Střední průmyslová škola elektrotechnická
a Vyšší odborná škola Pardubice

STŘEDNÍ PRŮMYSLOVÁ ŠKOLA ELEKTROTECHNICKÁ

MATURITNÍ PRÁCE – PROGRAMOVÁNÍ

Piškvorky

březen 2021 Jan Najman 4.D

„Prohlašuji, že jsem maturitní práci vypracoval(a) samostatně a použil(a) jsem literárních pramenů, informací a obrázků, které cituji a uvádím v seznamu použité literatury a zdrojů informací a v seznamu použitých obrázků a neporušil jsem autorská práva.

Souhlasím s umístěním kompletní maturitní práce nebo její části na školní internetové stránky a s použitím jejích ukázek pro výuku.“

V Pardubicích dne ....

podpis

Střední průmyslová škola elektrotechnická a Vyšší odborná škola Pardubice

ZADÁNÍ MATURITNÍ PRÁCE

Maturitní zkouška – profilová část – Maturitní projekt

Obor:	18-20-M/01 Informační technologie	**Školní rok:**	2020/2021
Jméno a příjmení žáka:	Jan Najman	**Třída:**	4.D
Téma maturitní práce:	Piškvorky		
Vedoucí maturitní práce:	RNDr. Jana Reslová		
Pracoviště vedoucího:	SPŠE a VOŠ Pardubice, Karla IV. 13, Pardubice		

Kategorie maturitní práce: PROGRAMOVÁNÍ

Téma maturitní práce (popis):

Naprogramujte aplikaci Piškvorky podle zadaných bodů.

Hlavní body – specifikace maturitní práce:

1) Hry se budou ukládat do databáze
2) Hry budou moci být hrány na tahy nebo odehrány v celku
3) Rozdělení uživatelů na přihlášené a nepřihlášené a administrátory
4) Tahy hráčů budou kontrolovány na serveru
5) Data odesílaná klientem nebo uživatelem budou mít co nejmenší velikost
6) Hry budou mít různé velikosti
7) Každý přihlášený uživatel bude mít statistiky

Způsob zpracování maturitní práce:

Maturitní práce musí být zpracována v souladu se zákonem č. 121/2000 Sb., autorský zákon.

Maturitní práce je tvořena praktickou částí (viz téma a specifikace maturitní práce výše) a písemnou prací.

Maturitní práce je realizována žákem převážně v rámci výuky ve 4. ročníku. Lze pokračovat na projektu z nižších ročníků.

Podrobné pokyny ke zpracování maturitní práce příslušné kategorie jsou uvedeny v dokumentu *MP_Zpracovani-podrobne_pokyny*.

Zpracování písemné práce musí odpovídat požadavkům uvedeným v dokumentu *MP_Formalni_stranka_dokumentace*. Písemná práce musí být rovněž zpracována v souladu s normou pro úpravu písemností [ČSN 01 6910 (2014)], s citační normou [ČSN ISO 690], se základními typografickými pravidly, pravidly sazby, gramatickými pravidly a pravidly českého pravopisu.

Pokyny k rozsahu a obsahu maturitní práce:

Rozsah a obsah praktické části maturitní práce je určen tématem a specifikací maturitní práce, rozsah písemné části maturitní práce je minimálně 15 normostran vlastního textu. Do uvedeného rozsahu se nezapočítávají úvodní listy (titulní list, prohlášení, zadání, anotace, obsah…), závěrečné listy (seznam literatury…) a přílohy. Podrobné pokyny k rozsahu a obsahu maturitní práce příslušné kategorie jsou uvedeny v dokumentu *MP_Zpracovani-podrobne_pokyny*.

Kritéria hodnocení maturitní práce a její obhajoby:

Maturitní práce a její obhajoba u maturitní zkoušky je hodnocena bodově. Celkový dosažený počet bodů je součtem bodů přidělených vedoucím práce a oponentem (maximální dosažitelný počet je 100 bodů). Výsledná známka u maturitní zkoušky je stanovena přepočtem celkového počtu bodů na známku pomocí tabulky uvedené níže.

V případě nesplnění tématu maturitní práce a v případě plagiátorství bude práce hodnocena stupněm „nedostatečný".

Tabulka bodového vyjádření hlavních kritérií a obhajoby:

- praktická část – zpracování tématu maturitní práce .....[max. 25 bodů]
- písemná část – zpracování, formální a obsahová stránka .....[max. 15 bodů]
- obhajoba maturitní práce před zkušební komisí .....[max. 10 bodů]

Podrobná kritéria pro hodnocení maturitní práce příslušné kategorie jsou uvedena v dokumentu *MP_Kriteria_hodnoceni*.

Tabulka přepočtu celkového počtu bodů na známku:

[0 .. 60 bodů]..... [5]
[61 .. 70 bodů]..... [4]
[71 .. 80 bodů]..... [3]
[81 .. 90 bodů]..... [2]
[91 .. 100 bodů]..... [1]

Požadavek na počet vyhotovení maturitní práce a její odevzdání:

Kompletní maturitní práce (praktická i písemná) se odevzdává ve stanoveném termínu vedoucímu maturitní práce. Písemná práce se odevzdává v jednom tištěném vyhotovení obsahující podepsaný přenosný nosič CD/DVD/SD s písemnou práci a sadou kompletních dat v elektronické podobě dle pokynů v dokumentu *MP_Zpracovani-podrobne_pokyny*.

Termín odevzdání maturitní práce: 26. března 2021

Délka obhajoby maturitní práce před zkušební maturitní komisí: 15 minut

Pardubice 10. října 2020

Petr Mikuláš

.....

Mgr. Petr Mikuláš, *ředitel školy*

Anotace

Práce se zabývá rychle rostoucím a čím dál více populárním systémovým jazykem Rust. Ukazuje, jak je tento systémový programovací jazyk vyspělý a jaké má výhody oproti ostatním systémovým jazykům.

Tato práce konkrétně se zabývá programováním RESTful API serveru v programovacím jazyku Rust pomocí frameworku Actix Web a programováním front-endové webové aplikace pomocí frameworku Yew.

Klíčová slova: Rust, API, RESTful, Actix, Actix Web, Web, Server, Aplikace, Front-end, Front-end aplikace, programování, framework, Yew

Annotation

This work deals with the rapidly growing and increasingly popular system programming language Rust. It shows how advanced this system programming language is and what its advantages are over other system languages.

This work specifically deals with programming RESTful API server in Rust programming language using the Actix Web framework and programming a front-end web application using framework Yew.

Keywords: Rust, API, RESTful, Actix, Actix Web, Web, Server, Application, Front-end, Front-end applications, programming, framework, Yew

Obsah

Úvod

Poslední dobou se všichni pokouší optimalizovat své web servery již při jejich programování, kvůli náporu, který by nemusely stíhat. Kdo tak neprovede může toho litovat a snaží se tento problém obejít jinak. Tím že změní jazyk nebo vytvoří repliky své aplikace a dají na ně load-balancer, tuto možnost nakonec musí využít všichni, při velmi vysokém náporu.

Bohužel nepoužívanější jazyky k naprogramování web serveru jsou PHP, JS (Node.js) nebo Python (Flask, Django). Dá se v nich rychle udělat co potřebujete, ale mají spoustu nevýhod, a hlavně všechny tyto jazyky jsou tzv. interpretované jazyky. To znamená, že na pozadí běží nějaký engine (interpreter), který musí zpracovat daný kód za běhu. Jejich největším problémem oproti kompilovaným programovacím jazykům jako jsou Rust, C++, C# je jejich rychlost a možnost zjisti chybu při kompilaci.

Tabulka 1: Porovnání rychlosti jazyků

Jazyk	Pomalejší než C++ (gcc -O2)
Rust	7 %
C#	78 %
Node.js	93 %
PHP	596 %
Python 3.5	1800 %
Python 2.7	2562 %

Rozhodl jsem se ukázat, že systémový programovací jazyk Rust je na tolik vyspělý, že se nejen zvládne vše, co jiné jazyky, ale i to, že je rychlejší než konkurence.

Vybral jsem si programování RESTful API serveru a front-endové webové aplikace z důvodu, že jsou to dnes nejpoužívanější technologie a mají budoucnost.

Téma Piškvorky jsem si vybral z více důvodů. Mělo by na nich jít perfektně předvést alespoň základy obou z frameworků. Tato hra by měla být každému povědomá a nemusím se zabývat vysvětlováním pravidel.

1 Analýza obdobných aplikací

Analýzu obdobných aplikací je dobré provádět, abyste získali představu, jak má vaše aplikace vypadat. Co chcete, aby uměla a v čem byla lepší něž ostatní aplikace.

1.1 turtlediary

Adresa: https://www.turtlediary.com/game/tic-tac-toe-multiplayer.html

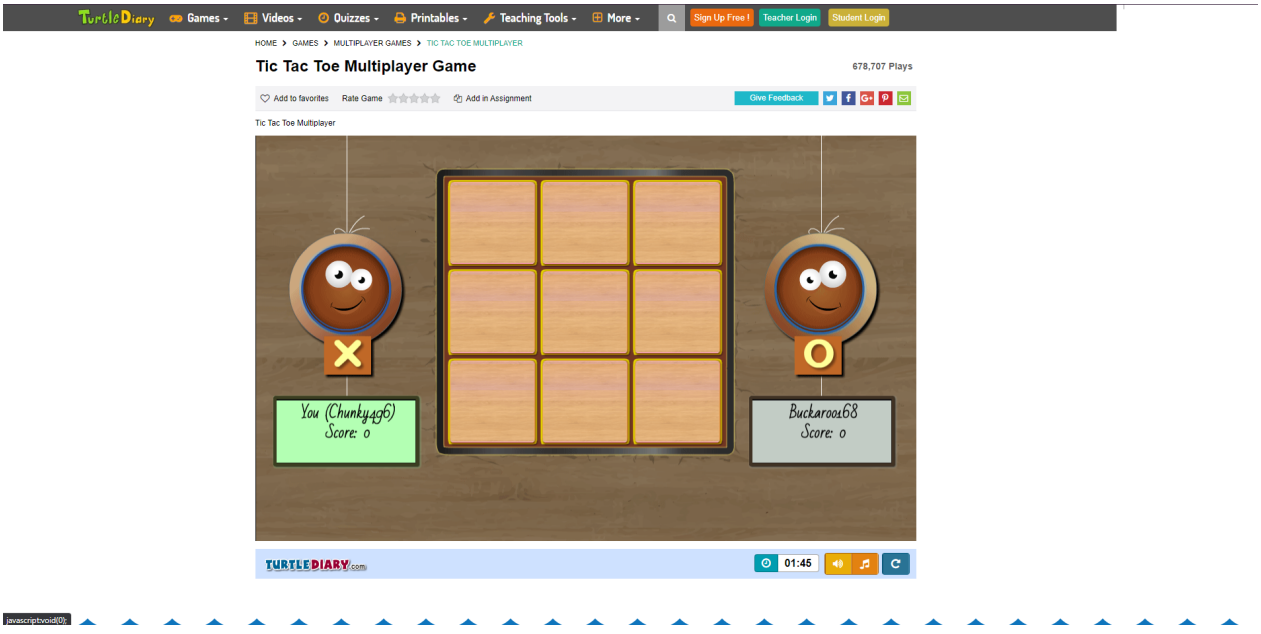
Web nabízí možnost hrát piškvorky 3x3 s náhodnými lidmi, nebo s kamarádem.

1.1.1 Kladné stránky

- ke hraní není potřeba registrace
- vizuální rozhraní hry

1.1.2 Záporné stránky

- web není responzivní
- hrací plocha je moc malá oproti zbylému volnému místu
- web nemá statistiky nebo výpis nejlepších hráčů
- na mobilním telefonu není hrací plocha vidět celá



Obrázek 1: turtlediary - https://www.turtlediary.com/game/tic-tac-toe-multiplayer.html

1.2 Ultimate Tic Tac Toe

Adresa: https://ultimate-t3.herokuapp.com

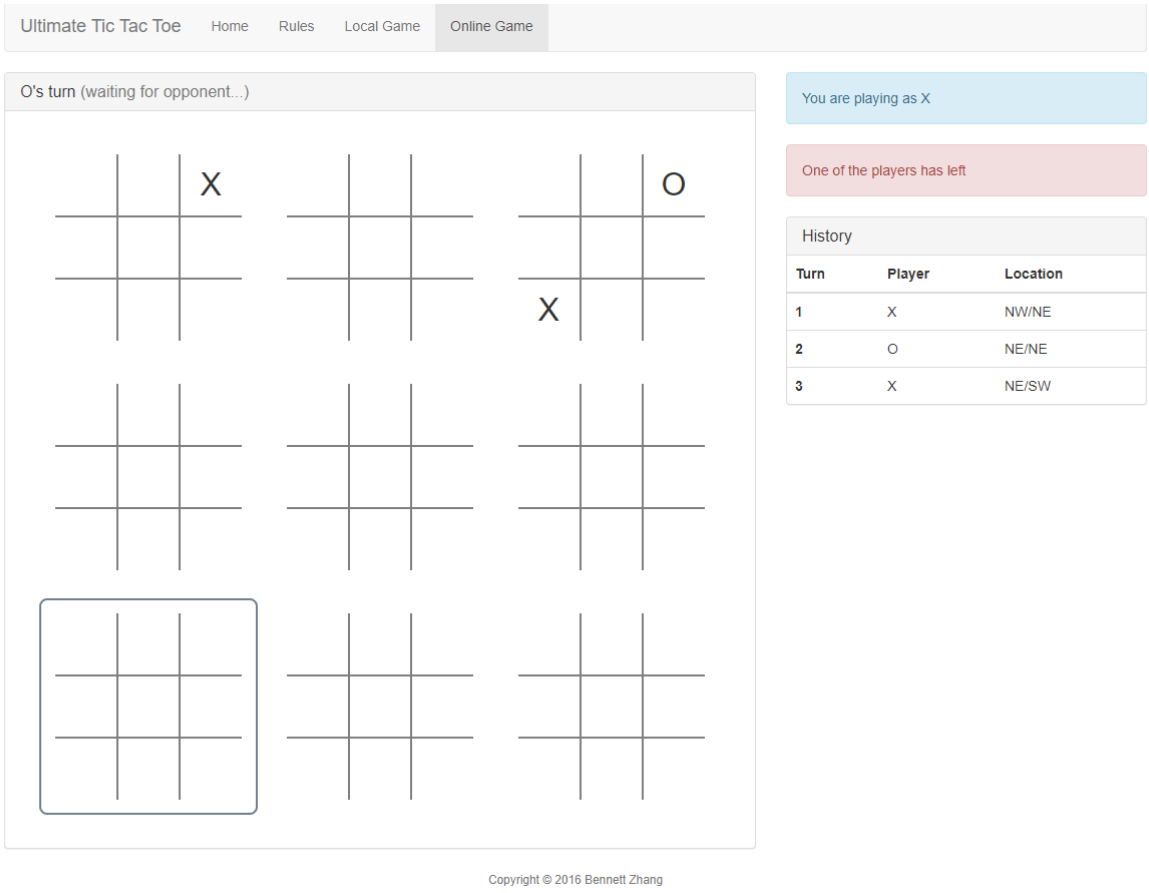
Web nabízí možnost hrát piškvorky 3x3 na více polích s kamarádem přes internet nebo lokálně.

1.2.1 Kladné stránky

- ke hraní není potřeba registrace
- čistý interface

1.2.2 Záporné stránky

- vice hracích ploch
- web nemá statistiky nebo výpis nejlepších hráčů



Obrázek 2: Ultimate Tic Tac Toe - https://ultimate-t3.herokuapp.com

1.3 gametable

Adresa: https://gametable.org/games/tic-tac-toe

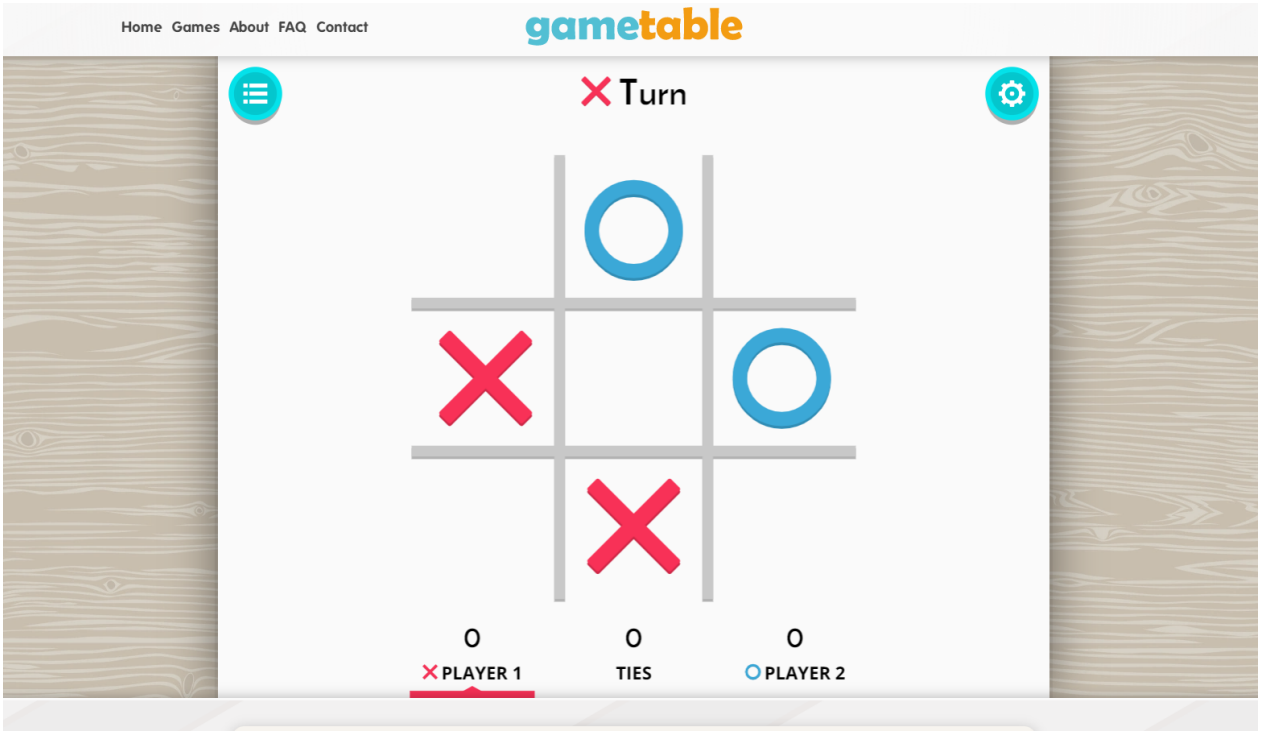
Web nabízí možnost hrát piškvorky 3x3 s kamarádem lokálně nebo proti AI.

1.3.1 Kladné stránky

- ke hraní není potřeba registrace
- čistý interface
- hra má vysvětleny pravidla pod hrací plochou

1.3.2 Záporné stránky

- není možnost hrát s někým přes internet
- web nemá statistiky nebo výpis nejlepších hráčů



Obrázek 3: gametable - https://gametable.org/games/tic-tac-toe

2 Návrh projektu

2.1 Obecná struktura

Veškeré požadavky, které nebudou odkazovat na front-end zodpovídá API. API je poté napojené na databázi a redis.

Front-end nemá sám o sobě žádný přístup k databázi nebo redisu.

Celá aplikace je napsaná tak, aby bylo možné ji dát do kontejnerů. S pomocí kontejnerů je možné aplikaci libovolně a jednoduše škálovat.

2.2 API

API spojuje vše dohromady. Poskytuje veškeré informace front-endu a zpracovává veškeré příchozí informace.

Ukládá a kontroluje uživatelské relace v redisu. Ukládá, upravuje a maže údaje v databázi.

2.3 Redis

V redisu jsou ukládány uživatelské relace, aby k nim byl rychlý přístup a jejich ověření. Každá relace má určitou životnost, kterou je možno změnit. V relaci je uložené id uživatele.

2.4 Databáze

Databáze ukládá informace o uživatelích a hrách pro dlouhodobé uložení dat.

Tabulka *users* ukládá informace o uživatelích. S vlastnostmi:

- *id* - unikátní id uživatele,
- *nick* - přezdívka uživatele,
- *gender* - pohlaví uživatele,
- *hash* - hashované heslo uživatele,
- *salt* - sůl pro hashování hesla,
- *email* - unikátní email uživatele,
- *created_at* - čas vytvoření účtu uživatele,

- *description* - popis uživatele.

Tabulka *games* ukládá informace o hrách. S vlastnostmi:
- *id* - unikátní id hry,
- *name* - název hry,
- *ended* - udává jestli hra skončila,
- *last_played* - udává kdo naposled hrál,
- *data* - data hry,
- *created_at* - čas vytvoření hry,
- *moves_needed* - počet tahů potřebných pro výhru,
- *winner* - výherce.

Tabulka *roles* ukládá role. S vlastnostmi:
- *id* - unikátní id role,
- *name* - unikátní jméno role.

Tabulka *game_requests* ukládá informace o pozvánkách na hru. S vlastnostmi
- *id* - unikátní id pozvánky,
- *name* - název pozvánky,
- *last_played* - ukládá kdy naposled hrál (určuje kdo "začal" - při stejném nastavení hry může pokaždé začínat někdo jiný)
- *created_at* - čas vytvoření pozvánky,
- *moves_needed* - počet tahů potřebných pro výhru.

Tabulka *roles_to_users* spojuje tabulky *roles* a *users*. Přiřazuje uživatelům role. S vlastnostmi:
- *user_id* - id uživatele,
- *role_id* - id role.

Tabulka *games_to_users* spojuje tabulky *games* a *users*. Přiřazuje hry k uživatelům. S vlastnostmi:
- *game_id* - id hry,
- *user_id* - id uživatele.

Tabulka *users_to_game_requests* spojuje tabulky *game_requests* a *users*. Přiřazuje pozvánky k uživatelům. S vlastnostmi:
- *user_id* - id uživatele,
- *game_request_id* - id pozvánky,

- *accepted* - udává zda uživatel pozvánku přijal.

Diagram fig. 4

2.5 Front-end

Slouží jako grafické zobrazení dat a druhořadá kontrola dat. Zobrazuje informace o uživatelích a hrách.

2.6 Administrace

Každý uživatel co má roli *Admin* má zvýšená práva. Může upravovat ostatní uživatele, vytvářet pozvánky ve kterých nemusí být, nebo může obsahovat uživatele, kteří nemohou být normálně přidáni do pozvánky. Může také vypnout skoro všechny kontroly, jako je formát jména, hesla, popisu uživatele, atd.

2.7 Design a responzivita

Design a responzivita je řešená pomocí css knihovny UIkit (https://getuikit.com/). Několik věcí je přepsáno pro vzhled aplikace. Tyto změny se nachází v souboru *uikit_addition.css*.

uikit_addition.css lst. 8

3 Zpracování praktické části

3.1 Použité technologie

Skoro celá aplikace je naprogramovaná v jazyce Rust, jen databáze se píše v jazyce sql a má několik procedur.

3.1.1 Knihovna roles

Mnou vytvořená knihovna, která při kompilaci načte role z databáze a převede je do datového typu `enum`.

- *quote* - poskytuje makro pro převod datových struktur jazyka Rust na tokeny zdrojového kódu
- *syn* - je knihovna načítaní tokenů zdrojového kódu jazyka Rust do datového typy zdrojového kódu jazyka Rust
- *proc-macro2* - umožňuje definovat makra
- *dotenv* - načte `.env` soubor do proměnných prostředí
- *postgres* - synchronní klient pro databázi PostgreSQL

Zdrojový kód knihovny `roles` lst. 12

3.1.2 Back-end

Back-end je naprogramován celý v jazyce Rust.

- *structopt* - získává parametry pro program z příkazového řádku, nebo z proměnných prostředí (Environment variables)
- *dotenv* - načte `.env` soubor do proměnných prostředí
- *thiserror* - slouží pro zacházení s chybami
- *actix-web* - hlavní knihovna pro web server
- *actix-redis* - používá se jako back-end pro knihovnu actix-session
- *actix-session* - používá se pro zacházení s uživatelskými relacemi
- *env_logger* - slouží jako back-end k výpisu akcí serveru (logging)
- *time* - používá se pro práci s časem
- *sqlx* - slouží k komunikaci s databází
- *lazy_static* - je použit k vyhodnocování proměnných jen jednou a jen při použití
- *fancy-regex* - slouží pro kontrolu dat pomocí regexu

- *serde* - knihovna k serializaci datových struktur
- *serde_json* - slouží k serializaci do formátu JSON
- *futures-util* - nástroje pro práci s `futures` (asynchronní procesy)
- *futures* - implementace `futures` knihovny a `std::future`
- *rand* - používá se pro generování náhodných věcí (čísel, vybírání položky s pole, atd.)
- *argon2rs* - hashuje hesla pomocí algoritmu Argon2
- *log* - používá se k výpisu akcí serveru (logging)
- *serde_repr* - slouží k serializaci `enum` data typu
- *bincode* - serializuje datové struktury do formátu bincode
- *uuid* - používá se pro práci s univerzálními unikátními identifikátory (UUID)
- *tokio* - slouží jako back-end pro asynchronní procesy
- *roles* - mnou vytvořená knihovna, která při kompilaci načte role z databáze a převede je do datového typu `enum`
- *actix-cors* - implementace CORS (Cross-origin resource sharing) pravidel pro Actix Web

3.1.3 Front-end

Front-end je naprogramován hlavně v jazyce Rust, ale využívají se tam i jiné jazyky, jako je HTML, CSS a JS.

Rust knihovny:

- *yew* - knihovna pro vytváření více vláknových front-endových webových aplikací s WebAssembly
- *wasm-bindgen* - knihovna usnadňující interakci na vysoké úrovni mezi moduly wasm (WebAssembly) a JavaScriptem
- *yew-router* - směrovací knihovna pro knihovnu yew
- *wee_alloc* - alokátor pro WebAssembly
- *wasm-logger* - slouží jako back-end pro k výpisu akcí do konzole prohlížeče
- *log* - používá se k výpisu akcí serveru (logging)
- *roles* - mnou vytvořená knihovna, která při kompilaci načte role z databáze a převede je do datového typu `enum`
- *lazy_static* - je použit k vyhodnocování proměnných jen jednou a jen při použití
- *fancy-regex* - slouží pro kontrolu dat pomocí regexu
- *serde* - knihovna k serializaci datových struktur

- *serde_json* - slouží k serializaci do formátu JSON
- *serde_repr* - slouží k serializaci `enum` data typu
- *time* - používá se pro práci s časem
- *bincode* - serializuje datové struktury do formátu bincode
- *strum* - poskytuje sadu maker pro snadnější práci s datovými typy `enum` a `String`

CSS a JS knihovny:

- *UIkit* - modulární front-end knihovna pro vývoj rychlých a výkonných webových rozhraní

3.1.3.1 Server

Speciálně vytvořený server pro správnou funkci front-endu.

- *structopt* - získává parametry pro program z příkazového řádku, nebo z proměnných prostředí (Environment variables)
- *dotenv* - načte `.env` soubor do proměnných prostředí
- *actix-web* - hlavní knihovna pro web server
- *env_logger* - slouží jako back-end k výpisu akcí serveru (logging)
- *actix-files* - slouží k práci se statickými soubory
- *thiserror* - slouží pro zacházení s chybami

3.2 Databáze

Spousta akcí, které back-end podniká jsou řešené skrz procedury. Tímto způsobem dojde k zjednodušení kódu na back-endu a k provedení akce není potřeba dělat několik dotazů na databázi.

Procedura pro vytvoření pozvánky lst. 9

Procedura pro úpravu uživatele lst. 10

Procedura pro úpravu pozvánky lst. 11

3.3 Back-end

3.3.1 Správa uživatelů

Uživatelé jsou umístěni v tabulce *users*. Role jsou uloženy v tabulce *roles* a jsou k uživatelům přiřazovány skrz tabulku *roles_to_users*.

Pokud má uživatel roli *Admin* tak mohou upravovat kohokoli údaje bez omezení včetně rolí a hesla.

Pokud má uživatel roli *Banned* tak nemůže vytvářet pozvánky, ani nemůže být zahrnut do pozvánky jiným uživatelem.

Úprava uživatele po kontrole dat je poté řízena procedurou lst. 10.

3.3.2 Vytváření žádostí o hru

Přihlášení uživatelé mají možnost vytvářet nové hry s různými parametry.

Nejprve se vytvoří žádost o hru, která se nachází v tabulce *game_requests*. K dané žádosti na hru se přiřadí uživatelé skrz tabulku *users_to_game_requests*.

Pozvánky nemohou vytvářet uživatelé s rolí *Banned*.

Vytvoření pozvánky po kontrole dat je poté řízeno procedurou lst. 9

3.3.3 Vytvoření hratelné hry

Po vytvoření žádosti o hru jí musí všichni hráči potvrdit a hra bude vytvořena, nebo někdo z pozvaných hráčů odmítne žádost a žádost o hru bude vymazána.

Jakmile je účast všech hráčů potvrzena, tak se vytvoří nová hra v tabulce *games*, přiřadí se k ní uživatelé skrz tabulku *games_to_users* a žádost o hru je poté vymazána.

Úpravu pozvánky po kontrole dat je poté řízena procedurou lst. 11

3.3.4 Hraní hry

Hrát můžete jen když jste na tahu a pokud hrané políčko ještě nebylo použito. Vyhraní hry se kontroluje na front-endu, pokud front-end usoudí, že hráč vyhrál tak výhru oznámí back-endu a ten výhru zkontroluje.

Back-end kontroluje, jestli je hráč na tahu, jestli hra neskončila, nebo jestli jeho tah je validní. V případě, že hráč ohlásí výhru, ale server zjistí, že to tak není, tak daný tah zahodí a odpoví chybou.

3.4 Front-end

3.4.1 Registrace

Všechna pole jsou kontrolována. Pokud nějaké pole není validní, tak se nepošlou data na back-end.

Back-end data zkontroluje a pokud zjistí, že nejsou validní, tak žádost zahodí a vrátí chybu.

Pro hashování hesla se používá 128 znaková sůl a algoritmus Argon2.

3.4.2 Profil

Zobrazuje informace o uživateli a hry, ve kterých se nachází.

Zobrazuje také počet výher, proher a remíz. Z těchto dat poté vypočítá winrate (výhry / prohry).

Pokud je uživatel na svém profilu, nebo pokud má uživatel roli *Admin*, tak se mu také zobrazí tlačítko na upravení profilu.

3.4.3 Výpis uživatelů

Zobrazuje všechny registrované uživatele a pár informací o nich.

Uživatelům s rolí *Admin* se navíc zobrazuje tlačítko upravení profilu.

3.4.4 Výpis her

Zobrazuje všechny rozehrané, nebo dohrané hry s jejich hráči a stavem hry.

3.4.5 Hraní hry

Hry jsou hrány na síti 30x30.

Uživatelé jsou zobrazováni s jejich symbolem před jménem a za jménem je napsáno, jestli jsou na tahu.

Hrát můžou jen uživatelé, kteří jsou v dané hře, ale dívat se může kdokoli.

Tahy uživatelů jsou kontrolovány, jestli jsou validní a jestli nastala výhra nebo remíza.

3.4.6 Výpis pozvánek

Zobrazuje název pozvánky (později název hry), počet tahů k vítězství a id pozvánky.

Uživatel může pozvánku přijmout, nebo odmítnout.

3.4.7 Vytváření pozvánky

Při vytváření pozvánky jsou skoro všechna pole kontrolována.

Uživatelé s rolí *Admin* mají práva na vypnutí skoro všech kontrol.

3.4.8 Úprava uživatele

Uživatel může upravovat vše, kromě jeho rolí.

Uživatel s rolí *Admin* může upravovat vše a má možnost vypnout kontrolu, která je vyžadována po ostatních uživatelích.

4 Manuál

Aplikace se dá spustit více způsoby. Buď kompilací ze zdrojového kódu, nebo pomocí kontejnerů.

4.1 Instalace potřebných nástrojů

4.1.1 Kompilace

Pro kompilaci potřebujeme nainstalovat kompilátor jazyka Rust.

Přejdeme na stránku stažení jazyka Rust https://www.rust-lang.org/tools/install a stáhneme exe soubor.

Otevřeme a zadáme 1 pro instalaci a stiskneme enter.

Po dokončení instalace všech komponentů nainstalujeme ještě wasm-pack pro kompilaci do WebAssembly.

Přejdeme na stránku stažení nástroje wasm-pack https://rustwasm.github.io/wasm-pack/installer/ a stáhneme exe soubor.

Po stažení jej nainstalujeme.

4.1.2 Použití kontejnerů

Pro použití kontejnerů potřebujeme nějaký software, který to umožňuje. Já jsem zvolil docker.

Přejdeme na stránku stažení https://hub.docker.com/editions/community/docker-ce-desktop-windows/ a stáhneme exe soubor.

Nainstalujeme a vyzkoušíme funkčnost (Hello world! kontejner).

4.2 Příprava

4.2.1 Kompilace

Kompilace není potřeba při použití kontejnerů.

Pro kompilaci aplikace je potřeba mít databázi připravenou dopředu. Stačí se přihlásit do administrace databáze a spustit sql příkazy v přiloženém sql souboru.

Zkompilujeme back-end. Vstoupíme do složky `backend` a spustíme příkaz:

Výpis 1 Příkaz pro kompilaci back-endu

```
cargo build --release
```

Zkompilujeme front-end server. Vstoupíme do složky `frontend/server` a spustíme příkaz:

Výpis 2 Příkaz pro kompilaci front-end serveru

```
cargo build --release
```

Zkompilujeme front-end. Vstoupíme do složky `frontend` a spustíme příkaz:

Výpis 3 Příkaz pro kompilaci front-endu

```
build.bat
```

4.2.2 Použití kontejnerů

Pro jednoduché použití používám docker-compose.

Pro jednoduchou konfiguraci jsem vytvořil funkční příklad nastavení.

Tento příklad si zkopírujeme a přejmenujeme z `docker-compose.yml.example` na `docker-compose.yml`.

Potom si ho upravíme podle vlastních preferencí.

Změníme tyto položky:

Výpis 4 Nastavení proměnných prostředí pro kontejnery

```
# Databáze
POSTGRES_USER: root             # Jméno uživatele databáze
POSTGRES_PASSWORD: password     # Heslo uživatele databáze

# Back-end
AUTH_TTL: 3600                  # Jak dlouho bude platit uživatelská relace
 ↪  v sekundách
AUTH_KEY: keykeykeykeykeykeykeykeykeykey   # Klíč pro uživatelské
 ↪  relace, musí mít minimálně 32 bajtů (32 znaků)
DATABASE_URL: postgres://root:password@postgres/maturitni_projekt   #
 ↪  Změníme jméno a heslo na to které jsme zvolili u databáze
ALLOWED_ORIGIN: http://mp.loc   # Adresa front-endu pro správné nastavení
 ↪  CORS
FRONTEND_DOMAIN: mp.loc         # Adresa front-endu bez protokolu pro
 ↪  přístup ke cookies na front-endu

# Front-end
API_DOMAIN: http://api.mp.loc   # Adresa back-endu pro komunikaci mezi
 ↪  front-endem a back-endem
```

Poté taky změníme překládání portů, abychom se na aplikaci vůbec dostali.

Výpis 5 Nastavení překládání portů pro kontejnery

```
backend:
  ports:
    - "127.0.5.2:80:80" # Toto znamená překládej port 80 na adresu a port
 ↪  127.0.5.2:80

frontend:
  ports:
    - "127.0.5.1:80:80" # Toto znamená překládej port 80 na adresu a port
 ↪  127.0.5.1:80
```

Samozřejmě tato aplikace má být postavená například za nginx server. Ale pokud jen testujeme na svém počítači tak můžeme do souboru `hosts` přidat tyto položky:

Výpis 6 Úprava hosts souboru pro kontejnery

```
127.0.5.1 mp.loc
127.0.5.2 api.mp.loc
```

4.3 Spuštění

4.3.1 Kompilace

Musíme mít spuštěnou databázi a redis.

1. Spustíme back-end. Stačí jen spustit exe soubor a popřípadě dodat další argumenty.
2. Spustíme front-end server. Stačí jen spustit exe soubor a popřípadě dodat další argumenty.

4.3.2 Použití kontejnerů

Spustíme příkaz, který nám stáhne, nastartuje a nakonfiguruje veškeré kontejnery:

Výpis 7 Příkaz pro start kontejnerů

```
docker-compose up -d
```

5 Závěr

6 Seznam použité literatury a zdrojů informací

7 Seznam použitých zkratek

Tabulka 2: Seznam použitých zkratek

Zkratka	Význam
API	Application Programming Interface
CORS	Cross-origin resource sharing
REST	Representational State Transfer
SPŠE	Střední průmyslová škola elektrotechnická
UUID	Universally unique identifier
VOŠ	Vysoká odborná škola

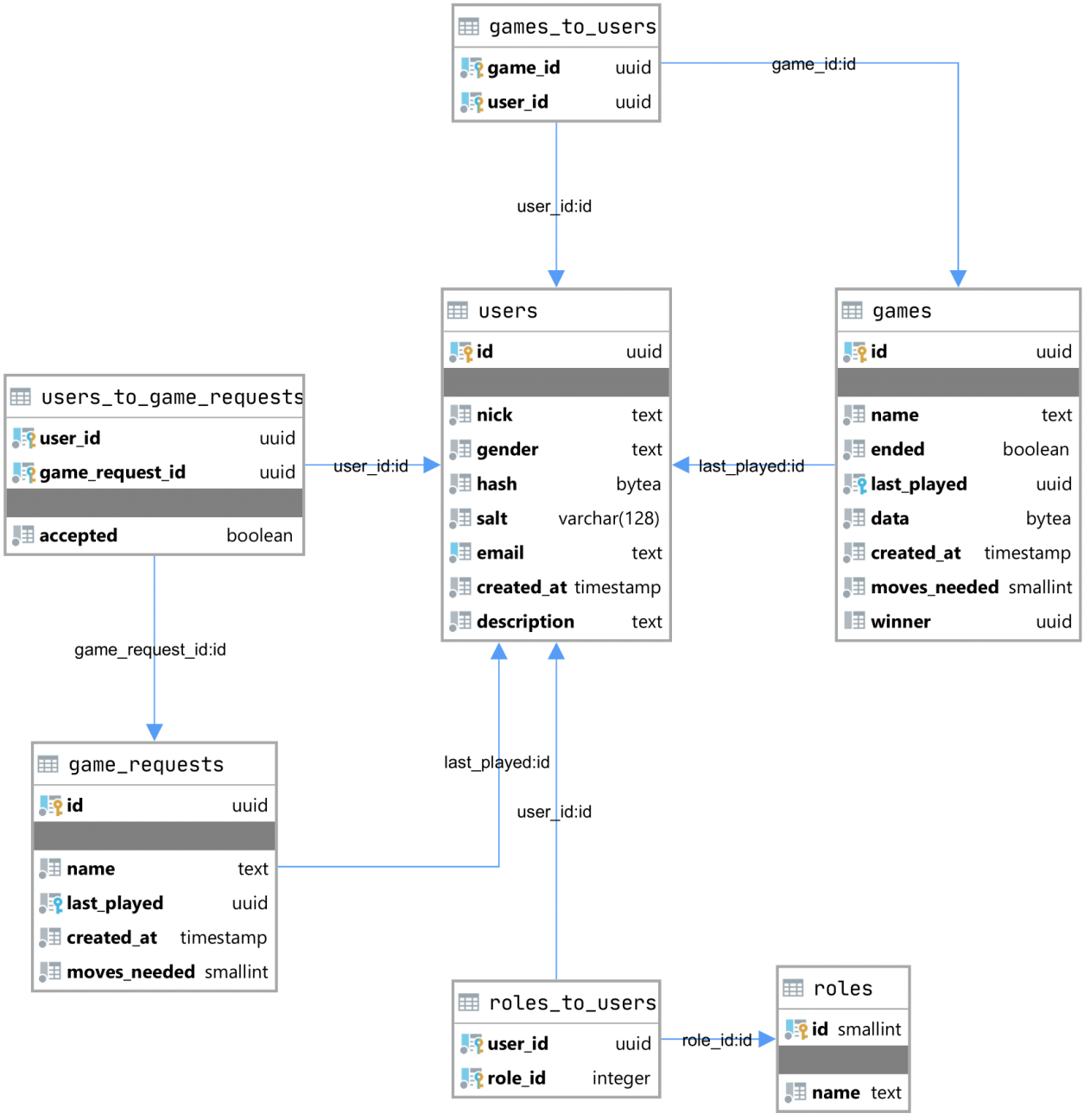
8 Seznam obrázků, tabulek, příloh

Seznam obrázků

Seznam tabulek

Seznam výpisů

9 Přílohy



Obrázek 4: ER Diagram

Výpis 8 uikit_addition.css

```
body {
    background-color: #545454;
}
.uk-navbar-container.uk-light:not(.uk-navbar-transparent)
 ↪  :not(.uk-navbar-primary) {
    background: #222;
}
.uk-dropdown.uk-light {
    background: #222;
}
.uk-dropdown li {
    padding-left: 5px;
    border-left: 2px solid transparent;
}
.uk-dropdown li.uk-active {
    border-color: #545454;
}
#mobile-navbar li {
    padding-left: 5px;
    border-left: 2px solid transparent;
}
#mobile-navbar li.uk-active {
    border-color: #545454;
}
.uk-navbar a {
    text-decoration: none
}
.uk-form-danger {
    color: #f0506e !important;
    border-color: #f0506e !important;
}
.uk-notification-message {
    background: #222;
}
body > div {
    padding-bottom: 1px;
}
```

Výpis 9 Procedura pro vytvoření pozvánky

```
create procedure new_game_request(_name text, _last_played uuid, _users_id
    uuid[], _moves_needed smallint)
    language plpgsql
as
$$
declare
    v_game_request_id uuid;
begin
    insert into game_requests (name, last_played, moves_needed)
    values (_name, _last_played, _moves_needed)
    returning game_requests.id into v_game_request_id;

    insert into users_to_game_requests (user_id, game_request_id)
    select user_id__, v_game_request_id
    from unnest(_users_id) user_id__;
    commit;
end;
$$;
```

Výpis 10 Procedura pro úpravu uživatele

```
create procedure update_user(_id uuid, _nick text DEFAULT NULL::text,
    _gender text DEFAULT NULL::text, _email text DEFAULT NULL::text, _hash
    bytea DEFAULT NULL::bytea, _salt character varying DEFAULT
    NULL::character varying, _roles integer[] DEFAULT NULL::integer[],
    _description text DEFAULT NULL::text)
    language plpgsql
as
$$
begin
    if _nick is not null then
        update users set nick = $2 where id = $1;
    end if;

    if _gender is not null then
        update users set gender = $3 where id = $1;
    end if;

    if _email is not null then
        update users set email = $4 where id = $1;
    end if;

    if _hash is not null then
        update users set hash = $5 where id = $1;
    end if;

    if _salt is not null then
        update users set salt = $6 where id = $1;
    end if;

    if _description is not null then
        update users set description = _description where id = _id;
    end if;

    if _roles is not null then
        delete from roles_to_users where user_id = _id;
        insert into roles_to_users (user_id, role_id) select _id, role_id__
    FROM unnest(_roles) role_id__;
    end if;

    commit;
end;
$$;
```

Výpis 11 Procedura pro úpravu pozvánky

```
create procedure update_invite (_user_id uuid, _game_request_id uuid,
    _accepted boolean, _data bytea)
    language plpgsql
as
$$
declare
    v_ready   bool;
    v_game_id uuid;
    v_exists  bool;
begin
    select exists(select * from users_to_game_requests where game_request_id
    = _game_request_id and user_id = _user_id)
    into v_exists;
    if not v_exists then
        raise exception 'User with id ''%'' is not part of game request with
    id ''%'' or game request with id ''%'' doesn''t exists', _user_id,
    _game_request_id, _game_request_id;
    end if;
    if _accepted then
        update users_to_game_requests set accepted = true where user_id =
    _user_id and game_request_id = _game_request_id;
        select not exists(
            select * from users_to_game_requests where game_request_id =
    _game_request_id and not accepted
        ) into v_ready;
        if v_ready then
            insert into games (name, data, last_played, moves_needed) select
    name, _data, last_played, moves_needed from game_requests where
    game_requests.id = _game_request_id returning games.id into v_game_id;
            insert into games_to_users (user_id, game_id) select
    users_to_game_requests.user_id, v_game_id from users_to_game_requests
    where game_request_id = _game_request_id;
            delete from users_to_game_requests where game_request_id =
    _game_request_id;
            delete from game_requests where id = _game_request_id;
        end if;
    else
        delete from users_to_game_requests where game_request_id =
    _game_request_id;
        delete from game_requests where id = _game_request_id;
    end if;
    commit;
end;
$$;
```

Výpis 12 Zdrojový kód knihovny `roles`

```rust
use postgres::{Client, NoTls};
use proc_macro::TokenStream;
use proc_macro2::Span;
use quote::quote;
use syn::punctuated::Punctuated;
use syn::{parse_macro_input, parse_quote, Ident, ItemEnum};

#[proc_macro_attribute]
pub fn get_roles_from_db(_attr: TokenStream, item: TokenStream) ->
    TokenStream {
    dotenv::dotenv().expect("Dotenv error");
    let mut parsed_enum = parse_macro_input!(item as ItemEnum);

    let database_url = std::env::var("DATABASE_URL").expect("Missing env
        variable DATABASE_URL");

    let mut client = Client::connect(&database_url, NoTls).expect("Couldn't
        create pool");

    let mut variants = Punctuated::new();
    for row in client
        .query("select name, id from roles", &[])
        .expect("Couldn't get roles from db")
    {
        let name = Ident::new(row.get(0), Span::call_site());
        let id: i16 = row.get(1);
        let variant: syn::Variant = parse_quote! {
            #name = #id as i32
        };
        variants.push(variant);
    }

    parsed_enum.variants = variants;

    (quote! {
        #parsed_enum
    })
    .into()
}
```