné z : `https://www.ibm.com/topics/software-testing`.

56. KOĎOUSKOVÁ, Barbora. *Jak Na Uživatelské testování Webu A Aplikací* [online]. 2021. [cit. 2023-05-02]. Dostupné z : `https://www.rascasone.com/cs/blog/uzivatelske-testovani-webu-aplikace`.

# Obsah priloženého média

- `readme.txt` ..... stručný popis obsahu média
- `src` ..... zdrojové kódy
  - `impl` ..... zdrojové kódy implementácie
  - `thesis` ..... zdrojová forma práce vo formáte LaTeX
- `text` ..... text práce
  - `thesis.pdf` ..... text práce vo formáte PDF