Web aplikace pro správu robotické soutěže

Bc. Eliška Obadalová

Diplomová práce
2024

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
Ústav informatiky a umělé inteligence

Akademický rok: 2023/2024

ZADÁNÍ DIPLOMOVÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Bc. Eliška Obadalová**
Osobní číslo:	**A22308**
Studijní program:	**N0613A140022 Informační technologie**
Specializace:	**Softwarové inženýrství**
Forma studia:	**Prezenční**
Téma práce:	**Web aplikace pro správu robotické soutěže**
Téma práce anglicky:	**Web Application for Robotic Competition Management**

Zásady pro vypracování

1. Proveďte průzkum možných vylepšení a nových funkcionalit web aplikace pro organizaci robotické soutěže.
2. Navrhněte a popište sadu nových funkcionalit, které jsou pro skutečnou použitelnost aplikace nejvíce potřebné.
3. Pro navrženou sadu funkcionalit popište jejich možnou implementaci na straně serveru, webové a popř. i mobilní aplikace.
4. Implementujte navržené funkcionality tak, aby výsledná aplikace byla funkční a použitelná.
5. Vytvořte implementační dokumentaci v takovém rozsahu, aby bylo v budoucnu možné aplikaci rozšiřovat o další funkcionality.

Forma zpracování diplomové práce: **tištěná/elektronická**

Seznam doporučené literatury:

1. WAGNER, Gerd a Mircea DIACONESCU. Web applications with javascript or java: volume 2: associations and class hierarchies. 1. vyd. Boston: De Gruyter Oldenbourg, 2021. Webapp. ISBN 978-3-11-050024-0.
2. FRAIN, Ben. Responsive Web Design with HTML5 and CSS: Develop future-proof responsive websites using the latest HTML5 and CSS techniques, 3rd Edition. B.m.: Packt Publishing, 2020. ISBN 978-1-83921-156-0.
3. BANKS, Alex a Eve PORCELLO. Learning React: modern patterns for developing React Apps. Second edition. Beijing Boston Farnham Sebastopol Tokyo: O'Reilly, 2020. ISBN 978-1-4920-5172-5.
4. WILKEN, Jeremy. Angular in Action. 1st edition. Shelter Island, NY: Manning, 2018. ISBN 978-1-61729-331-3.
5. BOHNER, Michael. Ionic Framework: Building mobile apps with Ionic Framework. B.m.: CreateSpace Independent Publishing Platform, 2016. ISBN 978-1-5227-5352-0.

Vedoucí diplomové práce: **Ing. Tomáš Dulík, Ph.D.**
Ústav informatiky a umělé inteligence

Datum zadání diplomové práce: **5. listopadu 2023**
Termín odevzdání diplomové práce: **13. května 2024**

UNIVERZITA TOMÁŠE BATI VE ZLÍNĚ
Fakulta aplikované informatiky
DĚKAN
L.S.

doc. Ing. Jiří Vojtěšek, Ph.D. v.r.
děkan

prof. Mgr. Roman Jašek, Ph.D., DBA v.r.
ředitel ústavu

Ve Zlíně dne 5. ledna 2024

Prohlašuji, že

- beru na vědomí, že odevzdáním diplomové práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb. o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších právních předpisů, bez ohledu na výsledek obhajoby;
- beru na vědomí, že diplomová práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk diplomové práce bude uložen v příruční knihovně Fakulty aplikované informatiky Univerzity Tomáše Bati ve Zlíně;
- byl/a jsem seznámen/a s tím, že na moji diplomovou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
- beru na vědomí, že podle § 60 odst. 1 autorského zákona má UTB ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- beru na vědomí, že podle § 60 odst. 2 a 3 autorského zákona mohu užít své dílo – diplomovou práci nebo poskytnout licenci k jejímu využití jen připouští-li tak licenční smlouva uzavřená mezi mnou a Univerzitou Tomáše Bati ve Zlíně s tím, že vyrovnání případného přiměřeného příspěvku na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše) bude rovněž předmětem této licenční smlouvy;
- beru na vědomí, že pokud bylo k vypracování diplomové práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tedy pouze k nekomerčnímu využití), nelze výsledky diplomové práce využít ke komerčním účelům;
- beru na vědomí, že pokud je výstupem diplomové práce jakýkoliv softwarový produkt, považují se za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se projekt skládá. Neodevzdání této součásti může být důvodem k neobhájení práce.

Prohlašuji,

- že jsem na diplomové práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků budu uveden jako spoluautor.
- že odevzdaná verze diplomové práce a verze elektronická nahraná do IS/STAG jsou totožné.

Ve Zlíně, dne …….

podpis studenta

ABSTRAKT

Tato práce se zabývá průzkumem stávající aplikace na správu robotické soutěže Robogames a jejího vylepšení. Tato soutěž se pravidelné koná na fakultě Univerzity Tomáše Bati na Fakultě Aplikované Informatiky. Navrhnuty a zdokumentovány byly nové funkcionality tak, aby bylo možné aplikaci použít pro organizace dané soutěže.

Klíčová slova: robotická soutěž, webová aplikace, organizace robotické soutěže, Robogames

ABSTRACT

This thesis explores the examination of the existing application for managing the Robogames robotic competition and its improvement. This competition is held regularly at the Faculty of Applied Informatics of Tomas Bata University. New functionalities were designed and documented so that the application could be used for the organization of the given competition.

Keywords: robotic competition, web application, organization of a robotic tournament, Robogames

Děkuji panu Ing. Tomáši Dulíkovi, Ph.D za odborné vedení této diplomové práce.
Prohlašuji, že odevzdaná verze diplomové práce a verze elektronická nahraná do IS/STAG jsou totožné.

OBSAH

ÚVOD

Tato diplomová práce se věnuje zdokonalení a finalizaci existující aplikace pro správu fakultní robotické soutěže Robogames, která se pravidelně koná na Fakultě Aplikované Informatiky Univerzity Tomáše Bati ve Zlíně. Soutěž, pořádaná jednou ročně, představuje platformu pro nadšence v oblasti konstrukce a programování robotů, nabízí jim prostor pro srovnání dovedností a získání cenných zkušeností pro další profesní růst. Robotika, rychle se rozvíjející disciplína, nalézá uplatnění ve vzdělávání, průmyslu i zábavním průmyslu, přičemž soutěže jako Robogames jsou zásadní pro podporu inovací v tomto sektoru.

Hlavní motivací pro vybrání tohoto tématu byla ambice poskytnout organizátorům soutěže Robogames komplexní nástroj pro zlepšení administrativních procesů spojených s organizací soutěže. Výzvy související s pořádáním soutěže jsou mnohotvárné a zahrnují úkoly od registrace účastníků, přes správu pravidel a sledování průběhu soutěže, až po evidenci výsledků a rozdělení soutěžících na stanoviště. Současná praxe organizace soutěže využívá různorodé aplikace a služby třetích stran, včetně Google formulářů pro registraci a Microsoft Excel pro vyhodnocení výsledků. Integrace těchto procesů do jednotné aplikace, která by automatizovala většinu administrativních úkonů, by přinesla značné zjednodušení a zvýšila by efektivitu správy soutěže.

Diplomová práce navazuje na bakalářskou práci Bc. Martina Krčmy [1], ve které byla navržena serverová část aplikace ve frameworku Spring v programovacím jazyce Java. Zaměřuje se na detailní průzkum této serverové implementace, na doplnění uživatelského rozhraní a optimalizaci serverové části s cílem umožnit rychlou a efektivní práci v systému.

V teoretické části práce je nejprve představen kontext robotických soutěží se zvláštním důrazem na Robogames a klíčové procesy soutěže, které jsou rozhodující pro definování funkcionalit a uživatelského rozhraní aplikace. Dále jsou analyzovány technologie použité v existující implementaci, zejména Java a framework Spring, a práce se dále zaměřuje na popis jednotlivých funkcionalit.

Praktická část obsahuje popis tvorby samotné aplikace – wireframy a poté samostatný postup tvorby uživatelského rozhraní a testování funkčnosti.

I. TEORETICKÁ ČÁST

1 ROBOTICKÉ SOUTĚŽE – TEORETICKÝ ZÁKLAD

Robotické soutěže představují místo pro nadšence robotiky, aby změřili své dovednosti v navrhování, konstrukci a programování robotů. Tyto soutěže nejsou prostým závoděním mechanických stvoření, ale jsou i platformou pro inovaci, vzdělávání a osobní rozvoj účastníků. Představují simulované výzvy, které mohou být v budoucnu aplikovány v reálném světě a často simulují nebo předvídají problémy, které bude potřeba řešit pomocí robotiky.

Cíle těchto soutěží jsou mnohostranné: podporují vědecký výzkum a vývoj v oblasti robotiky, poskytují platformu pro vzdělávání a spolupráci, a inspirují další generace technologů a inženýrů. Kromě toho, robotické soutěže přinášejí důležité společenské a kulturní přínosy, jako je zvyšování povědomí o technologických inovacích. [2]

1.1 Vzdělávací aspekty soutěží

Robotické soutěže poskytují studentům a nadšencům jedinečnou platformu pro učení se mimo tradiční učebnice a přednáškové sály. Jsou příležitostí pro praktické aplikace teoretických znalostí strojírenství, softwarového inženýrství a systémové integrace. Spolupráce na robotickém projektu může posílit také dovednosti v týmové práci, řešení problémů a inovativním myšlení. [2]

1.2 Přínosy pro výzkum

Robotické soutěže slouží jako testovací pole pro nové myšlenky a technologie. Soutěžící zde mohou experimentovat s nápady, novými materiály a svými vlastními designy robotů. Tyto nové myšlenky později mohou vést k rozvoji výzkumu v oblasti robotiky.

1.3 Společenský dopad

Soutěže zvyšují povědomí o významu a možnostech robotiky. Podporují technologickou gramotnost a zájem o STEM obory (vědu, technologie, inženýrství a matematiku). Pomáhají také pomoci změnit vnímání robotiky a umělé inteligence, když je společnost bude pozitivněji vnímat jako nástroje pro vzdělávání, usnadnění práce a inovaci pro zlepšení lidských životů.[2]

1.4 Robotické soutěže

V České republice a v Evropě je poměrně velký počet robotických soutěží. Zde je krátký seznam některých ze současných pořádaných soutěží.[2]

Tabulka 1- robotické soutěže

Název soutěže	Místo konání	Disciplíny
Robogames [4]	Fakulta aplikované informatiky, Zlín	Robosumo, Mini Robosumo, Sledovač čáry, Robot uklízeč
JedoBot [3]	SPŠ Jedovnice	Minisumo, Lego sumo, sumo, sledovač čáry, čárové bludiště, robokáry
Robotiáda [5]	VIDA! Science centru v Brně	Čára (Line follower), Sprint, Medvěd (Bear rescue) ,Freestyle, Sumo
Robotic Tournament [6]	Zespół Szkół Technicznych, Rybnik, Polsko	Sledovač čáry, Sumo, Lego Sumo, Mini Sumo, Robo drag race, Robo strong, Freestyle

2 ROBOGAMES

Hlavním zaměřením této práce je právě soutěž Robogames, pořádaná na Univerzitě Tomáše Bati. Tato část se bude zabývat popisem jednotlivých procesů organizace soutěže. Cílem zkoumání je rekapitulace a pochopení fungování organizace od jejího počátku až po konec, uvést všechny postupy a pravidla soutěží, což je nutné ke stanovení pozdější analýzy stávající aplikace a navrhnutí inovačních funkcionalit. Zdrojem informací je oficiální stránka Robogames [4] a několikaleté osobní zkušenosti s asistencí při samotné organizaci soutěže.

2.1 Organizační struktura Robogames

Soutěž se tradičně koná jednou ročně, zpravidla v lednu, což organizátorům umožňuje využít období po novém roce pro finální přípravy a zároveň poskytuje soutěžícím dostatečný čas na design a vývoj jejich robotických konstrukcí. Po příchodu na místo konání je od účastníků požadována online registrace prostřednictvím formuláře, který slouží nejen k potvrzení jejich přítomnosti, ale také jako platforma pro kontrolu a ověření jejich robotů, aby bylo zajištěno, že splňují všechna pravidla a specifikace pro dané disciplíny. Po registraci a kontrole robotů následuje samotná soutěž, kde se jednotlivé disciplíny jako robosumo, sledování čáry či robot uklízeč odehrávají souběžně v různých částech areálu fakulty, čímž je zajištěn plynulý a efektivní průběh celé akce. Organizátoři mají k dispozici přesně stanovený harmonogram, který je dynamicky upravován v reálném čase podle průběhu a potřeb jednotlivých disciplín. Po ukončení všech soutěžních klání přichází chvíle napětí a očekávání, kdy jsou vyhlašovány výsledky.

2.1.1 Organizační tým a jejich role

Organizační tým Robogames hraje klíčovou roli v celém procesu soutěže, od přípravné fáze až po samotné vyhlášení výsledků. Členové týmu jsou zodpovědní za celou řadu úkolů, včetně zapisování a vyhodnocování výsledků, což je zásadní pro transparentnost a spravedlnost soutěže.

Pro každou disciplínu jsou také vypracována specifická pravidla, jejichž dodržování je nezbytné pro legitimní průběh soutěže. Organizační tým se stará o to, aby byla tato pravidla všem účastníkům jasná a přístupná předem, aby se každý mohl řádně připravit a aby byla soutěž spravedlivá.

Během samotné soutěže je u každého stanoviště přítomen rozhodčí, jehož role je nepostradatelná. Rozhodčí má na starosti dohled nad dodržováním pravidel, což zahrnuje i technické aspekty soutěžních robotů a etiku soutěže. Zároveň rozhodčí zapisuje výsledky, které jsou následně předávány organizačnímu týmu pro vyhodnocení. V závěrečné fázi soutěže organizátoři vyhodnotí nejlepší výkony z každé disciplíny.

2.1.2 Příprava a logistika soutěže

Administrativní příprava je proces, který zahrnuje tvorbu a testování registračních formulářů, aby byly připraveny na snadnou a hladkou registraci účastníků. Důležitou součástí je i příprava sdílených elektronických tabulek, často v programu Microsoft Excel, které jsou navrženy tak, aby umožňovaly rozhodčím efektivní přístup k seznamům soutěžících a zároveň umožňovaly rychlé a transparentní zapisování výsledků jednotlivých disciplín.

Fyzická příprava zahrnuje promyšlené rozmístění soutěžních hřišť na vyhrazených plochách fakulty. Každé hřiště musí odpovídat specifickým potřebám jednotlivých disciplín – např. hřiště pro sledovače čáry musí být umístěny v místech s co nejmenší světelností, aby byl minimalizován vliv světla na senzory robotů, kteří pak kvůli tomuto světelnému vlivu mohou mít problémy se zvládnutím dráhy, a u hřišť pro disciplínu robosumo musí bát v okruhu hřiště naznačit milník, který během zápasu nemůže nikdo překročit, aby senzory robota nebyly ovlivněny okolními vlivy. Zároveň hřiště musí být uspořádané tak, aby diváci mohli soutěž pohodlně sledovat a účastníci měli dostatek prostoru pro manipulaci se svými roboty. Rovněž příprava cen, která zahrnuje nejen nákup či výrobu trofejí, ale i tisk diplomů.

2.2 Registrace účastníků

2.2.1 Proces registrace účastníků

Proces registrace účastníků na Robogames je navržen tak, aby zajišťoval dodržení všech

pravidel a požadavků soutěže. Tento proces se skládá ze dvou zásadních kroků.

V prvním kroku se účastník nebo tým účastníků registruje online prostřednictvím oficiální webové stránky soutěže, Robogames.utb.cz. Tento krok umožňuje organizátorům získat přehled o počtu a struktuře účastníků, což je nezbytné pro logistickou přípravu soutěže. Registrace zahrnuje vyplnění základních údajů, jako

jsou jména účastníků, datum narození a disciplíny, kterých se chtějí zúčastnit. Díky tomuto systému mohou organizátoři předem plánovat rozložení disciplín a připravit potřebnou infrastrukturu. Druhý krok procesu se odehrává přímo na místě konání soutěže, kde je pro účastníky připravena recepce. Zde účastníci potvrzují svou fyzickou přítomnost a jsou následně přesměrováni k další fázi, která zahrnuje kontrolu robotů. Tato kontrola je klíčová, neboť zajišťuje, že všichni soutěžící roboti splňují stanovené technické požadavky a pravidla pro danou disciplínu. Tím je zaručena spravedlnost soutěže a minimalizuje se riziko technických nesrovnalostí.
Tento dvoufázový proces registrace nejenže usnadňuje organizaci soutěže, ale také podporuje transparentnost a férovost. Účastníci jsou díky předchozí online registraci již seznámeni s pravidly a požadavky, zatímco kontrola robotů na místě slouží jako finální zabezpečení, že všechny soutěžící týmy startují z rovných pozic.

2.3 Pravidla pro akceptování robota do disciplín

V rámci soutěže Robogames jsou účastníci na základě informací o datu narození, uvedených v přihlášce, kategorizováni do dvou věkových skupin.
První skupinou jsou žáci ve věku do 15 let včetně, zatímco druhou tvoří studenti a dospělí starší 15 let.

2.3.1 Materiály robotů

Co se týče materiálů použitých na výrobu robotů, Robogames otevírají dveře kreativitě tím, že vedle tradičních LEGO stavebnic povzbuzují používání jakýchkoli dostupných materiálů. Tímto přístupem soutěž aspiruje na podporu všech nadšenců do robotiky, a to bez ohledu na finanční prostředky dostupné pro nákup "oficiálních" LEGO stavebnic. soutěž nabízí unikátní příležitost nechat se inspirovat a obohatit své konstrukce o silnější "neoficiální" díly. [4]

2.3.2 Soutěžící a týmy

S ohledem na soutěžní týmy, Robogames umožňují, aby jeden robot byl zastoupen vícečlenným týmem, avšak v případě vítězství je udělena pouze jedna hmotná cena. Další pravidlo stanovuje, že každý tým může do jednotlivých disciplín přihlásit

pouze jednoho robota. Nicméně, pokud se tým rozhodne účastnit se více disciplín, je možné pro každou z
nich přihlásit odlišného robota. [4]

2.3.3 Obecná kritéria a pravidla soutěže

Před samotným startem Robogames bude sestaven harmonogram startů, jehož dodržování je pro účastníky zásadní. Nerespektování tohoto rozpisu může vést k sankcím, včetně možnosti diskvalifikace z celé soutěže. Každý soutěžící je povinen reagovat na výzvu rozhodčího a přistoupit k soutěži ve stanoveném čase. Výjimky jsou povoleny v případech, kdy je nezbytné provést opravy robotů nebo vyměnit baterie.[4]

Pro každou disciplínu je předem určen soubor pravidel, která musí soutěžící a jeho robot splňovat. Popsány jsou v následující pasáži. [odstavec 2.4]

2.4 Jednotlivé disciplíny Robogames

Jak už bylo řečeno, každá disciplína se dělí na dvě věkové kategorie:

- Pod 15 let (včetně)
- Nad 15 let

2.4.1 Popis disciplíny Robosumo

2.4.1.1 Pravidla

- Váha robota do 1 kg
- Zápas nesmí překročit 9 minut – délka jednoho kola jsou 3 minuty
- Rozměr robota musí být 20x20 cm, při spuštění však může svou velikost zvětšit

2.4.1.2 Průběh

Disciplína robosumo zahrnuje souboj dvou robotů, kteří se snaží navzájem vytlačit ze hřiště.

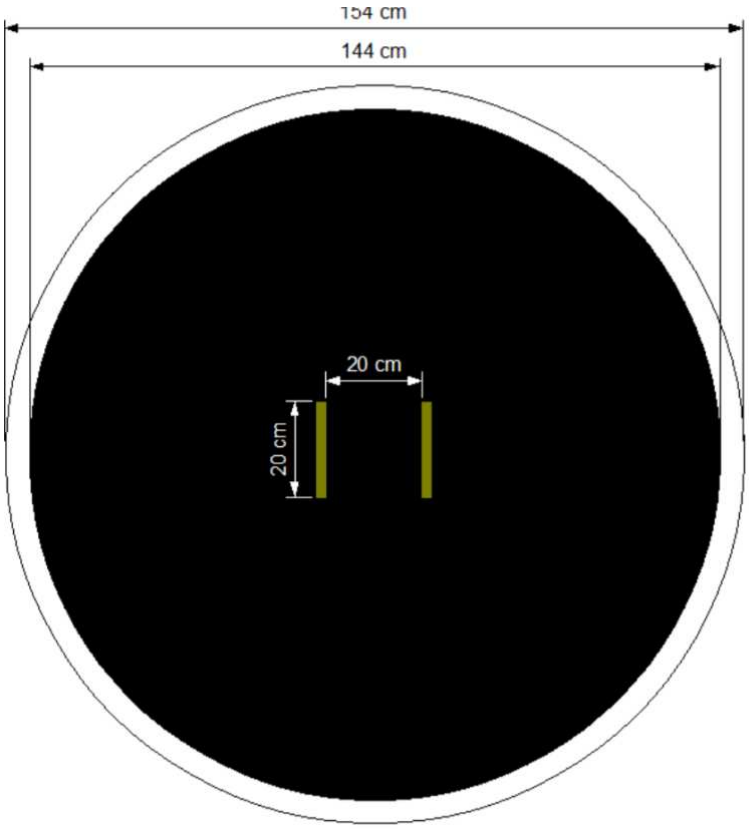
Hřiště je kulatého tvaru o průměru 154 cm. Hraje se na maximálně tři kola a na dva vítězné body – tedy kdo první dosáhne 2 bodů, vítězí daný zápas.

Na začátku každý soutěžící položí robota na vyznačené místo na hřišti „zády“ od sebe a na povel rozhodčího spustí robota. Robot nesmí vykazovat žádný pohyb po

dobu 5 sekund, poté se může otočit a zápas začíná. Robot prohrává v okamžiku, kdy se jakákoliv jeho část dotkne okolí mimo hřiště. To zahrnuje i části, které během zápasu mohly robotovi odpadnout.

Seznam soutěžících se před zahájením rozdělí mezi počty dostupných hřišť a organizátor má za úkol vytvořit přesnou strukturu, kdo proti sobě nastoupí - tzv. pavouka. Vítězové z jednotlivých hřišť se poté zúčastní finále proti sobě, které definitivně rozhodne o výherci.

[4]



Obrázek 1 - Hřiště robosumo

2.4.1.3 Varianta Minisumo

V této kategorii robotů je také možné soutěžit s roboty s menší velikostí, a to:

- Váha maximálně 0,5g
- Rozměry nesmí přesáhnout 10 x 10 cm
- Průměr hřiště 77 cm

Ostatní pravidla jsou shodná s disciplínou Robosumo. [4]

2.4.2 Popis disciplíny Sledování čáry

2.4.2.1 Pravidla

- Váha robota do 1 kg
- Čas na průjezd drahou – 3 minuty
- Rozměry robota maximálně 25 x 25 cm s neomezenou výškou

2.4.2.2 Sledovač čáry – rozdělení

Disciplína sledování čáry je rozdělena nejen podle věku účastníků, ale také na základě počtu senzorů, které robot využívá. Před každým startem je nutné, aby rozhodčí ověřil, zda robot odpovídá kategorii, do které byl přihlášen. Kategorie jsou definovány následovně:

- Sledovač čáry s jedním senzorem

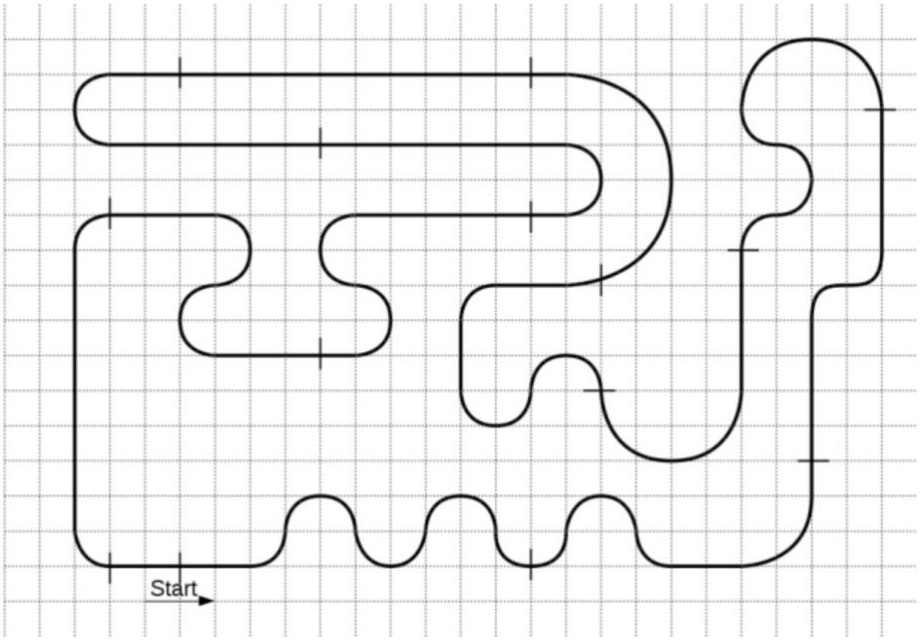
- Sledovač čáry s více senzory

2.4.2.3 Průběh

Soutěž ve sledování čáry vyžaduje, aby účastníci s jejich autonomními roboty úspěšně absolvovali dva různé okruhy. Ty jsou vyznačeny černou čarou na bílém podkladu hřiště. Robot má za úkol projet dráhy a hlavním kritériem pro umístění je rychlost projetí dané dráhy.

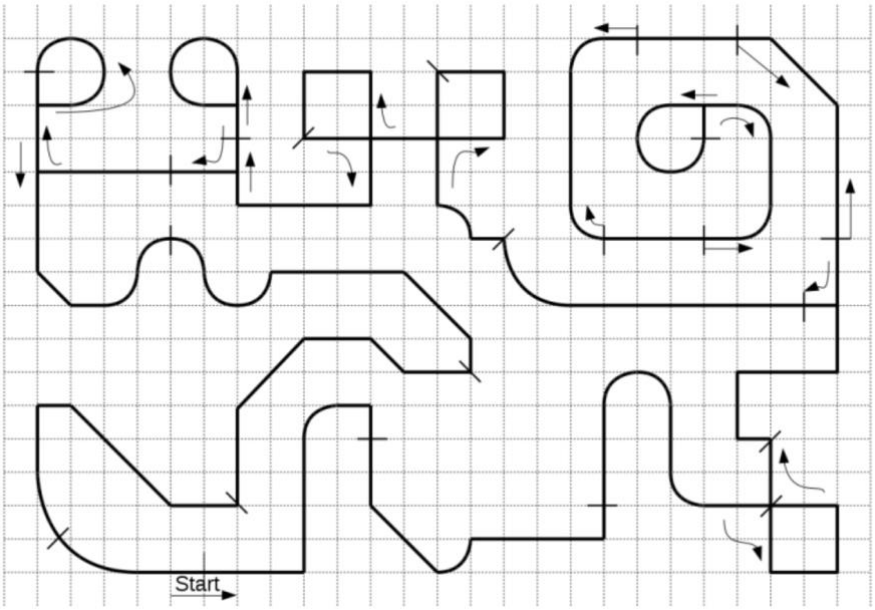
Pokud robot úspěšně dokončí celou trať, do výsledkové listiny se zaznamená čas, který na to potřeboval. V situaci, kdy robot nedokončí trať v celku, je zaznamenán nejvyšší dosažený checkpoint. Tyto checkpointy jsou na mapě trati jasně označeny a jsou seřazeny v pořadí od 1 až do N. Rozhodčí pak do výsledků uvede číslo posledního checkpointu, který robot zvládl úspěšně projít.

První z těchto okruhů je často označován jako "snadný", jelikož většina soutěžních robotů je schopna úspěšně dokončit jeho trasu. Vizuální reprezentace této dráhy je k dispozici v následujícím obrázku.



Obrázek 2 - sledovač čáry (snadná obtížnost)

Druhý okruh soutěže, často označovaný jako "složitý", představuje pro účastníky výrazně větší výzvu kvůli přítomnosti ostrých zatáček a míst, kde se trasy kříží. Tato komplikovaná konfigurace dráhy klade vyšší nároky na algoritmické schopnosti robotů, které musejí pro navigaci používat sofistikovanější metody zpracování dat ze senzorů. Vzhledem k těmto náročným požadavkům je počet soutěžících, jimž se podaří úspěšně dokončit celou dráhu, obvykle velmi malý. Proto je v této části soutěže kladen důraz zejména na zaznamenávání checkpointů, které slouží jako mezníky pro hodnocení dojezdu robotů, a to především v případech, kdy celá trasa není dokončena. [4]



Obrázek 3 - sledovač čáry (složitá obtížnost)

2.4.3 Popis disciplíny Robot uklízeč

2.4.3.1 Pravidla

- Doba trvání každého pokusu: omezená na tři minuty
- Omezení hmotnosti robota: neexistuje
- Limit rozměrů robota: základna do velikosti 25×25 cm, bez omezení výšky
- Velikost soutěžního pole: 140×100 cm

2.4.3.2 Průběh

Každý soutěžní robot má možnost třikrát se pokusit, přičemž každý pokus je časově omezen na maximálně tři minuty, během kterých musí přesunout co nejvíce krychlí do specifikované zóny.

V soutěži roboti navigují po ploše o rozměrech 140×100 cm, která je vyrobena z bílého panelu Sololak ohraničeného 30 mm širokou černou izolační páskou. Na této ploše jsou rozmístěny šest krychlí odpovídajících rozměrům kostek Lego Duplo. Zóna, do které musí robot kostky přesunout je ve tvaru kruhové výseče s poloměrem 30 cm a nachází se v rohu soutěžního pole. Startovní pozice robota je umístěna úhlopříčně od cílové

zóny, přičemž jeho orientace není předem určena. Uvnitř cílové zóny je umístěna alespoň 30 cm vysoká skříňka pod úhlem 45° vůči okrajům hrací plochy a 10 cm od

ní. Krychle jsou na ploše umístěny náhodně, ale jejich poloha je pro všechny soutěžící uniformní. Pokud robot neshromáždí všechny krychle během tříminutového limitu, za každou nesesbíranou krychli se k času připočte jedna minuta. Soutěžící mají právo kdykoliv svůj pokus ukončit, což následně oznámí rozhodčímu, jenž zaznamená jak čas, tak počet úspěšně přesunutých krychlí. Vítězem se stává robot, který dokončí úkol v nejkratším čase. [4]

Obrázek 4 - robot uklízeč – hřiště

2.4.3.3 Variace 1

- Robot má za úkol krychle pouze přemístit do definované zóny
- Krychle se počítá, pokud se po skončení pokusu nachází alespoň částečně uvnitř cílové zóny. Krychle, které robot přemístí a následně je vystrčí zpět, nebudou započítány

2.4.3.4 Variace 2

- Robot má za úkol krychle vhodit do boxu
- V rohu cílové oblasti je umístěn box, do kterého se krychle vhazují, namísto krabice
- Box má rozměry 210×210×100 mm s zadní stranou vysokou 300 mm

- Krychle jsou započítány pouze v případě, že zůstanou uvnitř boxu po skončení pokusu

2.5 Vyhodnocování výsledků

2.5.1 Systém bodování a stanovování vítězů

Pro zápis výsledků se používá Microsoft Excel a připravená tabulka v něm, která obsahuje seznam hráčů dané disciplíny. U robosumo obsahuje tabulku, která znázorňuje jednotlivé duely soutěžících a zapisuje se vítěz duelu, u sledování čáry a robota uklízeče se zaznamenává nejkratší čas, za který byl robot schopný úkol dokončit, popřípadě checkpoint, kterého dosáhl.

Rozhodčí mohou zapisovat výsledky paralelně do jednoho souboru, což umožňuje rychlejší vyhodnocení.

2.5.2 Proces oznámení výsledků a předávání cen

Jako závěr celého procesu organizace soutěže je vyhodnocení vítězů. V rámci tohoto kroku je potřeba ze strany organizace vytvořit diplomy a připravit ceny pro první tři místa pro každou disciplínu a její podkategorie.

3 ANALÝZA EXISTUJÍCÍ APLIKACE

Cílem následující části bude popsat stávající serverovou implementaci aplikace pro Robogames a její architekturu a strukturu, jehož autorem je Bc. Martin Krčma a tento projekt byla úkolem jeho bakalářské práce. Z poznatků analýzy poté budou učiněny závěry, jak danou aplikaci rozšířit a připravit na reálné použití.

3.1 Úvod do architektury projektu

Zaměříme se na architekturu serverové části stávajícího projektu, který je založen na frameworku Spring Boot. Spring Boot představuje nadstavbu tradičního frameworku Spring a poskytuje významné výhody, včetně integrovaného serveru, který umožňuje snadné spuštění a nasazení aplikací. [1] [7]

Jedním z aspektů této aplikace je využití MariaDB, což je populární open-source relační databázový systém, který slouží k ukládání a správě dat. [1]

Aplikace je navržena jako RESTful, což znamená, že komunikuje skrze definované API rozhraní a zpracovává HTTP požadavky klientů. Tento přístup zajišťuje vysokou míru modularity a umožňuje snadné rozšiřování funkcionalit aplikace. Aplikace je strukturovaná do více vrstev, čímž je zajištěno čisté oddělení prezentační logiky od aplikační business logiky, což napomáhá udržet kód přehledný a udržitelný. [1] [9]

Kromě toho aplikace využívá knihovnu Lombok, která slouží k automatizaci tvorby standardních metod, jako jsou gettery a settery, což významně redukuje množství boilerplate kódu a přispívá k vyšší efektivitě vývoje. Aplikace se spoléhá výhradně na knihovny Java Development Kitu (JDK), což zjednodušuje správu závislostí a zaručuje stabilitu aplikace. Výsledkem je serverová aplikace, která je dobře připravená na budoucí rozšíření a integrace.

[1] [10]

Následovat bude popis jednotlivých použitých technologií.

3.2 Použité technologie

3.2.1 Java a framework Spring Boot

Jádrem aplikace je programovací jazyk Java ve spojení s frameworkem Spring Boot, který je oblíbený pro svou efektivitu a vstřícnost k rychlému vývoji. Spring

Boot, poskytující řadu automatizovaných konfigurací, výrazně usnadňuje vývoj aplikací tím, že minimalizuje potřebu tradičních nastavení, která jsou jinak běžně nutná v Spring Frameworku [8]

3.2.1.1 Spring Framework a Spring Boot

Spring Framework je široce uznávaná platforma pro vývoj webových aplikací v jazyce Java, jejíž počátky sahají do roku 2002, kdy ji vytvořil Rod Johnson. Oficiální verze byla uvedena na trh v březnu 2004 a od té doby se stala klíčovým nástrojem pro mnoho vývojářů. Cílem Spring Frameworku je usnadnění a zefektivnění vývoje webových aplikací tím, že poskytuje komplexní podporu pro řešení běžných úkolů, jako jsou vazba dat, konverze typů, validace, ošetřování výjimek, správa zdrojů a událostí. Framework je zvláště ceněn pro svou podporu Dependency Injection, což je návrhový vzor, který značně zjednodušuje konfiguraci a integraci komponent aplikace. [7]

Spring Framework se skládá z několika modulů, z nichž každý slouží specifickému účelu:

- **Data access/Integration**: Tento modul usnadňuje práci s databázemi a integraci dat, nabízí podporu pro transakce.
- **Web (MVC/Remoting):** Modul Web poskytuje nástroje pro vývoj webových aplikací podle modelu MVC (Model-View-Controller) a podporuje vzdálenou komunikaci.
- **AOP (Aspect-oriented programming):** Umožňuje oddělit průřezové zájmy od hlavního kódu aplikace, čímž zvyšuje modularitu a usnadňuje údržbu.
- **Aspects**: Související s AOP, tento modul umožňuje definování a aplikaci aspektů.
- **Instrumentation**: Podporuje monitorování a správu aplikací v běhovém prostředí.
- **Core container:** Jádro Spring Frameworku, které řeší konfiguraci a správu beanů, včetně Dependency Injection.
- **Test:** Nabízí možnosti pro testování Spring aplikací, včetně mock objektů a testovacího kontextu. [8] [12]

Spring Boot, rozšíření Spring Frameworku, řeší jednu z největších výzev – potřebu

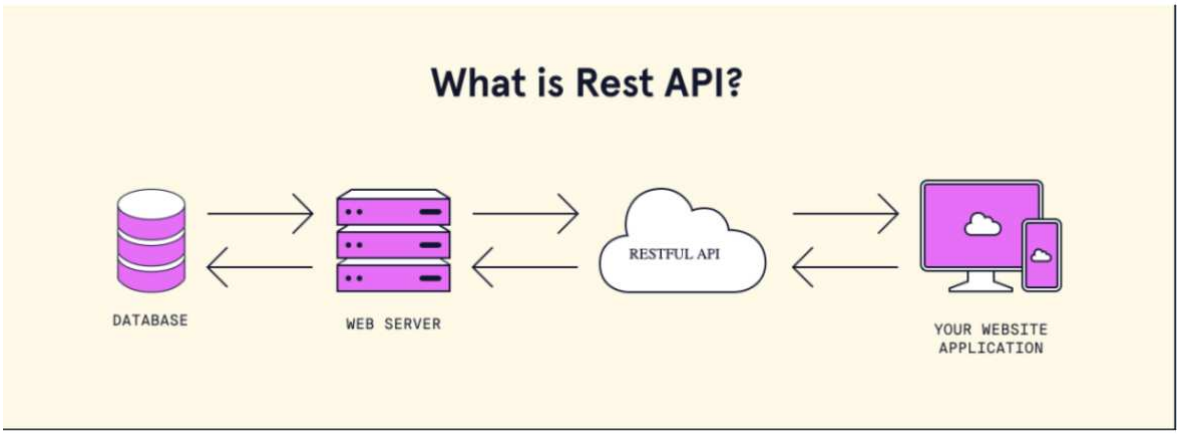
manuálního nastavení Servlet kontejneru. Díky zabudovanému Tomcat kontejneru, Spring Boot automatizuje konfiguraci serveru, což vývojářům ušetří čas a úsilí spojené s přípravou
prostředí pro zpracování HTTP požadavků.
Tato vlastnost činí Spring Boot obzvláště atraktivním řešením pro vývojáře, kteří chtějí vytvářet a nasazovat webové aplikace, a je důvodem, proč je Spring Boot často volbou i ve firemním prostředí. Navíc, použití Spring Bootu nevede k žádným významným nevýhodám, kromě možného nárůstu velikosti aplikace, což představuje jasnou volbu pro moderní vývoj webových projektů.

3.2.2 Popis MariaDB jakožto systému pro správu databází

MariaDB, open-source relační databázový systém, byl založen v roce 2009 jako forknutá verze MySQL, reagující na obavy komunity ohledně akvizice MySQL firmou Oracle. Tento krok byl motivován snahou udržet databázový systém volně přístupný a nezávislý. MariaDB a MySQL si i přes oddělení zachovávají vysokou úroveň vzájemné kompatibility, což platí pro verze MySQL 5.7 a starší, což umožňuje uživatelům MySQL snadný přechod na MariaDB bez potřeby významných změn ve svých aplikacích. Příkazy SQL, používané v obou systémech, jsou do velké míry identické, což zjednodušuje práci vývojářům, kteří jsou již se syntaxí MySQL obeznámeni. [15] [14]
MariaDB se od svého vzniku neustále vyvíjí a nabízí řadu vylepšení a nových funkcí, které přesahují původní možnosti MySQL. Mezi tyto vylepšení patří například pokročilé možnosti replikace, nové úložné enginy (např. Aria a TokuDB), které přinášejí lepší výkon a efektivitu práce s daty, a vylepšenou práci s transakcemi. Díky aktivní komunitě a otevřenému vývoji se MariaDB stala důležitým hráčem na poli databázových technologií, s širokou podporou v
mnoha aplikacích a vývojových nástrojích. Vývojáři a organizace si tak mohou vybrat MariaDB pro své projekty s důvěrou v robustní, bezpečné a vysoce výkonné řešení pro správu databází, které je zároveň plně kompatibilní s otevřenými standardy a podporuje širokou škálu operačních systémů.[13][14]

3.2.3 Charakteristika REST API a principy jeho komunikace

V rámci architektury stávajícího projektu je pro komunikaci mezi klientem a serverem využíváno rozhraní REST (Representational State Transfer), což představuje základní kámen moderních webových aplikací. Tento architektonický styl umožňuje systémům komunikovat prostřednictvím HTTP požadavků, čímž se zjednodušuje výměna dat a manipulace s nimi. RESTful API, implementované v projektu, poskytuje univerzální rozhraní pro různé typy klientů, jako jsou webové prohlížeče, mobilní aplikace či jiné serverové služby, což umožňuje efektivní práci s databází na serverové straně na základě přijatých požadavků. Jedním z klíčových principů REST je bezstavovost, která znamená, že každý požadavek od klienta na server musí obsahovat veškeré informace nutné k jeho zpracování, nezávisle na předchozích požadavcích. To přispívá k vyšší škálovatelnosti a odolnosti systému, protože server nemusí udržovat informace o stavu klienta. Tato nezávislost mezi klientem a serverem rovněž umožňuje jejich samostatný vývoj a údržbu, což je zvláště přínosné pro velké projekty, kde na vývoji backendové (serverové) a frontendové (klientské) části pracují odlišné týmy. Díky dodržování dohodnutých kontraktů v komunikaci mohou být tyto části vyvíjeny paralelně a nezávisle na sobě, což vede k efektivnějšímu vývojovému procesu a snadnějšímu testování.



Obrázek 5 - REST API [18]

V rámci REST architektury, klíčového prvků pro návrh distribuovaných systémů, dochází k interakci mezi klientem a serverem prostřednictvím požadavků a odpovědí. Tento model umožňuje klientům odesílat požadavky na server za účelem

vytvoření, smazání nebo modifikace dat uložených v databázi, zatímco server reaguje zasláním odpovědí. [16]

Každý požadavek v REST architektuře je definován pomocí několika základních komponent, které určují jeho strukturu a účel:
- **HTTP typ požadavku:** Specifikuje metodu požadavku, která určuje akci, jež má být na serveru provedena.
- **Hlavička:** Obsahuje metadatové informace, jako jsou typy obsahu, které klient může zpracovat, a další instrukce pro server.
- **Cesta ke zdroji:** Definuje unikátní identifikátor zdroje na serveru, s nímž se má interagovat.
- **Nepovinné/doplňující data:** Mohou obsahovat tělo požadavku ve formě posílaných dat nebo parametry dotazu.
[17] [18]

Mezi základní typy požadavků REST API patří:
- **POST:** Tento typ požadavku se používá pro vytvoření nového zdroje na serveru. Často je spojen s odesíláním dat, která mají být v zdroji uložena.
- **GET:** Metoda GET slouží k získání dat z určitého zdroje. Nezmění stav zdroje.
- **DELETE:** Jak název napovídá, DELETE se používá k odstranění specifikovaného zdroje ze serveru.
- **PUT:** Tento typ požadavku slouží k aktualizaci stávajícího zdroje nebo vytvoření nového, pokud specifikovaný zdroj neexistuje. Obvykle zahrnuje odeslání kompletního nového stavu zdroje.[17] [18]
Díky tomuto modelu komunikace mohou být aplikace navrženy tak, aby byly modulární, snadno rozšiřitelné a lépe udržovatelné, což z REST činí preferovanou volbu pro moderní webové aplikace.[18]

3.2.4 Knihovna Lombok

Knihovna Lombok představuje významný nástroj v ekosystému Javy, který výrazně zjednodušuje a zefektivňuje vývoj aplikací tím, že minimalizuje potřebu psaní opakujícího se, "boilerplate" kódu. Tento nástroj nachází své uplatnění zejména v kombinaci s frameworkem Spring Boot, kde synergicky působí na snížení

komplexity kódu a zlepšení čitelnosti, aniž by byla ovlivněna funkčnost nebo výkon aplikace. [19]

3.2.4.1 Využití knihovny Lombok

Lombok dosahuje své efektivity především automatizací tvorby standardních metod jako jsou gettery/settery, metody equals(), hashCode() a toString(), a také konstruktorů. Vývojáři díky tomu mohou věnovat více času logice aplikace než rutinnímu psaní kódu, který nevyžaduje zvláštní intelektuální úsilí. Kromě základních metod, Lombok poskytuje i anotace pro pokročilejší konstrukce jako jsou vzor builder, automatické zamykání (synchronized) a lazy inicializace. [20]

3.2.4.2 Integrace s Java Spring Boot

Spring Boot, což je framework usnadňující vývoj a nasazení springových aplikací, se stává ještě výkonnějším a uživatelsky přívětivějším díky integraci s Lombokem. Jednou z hlavních výhod Spring Bootu je jeho "opinionated" přístup k vývoji, kde mnoho aspektů aplikace je předkonfigurováno s cílem urychlit vývojový proces. Přidáním Lomboku do projektu vývojáři získávají další vrstvu abstrakce, která umožňuje ještě rychlejší iteraci a prototypování. [20] [21]

3.2.4.3 Praktické využití Lomboku v Spring Boot aplikacích

Při vývoji RESTful API s využitím Spring Bootu se často setkáme s potřebou definovat modelové třídy, které reprezentují entitní objekty v aplikaci. Tyto třídy obvykle obsahují mnoho getterů a setterů a konstruktorů, jejichž tvorba může být časově náročná. Lombok zde nabízí elegantní řešení pomocí anotací @Getter, @Setter a @NoArgsConstructor, @RequiredArgsConstructor, @AllArgsConstructor, které automaticky generují tyto metody v kompilačním čase. To nejen zjednodušuje kód, ale také zvyšuje jeho čitelnost. [20]

Další častou potřebou v Spring Boot aplikacích je implementace vrstvy přístupu k datům (DAO nebo Repository). Lombok zde může usnadnit práci s datovými přenosovými objekty (DTO) nebo projekcemi, kde anotace @Data kombinuje @Getter, @Setter, @EqualsAndHashCode a @ToString v jediné deklaraci. [20]

Integrace Lomboku do vývojového stacku Java Spring Boot aplikací poskytuje vývojářům výrazné zlepšení efektivity práce a čitelnosti kódu. Zatímco Spring Boot zjednodušuje konfiguraci a nasazení aplikací, Lombok minimalizuje nutnost psaní a udržování opakujícího se kódu. Společně tvoří tyto nástroje mocnou kombinaci, která umožňuje rychlejší vývoj, snazší údržbu a lepší škálovatelnost Javových aplikací. Vývojáři, kteří se naučí efektivně využívat oba nástroje, si mohou být jisti, že jejich práce bude produktivnější a jejich kód bude čistší a lépe strukturovaný.[21]

3.3 Analýza struktury projektu

3.3.1 Funkcionální a nefunkcionální požadavky

Projekt si klade za cíl vyvinout komplexní systém pro správu a organizaci robotických soutěží. Aplikace bude sloužit různým typům uživatelů – od organizátorů soutěží, přes soutěžící až po obecné diváky. Následující seznam popisuje klíčové funkcionální a nefunkcionální požadavky, které aplikace musí splňovat, aby byla účelná a uživatelsky přívětivá.

3.3.1.1 Uživatelská autentizace a autorizace

- **Registrace uživatele**: Systém umožní uživatelům vytvořit nový účet, vyžaduje zadání základních údajů jako jsou jméno, email a heslo.
- **Přihlášení uživatele**: Uživatelé se budou moci přihlásit k svým účtům pomocí emailu a hesla.
- **Přihlášení pomocí Google OAuth**: Nabídneme alternativní možnost přihlášení skrze Google účet pro zjednodušení přístupu.
- **Odhlášení uživatele**: Uživatelé budou mít možnost se bezpečně odhlásit ze svého účtu.

3.3.1.2 Správa soutěží

- **Zobrazit všechny ročníky soutěží**: Uživatelům se zpřístupní přehled historie soutěží.
- **Vytvořit soutěž**: Umožní organizátorům definovat novou soutěž, včetně jejího názvu, termínu a pravidel.
- **Zahájit soutěž**: Organizátoři budou moci oficiálně zahájit soutěž v systému.

- **Upravit soutěž**: Poskytne možnost modifikace detailů existujících soutěží.
- **Vymazat soutěž**: Umožní odstranění soutěže z systému.

3.3.1.3 Správa robotů a týmů

- **Zobrazit pořadí robotů v disciplíně**: Umožní sledovat aktuální umístění robotů v jednotlivých disciplínách.
- **Zobrazit skóre všech robotů v týmu**: Přehled skóre pro každý tým.
- **Vytváření a správa disciplín**: Nabídne nástroje pro přidání, úpravu a odstranění disciplín.
- **Vytváření, editace, a mazání hřišť**: Umožní správu hřišť, na kterých se soutěže odehrávají.
- **Registrace a správa týmů**: Systém umožní vytvoření týmů, jejich editaci a přidání či odebrání členů.

3.3.1.4 Správa zápasů a skupin

- **Vytváření a správa zápasů**: Umožní plánování, editaci a odstranění zápasů.
- **Vytváření zápasových skupin**: Organizátoři mohou vytvářet skupiny pro zápasy a řadit do nich týmy.
- **Modul pro řazení zápasů**: Automatické plánování zápasů podle výsledků a pravidel soutěže.

3.3.1.5 Správa uživatelů

- **Přidání uživatele do databáze a editace rolí**: Administrátoři budou moci spravovat role uživatelů a přidělovat jim různá oprávnění.
- **Změna hesla a zobrazení informací o uživateli**: Uživatelé budou mít možnost změnit své heslo a prohlížet si vlastní uživatelské informace.

3.3.2 Uživatelské rozhraní

V rámci této serverové aplikace, která je vyvíjena s využitím frameworku Java Spring Boot, se primárně setkáváme s omezenými možnostmi tradičního uživatelského rozhraní. Výchozí interakce s aplikací probíhá skrze konzolu nebo využitím CURL příkazů pro odesílání požadavků na API rozhraní. Tento přístup umožňuje vývojářům a administrátorům manipulovat s aplikací a testovat její funkčnost na serverové úrovni bez potřeby grafického uživatelského rozhraní (GUI).

Nicméně, pro zajištění vyšší uživatelské přívětivosti a přístupnosti systému konečným uživatelům, se tato práce zaměřuje na vývoj a integraci grafického uživatelského rozhraní webových stránek. Cílem je vytvořit intuitivní a snadno navigovatelné UI, které umožní uživatelům efektivně interagovat s aplikací, provádět požadované operace a získávat potřebné informace bez nutnosti hlubokých technických znalostí.

3.3.2.1 Klíčové aspekty navrhovaného uživatelského rozhraní:

1.Intuitivnost a jednoduchost: UI by mělo být navrženo tak, aby bylo intuitivně pochopitelné i pro uživatele, kteří nejsou technicky zdatní. To zahrnuje logickou organizaci prvků, jasné označení funkcí a minimalizaci kroků potřebných pro dosažení cíle.

2. Responzivní design: S ohledem na různorodost zařízení, která uživatelé v současnosti používají, je důležité, aby webové rozhraní bylo responzivní a dobře fungovalo na široké škále obrazovek – od desktopů po mobilní telefony a tablety.

3. Přístupnost: Rozhraní by mělo být přístupné podle mezinárodních standardů, což zahrnuje podporu pro čtečky obrazovky a další technologie, které umožní použití systému širokému spektru uživatelů, včetně těch se specifickými potřebami.

4. Bezpečnost: Jelikož bude uživatelské rozhraní sloužit k odesílání a přijímání dat mezi klientem a serverem, musí být zajištěna jeho bezpečnost. To znamená implementaci bezpečných autentizačních a autorizačních mechanismů, šifrování citlivých informací a ochranu proti běžným bezpečnostním hrozbám.

5. Interaktivita a zpětná vazba: Aplikace by měla poskytovat okamžitou zpětnou vazbu na akce uživatele, aby bylo jasné, že jejich požadavky byly úspěšně provedeny, nebo aby byli informováni o případných chybách a jak je řešit.

Vývojem a implementací takového grafického uživatelského rozhraní se projekt snaží nejen zvýšit uživatelskou přívětivost a přístupnost systému, ale také rozšířit jeho potenciální využití a dosah mezi širší skupinou uživatelů. Cílem je, aby konečný

produkt byl nejen funkčně bohatý a technologicky pokročilý, ale také jednoduchý na používání a vizuálně přitažlivý.

3.3.3 Technická architektura

V rámci této sekce diplomové práce se detailně zaměří na analýzu a rozbor klíčových aspektů serverové části implementace aplikace. Jedná se o zásadní proces, který umožňuje hlubší porozumění struktuře a funkcionalitě vyvíjeného systému.

Implementace serverové části je založena na jazyce Java, přičemž všechny relevantní zdrojové soubory jsou systematicky organizovány v projektové struktuře pod cestou „src/main/java/com/robogames/RoboCupMS“. [1] Tato organizace souborů nejenže usnadňuje navigaci v rámci projektu, ale také přispívá k efektivní správě a údržbě kódu. Mezi klíčové soubory patří:

- **AppInit.java:** Zodpovídá za inicializaci aplikace a nastavuje základní konfiguraci potřebnou pro spuštění.
- **ApplicationContextProvider.java:** Umožňuje přístup k aplikaci kontextu Springu v celé aplikaci, což je klíčové pro správu závislostí.
- **Communication.java**: Řídí komunikaci mezi serverem a klienty, zajišťuje zpracování požadavků a formátování odpovědí.
- **GlobalConfig.java**: Definuje globální nastavení aplikace, která jsou aplikována napříč celým systémem.
- **Response.java a ResponseHandler.java:** Tato dvojice souborů je zásadní pro správné formátování a odesílání odpovědí klientům, zajišťuje konzistentnost datových odpovědí.
- **RoboCupMsApplication.java:** Představuje vstupní bod aplikace, kde se spouští Spring Boot aplikace.

[1]

Struktura projektu je dále rozčleněna do logických složek, které odpovídají různým aspektům aplikace:

- **Business:** Obsahuje business logiku aplikace, zahrnující pravidla a operace spojené s logikou domény.

- **Controller:** Zodpovídá za zpracování uživatelských požadavků, mapuje je na příslušné servisní metody a vrací odpovědi.
- **Entity**: Definuje entity, které reprezentují datové modely aplikace a jsou mapovány na databázové tabulky.
- **Module:** Tato složka může obsahovat specifické moduly nebo funkcionality, které rozšiřují základní možnosti aplikace.
- **Repository**: Umožňuje interakci s databází pomocí Spring Data JPA, definuje metody pro přístup a manipulaci s daty.
- **Security:** Zajišťuje implementaci bezpečnostních aspektů aplikace, včetně autentizace a autorizace uživatelů.

[1]

Pochopení a správná implementace těchto komponent je zásadní pro vývoj robustní, bezpečné a škálovatelné serverové aplikace. V následujících oddílech se budeme věnovat podrobnému rozboru každé z těchto složek, jejich úlohy v celkovém kontextu aplikace a způsobu, jakým přispívají k efektivnímu a bezproblémovému provozu celého systému.

3.3.3.1 Soubor AppInit.java

AppInit.java hraje klíčovou roli v procesu inicializace aplikace vyvíjené v Java Spring Boot. Tento soubor je zodpovědný za řadu důležitých úkonů, jež připravují aplikaci k jejímu běhu, a to včetně nastavení konfigurace, inicializace databázových entit, a dalších nezbytných úvodních nastavení.

Soubor `AppInit.java` je umístěn v projektové struktuře pod cestou `src/main/java/com/robogames/RoboCupMS` a obsahuje několik metod, které jsou anotovány jako `@Bean`, což znamená, že Spring Boot při spuštění automaticky vyhledá tyto metody a spustí je jako součást konfigurace aplikace.

Tyto metody zahrnují:

- **contextProvider():** Vytváří a vrací instanci `ApplicationContextProvider`, která umožňuje přístup k aplikaci kontextu Springu v celé aplikaci.

- **loadConfigFromFile():** Načítá konfigurační data ze souboru `config.json`. Tato metoda je zásadní pro nastavení globálních konfiguračních proměnných, jako jsou například `HEADER_FIELD_TOKEN` a `TOKEN_VALIDITY_DURATION`. Použití

externího konfiguračního souboru umožňuje snadnou modifikaci nastavení bez nutnosti změn v kódu.

```
JSONParser parser = new JSONParser();
JSONObject obj = (JSONObject) parser.parse(new FileReader("config.json"));
```

Obrázek 6 - Načitání config.json [1]

- **initRole(RoleRepository repository):**
Inicializuje databázi s předdefinovanými rolemi uživatelů. Tato metoda zajišťuje, že systém má základní soubor rolí potřebných pro řízení přístupu a oprávnění uživatelů.

- **initScoreAggregation(ScoreAggregationRepository repository):**
Nastavuje základní agregace skóre, které se používají pro automatické vyhodnocení výsledků soutěže.

- **initCategory(CategoryRepository repository), initMatchState(MatchStateRepository repository), a initUsers(UserRepository repository):**
Tyto metody provádějí počáteční naplnění databáze kategoriemi soutěží, stavy zápasů a základními uživatelskými účty.

- **initDisciplines(DisciplineRepository repository, ScoreAggregationRepository aggregationRepository):**
Vytváří základní soubor disciplín, které jsou dostupné v rámci soutěží. Toto zahrnuje definice pravidel, časových limitů a počtu kol pro každou disciplínu.
Inicializační logika v `AppInit.java` je fundamentální pro správné nastavení aplikace a její připravenost k provozu ihned po spuštění. Umožňuje dynamické načítání konfigurace, což přináší flexibilitu v nastavení aplikace. Zároveň zajišťuje, že aplikace má všechny potřebné prvky pro svůj běh, včetně rolí uživatelů, kategorií, disciplín a pravidel soutěží.

3.3.3.2 Soubor ApplicationContextProvider.java

V rámci architektury aplikace vyvíjené pomocí frameworku Spring Boot je soubor **ApplicationContextProvider.java** prvek pro správu a přístup k aplikačnímu kontextu Springu v celé aplikaci. Tento soubor je implementací rozhraní **ApplicationContextAware**, což Springu umožňuje injektovat instanci **ApplicationContext** přímo do třídy. Tato funkcionalita je zásadní pro dynamickou správu beanů a závislostí v aplikaci. [1]

Soubor je umístěn v projektové struktuře pod cestou **src/main/java/com/robogames/RoboCupMS** a obsahuje definici třídy **ApplicationContextProvider**, která slouží jako most mezi Springovým aplikačním kontextem a zbytkem aplikace. Pomocí anotace **@Component** je třída označena jako komponenta Springu, což znamená, že bude automaticky detekována během skenování tříd a zaregistrována v kontextu aplikace. [8]

Klíčové metody třídy ApplicationContextProvider zahrnují:

- **getApplicationContext()**:
  - Tato metoda umožňuje ostatním částem aplikace získat přístup k aplikačnímu kontextu Springu. Díky tomu mohou různé komponenty snadno získávat spravované beany a využívat je ve své logice.
- **setApplicationContext(ApplicationContext ctx)**: Implementace metody rozhraní **ApplicationContextAware**.
  - Spring při inicializaci volá tuto metodu a předává ji aktuální instanci aplikačního kontextu. Metoda následně uloží tuto instanci do statické proměnné **context**, čímž umožňuje její další využití v aplikaci. [1]

3.3.3.3 Soubor Communication.java

V rámci architektury moderních aplikací je mechanismus zpětných volání (callbacků) klíčový pro efektivní komunikaci mezi objekty bez nutnosti tvrdého kódování závislostí mezi nimi. Tento přístup nejenže zvyšuje modularitu a testovatelnost aplikace, ale také usnadňuje implementaci reaktivních a asynchronních operací. V kontextu souboru `Communication.java` v projektu je tento mechanismus zajištěn prostřednictvím vzoru návrhu Singleton a poskytuje univerzální platformu pro správu zpětných volání v celé aplikaci.

Struktura `Communication.java`

Třída `Communication` je navržena jako Singleton, což znamená, že v rámci běhu aplikace může existovat pouze jedna instance této třídy. Tento návrh je ideální pro správu komunikace mezi různými částmi aplikace, jelikož poskytuje centralizovaný bod pro výměnu zpráv a dat. [22]

```
private static Communication communication = null;
```

Obrázek 7 - deklarace Singletonu [1]

Instance `Communication` uchovává seznam zpětných volání (`callBackList`), který je implementován jako synchronizovaný seznam `ArrayList`. Synchronizace je zde důležitá pro zajištění thread-safety při přístupu z různých vláken aplikace.

Klíčové Metody

- getCallBacks():
Tato metoda poskytuje přístup ke všem registrovaným zpětným voláním v seznamu. Umožňuje externím komponentám získat přehled o aktuálně aktivních callback funkcích.

- sendAll(Object sender, Object data):
Metoda `sendAll` prochází seznam zpětných volání a pro každý z nich volá jejich metodu `callBack`, předávajíc k tomu objekt odesílatele a data. Tímto způsobem lze efektivně distribuovat informace mezi různé části aplikace bez nutnosti znát přesnou implementaci nebo umístění příjemce.

- getInstance(): Statická metoda `getInstance` zajišťuje, že třída `Communication` bude využívat vzor Singleton. Pokud ještě neexistuje instance třídy, vytvoří ji. Tato metoda umožňuje ostatním komponentám aplikace získat přístup k instance `Communication` a využívat její funkcionality.

3.3.3.4 Soubor GlobalConfig.java

Soubor `GlobalConfig.java` představuje základní konfigurační soubor aplikace vyvinuté v rámci frameworku Spring Boot. Jeho úkolem je definovat statické proměnné, které slouží jako konstanty a globální nastavení pro různé části aplikace. Tyto proměnné jsou přístupné z celé aplikace, což umožňuje jednoduchou a centralizovanou správu důležitých nastavení.

- **API_PREFIX, AUTH_PREFIX, MODULE_PREFIX**: Tyto konstanty definují základní prefixy pro různé části API, což usnadňuje routování požadavků. `API_PREFIX` určuje obecný prefix pro všechna API, zatímco `AUTH_PREFIX` a `MODULE_PREFIX` specifikují prefixy pro autentizační rozhraní a modulové rozhraní.

- **HEADER_FIELD_TOKEN**: Tato proměnná určuje název hlavičky, která bude použita pro přenos autorizačních tokenů v HTTP requestech.

- **TOKEN_VALIDITY_DURATION**: Definuje dobu platnosti přístupového tokenu v minutách, což je kritický bezpečnostní parametr pro správu session a ověřování uživatelů.

- **USER_MIN_AGE, USER_MAX_AGE**: Stanovují minimální a maximální věk uživatele, což může být použito pro validaci registrace nebo pro specifické uživatelské role a přístupy.

-**ELEMENTARY_SCHOOL_MAX_AGE,HIGH_SCHOOL_MAX_AGE, UNIVERSITY_MAX_AGE**: Tyto proměnné definují věkové hranice pro různé vzdělávací kategorie. Systém je může využít k automatickému rozdělení soutěžících do kategorií na základě jejich věku.

- **MAX_ROBOTS_IN_DISCIPLINE, MAX_TEAM_MEMBERS:** Určují maximální počet robotů, které může mít jeden tým v disciplíně, a maximální počet členů v týmu. Tato nastavení umožňují administrátorům aplikace kontrolovat velikost týmů a jejich účast v jednotlivých disciplínách.

- **PASSWORD_ENCODER**: Inicializuje a definuje enkodér hesel, který je používán v aplikaci pro šifrování a verifikaci uživatelských hesel. Použití `BCryptPasswordEncoder` zajišťuje silné šifrování hesel, což je důležité pro zabezpečení uživatelských účtů.

Konfigurační soubor `GlobalConfig.java` je základním kamenem pro nastavení a správu klíčových parametrů aplikace. Umožňuje vývojářům a administrátorům efektivně spravovat konstanty a globální nastavení z jednoho místa.

3.3.3.5 Soubor Response.java

Třída `Response` v souboru `Response.java` je navržena tak, aby sloužila jako obecný obal pro data odesílaná ze serveru ke klientovi. Klíčovým aspektem této třídy je její schopnost reprezentovat různé typy odpovědí serveru – ať už se jedná o běžnou odpověď, varování, nebo chybu – a serializovat tato data do formátu JSON, což je standardně používaný formát pro výměnu dat mezi klientem a serverem ve webových aplikacích. [23]

Struktura Třídy

- **Enum Type:** Vnitřní výčtový typ definuje tři možné typy odezvy, které třída `Response` může zastupovat: `RESPONSE` pro standardní odpovědi, `WARNING` pro varování a `ERROR` pro chybové stavy. Tento přístup umožňuje jednoduchou diferenciaci mezi různými druhy odezvy.

- Atributy Třídy: Třída obsahuje dva hlavní atributy, `type` a `data`. Atribut `type` uchovává informaci o typu odpovědi, zatímco `data` slouží k uložení samotných dat odpovědi, která mohou být různého typu (například zpráva, objekt, číslo atd.).

- Konstruktor: Třída `Response` poskytuje konstruktor, který umožňuje inicializaci instancí s konkrétním typem odpovědi a příslušnými daty.

- Metoda toString(): Přepsání metody `toString()` slouží k serializaci instance třídy `Response` do formátu JSON. Pro tento účel využívá knihovnu Jackson

(`ObjectMapper`), což je standardní a rozšířený způsob práce s JSON v Javě. V případě, že serializace pomocí Jacksonu selže (například kvůli `JsonProcessingException`), metoda se pokusí vrátit jednoduchou reprezentaci dat ve formátu JSON jako řetězec.

Třída `Response` slouží tedy pro komunikaci mezi serverem a klientem v aplikacích postavených na architektuře klient-server. Umožňuje standardizovaný způsob odesílání odpovědí serveru. Díky použití JSON formátu pro serializaci dat je zajištěna vysoká úroveň interoperability a jednoduchá integrace s moderními webovými technologiemi. [23]

3.3.3.6 Soubor ResponseHandler.java

Třída `ResponseHandler` v souboru `ResponseHandler.java` je navržena jako klíčový prvek pro centralizované zpracování a generování odpovědí serveru ve všech kontrolérech aplikace. Skrze definované metody poskytuje unifikovaný způsob, jak vytvářet různé typy odpovědí (běžná odezva, varování, chyba) založené na společném modelu `Response`.

Metody `ResponseHandler`

- response(Object data): Metoda pro generování standardních odpovědí serveru. Jako parametr přijímá data, která mají být zaslána klientovi, a vrací instanci `Response` s typem `RESPONSE`. Tato metoda se používá, když operace na serveru proběhla úspěšně a je potřeba klientovi poskytnout výsledek.

- warning(Object data): Vytváří odpověď serveru s varováním. To může být užitečné v situacích, kdy je potřeba klienta informovat o potenciálním problému nebo upozornit na důležitou okolnost, která není přímo chybou, ale vyžaduje jeho pozornost.

- error(Object data): Metoda slouží k signalizaci chybových stavů. Použití anotace `@ResponseStatus(code = HttpStatus.BAD_REQUEST)` naznačuje, že v případě

chyby bude odpověď serveru doplněna o HTTP stavový kód `400 (Bad Request)`. Tento typ odpovědi je použit, když dojde k chybě na straně klienta, například při špatně formulovaném požadavku.

Třída je designována s ohledem na principy RESTful API, kde je důležité efektivně komunikovat stav operací a výsledků akcí mezi serverem a klientem.

3.3.3.7 Soubor RoboCupMSApplication.java

Třída `RoboCupMsApplication.java` slouží jako hlavní vstupní bod pro Spring Boot aplikaci, označenou anotací `@SpringBootApplication`. Automaticky nastavuje konfiguraci založenou na doporučeních frameworku Spring, aktivuje skenování komponent v balíčku, kde se třída nachází, a umožňuje také automatickou konfiguraci Spring MVC aplikací. [8]

```
@SpringBootApplication
public class RoboCupMsApplication {
    public static void main(String[] args) {
        SpringApplication.run(RoboCupMsApplication.class, args);
    }
}
```

Obrázek 8 - vstupní bod aplikace

- @SpringBootApplication: Tato anotace kombinuje tři důležité anotace **(`@Configuration`, `@EnableAutoConfiguration`, `@ComponentScan`)** do jedné, což značně zjednodušuje konfiguraci aplikace. **`@Configuration`** označuje třídu, která může obsahovat bean definice, **`@EnableAutoConfiguration`** povoluje Spring Boot auto-konfiguraci mechanismu, který se snaží automaticky nakonfigurovat aplikaci na základě přidaných závislostí, a **`@ComponentScan`** říká Springu, aby skenoval současný balíček a jeho podbalíčky za komponenty, servisy, konfigurace atd. [7] [8]

- main metoda: Tato statická metoda slouží jako vstupní bod do aplikace. Volání **`SpringApplication.run(RoboCupMsApplication.class, args);`** spouští Spring Boot aplikaci s použitím definicí nacházejících se v třídě `RoboCupMsApplication` a předává argumenty příkazové řádky, které mohou být použity v rámci aplikace.

3.3.3.8 Enum

ECategory.java

Soubor **ECategory.java** definuje výčtový typ **ECategory**, který slouží k kategorizaci týmů v rámci soutěžních aktivit. Výčet obsahuje čtyři kategorie:

- **ELEMENTARY_SCHOOL**: Tato kategorie je určena pro týmy základních škol.
- **HIGH_SCHOOL**: Tato kategorie pokrývá týmy středních škol.
- **UNIVERSITY**: Kategorie určená pro týmy vysokých škol.
- **OPEN**: Otevřená kategorie, která je dostupná pro širokou veřejnost bez omezení vzdělávacího stupně.

Tento enum umožňuje jednoznačně rozlišovat mezi týmy různých vzdělávacích stupňů a případně i veřejnosti.

EMatchState.java:

Soubor **EMatchState.java** definuje výčtový typ **EMatchState**, který reprezentuje stav zapasu v rámci soutěží. Obsahuje tři stavy:

- **WAITING**: Stav označuje, že zápas čeká na odehrání a výsledek ještě není znám.
- **DONE**: Označuje, že zápas byl již odehrán a skóre je známo.
- **REMATCH**: Stav určený pro situace, kdy je potřeba zápas z různých důvodů zopakovat, což znamená, že se původní skóre přepíše novým.

Enum **EMatchState** je zásadní pro správné plánování a vyhodnocení průběhu soutěží, poskytuje jasný přehled o aktuálním stavu každého zápasu.

ERole.java:

Soubor **ERole.java** popisuje výčtový typ **ERole**, který určuje možné role uživatelů v systému. Zahrnuje:

- **COMPETITOR**: Role pro soutěžící.
- **ADMIN**: Administrátorská role s vyššími oprávněními.
- **LEADER**: Role pro vedoucí týmu nebo skupiny.
- **ASSISTANT**: Asistent, který podporuje organizaci nebo tým.
- **REFEREE**: Rozhodčí odpovědný za dodržování pravidel a vyhodnocení zápasů.

Výčtový typ **ERole** je doplněn o vnitřní třídu **Names**, která obsahuje stringové konstanty pro jednotlivé role. Tato struktura je určena pro správu oprávnění a role uživatelů v aplikaci.

EScoreAggregation.java:

Enum obsahuje tři hodnoty:

- **MIN**: Minimální skóre získané robotem ve všech zápasech.
- **MAX**: Maximální skóre získané robotem.
- **SUM**: Součet skóre ze všech odehraných zápasů.

3.3.3.9 Object

Tato složka obsahuje soubory definujicí objekty v systému. Jsou zde přítomny –

CompetitionObj.java

Třída CompetitionObj slouží jako modelový objekt pro reprezentaci soutěže v rámci systému. Obsahuje základní informace o konkrétní soutěži, včetně roku konání, data konání, času zahájení a času ukončení soutěže.

Hlavní atributy:

year: Celé číslo reprezentující rok, ve kterém se soutěž koná.

date: Datum konání soutěže.

startTime: Čas, kdy soutěž začíná.

endTime: Čas, kdy soutěž končí.

[1]

DisciplineObj.java

Třída **DisciplineObj** obsahuje informace o názvu disciplíny, jejím popisu, způsobu agregace skóre pro vyhodnocení výsledků, časovém limitu pro jednotlivé zápasy a maximálním počtu kol, která může robot v disciplíně odehrát.

Hlavní atributy:

- **name**: Název disciplíny, který ji jednoznačně identifikuje.
- **description**: Textový popis disciplíny, poskytující účastníkům a organizátorům podrobnější informace.
- **scoreAggregation**: Enum **EScoreAggregation**, který určuje, jakým způsobem se bude počítat skóre z jednotlivých kol (MIN, MAX, SUM).
- **time**: Časový limit pro jednotlivé zápasy v disciplíně, vyjádřený v sekundách.
- **maxRounds**: Maximální počet kol, které může jeden robot v disciplíně odehrát.

MatchGroupObj.java

Třída `MatchGroupObj` slouží k reprezentaci zápasové skupiny v rámci soutěžního systému. Jejím základním účelem je uchovávat informaci o identitě tvůrce skupiny.

Hlavní atribut:

- creatorID: Celé číslo typu `Long`, které slouží jako unikátní identifikátor osoby (nebo systému), která zápasovou skupinu vytvořila.

PlaygroundObj.java

Třída `PlaygroundObj` v sobě integruje informace o hřištích používaných v rámci soutěžních disciplín, nabízí atributy pro název hřiště, jeho číselné označení pro snadnější identifikaci a ID disciplíny, ke které je hřiště přiřazeno. Konstruktor umožňuje inicializaci s těmito parametry, zatímco settery a gettery poskytují možnost jejich následného čtení a úpravy. Tento objektový model slouží jako základ pro organizaci a správu prostorů, ve kterých se konají jednotlivé soutěžní aktivity, a zjednodušuje přiřazování hřišť k specifickým disciplínám.

RobotMarchObj.java

Třída RobotMatchObj je určena pro správu informací o zápasech v robotických soutěžích. Každá instance této třídy reprezentuje konkrétní zápas, přičemž obsahuje identifikátory pro robota účastnícího se zápasu (robotID), hřiště, na kterém se zápas odehraje (playgroundID), a skupinu, do které zápas náleží (groupID).

Skupina může být specifikována negativním ID, což znamená, že zápas není součástí žádné skupiny.

RobotObj.java

Třída `RobotObj` je základní model pro reprezentaci robota v rámci robotických soutěží. Tento objekt se zaměřuje především na jméno robota, které slouží jako jeho primární identifikátor nebo charakteristika. Třída nabízí jak bezparametrický, tak parametrický konstruktor, čímž umožňuje vytváření instancí bez počátečních hodnot nebo s předdefinovaným jménem.

TeamObj.java

Třída TeamObj slouží jako model pro týmy účastnící se soutěží. Tento jednoduchý objekt se zaměřuje na uchování jména týmu, což je základní a nezbytný atribut pro identifikaci a organizaci týmů v rámci soutěžního prostředí.

TeamRegistrationObj.java

Třída TeamRegistrationObj slouží k modelování registrace týmu do soutěže, s důrazem na ročník soutěže a volbu kategorie registrace. Základním účelem je uchovávat informace týkající se rozhodnutí týmu registrace v konkrétním ročníku soutěže a určit, zda se tým přihlašuje do otevřené kategorie nebo bude zařazen do kategorie podle věku nejstaršího člena týmu.

3.3.3.10 Složka Controller

Jsou zde přítomny soubory zajišťujíci interakci mezi uživatelským rozhraním a servisní logikou aplikace. Například **CompetitionController.java** se zaměřuje na operace související s soutěžemi, jako je získání seznamu všech ročníků soutěží, správa registrací týmů, vytváření nových soutěží, jejich úprava, odstranění a spuštění. Jsou zde přítomny kontrolery pro disciplíny, hřiště, týmy atd.

3.3.3.11 Složka Entity

V této složce se nachází definice všech entit, vyskytujících se v systému. Například Třída **Category** v balíčku **com.robogames.RoboCupMS.Entity** představuje entitu kategorie, která je použita pro definici různých kategorií, ve kterých může tým

soutěžit. Tato entita je mapována na tabulku **category** v databázi, což umožňuje persistenci dat souvisejících s kategoriemi soutěže.

3.3.3.12 Složka Module

Zde jsou implementovány moduly pro automatizaci procesů spojených se soutěží. Modul **CompetitionEvaluation** usnadňuje práci s vyhodnocováním soutěže. Umožňuje získat skóre všech robotů účastnících se v daném ročníku soutěže a specifické kategorii, skóre robotů určitého týmu a také skóre konkrétního robota. Navíc poskytuje informace o umístění robotů v konkrétní disciplíně a kategorii. Všechny tyto operace jsou dostupné prostřednictvím REST API. [1]

OrderManagement se zabývá zobrazením aktuálního pořadí zápasů a jejich generováním. Umožňuje spustit modul pro řízení pořadí zápasů, kontrolovat, zda je servis spuštěn, vyžádat si obnovení systému v případě problémů, zobrazit aktuální zápasy probíhající na hřištích, požadovat změnu pořadí zápasů ve frontě a získat seznam všech nadcházejících zápasů pro daného robota. [1]

3.3.3.13 Složka Repository

Obsahuje repozitáře pro entity v systému.

Pro příklad třída **CategoryRepository** v balíčku **com.robogames.RoboCupMS.Repository** představuje repozitář pro entitu **Category**, která slouží k ukládání a získávání informací o kategoriích týmů z databáze. Tento repozitář rozšiřuje **JpaRepository**, což je součást Spring Data JPA a poskytuje soubor standardních metod pro práci s daty, jako je načítání, ukládání, mazání a další. [1]

3.3.3.14 Složka Security

Soubor **AuthController** slouží jako kontroler pro autentizační a autorizační operace v aplikaci. Zajišťuje správu uživatelských přihlášení, odhlášení, registrací a integraci s OAuth2 pro externí autentizaci. Využívá službu AuthService pro provedení bezpečnostních operací a generování přístupových tokenů.[1]

Třída **SecurityConfig** v balíčku **com.robogames.RoboCupMS.Security** je konfigurační třídou pro zabezpečení serverové části aplikace. Využívá Spring Security framework k definování a uplatňování bezpečnostních politik.

3.4 API endpointy

3.4.1 Definice API a endpointu

API (Application Programming Interface) je definováno jako sada pravidel, které umožňují interakci mezi různými softwarovými aplikacemi. Jedním z klíčových konceptů v oblasti API je endpoint. Endpoint představuje koncový bod komunikace, který je definován jedinečnou URL adresou, ke které se aplikace připojuje za účelem získání určitých dat nebo provádění konkrétních operací. [33]

V serverové části aplikace slouží jako hlavní rozhraní, jak bude klientská aplikace komunikovat se serverovou částí. [1]

4 PRŮZKUM MOŽNÝCH VYLEPŠENÍ A NOVÝCH FUNKCIONALIT WEB APLIKACE PRO ORGANIZACI ROBOTICKÉ SOUTĚŽE

Na základě podrobné analýzy stávající aplikace bylo identifikováno několik klíčových oblastí, které vyžadují zlepšení nebo rozšíření, aby bylo možné zvýšit celkovou efektivitu a uživatelskou přívětivost systému. Jedním z primárních cílů těchto vylepšení je rozvoj serverové části aplikace o klientské rozhraní. Toto rozšíření umožní uživatelům snadnější a přímější interakci se systémem, což vede k větší efektivitě a zlepšení celkového uživatelského zážitku.

Dalším významným krokem je aktualizace a optimalizace interních procesů a funkcionalit systému tak, aby lépe reflektovaly aktuální standardy a pravidla používaná v rámci soutěže Robogames. To zahrnuje revizi a možné přepracování datových modelů, autentizačních mechanismů a bezpečnostních protokolů, stejně jako zlepšení rozhraní pro správu soutěže, registraci uživatelů a zpracování výsledků. Tato úprava je zásadní pro zajištění, že aplikace nejenže odpovídá současným potřebám organizátorů a účastníků soutěže, ale je také schopna flexibilně reagovat na budoucí změny a vylepšení.

4.1 Uživatelské rozhraní

V této části se práce bude věnovat rozboru aspektů uživatelského rozhraní a možnostmi, jak jej implementovat do stávající aplikace.

4.1.1 Úvod do User Interface (UI)

V současné době představuje uživatelské rozhraní (UI) klíčový prvek softwarových aplikací. Jeho kvalita a funkčnost mají zásadní vliv na úspěch aplikace na trhu, neboť jsou přímým ovlivňovatelem uživatelské zkušenosti (UX). Kvalitně navržené UI by mělo být nejen intuitivní a responzivní, ale také esteticky lákavé, což umožňuje uživatelům efektivní a příjemnou interakci s aplikací. Z těchto důvodů se kvalita uživatelského rozhraní stává nezbytnou pro rozvoj a zdokonalení každého softwarového systému. [29] [28]

V kontextu navrhovaného vylepšení softwarové aplikace je zcela zásadní přistupovat k UI s ohledem na potřeby a očekávání všech uživatelů – od účastníků soutěže po její organizátory. Efektivní a intuitivní uživatelské rozhraní umožní uživatelům snadněji navigovat systémem, rychleji zobrazovat nebo upravovat informace, což přispívá k celkové spokojenosti a plynulosti uživatelského zážitku.

Je třeba věnovat pozornost nejen základní funkčnosti, ale i detailům, jako jsou barvy, fonty a rozložení prvků, které společně vytvářejí dojem z aplikace a ovlivňují, jak je systém vnímán a jak efektivně s ním uživatelé mohou pracovat. [29]

4.1.2 Možnosti

V procesu vývoje uživatelského rozhraní (UI) mají vývojáři možnost využít široké spektrum dostupných nástrojů, technologií a frameworků. Tato bohatá paleta možností nabízí vývojářům flexibilitu ve výběru řešení, které nejlépe vyhovuje konkrétním potřebám a cílům projektu. Volba vhodné technologie je zásadní, neboť má přímý vliv na údržbu, rozšiřitelnost, výkon a celkovou kvalitu výsledné aplikace. Existují dva základní přístupy k vývoji UI: integrace se serverovou částí (například jako součást Spring Boot aplikace) a vývoj nezávislého klientského rozhraní, které komunikuje se serverovou částí prostřednictvím API. I když oba přístupy mají své výhody, druhý zmiňovaný přístup, kdy je UI vyvíjeno odděleně od backendu, je často preferován z několika důvodů.

Rozdělení frontendu a backendu do samostatných částí a komunikace mezi nimi skrze API poskytuje výraznou míru flexibility. Tento model umožňuje týmům pracovat na různých částech aplikace paralelně a nezávisle na sobě, což usnadňuje koordinaci práce ve větších projektech a zrychluje vývojový cyklus. Například tým zaměřený na backend může implementovat a optimalizovat serverovou logiku, zatímco frontend tým může současně pracovat na uživatelském rozhraní, oba bez nutnosti čekat na dokončení práce druhého týmu.

Další výhodou odděleného UI je lepší škálovatelnost projektu. Aplikace může efektivněji využívat zdroje, neboť zátěž mezi klientskou a serverovou částí je jasně rozdělena. To umožňuje optimalizaci výkonu a zlepšení uživatelské zkušenosti, neboť frontend může být navržen s ohledem na rychlost a reaktivitu, zatímco backend se může zaměřit na efektivitu zpracování dat a bezpečnost.

4.1.2.1 Frameworky

V oblasti vývoje uživatelských rozhraní (UI) mají dnes vývojáři k dispozici širokou paletu frameworků, které značně usnadňují a zefektivňují celý proces tvorby. Tyto frameworky jsou obzvláště cenné díky nabídce předdefinovaných procesů a komponent, které umožňují vývojářům efektivně strukturovat uživatelské rozhraní. Kromě toho poskytují rozsáhlé knihovny a nástroje, které se zabývají řešením široké

škály běžných vývojových výzev, od routování a správy stavu po integraci s backendovými službami. [30]

Frameworky jako React, Angular a Vue.js se těší velké popularitě mezi vývojáři díky jejich flexibilitě, rozšiřitelnosti a efektivnímu způsobu řešení komplexních problémů spojených s moderním vývojem webových a mobilních aplikací. Každý z těchto frameworků přináší unikátní přístupy a filozofie, které ovlivňují způsob, jakým jsou aplikace navrhovány a vyvíjeny. [30]

React

React, vyvinutý Facebookem, se zaměřuje na deklarativní tvorbu uživatelských rozhraní s vysokým výkonem za použití komponent. Jeho virtuální DOM nabízí efektivní aktualizaci a renderování UI, což zlepšuje uživatelský zážitek i výkon aplikace. React také podporuje JSX, syntaxi, která umožňuje psát HTML struktury přímo v JavaScriptu, což vede k větší čitelnosti a udržitelnosti kódu. [24]

Angular

Angular, vyvíjený Googlem, je kompletní framework pro vývoj single-page aplikací (SPA). Nabízí robustní sadu nástrojů pro vývoj, včetně pokročilého datového vazebního mechanismu, podpory pro TypeScript pro lepší typovou kontrolu a modularitu, a silný systém injektování závislostí. Angular je ideální pro vývoj rozsáhlých, komplexních aplikací, kde je důraz kladen na architekturu a testovatelnost.[31]

Vue.js

Vue.js se vyznačuje svou jednoduchostí a flexibilitou. Je navržen tak, aby byl postupně přijímatelný, což znamená, že vývojáři mohou začít s malým jádrem a postupně přidávat další funkce a knihovny podle potřeby. Vue.js vyniká vytvořením dynamických uživatelských rozhraní s minimálním úsilím a je oblíben pro svoji snadnou naučitelnost a rychlý vývoj.[32]

4.1.3 Framework React

4.1.3.1 Funkce

React umožňuje vývojářům stavět aplikace pomocí opakovaně použitelných UI komponent. Tento komponentový přístup umožňuje izolovat jednotlivé části UI a snadno je spravovat. React také využívá virtuální DOM, což je lehká kopie skutečného DOM stromu, umožňující efektivnější aktualizace UI.

4.1.3.2 Výhody

- **Výkon**: Díky virtuálnímu DOM je React velmi rychlý, což vede k lepšímu uživatelskému zážitku, zejména na webových stránkách s intenzivní interakcí.
- **Komponentový model**: Umožňuje snadné znovupoužití kódu, což zvyšuje efektivitu vývoje.
- **Široká podpora a komunita**: Velká a aktivní komunita znamená mnoho dostupných zdrojů, knihoven a pluginů.

4.1.3.3 Nevýhody

- **Křivka učení**: Ačkoliv není React příliš složitý, vyžaduje od vývojářů pochopení některých klíčových konceptů, jako jsou komponenty, props a stavy.
- **Bohatý ekosystém**: Velké množství dostupných nástrojů a knihoven může být pro nováčky přesycující.

4.2 Úprava serverové části

Severová část z velké části splňuje požadavky pro použití na reálné soutěži Robogames, ale jsou tu přítomny některé procesy, které neodpovídají dnešní organizaci soutěže. Tato část se bude věnovat identifikaci problémů a jejich řešení, které budou zrealizovány a popsány v praktické části práce.

4.2.1 Způsob přidávání uživatelů do týmu

Původní způsob správy uživatelů v týmu představoval určité obtíže. Vedoucí týmu musel získat identifikační řetězec (UUID) jiného uživatele, kterého chtěl přidat. Tento řetězec byl dlouhý asi dvacet znaků a bylo obtížné si ho zapamatovat. Kvůli

tomu bylo nepraktické ho opakovaně sdílet mezi uživateli, a tak se často musel používat přes aplikace třetích stran. Tento proces zpomaloval a zbytečně komplikoval celkový pracovní postup. Aby se tento nedostatek vyřešil a zlepšil se tok práce v týmu, bylo nezbytné najít alternativní, uživatelsky přívětivější metodu pro správu uživatelů.

4.2.1.1 Nový způsob

V rámci aktualizace serverové části systému je navržen nový postup založený na konceptu pozvánek. Tato inovativní funkce umožní vedoucímu týmu snadněji vyhledávat uživatele v systému podle jejich jména nebo e-mailu a zaslat jim pozvánku ke vstupu do týmu. Díky této funkcionalitě se aplikace bude soustředit pouze na uživatele, kteří ještě nejsou součástí týmu, což zefektivní proces vytváření nových týmů.

Pro lepší uživatelskou přívětivost bude každý uživatel mít na horní liště aplikace ikonu upozornění, která zobrazí aktivní pozvánky. Tyto pozvánky může uživatel jednoduše přijmout nebo odmítnout, což výrazně urychlí proces přidávání nových členů do týmu a eliminuje potřebu složitého přeposílání UUID kódů.

4.2.2 Úprava věkových kategorií

Dnešní organizace soutěže Robogames počítá pouze s dvěma kategoriemi – pod 15 let a nad 15 let. Toto je nutné odrazit v úpravě serverové části, aby byl systém použitelný a daly se správně vyhodnocovat výsledky dle potřeb organizátorů Robogames.

4.2.2.1 Původní věkové kategorie

Původní systém zahrnoval 4 kategorie:

- Základní škola
- Střední škola
- Vysoká škola
- Open kategorie (kdokoliv bez omezení věku)

[1]

Vzhledem k tomu, že toto rozdělení neodpovídá současným požadavkům, je třeba ji upravit na serverové části aplikace.

4.2.2.2 Upravené věkové kategorie

Jak už bylo řečeno, úprava zahrnuje jinou konfiguraci věkových kategorií, a jejich redukce na pouze žáky základní školy (do 15 let) a studenty a dospělé (nad 15 let).

4.3 Shrnutí – návrh nových funkcionalit

Všechny nově implementované funkcionality aplikace, které jsou pro skutečnou použitelnost aplikace nejvíce potřebné, a proto budou v rámci této práce implementovány, jsou uvedeny v následujícím seznamu.

Obecné funkcionality zahrnují:

- Přihlášení, registrace a odhlášení uživatele
- Zobrazení a možnost úpravy uživatelského profilu
- Zobrazení obecných informací o soutěži
- Zobrazení výsledků ze všech ročníků, kategorií a disciplín soutěže

Funkcionality pro soutěžící:

- Možnost založení týmu a jeho správa
- Přidávání nových uživatelů do týmu pomocí systému pozvánek
- Registrace do soutěže
- Správa a registrace robotů do disciplín

Funkcionality pro vedení soutěže:

- Správa uživatelů v systému
- Přidělování rolí uživatelům
- Tvorba, úprava, odstranění a zahájení soutěží
- Správa disciplín a hřišť
- Automatické generování diplomů
- Potvrzování robotů
- Zobrazení týmů v systému

Funkcionality pro rozhodčí:

- Výběr ročníku a hřiště
- Generování a odstraňování zápasů pro konkrétní typ hřiště
- Zapsání výsledku a jeho úprava

Úprava serverové části:

- Přidání systému pozvánek
- Úprava věkových kategorií
- Přidání dalších API endpointů

V průběhu dalšího vývoje mohou vzniknout další potřeby úprav, především ohledně rozšíření stávajících API endpointů. Důvodem bude často potřeba získávat specifická data ze serveru na základě jiných parametrů, než které jsou momentálně dostupné. Tyto nové endpointy přinesou efektivnější práci na frontendové části aplikace, neboť umožní získávat požadovaná data přesně podle potřeb uživatelů a vývojářů. Veškeré úpravy budou důkladně zdokumentovány v praktické části práce, stejně jako všechny ostatní změny, které byly provedeny v rámci vývoje.

II. PRAKTICKÁ ČÁST

5 ÚVOD DO PRAKTICKÉ ČÁSTI

Tato sekce bude podrobně popisovat vytvořenou aplikaci a provedené úpravy na straně serverové aplikace. Jejím hlavním cílem je poskytnout čtenáři komplexní přehled o všech aspektech projektu v rozsahu, který umožní jasně porozumět fungování systému a provedeným změnám. Kromě toho tato část slouží jako dokumentace pro budoucí potřebné úpravy, poskytující důkladné vysvětlení všech detailů projektu.

Popis vytvořené aplikace bude zahrnovat klíčové funkce, uživatelské rozhraní, procesy interakce a průběh práce s aplikací pro různé typy uživatelů. Bude zde také popsána architektura aplikace a využité technologie.

Důraz bude kladen i na úpravy provedené na straně serverové aplikace, včetně nově implementovaných funkcionalit, rozšíření existujících API endpointů a jakékoliv další úpravy v databázovém modelu či procesech zpracování dat.

Všechny informace budou prezentovány s důrazem na srozumitelnost a kompletnost, aby poskytly budoucím vývojářům jasný a podrobný pohled na vyhotovenou aplikaci a provedené úpravy na serverové straně.

5.1 Použité technologie

Na vývoj aplikace byl použit framework React.
Kromě Reactu využijeme také další technologie pro zajištění funkčnosti a výkonnosti aplikace. Mezi ně patří JavaScript, HTML5 a CSS pro tvorbu uživatelského rozhraní, společně s knihovnami a nástroji jako React Router pro navigaci v aplikaci a další.
Taktéž je využita knihovna jsPDF, sloužící k automatickému generování diplomů. [34]

5.2 Popis implementace projektu

Výsledkem praktické části je zcela nová implementace front end webové aplikace „RobogamesApp“, která umožňuje přehlednou komunikaci a správu se serverem určeným ke plánování a realizaci samotné soutěže Robogames. Aplikace využívá všechna poskytnutá API a umožnuje tak komplexní správu celé soutěže pro organizátory a soutěžící.

Dalším výstupem tohoto projektu jsou úpravy serverové části aplikace, která původně vznikla v roce 2022 jako výsledek bakalářské práce Bc. Martina Krčmy [1]. Podrobný popis je uveden v následujících kapitolách.

6 INSTALACE A SPUŠTĚNÍ APLIKACE

Tato část se zabývá požadavky a postupem instalace této aplikace.

6.1 Vývojářské prostředí (IDE)

Aby bylo možné aplikaci upravovat a spustit, je nutné mít nainstalováno vhodné uživatelské prostředí pro vývoj React aplikací. Vývojářské prostředí slouží pro přehlednou úpravu a manipulaci s projektovými soubory a kódem. Zde jsou uvedeny nejpoužívanější volby mezi React komunitou vývojářů.

1. **Visual Studio Code**

   Nejpoužívanější IDE v rámci vývoje v React, která byla použita i pro tvorbu této aplikace. Jedná se o prostředí od firmy Microsoft a je dostupné pro operační systémy

   MacOS, Windows i Linux s podporou verzovacího systému Git. Jedná se o bezplatný open source software registrovaný pod licencí MIT. [35]

   Odkaz na stažení: https://code.visualstudio.com/

2. **WebStorm**

   WebStorm: WebStorm je plnohodnotné IDE vyvinuté společností JetBrains, které nabízí pokročilé funkce pro vývoj webových aplikací včetně React. Jeho hlavními výhodami jsou pokročilé nástroje pro refaktoring kódu, integrace s dalšími nástroji společnosti JetBrains (např. IntelliJ IDEA pro Java), a možnost debugování a testování aplikací přímo z rozhraní IDE. Jde o zpoplatněný software. [36]

   Odkaz na stažení: https://www.jetbrains.com/webstorm/

6.2 Node.js

Tento projekt využívá JSX (JavaScript XML), což je rozšíření syntaxe JavaScriptu, které umožňuje psát komponenty uživatelského rozhraní v Reactu jednodušeji a srozumitelněji.

Pro kompilaci JSX do běžného JavaScriptu a spuštění projektu je nezbytná instalace Node.js. Node.js poskytuje prostředí pro běh JavaScriptových aplikací mimo prohlížeč a obsahuje npm (Node Package Manager), který umožňuje snadnou instalaci a správu závislostí projektu.

Odkaz ke stažení: https://nodejs.org/en/download/current

6.3 Spuštení projektu

1. Nejdříve je nutné otevřít složku projektu ve zvoleném IDE.
2. Poté v terminálu ve složce projektu spustit příkaz „**npm install**“. Ten nainstaluje node moduly a další závislosti potřebné ke spuštění aplikace.

```
npm install
```

3. Zapnutí samotné aplikace se provádí pomocí příkazu v terminálu „**npm start**“

```
npm start
```

4. Aplikace se spustí ve výchozím prohlížeči počítače na **localhost:3000**.

7 STRUKTURA PROJEKTU

Tato část rozebírá adresář projektu a jeho součásti, aby pro čtenáře bylo srozumitelné kde se jaká část projektu nachází. Samostatný základ kódu uživatelského rozhraní je převzat z volně šiřitelné šablony Black Dashboard React, která je pod licencí MIT, tudíž opravňuje kohokoliv k manipulaci s projektem a jeho šíření i pro komerční účely za podmínek uvedení licence od původního autora. [39]

7.1 Adresářová struktura

Ve složce projektu se nachází následující soubory. Zde je popsán jejich stručný obsah.

1. **node_modules** - Složka obsahující všechny knihovny třetích stran, které projekt využívá, stažené pomocí npm (Node Package Manager).
2. **public** – Obsahuje statické assety, jako jsou HTML soubory, obrázky a soubory manifestů, které nejsou zpracovávány Webpackem nebo jiným buildovacím nástrojem.
3. **src** – Zdrojové soubory aplikace, včetně JavaScriptu, CSS a dalších React komponent. Nacházejí se zde také použité assety jako obrázky nebo loga.
4. **.env** - Soubor pro definování proměnných prostředí, které nejsou zahrnuty do zdrojového kódu z bezpečnostních nebo jiných důvodů.
5. **.gitignore** - Soubor určující, které soubory nebo složky Git ignoruje a nezahrnuje do verzování.
6. **.npmrc** - Konfigurační soubor pro npm, který obsahuje specifikace, jako jsou registry, z kterých se mají stahovat balíčky, nebo nastavení přístupu.
7. **CHANGELOG.md** - Markdown soubor s protokolem změn, který dokumentuje všechny změny a aktualizace provedené v projektu.
8. **ISSUE_TEMPLATE.md** - Markdown šablona pro hlášení problémů v GitHub repozitáři, pomáhá uživatelům poskytovat konzistentní a užitečné informace při hlášení nových problémů.
9. **jsconfig.json** - Konfigurační soubor pro Visual Studio Code zlepšující podporu pro JavaScript projekty.

10. **LICENSE.md** - Soubor obsahující licenční informace o tom, jak mohou být projekt a jeho soubory použity.
11. **package-lock.json** - Automaticky generovaný soubor, který přesně popisuje strom závislostí, které byly nainstalovány, což zajišťuje konzistenci instalací napříč instalacemi.
12. **package.json** - Srdce každého Node.js projektu nebo projektu používajícího npm. Definuje vlastnosti projektu, skripty, závislosti a další náležitosti.
13. **README.md** - Markdown soubor používaný k poskytování přehledu o projektu, instrukcích pro instalaci a dalších užitečných informacích pro uživatele nebo vývojáře.

7.2 Popis hlavních souborů a složek

Tato sekce podrobně popisuje klíčové části aplikace, které jsou zásadní pro její budoucí úpravy. Zahrnuje to složku **public**, **src** a soubor **.env**.

7.2.1 Složka „public“

1. **apple-icon.png** - Tento obrázek je verze ikony aplikace optimalizovaná pro produkty Apple, jako jsou iPhone a iPad. Používá se na domovských obrazovkách iOS zařízení, když je aplikace přidána jako webová aplikace.
2. **favicon.png** - Favicon je malý obrázek, který reprezentuje vaši stránku v záložkách prohlížeče. Tento konkrétní soubor je ve formátu PNG.
3. **index.html** - Toto je hlavní HTML soubor pro vaši React aplikaci. Všechny React komponenty jsou nakonec vloženy do elementu s ID **root** uvnitř tohoto souboru. HTML soubor také načítá externí skripty a styly a může obsahovat další meta informace nezbytné pro správnou funkci aplikace.
4. **manifest.json** - Tento JSON soubor definuje metadata aplikace potřebná pro přidání webové aplikace na domovskou obrazovku zařízení. Obsahuje informace jako jméno aplikace, ikony aplikace pro různá zařízení a zobrazovací režim. Manifest obsahuje informace jako krátké jméno aplikace, celé jméno odkazy na ikony, počáteční URL, barvu pozadí a téma.

7.2.2 Soubor „.env“

```
# .env
REACT_APP_API_URL=https://ens10.vas-server.cz:8080/
```

REACT_APP_API_URL: Tato proměnná prostředí definuje základní URL adresu pro API, ke kterému React aplikace provádí síťové požadavky. V tomto případě je nastavena na **https://ens10.vas-server.cz:8080/,** což je adresa serveru, pomocí kterého byla plikace vyvíjena. Použití této proměnné umožňuje snadno změnit cílové API server bez nutnosti změny zdrojového kódu.

7.2.3 Složka „src“

V této složce se nacházejí veškeré použité React komponenty, které aplikace používá, včetně assetů jako jsou obrázky či loga, soubor pro definici cest v projektu.

1. **css** – Tento adresář obsahuje kaskádové styly (CSS soubory), které definují vzhled a formátování HTML prvků aplikace.
2. **fonts** – Tady jsou uloženy písma používané v aplikaci.
3. **img** – Tento adresář uchovává obrázky používané v rámci aplikace.
4. **scss** – Složka **scss** obsahuje zdrojové soubory napsané v jazyce SCSS (Sassy CSS), který je preprocesor pro CSS. SCSS umožňuje použití proměnných, vnořených pravidel, mixinů a dalších funkcí, které usnadňují psaní a správu stylů.

7.2.4 Soubor „index.js“

Soubor **index.js** v projektu je hlavním vstupním bodem pro React aplikaci.

- **Importy knihoven a modulů**:

Kód importuje React a několik dalších knihoven, jako je ReactDOM pro manipulaci s DOM a react-router-dom pro routing v aplikaci. Importuje se také **AdminLayout** a **LoginLayout**, které jsou vlastní layoutové komponenty pro různé části aplikace.

- **Vytvoření kořenového elementu React**:
  - Využití **ReactDOM.createRoot()** pro vytvoření kořenového elementu aplikace, který je vázán na HTML element s ID **root**. To je standardní přístup pro React 18 a vyšší, který zavádí nový model vykreslování.
- **Vykreslování aplikace**:
  - Vnitřní struktura **BrowserRouter** používá **Routes** a **Route** pro definici routingu aplikace. Specificky, /**admin**/* vede na **AdminLayout** a /**robogames**/* na **LoginLayout**. Pro všechny ostatní cesty je nastavena přesměrovací route na /**admin**/**dashboard**.
  -
- **Správa cest a kontextu**:
  - Aplikace zahrnuje dynamické routování pomocí **react-router-dom**, které umožňuje uživatelům navigovat mezi různými částmi aplikace.

8 KOMPONENTY APLIKACE

V této části se zaměříme na klíčové komponenty použité v klientské části projektu a metody, jakými jsou informace zobrazovány uživatelům. Cílem je poskytnout čtenáři detailní pohled na strukturu a funkčnost aplikace.

8.1 Navbar – AdminNavbar

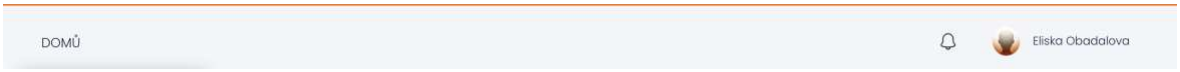
Komponenta **AdminNavbar** představuje hlavní navigační lištu administračního rozhraní aplikace. Je implementována pomocí technologií React a Reactstrap a zajišťuje následující funkcionalitu:

- **Zobrazení jména a profilové fotografie uživatele**: Uživatelské jméno a profilová fotografie jsou zobrazeny v rozbalovacím menu, které umožňuje rychlý přístup k profilu uživatele a dalším souvisejícím akcím.

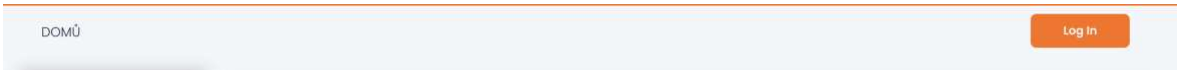
- **Zobrazení odkazů na uživatelský profil a tým**: V dropdownu po kliknutí na jméno se zobrazí dvě tlačítka – Můj profil a Můj tým.

- **Zobrazení čekajících pozvánek do týmu**: V navigační liště jsou zobrazeny čekající pozvánky do týmu s možnostmi přijetí nebo odmítnutí.

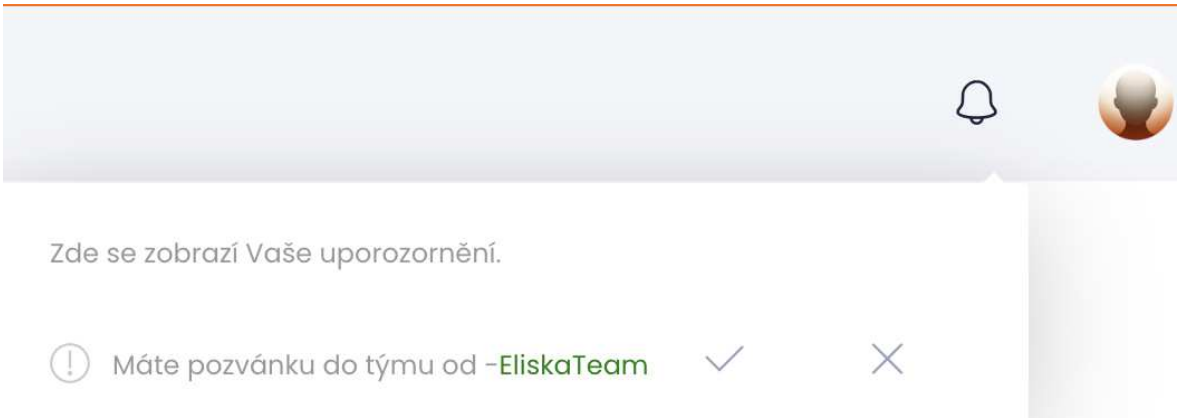
- **Odhlášení**: Komponenta obsahuje tlačítko pro odhlášení, které umožňuje uživateli bezpečně se odhlásit z aplikace.



Obrázek 9 - navbar - přihlášený uživatel



Obrázek 10 - navbar - nepřihlášený uživatel



Obrázek 11 - zobrazení pozvánek

8.1.1 Metody v komponentě AdminNavbar

8.1.1.1 fetchInvitations

- **Popis**: Tato asynchronní funkce načítá pozvánky do týmu pro přihlášeného uživatele. Pozvánky jsou získávány z API a uloženy do stavu komponenty.
- **Implementace**: Funkce volá endpoint API s autorizačním tokenem uživatele. V případě úspěšné odpovědi API se pozvánky ukládají do stavu. V případě chyby je zaznamenána do konzole.

8.1.1.2 acceptInvitation

- **Popis**: Funkce zpracovává přijetí týmové pozvánky. Po úspěšném přijetí je uživatel přidán do týmu.
- **Implementace**: Funkce vytváří **PUT** požadavek na API s ID pozvánky jako parametrem. Úspěšná odpověď signalizuje, že pozvánka byla přijata. V opačném případě se vypíše chybová hláška.

8.1.1.3 rejectInvitation

- **Popis**: Funkce umožňuje uživateli odmítnout týmovou pozvánku.
- **Implementace**: Podobně jako **acceptInvitation**, tato metoda volá API pomocí **PUT** požadavku s ID pozvánky a při úspěšné odpovědi oznamuje, že pozvánka byla odmítnuta.

8.1.1.4 fetchUserInfo

- **Popis**: Asynchronní funkce, která načítá informace o uživateli z API a ukládá je do stavu.

- **Implementace**: Funkce volá endpoint API a v případě úspěšné odpovědi uloží jméno a příjmení uživatele do stavu komponenty.

8.1.1.5 handleLogout

- **Popis**: Funkce, která zpracovává odhlášení uživatele z aplikace.
- **Implementace**: Funkce odstraní uživatelský token a další relevantní informace z lokálního úložiště a přesměruje uživatele na přihlašovací stránku.

Podmíněné renderování pak v JSX kódu zajišťuje, za jakých podmínek se budou zobrazovat jednotlivé objekty. Zde je uveden příklad, jak je vyřešeno zobrazení jiných prvků pro nepřihlášené a přihlášené uživatele.

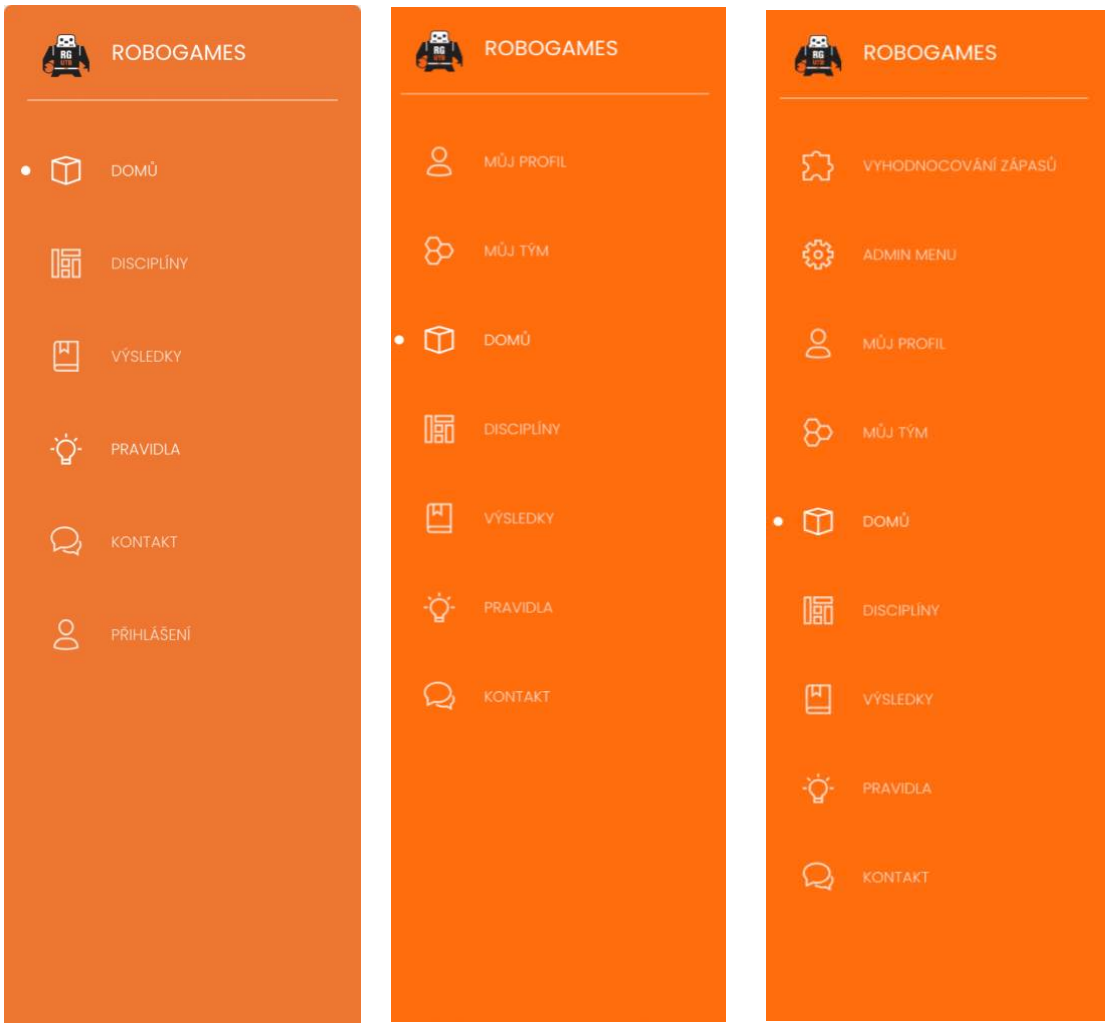
Je použita proměnná isLoggedIn, která se nastaví na true, pokud je v localstorage aplikace přítomen „token“. V případě, že je proměnná true, zobrazí se uživatelovo jméno, profilová fotka a upozornění. V opačném případě se zobrazí tlačítko odkazující na přihlašovací formulář.

```
{isLoggedIn ? (
  <>
    {/* Komponenty pro přihlášené uživatele */}
  </>
) : (
  <NavLink tag="li" className="nav-item">
    <Button color="primary" onClick={() => navigate('/robogames/login')}>
      Log In
    </Button>
  </NavLink>
)}
```

Obrázek 12 - Způsob podmíněného renderování

8.2 Sidebar

Komponent Sidebar.js slouží k zobrazení lišty na levé straně stránky. Obsahuje důležité odkazy na základní stránky aplikace. Zobrazení některých odkazů je limitováno pouze pro přihlášené uživatele nebo pro organizátory se specifickou rolí.



Obrázek 13 - sidebar nepřihlášený uživatel – přihlášený soutěžící - admin

8.2.1 activeRoute metoda

- **Popis**: Určuje, zda je aktuální cesta v prohlížeči shodná s cestou navigačního odkazu.
- **Implementace**: Porovnává aktuální cestu s cestou z props a vrací třídu "active", pokud se shodují, což umožňuje vizuálně zvýraznit aktivní odkaz.

8.2.2 Generování odkazů

V komponentě Sidebar jsou navigační odkazy generovány dynamicky z pole routes, které je předáno jako prop komponentě. Každý odkaz je reprezentován objektem v tomto poli, který obsahuje atributy jako path, name, icon, a layout. Zde je detailní popis procesu generování těchto odkazů:

1. **Iterace přes pole routes**:
   Komponenta prochází pole **routes** pomocí metody **map**, kde každý prvek pole (**prop**) reprezentuje jeden navigační odkaz.

2. **Element <NavLink>**: Pro každý platný odkaz se vytvoří element **<NavLink>**, který obsahuje ikonu a název cesty. Tento element zajišťuje navigaci pomocí React Router.

3. **Funkce activeRoute**: Tato funkce je volána v atributu **className** u **<NavLink>**. Funkce porovnává aktuální URL s **path** odkazu a pokud se shodují, přidá k elementu třídu **active**, která vizuálně zvýrazní aktivní odkaz.

8.2.2.1 Filtrování navigačních odkazů

```
<Nav>
  {routes.map((prop, key) => {
    if (prop.redirect || (prop.path === "/login" && isLoggedIn)) return null; // Hide login when logged in
    if (prop.path === "/register") return null; // Always hide register
    if (prop.path === "/user-management") return null; // Always hide user management
    if (prop.path === "/competition-management") return null; // Always hide competition management
    if (prop.path === "/competition-detail") return null; // Always hide competition detail
    if (prop.path === "/competition-registration") return null; // Always hide competition registration
    if (prop.path === "/robot-registration") return null; // Always hide robot registration
    if (prop.path === "/all-teams") return null; // Always hide all teams
    if (prop.path === "/playground-management") return null; // Always hide playground management
    if (prop.path === "/robot-confirmation") return null; // Always hide robot confirmation
    if (prop.path === "/playground-detail") return null; // Always hide playground detail
    if (prop.path === "/match-generation") return null; // Always hide match generation
    if (prop.path === "/match-management" && !isAdminOrLeaderOrAssistantOrReferee) return null; // Hide match management if the user doesn't have the required roles
    if (prop.path === "/admin-dashboard" && !isAdminOrLeaderOrAssistant) return null; // Hide admin dashboard if the user doesn't have the required roles
    if (prop.path === "/user-profile" && !isLoggedIn) return null; // Show user profile only when logged in
    if (prop.path === "/my-team" && !isLoggedIn) return null; // Show my team only when logged in
    return (
      <li className={activeRoute(prop.layout + prop.path) + (prop.pro ? " active-pro" : "")} key={key}>
        <NavLink to={prop.layout + prop.path} className="nav-link" onClick={linkOnClick}>
          <i className={prop.icon} />
          <p>{rtlActive ? prop.rtlName : prop.name}</p>
        </NavLink>
      </li>
```

Obrázek 14 - generování a filtrování odkazů

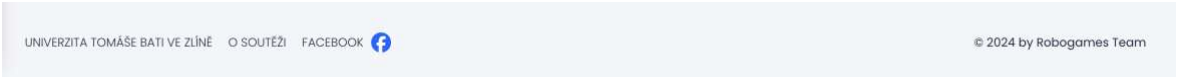
- **Popis**: Navigační odkazy jsou filtrovány na základě role uživatele a jeho stavu přihlášení. Odkazy jsou skryty nebo zobrazeny podle toho, zda uživatel splňuje specifické role nebo je přihlášen.

- **Implementace**:
    - Odkazy jako "Přihlášení" jsou skryty, pokud je uživatel přihlášen.Zda je uživatel přihlášen je zjištěno přítomností nebo nepřítomností tokenu v localstorage.

    - Některé administrativní odkazy jsou dostupné pouze pro specifické role, jako **ADMIN**, **LEADER**, **ASSISTANT** a **REFEREE**.
    - Odkazy na uživatelský profil a tým jsou dostupné jen když je uživatel přihlášen.

8.3 Footer

Footer slouží jak komponenta, která je umístěna na dolní strany obrazovky a obsahuje odkazy na stránku Univerzity Tomáše Bati, na hlavní stránku aplikace a na facebook Robogames. Kód vygeneruje Nav komponent a umístí všechny odkazy v něm jako NavItem. Nijak se v závisloti na podmínkách nemění.



Obrázek 15 - footer

```jsx
function Footer() {
  const navigate = useNavigate();

  return (
    <footer className="footer">
      <Container fluid>
        <Nav>
          <NavItem>
            <NavLink href="https://www.utb.cz">
              Univerzita Tomáše Bati ve Zlíně
            </NavLink>
          </NavItem>
          <NavItem>
            <NavLink onClick={() => navigate('/admin/dashboard')}>
              O soutěži
            </NavLink>
          </NavItem>
          <NavItem>

            <NavLink href="https://www.facebook.com/robogames.utb.cz">
              Facebook
            </NavLink>
          </NavItem>
          <img src={facebookLogo} alt="FB logo" style={{ maxWidth: '50%', height: '20px' }} />
        </Nav>
        <div className="copyright">
          © {new Date().getFullYear()} by Robogames Team
        </div>
      </Container>
    </footer>
  );
}

export default Footer;
```

Obrázek 16 - Footer komponent

8.4 Layouty

Aplikace využívá dva layouty

- **LoginLayout**: je použit u zobrazení Login.js a Register.js, pro tyto komponenty bylo nutné oddělat Sidebar pro větší přehlednost, proto mají svůj vlastní layout.
- **Admin Layout**: Ostatní zobrazení používají tento layout, který zahrnuje Sidebar

8.4.1 AdminLayout

Poskytuje celkové rozvržení stránky, zahrnující sidebar (boční panel), navigační lištu (navbar) a hlavní obsahovou oblast.

8.4.1.1 Základní Vlastnosti

- **Dynamic Background**: Využívá **BackgroundColorContext** pro dynamické úpravy pozadí aplikace.
- **Sidebar Toggling**: Spravuje stav otevírání a zavírání sidebaru na malých zařízeních.
- **Routes Rendering**: Renderuje cesty definované v souboru **routes.js**.
- **Brand Text**: Používá funkci **getBrandText** pro zjištění aktuálního textu značky zobrazeného v navigační liště.

8.4.1.2 Metody a Funkce

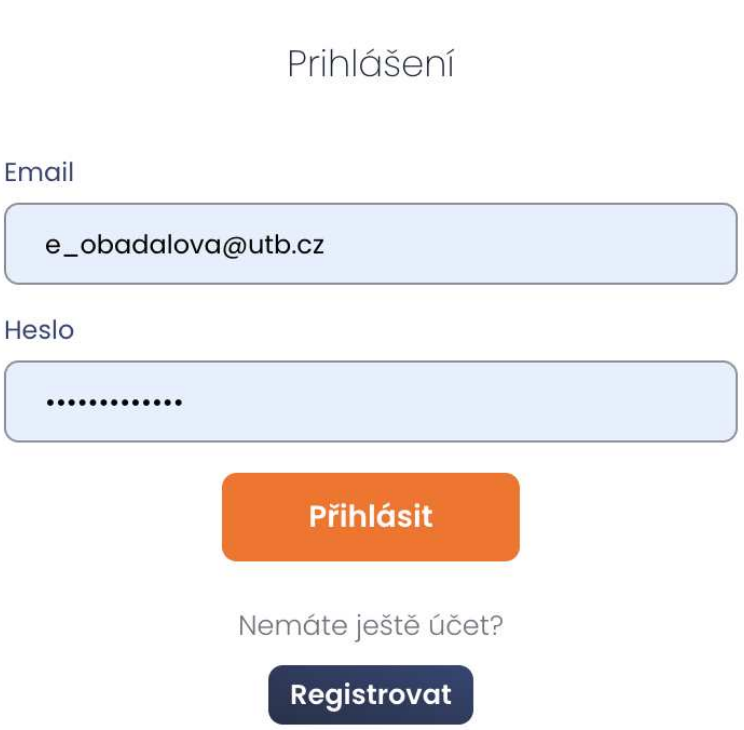
- **toggleSidebar**: Otevírá a zavírá sidebar na malých zařízeních. Tato funkce mění třídy na **document.documentElement** pro správu zobrazení sidebaru.
- **getRoutes**: Generuje **Route** komponenty pro každou cestu v **routes**, které jsou specifické pro administrační layout (/**admin**).

8.4.1.3 Struktura JSX

- **Sidebar Component**: Inicializuje a zobrazuje sidebar s logem a odkazy podle definovaných tras.
- **AdminNavbar Component**: Zobrazuje navigační lištu s dynamicky získaným textem značky.
- **FixedPlugin**: Komponenta pro změnu barvy pozadí, umožňuje uživatelům měnit téma aplikace.

8.5 Login

Komponenta **Login** je zodpovědná za zobrazení formuláře pro přihlášení a zpracování procesu přihlášení uživatele. Používá vazbu na kontext uživatele prostřednictvím hooku **useUser** pro přístup k funkci **login**, která slouží k autentizaci uživatele.

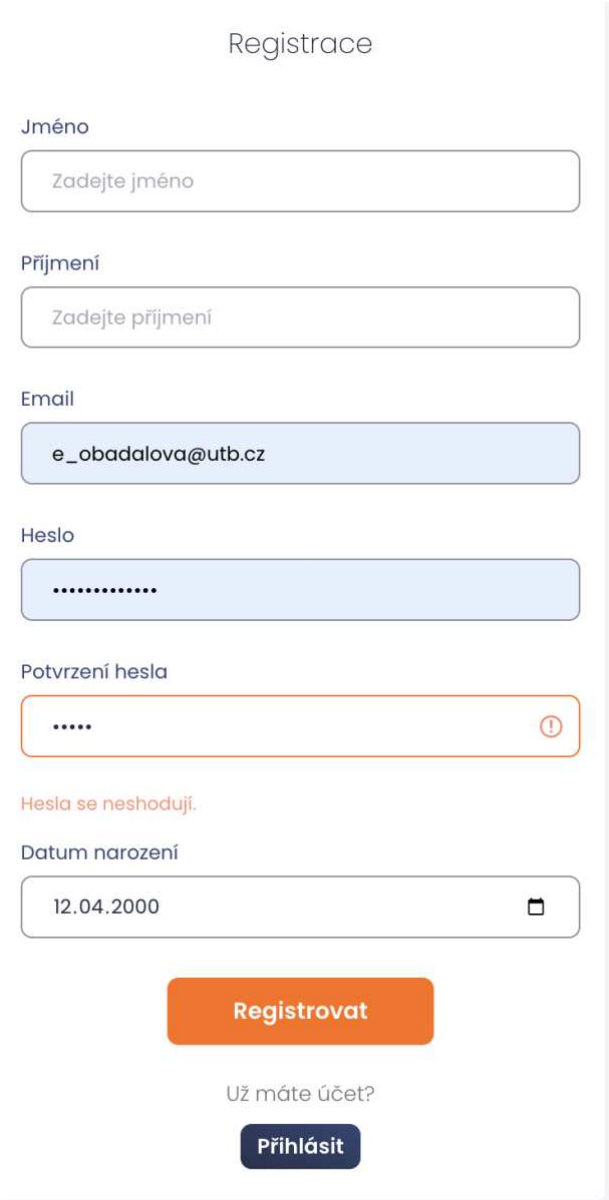


Obrázek 17 - formulář přihlášení

- **Formulář pro přihlášení**: Obsahuje vstupní pole pro email a heslo a tlačítko "Přihlásit".
- **Navigační tlačítka**: Kromě tlačítka pro přihlášení obsahuje komponenta také tlačítko "Registrovat", které uživatele přesměruje na stránku pro registraci.
- **useState**: Hook **useState** je použit pro správu stavů pro email a heslo.
- **useNavigate**: Hook **useNavigate** z **react-router-dom** je použit pro programové navigace, jako je přesměrování po úspěšném přihlášení nebo přechod na registrační stránku.
- **handleLogin**: Funkce **handleLogin** je volána při kliknutí na tlačítko "Přihlásit". Tato funkce volá funkci **login** z **UserContext**, která zpracovává přihlašovací údaje. Pokud je přihlášení úspěšné, uživatel je přesměrován na příslušnou stránku.

8.6 Register

Komponenta **Register** je zodpovědná za poskytnutí uživatelského rozhraní pro registraci nových uživatelů. Umožňuje uživatelům zadat potřebné údaje, které jsou následně odeslány na server pro zpracování.



Obrázek 18 - registrační formulář

8.6.1 Formulář

Obsahuje vstupní pole pro jméno, příjmení, email, heslo, potvrzení hesla a datum narození. Formulář je odeslán při kliknutí na tlačítko "Registrovat".

8.6.2 Validace a Odesílání Dat

- **Validace Emailu**: Email je validován na klientské straně pomocí regulárního výrazu pro kontrolu formátu emailové adresy.

Obrázek 19- regulární výraz pro validaci emailu

- **Validace Hesla**: Kontroluje, zda se zadané heslo a potvrzení hesla shodují.
- **Odesílání Dat na Server**: Po úspěšné validaci jsou data odeslána na server pomocí **fetch** API s metodou POST. Data jsou zahrnuta v těle požadavku ve formátu JSON.

8.6.3 Zachycení Chyb a Uživatelské Upozornění

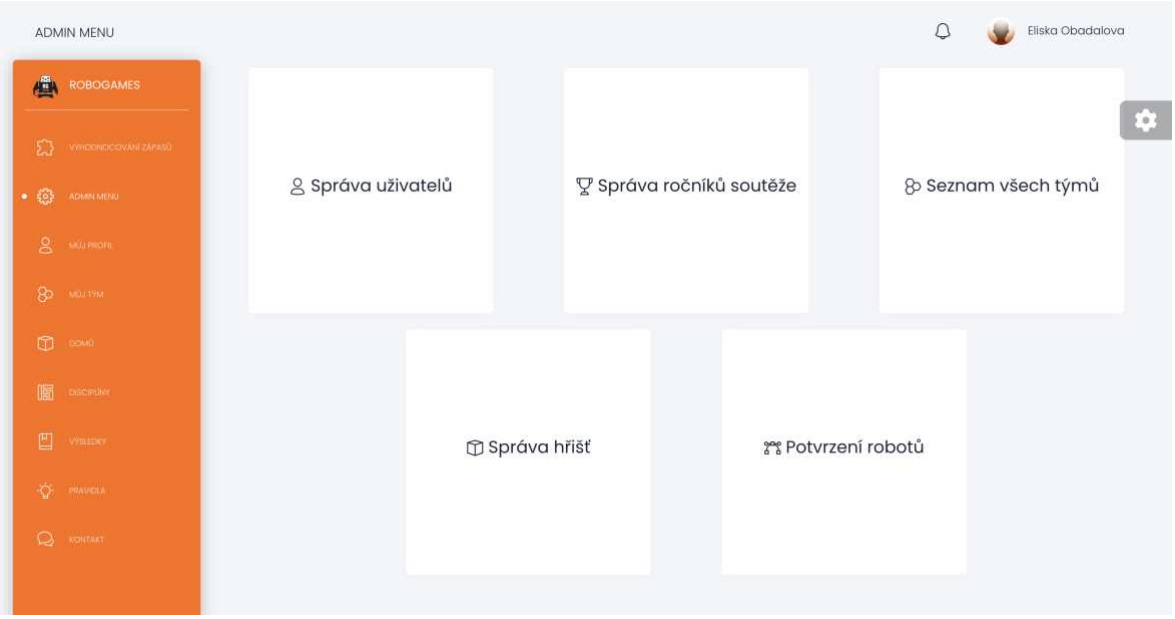
- **Zobrazení Chyb ve Formuláři**: Pokud dojde k chybě ve validaci (např. nevalidní email nebo neshodující se hesla), chyby jsou zobrazeny přímo ve formuláři pomocí **FormFeedback**.
- **Zpracování Odpovědi Serveru**: Odpověď od serveru je analyzována a v případě chyby (například uživatel s daným emailem již existuje) je uživatel upozorněn prostřednictvím vyskakovacího okna (alert).

8.6.4 Přesměrování a Další Interakce

- **Úspěšná Registrace**: Po úspěšné registraci je uživatel přesměrován na přihlašovací stránku.
- **Navigace na Přihlašovací Stránku**: Pro uživatele, kteří již mají účet, je poskytnuto tlačítko pro přesměrování na přihlašovací stránku.

8.7 AdminDashboard

Komponenta **AdminDashboard** slouží jako hlavní řídící panel pro administrační sekci aplikace, poskytujíc rychlý přístup k různým administračním úlohám prostřednictvím interaktivních karet. Každá karta představuje jednu oblast správy a umožňuje navigaci na příslušnou stránku.



Obrázek 20 - admin dashboard

8.8 AllTeams

AllTeams.js je jedna z administrátorských stránek, sloužící pro zobrazení všech týmů v systému. Využiví api

8.8.1 useState a useEffect Hooks:

Tyto React hooky jsou použity pro správu stavu týmů a pro asynchronní načítání dat při inicializaci komponenty.

8.8.2 fetchTeams Funkce:

Asynchronní funkce, která volá API endpoint pro získání seznamu týmů. Funkce je definována a volána v rámci **useEffect** hooku.

8.8.3 Použitý API Endpoint

- **Endpoint**: **${process.env.REACT_APP_API_URL}api/team/all**

- **Metoda**: GET
- **Headers**: Přikládá se autorizační token uživatele pro autentizaci požadavku.

8.8.4 Logika a Chování

- **Načítání dat**: Při prvním načtení komponenty se volá funkce **fetchTeams**, která z API získá seznam týmů. Před voláním API se kontroluje přítomnost autentizačního tokenu v **localStorage**.
- **Zpracování odpovědí API**: Po úspěšném načtení dat jsou data týmů uložena do stavu a zobrazena v tabulce. V případě chyby je zobrazena chybová zpráva.
- **Podmínky pro načítání**: Pokud není token k dispozici, data nejsou načtena a je zalogována chybová zpráva.

8.9 CompetitionDetail

Tato stránka slouží pro administrátory k zobrazení týmů v daném roce. Protože se na tuto stránku uživatel dostane po kliknutí na jednu z existujících objektů vytvořené soutěže, je předán rok v query stringu a poté využit pro volání API.
Každý řádek v tabulce reprezentuje jednu registraci týmu s detaily, které jsou relevantní pro administrátory soutěže.
Vizualizace dat umožňuje rychlou orientaci a analýzu registrací týmů pro konkrétní rok

8.9.1 Funkce a Metody

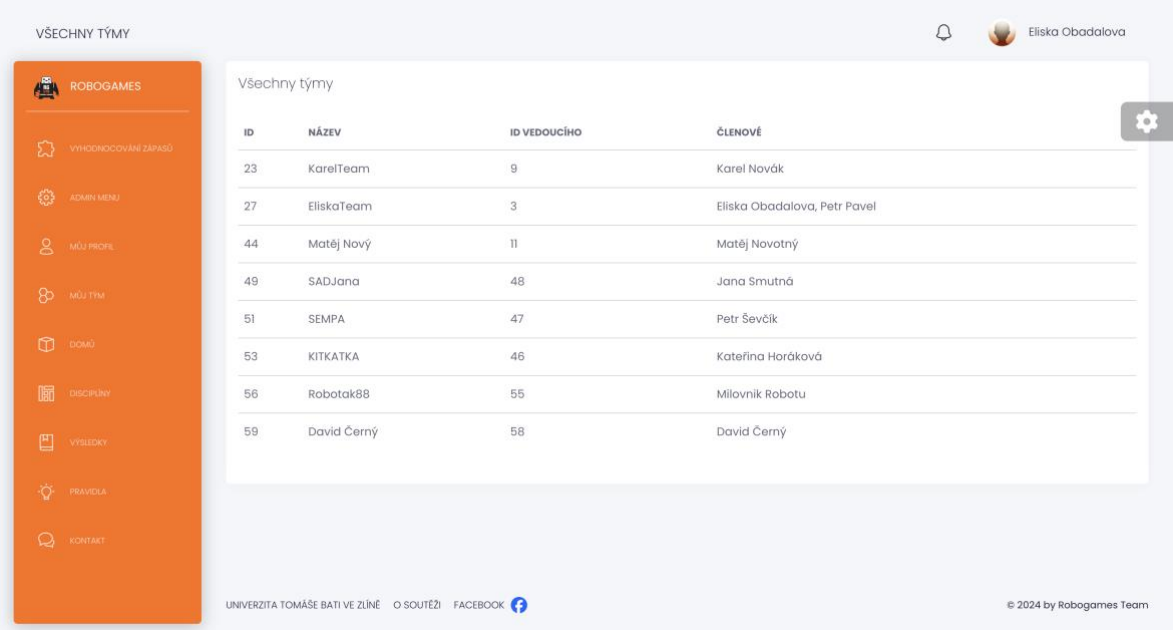
- **useQuery**: Vlastní hook pro získání parametrů z URL. V tomto případě je použit k získání hodnoty **year** z query parametrů.
- **useState a useEffect Hooks**: Slouží k udržení stavu registrací a k asynchronnímu načítání dat z API při změně roku nebo při prvním načtení komponenty.
- **fetchData Funkce**: Asynchronní funkce, která získává data z API pomocí fetch requestu. Data jsou načítána z endpointu, který je specifikovaný dynamicky na základě vybraného roku.

8.9.2 Použitý API Endpoint

- **Endpoint**: **${process.env.REACT_APP_API_URL}api/competition/allRegistrations ?year=${year}**
- **Metoda**: GET

8.9.3 Tabulka:

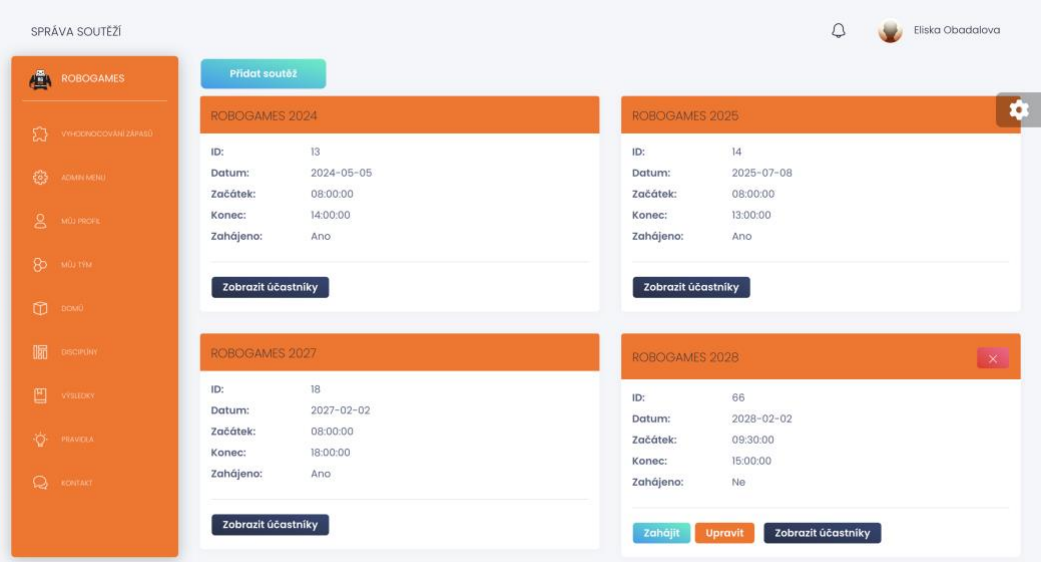
Zobrazuje ID registrace, ID týmu, název týmu a kategorii, ve které tým soutěží.



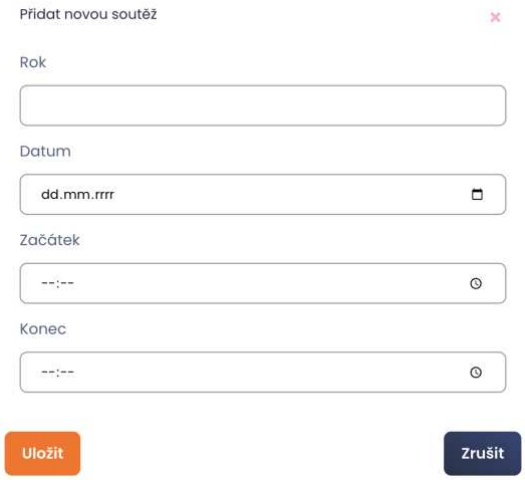
Obrázek 21 -AllTeams komponent

8.10 CompetitionManagement

Tato komponenta slouží vedení soutěže pro správu jednotlivých ročníků soutěže, zahrnuje operace jako přehledné zobrazení všech existujících ročníků, přidání nového ročníku nebo jeho úprava či smazaní.



Obrázek 22 - přehled soutěží (CompetitionManagement)



Obrázek 23 - modální okno pro tvorbu nové soutěže

8.10.1 Funkce a Metody

- **useState a useEffect**: Používají se pro uchování stavu soutěží a modálních oken a pro asynchronní načítání seznamu soutěží.
- **toggleModal a toggleModalEdit**: Funkce pro otevírání a zavírání modálních oken pro přidání a úpravu soutěží.
- **handleInputChange a handleInputChangeEdit**: Zpracovávají změny vstupních polí formulářů v modálních oknech.
- **handleSubmit a handleEditSubmit**: Funkce zpracovávají odeslání formulářů pro vytvoření a úpravu soutěží.

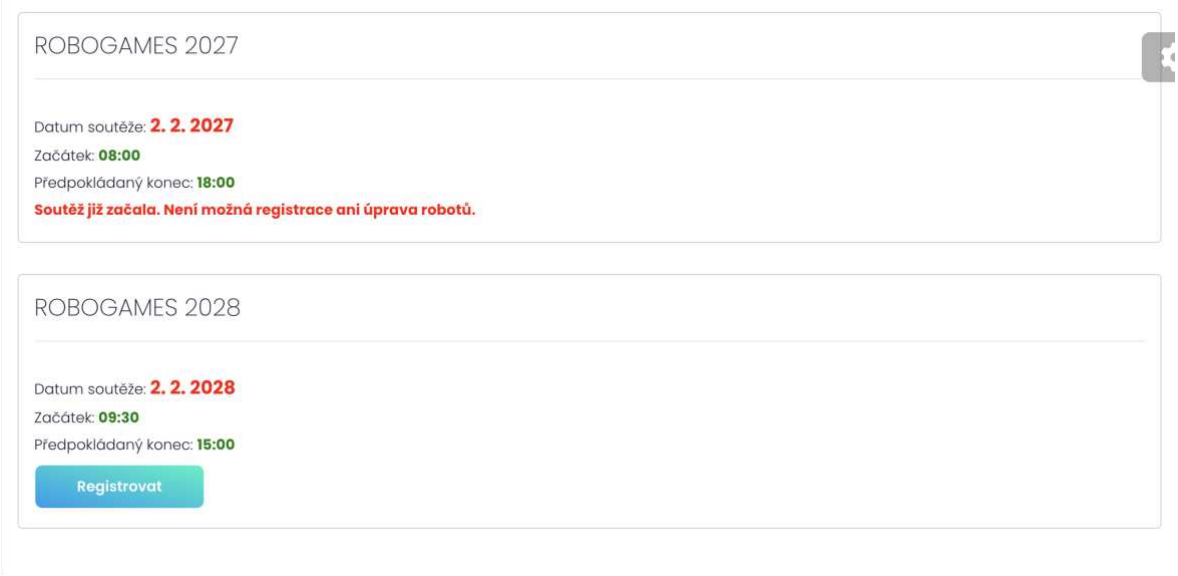
- **startCompetition a handleRemoveCompetition**: Umožňují spustit nebo odstranit soutěž.

8.10.2 JSX Kód

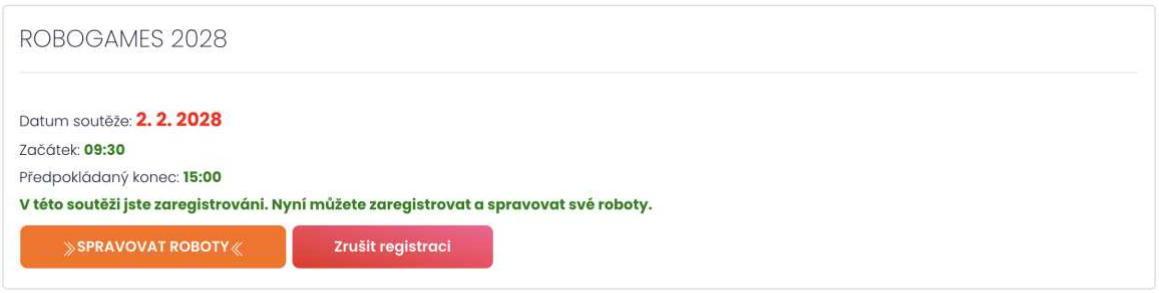
- **Karty a Tabulka**: Každá soutěž je zobrazena v samostatné kartě, která obsahuje informace jako ID, datumy, časy a stav soutěže.
- **Modální Okna**: Dvě modální okna slouží k přidání nové soutěže nebo úpravě existující. Obsahují formuláře se vstupními poli pro rok, datum, čas začátku a konec.
- **Tlačítka**: Pro každou soutěž jsou dostupná tlačítka pro zahájení, úpravu, odstranění a zobrazení účastníků.

8.11 CompetitionRegistration

Komponent slouží jako rozhraní pro soutěžící, kteří mají vytvořen svůj tým v systému a chtějí se zaregistrovat do soutěže. Kód funguje tak, že se ze serveru stáhnou informace o dostupných soutěžích a zobrazí se v kartách. Na základě informací přejatých z API se mění UI a informuje uživatele o datumu a času konání, zda soutěž již začala nebo tlačítko k registraci či zrušení registrace.
Pro zjištění, zda je tým v daném ročníku zaregistrován, se porovnávají data přijatá z API pro stažení informací o registracích do soutěže a rok konání dané soutěže. V případě, že je daný tým zaregistrován do ročníku, zobrazí se tlačítko „Spravovat roboty“.



Obrázek 24 - registrace do soutěže



Obrázek 25 - registrace byla úspěšná

8.11.1 Funkce a Metody

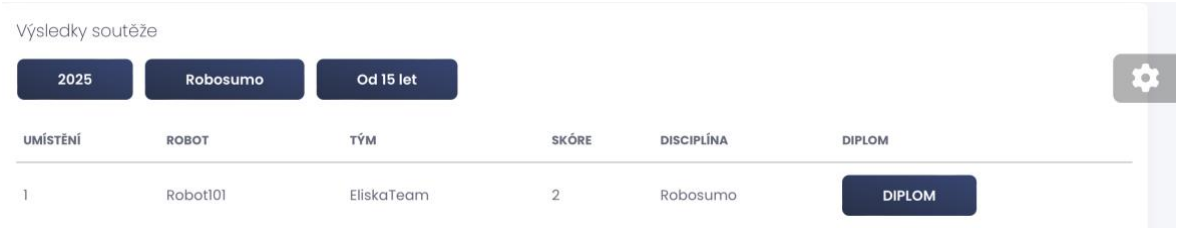
- **useState a useEffect**: Hooky pro správu stavu soutěží, registrací a načítání dat.
- **formatDate a formatTime**: Pomocné funkce pro formátování datumů a časů soutěží.
- **handleManageRobots**: Navigace na stránku pro registraci robotů s rokem soutěže jako parametrem.
- **unregisterTeam a registerTeam**: Funkce pro registraci a odregistraci týmu ze soutěže. Každá z těchto funkcí zahrnuje potvrzovací dialog a asynchronní volání API.

8.11.2 JSX Kód

- **Karty**: Každá soutěž je zobrazena v samostatné kartě, která obsahuje důležité informace o soutěži a tlačítka pro akce, jako je registrace, zrušení registrace a správa robotů.
- **Podmínkové zobrazení**: Zobrazení tlačítek pro akce je podmíněno stavem registrace uživatele ve soutěži a zda soutěž již začala.

8.12 CompetitionResults

Tato stránka zobrazuje výsledky soutěže, na základě uživatelsky zadaných parametrů. Jsou zde přitomny 3 filtry, které fungují na principu „dropdownu“ – uživatel je rozklikne a vybere si požadovaný ročník, disciplínu a kategorii. Poté je mu vygenerována vyfiltrovaná tabulka, která zobrazuje umístění soutěžících. Pokud je uživatel organizátor Robogames, zobrazí se další sloupec s tlačítkem pro vygenerování diplomu. Diplom je generován lokálně pomocí knihovny jsPDF, a je shodný s nejnovější šablonou diplomů udělovaných na Robogames soutěži.



Obrázek 26 - výsledky soutěže

Obrázek 27 - vygenerovaný diplom

```
doc.addFileToVFS('Roboto-Black-normal.ttf', font);
doc.addFont('Roboto-Black-normal.ttf', 'Roboto-Black', 'normal');
doc.setFont("Roboto-Black");


const img = new Image();
img.src = logo2;
img.onload = () => {

    doc.addImage(img, 'PNG', 40, 10, 110, 60);


    const imgFai = new Image();
    imgFai.src = failogo;
    imgFai.onload = () => {
        const currentYear = new Date().getFullYear();

        const currentDate = new Date();
        const formattedDate = `${currentDate.getDate()}.${currentDate.getMonth() + 1}.${currentDate.getFullYear()}`;

        doc.addImage(imgFai, 'PNG', 40, 70, 140, 30);


        doc.text("uděluje", doc.internal.pageSize.getWidth() / 2, 110, { align: "center" });
        doc.setFontSize(80);
        doc.text("DIPLOM", doc.internal.pageSize.getWidth() / 2, 140, { align: "center" });
        doc.setFontSize(15);
        doc.text("týmu", doc.internal.pageSize.getWidth() / 2, 160, { align: "center" });
        doc.setFontSize(42);
        doc.text(robot.teamName, doc.internal.pageSize.getWidth() / 2, 180, { align: "center" });
```

Obrázek 28 - kód generování diplomu

8.12.1 Funkce a Metody

- **useState a useEffect**: Slouží k uchování a správě stavů dat (soutěže, roky, disciplíny, výsledky) a načítání dat při změně zvoleného roku, disciplíny, nebo kategorie.

- **fetchCompetitionYears, fetchDisciplines, fetchResults**: Asynchronní funkce pro načítání dat o letech, disciplínách a výsledcích soutěží z API.

- **generatePDF**: Funkce pro generování PDF diplomů pro umístěné týmy s použitím knihovny **jsPDF**.

- **handleManageRobots, registerTeam, unregisterTeam**: Funkce pro správu registrací týmů a robotů.

JSX Kód

- **Dropdown Menu**: Umožňuje uživatelům vybrat rok, disciplínu a kategorii pro zobrazení výsledků.
- **Tabulka Výsledků**: Zobrazuje umístění, název robota, název týmu, skóre a disciplínu pro každý záznam výsledků. Pokud má uživatel příslušná oprávnění, zobrazuje se také tlačítko pro generování diplomů.

8.13 Contact

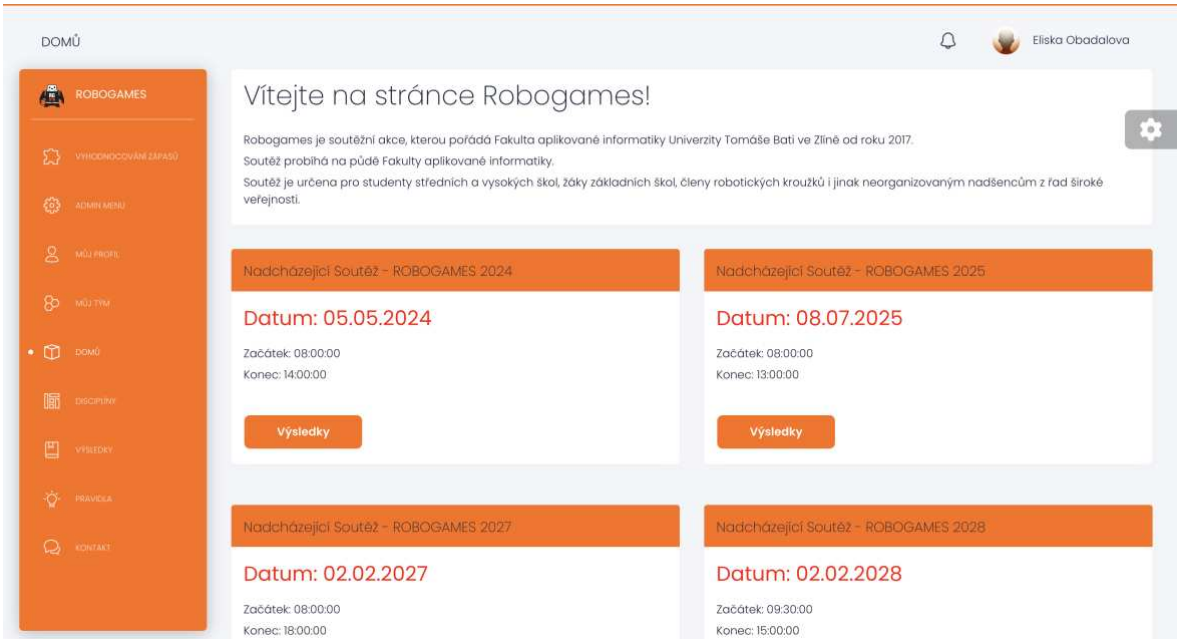
Stránka Contact je jednoduchý komponent složen ze dvou karet. Tyto karty uvádějí kontaktní informace na organizátory soutěže.

Obrázek 29 - karty pro vypsání kontaktů

8.14 Dashboard

Dashboard je domovská stránka aplikace, a obsahuje základní informace o soutěži. Zobrazí také aktivní ročníky soutěží, které mají proběhnout – datum začátku soutěže je větší než dnešní datum. Pokud je možné se do soutěže přihlásit, zobrazí se tlačítko **Registrovat** s odkazem na stránku **my-team**. V případě, že už soutěž

začala, zobrazí se tlačítko s odkazem na výsledky **competition-results**. Dále jsou v dolní části stránky zobrazeny loga sponzorů Robogames.



Obrázek 30 - domovská stránka

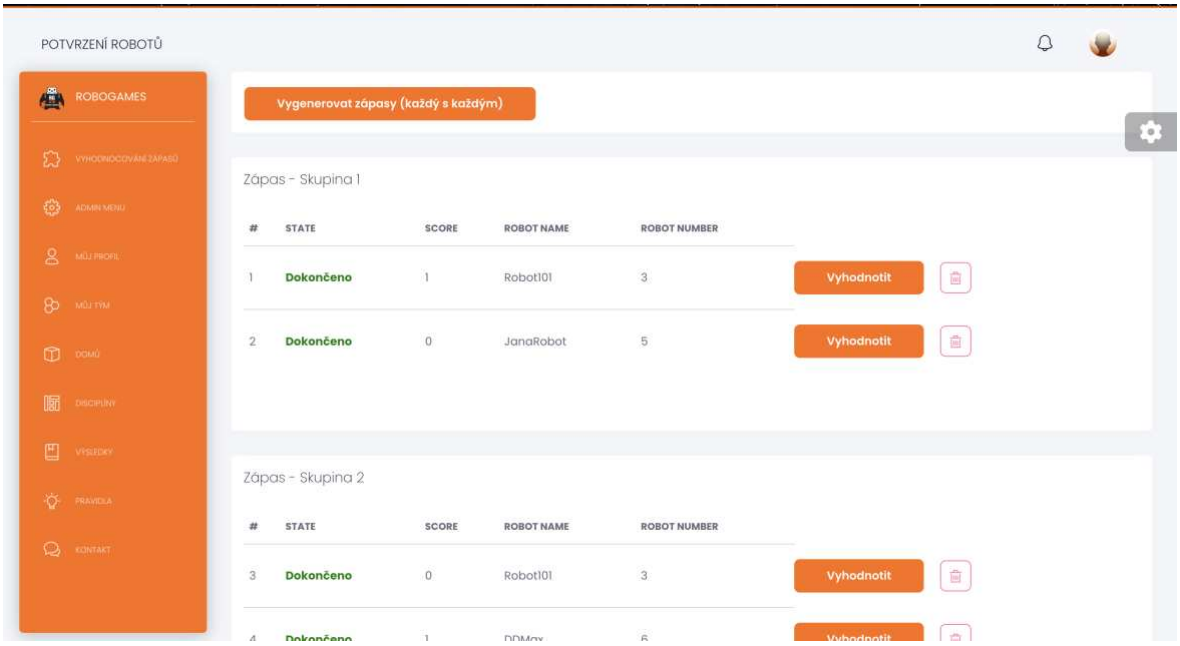
```
<Row>
    {competitions.map((comp, index) => (
        <Col md="6" key={index} style={{ marginBottom: '20px' }}>
            <Card>
                <CardHeader className="bg-primary">
                    <CardTitle tag="h4">Nadcházející Soutěž - ROBOGAMES {comp.year}</CardTitle>
                </CardHeader>
                <CardBody>
                    <h4 style={{ color: 'red', fontSize: '25px' }}>Datum: {formatDate(comp.date)}</h4>

                    <p>Začátek: {comp.startTime}</p>
                    <p>Konec: {comp.endTime}</p>
                </CardBody>
                {!comp.started && (
                    <CardFooter>
                        <Button color="primary" onClick={() => navigate('/admin/my-team')}>
                            Registrovat
                        </Button>
                    </CardFooter>
                )}
                {comp.started && (
                    <CardFooter>
                        <Button color="primary" onClick={() => navigate('/admin/competition-results')}>
                            Výsledky
                        </Button>
                    </CardFooter>
                )}
            </Card>
        </Col>
    ))}
```

Obrázek 31 - kód pro zobrazení soutěží

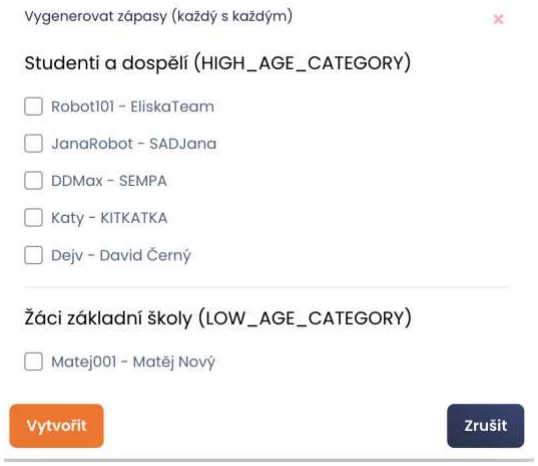
8.15 PlaygroundDetail

Zde je implementováno zobrazení, generování a vyhodnocení zápasů na určitém hřišti a ročníku. Na základě toho, pro jakou disciplínu je hřiště určeno se mění způsob generování zápasů. Pro generování zápasů musí rozhodčí kliknout na tlačítko, poté se objeví modální okno, kde zadá jednoho nebo více robotů, kteří mají zápasit. Zápasy se poté zobrazí v kartě společně se stavem zápasu a tlačítkem pro vyhodnocení. Toto tlačítko otevře další modální okno s jedním vstupem, do kterého se zapíše dosažené skóre. Rozhodčí má také možnost zápas vymazat.

Robosumo, Mini Robosumo – generování zápasů „všichni proti všem“
Linefollower, robot uklízeč – generování nových pokusů pro určitého robota



Obrázek 32 - robosumo zápasy



Obrázek 33 - výběr robotů na zápas

8.15.1 Funkce a Metody

- **useState a useEffect**: Slouží k správě stavů a asynchronnímu načítání dat.
- **fetchMatchesForPlayground, fetchPlaygroundDetails, fetchRobots**: Asynchronní funkce pro načítání dat o hřištích, robotech a zápasech.
- **handleMatchCreation**: Funkce pro vytváření zápasů.
- **handleRemoveMatch, handleRemoveAllGroupMatches**: Funkce pro odstranění zápasů a skupin zápasů.
- **handleScoreSubmit**: Funkce pro zadání skóre pro zápasy.
- **toggleModal**: Funkce pro ovládání modálních oken.

Tabulka Zápasů: Zobrazuje informace o zápasech včetně skóre, názvu robota a čísla robota.

Modální Okna: Slouží pro vytváření nových zápasů a zadávání skóre pro existující zápasy.

```
const handleScoreSubmit = async () => {
    const scoreValue = parseFloat(score);
    const apiUrl = `${process.env.REACT_APP_API_URL}api/match/writeScore?id=${selectedMatchId}&score=${scoreValue}`;

    try {
        const response = await fetch(apiUrl, {
            method: 'PUT',
            headers: {
                'Content-Type': 'application/json',
                'Authorization': `Bearer ${localStorage.getItem('token')}`
            }
        });
        const data = await response.json();
        if (response.ok && data.type === 'RESPONSE') {
            alert('Score successfully recorded.');
            setScoreModal(false); // Close the modal

            window.location.reload();
        } else {
            throw new Error(data.data || 'Unknown error occurred');
        }
    } catch (error) {
        alert(`Error submitting score: ${error.message || 'Failed to communicate with the server'}`);
    }
};
```

Obrázek 34 - ukázka kódu pro odeslání hodnocení zápasu

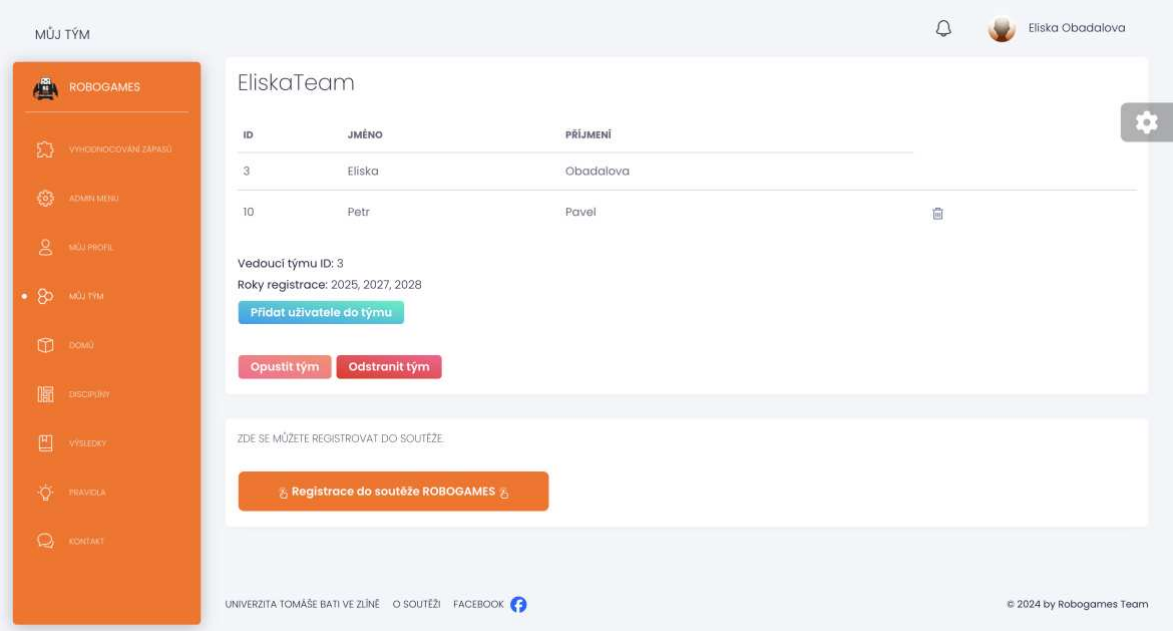
8.16 MatchManagement

MatchManagement slouží jako rozhraní pro rozhodčí na výběr ročníku soutěže a hřiště, na kterém mají zapisovat výsledky. Pomocí API jsou načteny ročníky soutěže a jednotlivá hřiště. Potom co rozhodčí vybere rok a klikne na některé z hřišť, je navigován na stránku PlaygroundDetail společně s parametry query string: year – rok soutěže, id – id hřiště.

Obrázek 35 - výběr hřiště

8.17 MyTeam

Tato stránka nabízí informace o týmu přihlášeného uživatele. V případě, že žádný tým nemá, je o této skutečnosti informován a je mu nabídnuta možnost jej vytvořit. Pokud uživatel má tým, je zde přehled o členech týmu a možnost z týmu odejít. Vedoucí týmu má oprávnění poslat pozvánku jinému uživateli výběrem ze seznamu. V tomto sezamu jsou uživatelé bez žádného týmu. Dále vedoucí disponuje tlačítkem **Odstranit tým**.



Obrázek 36 - stránka správy týmu

Obrázek 37 - vyhledávání uživatele

Je zde přítomna také sekce pro registraci do soutěže, která uživatele převede na stránku competition-registration, kde dále pokračuje proces registrace.

8.17.1 Funkce a Metody

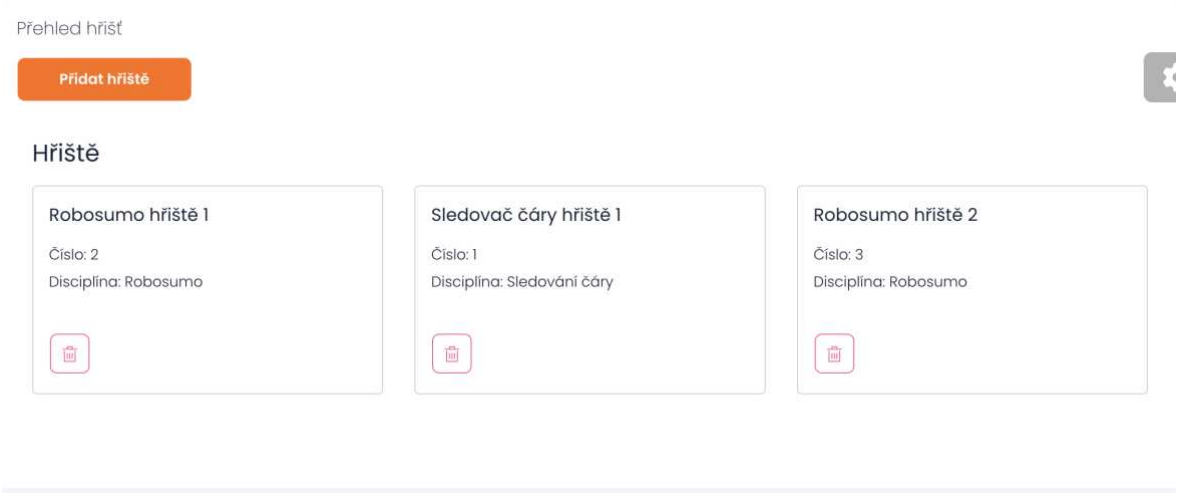
- **useState a useEffect**: Hooky pro správu stavu (např. tým, chyby, načítání) a načítání dat z API.
- **fetchMyTeam**: Asynchronní funkce pro načítání informací o týmu uživatele.
- **handleCreateTeam**: Funkce pro vytvoření nového týmu.
- **handleAddUser, removeMember**: Funkce pro správu členů týmu.
- **leaveTeam**: Funkce umožňující uživateli opustit tým.
- **handleRemoveTeam**: Funkce pro odstranění celého týmu.

8.17.2 JSX Kód

- **Modální Okna**: Používána pro vytvoření týmu a vyhledávání uživatelů.
- **Tabulka**: Zobrazuje členy týmu s možností odstranění, pokud je uživatel vedoucím týmu.
- **Tlačítka**: Akce jako vytvoření týmu, opuštění týmu, přidání členů, a odstranění týmu jsou dostupné prostřednictvím tlačítek.

8.18 PlaygroundManagement

PlaygroudManagement komponent slouží pro zobrazení hřišť a jejich správu. Stránka je určena pro administrátory a umožnuje tvorbu a smazání. Po kliknutí na Přidat hřiště se zobrazí modální okno s formulářem a po validaci zadaných dat se požadavek o vytvoření pošle na backendový server.



Obrázek 38 - přehled existujících hřišť

Obrázek 39 - formulář pro tvorbu hřiště

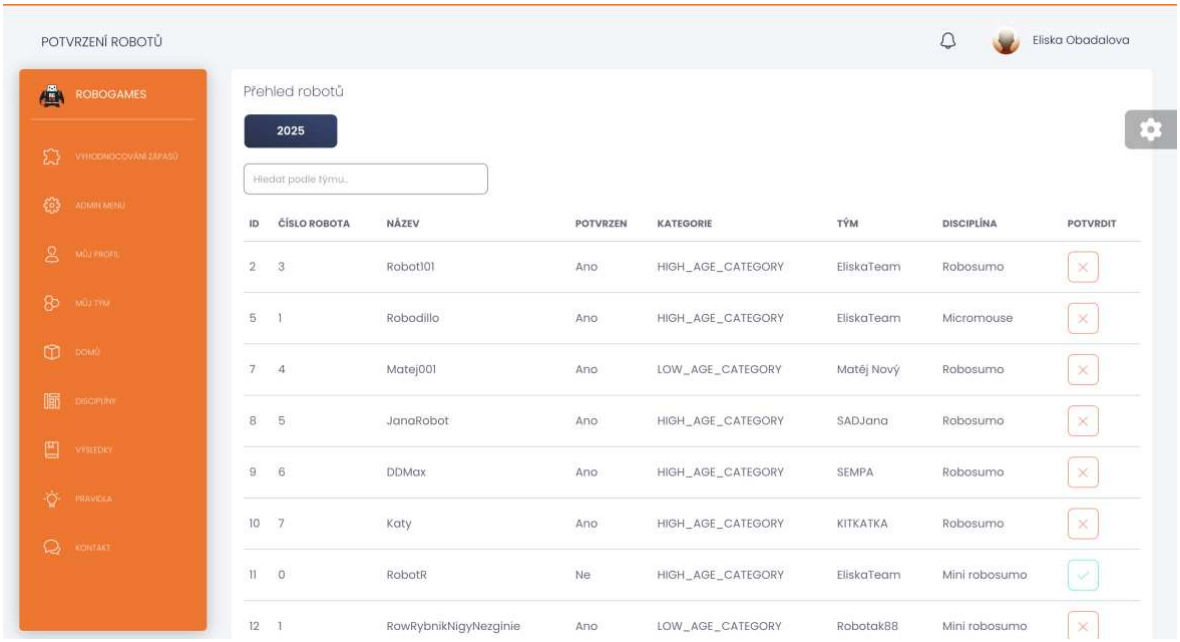
8.18.1 Funkce a Metody

- **useState a useEffect**: Slouží k správě stavů, jako jsou hřiště, disciplíny, načítání dat a řízení viditelnosti modálních oken.
- **fetchPlaygrounds a fetchDisciplines**: Asynchronní funkce pro načítání seznamu hřišť a disciplín.
- **handleInputChange**: Zpracovává změny ve formulářových polích a aktualizuje stav komponenty.
- **handleSubmit**: Odesílá požadavek na server pro přidání nového hřiště.

- **handleRemovePlayground**: Odesílá požadavek na server pro odstranění hřiště.
- **toggleModal a toggleDropdown**: Funkce pro ovládání viditelnosti modálních oken a dropdown menu.

8.19 RobotConfirmation

Na této stránce organizátoři mohou potvrdit roboty, kteří se na soutěž registrovali. Bez potvrzení se robot nemůže účastnit soutěže. Umožňuje potvrzovat nebo zamítat registrace robotů pro vybraný rok soutěže a nabízí rozhraní pro filtrování robotů podle názvu týmu.



Obrázek 40 - potvrzovaní robotů

8.20 RobotRegistration

8.20.1 Funkce a Metody

- **Správa stavu**: Používá React hooky **useState** pro uchování stavů, jako jsou roky soutěží, vybraný rok, roboti, vyhledávací termín a viditelnost modálních oken.
- **Effect hooky**:
  - **useEffect** pro načtení let soutěží ihned po načtení komponenty.

- Další **useEffect** reaguje na změnu vybraného roku a načítá roboty pro daný rok.

- **Event Handlery**:
  - **handleConfirmRegistration**: Zpracovává potvrzení nebo zamítnutí registrace robota. Před provedením změny stavu se zobrazí potvrzovací dialog.
  - **fetchCompetitionYears** a **fetchRobotsForYear**: Asynchronní funkce pro načítání dostupných let soutěží a seznamu robotů pro zvolený rok.
  - **toggleModal** a **toggleDropdown**: Funkce pro ovládání viditelnosti modálních oken a rozbalovacích seznamů.

8.20.2 UI/UX

- **Dropdown**: Umožňuje uživateli vybrat rok, pro který chce zobrazit roboty.
- **Tabulka**: Zobrazuje seznam robotů s možnostmi potvrdit nebo zamítnout jejich registraci.
- **Vyhledávací pole**: Umožňuje filtraci robotů podle názvu týmu.

8.21 Rules

Stránka Rules zobrazuje v přehledném formátu všechny pravidla soutěže a disciplín. Jednotlivá témata mají svůj vlastní otevíratelný oddíl.

8.21.1 Struktura a funkce komponenty

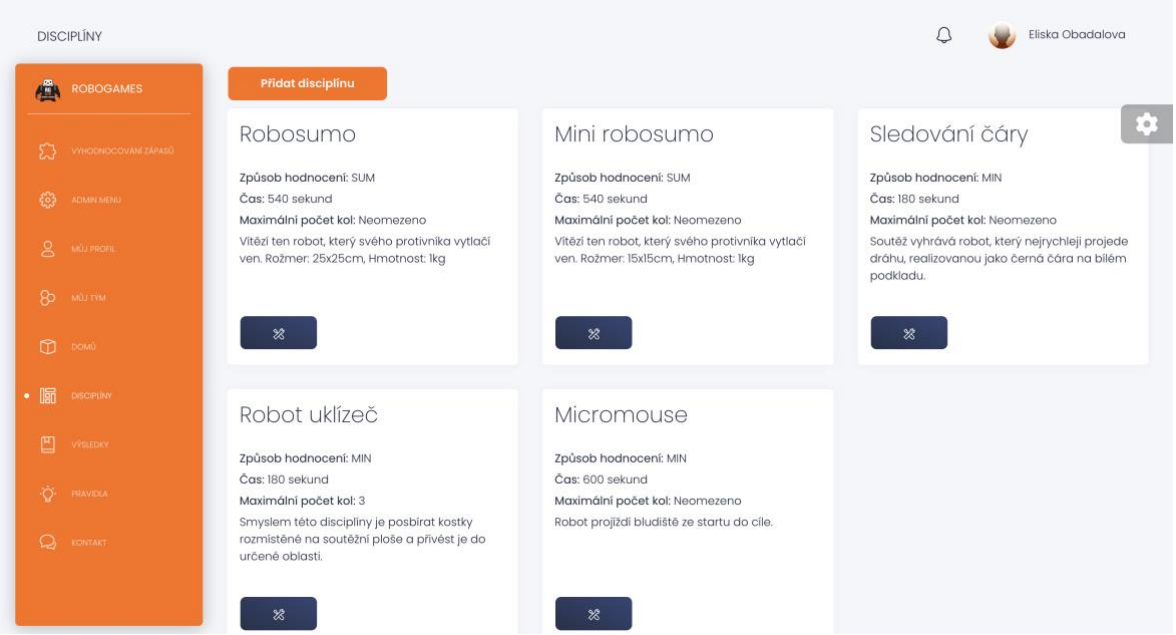
- **useState**:
  - **isOpen**: Stav pro správu viditelnosti jednotlivých sekce pravidel. Objekt, kde klíče odpovídají identifikátorům sekce a hodnoty jsou boolean určující, zda je sekce otevřená či zavřená.
- **toggle**:
  - Funkce pro přepínání viditelnosti sekce. Změní stav **isOpen** pro konkrétní sekci na opačnou hodnotu (otevřeno/zavřeno).
- **useEffect**:
  - Nejsou zde žádné useEffect hooky, což naznačuje, že data pravidel nejsou dynamicky načítána z externího zdroje při načítání komponenty.

8.21.2 JSX kód

- **Card, CardHeader, CardBody, CardTitle**:
  - Používají se pro strukturované zobrazení obsahu.
- **Button**:
  - Pro každou sekci pravidel je přiřazen tlačítko, které umožňuje rozbalení detailů této sekce.
- **Collapse**:
  - Rozbalovací komponenta, která zobrazuje obsah sekce na základě toho, zda je sekce aktivní (**isOpen**).

8.22 Disciplines

Zde jsou vypsány všechny dostupné disciplíny. Běžný uživatel uvidí pouze informace k jednotlivým disciplínám, zatímco uživatel s rolí ADMIN a LEADER uvidí i tlačítka pro přidání discíplíny, úpravu či smazání.



Obrázek 41 - správa disciplín

8.22.1 Struktura a funkce

- **useState**
  - **disciplines**: Ukládá seznam všech disciplín načtených z API.
  - **modal**: Ovládá viditelnost modálního okna pro přidání nebo úpravu disciplín.
  - **editMode**: Určuje, zda je modální okno v režimu úpravy existující disciplíny.
  - **formData**: Obsahuje data formuláře pro vytváření nebo úpravu disciplíny.
  - **openDropdownId**: Udržuje ID rozbaleného dropdown menu pro konkrétní disciplínu.

- **useEffect**
  - Dva useEffect hooky slouží k načtení disciplín při inicializaci komponenty. Tyto hooky využívají asynchronní volání k API pro získání dat.

8.22.2 Funkce:

- **toggleDropdown** a **toggleModal**: Funkce pro přepínání stavu dropdown menu a modálního okna.

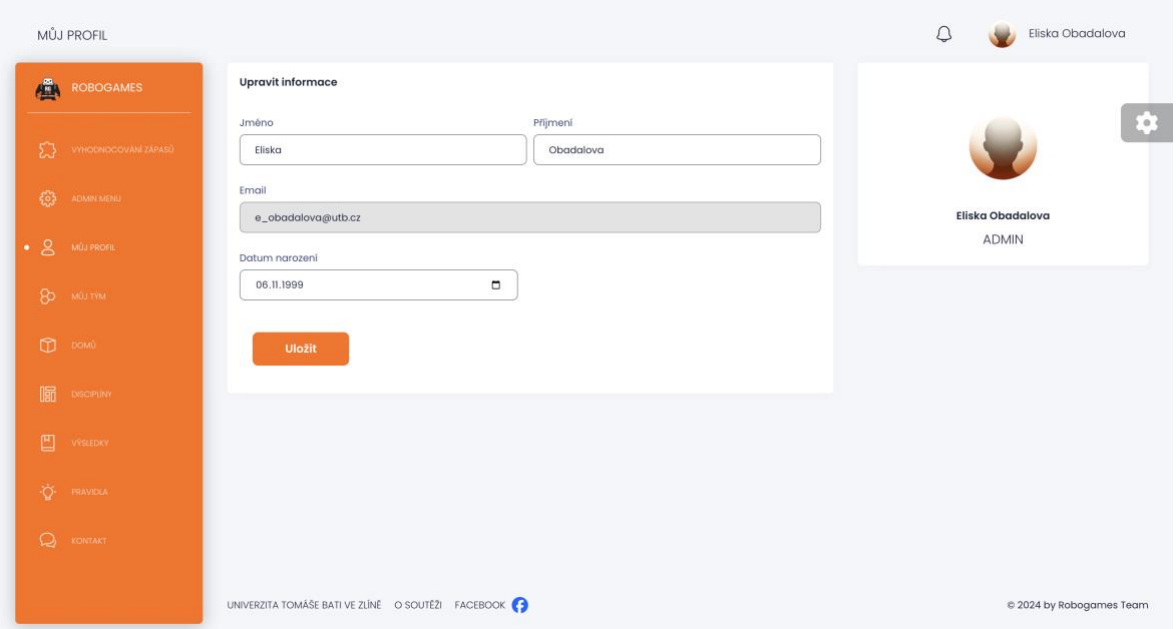
- **handleInputChange**: Zpracovává změny ve formulářových polích a aktualizuje **formData**.
- **handleEdit**: Nastaví data formuláře pro úpravu a otevře modální okno v režimu úpravy.
- **handleRemove**: Odstraňuje disciplínu z API a aktualizuje stav **disciplines**.
- **handleSubmit**: Odesílá upravená nebo nová data do API a aktualizuje seznam disciplín.

8.22.3 UI komponenty

- **Card, CardHeader, CardBody, CardFooter**: Slouží k vizuální prezentaci jednotlivých disciplín.
- **Modal**: Modální okno pro přidání nebo úpravu disciplíny.
- **FormGroup, Label, Input, Button**: Formulářové prvky pro zadávání dat disciplíny.
- **Dropdown**: Nabídka s možnostmi pro úpravu nebo odstranění disciplíny, dostupná pouze pro administrátory a vedoucí týmů.

8.23 UserProfile

Komponenta **UserProfile** v Reactu slouží k zobrazení a úpravě uživatelských informací. Uživatelé mohou aktualizovat své jméno, příjmení a datum narození.



Obrázek 42 - uživatelský profil

8.23.1 Funkce a logika komponenty

Stavové proměnné:

- **userData**: Objekt obsahující aktuální data uživatele.
- **initialUserData**: Kopie dat uživatele načtených při prvním zavolání pro porovnání změn před odesláním.
- **modal**: Boolean určující, zda je modální okno otevřené nebo zavřené.
- **useEffect:**
- První useEffect se spustí při montáži komponenty a načítá uživatelská data z API.
- Veškerá komunikace s API je zabezpečená pomocí tokenu získaného z localStorage.

Formulářové funkce:

- **handleSubmit**: Funkce, která se spustí při odeslání formuláře. Zkontroluje změny v údajích a pokud jsou změny nalezeny a všechny povinné pole jsou vyplněné, odešle aktualizovaná data na server.
- **handleInputChange**: Aktualizuje stav **userData** při změně hodnot vstupních polí formuláře.

8.23.2 UI Struktura

- **Card**: Slouží jako kontejner pro formulář a profilové informace.
- **FormGroup a Input**: Používají se pro vytvoření interaktivních formulářových polí.
- **Button**: Tlačítko pro odeslání formuláře.
- Modální okno pro editaci, které se zobrazí po kliknutí na tlačítko, není v této komponentě přítomno – setModal a **modal** jsou definovány, ale nejsou použity.

9 ÚPRAVA SERVEROVÉ ČÁSTI

9.1 Věkové kategorie

Tato část vysvětluje, jak byla upravena serverová část tak, aby výsledné kategorie byly dvě – **do 15** a **od 15 let**.

Nejprve bylo nutné přepsat dosavadní kategorie v souboru *src/main/java/com/robogames/RoboCupMS/Business/Enum/ECategory.java*

Zde byly vytvořeny dvě nové ENUM kategorie:

- **LOW_AGE_CATEGORY** (Do 15 let)
- **HIGH_AGE_CATEGORY** (Od 15 let)

```
/**
 * Kategorie tymu
 */
public enum ECategory {
    /**
     * Nizka vekova kategorie (defaultne nastaveno max do 15 let)
     */
    LOW_AGE_CATEGORY,

    /**
     * Vyskova vekova kategorie (defaultne nad 15 let)
     */
    HIGH_AGE_CATEGORY;

}
```

Obrázek 43 - změna soubotu ECategory.java

Dále bylo třeba upravit definici maximálního věku v kategorii v souboru *src/main/java/com/robogames/RoboCupMS/AppInit.java*. Zde stačilo odebrat původní definice pro původní kategorie a nahradit novou. Poté následovalo přidání nové konfigurační hodnoty **LOW_AGE_CATEGORY_MAX_AGE** v souboru *config.json,* která byla nastavena na 15.

```
// maximalni vek pro nizsi vekovou kategorii
Long LOW_AGE_CATEGORY_MAX_AGE = (Long) obj.get("LOW_AGE_CATEGORY_MAX_AGE");
if (LOW_AGE_CATEGORY_MAX_AGE != null) {
    GlobalConfig.LOW_AGE_CATEGORY_MAX_AGE = (int) LOW_AGE_CATEGORY_MAX_AGE.longValue();
    logger.info("LOW_AGE_CATEGORY_MAX_AGE set on: " + LOW_AGE_CATEGORY_MAX_AGE);
}
```

Obrázek 44 - změna AppInit.java

```
/**
 * Prvni inicializace kategorii
 *
 * @param repository RoleRepository
 */
@Bean
public ApplicationRunner initCategory(CategoryRepository repository) {
    if (repository.count() == 0) {
        return args -> repository.saveAll(Arrays.asList(
                new Category(ECategory.LOW_AGE_CATEGORY),
                new Category(ECategory.HIGH_AGE_CATEGORY)));
    } else {
        return null;
    }
}
```

Obrázek 45 - změna inicializovaných kategorií

Další úprava se týkala entity Team v souboru *src/main/java/com/robogames/RoboCupMS/Entity/Team.java,* kde byla upravena funkce **determinateCategory()**, která slouží k rozřazení týmu do kategorie podle věku nejstaršího člena.

```java
/**
 * Urci kategorii tymu, ve ktere bude soutezit (dle vekou soutezicich)
 *
 * @return Soutezni kategorie
 */
public ECategory determinateCategory() {
    List<UserRC> users = this.getMembers();

    // vypocet veku nejstarsiho uzivatele v tymu
    int max_age = 0;
    for (UserRC u : users) {
        max_age = Math.max(max_age, u.getAge());
    }

    // podle veku urci kategorii
    if (max_age <= GlobalConfig.LOW_AGE_CATEGORY_MAX_AGE) {
        return ECategory.LOW_AGE_CATEGORY;
    } else {
        return ECategory.HIGH_AGE_CATEGORY;
    }
}
```

Obrázek 46 - úprava rozřazení týmu do kategorií

9.2 Systém pozvánek do týmu

Pro vhodnější způsob přidávání uživatelů do týmu byl přidán nový systém pozvánek. Cílem bylo vytvořit novou funkcionalitu, kde vedoucí týmu pošle pozvánku uživateli, a ten ji bude moci přijmout anebo odmítnout. To nejdříve vyžadovalo tvorbu nové entity v systému *src/main/java/com/robogames/RoboCupMS/Entity/TeamInvitation.java.*

```java
@Entity(name = "team_invitation")
public class TeamInvitation {

    @Id
    @GeneratedValue
    private Long id;

    @OneToOne
    @JoinColumn(name = "user_id", referencedColumnName = "id")
    private UserRC leader;

    @ManyToOne
    private Team team;

    public TeamInvitation() {
    }

    public TeamInvitation(Long id, UserRC leader, Team team) {
        this.id = id;
        this.leader = leader;
        this.team = team;
    }

    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }

    public UserRC getLeader() {
        return leader;
    }
```

Obrázek 47 - nová entita TeamInvitation

9.2.1.1 Popis kódu

- **Entity Annotation**: Anotace **@Entity(name = "team_invitation")** označuje, že tato třída je mapována na tabulku **team_invitation** v databázi. Slouží k mapování objektů na databázové tabulky.

- **Identifikátor**: Atribut **id** s anotacemi **@Id** a **@GeneratedValue** slouží jako primární klíč entity v databázi.

- **Relace s dalšími entitami**:

- **UserRC**: Atribut **user** má anotace **@OneToOne** a **@JoinColumn(name = "user_id", referencedColumnName = "id")**. Relace je specifikována tak, že sloupec **user_id** v tabulce **team_invitation** odkazuje na **id** v tabulce přidružené k entitě **User**.
- **Team**: Atribut **team** má anotace **@ManyToOne**, což znamená, že jedna instance **TeamInvitation** může být spojena s jedním **Team**. To znamená, že více pozvánek může odkazovat na jeden tým.

9.2.1.2 Konstruktory

- Třída obsahuje dva konstruktory:
  - Bezparametrický konstruktor (**public TeamInvitation()**)
  - Parametrický konstruktor (**public TeamInvitation(Long id, UserRC user, Team team)**), který umožňuje inicializaci nové instance **TeamInvitation** s konkrétními hodnotami.

9.2.1.3 Metody

- Gettery a settery (**getId**, **setId**, **getUser**, **setUser**, **getTeam**, **setTeam**) umožňují manipulaci s atributy instance **TeamInvitation**. Tato metody umožňují snadný přístup a aktualizaci hodnot atributů.

9.2.1.4 Úprava UserService.java

Nová metoda v souboru *src/main/java/com/robogames/RoboCupMS/Business/Service/UserService.java* **getTeamInvitations()** je navržena k získání seznamu všech týmových pozvánek (**TeamInvitation**), které byly odeslány aktuálně přihlášenému uživateli.

```
public List<TeamInvitation> getTeamInvitations() {
    UserRC user = (UserRC) SecurityContextHolder.getContext().getAuthentication().getPrincipal();

    List<TeamInvitation> invitations = invitationRepository.findAll().stream()
            .filter(e -> e.getUser().getID() == user.getID()).collect(Collectors.toList());

    return invitations;
}
```

Obrázek 48 - úprava UserService controlleru

Pro přidání endpointu pro získání pozvánek daného uživatele byl upraven soubor *src/main/java/com/robogames/RoboCupMS/Controller/UserControler.java.*

Metoda **getTeamInvitations**, anotovaná **@GetMapping("/getTeamInvitations")**, je určena k poskytování REST API endpointu v naší webové aplikaci. Tato metoda je součástí controlleru, který využívá Spring MVC framework, a slouží k získání seznamu týmových pozvánek pro aktuálně přihlášeného uživatele a jejich převodu na formát, který je vhodný pro přenos přes síť.

```
@GetMapping("/getTeamInvitations")
Response getTeamInvitations() {
    List<TeamInvitationObj> all = new ArrayList<TeamInvitationObj>();
    for (TeamInvitation inv : this.userService.getTeamInvitations()) {
        all.add(new TeamInvitationObj(inv.getId(), inv.getUser().getID(), inv.getTeam().getID(),
                inv.getTeam().getName()));
    }
    return ResponseHandler.response(all);
}
```

Obrázek 49 - přidání endpointu getTeamInvitations

9.2.1.5 API endpointy pro přidání uživatele a přijmutí/odmítnutí pozvánky

Dalším krokem byla tvorba API pro přidání nového člena do týmu a zamítnutí či přijetí pozvánky. Ty byly přidány do controlleru spravující tým *src/main/java/com/robogames/RoboCupMS/Controller/TeamControler.java.*

```java
/**
 * Prida do tymu noveho clena
 *
 * @param id ID clena, ktery ma byt pridat do tymu
 * @return Informace o stavu provedeneho requestu
 */
@PutMapping("/addMember")
Response addMember(@RequestParam Long id) {
    try {
        this.teamService.addMember(id);
        return ResponseHandler.response(data:"success");
    } catch (Exception ex) {
        return ResponseHandler.error(ex.getMessage());
    }
}

/**
 * Odebere z tymu jednoho clena
 *
 * @param id ID clena, ktery ma byt odebran z tymu
 * @return Informace o stavu provedeneho requestu
 */
@PutMapping("/removeMember")
Response removeMember(@RequestParam Long id) {
    try {
        this.teamService.removeMember(id);
        return ResponseHandler.response(data:"success");
    } catch (Exception ex) {
        return ResponseHandler.error(ex.getMessage());
    }
}

@PutMapping("/acceptInvitation")
Response acceptInvitation(@RequestParam Long id) {
    try {
        this.teamService.acceptInvitation(id);
        return ResponseHandler.response(data:"success");
    } catch (Exception ex) {
        return ResponseHandler.error(ex.getMessage());
    }
}

@PutMapping("/rejectInvitation")
Response rejectInvitation(@RequestParam Long id) {
    try {
        this.teamService.acceptInvitation(id);
        return ResponseHandler.response(data:"success");
    } catch (Exception ex) {
        return ResponseHandler.error(ex.getMessage());
    }
}
```

Obrázek 50 - API endpointy pro pozvánky

1. /**addMember**:
    - **Operace**: Přidání člena do týmu.
    - **Parametr**: **id** (typu **Long**), který identifikuje uživatele nebo člena, který má být přidán do týmu.

- **Proces**: Volání metody **addMember(id)** v **teamService**, která zpracuje přidání uživatele do týmu.
- **Odpověď**: Pokud operace proběhne úspěšně, vrátí "success". V případě výjimky vrátí chybovou zprávu.

2. /**removeMember**:
    - **Operace**: Odebrání člena z týmu.
    - **Parametr**: **id** (typu **Long**), který identifikuje člena, který má být z týmu odebrán.
    - **Proces**: Volání metody **removeMember(id)** v **teamService**, která zpracuje odebrání uživatele z týmu.
    - **Odpověď**: Podobně jako u /**addMember**, endpoint vrátí "success" při úspěchu a chybovou zprávu při selhání.

3. /**acceptInvitation**:
    - **Operace**: Přijetí týmové pozvánky.
    - **Parametr**: **id** (typu **Long**), který identifikuje pozvánku, jež má být přijata.
    - **Proces**: Metoda **acceptInvitation(id)** v **teamService** zpracuje přijetí pozvánky.
    - **Odpověď**: Endpoint vrátí "success" po úspěšném přijetí pozvánky nebo chybovou zprávu, pokud dojde k nějaké chybě.

4. /**rejectInvitation**:
    - **Operace**: Odmítnutí týmové pozvánky.
    - **Parametr**: **id** (typu **Long**), který identifikuje pozvánku, jež má být odmítnuta.
    - **Proces**: Metoda **rejectInvitation(id)** v **teamService** zpracuje odmítnutí pozvánky.
    - **Odpověď**: Podobně jako ostatní endpointy, vrátí "success" pokud operace proběhne hladce, a chybovou zprávu v případě výjimky.

Posledním krokem bylo přidání souvisejících servisních metod v souboru *src/main/java/com/robogames/RoboCupMS/Business/Service/TeamService.java*, které budou přijatá data z endpointů zpracovávat.

```
public void acceptInvitation(Long id) throws Exception {
    Optional<TeamInvitation> invitation = this.invitationRepository.findById(id);
    if (invitation.isPresent()) {
        UserRC currentUser = (UserRC) SecurityContextHolder.getContext().getAuthentication().getPrincipal();
        UserRC u = invitation.get().getUser();
        Team t = invitation.get().getTeam();

        // pokud pozvanka nepatri uzivateli, kteremu realne byla odeslana
        if(currentUser != u) {
            throw new Exception("failure, this is not your invitation");
        }

        // prida uzivatele do tymu a ulozi zmeni v databazi pro tym i pro uzivatele
        t.getMembers().add(u);
        u.setTeam(t);
        this.teamRepository.save(t);
        this.userRepository.save(u);

        // odstraneni z databaze
        this.invitationRepository.delete(invitation.get());
    } else {
        throw new Exception(String.format("failure, invitaton with ID [%s] not found", id));
    }
}

public void rejectInvitation(Long id) throws Exception {
    Optional<TeamInvitation> invitation = this.invitationRepository.findById(id);
    if (invitation.isPresent()) {
        UserRC currentUser = (UserRC) SecurityContextHolder.getContext().getAuthentication().getPrincipal();
        UserRC u = invitation.get().getUser();

        // pokud pozvanka nepatri uzivateli, kteremu realne byla odeslana
        if(currentUser != u) {
            throw new Exception("failure, this is not your invitation");
        }

        // odstraneni z databaze
        this.invitationRepository.delete(invitation.get());
    } else {
        throw new Exception(String.format("failure, invitaton with ID [%s] not found", id));
    }
}
```

Obrázek 51 - nové metody accept/rejectInvitation

9.2.1.6 Metoda acceptInvitation

- **Parametr**: **id** typu **Long**, který identifikuje konkrétní týmovou pozvánku.
- **Průběh**:
    - Načtení pozvánky z databáze pomocí **invitationRepository.findById(id)**.
    - Kontrola, zda pozvánka existuje. Pokud neexistuje, vyhodí výjimku.
    - Ověření, že aktuálně přihlášený uživatel (získaný z kontextu Spring Security) je uživatel, kterému byla pozvánka skutečně určena.
    - Přidání uživatele do týmu a aktualizace odpovídajících záznamů v databázi:

- Uživatel je přidán do seznamu členů týmu.
- Uživatelově entitě je nastaven příslušný tým.
- Změny jsou uloženy do databáze pomocí **teamRepository** a **userRepository**.

- Po úspěšném zpracování je pozvánka odstraněna z databáze.

9.2.1.7 Metoda rejectInvitation

- **Parametr**: **id** typu **Long**, který identifikuje konkrétní týmovou pozvánku.
- **Průběh**:
  - Načtení pozvánky z databáze pomocí **invitationRepository.findById(id)**.
  - Kontrola, zda pozvánka existuje. Pokud neexistuje, vyhodí výjimku.
  - Ověření, že aktuálně přihlášený uživatel (získaný z kontextu Spring Security) je uživatel, kterému byla pozvánka skutečně určena.
  - Pozvánka je odstraněna z databáze.

```java
/**
 * Prida do tymu noveho clena
 *
 * @param id ID clena, ktery ma byt pridat do tymu
 * @throws Exception
 */
public void addMember(Long id) throws Exception {
    UserRC leader = (UserRC) SecurityContextHolder.getContext().getAuthentication().getPrincipal();

    Optional<Team> t = this.teamRepository.findAllByLeader(leader).stream().findFirst();
    if (t.isPresent()) {
        // overi zda nebyl jiz prekrocen pocet clenu v tymu
        if (t.get().getMembers().size() >= GlobalConfig.MAX_TEAM_MEMBERS) {
            throw new Exception("failure, team is full");
        }

        Optional<UserRC> u = this.userRepository.findById(id);
        if (u.isPresent()) {

            // vytvori pozvanku do tymu
            TeamInvitation invitation = new TeamInvitation();
            invitation.setUser(u.get());
            invitation.setTeam(t.get());
            this.invitationRepository.save(invitation);

        } else {
            throw new Exception(String.format("failure, user with ID [%s] not found", id));
        }
    } else {
        throw new Exception("failure, you are not the leader of any existing team");
    }
}
```

Obrázek 52 - metoda addMember

Popis metody addMember:

Parametry a výjimky

- **Parametr**: **id** - ID uživatele, který má být přidán do týmu.
- **Výjimka**: Metoda může vyhodit výjimku v případě chyb.

Kódové bloky a jejich funkce

Získání informací o aktuálně přihlášeném uživateli:

- **UserRC leader = (UserRC) SecurityContextHolder.getContext().getAuthentication().getPrincipal();**
- Toto načítá aktuálně přihlášeného uživatele jako leadera týmu z kontextu Spring Security.

Načtení týmu vedeného leaderem:

- Optional<Team> t = this.teamRepository.findAllByLeader(leader).stream().findFirst();
- Hledá tým v databázi, který je veden uvedeným leaderem, a pokud existuje, bere první takový tým.

Kontrola kapacity týmu:

- Pokud je tým nalezen, zkontroluje, zda počet členů týmu nepřekročil maximální povolený počet (definovaný v **GlobalConfig.MAX_TEAM_MEMBERS**).
- Pokud je tým plný, vyvolá výjimku.

Hledání uživatele podle ID:

- **Optional<UserRC> u = this.userRepository.findById(id);**
- Pokusí se najít uživatele v databázi podle ID.

Vytvoření a uložení pozvánky:

- Pokud je uživatel nalezen, vytvoří se nová týmová pozvánka (**TeamInvitation**) a nastaví se s referencemi na uživatele a tým.
- Pozvánka je uložena do databáze.

Ošetření chyb:

- Pokud uživatel není nalezen, vyvolá se výjimka s chybovým hlášením, že uživatel s daným ID nebyl nalezen.
- Pokud aktuálně přihlášený uživatel není leaderem žádného týmu, vyvolá se výjimka.

9.3 Další přidané API endpointy

Pro potřeby vývoje frontendové aplikace se vyskytla nutnost pro API navracející všechny registrované roboty pro daný ročník soutěže z důvodu komponentu řešícího potvrzení robota organizátory soutěže.

Byl přidán úpravou souborů *src/main/java/com/robogames/RoboCupMS/Business/Service/RobotService.java* a

src/main/java/com/robogames/RoboCupMS/Controller/RobotControler.java
vytvořením nového endpointu a servisní metody v kontroleru.

```
public List<Robot> allForYear(int year) throws Exception {
    Stream<Robot> robots = this.robotRepository.findAll().stream()
            .filter((r) -> (r.getTeamRegistration().getCompatitionYear() == year));
    return robots.collect(Collectors.toList());
}
```

Obrázek 53 - nová servisní metoda allForYear()

```
/**
 * Navrati vsehcny roboty pro dany rocnik
 *
 * @param year Rocnik souteze
 * @return Seznam robotu s potvrzenou registraci
 */
@GetMapping("/allForYear")
Response allForYear(@RequestParam int year) {
    List<Robot> robots;
    try {
        robots = this.robotService.allForYear(year);
    } catch (Exception ex) {
        return ResponseHandler.error(ex.getMessage());
    }
    return ResponseHandler.response(robots);
}
```

Obrázek 54 - přidání nového GET endpointu

ZÁVĚR

V této diplomové práci jsem se zabývala zdokonalením a implementací webové aplikace pro správu robotické soutěže Robogames, což představuje klíčový krok k zefektivnění organizace těchto soutěží. Provedené vylepšení nejen že usnadňují administrativní procesy, ale také poskytují uživatelům lepší zážitek z interakce s aplikací díky nově implementovaným funkcionalitám a uživatelskému rozhraní.

Výsledkem práce je aplikace ve frameworku React, která získává data ze serverových API a zpracovává je. Výstupem je také úprava samotné serverové části, která byla rozšířena o potřebné funkcionality pro vylepšení systému celé aplikace. Řešení navíc nabízí mnoho možností pro další rozvoj projektu. V budoucnu by bylo vhodné rozšířit aplikaci o další moduly a případně i přizpůsobení na další robotické soutěže s jinými požadavky. Děkuji svému vedoucímu práce za odborné vedení a podporu, bez které by tento projekt nebyl možný.

SEZNAM POUŽITÉ LITERATURY

[1] KRČMA, Martin. Implementace serverové části webové aplikace pro správu robotické soutěže. Bakalářská práce, vedoucí Dulík, Tomáš. Zlín: Univerzita Tomáše Bati ve Zlíně. Fakulta aplikované informatiky, Ústav informatiky a umělé inteligence, 2022. Dostupné také z: http://hdl.handle.net/10563/50522.

[2] JANOTA, Patrik. Robotické soutěže – prostředek podpory výuky obecně technického předmětu na základní škole [online]. Olomouc, 2020 [cit. 2024-02-25]. Available from: https://theses.cz/id/6tn9vs/. Master's thesis. Palacký University Olomouc, Faculty of Education. Thesis supervisor doc. PhDr. PaedDr. Jiří Dostál, Ph.D.

[3] Soutěž JedoBot. Online. Robotikahrave.cz. 2023. Dostupné z: https://robotikahrave.cz/jedobot/. [cit. 2024-02-25].

[4] Robogames. Online. Robogames.utb.cz. 2024. Dostupné z: https://robogames.utb.cz. [cit. 2024-02-25].

[5] Robotiáda. Online. Robotiada.cz. 2024. Dostupné z: https://robotiada.cz. [cit. 2024-02-25].

[6] Robotic Tournament. Online. Robotictournament.pl. 2024. Dostupné z: https://robotictournament.pl. [cit. 2024-02-25].

[7] Spring Boot. Online. Wikipedia.org. 2024. Dostupné z: https://en.wikipedia.org/wiki/Spring_Boot. [cit. 2024-03-01]

[8] Spring Boot dokumentace. Online. Spring.io. 2024. Dostupné z: https://spring.io/guides/gs/spring-boot. [cit. 2024-03-01].

[9] REST. Online. Wikipedia.org. 2024. Dostupné z: https://en.wikipedia.org/wiki/REST. [cit. 2024-03-01].

[10] Introduction to Project Lombok. Online. Www.baeldung.com. 2024. Dostupné z: https://www.baeldung.com/intro-to-project-lombok. [cit. 2024-03-01].

[11] Úvod do spring boot. Online. Itnetwork.cz. 2024. Dostupné z: https://www.itnetwork.cz/java/spring-boot/zaklady/uvod-do-spring-boot-frameworku-pro-javu. [cit. 2024-03-01].

[12] Spring Modules. Online. Www.adservio.fr. 2022. Dostupné z: https://www.adservio.fr/post/spring-modules. [cit. 2024-03-01].

[13] MariaDB Documentation. Online. Mariadb.com. 2024. Dostupné z: https://mariadb.com/kb/en/documentation/. [cit. 2024-03-01].

[14] MariaDB. Online. Wikipedia.org. 2024. Dostupné

z: https://cs.wikipedia.org/wiki/MariaDB. [cit. 2024-03-01].

[15] MariaDB a MySQL porovnání. Online. Www.websupport.cz. 2023. Dostupné z: https://www.websupport.cz/blog/2023/09/mariadb-a-mysql-porovnani-vykon-historie/. [cit. 2024-03-01].

[16] Representational State Transfer. Online. Cs.wikipedia.org. 2024. Dostupné z: https://cs.wikipedia.org/wiki/Representational_State_Transfer. [cit. 2024-03-01].

[17] RESTful API. Online. Techtarget.com. 2020. Dostupné z: https://www.techtarget.com/searchapparchitecture/definition/RESTful-API. [cit. 2024-03-01].

[18] What is REST API. Online. Codecademy.com. 2024. Dostupné z: https://www.codecademy.com/article/what-is-rest. [cit. 2024-03-01].

[19] Project Lombok. Online. Projectlombok.org. 2009. Dostupné z: https://projectlombok.org. [cit. 2024-04-09].

[20] What is Project Lombok? Online. Www.javatpoint.com. 2024. Dostupné z: https://www.javatpoint.com/lombok-java. [cit. 2024-04-09].

[21] Introduction to project Lombok in Java and how to get started. Online. Https://www.geeksforgeeks.org/. 2023. Dostupné z: https://www.geeksforgeeks.org/introduction-to-project-lombok-in-java-and-how-to-get-started/. [cit. 2024-04-09].

[22] Singleton. Online. Refactoring.guru. 2024. Dostupné z: https://refactoring.guru/design-patterns/singleton. [cit. 2024-04-09].

[23] JSON - JavaScript Object Notation. Online. Wikipedia.org. 2010, 2021. Dostupné z: https://cs.wikipedia.org/wiki/JavaScript_Object_Notation. [cit. 2024-04-09].

[24] ReactJS. Online. Legacy.reactjs.org. 20n. l. Dostupné z: https://legacy.reactjs.org. [cit. 2024-04-10].

[25] React (software). Online. Wikipedia.org. 2024. Dostupné z: https://en.wikipedia.org/wiki/React_(software). [cit. 2024-04-10].

[26] Pros and Cons of React. Online. Thecodest.co. 2022. Dostupné z: https://thecodest.co/blog/pros-and-cons-of-react/. [cit. 2024-04-10].

[27] How is React different from other JavaScript frameworks like Angular or Vue? Online. Codedamn.com. 2023. Dostupné z: https://codedamn.com/news/reactjs/how-is-react-different-from-other-javascript-frameworks-like-angular-or-vue. [cit. 2024-04-10].

[28] What is the difference between UI and UX? Online. Www.figma.com. 2024. Dostupné z: https://www.figma.com/resource-library/difference-between-ui-and-ux/. [cit. 2024-04-10].

[29] What is User Interface (UI)? (Types & Features). Online. Browserstack.com. 2023. Dostupné z: https://www.browserstack.com/guide/what-is-user-interface. [cit. 2024-04-10].

[30] Web development frameworks. Online. Browserstack.com. 2023. Dostupné z: https://www.browserstack.com/guide/web-development-frameworks. [cit. 2024-04-10].

[31] Angular (web framework). Online. Wikipedia.org. 2024. Dostupné z: https://en.wikipedia.org/wiki/Angular_(web_framework). [cit. 2024-04-10].

[32] Vue.js. Online. Wikipedia.org. 2024. Dostupné z: https://en.wikipedia.org/wiki/Vue.js. [cit. 2024-04-10].

[33] What is an API. Online. Mulesoft.com. 2024. Dostupné z: https://www.mulesoft.com/resources/api/what-is-an-api. [cit. 2024-05-06].

[34] jsPDF dokumentace. Online. Github.com/. 2020. Dostupné z: https://github.com/parallax/jsPDF. [cit. 2024-05-06].

[35] Visual Studio Code. Online. Wikipedia.org/. 2024. Dostupné z: https://en.wikipedia.org/wiki/Visual_Studio_Code. [cit. 2024-05-06].

[36] Getting started with WebStorm. Online. Jetbrains.com. 2024. Dostupné z: https://www.jetbrains.com/help/webstorm/getting-started-with-webstorm.html. [cit. 2024-05-06].

[37] ATOM Manual. Online. Flight-manual.atom-editor.cc. 2024. Dostupné z: https://flight-manual.atom-editor.cc/getting-started/sections/why-atom/. [cit. 2024-05-06].

[38] ReactJS vs NodeJS. Online. Simplilearn.com. 2023. Dostupné z: https://www.simplilearn.com/tutorials/nodejs-tutorial/reactjs-vs-nodejs. [cit. 2024-05-06].

[39] Black Dashboard React. Online. Creative-tim.com. 2023. Dostupné z: https://www.creative-tim.com/product/black-dashboard-react. [cit. 2024-05-06].

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

HTTP	Hypertext transfer protocol
JDK	Java Development Kit
MVC	Model View Controller
JDK	Java Development Kit
REST	Representational State Transfer
IDE	Integrated Development Enviroment
JSX	JavaScript XML
NPM	Node Package Module

SEZNAM OBRÁZKŮ

Příloha P I: prilohy.zip