

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 použitych obrázků a neporušil jsem autorská práva.*

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

*V Pardubicích dne ..... ....*

*podpis*



## ZADÁNÍ MATURITNÍ PRÁCE

### Maturitní zkouška – profilová část – Maturitní projekt

<b>Obor:</b>	18-20-M/01 Informační technologie	<b>Školní rok:</b>	2020/2021
<b>Jméno a příjmení žáka:</b>	Jan Najman	<b>Třída:</b>	4.D
<b>Téma maturitní práce:</b> Piškvorky			
<b>Vedoucí maturitní práce:</b> RNDr. Jana Reslová			
<b>Pracoviště vedoucího:</b> 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**

<b>Úvod</b>	<b>10</b>
<b>1 Analýza obdobných aplikací</b>	<b>11</b>
1.1 turtlediary . . . . .	11
1.1.1 Kladné stránky . . . . .	11
1.1.2 Záporné stránky . . . . .	11
1.2 Ultimate Tic Tac Toe . . . . .	12
1.2.1 Kladné stránky . . . . .	12
1.2.2 Záporné stránky . . . . .	12
1.3 gametable . . . . .	13
1.3.1 Kladné stránky . . . . .	13
1.3.2 Záporné stránky . . . . .	13
<b>2 Návrh projektu</b>	<b>14</b>
2.1 Obecná struktura . . . . .	14
2.2 API . . . . .	14
2.3 Redis . . . . .	14
2.4 Databáze . . . . .	15
2.5 Front-end . . . . .	16
2.6 Administrace . . . . .	16
2.7 Design a responzivita . . . . .	16
<b>3 Zpracování praktické části</b>	<b>17</b>
3.1 Použité technologie . . . . .	17
3.1.1 Rust . . . . .	17
3.1.2 SQL . . . . .	17
3.1.3 Knihovna roles . . . . .	17
3.1.4 Back-end . . . . .	18
3.1.5 Front-end . . . . .	19
3.1.5.1 Server . . . . .	19
3.2 Databáze . . . . .	20
3.3 Datové struktury . . . . .	21
3.3.1 Enum . . . . .	21
3.3.2 Struct . . . . .	22
3.3.3 Trait . . . . .	22

3.3.4 Type . . . . .	24
3.4 Back-end . . . . .	24
3.4.1 Správa uživatelů . . . . .	24
3.4.2 Vytváření žádostí o hru . . . . .	24
3.4.3 Vytvoření hratelné hry . . . . .	25
3.4.4 Hraní hry . . . . .	25
3.5 Front-end . . . . .	25
3.5.1 Přihlášení . . . . .	25
3.5.2 Profil . . . . .	25
3.5.3 Výpis uživatelů . . . . .	26
3.5.4 Výpis her . . . . .	26
3.5.5 Hraní hry . . . . .	26
3.5.6 Výpis pozvánek . . . . .	26
3.5.7 Vytváření pozvánky . . . . .	26
3.5.8 Úprava uživatele . . . . .	26
3.5.9 Registrace . . . . .	27
<b>4 Manuál pro spuštění aplikace</b>	<b>28</b>
4.1 Instalace potřebných nástrojů . . . . .	28
4.1.1 Kompilace . . . . .	28
4.1.2 Použití kontejnerů . . . . .	28
4.2 Příprava . . . . .	29
4.2.1 Kompilace . . . . .	29
4.2.2 Použití kontejnerů . . . . .	30
4.3 Spuštění . . . . .	32
4.3.1 Kompilace . . . . .	32
4.3.2 Použití kontejnerů . . . . .	32
<b>5 Manuál pro používání aplikace</b>	<b>33</b>
5.1 Přihlášení . . . . .	33
5.1.1 První přihlášení . . . . .	34
5.2 Registrace . . . . .	35
5.3 Profil uživatele . . . . .	36
5.4 Seznam uživatelů . . . . .	37
5.4.1 Administrátori . . . . .	38
5.5 Úprava profilu . . . . .	39
5.5.1 Administrátori . . . . .	40

5.6 Seznam her . . . . .	40
5.7 Seznam pozvánek . . . . .	41
5.8 Vytvoření pozvánky . . . . .	42
5.8.1 Administrátoři . . . . .	43
5.9 Hraní hry . . . . .	43
<b>6 Závěr</b>	<b>45</b>
6.1 Řešení problémů . . . . .	46
6.1.1 Databáze . . . . .	46
6.1.2 Vyšší nápor na síť . . . . .	46
6.1.3 Cross-origin resource sharing . . . . .	47
6.2 Vylepšení . . . . .	47
<b>7 Seznam použité literatury a zdrojů informací</b>	<b>48</b>
<b>8 Seznam použitých zkratek</b>	<b>50</b>
<b>9 Seznam obrázků, tabulek, příloh</b>	<b>51</b>
<b>10 Přílohy</b>	<b>53</b>

# Úvod

Poslední dobou se všichni pokouší optimalizovat své web servery již při jejich programování, kvůli náporu, který by nemusely stíhat. Kdo tak neproveďe 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 komplikovaným programovacím jazykům jako jsou Rust, C++, C# je jejich rychlosť a možnost zjistit chybu při komplikaci.

Tabulka 1: Porovnání rychlosti jazyků [1]

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 než ostatní aplikace.

## 1.1 turtlediary

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

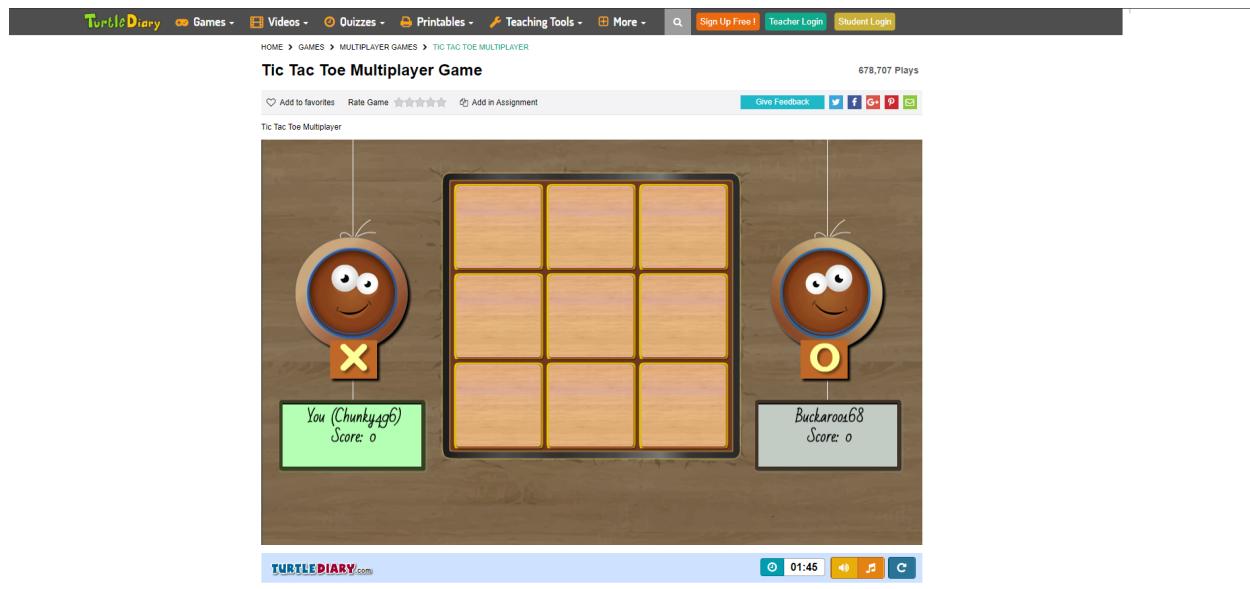
Web nabízí možnost hrát piškvorky 3x3 s náhodnými lidmi, nebo s kamarádem.

### 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>

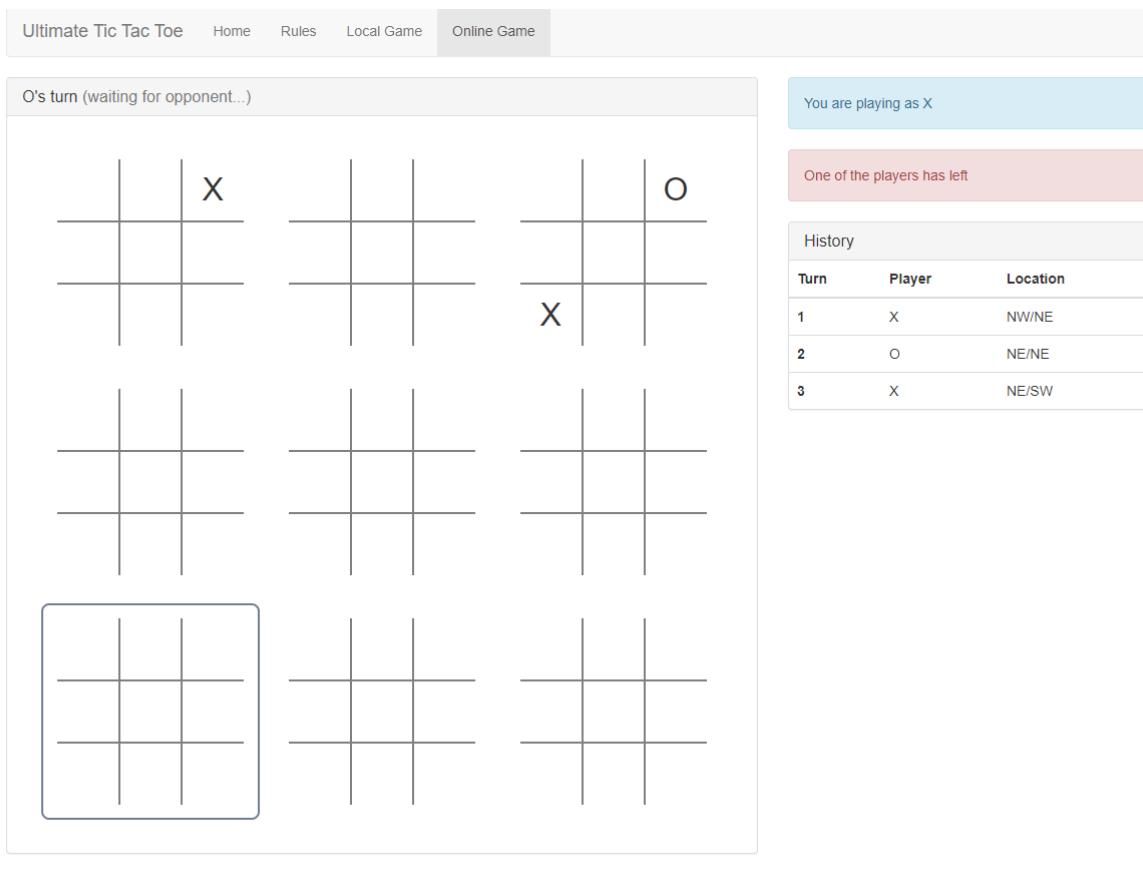
Web nabízí možnost hrát piškvorky 3x3 na více polích s kamarádem přes internet nebo lokálně.

### 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>

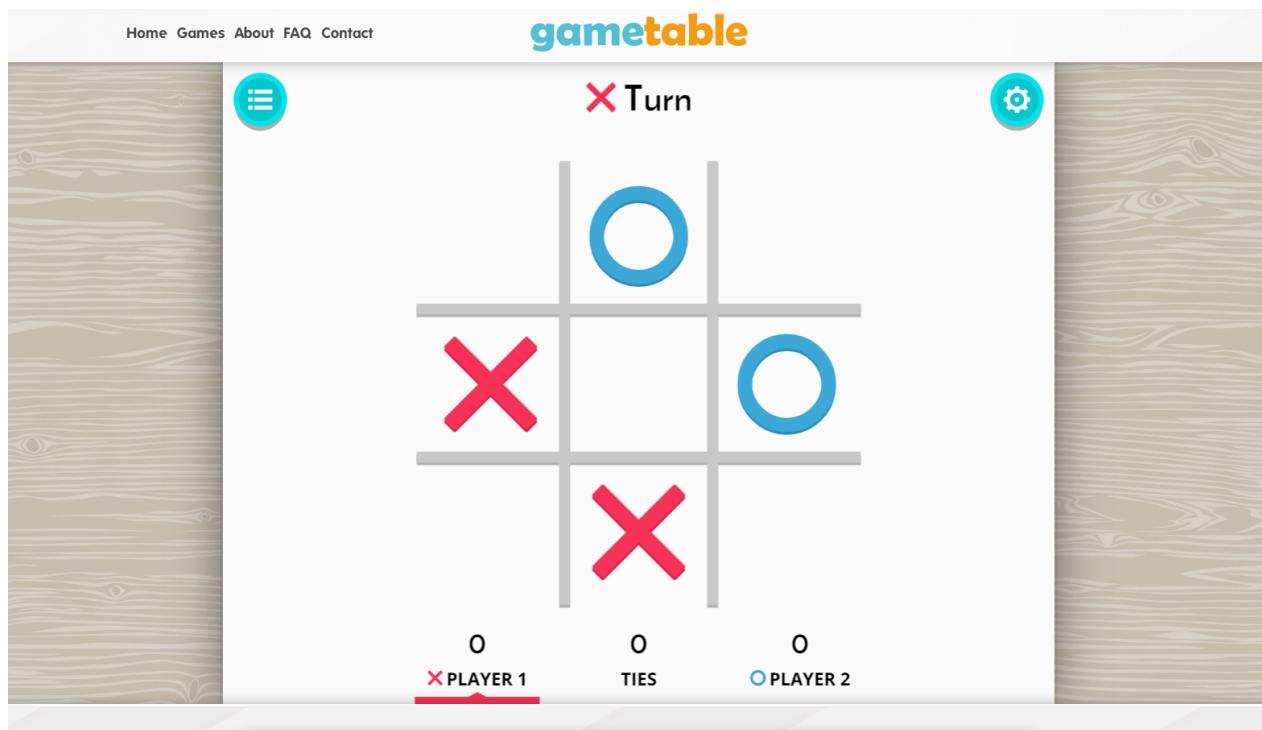
Web nabízí možnost hrát piškvorky 3x3 s kamarádem lokálně nebo proti AI.

### 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. 18

## 2.5 Front-end

Slouží jako grafické zobrazení dat a druhohradá 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 [2]. Několik věcí je přepsáno pro vzhled aplikace. Tyto změny se nachází v souboru *uikit\_addition.css*.

*uikit\_addition.css* lst. 13

## 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 Rust

Rust je víceúčelový, kompilovaný programovací jazyk, vyvinutý organizací Mozilla Research. Je navržen jako bezpečný a paralelní programovací jazyk. Podporuje funkcionální, imperativně-procedurální, strukturované a objektově orientované programování.

Vývoj je sponzorován Mozillou, ale jde o open source projekt. Velké množství příspěvků pochází od členů komunity.

#### 3.1.2 SQL

SQL je jazyk specificky určený ke správě dat uchovávaných v systému správy relačních databází. Je obzvláště užitečný při zpracování strukturovaných dat. Dat zahrnujících vztahy mezi subjekty a proměnnými.

Umožňuje přístupu k mnoha záznamům pomocí jediného příkazu. Vyučuje potřebu specifikovat, jak dosáhnout záznamu, např. s indexem nebo bez něj.

#### 3.1.3 Knihovna roles

Mnou vytvořená knihovna, která při kompliaci 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 načítání tokenů zdrojového kódu jazyka Rust do datového typu zdrojového kódu jazyka Rust
- *proc-macro2* - umožňuje definovat makra
- *dotenv* - načte .env soubor do proměnných prostředí
- *postgres* - synchronní klient pro databázi PostgreSQL

### 3.1.4 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 ke 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 z 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 kompliaci načte role z databáze a převede je do datového typu enum
- *actix-cors* - implementace CORS (Cross-origin resource sharing) pravidel pro Actix Web

### 3.1.5 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 komplikaci 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

#### 3.1.5.1 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. [14](#)

Procedura pro úpravu uživatele lst. [15](#)

Procedura pro úpravu pozvánky lst. [16](#)

## 3.3 Datové struktury

### 3.3.1 Enum

Datový typ enum v Rustu je podobný jako v jiných komplilovaných jazycích, jako je C, ale má důležité rozdíly, díky nimž je podstatně výkonnější. To, co Rust nazývá enumy, je běžněji známé jako algebraické datové typy, pokud přicházíte z pozadí funkčního programování. Důležitým detailem je, že každá varianta enumu může mít k sobě další data.

---

#### Výpis 1 Rust - datový typ enum

---

```
1 enum JednoduchyEnum {
2     PrvniVarianta,
3     DruhaVarianta,
4     TretiVarianta,
5 }
6
7 enum Lokace {
8     Neznama,
9     Anonymni,
10    Znama(Mesto),
11 }
12
13 enum KomplexniEnum {
14     Nic,
15     Neco(u32),
16     HodneVeci {
17         publikovat: bool,
18         text: String,
19     }
20 }
```

---

První enum je obvyklý druh enumu, který najdete například v jazyce C.

Druhý ukazuje hypotetický příklad něčeho, co ukládá údaje o poloze, přičemž *Mesto* je jakýkoli jiný typ, který je potřeba, například struct.

Třetí příklad ukazuje druh dat, které může varianta ukládat, od ničeho, přes tuple, až po anonymní strukturu.

### 3.3.2 Struct

Datový typ `struct` v Rustu můžeme nalézt ve třech druzích. Regulární s jmennými vlastnostmi, takzvané tuple struktury a unit struktury.

---

#### Výpis 2 Rust - datový typ struct

---

```
1 struct Regulární {  
2     vlastnost1: f32,  
3     vlastnost2: String,  
4     pub vlastnost3: bool  
5 }  
6  
7 struct Tuple(u32, String);  
8  
9 struct Unit;
```

---

Regulární struktury jsou nejvíce používané. Každá vlastnost je definována jménem a typem, poté se k daným vlastnostem dá dostat pomocí `struktura.vlastnost`.

Všechny vlastnosti struktury sdílejí stejnou proměnlivost jako samotná struktura, takže `struktura.vlastnost = 2`; by bylo validní, jen když by celá struktura byla proměnlivá. Přidáním `pub` před jméno vlastnosti způsobí, že daná vlastnost je viditelná i v jiných modulech a je možno jí přímo číst nebo měnit.

Tuple struktury jsou podobné to regulárním strukturám, ale jejich vlastnosti nemají žádná jména. Pro přístup k vlastnostem se použije pozice vlastnosti, začínající od nuly `struktura.0`, `struktura.1` atd.

Unit struktury jsou hlavně používány jako markery. Nemají žádnou velikost, ale můžou být instancovány, to z nich dělá typ stejný jako `unit` – `()`. Jsou užitečné, když potřebujete implementovat trait na něčem, ale nepotřebujete ukládat žádná data.

### 3.3.3 Trait

Trait je jako interface, který datové typy mohou implementovat.

Traity můžou být složeny až ze tří druhů položek:

- funkce a metody
- typy
- konstanty

Traity také mohou sloužit jako markery, nebo mohou nést jiné logiku, která není vyjádřená v jejich názvech.

---

### Výpis 3 Rust - typ trait

---

```
1 struct Pes {
2     vaha_kg: u8,
3 }
4 struct Kocka {
5     jmeno: String,
6     vaha_g: u16,
7 }
8 trait Vaha {
9     fn vaha_v_kg(&self) -> f32;
10 }
11 impl Vaha for Pes {
12     fn vaha_v_kg(&self) -> f32 {
13         self.vaha_kg as f32
14     }
15 }
16 impl Vaha for Kocka {
17     fn vaha_v_kg(&self) -> f32 {
18         self.vaha_g as f32 / 1000f32
19     }
20 }
21 fn main() {
22     let kocka = Kocka {
23         jmeno: "Líza".into(),
24         vaha_g: 2000,
25     };
26     let pes = Pes {
27         vaha_kg: 2,
28     };
29     assert_eq!(kocka.vaha_v_kg(), pes.vaha_v_kg());
30 }
```

---

### 3.3.4 Type

Definuje alias pro existující datový typ.

---

#### Výpis 4 Rust - type

---

```
1 type Milimetr = u32;
2 type Kilogram = u32;
3
4 let m: Milimetr = 3;
5 let k: Kilogram = 3;
6
7 assert_eq!(m, k);
```

---

Nevytváří nové datové typy, proto se ve výše uvedeném příkladu tři milimetry rovnají třem kilogramům.

## 3.4 Back-end

### 3.4.1 Správa uživatelů

Uživatelé jsou umístěni v tabulce *users*. Role jsou uloženy v tabulce *roles* a jsou k uživatelům přiřazovány skrz tabulkou *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. 15.

### 3.4.2 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 tabulkou *users\_to\_game\_requests*.

Pozvánky nemohou vytvářet uživatelé s rolí *Banned*.

Vytvoření pozvánky po kontrole dat je poté řízeno procedurou lst. 14

### **3.4.3 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 tabulkou *games\_to\_users* a žádost o hru je poté vymazána.

Úpravu pozvánky po kontrole dat je poté řízena procedurou lst. 16

### **3.4.4 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 zkонтroluje.

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.

## **3.5 Front-end**

### **3.5.1 Přihlášení**

Pole emailu je kontrolováno. Pokud pole není validní, tak se nepošlou data na back-end.

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

### **3.5.2 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.

### **3.5.3 Výpis uživatelů**

Zobrazuje všechny registrované uživatele a pář informací o nich.

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

### **3.5.4 Výpis her**

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

### **3.5.5 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.

### **3.5.6 Výpis pozvánek**

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

Uživatel může pozvánku přijmout, nebo odmítnout.

### **3.5.7 Vytváření pozvánky**

Při vytváření pozvánky jsou skoro všechna pole kontrolována.

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

### **3.5.8 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.

### **3.5.9 Registrace**

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

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

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

## 4 Manuál pro spuštění aplikace

Aplikace se dá spustit více způsoby. Bud' kompliací ze zdrojového kódu, nebo pomocí kontejnerů.

### 4.1 Instalace potřebných nástrojů

#### 4.1.1 Kompilace

Pro komplaci potřebujeme nainstalovat komplátor 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 nainstaluji ještě wasm-pack pro komplaci do WebAssembly.

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

Po stažení jej nainstaluji.

#### 4.1.2 Použití kontejnerů

Pro použití kontejnerů potřebujeme nějaký software, který to umožňuje spouštění kontejnerů a manipulaci s nimi. Já jsem zvolil docker, jeden z nejznámějších nástrojů pro používání a výrobu kontejnerů.

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

Nainstaluji a vyzkouším funkčnost (Hello world! kontejner).

## 4.2 Příprava

### 4.2.1 Kompilace

Kompilace není potřeba při použití kontejnerů.

Pro komplikaci 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 5 Příkaz pro komplikaci back-endu

---

```
cargo build --release
```

---

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

---

#### Výpis 6 Příkaz pro komplikaci front-end serveru

---

```
cargo build --release
```

---

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

---

#### Výpis 7 Příkaz pro komplikaci 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 8** Nastavení proměnných prostředí pro kontejnery

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

---

### Výpis 9 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 položky, pro přeložení adresy front-endu a back-endu:

---

### Výpis 10 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.

Pro jednoduché použití stačí vytvořit .env soubor, ze kterého se načtou hodnoty do proměnných prostředí. Program si je potom bere odtud.

### 4.3.2 Použití kontejnerů

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

---

**Výpis 11** Příkaz pro start kontejnerů

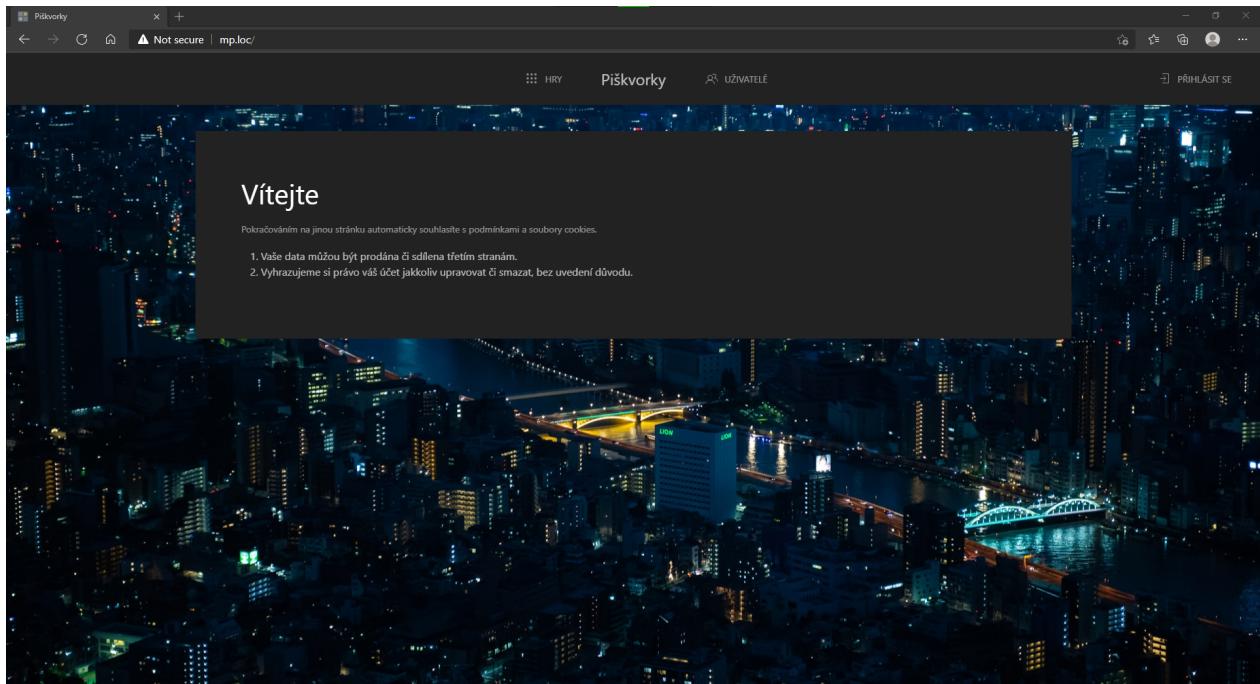
---

docker-compose up -d

---

## 5 Manuál pro používání aplikace

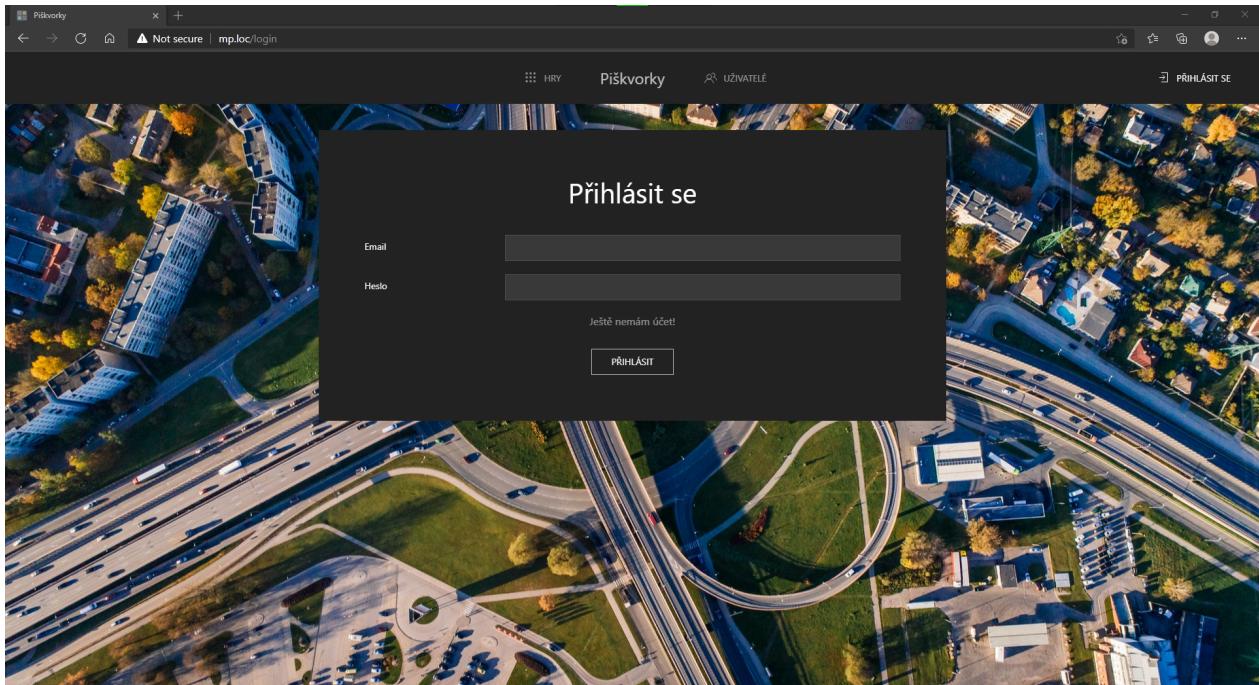
Po spuštění aplikace přejdeme na doménu front-endu.



Obrázek 4: Hlavní stránka aplikace

### 5.1 Přihlášení

Vpravo nahoře klikneme na PŘIHLÁSIT SE a dostaneme se na stránku přihlášení.



Obrázek 5: Stránka přihlášení

Vyplníme údaje a klikneme na PŘIHLÁSIT.

### 5.1.1 První přihlášení

Pro první přihlášení se přihlásíme na administrátora s těmito údaji:

---

#### **Výpis 12** Údaje administrátora při prvním přihlášení

---

**EMAIL:** r@r.r

**HESLO:** root

---

The screenshot shows a dark-themed login form titled "Přihlásit se". It has two input fields: "Email" containing "r@r.r" and "Heslo" containing "root". Below the fields is a link "Ještě nemám účet!". A "PŘIHLÁSIT" button is at the bottom. The background features a colorful abstract pattern.

Obrázek 6: Stránka přihlášení - údaje administrátora

Po přihlášení změňte heslo a ostatní údaje, pokud tak neprováděte může dojít k odcizení administrátorského účtu!

## 5.2 Registrace

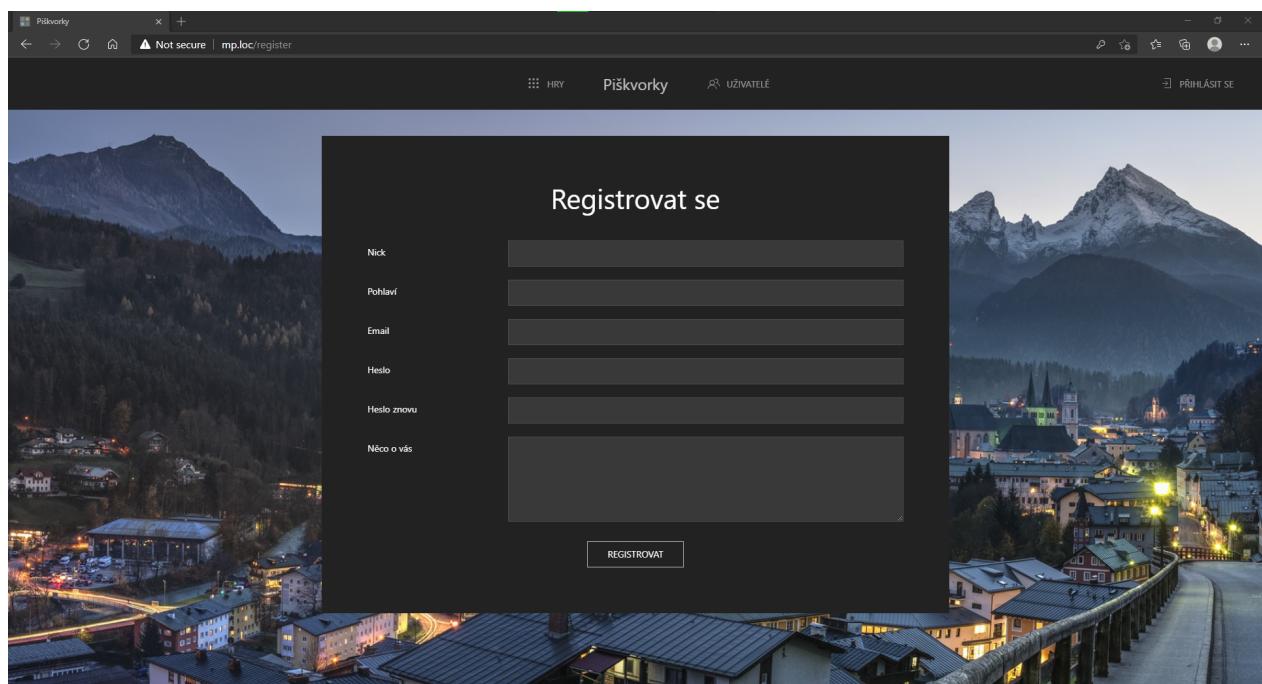
Vpravo nahoře klikneme na PŘIHLÁSIT SE a dostaneme se na stránku přihlášení, poté klikneme na Ještě nemám účet!

Vyplníme veškeré údaje.

Údaje jsou kontrolovány a musí odpovídat těmto parametrům:

- *Nick* - musí obsahovat 3 až 12 znaků, povolené znaky jsou:
  - malá písmena od a až po z
  - velká písmena od A až po Z
  - číslice od 0 až do 9
- *Pohlaví* - musí obsahovat 1 až 50 znaků, povolené znaky jsou:
  - malá písmena od a až po z včetně diakritiky
  - velká písmena od A až po Z včetně diakritiky
  - číslice od 0 až do 9
  - speciální znaky & \_ - ' , ., mezi tyto znaky patří také mezera
- *Email* - musí obsahovat 5 až 25 znaků, musí být podle standardu RFC2822

- *Heslo* - musí obsahovat 8 až 25 znaků, musí obsahovat:
  - 2 malá písmena
  - 2 velká písmena
  - 2 speciální znaky
- *Heslo znova* - musí mít stejnou hodnotu jako pole *Heslo*
- *Něco o vás* - musí obsahovat 20 až 650 znaků, povolené znaky jsou:
  - malá písmena od a až po z včetně diakritiky
  - velká písmena od A až po Z včetně diakritiky
  - číslice od 0 až do 9
  - speciální znaky & \_ ' , ., mezi tyto znaky patří také mezera

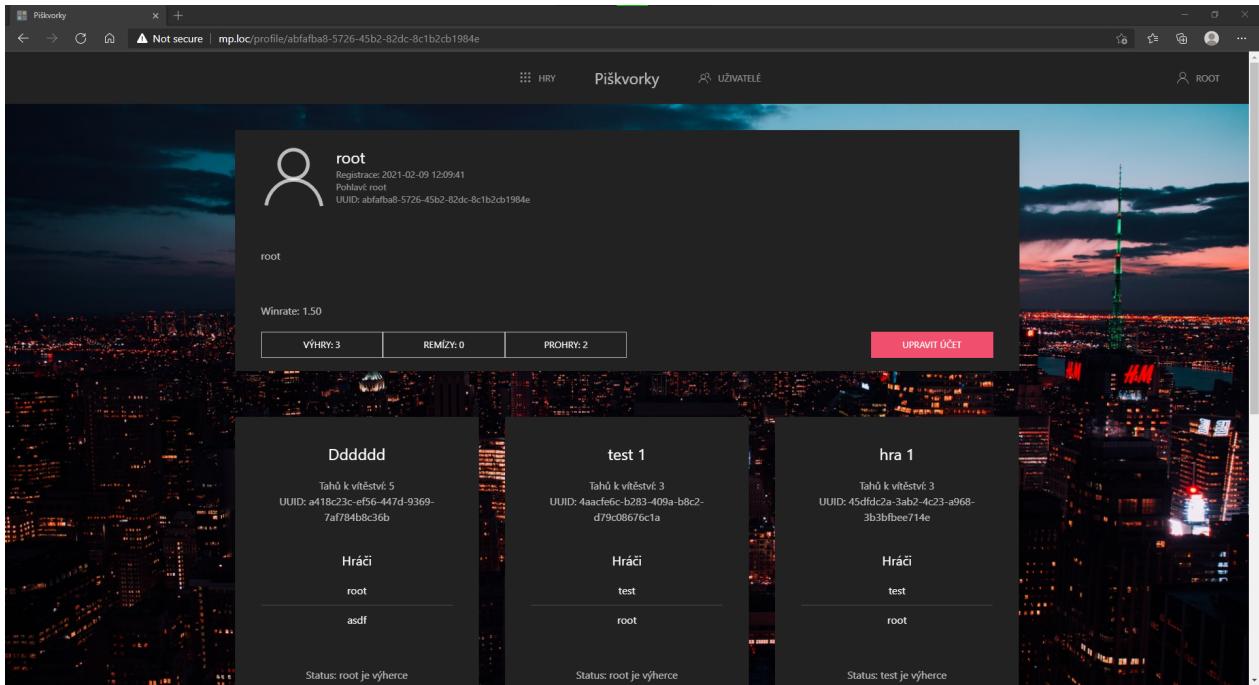


Obrázek 7: Stránka registrace

### 5.3 Profil uživatele

Na vlastní profil se můžeme dostat dvěma způsoby:

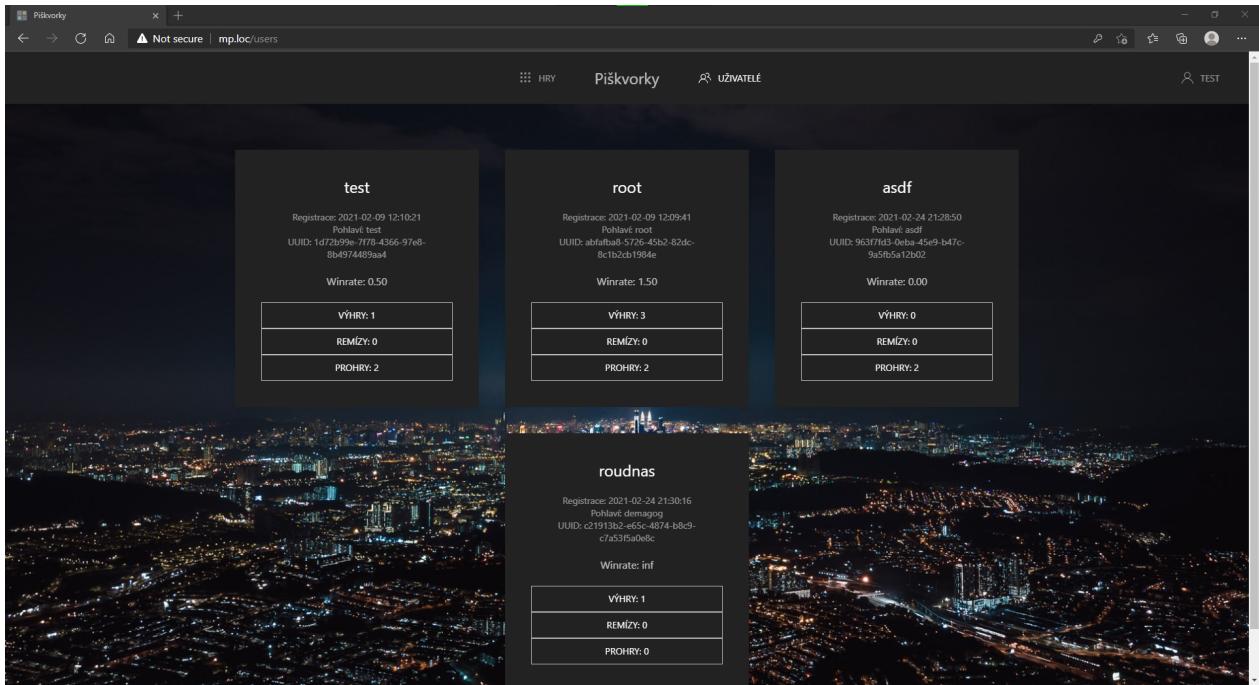
1. Vpravo nahoře najedeme myší na své jméno, rozbalí se nám nabídka a klikneme na **Profil**.
2. Nahoře uprostřed klikneme na **UŽIVATELÉ**, najdeme se v seznamu uživatelů a klikneme na své jméno.



Obrázek 8: Stránka profilu

## 5.4 Seznam uživatelů

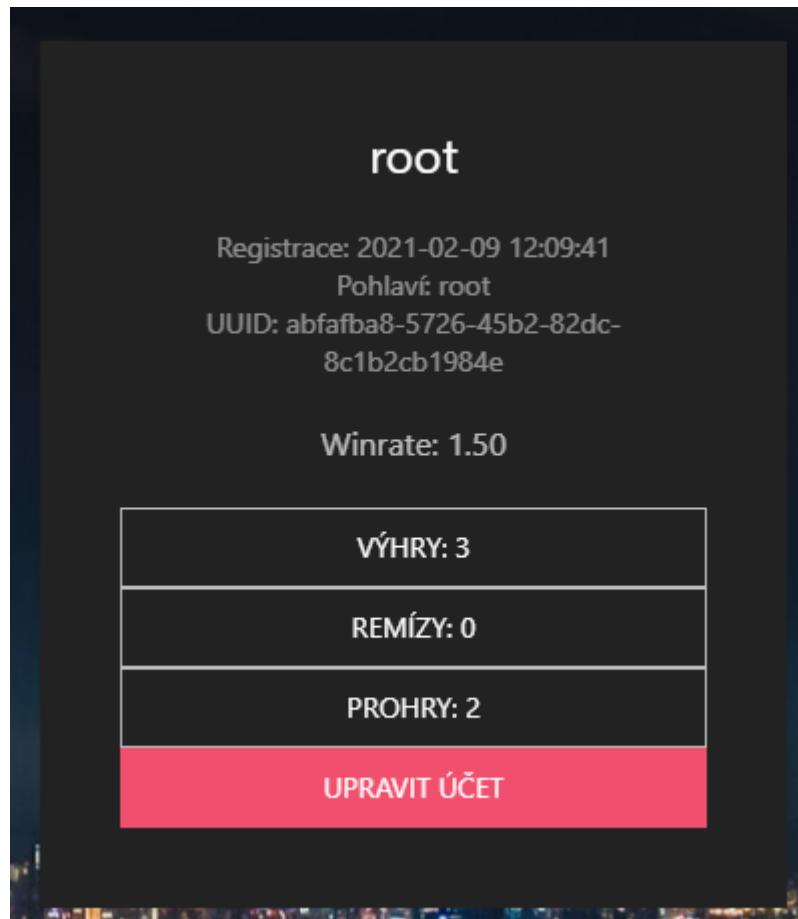
Na seznam uživatelů se dostaneme tak, že nahoře uprostřed klikneme na UŽIVATELÉ.



Obrázek 9: Stránka seznamu uživatelů

#### 5.4.1 Administrátoři

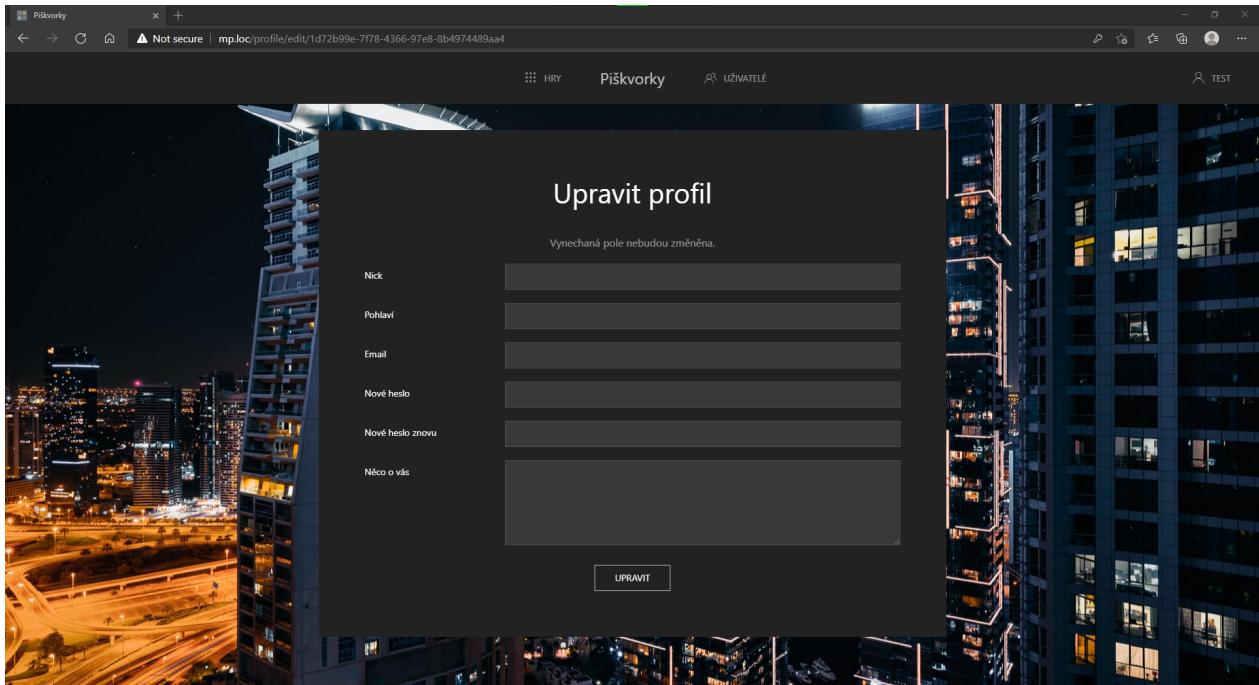
Administrátoři mají navíc vespoďu každé karty uživatele ještě tlačítko pro úpravu profilu daného uživatele.



Obrázek 10: Stránka seznamu uživatelů - možnosti administrátora

## 5.5 Úprava profilu

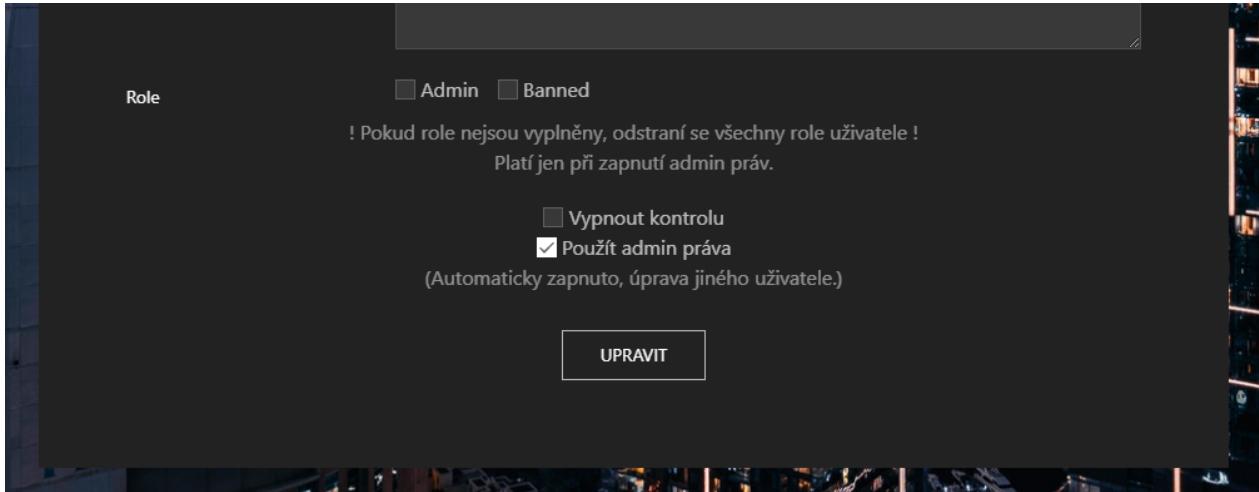
Na stránce svého profilu (sec. 5.3) klikneme na tlačítko UPRAVIT ÚČET.



Obrázek 11: Stránka úpravy profilu

### 5.5.1 Administrátoři

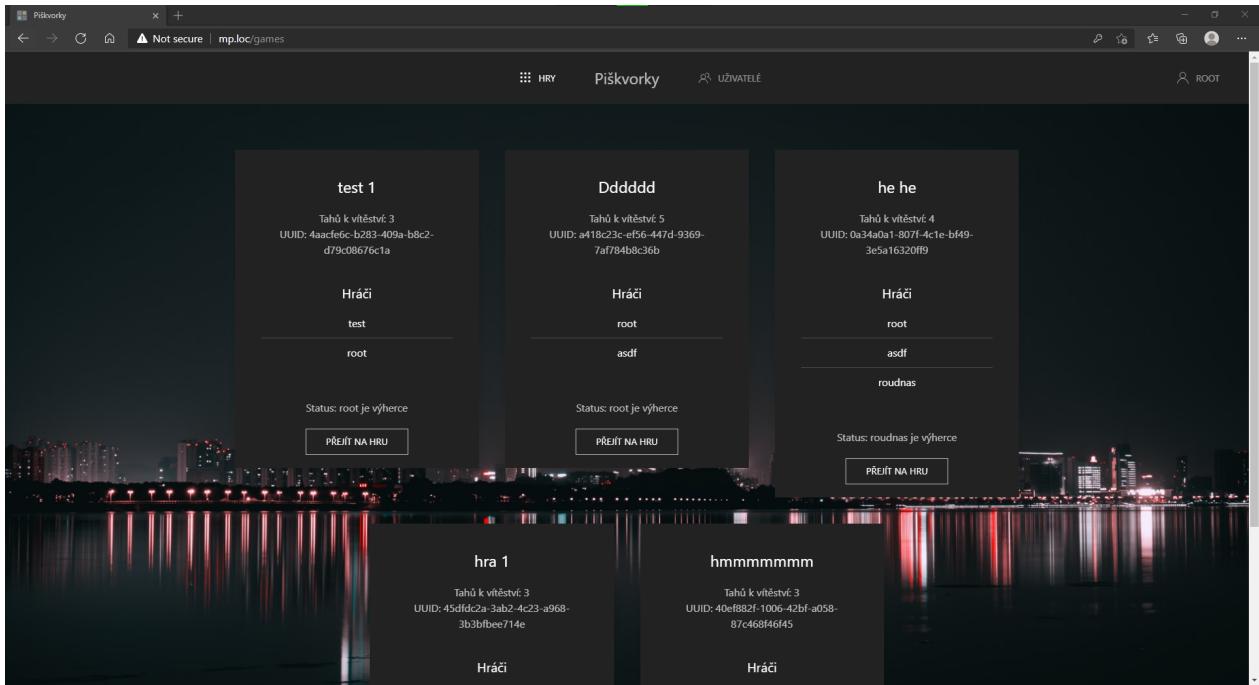
Administrátoři mají na konci stránky více možností úpravy.



Obrázek 12: Stránka úpravy profilu - možnosti administrátora

## 5.6 Seznam her

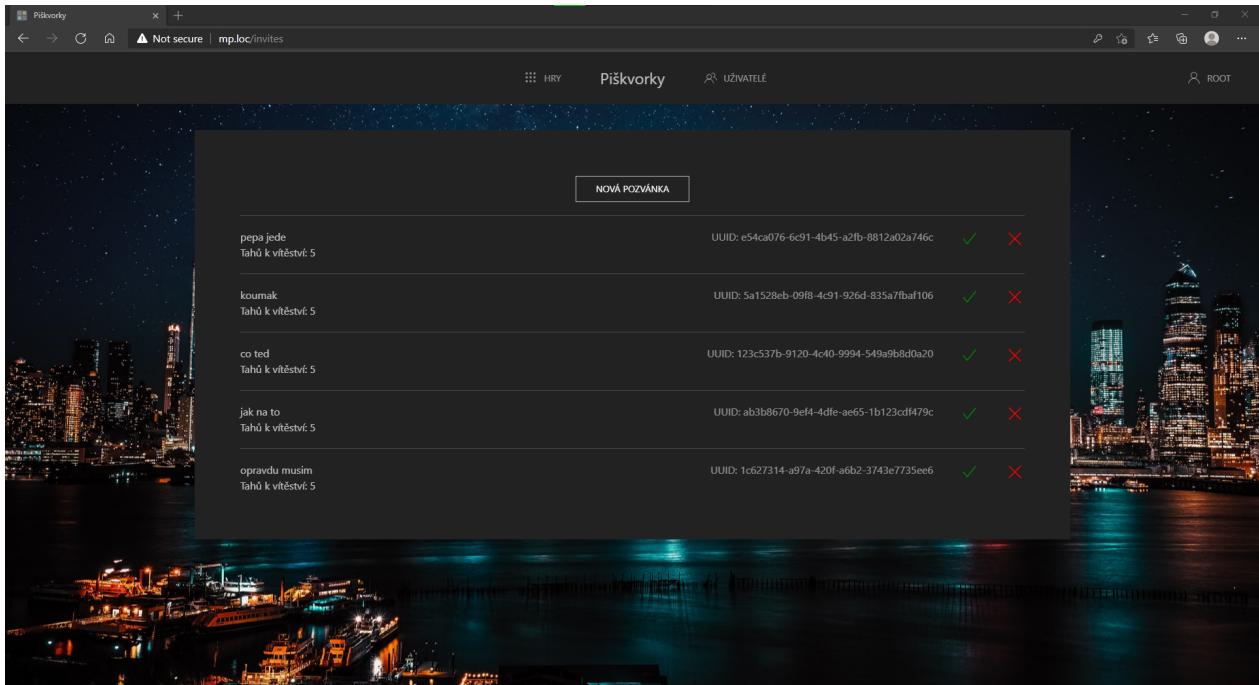
Na seznam her se dostaneme tak, že nahoře uprostřed klikneme na HRY.



Obrázek 13: Stránka seznamu her

## 5.7 Seznam pozvánek

Na seznam pozvánek se dostaneme tak, že vpravo nahoře najedeme myší na své jméno, rozbalí se nám nabídka a klikneme na Pozvánky.



Obrázek 14: Stránka seznamu pozvánek

Pro přijetí pozvánky klikněte na fajfku (✓) a pro zamítnutí na křížek (✗).

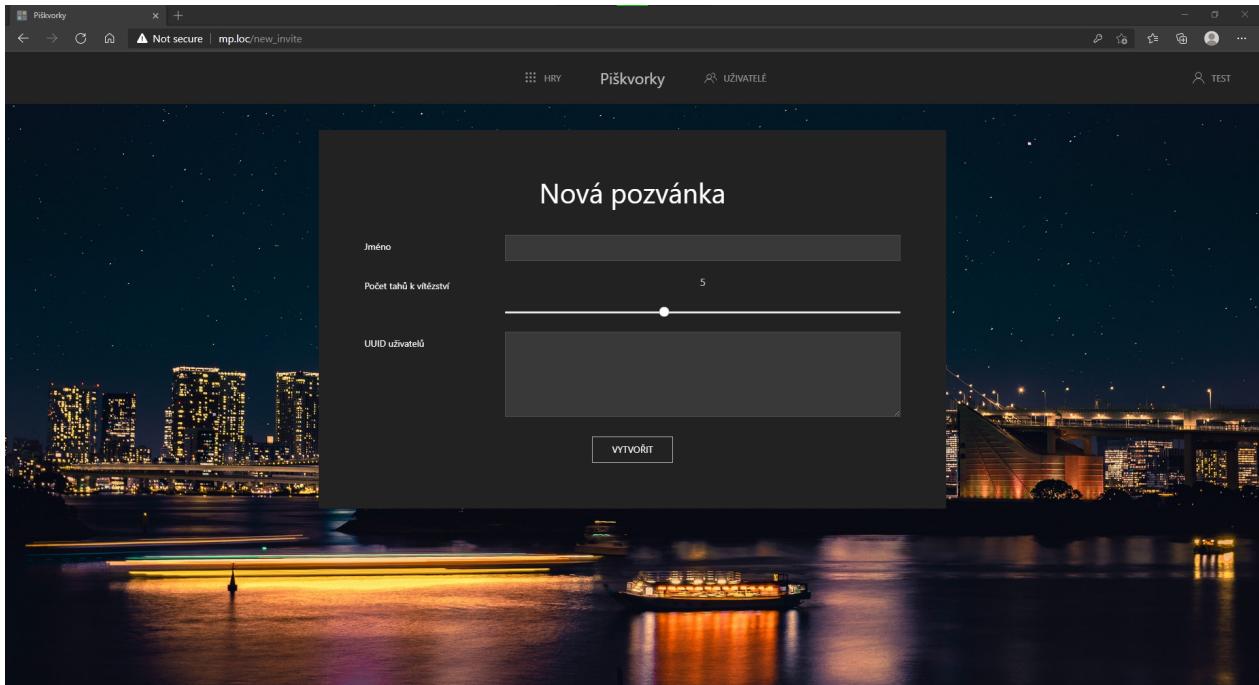
## 5.8 Vytvoření pozvánky

Na seznamu pozvánek (sec. 5.7) klikneme na tlačítko NOVÁ POZVÁNKA.

Vyplníme veškeré údaje.

Údaje jsou kontrolovány a musí odpovídat těmto parametrům:

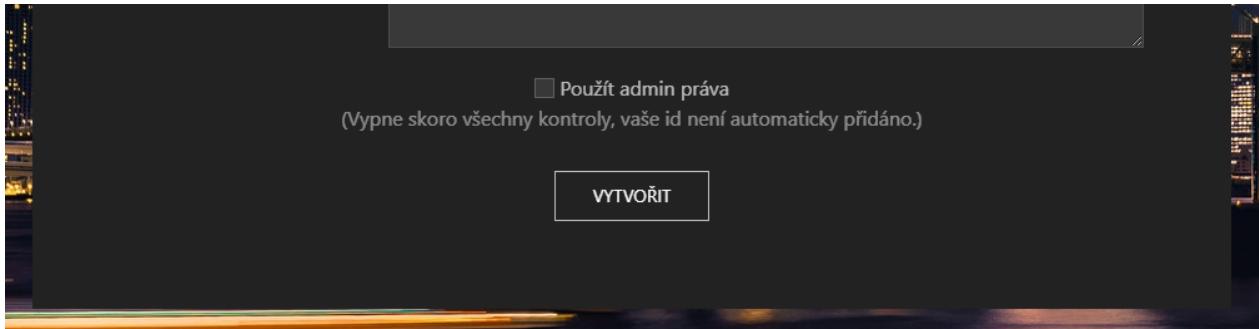
- *Jméno* - musí obsahovat 5 až 15 znaků, povolené znaky jsou:
  - malá písmena od a až po z
  - velká písmena od A až po Z
  - číslice od 0 až do 9
  - speciální znaky & \_ ' , ., mezi tyto znaky patří také mezera
- *Počet tahů k vítězství* - musí být číslice od 3 až do 8
- *UUID uživatelů* - tento údaj není kontrolovaný na front-endu, ale je kontrolovaný na back-endu, toto pole musí obsahovat:
  - UUID uživatelů (lze získat ze seznamu uživatelů) oddělená novými řádky



Obrázek 15: Stránka vytvoření pozvánky

### 5.8.1 Administrátoři

Administrátoři mají na konci stránky více možností úpravy.

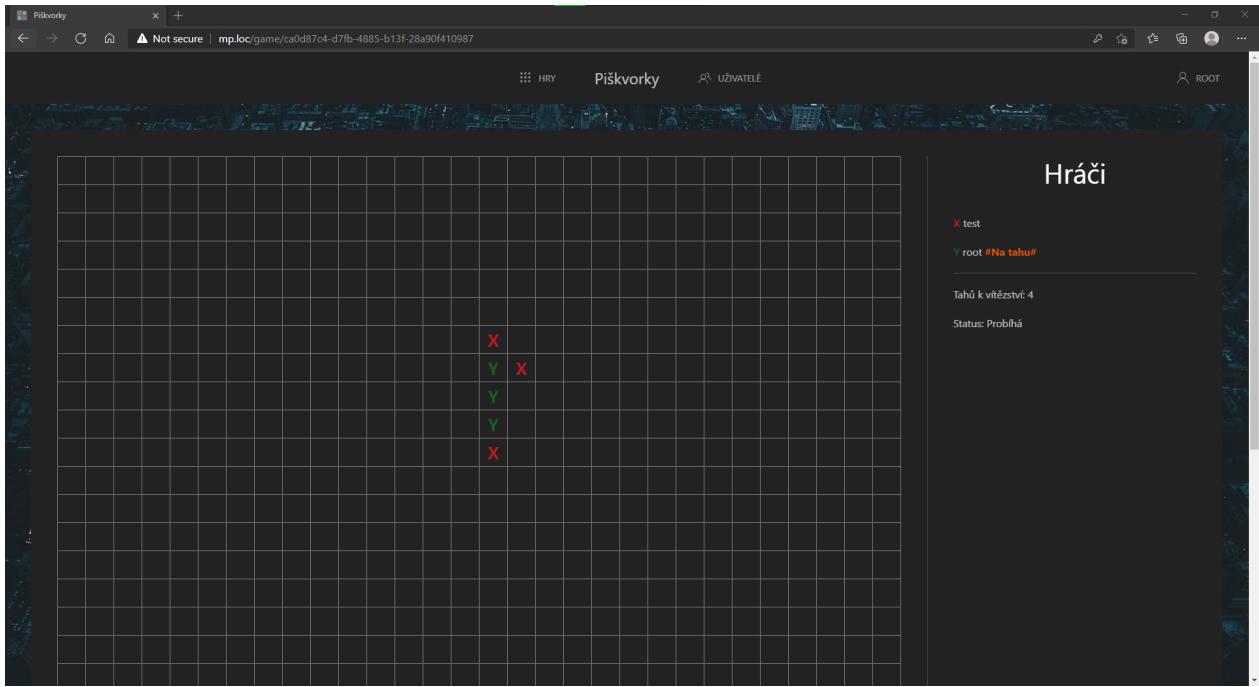


Obrázek 16: Stránka vytvoření pozvánky - možnosti administrátora

## 5.9 Hraní hry

Pro hraní hry musíte mít hru v probíhajícím stavu a být hráč dané hry. Pokud žádnou z takových her nemáte, můžete ji vytvořit (sec. 5.8).

Hru, kterou chcete hrát můžete najít na svém profilu (sec. 5.3). Potom co jí najdete klikněte na tlačítko PŘEJÍT NA HRU.



Obrázek 17: Stránka hraní hry

Před sebou bychom měli vidět mřížku a vpravo nebo pod ní seznam uživatelů. Před uživatelským jménem je jeho symbol a za jménem je napsáno, jestli je na tahu.

Pod seznamem uživatelů můžeme vidět, kolik tahů potřebujeme k vítězství a jaký je status hry.

## 6 Závěr

Tento projekt vznikl proto, aby ukázal, že frameworky v Rustu jsou dostatečně vyspělé a výkonnější něž alternativy. Tento fakt ukazuje i čím dál více rostoucí komunita a stránka “Are we web yet?” [3], která sleduje vývoj webových technologií v Rustu.

Jedním z mých úkolů, také bylo držet se open source a s radostí mohu prohlásit, že se mi to úspěšně povedlo.

Co se týče hlavních bodů maturitní práce, tak mohu bez starostí říci, že jsem je splnil do puntíku.

1. “Hry se budou ukládat do databáze” - hry jsou uloženy v databázi PostgreSQL v tabulce *games*
2. “Hry budou moct být hrány na tahy nebo odehrány v celku” - hry můžete odehrát v jedno sezení nebo ve více, nezáleží na tom, jak dlouho bude trvat 5 minut, nebo 5 let
3. “Rozdělení uživatelů na přihlášené a nepřihlášené a administrátory” - uživatelé jsou rozděleni na nepřihlášené a přihlášené a pomocí role *Admin* na přihlášené a administrátory
4. “Tahy hráčů budou kontrolovány na serveru” - tahy jsou kontrolovány sec. 3.4.4
5. “Data odesílaná klientem nebo uživatelem budou mít co nejmenší velikost” - skoro všechna data jsou posílány ve formát JSON, data herního pole jsou posílána v ještě menším formátu bincode
6. “Hry budou mít různé velikosti” - hry mohou být hrány na různý počet tahů
7. “Každý přihlášený uživatel bude mít statistiky” - každý uživatel má počet výher, počet remíz a počet proher, z těchto dat je poté vypočítán winrate

## 6.1 Řešení problémů

Při řešení tohoto tématu jsem narazil na pár problémů, které mě ale moc nevykolejily. Dalo by se říct, že jsem s nimi počítal.

### 6.1.1 Databáze

Jedním ze závažnějších problémů byla databáze. Nejprve jsem pracoval s MySQL databází, ale později se ukázalo, že má mnoho limitací a nevýhod.

Jednou z nich byla limitace datového typu string, neboli varchar (nebo char), který neměl dostatečnou délku. Později jsem taky zjistil, že datový typ varchar při různých operacích je převáděn na datový typ char a zbité bajty jsou vyplněny. Takže místo porovnávání textu s různou délkou, dochází k porovnávání textu se stejnou délkou (255 znaků). MySQL má datový typ, který umožňuje ukládat delší text, ale jak s ním pracuje mi vůbec nevyhovovalo.

Druhou z nevýhod byly procedury, které se v tomto jazyce nedají psát! Jakým způsobem se procedury píší nedává žádný smysl. Dokonce si dovoluji říct, že psát kód v assembly je jednodušší.

Z výše uvedených důvodů jsem přešel na open source databázi PostgreSQL. Tato databáze vyřešila všechny mé problémy, a více. Díky této databázi jsem byl schopný použít pro id tabulek datový typ UUID, který nevyčerpám, ani když se budu snažit a také používat json data nativně. Mohu používat datový typ text, bez jakéhokoli problému, či snížení výkonu. Mohu vracet výsledky z insertů či updatů. Ale jedna z nejdůležitějších vlastností byla pro mě možnost psát procedury, které podporují základní věci jako je cyklus (viz. lsts. 14, 15, 16).

### 6.1.2 Vyšší nápor na síť

Další problém, na který jsem narazil, byla komunikace mezi databází a back-endem. Díky tomu, že back-end potřeboval několik sql dotazů, pro správné vyhodnocení požadavku klienta, tak jsem implementoval většinu do procedury. Tímto způsobem se zmenšila zátěž jak na back-end, tak na databází a samozřejmě nápor na síť klesl.

### 6.1.3 Cross-origin resource sharing

Třešničkou na dortu samozřejmě byly CORS. Nejdříve jsem je neřešil a používal upravený prohlížeč, ale dříve či později jsem je musel řešit. Naštěstí na tento otravný problém autoři frameworku Actix Web mysleli a vydaly knihovnu `actix-cors`, v té stačí je nakonfigurovat cors pravidla, buď pro celou aplikaci, nebo pro každý koncový bod, nebo pro obojí.

## 6.2 Vylepšení

Jedině co bych vylepšil je zobrazování her (fig. 17) a přidal bych více symbolů pro hráče, tím bych navýšil maximální počet hráčů ve hře. Zobrazování her bych vyřešil, tak že bych přidal pořádný renderer, například přes WebGL. To by mi dovolilo využít plný prostor pro hraní (256x256 - limitován datovým typem u8), který je momentálně limitován na 30x30, kvůli současném zobrazování her.

## 7 Seznam použité literatury a zdrojů informací

- [1] ZAHARIEV, Ivan. *C++ vs. Python vs. PHP vs. Java vs. Others performance benchmark (2016 Q3)* [online]. 10. září 2016 [vid. 2020-11-05]. Dostupné z: <https://blog.famzah.net/2016/09/10/cpp-vs-python-vs-php-vs-java-vs-others-performance-benchmark-2016-q3/>
- [2] YOOTHEME. *UIkit documentation* [online]. 2021 [vid. 2020-09-30]. Dostupné z: <https://getuikit.com/docs/introduction>
- [3] THE RUST TEAM. *Are we web yet?* [online]. 21. únor 2021 [vid. 2020-09-30]. Dostupné z: <https://www.arewewebyet.org/>
- [4] THE POSTGRESQL GLOBAL DEVELOPMENT GROUP. *PostgreSQL 13.2 Documentation* [online]. 1996-2021 [vid. 2020-09-30]. Dostupné z: <https://www.postgresql.org/docs/13/index.html>
- [5] THE ACTIX TEAM. *Actix Web API Documentation* [online]. 2021 [vid. 2020-09-30]. Dostupné z: [https://docs.rs/actix-web/3.3.2/actix\\_web/](https://docs.rs/actix-web/3.3.2/actix_web/)
- [6] THE ACTIX TEAM. *Actix Redis API Documentation* [online]. 2021 [vid. 2020-09-30]. Dostupné z: [https://docs.rs/actix-redis/0.9.1/actix\\_redis/](https://docs.rs/actix-redis/0.9.1/actix_redis/)
- [7] THE ACTIX TEAM. *Actix Cors API Documentation* [online]. 2021 [vid. 2020-09-30]. Dostupné z: [https://docs.rs/actix-cors/0.5.4/actix\\_cors/](https://docs.rs/actix-cors/0.5.4/actix_cors/)
- [8] THE ACTIX TEAM. *Actix Session API Documentation* [online]. 2021 [vid. 2020-09-30]. Dostupné z: [https://docs.rs/actix-session/0.4.0/actix\\_session/](https://docs.rs/actix-session/0.4.0/actix_session/)
- [9] REDIS COMMUNITY. *Redis documentation* [online]. 2021 [vid. 2020-09-30]. Dostupné z: <https://redis.io/documentation>
- [10] THE YEW TEAM. *Yew documentation* [online]. 25. únor 2021 [vid. 2020-09-30]. Dostupné z: <https://yew.rs/docs/en/>
- [11] THE YEW TEAM. *Yew API Documentation* [online]. 2021 [vid. 2020-09-30]. Dostupné z: <https://docs.rs/yew/0.17.4/yew/>

- [12] THE YEW TEAM. *Yew Router API Documentation* [online]. 2021 [vid. 2020-09-30]. Dostupné z: [https://docs.rs/yew-router/0.14.0/yew\\_router/](https://docs.rs/yew-router/0.14.0/yew_router/)
- [13] THE RUST TEAM. *Rust std API Documentation* [online]. 2021 [vid. 2020-09-30]. Dostupné z: <https://doc.rust-lang.org/std/>
- [14] WASP-PACK TEAM. *WASM-PACK documentation* [online]. 2021 [vid. 2020-09-30]. Dostupné z: <https://rustwasm.github.io/docs/wasm-pack/>
- [15] DOCKER INC. *Docker documentation* [online]. 2013-2021 [vid. 2020-09-30]. Dostupné z: <https://docs.docker.com/>
- [16] MACFARLANE, John. *Pandoc documentation* [online]. 2006-2021 [vid. 2020-09-30]. Dostupné z: <https://pandoc.org/MANUAL.html>
- [17] YAKIMOV, Nikolay. *Pandoc-crossref documentation* [online]. 2016-2021 [vid. 2020-09-30]. Dostupné z: <https://lierdakil.github.io/pandoc-crossref/>
- [18] CORPORATION FOR DIGITAL SCHOLARSHIP. *Zotero Style Repository* [online]. 2021 [vid. 2021-02-01]. Dostupné z: <https://www.zotero.org/styles>

## 8 Seznam použitých zkratek

Tabulka 2: Seznam použitých zkratek

Zkratka	Význam
API	Application Programming Interface
CORS	Cross-origin resource sharing
EXE	Executable
JSON	JavaScript Object Notation
Regex	Regular expression
REST	Representational State Transfer
SPŠE	Střední průmyslová škola elektrotechnická
UUID	Universally unique identifier
VOŠ	Vysoká odborná škola

## 9 Seznam obrázků, tabulek, příloh

### Seznam obrázků

1	turtlediary - <a href="https://www.turtlediary.com/game/tic-tac-toe-multiplayer.html">https://www.turtlediary.com/game/tic-tac-toe-multiplayer.html</a>	11
2	Ultimate Tic Tac Toe - <a href="https://ultimate-t3.herokuapp.com">https://ultimate-t3.herokuapp.com</a>	12
3	gametable - <a href="https://gametable.org/games/tic-tac-toe">https://gametable.org/games/tic-tac-toe</a>	13
4	Hlavní stránka aplikace	33
5	Stránka přihlášení	34
6	Stránka přihlášení - údaje administrátora	35
7	Stránka registrace	36
8	Stránka profilu	37
9	Stránka seznamu uživatelů	38
10	Stránka seznamu uživatelů - možnosti administrátora	39
11	Stránka úpravy profilu	40
12	Stránka úpravy profilu - možnosti administrátora	40
13	Stránka seznamu her	41
14	Stránka seznamu pozvánek	42
15	Stránka vytvoření pozvánky	43
16	Stránka vytvoření pozvánky - možnosti administrátora	43
17	Stránka hraní hry	44
18	ER Diagram	53

### Seznam tabulek

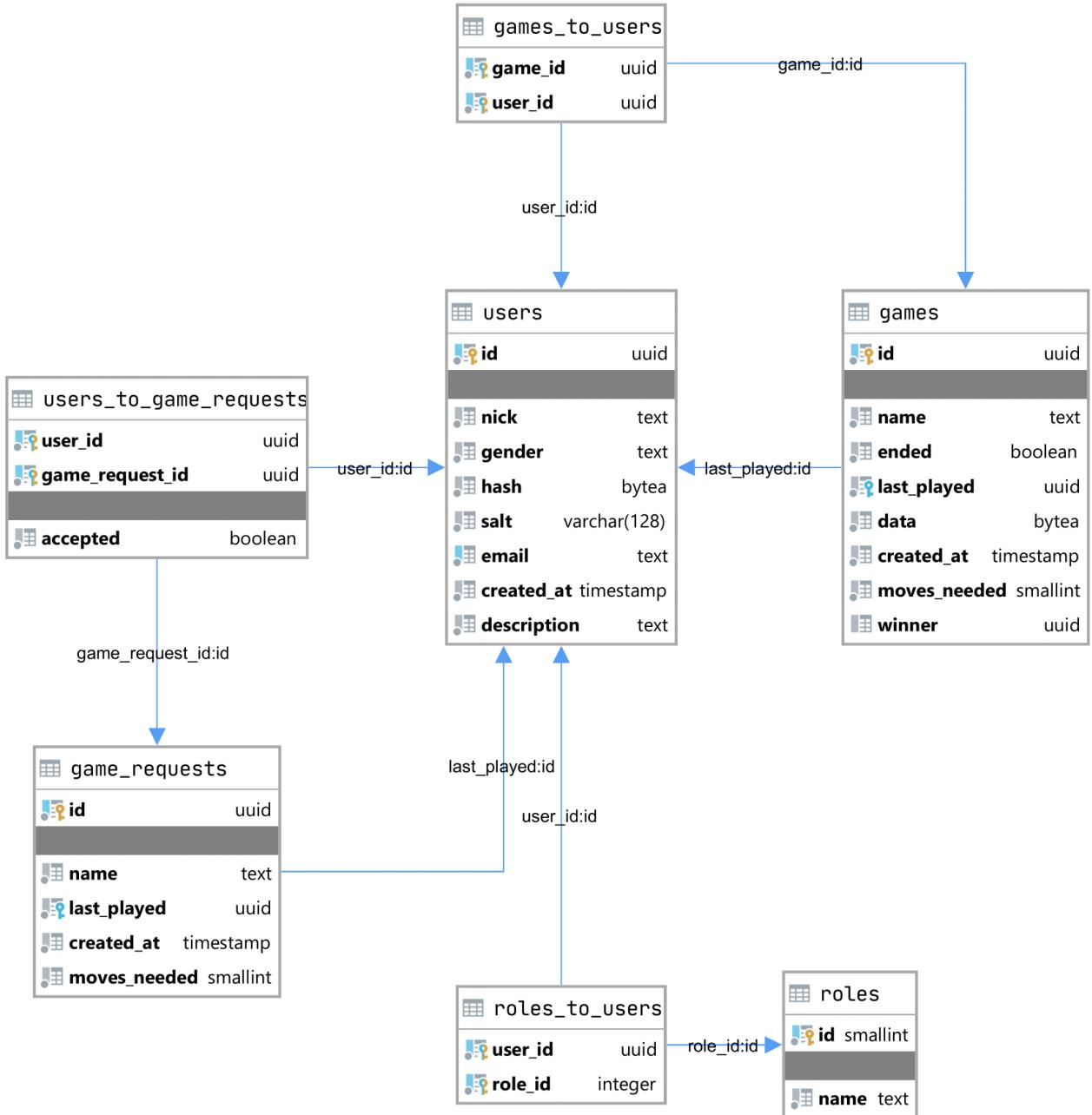
1	Porovnání rychlosti jazyků [1]	10
2	Seznam použitých zkratek	50

### Seznam výpisů

1	Rust - datový typ enum	21
2	Rust - datový typ struct	22
3	Rust - typ trait	23
4	Rust - type	24
5	Příkaz pro komplikaci back-endu	29

6	Příkaz pro kompliaci front-end serveru . . . . .	29
7	Příkaz pro kompliaci front-endu . . . . .	29
8	Nastavení proměnných prostředí pro kontejnery . . . . .	30
9	Nastavení překládání portů pro kontejnery . . . . .	31
10	Úprava hosts souboru pro kontejnery . . . . .	31
11	Příkaz pro start kontejnerů . . . . .	32
12	Údaje administrátora při prvním přihlášení . . . . .	34
13	uikit_addition.css . . . . .	54
14	Procedura pro vytvoření pozvánky . . . . .	55
15	Procedura pro úpravu uživatele . . . . .	56
16	Procedura pro úpravu pozvánky . . . . .	57
17	Zdrojový kód knihovny roles . . . . .	58

## 10 Přílohy



Obrázek 18: ER Diagram

---

### Výpis 13 uikit\_addition.css

---

```
1 body {
2     background-color: #545454;
3 }
4 .uk-navbar-container.uk-light:not(.uk-navbar-transparent)
5     ↵ :not(.uk-navbar-primary) {
6     background: #222;
7 }
8 .uk-dropdown.uk-light {
9     background: #222;
10 }
11 .uk-dropdown li {
12     padding-left: 5px;
13     border-left: 2px solid transparent;
14 }
15 .uk-dropdown li.uk-active {
16     border-color: #545454;
17 }
18 #mobile-navbar li {
19     padding-left: 5px;
20     border-left: 2px solid transparent;
21 }
22 #mobile-navbar li.uk-active {
23     border-color: #545454;
24 }
25 .uk-navbar a {
26     text-decoration: none
27 }
28 .uk-form-danger {
29     color: #f0506e !important;
30     border-color: #f0506e !important;
31 }
32 .uk-notification-message {
33     background: #222;
34 }
35 body > div {
36     padding-bottom: 1px;
37 }
```

---

## **Výpis 14** Procedura pro vytvoření pozvánky

```
1  create procedure new_game_request(_name text, _last_played uuid, _users_id
2    ↵  uuid[], _moves_needed smallint)
3    language plpgsql
4  as
5  $$
6  declare
7    v_game_request_id uuid;
8  begin
9    insert into game_requests (name, last_played, moves_needed)
10   values (_name, _last_played, _moves_needed)
11   returning game_requests.id into v_game_request_id;
12
13   insert into users_to_game_requests (user_id, game_request_id)
14   select user_id__, v_game_request_id
15   from unnest(_users_id) user_id__;
16
17 end;
$$;
```

---

## Výpis 15 Procedura pro úpravu uživatele

---

```
1  create procedure update_user(_id uuid, _nick text DEFAULT NULL::text,
2    _gender text DEFAULT NULL::text, _email text DEFAULT NULL::text, _hash
3    bytea DEFAULT NULL::bytea, _salt character varying DEFAULT
4    NULL::character varying, _roles integer[] DEFAULT NULL::integer[],
5    _description text DEFAULT NULL::text)
6    language plpgsql
7
8  as
9
10 $$
11 begin
12
13   if _nick is not null then
14     update users set nick = $2 where id = $1;
15   end if;
16
17
18   if _gender is not null then
19     update users set gender = $3 where id = $1;
20   end if;
21
22
23   if _email is not null then
24     update users set email = $4 where id = $1;
25   end if;
26
27
28   if _hash is not null then
29     update users set hash = $5 where id = $1;
30   end if;
31
32
33   if _salt is not null then
34     update users set salt = $6 where id = $1;
35   end if;
36
37   if _description is not null then
38     update users set description = _description where id = _id;
39   end if;
40
41
42   if _roles is not null then
43     delete from roles_to_users where user_id = _id;
44     insert into roles_to_users (user_id, role_id) select _id, role_id_
45     FROM unnest(_roles) role_id__;
46   end if;
47
48
49   commit;
50
51 end;
52 $$;
```

---

---

## Výpis 16 Procedura pro úpravu pozvánky

---

```
1  create procedure update_invite (_user_id uuid, _game_request_id uuid,
2      ↵ _accepted boolean, _data bytea)
3      language plpgsql
4  as
5  $$
6  declare
7      v_ready    bool;
8      v_game_id uuid;
9      v_exists   bool;
10 begin
11     select exists(select * from users_to_game_requests where game_request_id
12      ↵ = _game_request_id and user_id = _user_id)
13     into v_exists;
14     if not v_exists then
15         raise exception 'User with id ''%'' is not part of game request with
16         ↵ id ''%'' or game request with id ''%'' doesn''t exists', _user_id,
17         ↵ _game_request_id, _game_request_id;
18     end if;
19     if _accepted then
20         update users_to_game_requests set accepted = true where user_id =
21         ↵ _user_id and game_request_id = _game_request_id;
22         select not exists(
23             select * from users_to_game_requests where game_request_id =
24             ↵ _game_request_id and not accepted
25             ) into v_ready;
26         if v_ready then
27             insert into games (name, data, last_played, moves_needed) select
28             ↵ name, _data, last_played, moves_needed from game_requests where
29             ↵ game_requests.id = _game_request_id returning games.id into v_game_id;
30             insert into games_to_users (user_id, game_id) select
31             ↵ users_to_game_requests.user_id, v_game_id from users_to_game_requests
32             ↵ where game_request_id = _game_request_id;
33             delete from users_to_game_requests where game_request_id =
34             ↵ _game_request_id;
35             delete from game_requests where id = _game_request_id;
36         end if;
37     else
38         delete from users_to_game_requests where game_request_id =
39             ↵ _game_request_id;
40         delete from game_requests where id = _game_request_id;
41     end if;
42     commit;
43 end;
44 $$;
```

---

---

## Výpis 17 Zdrojový kód knihovny roles

---

```
1 use postgres::{Client, NoTls};
2 use proc_macro::TokenStream;
3 use proc_macro2::Span;
4 use quote::quote;
5 use syn::punctuated::Punctuated;
6 use syn::{parse_macro_input, parse_quote, Ident, ItemEnum};
7
8 #[proc_macro_attribute]
9 pub fn get_roles_from_db(_attr: TokenStream, item: TokenStream) ->
10    TokenStream {
11     dotenv::dotenv().expect("Dotenv error");
12     let mut parsed_enum = parse_macro_input!(item as ItemEnum);
13
14     let database_url = std::env::var("DATABASE_URL").expect("Missing env
15     variable DATABASE_URL");
16
17     let mut client = Client::connect(&database_url, NoTls).expect("Couldn't
18     create pool");
19
20     let mut variants = Punctuated::new();
21     for row in client
22         .query("select name, id from roles", &[])
23         .expect("Couldn't get roles from db")
24     {
25         let name = Ident::new(row.get(0), Span::call_site());
26         let id: i16 = row.get(1);
27         let variant: syn::Variant = parse_quote! {
28             #name = #id as i32
29         };
30         variants.push(variant);
31     }
32
33     parsed_enum.variants = variants;
34
35     (quote! {
36         #parsed_enum
37     })
38     .into()
39 }
40 }
```

---