Univerzita Hradec Králové
Fakulta informatiky a managementu
Katedra informačních technologií

Informační systém pro správu e-sportových turnajů

Návrh a implementace backendu

Bakalářská práce

Autor: Jan Najman
Studijní obor: Aplikovaná informatika

Vedoucí práce: Mgr. Hana Rohrová

Hradec Králové duben 2024

Prohlášení:

Prohlašuji, že jsem bakalářskou práci zpracoval samostatně a s použitím uvedené literatury.

V Hradci Králové dne 20. 4. 2024

Jan Najman

Poděkování:

Děkuji vedoucí bakalářské práce Mgr. Haně Rohrové za metodické vedení práce a cenné rady, které mi byly poskytnuty během celého procesu psaní.

Abstrakt

Cílem bakalářské práce je návrh a implementace backendu informačního systému pro správu e-sportových turnajů. Práce se zabývá tím, co je to e-sport, jak lze využít informační systém při správě e-sportových turnajů a jakou roli hraje backend v informačním systému. Probírána jsou i různá témata, která je nutné brát v potaz při návrhu a implementaci backendu. Mezi hlavní témata, která jsou probírána patří analýza požadavků, výběr frameworku a architektury backendu, zabezpečení a různé typy autentizace. Tato témata jsou následně využita v implementační části backendu a databáze. Práce je primárně určena pro studenty či vývojáře, kteří se zabývají softwarovým inženýrstvím nebo informačními technologiemi a mají zájem o management e-sportových turnajů.

Klíčová slova: Rust, API, REST API, Actix, Actix Web, Backend, Server, framework, esport, e-sport, turnaje

Abstract

Title: Information system for e-sports tournament management - design and implementation of backend

The goal of this bachelor thesis was to design and implement a backend of an information system for e-sports tournament management. This thesis discusses what e-sport is, how an information system can be used to manage e-sports tournaments and what role does backend play in an information system. A variety of topics are also discussed, that need to be taken into account when designing and implementing backend for an information system. The main topics that are discussed include requirements analysis, selection of backend framework and architecture, security and different types of authentication. These topics are then used during the implementation of backend and database. This thesis is primarily intended for students or developers, who are involved in software engineering or in information technologies and are interested in e-sports tournament management.

Keywords: Rust, API, REST API, Actix, Actix Web, Backend, Server, framework, esport, e-sport, tournaments

Obsah

1 Úvod

E-sport, moderní fenomén, který rozproudil vášně a soutěživost našich časů, přináší s sebou nejen radost ze hraní, ale i nespočet výzev v oblasti správy a organizace turnajů. Je nezbytné vytvářet efektivní informační systémy pro správu těchto událostí. Tato práce se zaměřuje na návrh a implementaci backendového systému pro podporu správy e-sportových turnajů, přičemž se snaží uspokojit rostoucí poptávku po profesionálních, bezproblémově organizovaných turnajích s dostatečnou technologickou infrastrukturou.

Rozvoj e-sportu přináší s sebou unikátní výzvy, od správy velkého množství údajů o hráčích a týmech po zajištění plynulého průběhu turnajů. Tato práce vychází z potřeby vytvořit komplexní informační systém, který umožní efektivní správu e-sportových turnajů. Klade si za cíl analyzovat současný stav a identifikovat klíčové oblasti, ve kterých lze pomocí informačních systémů dosáhnout zlepšení a vyšší efektivity.

Důvodem volby tohoto tématu je nejen rostoucí zájem o e-sport, ale také komplexnost a náročnost správy turnajů, které často zahrnují mnoho účastníků, různé typy her a specifické požadavky na organizaci. Zjednodušení a optimalizace těchto procesů prostřednictvím kvalitního backendového systému může mít značný vliv na kvalitu a jednoduchost pořádání e-sportových událostí.

Tato práce podrobně zkoumá problematiku e-sportových turnajů a následně navrhuje a implementuje backendový systém, který je schopen efektivně zpracovávat a spravovat data spojená s těmito událostmi. Důležitým hlediskem je nejen technologická stránka implementace, ale také zajištění bezpečnosti, kompatibility a uživatelské přátelskosti systému.

Vzhledem k rostoucímu významu e-sportu a potřebě profesionální správy turnajů se práce zaměřuje na detailní analýzu požadavků a následně na návrh a implementaci backendového systému, který odpovídá specifickým potřebám e-sportových turnajů. Tímto způsobem tato bakalářská práce přispěje k rozvoji informačních systémů v oblasti e-sportu a poskytne konkrétní řešení pro efektivní správu a organizaci e-sportových událostí.

2 Teoretická část

Text v této kapitole se bude zabývat tím, co to jsou e-sportové turnaje a rolí informačního systému v jejich správě.

2.1 E-sportové turnaje a jejich správa

V posledních letech zaznamenává e-sport velký růst a stává se fenoménem s obrovským ekonomickým obratem v řádech miliard dolarů. S přibývajícím počtem diváků, přesáhla popularita e-sportů očekávání, a podle odhadů by v roce 2023 mohlo být přibližně 650 milionů diváků. Tento vzrůstající zájem o e-sport ukazuje, že sledování těchto turnajů se stává běžným jevem, přičemž více než polovina dospělých ve Spojených státech někdy nějaký e-sport turnaj viděla. [1]

Vzhledem k rostoucímu počtu hráčů, týmů a fanoušků, kteří se zapojují do této formy soutěže, se stala správa e-sportových turnajů náročným úkolem. Organizátoři turnajů se musí vypořádat s rozmanitými výzvami, včetně logistiky, dodržování pravidel a standardů, zajištění bezpečnosti a zajímavého a příjemného zážitku pro účastníky i diváky.

S rostoucí složitostí a rozsahem e-sportových turnajů se stává klíčovým faktorem úspěchu využití moderních informačních systémů. Tyto systémy umožňují organizátorům účinně spravovat a monitorovat všechny aspekty turnaje, od registrace týmů a hráčů po sledování průběhu zápasů a poskytování živých informací divákům.

2.2 Informační systémy a jejich role ve správě turnajů

Informační systémy jsou softwarové a hardwarové systémy, které podporují datově náročné aplikace. Obsahují komplexní a integrované sady prvků, procesů a technologií, které slouží ke sběru, ukládání, zpracování, analýze a distribuci informací v organizaci nebo jiném kontextu. [2, 3, 4]

V éře informačního věku se neustále objevují různé informační systémy. Informace v reálném světě se prostřednictvím digitalizace mění v informace obsažené v informačních systémech, které následně skutečný svět zachycují. [5]

Základním cílem správy e-sportových turnajů je nejen usnadnit průběh událostí, ale také vytvořit atraktivní prostředí pro všechny zúčastněné strany. Zde hraje klíčovou roli efektivní a inovativní využití informačních systémů, které umožňují automatizaci procesů, centralizovanou správu dat a vytváření nezapomenutelného zážitků pro hráče a diváky.

2.3 Backend jako klíčová součást informačního systému

V kontextu informačního systému pro správu e-sportových turnajů hraje backend rozhodující roli, poskytuje základní infrastrukturu a funkcionalitu pro efektivní řízení a koordinaci celého systému. Jeho úloha v procesu správy turnajů spočívá v zajištění plynulé a bezproblémové komunikace mezi frontendem a databází.

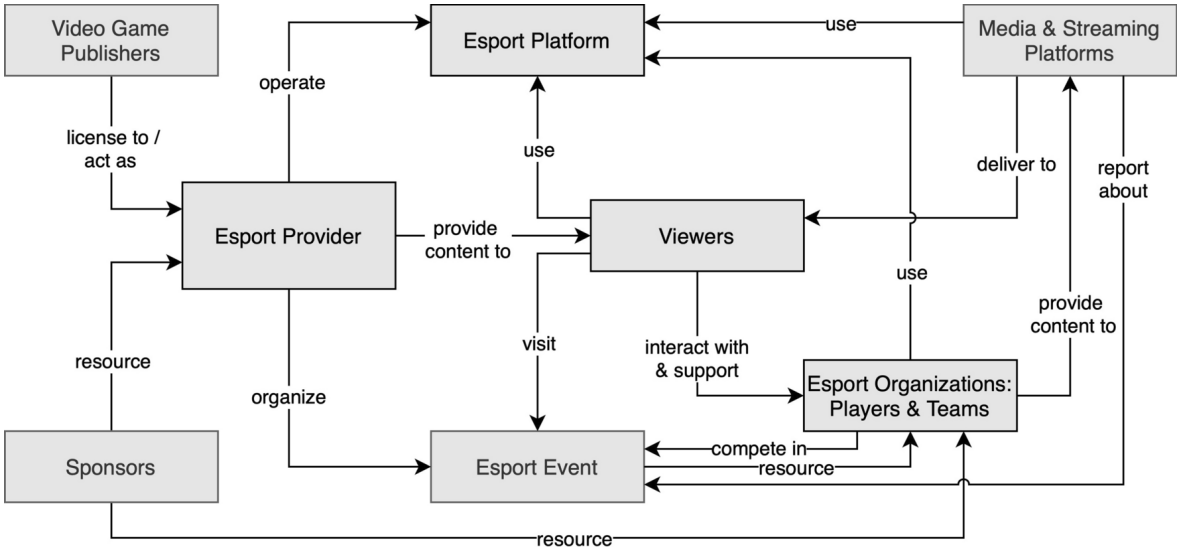
Funguje tedy jako klíčový prvek, který zpracovává a uchovává data o turnajích, týmech a hráčích. Zajišťuje manipulaci s daty, kontrolu přístupových práv a zabezpečení informačního systému proti neoprávněnému přístupu.

3 Analýza požadavků

Tato kapitola se zabývá identifikací potřeb uživatelů a specifikací požadavků na implementaci backendu.

3.1 Identifikace potřeb uživatelů

Současný výzkum e-sportu je velice rozmanitý a týká se celého ekosystému e-sportu, obrázek 1 shrnuje ekosystém. Následující text se bude zabývat potřebami jednotlivých aktéru v tomto ekosystému, bude zaměřen na tři hlavní subjekty, které provozují a využívají platformu e-sportových turnajů: organizátory turnajů, diváky a profesionální týmy a hráče. [6]



Obrázek 1: Přehled ekosystému e-sportu a zúčastněných stran [6]

3.1.1 Hráči

Potřeby hráčů spočívají v přístupu k přehledné a intuitivní platformě pro snadnou registraci na turnaje. Dále vyžadují snadný přístup k informacím o pravidlech turnajů, herních formátech a možnost sledování svého postupu v turnaji spolu s přístupem k vlastním statistikám.

3.1.2 Organizátoři turnajů

Pro organizátory turnajů je klíčové poskytnout efektivní systém pro správu a registraci týmů a hráčů. Důležitá je také možnost vytvářet a publikovat pravidla turnajů a samozřejmě sledovat průběh turnaje v reálném čase s možností okamžitého řešení případných problémů.

3.1.3 Diváci

Diváci vyžadují přístup k živým vysíláním a komentářům během turnajů. Dále chtějí mít možnost interakce s ostatními diváky a vyjádření svých názorů na průběh turnaje. Rovněž je důležitý snadný přístup ke statistikám a informacím o hráčích a týmech.

3.1.4 Ostatní aktéři

Pro ostatní aktéry, jako jsou techničtí operátoři, administrátoři a jiní členové administrátorského týmu, je nezbytné zajistit bezpečné a spolehlivé prostředí pro všechny účastníky. Implementace systému zabezpečení a autentizace je nezbytná pro ochranu citlivých informací.

3.2 Specifikace funkčních požadavků

V následující kapitole se budou zaměřovat na funkcionalitu a operace, které uživatelé mohou očekávat od systému.

3.2.1 Registrace turnaje

Uživatelé by měli mít snadný přístup k registraci do turnajů. [7, 8, 9] Proces registrace by měl obsahovat relevantní informace, jako jsou jména týmů, údaje o hráčích a kontaktní informace. [7, 8] Zároveň by měla být k dispozici intuitivní navigace a uživatelsky přívětivé rozhraní, aby se minimalizovala obtížnost při přihlašování.

3.2.2 Správa týmů

Pro kapitány týmů je klíčové poskytnout nástroje, které usnadní správu jejich týmu. [7, 8] To zahrnuje přidávání a odebírání hráčů, aktualizaci profilů hráčů a sledování statistik týmu. [7, 8] Rozhraní pro správu týmů by mělo být intuitivní, aby vyhovovalo potřebám každého týmu.

3.2.3 Rozpis zápasů

Uživatelé by měli mít k dispozici přehledný rozpis zápasů, který zahrnuje data, časy a informace o soupeřích. [7, 8, 9] Strukturované a snadno čitelné informace o plánovaných zápasech jsou klíčové pro to, aby si uživatelé mohli jednoduše naplánovat sledování turnajů a aktivně se zapojit.

3.2.4 Přímé přenosy a VOD

Měly by být zajištěny možnosti živých přenosů, umožňující divákům sledovat dění v reálném čase. [7, 8] Diváci, kteří nestihli živé přenosy, by jistě ocenili funkci pro přístup k nahraným videím/turnajům (Video on Demand). [7, 8] Tímto se zajistí maximální flexibilita pro fanoušky a účastníky turnaje.

3.2.5 Hráčské profily

Vytvoření individuálních hráčských profilů, prezentující herní historii, úspěchy a statistiky, je klíčové pro poskytnutí informací o hráčích. [7, 8, 9] Hráčské profily umožní lepší sledování oblíbených hráčů.

3.2.6 Závěrečné tabulky a pořadí

Pro uživatele je důležité mít přístup k aktuálním turnajovým žebříčkům, které ukazují postup týmů skrze různé fáze soutěže. [7, 8, 9] Pravidelně aktualizované tabulky a pořadí poskytují jasný přehled o výkonech týmů a vytvářejí napětí a rivalitu v průběhu turnaje.

3.3 Specifikace nefunkčních požadavků

Tato kapitola se zaměřuje na popis vlastností a charakteristiky systému, které nejsou přímo spojeny s funkcionalitou, ale ovlivňují jeho celkový výkon, spolehlivost a bezpečnost.

3.3.1 Dostupnost a spolehlivost

Aplikace by měla být dostupná, když ji uživatelé potřebují. Udává očekávanou doba provozuschopnosti v procentech (např. „99,9% dostupnost“ znamená, že systém může mít až 8,76 hodin výpadku za rok). [10]

Systém by měl fungovat za stanovených podmínek po určitou dobu. [10] Za těchto podmínek by systém měl běžet bez jakéhokoli problému teoreticky do nekonečna.

3.3.2 Výkon a škálovatelnost

Škálovatelnost je schopnost systému zvládnout zvýšenou zátěž, aniž by to mělo vliv na uživatelský komfort. [10] Systém by tedy měl efektivně reagovat na náhlou zátěž a nezkolabovat během průběhu turnajů. Zajištěna by také měla být stabilita a rychlost komunikace mezi frontendem a backendem pro plynulý průběh turnajů.

Backendová architektura musí být navržena s ohledem na škálovatelnost, aby bylo možné bezproblémově přizpůsobit systém růstu uživatelské základny.

3.3.3 Zabezpečení a ochrana osobních údajů

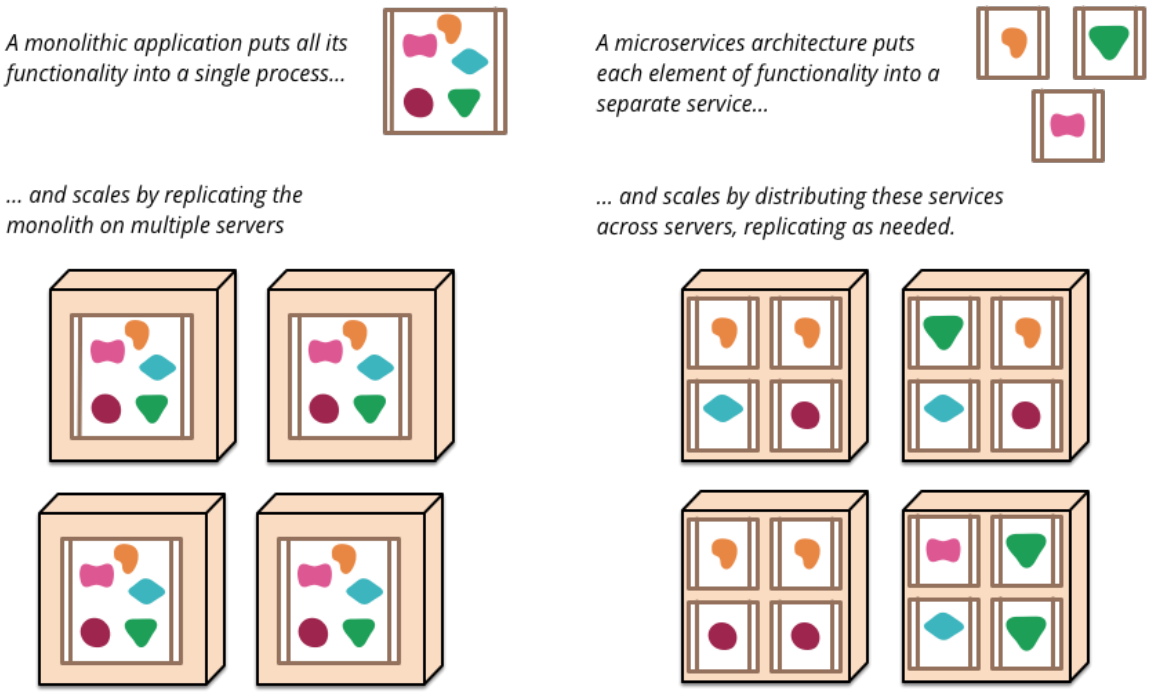
Bezpečné přihlašovací systémy, šifrování dat a opatření na ochranu soukromí jsou nezbytné, aby byla zajištěna bezpečnost osobních údajů uživatelů. [7, 8] Důkladné zabezpečení informací včetně osobních údajů hráčů a týmů je prioritou pro vytvoření důvěryhodné a bezpečné platformy pro správu e-sportových turnajů.

4 Návrh backendu

V této kapitole se budou probírat různá témata patřící do návrhu backendu.

4.1 Architektura backendu

Následující text se bude zabývat jednotlivými architekturami, které se dají použít při vytváření backendu. Hlavně se zaměří na dvě architektury, a to na monolit a mikroslužby. Obrázek 2 znázorňuje jejich škálování a strukturu.



Obrázek 2: Monolity a Mikroslužby [11]

4.1.1 Mikroslužby

Softwarová architektura založená na mikroslužbách (MSSA), je preferovaným návrhovým modelem pro rostoucí počet společností v softwarovém odvětví. Tento návrhový model byl poprvé představen v roce 2011 jako výsledek neustálých změn, které mají vyhovět současným požadavkům na vývoj softwaru. Popularita MSSA rychle roste. Dnešní poskytovatelé služeb, jako jsou Amazon, LinkedIn, Netflix, SoundCloud, Uber a Verizon, si již osvojili tuto architekturu. [12]

Mikroslužby jsou distribuované aplikace s volnými vazbami, které pracují v jednotě. Mohou být vyvíjeny, zaváděny, testovány a škálovány nezávisle, komunikují pomocí zpráv přes nenáročné komunikační mechanismy. [12]

Mikroslužby využívají distribuovaný systém ke zlepšení modularity. Distribuovaný software má však jednu zásadní nevýhodu, a to, že je distribuovaný. Jakmile začnete používat distribuci, vzniká celá řada problémů. [11]

Jedním z nich je i výkonnost. Pokud služba volá několik dalších vzdálených služeb, z nichž každá volá dalších vzdálené služby, tak délky těchto volání se sčítají a narostou do děsivých latenčních charakteristik. Tento problém lze ale jednoduše vyřešit použitím asynchronního režimu. Pokud služba provede několik asynchronních volání paralelně, místo součtu jejich latencí bude nyní pomalá jen tak, jak je pomalé její nejpomalejší volání. Toto řešení může značně zvýšit výkon, ale přináší sebou další problémy. Jeden z těchto problémů je, že asynchronní programování je obtížné, je těžké ho správně naprogramovat a ještě těžší ho ladit. Avšak většina společností, které používají mikroslužby, používá asynchronní programování, aby dosáhly přijatelného výkonu. [13]

Dalším z problémů je spolehlivost. Očekáváte, že volání služeb bude fungovat, ale vzdálené volání může kdykoli selhat. Při velkém množství mikroslužeb existuje čím dál více potenciálních míst, kde služba může selhat. [13]

Mikroslužby jsou samostatně nasaditelnou částí, takže pro každou z nich je velká volnost při výběru technologie pro její vývoj. Mikroslužby mohou být napsány v různých programovacích jazycích, používat různé knihovny a různá datová úložiště. Díky tomuto principu si mohou týmy vybrat vhodný nástroj pro danou činnost, některé jazyky a knihovny se lépe hodí pro určité typy problémů. Často se diskutuje o nejlepším nástroji pro danou činnost, ale mnohdy největší přínos mikroslužeb spočívá v jednodušší správě verzí při vývoji. [13]

4.1.2 Monolit

Monolitická architektura je aplikace s jedinou kódovou základnou, která zahrnuje více služeb. Tyto služby komunikují s externími systémy nebo uživateli prostřednictvím různých rozhraní, jako jsou stránky HTML nebo rozhraní REST API. Monolitická aplikace se obvykle skládá z databáze, klientského uživatelského rozhraní a serverové aplikace. [14, 15]

Největší výhodou monolitické architektury je její jednoduchost oproti distribuovaným aplikacím. [16] Monolitické aplikace se mnohem snadněji testují, nasazují, ladí a monitorují. [15, 16] Všechna data jsou uchovávána v jedné databázi bez nutnosti jejich synchronizace a veškerá interní komunikace probíhá prostřednictvím vnitroprocesových mechanismů. [16] Díky tomu je tento proces rychlý a nedochází u něj k problémům typickým pro komunikaci mezi procesy. [16] Monolitický přístup pro vytváření aplikací je přirozeným a primárním přístupem k vytváření aplikací. [16] Díky tomu, že veškerá logika pro zpracování požadavků je obsažena a běží v jediném procesu, je práce s touto architekturou jednodušší.

Monolitická architektura je vhodná pro malé týmy, protože usnadňuje vývoj. Každý vývojář bude moci provést změny v aplikaci nebo vytvořit něco nového, protože všechna potřebná data a prvky jsou koncentrovány v jednom pracovním prostoru. Komponenty monolitického softwaru jsou vzájemně propojené a závislé, což umožňuje, aby byl software samostatný. [15]

Většina aplikací se spoléhá na mnoho vzájemně provázaných funkcí, jako jsou kontrolní záznamy, protokolování, omezování rychlosti atd. U monolitických aplikací je řešení těchto problémů mnohem snazší, protože mají společnou kódovou základnu. Je snazší propojit komponenty s těmito úlohami, když je vše funkční v jedné aplikaci. Propojení komponent s těmito úlohami je snazší, protože vše je obsaženo v rámci jedné aplikace. [15]

Malé monolitické aplikace mají tendenci dosahovat lepších výsledků než aplikace založené na mikroslužbách. To lze vysvětlit tím, že monolitické aplikace mají společný kód a využívanou společnou paměť. [15]

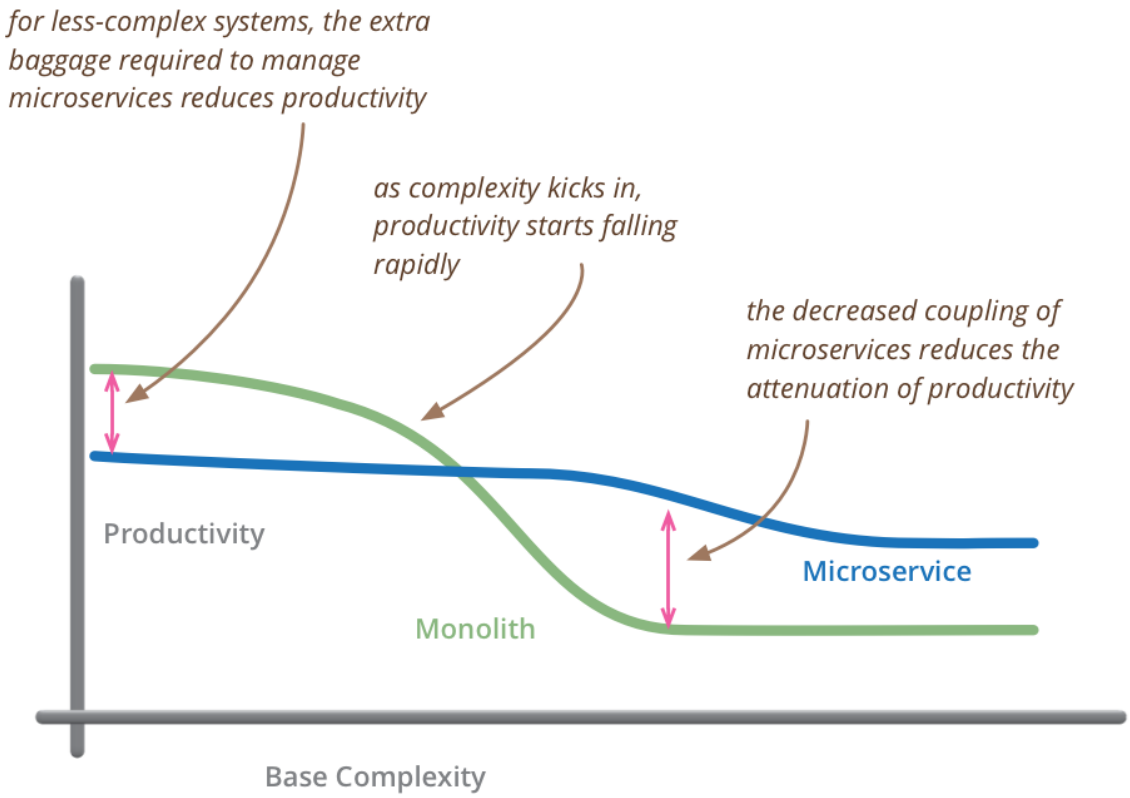
4.1.3 Monolit vs Mikroslužby

Použití mikroslužeb vyžaduje automatické nasazení, monitorování, vypořádání se s konzistencí a komplexnosti systému. Existují známá řešení jak se s těmito problémy vypořádat, ale tato řešení zabírají velké množství drahocenného času. [17]

Monolit je jednodušší programovat a nezabírá tolik času na správu. Toto však přestává platit, když se komplexita systémů zvýší na úroveň, kde správa a přidávání nových funkcí se stává obtížné. Nevýhodou však je, že orientace v něm bývá těžší, protože jsou všechny služby obsaženy v jednom celku.

> *„O mikroslužbách vůbec neuvažujte, pokud nemáte systém, který je příliš složitý na to, abyste jej spravovali jako monolit. Většina softwarových systémů by měla být vytvořena jako jediná monolitická aplikace. V rámci tohoto monolitu dbejte na vhodnou modularitu, ale nesnažte se jej rozdělit na samostatné služby.“* - Martin Fowler [17]

Tento úryvek je nádherně znázorněn v obrázku 3 níže.



Obrázek 3: Komplexnost vs Produktivita u monolitu a mikroslužeb [17]

4.2 Porovnání frameworků pro vývoj backendu

Hlavní porovnávané frameworky jsou express, axum, actix, rocket, fastapi, django, flask a php, které bylo přidáno jen kvůli porovnání, ale nebylo nikdy uvažováno nad jejím použitím. Hlavním důvodem je, že je to zastaralá technologie, která už dávno neměla existovat, ale i přesto je dále vyučována a používána.

Bylo provedeno porovnání funkcí těchto frameworků (tabulka 1). Porovnány byly také podle toho, kolik dotazů za sekundu zvládnou (tabulka 2, obrázek 38). A nakonec byly porovnány podle latencí. Průměrné latence frameworků jsou uvedeny v tabulce 3 a znázorněny grafem v obrázku 39. Latence P99 jsou uvedeny v tabulce 4 a znázorněny grafem v obrázku 40. Stejně tak jsou latence P90 uvedeny v tabulce 5 a znázorněny grafem v obrázku 41 a jako poslední byly porovnány latence P75, které jsou uvedeny v tabulce 6 a znázorněny grafem v obrázku 42.

Tyto tabulky a grafy byly vytvořeny pomocí autorem napsaných skriptů, které lze najít na GitHubu.[1] Tyto skripty je poté nutné vložit do konzole prohlížeče na stránce Web Frameworks Benchmark.[2] Pro aplikaci stejných filtrů je nutno použít stejný odkaz jako je uveden ve zdroji 18.

4.3 Návrh API pro komunikaci s frontendem

Úspěšný vývoj webových aplikací vyžaduje poskytnutí kvalitní komunikace mezi backendovými servery a frontendovými aplikacemi. Koncový uživatel rozhraní API musí snadno pochopit, jak použít toto API k vývoji funkcí a vylepšení aplikace. [19]

[1] https://github.com/HANDZCZ/bc/tree/main/thesis/utils
[2] https://web-frameworks-benchmark.netlify.app/result

4.3.1 REST

Nejvíce používané komunikační schéma, které slouží k popisu komunikace mezi frontendovým a backendovým serverem. Poskytuje backendu jednoduché rozhraní pro přístup k datům a provádění operací na straně serveru. Toto rozhraní poskytuje společně dohodnutou skupinu metod, které jsou použity při vytváření požadavků HTTP, jež poté může použít libovolná webová stránka. Jedná se o existující standard, který se běžně používá napříč aplikacemi vyvinutými v posledních letech. [19]

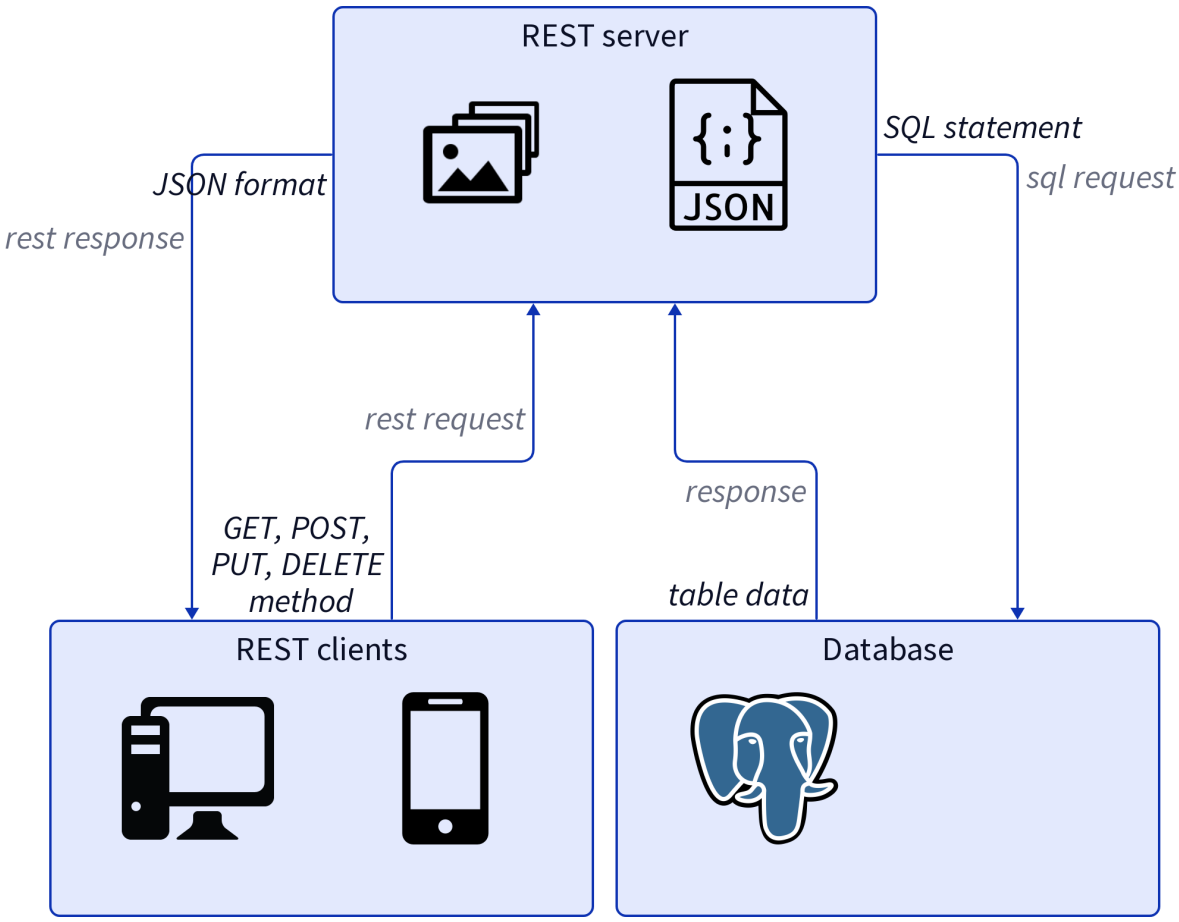
Při použití schématu REST lze implementovat klienta a server nezávisle. Což znamená, že kód na straně klienta může být kdykoli změněn, aniž by byl ovlivněn provoz serveru, a kód na straně serveru může být změněn, aniž by byl ovlivněn provoz klienta. [20]

Pokud každá strana zná formát zpráv, které má posílat té druhé, mohou být tyto strany modulární a oddělené. Když se oddělí problematika uživatelského rozhraní od problematiky ukládání dat, zlepší se flexibilita rozhraní napříč platformami a zjednoduší se škálovatelnost serverových komponent. Díky tomuto oddělení je navíc možné, aby se každá komponenta vyvíjela nezávisle. [20]

Použití tohoto typu schématu také zajistí, že různí klienti navštěvující stejné koncové body dostávají zcela stejné odpovědi.

Systémy, založené na schématu REST, jsou bezstavové, což znamená, že server nepotřebuje vědět nic o tom, v jakém stavu se nachází klient, a naopak. [20] Tímto způsobem může server i klient snadno porozumět jakékoli přijaté zprávě, bez toho aniž by viděl předchozí zprávy. [20] Díky tomuto přístupu lze výrazně zjednodušit vývoj jak backendu, tak frontendu.

Aplikace založené na schématu REST dosahují vysoké spolehlivosti, rychlého výkonu a škálovatelnosti, protože komponenty, které lze spravovat, aktualizovat a znovu používat, neovlivňují systém jako celek. [20]



Obrázek 4: REST komunikační diagram [Vlastní tvorba]

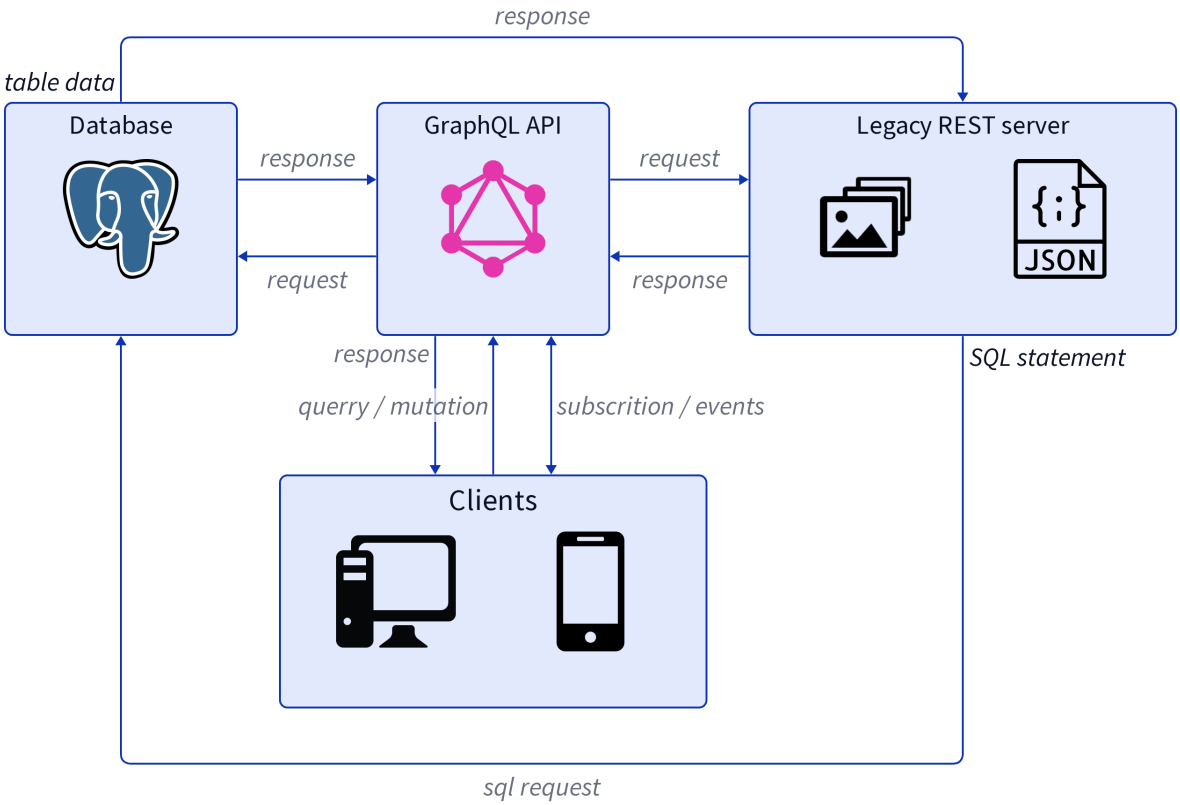
4.3.2 GraphQL

GraphQL je open-source dotazovací a manipulační jazyk pro API a runtime pro realizaci dotazů s existujícími daty. Jazyk GraphQL byl vyvinut interně společností Facebook v roce 2012 a v roce 2015 byl zveřejněn. [21]

Původ jazyka GraphQL pramení ze snahy společnosti Facebook škálovat svou mobilní aplikaci. V té době byla jejich aplikace adaptací jejich webových stránek, kdy jejich strategie pro mobilní zařízení spočívala v jednoduché „adopci“ HTML5 na mobilní zařízení. Kvůli problémům spojeným s velkým vytížením sítě a neideálním UX se však tým rozhodl vytvořit aplikaci pro iOS od základu pomocí nativních technologií. [21]

„Hlavní problém implementace kanálu novinek v mobilních zařízeních byl to, že nebylo jednoduché získat zprávu, kdo ji napsal, co v ní stojí, seznam komentářů a kdo příspěvku dal lajk. Stávající rozhraní API nebyla navržena tak, aby umožnila vývojářům poskytnout přístup k informacím potřebných k vývoji kanálu novinek na mobilních zařízeních. Neměly hierarchickou strukturu, neumožňovaly vývojářům vybrat si jaká data potřebují, ani možnost zobrazit seznam různorodých příběhů v kanálu.“ - Brenda Clark [22]

GraphQL je dotazovací jazyk pro rozhraní API a runtime pro realizaci dotazů pomocí existujících dat. Největší výhoda jazyka GraphQL spočívá především v tom, že GraphQL poskytuje kompletní a srozumitelný popis dat v rozhraní API, klientům dává možnost žádat přesně to, co potřebují, a nic navíc. Při zasílání dotazů na rozhraní API vrací jazyk GraphQL zcela předvídatelné výsledky, aniž by docházelo k získávání více dat, nebo méně dat než je potřeba, což zajišťuje, že aplikace využívající jazyk GraphQL jsou rychlé, stabilní a škálovatelné. [21, 22]



Obrázek 5: GraphQL komunikační diagram [Vlastní tvorba]

4.4 Zabezpečení

Následující texty probírají nejčastější bezpečnostní problémy a jejich vliv na zabezpečení.

4.4.1 SQL injection

Velkým bezpečnostním problémem webových infrastruktur jsou útoky typu injection, zejména SQL injection. Útok SQL injection umožňuje útočníkovi zasahovat do databáze webové aplikace a krást informace nebo dokonce měnit či mazat legitimní data uložená v aplikaci. Open Web Application Security Project (OWASP) je celosvětový neziskový projekt usilující o zlepšení bezpečnosti softwaru. Tato komunita vydává „OWASP Top 10“, dokument pro zvýšení povědomí vývojářů a zabezpečení webových aplikací. Tento dokument obsahuje žebříček nejkritičtějších bezpečnostních rizik webových aplikací. OWASP Top 10 řadí injekce jako třetí nejzávažnější bezpečnostní riziko webových aplikací v roce 2021. Podobně i společnost MITRE zveřejňuje žebříček CWE Top 25 Most Dangerous Software Weaknesses. V tomto žebříčku zaujímají SQL injekce rovněž třetí místo. [23]

SQL injection umožňuje útočníkovi manipulovat s dotazy aplikace na databázi. Obvykle umožňuje útočníkovi zobrazit data, ke kterým by neměl mít přístup. Může jít například o data patřící jiným uživatelům, přihlašovací údaje, struktury tabulek nebo jakákoli jiná data, ke kterým má aplikace přístup. V mnoha případech může útočník tato data také upravovat nebo mazat, čímž způsobí trvalé změny obsahu nebo i změny v chování aplikace. V některých situacích je také možné, že útočník může eskalovat svá práva pomocí SQL injection a napadnout server, na kterém tento útok byl proveden, nebo jinou back-endovou infrastrukturu či provést útoky typu „denial of service“. [23, 24]

4.4.2 Nezabezpečená deserializace

Nezabezpečená deserializace je bezpečnostní riziko, při kterém jsou nedůvěryhodná nebo neznámá data použita ke spuštění útoku typu „denial of service,“ spuštění libovolného kódu, obejití autentizace nebo jinému zneužití logiky aplikace. [24, 25]

Serializace je proces převodu složitých datových struktur, jako jsou objekty a jejich pole, do formátu, který lze odeslat a přijmout jako sekvenční proud bajtů. Deserializace je proces obnovení tohoto proudu bajtů do plně funkční repliky původního objektu, přesně ve stavu, v jakém byl při serializaci. Logika webové stránky pak může s tímto deserializovaným objektem pracovat stejně jako s jakýmkoli jiným objektem. [24, 25, 26]

V některých případě je možné nahradit serializovaný objekt objektem zcela jiné třídy. Alarmující je, že objekty libovolné třídy, které jsou na webové stránce k dispozici, budou deserializovány a instancovány bez ohledu na to, která třída byla očekávána. Z tohoto důvodu se nezabezpečená deserializace někdy označuje jako „object injection.“ [26]

Při přijetí objektu neočekávané třídy může dojít k výjimce, ale v této době však již mohlo dojít ke škodám. Mnoho útoků založených na deserializaci je dokonáno ještě před dokončením deserializace. To znamená, že samotný proces deserializace může zahájit útok, i když funkce webové stránky přímo nepracují se škodlivým objektem. Z tohoto důvodu mohou být vůči těmto technikám zranitelné i webové stránky, jejichž logika je založena na silně typovaných jazycích. [26]

Nezabezpečená deserializace obvykle vzniká z důvodu obecného nepochopení toho, jak nebezpečná může být deserializace dat ovládaných uživatelem. V ideálním případě by uživatelský vstup neměl být deserializován vůbec. [26]

Dokonce ani v případě, že je na deserializovaná data implementována nějaká forma dodatečné kontroly. Tento přístup je často neúčinný, protože je prakticky nemožné implementovat validaci nebo sanitizaci, která by zohlednila všechny eventuality. Uvedené kontroly jsou také zásadně chybné, protože se spoléhají na kontrolu dat po jejich deserializaci, což v mnoha případech bude příliš pozdě na to, aby se zabránilo útoku. [26]

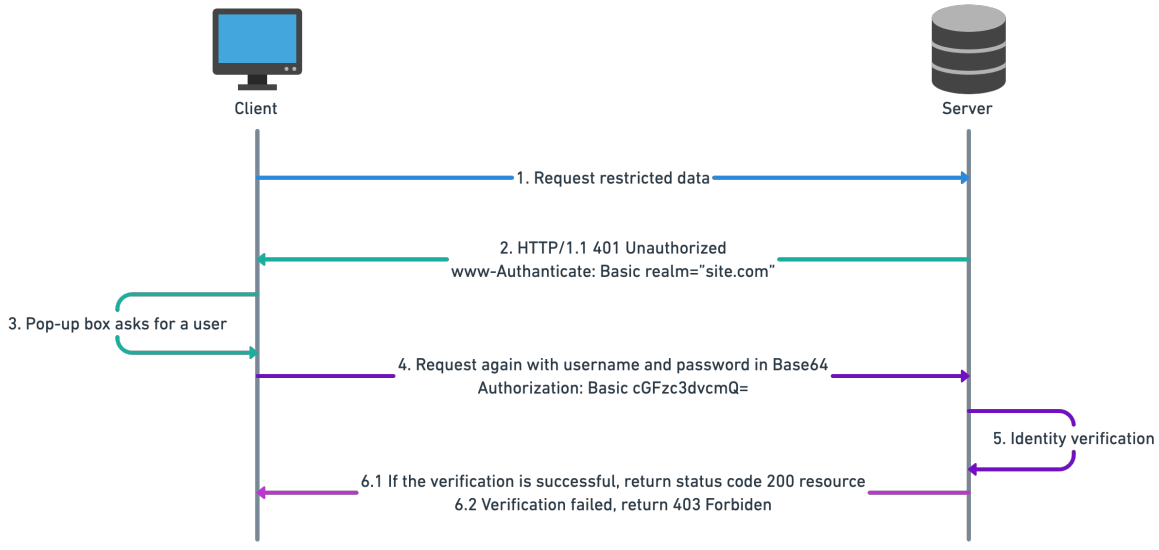
Další z důvodů proč tento útok může být úspěšný je, že deserializované objekty jsou často považovány za důvěryhodné. Zejména při použití jazyků s binárním serializačním formátem se vývojáři mohou domnívat, že uživatelé nemohou data efektivně číst nebo s nimi manipulovat. Nicméně, i přestože tento serializační formát může vyžadovat více úsilí, útočník je schopen zneužít binární serializované objekty stejně tak jako řetězcové formáty (JSON, XML, atd.). [26]

4.5 Autentizace

Tato kapitola se zabývá různými typy autentizace a jejich principy, výhodami či nevýhodami.

4.5.1 Základní HTTP autentizace

Jedná se o nejjednodušší metodu při, které umístí odesílatel do hlavičky požadavku uživatelské jméno a heslo. Uživatelské jméno a heslo jsou zakódovány pomocí Base64, toto kódování převádí uživatelské jméno a heslo na sadu 64 znaků, aby byl zajištěn bezpečný přenos. Tato metoda nevyžaduje soubory cookies, identifikaci relace, přihlašovací stránky a další podobné možnosti identifikace uživatele, protože využívá samotnou hlavičku HTTP. [27, 28, 29, 30]



Obrázek 6: HTTP autentizace [27]

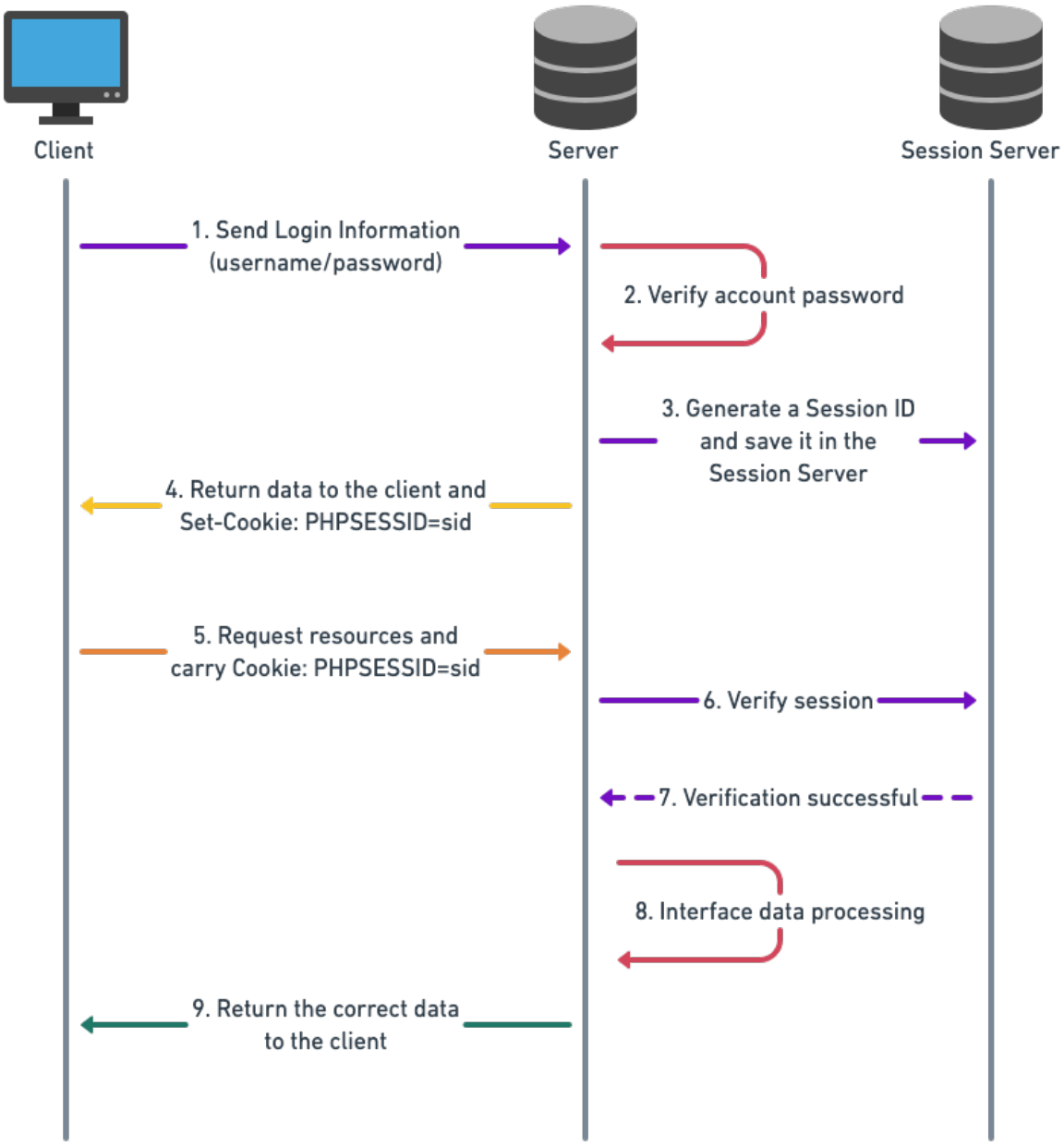
Tato metoda autentizace se doporučuje jen zřídka kvůli tomu, že se dá snadno napadnout. I přesto, že se používá kódování Base64, lze toto kódování snadno dekódovat. Dokonce i když obsah autentizace nelze dekódovat do původního uživatelského jména a hesla, je tento typ autentizace stále nedostatečně zabezpečený. Útočníci mohou získat autentizační obsah a opakovaně odesílat požadavky na server. Tento typ útoku se označuje jako replay útok. [27, 28, 29, 30]

4.5.2 Autentizace pomocí souborů cookies

Autentizace pomocí souborů cookies je model ověřování komunikace mezi relací na straně serveru a souborem cookies prohlížeče (na straně klienta). [27]

Webové stránky a webové aplikace používající soubory cookies k ověřování uživatelů, nejprve požádají uživatele, aby se přihlásil na webové stránky. Po přihlášení je vytvořen unikátní malý textový soubor. Tento soubor je zvláštním identifikátorem spojeným s účtem uživatele. Zařízení uživatele pak tento soubor cookies obdrží a uloží do svého prohlížeče. [31]

Webové stránky mohou uživatele ověřit, aniž by se musel znovu přihlašovat, díky tomu, že při dalších návštěvách odešle tento soubor cookies. [31]



Obrázek 7: Autentizace pomocí souborů cookies [27]

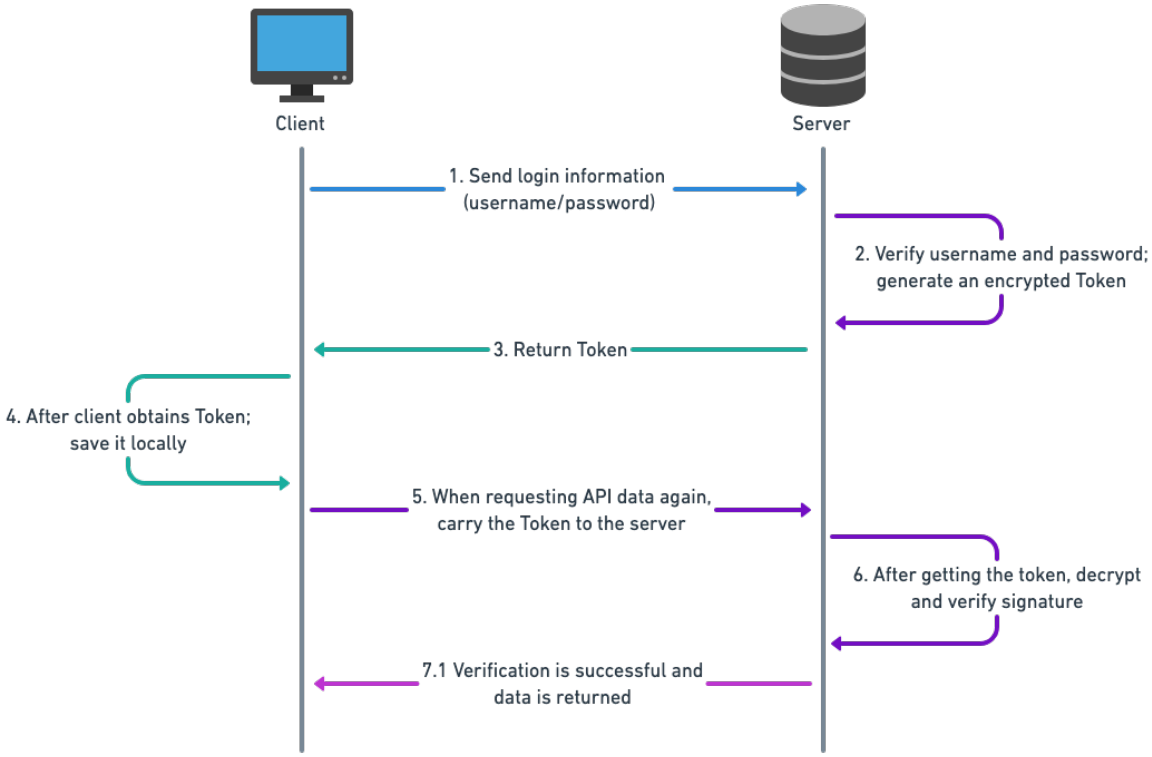
Hlavní výhoda této metody je, že se uživatelé nemusí opakovaně přihlašovat, aby získali přístup ke svým účtům. U této metody je, ale nezbytné zajistit, že soubory cookies používané k ověřování identity uživatele byly zabezpečené a těžko manipulovatelné, aby nedošlo k ohrožení bezpečnosti uživatelského účtu. [31]

4.5.3 Ověření pomocí tokenu

Ověření pomocí souborů cookies má několik nevýhod, včetně obtížné údržby na straně serveru, zejména v distribuovaných systémech. To vedlo k hledání efektivnější alternativy, která by tyto problémy vyřešila. [27, 28, 29, 30]

Jako odpověď na tento problém se našla autentizace pomocí tokenu. [27, 28, 29, 30]

Ověření pomocí tokenu je typ ověřování, který k ověření identity uživatele používá tokeny. Tokeny jsou malé části dat, které generuje server a posílá je klientovi. Klient pak token uloží a použije jej k ověření na serveru při vytváření dotazů. [27, 28, 29, 30]

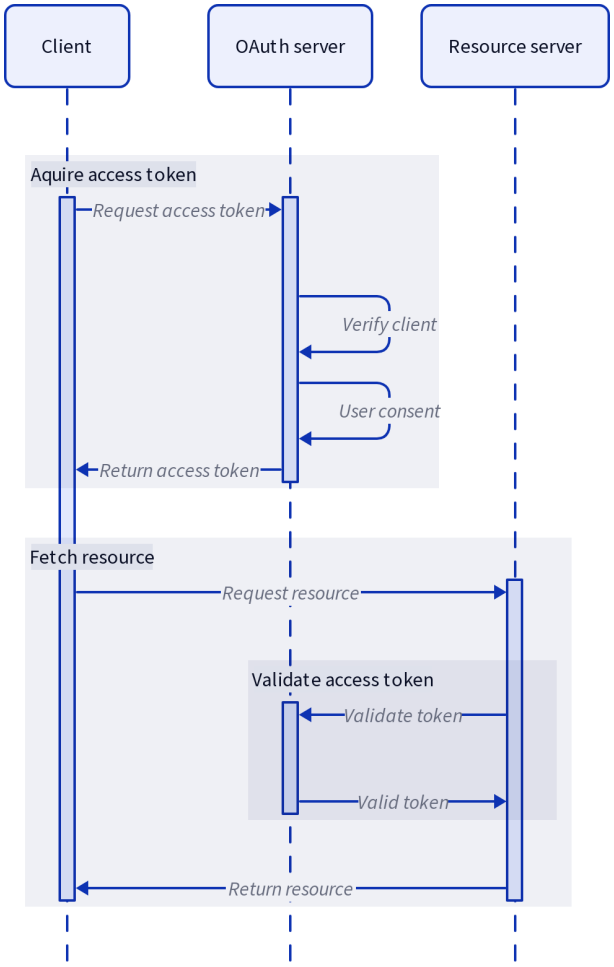


Obrázek 8: Ověření pomocí tokenu [27]

4.5.4 OAuth 2.0

OAuth je otevřený standard, který umožňuje uživatelům udělovat ostatním stránkám oprávnění pro přístup k jejich uloženým informacím, uložených u jiných poskytovatelů různých služeb. Uživatelé tak nemusí poskytovat své uživatelské jméno a heslo přímo těmto stránkám. [27, 28, 29, 30]

OAuth slouží jako autorizační mechanismus, který umožňuje vlastníkovi dat udělovat aplikacím třetích stran oprávnění k přístupu do systému a získávání dat. Systém pro tento účel vygeneruje dočasný přístupový token, který při použití v aplikacích třetích stran nahradí nutnost zadávání hesla. [27, 28, 29, 30]



Obrázek 9: Ověření pomocí OAuth 2.0 [Vlastní tvorba]

5 Implementace

Následující text se bude zabývat tím, jak byly implementovány jednotlivé části backendu.

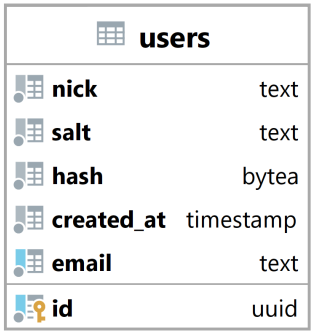
5.1 Databázová část

Tato část se bude zabývat tím, jak byla implementována databázová část. Celý diagram databázových tabulek je uveden v obrázku 37.

5.1.1 Tabulka users

Tabulka users reprezentuje uživatele v systém. Termín uživatel v tomto kontextu může označovat organizátora turnaje, manažera týmu či hráče v týmu.

Jejím hlavním úkolem je umožnit backendu identifikovat uživatele a uživateli umožnit přihlášení.



Obrázek 10: Tabulka users [Vlastní tvorba]

Nick slouží jako přezdívka pro uživatele, tento termín je hlavně používán v herní komunitě.

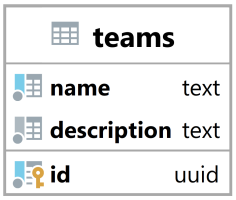
Salt a hash jsou použity k ověření uživatele při přihlášení.

Email je nejen použit při přihlašování, ale také při registraci. Slouží jako unikátní identifikátor uživatele.

Primární klíč id je primárně použit k identifikaci a vázání uživatele na ostatních tabulky, protože email je možné kdykoli změnit a také hlavně proto, že email je soukromá informace a neměla by být sdílena s ostatními tabulkami.

5.1.2 Tabulka teams

Tabulka teams slouží k reprezentaci týmu v systému. Jejím hlavním úkolem je umožnit backendu identifikovat týmy.



Obrázek 11: Tabulka teams [Vlastní tvorba]

Sloupec name slouží k identifikaci týmu z pohledu uživatele.

Description neboli popis slouží k popisu týmu, zde mohou manažeři týmu uvádět detaily o jejich týmu.

Primární klíč id je primárně použit k identifikaci a vázání týmu na ostatní tabulky.

5.1.3 Tabulka games

Tabulka games slouží k reprezentaci hry v systému. Jejím hlavním úkolem je umožnit backendu identifikovat hry a zjistit jejich vlastnosti.

Obrázek 12: Tabulka games [Vlastní tvorba]

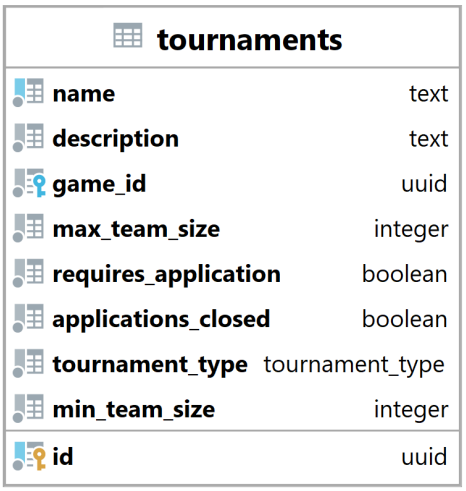
Sloupec name slouží k identifikaci hry z pohledu uživatele a vyjadřuje název hry.

Description neboli popis slouží k popisu hry, zde mohou organizátoři turnaje specifikovat detaily o dané hře.

Primární klíč id je primárně použit k identifikaci a vázání her na ostatní tabulky.

5.1.4 Tabulka tournaments

Tabulka tournaments slouží k reprezentaci turnaje v systému. Jejím hlavním úkolem je umožnit backendu identifikovat turnaje a zjistit jejich vlastnosti.



Obrázek 13: Tabulka tournaments [Vlastní tvorba]

Sloupec name slouží k identifikaci turnaje z pohledu uživatele.

Description neboli popis slouží k popisu turnaje, zde mohou organizátoři turnaje specifikovat detaily turnaje, jako jsou pravidla turnaje.

Game_id slouží k navázaní hrané hry (sekce 5.1.3) k turnaji.

Max_team_size a min_team_size slouží k specifikaci velikosti týmu.

Requires_application označuje, zda turnaj vyžaduje podání přihlášky. Pokud ano, tak se přihlášky budou nacházet v tabulce teams_to_tournaments_applications (sekce 5.1.8), a pokud ne, tak týmy, které se hlásí na tento turnaj, jsou automaticky přihlášeny a přidány do tabulky teams_to_tournaments (sekce 5.1.7). Tuto funkci obstarává procedura apply_for_tournament, více v sekci 5.1.36.

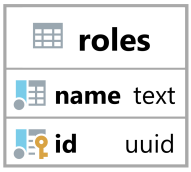
Applications_closed vyjadřuje zda jsou žádosti o přihlášení uzavřeny.

Tournament_type slouží k specifikaci typu turnaje, více v sekci 5.1.14.

Primární klíč id je primárně použit k identifikaci a vázání turnajů na ostatní tabulky.

5.1.5 Tabulka roles

Tabulka roles slouží k reprezentaci role uživatele v systému. Jejím hlavním úkolem je umožnit backendu identifikovat typ role.



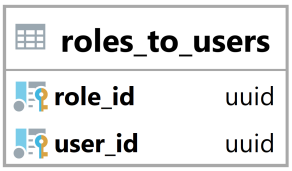
Obrázek 14: Tabulka roles [Vlastní tvorba]

Sloupec name vyjadřuje název role a je dále používán v backendu ke kontrole oprávnění.

Primární klíč id je primárně použit k identifikaci a vázání role na ostatní tabulky.

5.1.6 Tabulka roles_to_users

Tabulka roles_to_users slouží ke spojení uživatele s rolemi. Jejím hlavním úkolem je umožnit backendu zjistit jaké má uživatel role.



Obrázek 15: Tabulka roles_to_users [Vlastní tvorba]

Role_id vyjadřuje id navázané role (sekce 5.1.5).

User_id vyjadřuje id navázaného uživatele (sekce 5.1.1).

Primární klíč je složen z role_id a user_id.

5.1.7 Tabulka teams_to_tournaments

Tabulka teams_to_tournaments slouží ke spojení týmů a turnajů. Jejím hlavním úkolem je umožnit backendu zjistit, jaké týmy jsou přihlášeny na jaké turnaje. Tato tabulka neobsahuje týmy, které teprve podaly přihlášku, ale ty které už mají potvrzenou účast.

Obrázek 16: Tabulka teams_to_tournaments [Vlastní tvorba]

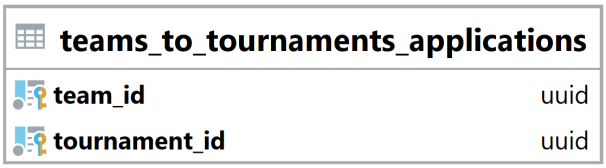
Team_id vyjadřuje id navázaného týmu (sekce 5.1.2).

Tournament_id vyjadřuje id navázaného turnaje (sekce 5.1.4).

Primární klíč je složen z team_id a tournament_id.

5.1.8 Tabulka teams_to_tournaments_applications

Tabulka teams_to_tournaments_applications slouží ke spojení týmů a turnajů pro přihlášky. Jejím hlavním úkolem je umožnit backendu zjistit, jaké týmy podaly přihlášku na jaké turnaje. Tato tabulka obsahuje týmy, které podaly přihlášku na turnaj, jež vyžaduje přihlášky, a zároveň tyto přihlášky nesmí být potvrzeny.



Obrázek 17: Tabulka teams_to_tournaments_applications [Vlastní tvorba]

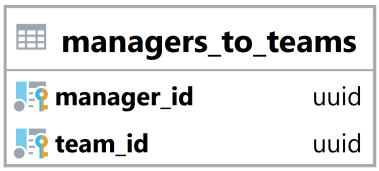
Team_id vyjadřuje id navázaného týmu (sekce 5.1.2).

Tournament_id vyjadřuje id navázaného turnaje (sekce 5.1.4).

Primární klíč je složen z team_id a tournament_id.

5.1.9 Tabulka managers_to_teams

Tabulka managers_to_teams slouží ke spojení uživatelů a týmů. Jejím hlavním úkolem je umožnit backendu zjistit, jací uživatelé jsou manažeři daného týmu. Tato tabulka obsahuje jen manažery, kteří již potvrdily pozvánku do týmu.



Obrázek 18: Tabulka managers_to_teams [Vlastní tvorba]

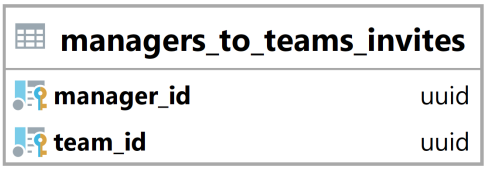
Team_id vyjadřuje id navázaného týmu (sekce 5.1.2).

Manager_id vyjadřuje id navázaného uživatele (sekce 5.1.1).

Primární klíč je složen z team_id a manager_id.

5.1.10 Tabulka managers_to_teams_invites

Tabulka managers_to_teams_invites slouží ke spojení uživatelů a týmů pro pozvánky. Jejím hlavním úkolem je umožnit backendu zjistit, jací uživatelé jsou pozvání do týmu jako manažeři. Tato tabulka obsahuje jen pozvánky, které ještě nebyly přijaty.



Obrázek 19: Tabulka managers_to_teams_invites [Vlastní tvorba]

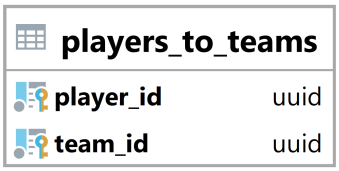
Team_id vyjadřuje id navázaného týmu (sekce 5.1.2).

Manager_id vyjadřuje id navázaného uživatele (sekce 5.1.1).

Primární klíč je složen z team_id a manager_id.

5.1.11 Tabulka players_to_teams

Tabulka players_to_teams slouží ke spojení uživatelů a týmů. Jejím hlavním úkolem je umožnit backendu zjistit, jací uživatelé jsou patří do daného týmu. Tato tabulka obsahuje jen uživatelé, kteří již potvrdily pozvánku do týmu.



Obrázek 20: Tabulka players_to_teams [Vlastní tvorba]

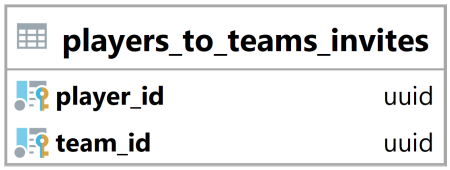
Team_id vyjadřuje id navázaného týmu (sekce 5.1.2).

Player_id vyjadřuje id navázaného uživatele (sekce 5.1.1).

Primární klíč je složen z team_id a player_id.

5.1.12 Tabulka players_to_teams_invites

Tabulka players_to_teams_invites slouží ke spojení uživatelů a týmů pro pozvánky. Jejím hlavním úkolem je umožnit backendu zjistit, jací uživatelé jsou pozvání do týmu jako hráči. Tato tabulka obsahuje jen pozvánky, které ještě nebyly přijaty.



Obrázek 21: Tabulka players_to_teams_invites [Vlastní tvorba]

Team_id vyjadřuje id navázaného týmu (sekce 5.1.2).

Player_id vyjadřuje id navázaného uživatele (sekce 5.1.1).

Primární klíč je složen z team_id a player_id.

5.1.13 Tabulka players_to_tournaments_playing

Tabulka players_to_tournaments_playing slouží ke spojení uživatelů, týmů a turnajů. Jejím hlavním úkolem je umožnit backendu zjistit, jací uživatelé hrají v jakých turnajích a za jaký tým. Díky této tabulce se také jeden tým může účastnit více turnajů současně.

Obrázek 22: Tabulka players_to_tournaments_playing [Vlastní tvorba]

Team_id vyjadřuje id navázaného týmu (sekce 5.1.2).

Player_id vyjadřuje id navázaného uživatele (sekce 5.1.1).

Tournament_id vyjadřuje id navázaného turnaje (sekce 5.1.4).

Primární klíč je složen z tournament_id a player_id a zajišťuje, že uživatel může hrát v jednom turnaji jen za jeden tým.

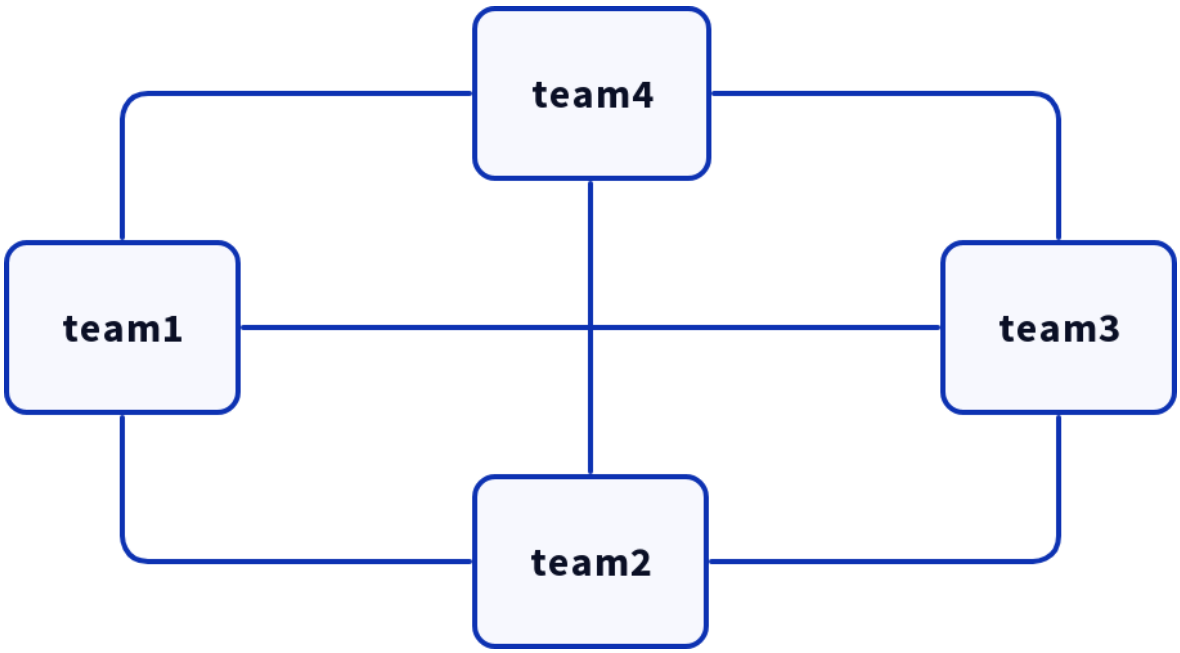
5.1.14 Typ tournament_type

Typ tournament_type slouží k určení typu turnaje. Jeho hlavním úkolem je signalizovat backendu jak má probíhat vytváření bracketů.

Kód 1: Kód na vytvoření tournament_type typu

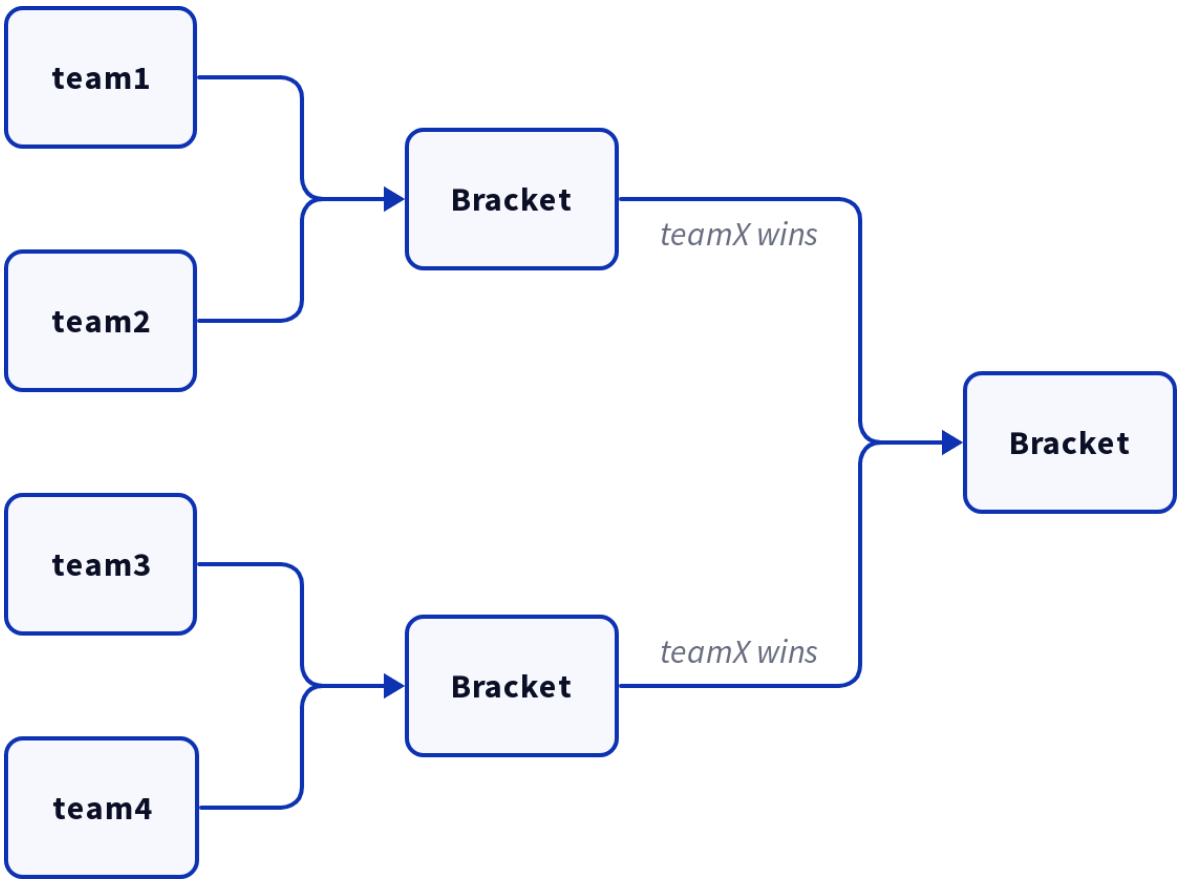
```
create type tournament_type as enum (
    'FFA',
    'OneBracketTwoFinalPositions',
    'OneBracketOneFinalPositions'
    );
```

Tento typ může nabýt 3 hodnot. V turnaji typu FFA (Free For All) bojuje každý tým s každým. Obrázek níže znázorňuje FFA turnaj pro čtyři týmy, každé spojení znázorňuje souboj.



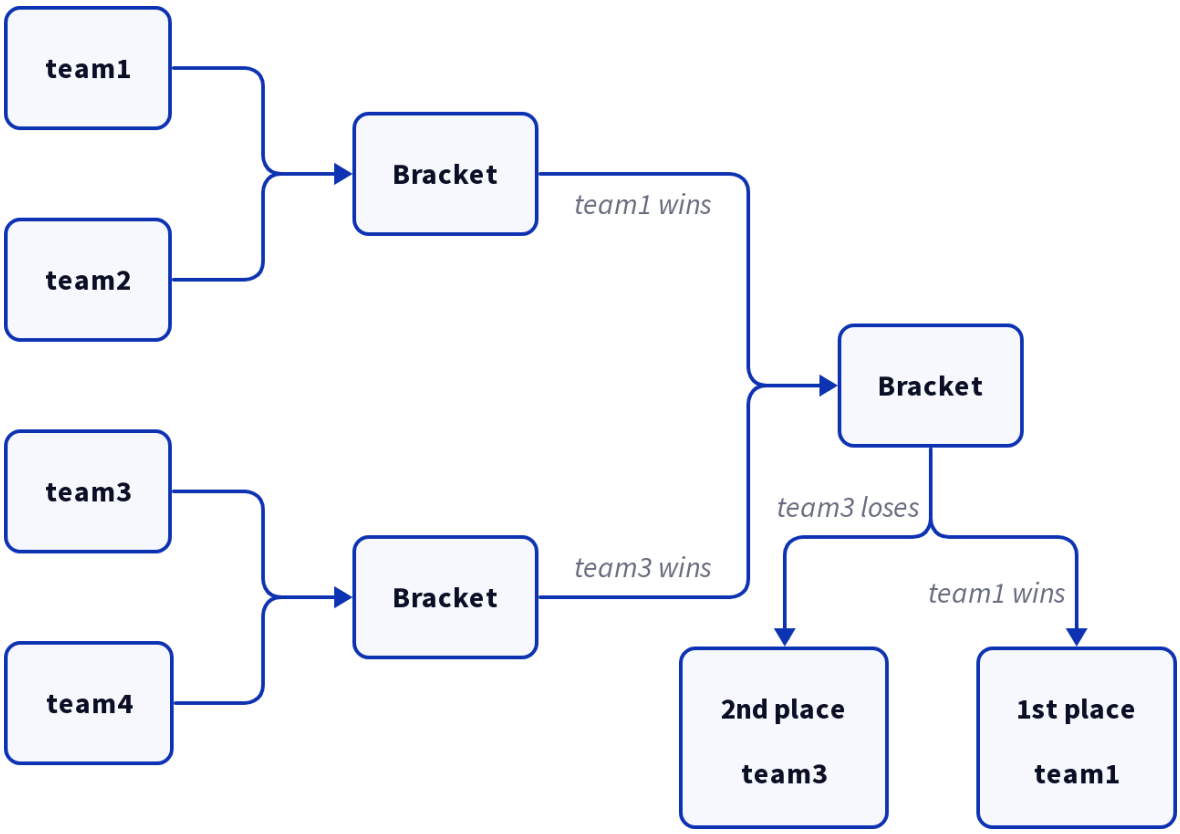
Obrázek 23: Znázornění typu turnaje FFA [Vlastní tvorba]

V turnaji typu OneBracketTwoFinalPositions a OneBracketOneFinalPositions jsou týmy náhodně rozmístěny do poslední vrstvy stromu. Podle tohoto rozmístění poté spolu bojují a postupují do vyšších vrstev. Obrázek níže znázorňuje jednoduchou stromovou strukturu pro oba typy turnaje (OneBracketTwoFinalPositions a OneBracketOneFinalPositions) se čtyřmi týmy.



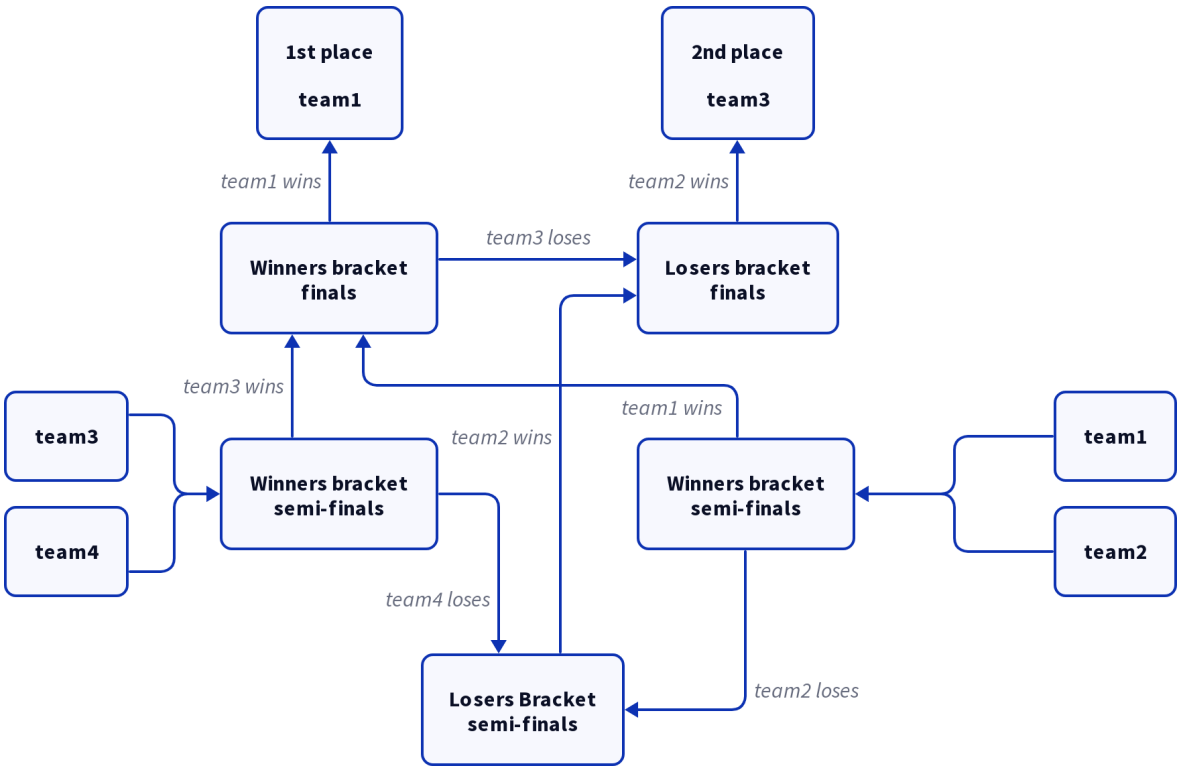
Obrázek 24: Znázornění stromové struktury [Vlastní tvorba]

V turnaji typu OneBracketTwoFinalPositions se na konci stromu vždy rozhodují dvě finální pozice. Obrázek níže znázorňuje tento typ turnaje pro 4 týmy.



Obrázek 25: Znázornění typu turnaje OneBracketTwoFinalPositions [Vlastní tvorba]

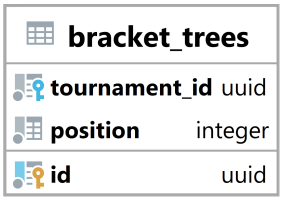
V turnaji typu OneBracketOneFinalPositions se na konci stromu vždy rozhoduje pouze jedna finální pozice. Obrázek níže znázorňuje tento typ turnaje pro 4 týmy.



Obrázek 26: Znázornění typu turnaje OneBracketOneFinalPositions [Vlastní tvorba]

5.1.15 Tabulka bracket_trees

Tabulka bracket_trees slouží k reprezentaci části stromové struktury v systému. Jejím hlavním úkolem je umožnit backendu identifikovat k jakému stromu patří brackety a o jaký typ stromu se jedná.



Obrázek 27: Tabulka bracket_trees [Vlastní tvorba]

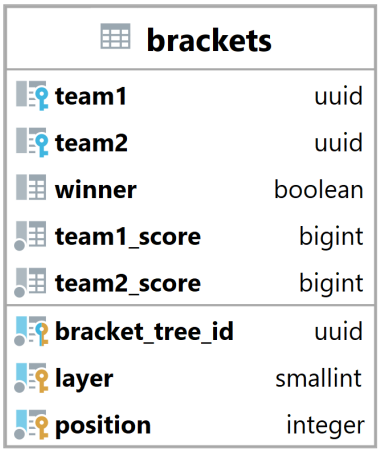
Tournament_id vyjadřuje id navázaného turnaje (sekce 5.1.4).

Position vyjadřuje pozici a vlastnosti stromu. Pozice -1 říká, že se jedná o FFA strom. Pozice 0 říká, že tento strom je na první pozici a určuje výherce turnaje. Pozice 1 říká, že se jedná o první strom, kde jsou hráči, co prohráli ve hlavním stromu na pozici 0. Pozice 2 až n říká, že se jedná o n-1 strom, kde jsou hráči, co prohráli ve stromu n-1. Pozice 0 až n mohou určovat pouze jednoho výherce tohoto stromu, nebo mohou určovat výherce a prohraného. Toto určování záleží na typu turnaje, více v sekci 5.1.14.

Primární klíč id je primárně použit k identifikaci stromů a vázání bracketů stromy.

5.1.16 Tabulka brackets

Tabulka brackets slouží k reprezentaci části stromové struktury v systému. Jejím hlavním úkolem je umožnit backendu získat data o průběhu turnaje nebo zápasu.



Obrázek 28: Tabulka brackets [Vlastní tvorba]

Team1 a team2 vyjadřují id navázaných týmu (sekce 5.1.2). Když je hodnota null, tak ještě tým nepostoupil do tohoto bracketu.

Winner říká jaký tým vyhrál. Pokud je hodnota true, tak vyhrál první tým, pokud je hodnota false, tak vyhrál druhý tým, a pokud je hodnota null, tak ještě nevyhrál žádný tým.

Team1_score a team2_score udává score kterého dosáhly týmy v zápasu.

Bracket_tree_id vyjadřuje id navázaného stromu (sekce 5.1.15).

Layer a position udávají přesnou pozici bracketu ve stromu. Layer neboli vrstva udává jak hluboko (na jaké vrstvě) se bracket nachází. Position neboli pozice udává na jaké pozici ve vrstvě se bracket nachází.

Primární klíč je složen z bracket_tree_id, layer a position a zaručuje unikátnost bracketu ve stromě.

5.1.17 Pohled teams_tournaments_playing_players

Pohled teams_tournaments_playing_players slouží ke přehlednějšímu zobrazení a jednoduššímu načtení dat. Jeho hlavním úkolem je zobrazit, jestli hráči v týmu hrají v daném turnaji nebo ne. Tento pohled umožňuje jednoduché načtení a vyhledání dat backendem.

Obrázek 29: Pohled teams_tournaments_playing_players [Vlastní tvorba]

Sloupce team_id a team_name náleží příslušným sloupcům v tabulce teams (sekce 5.1.2), bez prefixu team_.

Sloupce tournament_id a tournament_name náleží příslušným sloupcům v tabulce tournaments (sekce 5.1.4), bez prefixu tournament_.

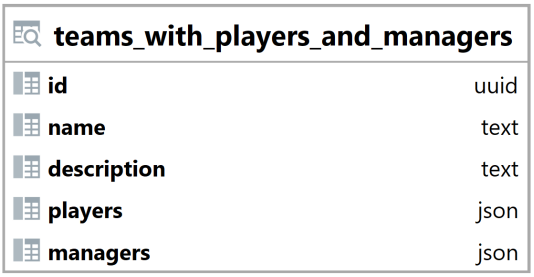
Sloupec players obsahuje pole s hráči ve formátu json. Hráči dále obsahují informace jak o sobě, tak i o tom, jestli hrají v turnaji za daný tým.

Kód 2: Pohled teams_tournaments_playing_players (sekce 5.1.17) - příklad hodnoty sloupce players

```
1 [{"player_id": "d685f026-f505-4e59-a927-e91f11f92cf0", "nick":
   ↪  "TEST :)", "playing": true},
2  {"player_id": "264fb521-ba85-4572-931b-d22157b69b2d", "nick":
    ↪  "TEST 2", "playing": false}]
```

5.1.18 Pohled teams_with_players_and_managers

Pohled teams_with_players_and_managers slouží ke přehlednějšímu zobrazení a jednoduššímu načtení dat. Jeho hlavním úkolem je zobrazit, jací hráči a manažeři patří do týmu. Tento pohled umožňuje jednoduché načtení a vyhledání dat backendem.



Obrázek 30: Pohled teams_with_players_and_managers [Vlastní tvorba]

Id vyjadřuje id týmu (sekce 5.1.2).

Name a description jsou brány z tabulky teams (sekce 5.1.2).

Sloupec players obsahuje pole s hráči ve formátu json.

Kód 3: Pohled teams_with_players_and_managers (sekce 5.1.18) - příklad hodnoty sloupce players

```
[{"id": "d685f026-f505-4e59-a927-e91f11f92cf0", "nick": "TEST :)"},
 {"id": "264fb521-ba85-4572-931b-d22157b69b2d", "nick": "TEST 2"}]
```

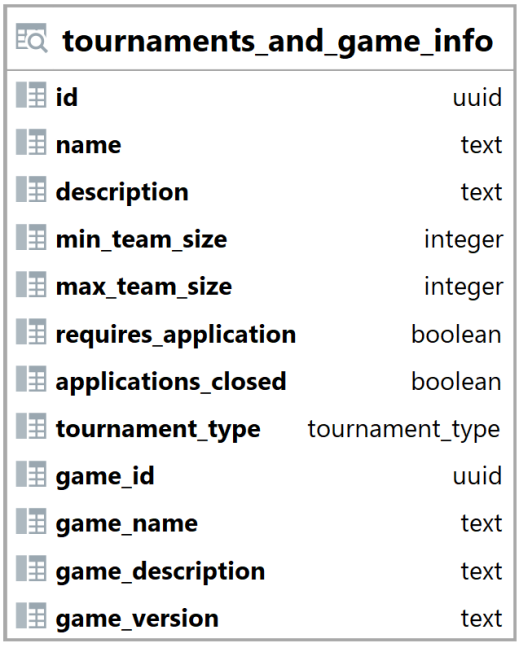
Sloupec managers obsahuje pole s manažery ve formátu json.

Kód 4: Pohled teams_with_players_and_managers (sekce 5.1.18) - příklad hodnoty sloupce managers

```
[{"id": "83230549-e822-484f-91ce-b93dd789633c", "nick": "TEST 3"}]
```

5.1.19 Pohled tournaments_and_game_info

Pohled tournaments_and_game_info spojuje tabulku tournaments a games a vytváří tak novou tabulku. Jeho hlavním úkolem je zobrazit turnaje a hranou hru. Tento pohled umožňuje backendu jednoduše načíst a vyhledat data.



Obrázek 31: Pohled tournaments_and_game_info [Vlastní tvorba]

Sloupce id, name, description, min_team_size, max_team_size, requires_application, applications_closed a tournament_type jsou brány z tabulky tournaments (sekce 5.1.4).

Sloupce game_id, game_name, game_description a game_version náleží příslušným sloupcům v tabulce games (sekce 5.1.3), bez prefixu game_.

5.1.20 Pohled tournaments_signed_up_teams

Pohled tournaments_signed_up_teams slouží k jednoduchému vyhledání a načtení dat. Jeho hlavním úkolem je usnadnit vyhledání informací o týmech, které hrají v turnaji. Tento pohled zjednodušuje načtení a vyhledání dat backendem.

Obrázek 32: Pohled tournaments_signed_up_teams [Vlastní tvorba]

Id vyjadřuje id turnaje (sekce 5.1.4).

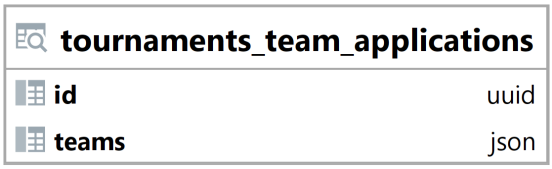
Sloupec teams obsahuje pole s týmy ve formátu json. Týmy dále obsahují informace jak o sobě, tak i o tom, kolik hráčů v turnaji hraje, a jestli je toto číslo v rozsahu pro validní velikost týmu.

Kód 5: Pohled tournaments_signed_up_teams (sekce 5.1.20) - příklad hodnoty sloupce teams

```
[{"id": "ed20021d-571a-4d19-a4af-6e829df48e66", "name": "team12",
 ↪  "description": "team", "num_playing": 1, "valid_team_size":
 ↪  false},
 {"id": "9e6f1c54-202a-4f5e-8d3e-8f1a070c3866", "name": "team15",
  ↪  "description": "team", "num_playing": 4, "valid_team_size":
  ↪  true}]
```

5.1.21 Pohled tournaments_team_applications

Pohled tournaments_team_applications slouží k jednoduchému vyhledání a načtení dat. Jeho hlavním úkolem je usnadnit vyhledání informací o týmech, které podaly přihlášku na turnaj. Tento pohled zjednodušuje načtení a vyhledání dat backendem.



Obrázek 33: Pohled tournaments_team_applications [Vlastní tvorba]

Id vyjadřuje id turnaje (sekce 5.1.4).

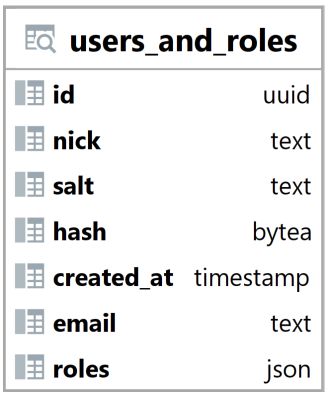
Sloupec teams obsahuje pole s týmy ve formátu json.

Kód 6: Pohled tournaments_team_applications (sekce 5.1.21) - příklad hodnoty sloupce teams

```
[{"id": "ed20021d-571a-4d19-a4af-6e829df48e66", "name": "team12",
 ↪  "description": "team"},
 {"id": "9e6f1c54-202a-4f5e-8d3e-8f1a070c3866", "name": "team15",
  ↪  "description": "team"}]
```

5.1.22 Pohled users_and_roles

Pohled users_and_roles spojuje tabulku users a roles a vytváří tak novou tabulku. Jeho hlavním úkolem je zobrazit jaké role uživatel má. Tento pohled umožňuje backendu jednoduše načíst a vyhledat jaké role uživatel má.



Obrázek 34: Pohled users_and_roles [Vlastní tvorba]

Sloupce id, nick, salt, crated_at a email jsou brány z tabulky users (sekce 5.1.1).

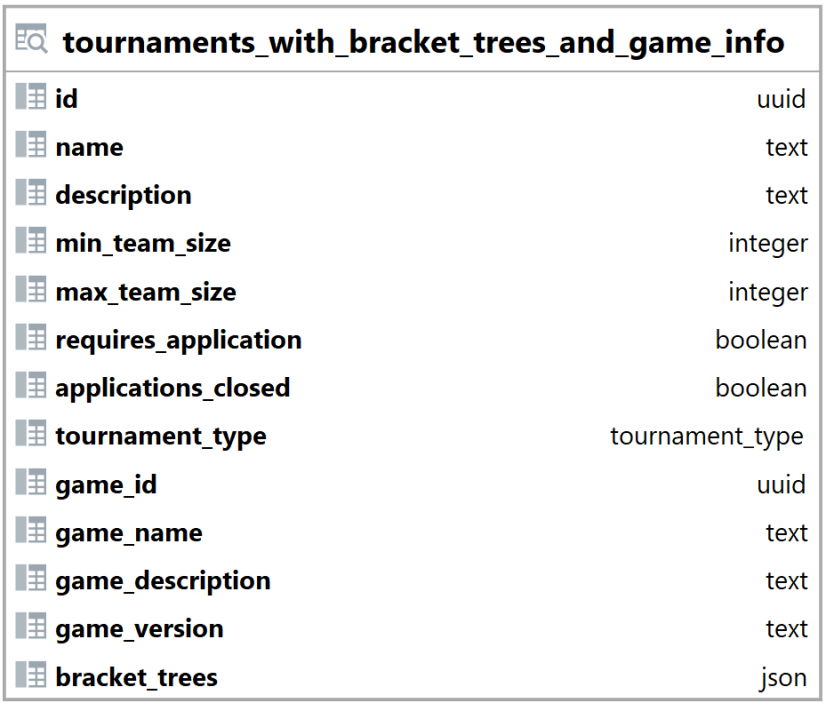
Sloupec roles obsahuje pole s rolemi uživatele ve formátu json.

Kód 7: Pohled users_and_roles (sekce 5.1.22) - příklad hodnoty sloupce roles

```
["Tournament Manager"]
```

5.1.23 Pohled tournaments_with_bracket_trees_and_game_info

Pohled tournaments_with_bracket_trees_and_game_info shromažďuje informace z tabulek tournaments, games, bracket_trees a brackets. Jeho hlavním úkolem je zobrazit informace o turnaji, hrané hře a stromech zápasů. Tento pohled backendu výrazně zjednodušuje získávání a načítání dat o stromech zápasů.



Obrázek 35: Pohled tournaments_with_bracket_trees_and_game_info [Vlastní tvorba]

Sloupce id, name, description, min_team_size, max_team_size, requires_application, applications_closed a tournament_type jsou brány z tabulky tournaments (sekce 5.1.4).

Sloupce game_id, game_name, game_description a game_version náleží příslušným sloupcům v tabulce games (sekce 5.1.3), bez prefixu game_.

Sloupec bracket_trees obsahuje pole se stromy.

Kód 8: Pohled tournaments_with_bracket_trees_and_game_info (sekce 5.1.23) - příklad hodnoty sloupce bracket_trees

```
[
  { // second bracket tree (1st losers tree)
    "id": "7951a202-0539-4e86-b5bb-ae6151606859",
    "position": 1,
    // brackets inside the tree
    "brackets": [
      { // bracket information
        "team1": {"id": "a65c56d4-6936-4b8b-ab07-1dca471ad8d4",
        ↪  "name": "team6"},
        "team2": {"id": "b5f99b63-45cc-455f-b3fe-c56907c2893a",
        ↪  "name": "team7"},
        "winner": null,
        "layer": 0,
        "team1_score": 0,
```

```
            "team2_score": 0,
            "position": 0
        }
    ]
  }
]
```

5.1.24 Pohled user_invites

Pohled user_invites slouží k přehlednějšímu zobrazení a jednoduššímu načtení dat. Jeho hlavním úkolem je zobrazit, do jakých týmů byl uživatele pozván, buď jako hráč nebo manažer. Tento pohled umožňuje jednoduché načtení a vyhledání dat backendem.

Obrázek 36: Pohled user_invites [Vlastní tvorba]

Id vyjadřuje id uživatele (sekce 5.1.1).

Sloupec player_invites obsahuje pole s týmy ve formátu json.

Kód 9: Pohled user_invites (sekce 5.1.24) - příklad hodnoty sloupce player_invites

```
[{"id": "58e99ccc-0bd8-4b42-98b7-843f4879f91d", "name":
 ↪  "test-team"},
 {"id": "c299cb77-8394-4842-b362-f15a4ae11e33", "name":
  ↪  "test-team2"}]
```

Sloupec managers obsahuje pole s týmy ve formátu json.

Kód 10: Pohled user_invites (sekce 5.1.24) - příklad hodnoty sloupce manager_invites

```
[{"id": "58e99ccc-0bd8-4b42-98b7-843f4879f91d", "name":
 ↪  "test-team"}]
```

5.1.25 Trigger pro tabulku players_to_teams_invites

Tento trigger zajišťuje, že se do tabulky players_to_teams_invites nedají vložit záznamy, které již existují v tabulce players_to_teams. Jinak řečeno zajišťuje, že se nedají pozvat hráči, kteří už jsou v týmu.

5.1.26 Trigger pro tabulku managers_to_teams_invites

Tento trigger zajišťuje, že se do tabulky managers_to_teams_invites nedají vložit záznamy, které již existují v tabulce managers_to_teams. Tímto se zajistí, že manažeři nemohou být pozváni do týmu, ve kterém se již nachází.

5.1.27 Trigger pro tabulku tournaments

Tento trigger se spustí při editaci turnaje a má za úkol udržet turnaj v logickém stavu. Jedna z funkcí je, že vymaže všechny přihlášky na turnaj, pokud se při editaci nastaví, že turnaj nepřijímá přihlášky. Další funkcí je automatické přijetí všech přihlášek, když se turnaj změní z turnaje, který vyžaduje přihlášku, na turnaj, který přihlášku nevyžaduje. Dále také kontroluje, zda se změnil typ turnaje. Pokud by se změnil a přihlášky jsou stále uzavřeny, tak vrátí chybu. Další kontrolou je kontrola, zda byly přihlášky na turnaj otevřeny. Pokud by byly otevřeny, tak se smažou veškeré stromové struktury patřící k turnaji.

5.1.28 Trigger pro tabulku teams_to_tournaments_applications

Tento trigger zajišťuje, že se do tabulky teams_to_tournaments_applications nedají vložit záznamy, které již existují v tabulce teams_to_tournaments. Jeho hlavní funkcí je zajištění, že se nadá podat přihláška na turnaj, ve kterém už tým hraje.

5.1.29 Trigger pro tabulku brackets

Tento trigger při editaci zajistí, že se týmy nepřepíšou na null, když se na backend pošle žádost o zapsání vítěze bez hodnot team1 a team2. Dále také zajišťuje, že týmy, které se nastavují do bracketu jsou přihlášeny na turnaj, ke kterému bracket patří. Pokud nějaký z týmů není přihlášený na turnaj, ke kterému bracket patří, tak se editace nepovede a vrátí se chyba.

5.1.30 Trigger pro tabulku managers_to_teams

Tento trigger zajišťuje, že když se z tabulky managers_to_teams vymaže záznam, tak proběhne kontrola, jestli tým má alespoň jednoho manažera. Pokud by manažera neměl, tak se tým smaže. Pokud tým má alespoň jednoho manažera, tak se tým nesmaže. Jinak řečeno tento trigger zajišťuje, že když se smaže uživatel, tak po něm nezůstane tým bez manažerů, který by poté musel být odstraněn manuálně.

5.1.31 Procedury handle_player_invite a handle_manager_invite

Tyto procedury obstarávají přijetí nebo odmítnutí pozvánky do týmu. Nejdříve zjistí, jestli uživatel dostal pozvánku do týmu, pokud ne, tak vrátí chybu. Poté zjistí, jestli uživatel pozvánku přijímá nebo odmítá. Podle toho, jestli pozvánku přijal nebo odmítl, tak procedura handle_player_invite upravuje tabulky players_to_teams (sekce 5.1.11) a players_to_teams_invites (sekce 5.1.12). V případě procedury handle_manager_invite jsou upravovány tabulky managers_to_teams (sekce 5.1.9) a managers_to_teams_invites (sekce 5.1.10). Konkrétněji, když uživatel pozvánku přijal, tak je přidán do týmu a pozvánka je smazána, a když pozvánku odmítl, tak je pozvánka smazána a do týmu přidán není.

5.1.32 Procedury invite_players_to_team a invite_managers_to_team

Tyto procedury obstarávají pozvání uživatelů do týmu. Nejdříve zjistí, jestli tým existuje, pokud neexistuje, tak vrátí chybu. Dále zjistí, jestli uživatel, který chce pozvat jakéhokoli jiného uživatele do týmu, je manažerem týmu, pokud není manažerem, tak vrátí chybu. Poté, co proběhnou tyto kontroly, je vytvořena pozvánka v tabulce players_to_teams_invites (sekce 5.1.12) pro pozvání hráče a pro pozvání manažera je pozvánka vytvořena v tabulce managers_to_teams_invites (sekce 5.1.10).

5.1.33 Procedury remove_players_from_team a remove_managers_from_team

Tyto procedury obstarávají odebrání uživatelů z týmu. Nejdříve zjistí, jestli tým existuje, pokud neexistuje, tak vrátí chybu. Dále zjistí, jestli uživatel, který chce odebrat jiné uživatele z týmu, je manažerem týmu, pokud není manažerem, tak vrátí chybu. Poté, co proběhnou tyto kontroly, jsou uživatelé odebráni z tabulky players_to_teams (sekce 5.1.11). Pro odebrání manažera je prováděna ještě jedna kontrola a to kontrola, jestli odebíraný uživatel je stejný jako uživatel, který zavolal tuto proceduru. Pokud je odebíraný uživatel stejný, tak se z tabulky managers_to_teams (sekce 5.1.9) neodebere, ale pokud je odebíraný uživatel jiný než uživatel, co zavolal tuto proceduru, tak je odebrán.

5.1.34 Procedury delete_team a edit_team

Tyto procedury obstarávají správu týmu. Nejdříve zjistí, jestli tým existuje, pokud neexistuje, tak vrátí chybu. Dále zjistí, jestli uživatel, který zavolal tyto procedury, je manažerem týmu, pokud není manažerem, tak vrátí chybu. V případě zavolání procedury delete_team je ještě kontrolováno, jestli uživatel, který zavolal tuto proceduru, je manažer turnaje, pokud ano, tak se nevrací chyba, že není manažer týmu. Poté, co proběhnou tyto kontroly, je v případě zavolání procedury edit_team provedena editace a v případě zavolání procedury delete_team je tým smazán.

5.1.35 Funkce new_team

Tato funkce vytváří nový tým a vrací jeho id. Při vytváření nového týmu přidá uživatele, který ji zavolal, jako manažera týmu.

5.1.36 Procedura apply_for_tournament

Tato procedura obstarává podání přihlášky na turnaj. Nejprve zjistí, jestli tým již podal přihlášku na tento turnaj, pokud ano, tak vrátí chybu. Dále zjistí, jestli turnaj a tým existují, pokud ne, tak vrátí chybu. Poté zjistí, jestli uživatel, co volá tuto proceduru, je manažerem týmu, pokud není, tak vrátí chybu. Poslední kontrolou je kontrola, jestli jsou přihlášky uzavřeny, pokud jsou uzavřeny, tak se také vrátí chyba. Když proběhnou všechny tyto kontroly a ani jedna nevrátí chybu, tak je podle hodnoty sloupce requires_application uložené v tabulce tournaments (sekce 5.1.4) přidán tým buď do tabulky teams_to_tournaments_applications (sekce 5.1.8), a nebo do tabulky teams_to_tournaments (sekce 5.1.7).

5.1.37 Procedura set_playing

Tato procedura přidává nebo odebírá hráče z turnaje. Jinak řečeno nastavuje, jestli hráč v tomto turnaji bude hrát hrát či ne. Nejprve zjistí, jestli tým existuje, pokud neexistuje, tak vrátí chybu. Dále zjistí, jestli uživatel, který zavolal tuto proceduru, je manažerem týmu, pokud není manažerem, tak vrátí chybu. Další kontrola zjistí, jestli je tým přijat na tento turnaj, pokud ne, tak vrátí chybu. Poslední kontrola je, zda jsou přihlášky uzavřeny, pokud jsou uzavřeny, tak se také vrátí chyba, že již nelze měnit kdo hraje v turnaji. Poté, co proběhnou všechny tyto kontroly a ani jedna nevrátí chybu, tak jsou do tabulky players_to_tournaments_playing (sekce 5.1.13) hráči přidáni, nebo jsou z ní hráči odebráni.

5.1.38 Procedura handle_application

Tato procedura obstarává přijetí nebo odmítnutí přihlášky na turnaj. Nejdříve zjistí, jestli přihláška existuje, pokud ne, tak vrátí chybu. Podle toho, jestli je přihláška přijata nebo odmítnuta, tak procedura handle_application upravuje tabulky teams_to_tournaments (sekce 5.1.7) a teams_to_tournaments_applications (sekce 5.1.8). Konkrétněji, když manažer turnaje přihlášku přijme, tak je přidán záznam do tabulky teams_to_tournaments (sekce 5.1.7) a přihláška je smazána z tabulky teams_to_tournaments_applications (sekce 5.1.8), a když přihlášku odmítne, tak je přihláška smazána z tabulky teams_to_tournaments_applications (sekce 5.1.8) a do turnaje tým přidán není.

5.1.39 Funkce handle_leave_tournament

Tato funkce obstarává odstranění týmu z turnaje a vrací typ boolean. Když tato funkce vrátí hodnotu true, tak vše proběhlo v pořádku a není potřeba žádné manuální úpravy. Pokud ale tato funkce vrátí hodnotu false, tak to znamená, že turnaj již začal a byly vygenerovány brackety, které je potřeba upravit manuálně.

Nejdříve je zjištěno, jestli tým existuje, pokud neexistuje, tak se vrátí chyba. Dále se zjistí, jestli uživatel, který zavolal tuto funkci, je manažerem týmu nebo manažerem turnaje, pokud není, tak vrátí chybu. Poslední kontrola je, zda jsou přihlášky uzavřeny, pokud jsou uzavřeny, tak tato funkce vrátí hodnotu false. Poté, co proběhnou všechny kontroly a ani jedna nevrátí chybu, tak je vrácena hodnota true a tým je vymazán z tabulek teams_to_tournaments (sekce 5.1.7), teams_to_tournaments_applications (sekce 5.1.8) a players_to_tournaments_playing (sekce 5.1.13).

5.2 Backendová část

Tato část se zabývá implementací serverových funkcí.

5.2.1 JWT (Json Web Token)

Pro autentizaci uživatele je použit json web token, ve kterém je uloženo id uživatele.

K realizaci autentizace pomocí json web tokenu je vytvořen JwtMiddleware. Toto softwarové lepidlo slouží k extrakci dat z tokenu, co uživatel pošle na server v hlavičce požadavku. Token extrahovaný z hlavičky je dále dekódován a ověřen. U tokenu je ověřováno několik věcí, jako je datum expirace, podpis a datum, od kdy je platný. Dále jsou také načteny role uživatele a vloženy do úložiště, kde je později zpracovatel požadavku může nalézt.

Při extrakci je nutné používat hlavičku „authorization,“ protože je výhradně používaná na autentizaci nebo autorizaci pomocí tokenů.

Kód 11: JWT extrakce hlavičky

```
let auth_header_value =
    req.headers().get(header::AUTHORIZATION).map(|v| v.clone());
```

Dále je hlavička dekódována pomocí funkce decode_jwt, která vrátí dekódovaná data tokenu, nebo nevrátí nic. Když nevrátí nic, tak to může znamenat, že uživatel poslal neplatný token, nebo není přihlášen. Z pohledu serveru je to jedno, protože tak jako tak uživatel nesmí mít přístup k funkcím, které vyžadují přihlášení.

Kód 12: JWT dekódování tokenu

```
fn decode_jwt(
    header_value: Option<&HeaderValue>,
    decoding_key: &DecodingKey,
    validation: &Validation,
) -> Option<Claims> {
    let Some(val) = header_value else {
        return None;
    };
    let Ok(val) = val.to_str() else {
        return None;
    };
    if !val.starts_with("Bearer ") {
        return None;
    }
    match jsonwebtoken::decode::<Claims>(&val[7..], decoding_key,
        validation) {
        Ok(data) => Some(data.claims),
        Err(_) => None,
    }
}
```

Poté, co je token dekódován a jsou načteny role uživatele (pokud je přihlášen), tak jsou uživatelská data, která jsou oddělena od dat specifických ke správě tokenu, vložena do požadavkového úložiště. Tato data mají také speciální funkce a strukturu pro detekci změny uživatelských dat. Po vložení dat do požadavkového úložiště je zavolána další funkce v řetězci, tato funkce může být další middleware nebo zpracovatel požadavku.

Kód 13: JWT vložení dat do úložiště a zavolání další funkce

```
let ext = Rc::new(RefCell::new(AuthDataInner {
    changed: false,
    data: claims.map(|c| c.data),
}));

```

```
req.extensions_mut().insert(ext.clone());

let mut res = svc.call(req).await?;
```

Po zavolání další funkce v řetězci, je získána odpověď. Tato odpověď je dále upravena, aby vracela token v hlavičce odpovědi. Uživatelská data jsou zkontrolována, jestli byla změněna, pokud ano, tak je vytvořen nový token, pokud ne, tak se použije starý token nebo žádný token. Nevracení tokenu v hlavičce je použito, jen když uživatel není přihlášen, nebo byl právě odhlášen, při této možnosti není do odpovědi vkládána hlavička „authorization.“

Kód 14: JWT práce s tokenem v odpovědi

```
let inner_data = ext.borrow();
if inner_data.changed {
    if let Some(user_data) = inner_data.data.as_ref() {
        let token = encode_jwt(user_data.to_owned(), &encoding_key,
         ↪  token_ttl);
        res.headers_mut().insert(
            header::AUTHORIZATION,
            HeaderValue::from_str(&format!("Bearer
             ↪  {token}")).unwrap(),
        );
    } else {
        res.headers_mut().remove(header::AUTHORIZATION);
    }
} else if let Some(val) = auth_header_value {
    res.headers_mut().insert(header::AUTHORIZATION, val);
}
```

Nový token je vytvořen pomocí funkce encode_jwt, která bere řadu parametrů. Tato funkce vytvoří nový token z uživatelských dat pomocí klíče a délky platnosti tokenu v sekundách.

Kód 15: JWT vytvoření tokenu

```
fn encode_jwt(user_data: UserData, encoding_key: &EncodingKey,
 ↪  token_ttl: u64) -> String {
    let now =
     ↪  SystemTime::now().duration_since(UNIX_EPOCH).unwrap();
    let claims = Claims {
        exp: now.as_secs() + token_ttl,
        iat: now.as_secs(),
        nbf: now.as_secs(),
        data: user_data,
    };
    jsonwebtoken::encode(&jsonwebtoken::Header::default(), &claims,
     ↪  encoding_key).unwrap()
```

```
}
```

5.2.2 Pomocné hašovací funkce

Pro registraci a přihlášení uživatele je použit hašovací algoritmus argon2. Pro jednodušší použití byly vytvořeny pomocné hašovací funkce.

Jednou z těchto funkcí je funkce make_salt, která vytvoří unikátní sůl pro hašování. Tato sůl je 128 znaků dlouhá a každý znak v této sekvenci může nabýt jedné ze 71 hodnot. Jinak řečeno tento řetězec může nabýt $9.1426 \cdot 10^{236}$ unikátních hodnot.

Kód 16: Pomocná funkce make_salt

```rust
pub fn make_salt() -> String {
    use rand::Rng;
    const CHARSET: &[u8] = b"ABCDEFGHIJKLMNOPQRSTUVWXYZ\
                             abcdefghijklmnopqrstuvwxyz\
                             0123456789)(*&^%$#@!~";
    const SALT_LEN: usize = 128;
    let mut rng = rand::thread_rng();

    let salt: String = (0..SALT_LEN)
        .map(|_| {
            let idx = rng.gen_range(0..CHARSET.len());
            CHARSET[idx] as char
        })
        .collect();
    salt
}
```

Další funkce jsou na ověřování a hašování. Hašovací funkce z hesla a soli vytvoří hash o 32 bytech. Ověřovací funkce ověřuje hash oproti zadanému heslu a soli a vrací, jestli jsou stejné.

Kód 17: Pomocné funkce make_hash a verify_password

```rust
pub fn make_hash(password: &str, salt: &str) -> [u8; 32] {
    argon2rs::argon2i_simple(password, salt)
}

pub fn verify_password(hash: &[u8; 32], salt: &str, password: &str)
 ↪  -> bool {
    make_hash(password, salt) == *hash
}
```

5.2.3 Autorizace

Autorizace slouží k zjištění práv uživatele. Pro tento účel jsou implementovány dvě struktury, jedno makro a extrakční funkce pro autority.

Funkce pro extrakci autorit z požadavku slouží k jednoduché a jednotné kontrole oprávnění uživatele. Tato funkce extrahuje jak role, tak stav uživatele. Níže je popsaná implementace funkce pro extrakci autorit.

Kód 18: Implementace funkce pro extrakci autorit s popiskama

```
pub async fn extract(req: &ServiceRequest) ->
 ↪  Result<HashSet<String>, Error> {
    // požadavkové úložiště si jen pučíme, nebudeme do něj
     ↪  zapisovat
    let extensions = req.extensions();
    // získání uživatelských údajů
    let auth_data = extensions
        .get::<Rc<RefCell<AuthDataInner>>>()
        .expect("You most likely forgot to add JwtMiddleware");
    // data si jen pučíme, bez možnosti editace
    let borrow = auth_data.borrow();
    // vnitřní data checeme jako referenci
    let data = borrow.data.as_ref();
    // vytvoříme kolekci autorit z rolí
    let mut set = data
        .map(|data| {
            data.roles
                .clone()
                .into_iter()
                .map(|e| format!("role::{e}"))
                .collect()
        })
        .or(Some(HashSet::new()))
        .unwrap();
    // do kolekce vložíme stav uživatele
    if data.is_some() {
        set.insert("state::LoggedIn".to_owned());
    } else {
        set.insert("state::LoggedOut".to_owned());
    }
    // vrátíme vytvořenou kolekci
    Ok(set)
}
```

Struktura LoggedInUser zajišťuje, že je uživatel přihlášen a vrací id uživatele. Níže je popsaná implementace traitu FromRequest pro strukturu LoggedInUser.

Kód 19: Implementace LoggedInUser s popiskama

```rust
fn from_request(req: &HttpRequest, _: &mut Payload) -> Self::Future {
    // požadavkové úložiště si jen pučíme, nebudeme do něj zapisovat
    let extensions = req.extensions();
    // získání uživatelských údajů
    let auth_data = extensions
        .get::<Rc<RefCell<AuthDataInner>>>()
        .expect("You most likely forgot to add JwtMiddleware");
    // data si jen pučíme, bez možnosti editace
    let borrow = auth_data.borrow();
    // vnitřní data checeme jako referenci
    let data = borrow.data.as_ref();

    // zkontrolujeme zda je uživatel přihlášen
    let res = if let Some(user_data) = data {
        // pokud ano získáme jeho id
        Ok(LoggedInUser { id: user_data.id })
    } else {
        // pokud ne, tak vrátíme chybu
        let err = common::Error::new("not logged in");
        Err(JsonError::new(err, StatusCode::UNAUTHORIZED))
    };

    // vrátíme výsledek
    ready(res)
}
```

Další strukturou je struktura LoggedInUserWithAuthorities. Tato struktura zajišťuje, že je uživatel přihlášen a vrací id uživatele a jeho autority. Níže je popsaná implementace traitu FromRequest pro strukturu LoggedInUserWithAuthorities.

Kód 20: Implementace LoggedInUserWithAuthorities s popiskama

```rust
fn from_request(req: &HttpRequest, _: &mut Payload) -> Self::Future {
    // požadavkové úložiště si jen pučíme, nebudeme do něj zapisovat
    let extensions = req.extensions();
    // získání uživatelských údajů
    let auth_data = extensions
        .get::<Rc<RefCell<AuthDataInner>>>()
        .expect("You most likely forgot to add JwtMiddleware");
    // data si jen pučíme, bez možnosti editace
    let borrow = auth_data.borrow();
    // vnitřní data checeme jako referenci
    let data = borrow.data.as_ref();

    // zkontrolujeme zda je uživatel přihlášen
    let res = if let Some(user_data) = data {
        // získáme autority uživatele
```

```
        let authorities =
         ↪  extensions.get::<AuthDetails>().unwrap().clone();
        // vrátíme jeho id a autority
        Ok(LoggedInUserWithAuthorities {
            id: user_data.id,
            authorities,
        })
    } else {
        // pokud není přihlášen, tak vrátíme chybu
        let err = common::Error::new("not logged in");
        Err(JsonError::new(err, StatusCode::UNAUTHORIZED))
    };

    // vrátíme výsledek
    ready(res)
}
```

Makro check_user_authority slouží ke kontrole, jestli uživatel vlastní danou autoritu. Pokud uživatel vlastní danou autoritu, tak se nic nestane, ale pokud danou autoritu nevlastní, tak se vrátí chyba.

Kód 21: Implementace check_user_authority makra

```
#[macro_export]
macro_rules! check_user_authority_macro {
    ($user:ident, $a:expr) => {
        use actix_web_grants::authorities::AuthoritiesCheck;
        if !$user.authorities.has_authority($a) {
            let err = crate::common::Error::new(format!("user is
             ↪  missing \"{}\" authority", $a));
            return crate::macros::resp_403_Forbidden_json!(err);
        }
    };
}

#[allow(unused_imports)]
pub use check_user_authority_macro as check_user_authority;
```

5.3 Testování a ladění

Testování backendu je výhradně děláno přes unit testy. Pro vytváření unit testu je ve frameworku actix-web vytvořeno plno pomocných funkcí. Spolu s vlastními implementacemi různých testovacích a pomocných funkcí probíhá testování. Vlastní implementace těchto funkcí mohou být nalezeny v souboru tests.rs[1] v hlavním zdrojovém adresáři.

[1] https://github.com/HANDZCZ/bc/blob/main/backend/src/tests.rs

Kód 22: Zkrácená ukázka výsledku běhu unit testů

```
┌« bc/backend on  main
└» cargo test
    Finished test [unoptimized + debuginfo] target(s) in 0.29s
     Running unittests src\main.rs
      ↪  (target\debug\deps\backend-176820a2e66a0de0.exe)

running 70 tests
test games::get::tests::test_bad_req ... ok
test games::get::tests::test_ok ... ok
test users::get_all::tests::test_ok ... ok
.
.
.
test tournaments::signed_up_teams::get::tests::test_ok ... ok
test tournaments::team_applications::get::tests::test_ok ... ok
test users::login::tests::test_ok ... ok

test result: ok. 70 passed; 0 failed; 0 ignored; 0 measured; 0
 ↪  filtered out; finished in 0.33s
```

Jedna z nejjednodušších implementací unit testů je v ukázce níže.

Kód 23: Ukázka jedné z nejjednodušší implementací unit testů

```rust
#[cfg(test)]
mod tests {
    use actix_web::test;

    use super::*;
    use crate::tests::*;
    const URI: &str = "/users";

    #[actix_web::test]
    async fn test_ok() {
        let (app, rollbacker, _pool) = get_test_app().await;
        let (_auth_header, id) =
         ↪  new_user_insert_random(&app).await;

        let req = test::TestRequest::get()
            .uri(&format!("{}/{}", URI, id))
            .to_request();
        let resp = test::call_service(&app, req).await;
        rollbacker.rollback().await;
        assert_eq!(resp.status().as_u16(), 200);
    }

    #[actix_web::test]
    async fn test_bad_req() {
        let (app, rollbacker, _pool) = get_test_app().await;

        let req = test::TestRequest::get()
```

```
            .uri(&format!("{}/{}", URI, Uuid::new_v4()))
            .to_request();
        let resp = test::call_service(&app, req).await;
        rollbacker.rollback().await;
        assert_eq!(resp.status().as_u16(), 400);
    }
}
```

6 Dostupnost a manuál pro spuštění

Tato kapitola se zabývá tím, jak spustit naprogramovanou implementaci backendu a ukázkový frontend. V této kapitole je také obsaženo, kde lze nalézt veškeré zdrojové soubory.

6.1 Dostupnost

Kódové zdroje implementace backendu, databáze a ukázkového frontendu lze nalézt na autorem vytvořeném repozitáři na GitHubu. V tomto repozitáři se také nacházejí zdrojové soubory pro vytvoření tohoto pdf souboru. Tento repozitář lze nalézt na adrese https://github.com/HANDZCZ/bc.

6.2 Spuštění pomocí kontejnerů

Pro spuštění aplikace pomocí kontejnerů je potřeba mít nainstalovaný nějaký software, který umožní spouštění a manipulaci kontejnerů, jako je například Docker.

Nejprve je potřeba naklonovat kódový repozitář. K naklonování kódového repozitáře je potřeba mít nainstalovaný git. Poté jen stačí naklonovat git repozitář pomocí příkazu níže.

Kód 24: Příkaz pro naklonování kódového repozitáře

```
git clone https://github.com/HANDZCZ/bc.git
```

Po naklonování kódového repozitáře je potřeba zkopírovat příklad docker-compose souboru a přejmenovat ho na docker-compose.yaml. Poté je potřeba tento soubor upravit podle komentářů uvedených v tomto souboru.

Na konec už jen stačí spustit aplikaci pomocí příkazu níže.

Kód 25: Příkaz pro spuštění kontejnerizované aplikace

```
docker compose up -d
```

Pokud je na systému nainstalovaný software make stačí spustit příkaz níže.

Kód 26: Příkaz pro spuštění kontejnerizované aplikace pomocí make

```
make run
```

Dále lze také využít dalších pomocných příkazů jako je zastavení aplikace (`make stop`), zastavení aplikace a odstranění dat vlastněných touto aplikací (`make nuke`) a zastavení aplikace, odstranění dat vlastněných touto aplikací a spuštění aplikace (`make run-clean`).

6.3 Přihlašovací údaje

Po prvním spuštění aplikace je do databáze přidán uživatel s právy manažera turnaje. Pro přihlášení na tohoto uživatele je nutno použít přihlašovací údaje uvedené níže.

Kód 27: Přihlašovací údaje výchozího uživatele

```
email: root@ui.ui
password: ui
```

7 Závěr

Hlavním cílem bakalářské práce bylo provést analýzu požadavků, navržení a implementace backendu pro správu e-sportových turnajů.

Prvním krokem bylo provedení analýzy požadavků v oblasti správy e-sportových turnajů. Tato analýza poskytla důležité informace pro návrh informačního systému, který by mohl být použit ke správě e-sportových turnajů.

Následně byl navržen a implementován backend informačního systému, který se zaměřuje na klíčové aspekty správy turnajů. Mezi tyto klíčové aspekty nepatří jen efektivní správa registrací, plánování a monitorování průběhu turnajů, ale také i sledování výsledků, generování bracketů a hlavně zabezpečení celého systému.

Backend je možno použít v různých typech turnajů s minimální úpravou. Vyvinutý systém je, ale pouze jen základem, na kterém by se mělo stavět. Tento základ lze samozřejmě použít i bez jakýchkoliv úprav, ale nebude nikdy plně vyhovovat organizátorům turnajů, kteří hledají specifické funkce.

Mezi možná vylepšení může patřit ověření uživatele pomocí emailu, zamčení registrace týmů, zamčení registrace uživatelů, povolení registrace uživatelů, kteří mají email na specifické doméně. Toto je jen pár vylepšení, které je možno snadněji implementovat.

Pokud by tento systém byl využit pro pořádání otevřených turnajů, například turnaje pořádány influencery pro své fanoušky, tak by bylo nutno implementovat filtrování nevhodného textu ve jméně uživatele a týmu. Zároveň by muselo být přidáno více manuální administrace, pro speciální případy. Mezi tyto případy může patřit uživatel, který vytvoří tým s nevhodným jménem, jež filtr nezachytil.

Na základě provedené analýzy požadavků byl navržen a implementován backend pro správu e-sportových turnajů a tím byl naplněn cíl teto bakalářské práce.

Toto téma je velice rozsáhle, komplikované a musí být implementováno podle specifických požadavků koncových uživatelů systému.

8 Seznam použité literatury

[1] BARNEY, Joshua a Natalie PENNINGTON. *An exploration of esports fan identity, engagement practices, and motives* [online]. 2023 [vid. 2023-11-13]. ISSN 2772-5030. Dostupné z: https://www.sciencedirect.com/science/article/pii/S2772503023000415

[2] ZEMMOUCHI-GHOMARI, Leila. *Basic Concepts of Information Systems* [online]. B.m.: IntechOpen. [vid. 2023-11-13]. Dostupné z: https://doi.org/10.5772/intechopen.97644

[3] ZWASS, Vladimir. *Information System* [online]. 10. říjen 2023 [vid. 2023-11-13]. Dostupné z: https://www.britannica.com/topic/information-system

[4] SHASHA, Dennis. *Information Systems* [online]. 2022 [vid. 2023-11-13]. Dostupné z: https://www.sciencedirect.com/journal/information-systems

[5] XU, Jianfeng, Zhenyu LIU, Shuliang WANG, Tao ZHENG, Yashi WANG, Yingfei WANG a Yingxu DANG. *Foundations and Applications of Information Systems Dynamics* [online]. 2022 [vid. 2023-11-13]. ISSN 2095-8099. Dostupné z: https://www.sciencedirect.com/science/article/pii/S2095809922004817

[6] WERDER, Karl. Esport. *Business & Information Systems Engineering* [online]. 2022, 393–399 [vid. 2023-11-13]. ISSN 1867-0202. Dostupné z: https://doi.org/10.1007/s12599-022-00748-w

[7] MOBILECODERZ. *How to Make an Esports Tournament Website?* [online]. 2023 [vid. 2023-11-13]. Dostupné z: https://mobilecoderz.com/blog/esports-tournament-website-development-guide

[8] PERFECTIONGEEKS. *How to Design an Esports Tournament Website?* [online]. 2023 [vid. 2023-11-13]. Dostupné z: https://www.perfectiongeeks.com/esports-tournament-website-design-company

[9] DAVIES, Aran. *How to Make an Esports Tournament Website?* [online]. 2023 [vid. 2023-11-13]. Dostupné z: https://www.devteam.space/blog/how-to-develop-an-esports-tournament-website/

[10] BANSAL, Prateek. *Understanding Non-Functional Requirements for Load/Performance Testing of a Backend Application on Kubernetes* [online]. 31. květen 2023 [vid. 2023-11-13]. Dostupné z: https://medium.com/@prateekbansalind/understanding-non-functional-requirements-for-load-performance-testing-of-a-backend-application-on-7602cf1e8805

[11] FOWLER, Martin a James FOWLER. *Microservices* [online]. 25. březen 2014 [vid. 2023-11-14]. Dostupné z: https://martinfowler.com/articles/microservices.html

[12] ÜNLÜ, Hüseyin, Dhia Eddine KENNOUCHE, Görkem Kılınç SOYLU a Onur DEMIRÖRS. *Microservice-based projects in agile world: A structured interview* [online]. 2024 [vid. 2023-11-14]. ISSN 0950-5849. Dostupné z: https://www.sciencedirect.com/science/article/pii/S0950584923001891

[13] FOWLER, Martin. *Microservices Trade-Offs* [online]. 1. červen 2015 [vid. 2023-11-14]. Dostupné z: https://martinfowler.com/articles/microservice-trade-offs.html

[14] AL-DEBAGY, Omar a Peter MARTINEK. *A Comparative Review of Microservices and Monolithic Architectures* [online]. 2019 [vid. 2023-11-14]. Dostupné z: https://arxiv.org/abs/1905.07997

[15] BALZAMOV, A V, D S FOMIN, A V SAVKINA, V V NIKULIN a S A FEDOSIN. Development of a methodology for migration of monolithic systems to microservice architecture using cloud technologies. *Journal of Physics: Conference Series* [online]. 2021, 012036 [vid. 2023-11-14]. Dostupné z: https://dx.doi.org/10.1088/1742-6596/2091/1/012036

[16] BLINOWSKI, Grzegorz, Anna OJDOWSKA a Adam PRZYBYŁEK. Monolithic vs. Microservice Architecture: A Performance and Scalability Evaluation. *IEEE Access* [online]. 2022 [vid. 2023-11-14]. Dostupné z: https://ieeexplore.ieee.org/document/9717259

[17] FOWLER, Martin. *MicroservicePremium* [online]. 1. červen 2015 [vid. 2023-11-14]. Dostupné z: https://martinfowler.com/bliki/MicroservicePremium.html

[18] SUSPICIOUSLOOKINGOWL. *Web Frameworks Benchmark - Benchmark Result* [online]. 20. listopad 2023 [vid. 2023-11-16]. Dostupné z: https://web-frameworks-benchmark.netlify.app/result?f=express,axum,actix,rocket,fastapi,django,flask,basicphp

[19] HOFFERBER, Ben. *Designing APIs that Enable Scalable Frontends* [online]. 21. prosinec 2017 [vid. 2023-11-20]. Dostupné z: https://rangle.io/blog/designing-apis-that-enable-scalable-frontends

[20] CODECADEMY TEAM. *What is REST?* [online]. 2023 [vid. 2023-11-20]. Dostupné z: https://www.codecademy.com/article/what-is-rest

[21] HYGRAPH. *What is GraphQL?* [online]. 2023 [vid. 2023-11-21]. Dostupné z: https://hygraph.com/learn/graphql

[22] CLARK, Brenda. *What is GraphQL: History, Components, and Ecosystem* [online]. 13. srpen 2019 [vid. 2023-11-21]. Dostupné z: https://levelup.gitconnected.com/what-is-graphql-87fc7687b042

[23] CRESPO-MARTÍNEZ, Ignacio Samuel, Adrián CAMPAZAS-VEGA, Ángel Manuel GUERRERO-HIGUERAS, Virginia RIEGO-DELCASTILLO, Claudia ÁLVAREZ-APARICIO a Camino FERNÁNDEZ-LLAMAS. SQL injection attack detection in network flow data. *Computers & Security* [online]. 2023, 103093 [vid. 2023-11-24]. ISSN 0167-4048. Dostupné z: https://www.sciencedirect.com/science/article/pii/S0167404823000032

[24] EDITORIAL STAFF. *Web Backend Security Risks* [online]. 12. listopad 2023 [vid. 2023-11-24]. Dostupné z: https://geekflare.com/web-backend-security-risk/

[25] GILLIS, Alexander. *What is insecure deserialization?* [online]. 2023 [vid. 2023-11-24]. Dostupné z: https://www.techtarget.com/searchsecurity/definition/insecure-deserialization

[26] PORTSWIGGER. *Insecure deserialization* [online]. 2023 [vid. 2023-11-24]. Dostupné z: https://portswigger.net/web-security/deserialization

[27] ZIVUKU, Shingai. *Everything You Need To Know About Frontend And Backend Authentication: Ultimate Guide* [online]. 7. červenec 2023 [vid. 2023-12-01]. Dostupné z: https://zivukushingai.medium.com/everything-you-need-to-know-about-frontend-and-backend-authentication-ultimate-guide-7142a752249c

[28] ELLIS, Danielle. *4 API Authentication Methods for a Secure REST API* [online]. 1. únor 2023 [vid. 2023-12-01]. Dostupné z: https://blog.hubspot.com/website/api-authentication

[29] LEVIN, Guy. *4 Most Used REST API Authentication Methods* [online]. 26. červenec 2019 [vid. 2023-12-01]. Dostupné z: https://blog.restcase.com/4-most-used-rest-api-authentication-methods/

[30] GUPTA, Priyank. *5 fundamental strategies for REST API authentication* [online]. 23. květen 2022 [vid. 2023-12-01]. Dostupné z: https://www.techtarget.com/searchapparchitecture/tip/5-fundamental-strategies-for-REST-API-authentication

[31] OKOLIE, Emmanuel. *Cookies-Based Authentication Vs Session-Based Authentication* [online]. 17. březen 2023 [vid. 2023-12-21]. Dostupné z: https://dev.to/emmykolic/cookies-based-authentication-vs-session-based-authentication-1f6

[32] SUSPICIOUSLOOKINGOWL. *Web Frameworks Benchmark - Compare Frameworks* [online]. 20. listopad 2023 [vid. 2023-11-16]. Dostupné z: https://web-frameworks-benchmark.netlify.app/compare?f=express,axum,actix,rocket,fastapi,django,flask,basicphp

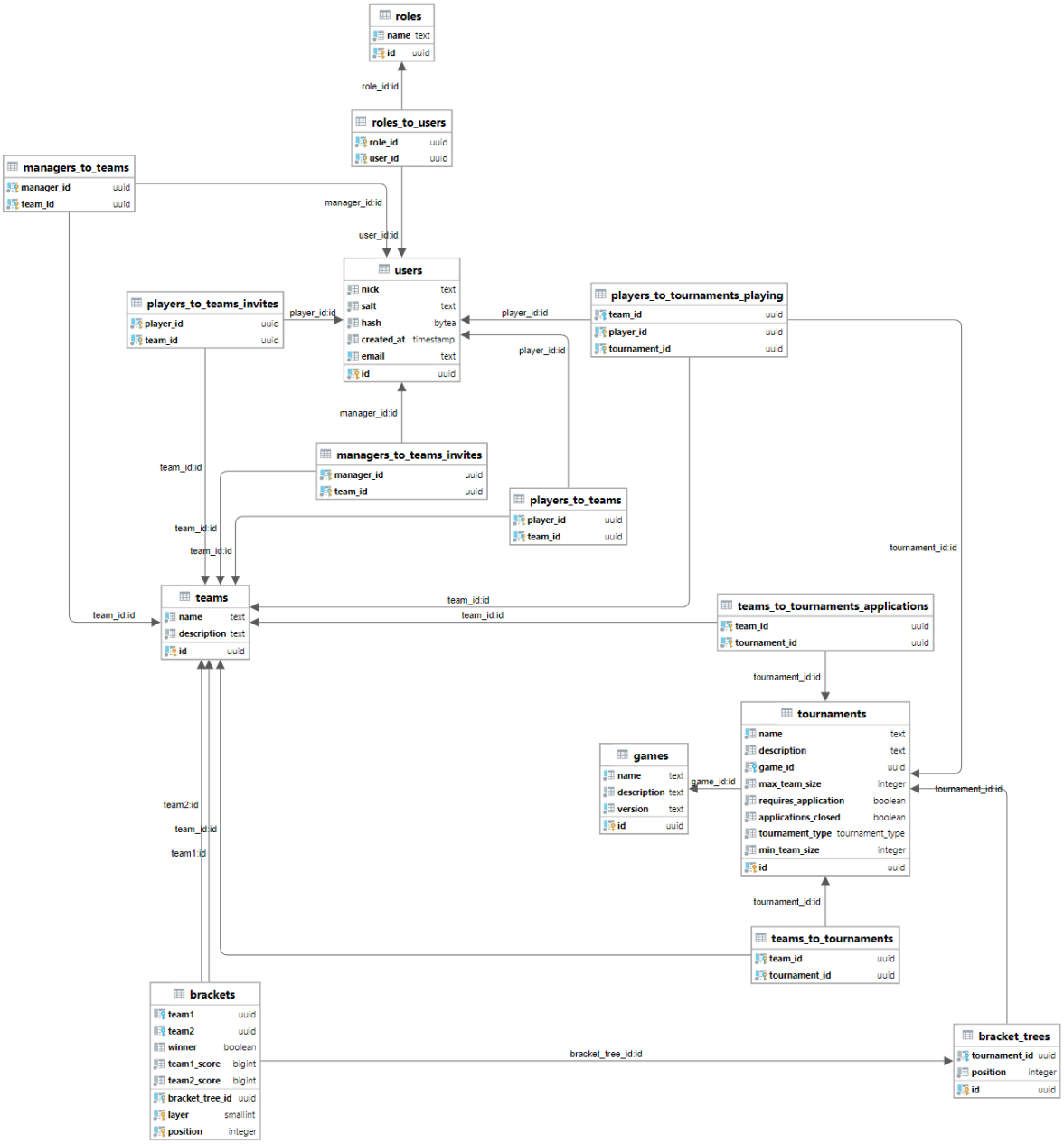
9 Seznam obrázků, tabulek a kódů

Seznam obrázků

Seznam tabulek

Seznam kódů

10 Přílohy



Obrázek 37: Diagram tabulek databáze [Vlastní tvorba]

Tabulka 1: Porovnání funkcí frameworků [Vlastní tvorba]

Feature	actix[1] (4.4)	axum[2] (0.6)	rocket[3] (0.5.0)	express[4] (4.18)	fastapi[5] (0.104)	basicphp[6] (0.9)	flask[7] (3)	django[8] (4.2)
Type-safe information extraction	✅	✅	✅	❌	❌	❌	❌	❌
Shared mutable state	✅	✅	✅	✅	✅	✅	✅	✅
Scopes	✅	❌	❌	❌	❌	❌	❌	❌
Guards	✅	❌	❌	❌	❌	❌	❌	❌
Multi-Threading	✅	✅	✅	❌	❌	❌	✅	✅
TLS / HTTPS	✅	✅	✅	✅	✅	✅	✅	❌
Keep-Alive	✅	✅	✅	✅	✅	❌	❌	❌
Graceful shutdown	✅	✅	✅	✅	✅	❌	✅	❌
Streaming response body	✅	✅	✅	✅	✅	✅	✅	✅
Configurable logging	✅	✅	❌	❌	❌	❌	❌	✅
Path normalization	✅	✅	✅	❌	❌	❌	❌	❌
Default response configuration	✅	✅	✅	❌	✅	❌	❌	❌
Multipart body	✅	✅	✅	❌	❌	✅	❌	✅
Urlencoded body	✅	✅	✅	✅	❌	✅	✅	✅

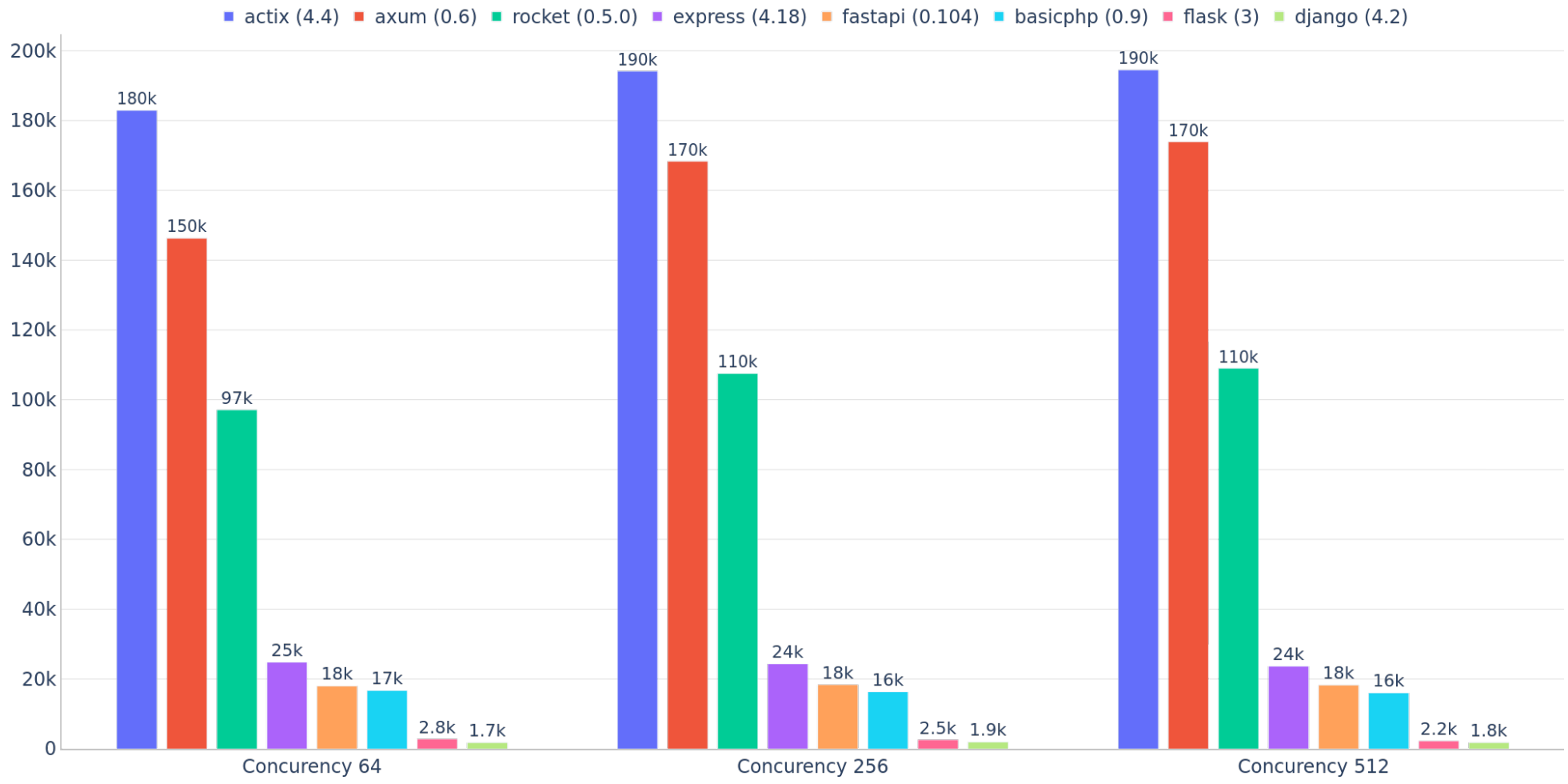
Feature	actix (4.4)	axum (0.6)	rocket (0.5.0)	express (4.18)	fastapi (0.104)	basicphp (0.9)	flask (3)	django (4.2)
Compression	✅	✅	✅	❌	✅	❌	❌	❌
Integrated unit testing	✅	✅	✅	❌	❌	❌	❌	✅
Middleware	✅	✅	✅	✅	✅	❌	✅	✅
Built-in session management	❌	❌	❌	❌	❌	✅	✅	✅
Error handlers	✅	✅	✅	✅	✅	❌	✅	❌
Static files	✅	✅	✅	✅	✅	✅	✅	✅
Websockets	✅	✅	✅	❌	✅	❌	❌	❌

[1] https://actix.rs/
[2] https://github.com/tokio-rs/axum
[3] https://rocket.rs/
[4] https://expressjs.com/
[5] https://fastapi.tiangolo.com/
[6] https://github.com/ray-ang/basicphp
[7] https://www.palletsprojects.com/p/flask/
[8] https://djangoproject.com/

Tabulka 2: Porovnání frameworků - dotazy za sekundu [18]

Jazyk	Framework	Dotazy za sekundu (64)	Dotazy za sukundu (256)	Dotazy za sukundu (512)	Rozdíl oproti prvnímu (512)
rust (1.73)	actix[1] (4.4)	182 895	194 120	194 502	0.00%
rust (1.73)	axum[2] (0.6)	146 270	168 330	173 889	11.19%
rust (1.73)	rocket[3] (0.5.0)	97 071	107 490	109 007	56.34%
javascript (ES2019)	express[4] (4.18)	24 779	24 305	23 616	156.69%
python (3.12)	fastapi[5] (0.104)	17 963	18 339	18 227	165.73%
php (8.2)	basicphp[6] (0.9)	16 652	16 305	15 977	169.64%
python (3.12)	flask[7] (3)	2 787	2 545	2 242	195.44%
python (3.12)	django[8] (4.2)	1 739	1 912	1 754	196.43%

[1] https://actix.rs/
[2] https://github.com/tokio-rs/axum
[3] https://rocket.rs/
[4] https://expressjs.com/
[5] https://fastapi.tiangolo.com/
[6] https://github.com/ray-ang/basicphp
[7] https://www.palletsprojects.com/p/flask/
[8] https://djangoproject.com/

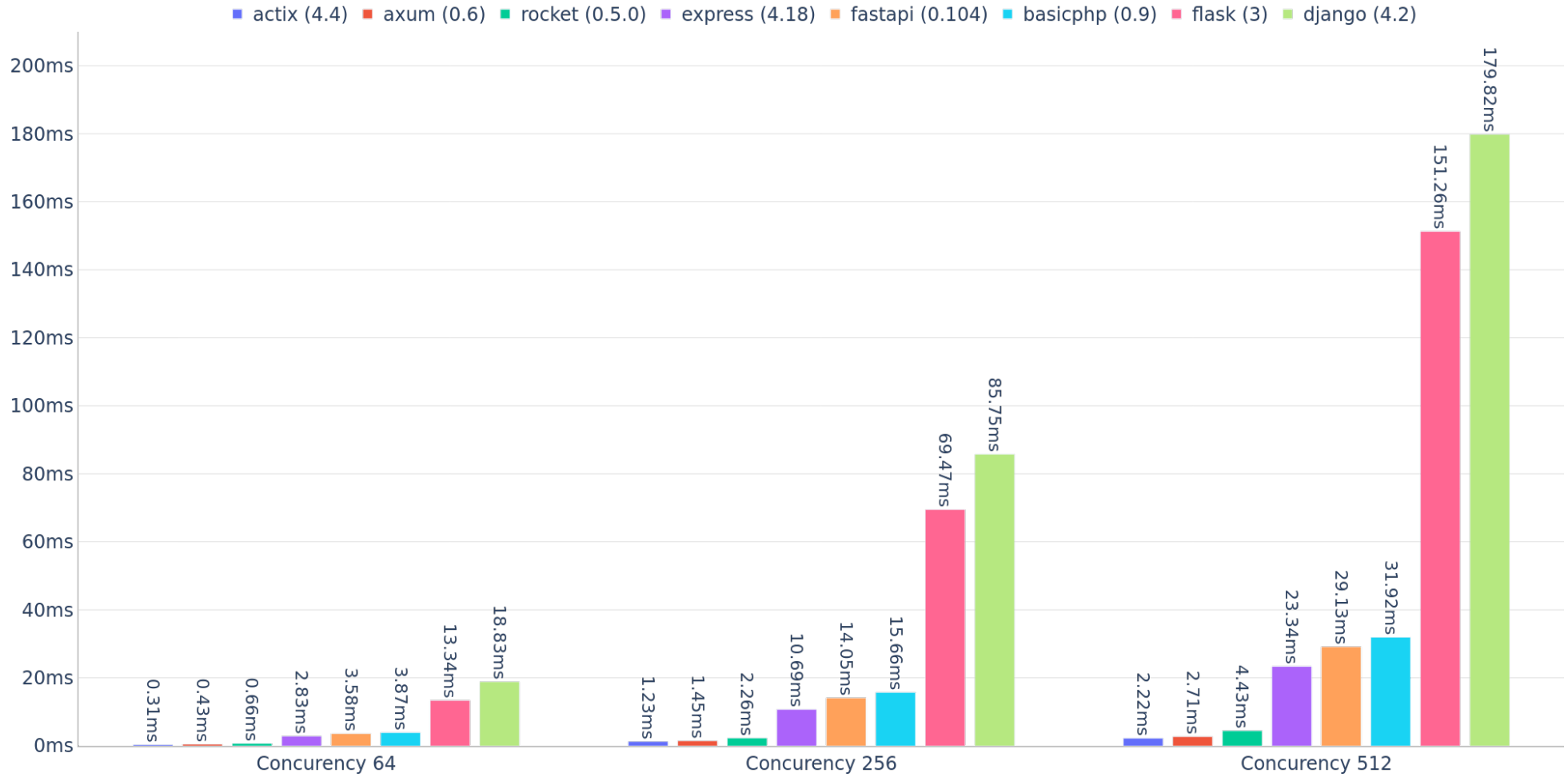


Obrázek 38: Porovnání frameworků - dotazy za sekundu [32]

Tabulka 3: Porovnání frameworků - průměrná latence [18]

Jazyk	Framework	Průměrná latence (64)	Průměrná latence (256)	Průměrná latence (512)	Rozdíl oproti prvnímu (512)
rust (1.73)	actix[1] (4.4)	0.31ms	1.23ms	2.22ms	0.00ms
rust (1.73)	axum[2] (0.6)	0.43ms	1.45ms	2.71ms	0.49ms
rust (1.73)	rocket[3] (0.5.0)	0.66ms	2.26ms	4.43ms	2.21ms
javascript (ES2019)	express[4] (4.18)	2.83ms	10.69ms	23.34ms	21.12ms
python (3.12)	fastapi[5] (0.104)	3.58ms	14.05ms	29.13ms	26.91ms
php (8.2)	basicphp[6] (0.9)	3.87ms	15.66ms	31.92ms	29.70ms
python (3.12)	flask[7] (3)	13.34ms	69.47ms	151.26ms	149.04ms
python (3.12)	django[8] (4.2)	18.83ms	85.75ms	179.82ms	177.60ms

[1] https://actix.rs/
[2] https://github.com/tokio-rs/axum
[3] https://rocket.rs/
[4] https://expressjs.com/
[5] https://fastapi.tiangolo.com/
[6] https://github.com/ray-ang/basicphp
[7] https://www.palletsprojects.com/p/flask/
[8] https://djangoproject.com/

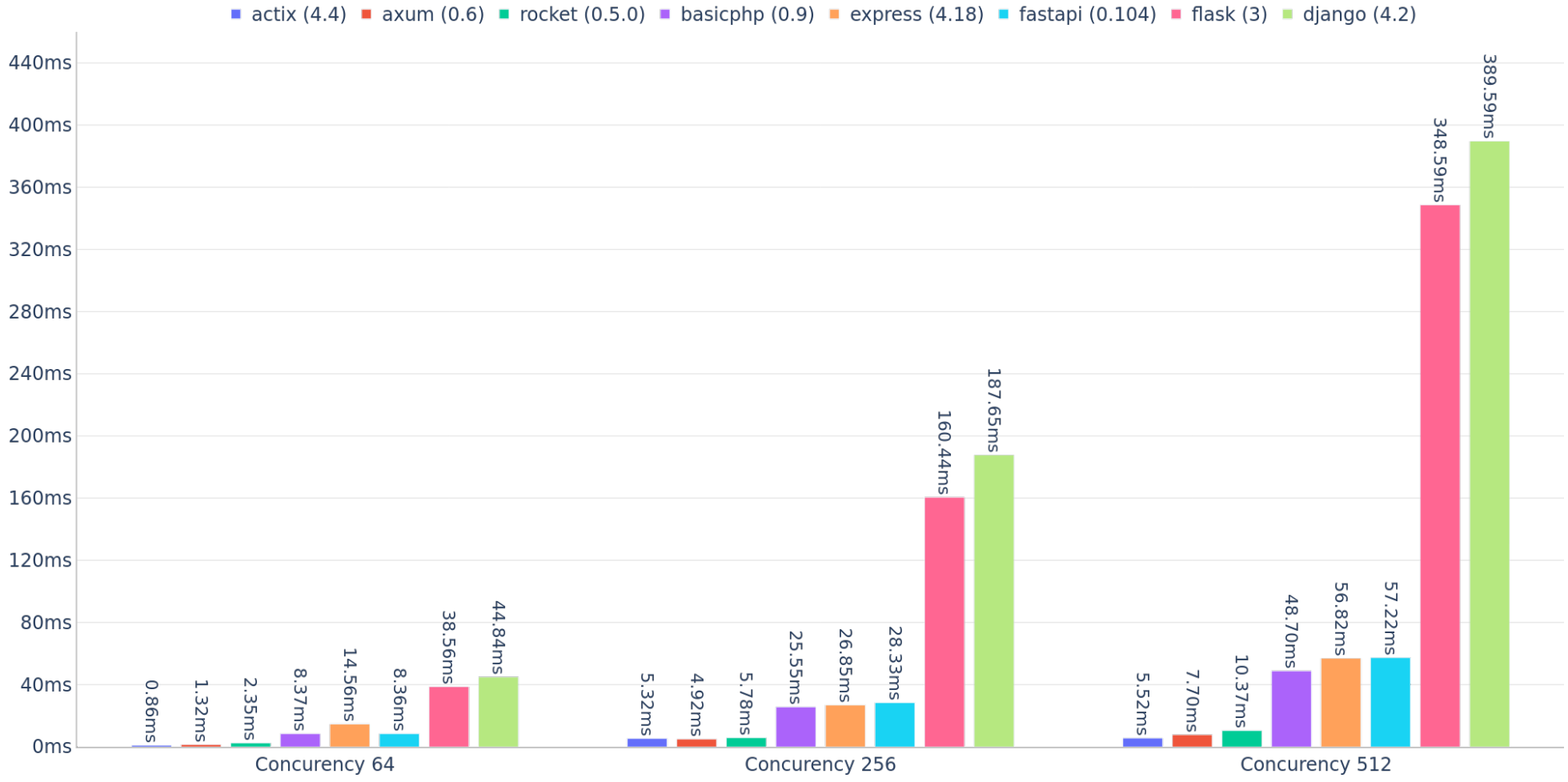


Obrázek 39: Porovnání frameworků - průměrná latence [32]

Tabulka 4: Porovnání frameworků - P99 latence [18]

Jazyk	Framework	P99 latence (64)	P99 latence (256)	P99 latence (512)	Rozdíl oproti prvnímu (512)
rust (1.73)	actix[1] (4.4)	0.86ms	5.32ms	5.52ms	0.00ms
rust (1.73)	axum[2] (0.6)	1.32ms	4.92ms	7.70ms	2.18ms
rust (1.73)	rocket[3] (0.5.0)	2.35ms	5.78ms	10.37ms	4.85ms
php (8.2)	basicphp[4] (0.9)	8.37ms	25.55ms	48.70ms	43.18ms
javascript (ES2019)	express[5] (4.18)	14.56ms	26.85ms	56.82ms	51.30ms
python (3.12)	fastapi[6] (0.104)	8.36ms	28.33ms	57.22ms	51.70ms
python (3.12)	flask[7] (3)	38.56ms	160.44ms	348.59ms	343.07ms
python (3.12)	django[8] (4.2)	44.84ms	187.65ms	389.59ms	384.07ms

[1] https://actix.rs/
[2] https://github.com/tokio-rs/axum
[3] https://rocket.rs/
[4] https://github.com/ray-ang/basicphp
[5] https://expressjs.com/
[6] https://fastapi.tiangolo.com/
[7] https://www.palletsprojects.com/p/flask/
[8] https://djangoproject.com/

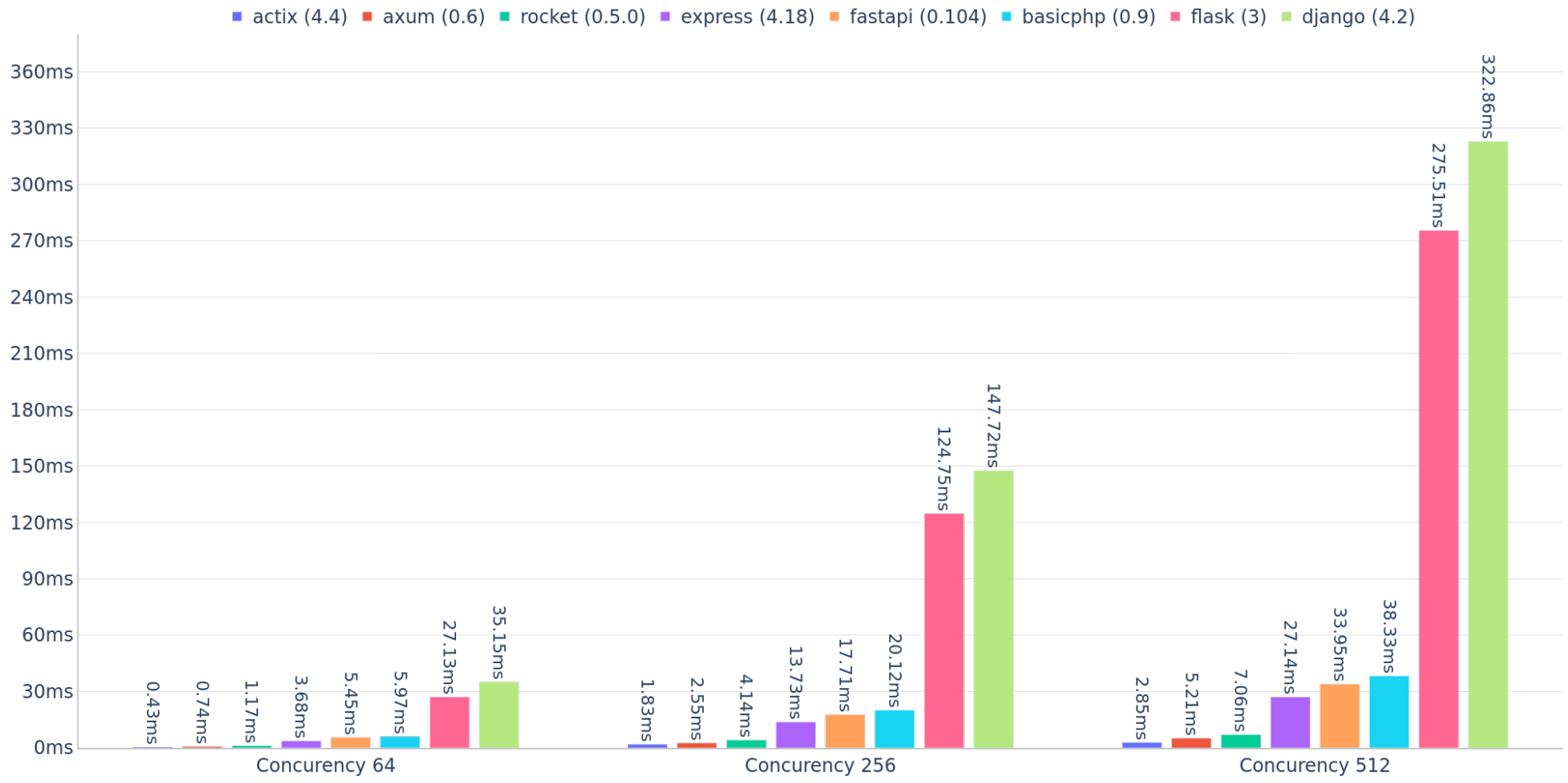


Obrázek 40: Porovnání frameworků - P99 latence [32]

Tabulka 5: Porovnání frameworků - P90 latence [18]

Jazyk	Framework	P90 latence (64)	P90 latence (256)	P90 latence (512)	Rozdíl oproti prvnímu (512)
rust (1.73)	actix[1] (4.4)	0.43ms	1.83ms	2.85ms	0.00ms
rust (1.73)	axum[2] (0.6)	0.74ms	2.55ms	5.21ms	2.36ms
rust (1.73)	rocket[3] (0.5.0)	1.17ms	4.14ms	7.06ms	4.21ms
javascript (ES2019)	express[4] (4.18)	3.68ms	13.73ms	27.14ms	24.29ms
python (3.12)	fastapi[5] (0.104)	5.45ms	17.71ms	33.95ms	31.10ms
php (8.2)	basicphp[6] (0.9)	5.97ms	20.12ms	38.33ms	35.48ms
python (3.12)	flask[7] (3)	27.13ms	124.75ms	275.51ms	272.66ms
python (3.12)	django[8] (4.2)	35.15ms	147.72ms	322.86ms	320.01ms

[1] https://actix.rs/
[2] https://github.com/tokio-rs/axum
[3] https://rocket.rs/
[4] https://expressjs.com/
[5] https://fastapi.tiangolo.com/
[6] https://github.com/ray-ang/basicphp
[7] https://www.palletsprojects.com/p/flask/
[8] https://djangoproject.com/

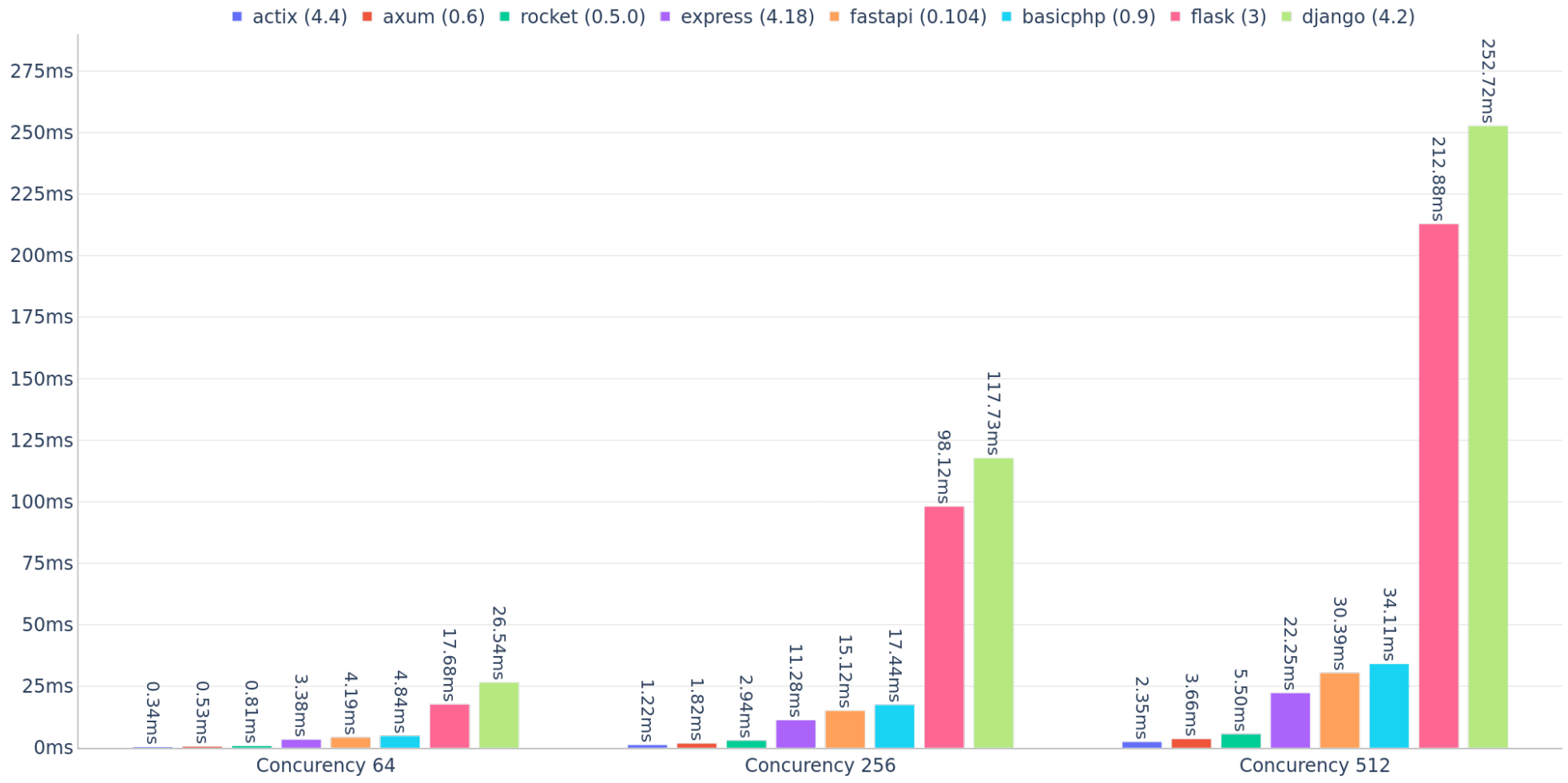


Obrázek 41: Porovnání frameworků - P90 latence [32]

Tabulka 6: Porovnání frameworků - P75 latence [18]

Jazyk	Framework	P75 latence (64)	P75 latence (256)	P75 latence (512)	Rozdíl oproti prvnímu (512)
rust (1.73)	actix[1] (4.4)	0.34ms	1.22ms	2.35ms	0.00ms
rust (1.73)	axum[2] (0.6)	0.53ms	1.82ms	3.66ms	1.31ms
rust (1.73)	rocket[3] (0.5.0)	0.81ms	2.94ms	5.50ms	3.15ms
javascript (ES2019)	express[4] (4.18)	3.38ms	11.28ms	22.25ms	19.90ms
python (3.12)	fastapi[5] (0.104)	4.19ms	15.12ms	30.39ms	28.04ms
php (8.2)	basicphp[6] (0.9)	4.84ms	17.44ms	34.11ms	31.76ms
python (3.12)	flask[7] (3)	17.68ms	98.12ms	212.88ms	210.53ms
python (3.12)	django[8] (4.2)	26.54ms	117.73ms	252.72ms	250.37ms

[1] https://actix.rs/
[2] https://github.com/tokio-rs/axum
[3] https://rocket.rs/
[4] https://expressjs.com/
[5] https://fastapi.tiangolo.com/
[6] https://github.com/ray-ang/basicphp
[7] https://www.palletsprojects.com/p/flask/
[8] https://djangoproject.com/



Obrázek 42: Porovnání frameworků - P75 latence [32]

Univerzita Hradec Králové
Fakulta informatiky a managementu

Zadání bakalářské práce

Autor: **Jan Najman**

Studium: I2100250

Studijní program: B1802 Aplikovaná informatika

Studijní obor: Aplikovaná informatika

Název bakalářské práce: **Informační systém pro správu e-sportových turnajů - návrh a implementace backendu**

Název bakalářské práce AJ: Information system for e-sports tournament management - design and implementation of backend

Cíl, metody, literatura, předpoklady:

Cílem bakalářské práce je analýza vhodných technologií, návrh a implementace backendové části informačního systému pro správu e-sportových turnajů.

Osnova:

I. Úvod
- A. Kontext práce
- B. Cíle a záměr práce
- C. Struktura práce

II. Teoretická část
- A. E-sportové turnaje a jejich správa
- B. Informační systémy a jejich role ve správě turnajů
- C. Backend jako klíčová součást informačního systému

III. Analýza požadavků
- A. Identifikace potřeb uživatelů
- B. Specifikace funkčních a nefunkčních požadavků
- C. Návrh datového modelu

IV. Návrh backendu
- A. Architektura backendu
- B. Výběr technologií a programovacích jazyků
- C. Návrh API pro komunikaci s frontendem
- D. Zabezpečení a autentizace

V. Implementace
- A. Vývoj backendového systému
- B. Testování a ladění
- C. Integrace s frontendem

VI. Zhodnocení
- A. Splnění cílů práce
- B. Hodnocení výkonu backendu
- C. Uživatelská spokojenost

VII. Závěr
- A. Shrnutí výsledků
- B. Diskuse o možných vylepšeních a budoucím vývoji

C. Důležitost a aplikovatelnost výsledků

VIII. Seznam použité literatury

IX. Přílohy
A. Diagramy a schémata
B. Kódový repozitář
C. Uživatelská příručka

ZEMMOUCHI-GHOMARI, Leila.Basic Concepts of Information Systems[online]. B.m.: IntechOpen. [vid. 2023-11-13]. Dostupné z:https://doi.org/10.5772/intechopen.97644

WERDER, Karl. Esport.Business & Information Systems Engineering[online]. 2022, 393–399 [vid. 2023-11-13]. ISSN 1867-0202. Dostupnéz: https://doi.org/10.1007/s12599-022-00748-w

BANSAL, Prateek.Understanding Non-Functional Requirements forLoad/Performance Testing of a Backend Application on Kubernetes[online].31. květen 2023 [vid. 2023-11-13]. Dostupné z:https://medium.com/@prateekbansalind/understanding-non-functional-requirements-for-load-performance-testing-of-a-backend-application-on-7602cf1e8805

FOWLER,MartinaJamesFOWLER.Microservices[online].25.březen2014[vid. 2023-11-14]. Dostupné z:https://martinfowler.com/articles/microservices.html

ÜNLÜ, Hüseyin, Dhia Eddine KENNOUCHE, Görkem Kılınç SOYLU a OnurDEMIRÖRS.Microservice-based projects in agile world: A structuredinterview[online]. 2024 [vid. 2023-11-14]. ISSN 0950-5849. Dostupnéz: https://www.sciencedirect.com/science/article/pii/S0950584923001891

FOWLER, Martin.Microservices Trade-Offs[online]. 1. červen 2015[vid. 2023-11-14]. Dostupné z:https://martinfowler.com/articles/microservice-trade-offs.html

AL-DEBAGY, Omar a Peter MARTINEK.A Comparative Review ofMicroservicesandMonolithicArchitectures[online].2019[vid.2023-11-14].Dostupné z:https://arxiv.org/abs/1905.07997

Zadávající pracoviště: Katedra informačních technologií, Fakulta informatiky a managementu

Vedoucí práce: Mgr. Hana Rohrová

Datum zadání závěrečné práce: 15.10.2021