Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky

Bakalářská práce

Systém pro realizaci programátorských soutěží

Plzeň 2022 Martin Bruna

ZÁPADOČESKÁ UNIVERZITA V PLZNI
Fakulta aplikovaných věd
Akademický rok: 2021/2022

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Martin BRUNA**
Osobní číslo:	**A20B0066P**
Studijní program:	**B0613A140015 Informatika a výpočetní technika**
Specializace:	**Informatika**
Téma práce:	**Systém pro realizaci programátorských soutěží**
Zadávající katedra:	**Katedra informatiky a výpočetní techniky**

Zásady pro vypracování

1. Prostudujte existující řešení pro realizaci programátorských soutěží (výhody, nevýhody, možnosti a omezení pro vytvoření vlastní soutěže).
2. Navrhněte a implementujte řešení, které umožní vytvoření vlastní soutěže a bude pokrývat tyto oblasti:
   2.1. Část pro soutěžící (přihlášení, přístup k zadáním, odevzdání řešení, výsledková listina),
   2.2. Část pro organizátory (tvorba a parametrizace soutěže, sledování průběhu, změny parametrů),
   2.3. Vyhodnocovací část (server, na kterém se budou validovat odevzdané úlohy),
   2.4. Bezpečnost a robustnost (správa uživatelských dat, řešení zahlcení a pádu serveru).
3. Ověřte systém v praxi a zhodnoťte dosažené výsledky.

Rozsah bakalářské práce: **doporuč. 30 s. původního textu**
Rozsah grafických prací: **dle potřeby**
Forma zpracování bakalářské práce: **tištěná/elektronická**

Seznam doporučené literatury:

Dodá vedoucí bakalářské práce.

Vedoucí bakalářské práce: **Mgr. Martin Maňák, Ph.D.**
Nové technologie pro informační společnost

Datum zadání bakalářské práce: **4. října 2021**
Termín odevzdání bakalářské práce: **5. května 2022**

L.S.

Doc. Ing. Miloš Železný, Ph.D.
děkan

Doc. Ing. Přemysl Brada, MSc., Ph.D.
vedoucí katedry

V Plzni dne 14. října 2021

Prohlášení

Prohlašuji, že jsem bakalářskou práci vypracoval samostatně a výhradně s použitím citovaných pramenů.

V Plzni dne 23. června 2022

Martin Bruna

Abstract

This bachelor thesis deals with the analysis of existing solutions for programming competitions, the actual implementation and verification of the system in practice. The client and administration interface is developed as a web application. The whole application is divided into smaller logical parts that can scale individually to form a micro-architecture.

Abstrakt

Tato bakalářská se zabývá analýzou existujících řešení programátorských soutěží, vlastní implementací a ověření systému v praxi. Klientské i administrační rozhraní je zpracováno jako webová aplikace. Celá aplikace je dělena do menších logických celků, které se dokáží jednotlivě škálovat a tvoří tak mikro-architekturu.

Obsah

1 Úvod

Tato bakalářská práce se zabývá analýzou, návrhem a implementaci systému pro soutěžní programování. Hlavní motivací této práce bylo prozkoumání a vytvoření otevřeného systému nad mikroservisní architekturou pro možné škálování.

První část této práce popisuje soutežní programování jako celek, obsahuje přehled různých typů soutěží. Zároveň popisuje význam a potřebu validačního systému s přehledem vybraných existujících řešení, jejich výhod a nevýhod.

Na základě analyzovaných existujících soutěží a řešení následuje vyslovení požadavků na obecný validační systém. Z obecných požadavků je následně vyvozena nutná specifikace a návrh celé aplikace. V rámci této části jsou rozebrané možné modely architektury a také rozdělení aplikace do logických celků.

Následně se práce bude zabývat samotnou implementací. Jak je následně rozdělen kód v rámci jednotlivých částí systému a také jaké technologie jsou pro práci využity.

Na závěr se tato práce věnuje otestování celého systému. Práce byla otestována na živé soutěži a byl proveden test robustnosti na validační části.

2 Soutěžní programování

Soutěžní programování je programátorsky sportovní disciplínou pro porovnání sebe nebo svého týmu s ostatními. Soutěže mají většinou několik různých zadání s jasně určenými vstupy a výstupy. V zadání je popsán problém, formát vstupu i výstupu a ve většině případů i ukázkový vstup a jeho výstup. Zadání také obsahuje časové a paměťové limity, které musí řešení splnit. Na každém pak je vytvořit program, který dokáže podle zadání ze zadaných vstupů vytvořit správné výstupy. Během řešení je důležité přemýšlet nad časovou a paměťovou složitostí daného problému, ale také nad krajními případy, které v zadání nebyly popsány, ale je potřeba aby je program vyřešil.

Dnešní soutěže bývají konané ve většině případů napříč internetem, ale některé (včetně vyšších kol některých, které jsou z počátku on-line) stále fungují v lokálním prostředí. V obou případech v rámci soutěže existuje ať už na lokální síti nebo v internetu systém, který slouží k předání zadání, odevzdávání řešení a následnému náhledu výsledku validace a pořadí.

V období před internetem, kdy probíhaly první ročníky asi nejstarší soutěže ICPC (International Collegiate Programming Contest), se zadání roznášelo pouze v tištěné podobě, odevzdávání řešení fungovala nahráním na disketu a odnesením k validaci. A pro představu aktuálního pořadí za každou vyřešenou úlohu byl týmu přivázán ke stolu balónek s heliem. Tato tradice přetrvala dodnes. Množství heliových balonků se tak stalo charakteristickým znakem evropského kola této soutěže a je součástí loga.

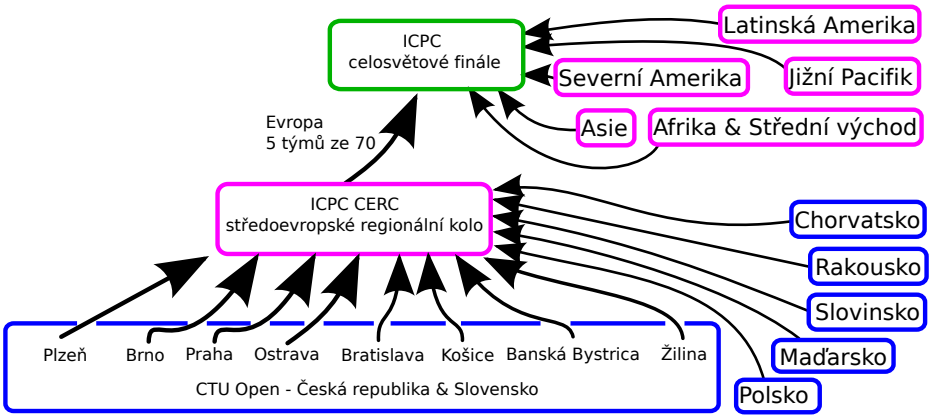
2.1 Soutěže

Následující podkapitola obsahuje přehled existujících soutěží. Vzhledem k velkému množství existujících soutěžích se budeme zaměřovat pouze na výběr známějších soutěží ať už ve světě nebo v České a Slovenské Republice.

Výběr těchto soutěži se zaměřuje na soutěže, které jsou vzhledem ke svému průběhu relevantní pro tuto práci.

2.1.1 ICPC

Již zmíněná a nejznámější soutěž v univerzitní sféře je ICPC – soutěž pro až 3 členné týmy vysokoškolských studentů z celého světa. Soutěž je rozdělena do několika regionů, ze kterých se postupuje až do světového finále, jak je znázorněno na obrázku 2.1. Soutěž probíhá lokálně.



Obrázek 2.1: Struktura soutěže ICPC. Převzato z [8].

2.1.2 Google Code Jam

Naopak Google Code Jam je soutěž provozovaná soukromým firemním subjektem, je určena jednotlivcům nezávisle na jejich věku a vzdělání. Soutěž se dělí na 3 na sebe navazující on-line kola a následné off-line finále.

V rámci každého kola existuje v soutěži několik problémů, body mezi problémy nejsou rozděleny ekvivalentně, ale podle složitosti, zároveň standartně má problém 3 typy testovacích sad a za každou sadu pro kterou je řešení úspěšné dostane soutěžící různé body. Výsledek třetí testovací sady je zpravidla skrytý do konce kola.

2.1.3 PilsProg

PilsProg je soutěž pořádaná Fakultou aplikovaných věd Západočeské univerzity pro studenty středních škol od 15ti let. Pořádaná od roku 2008 [11].

Soutěž má dvě kola, prvním kolem je online kvalifikace a následuje offline finálové kolo.

V rámci této soutěže existuje také kategorie PilsProg Lite, která se skládá pouze z jednoho online kola a je určena začínajícím programátorům.

Soutěžící odevzdává řešení jako zdrojový kód, který validační server následně spustí a zkontroluje oproti očekávaným vstupům.

2.1.4 Kasiopea

Kasiopea je soutěž pořádaná Matematicko-fyzikální fakultou Univerzity Karlovi. Soutěž je určená studentům středních škol [6].

Soutěž se skládá ze dvou kol, domácího kola, které je otevřeno několik dní a probíhá online a z finále, které se koná offline.

Úlohy v rámci soutěže se dělí na jejich lehčí a těžší variantu. V případě vyřešení těžší varianty dostane soutěžící více bodů.

Soutěžící mají k dispozici vstupní data a odevzdávají pouze výstup. Díky tomu je možné řešení napsat v jakémkoliv programovacím jazyce.

2.1.5 Matematická olympiáda kategorie P

Soutěž částečně podobná PilsProgu je pořádána Ministerstvem školství. Soutěž má tři úrovně první online a zbylé offline. Kromě úloh s automatickou validací je součástí soutěže i teoretická část, kde soutěžící řešení problému pouze popisují pseudokódem na papír [10].

2.1.6 HackerRank

HackerRank není klasickou soutěží, nemá totiž jednu hlavní soutěž, ale během celého roku běží menší soutěže, kterých se může zúčastnit kdokoliv a kdykoliv. Vše se odehrává on-line.

2.1.7 Coding Game

Nejpestřejší platforma pro soutěžní programování s aktivní komunitou a možností získávat různá ocenění.

- Klasické úlohy s validací soutěžícímu neviditelných vstupů
- Programování botů, kteří proti sobě soupeří v různých prostředích a hrách
- Code golf - soutěž v naprogramování algoritmu v jakémkoliv jazyce na co nejméně znaků
- Reverse - Úloha bez zadání pouze s větším množstvím ukázkových vstupů a výstupů, na soutěžícím je zjistit jakým způsobem získat ze vstupu dané výstupy

2.1.8 Cloudflight Coding Contest

Jedno-kolová soutěž konající se zároveň off-line i online, soutěžící mají k dispozici vstupy a odevzdávají pouze výstupy. Jednotlivá zadáni na sebe navazují a odemykají se vždy až při vyřešení předchozích.

2.1.9 Facebook Hacker Cup

Soutěž dělící se na 5 online kol, jednotlivé úlohy jsou bodované podle složitosti, soutěžící dostanou vstupy, které validují spuštěním řešení na vlastním počítači.

3 Existující řešení

3.1 HackerRank

HackerRank je uzavřenou platformou, pro soutěžní programování. Pro programátory nabízí velké množství úloh. Pravidelně se zde pořádají veřejné soutěže od samotného HackerRanku. Platforma je také využívána společnostmi pro hledání talentů.

3.1.1 Výhody

Není potřeba cokoliv nasazovat a stačí jednoduše nakonfigurovat. Systém má zároveň velkou knihovnu již vytvořených problémů. A integrovaný editor zdrojových kódů pro tvorbu řešení jednotlivých úloh.

3.1.2 Nevýhody

Systém je závislý na jeho poskytovateli, nelze jej nasadit v lokálních podmínkách a potřebuje neustálý přístup k internetu. Absence kompletního API.

3.1.3 Tvorba vlastní soutěže

3.1.3.1 Možnosti

V rámci konfigurace soutěže je možné nastavit texty pro úvodní stránku soutěže, typ skórování a parametry pro rozhodnutí remízy. Nebinární vyhodnocení úlohy.[1]

3.1.3.2 Omezení

Neobsahuje možnost importu nastavení ze souboru. Nemožnost nastavení povinných úloh pro otevření další úlohy.

3.2 DOMjudge

DOMjudge je opensource systém na míru postavený pro soutěž ICPC, kde se částečně začal používat v roce 2004. Od roku 2012 se systém, alespoň z části používá i na světovém finále.

[1] Částečné body za úlohu jsou uděleny i když řešení nesplní všechny testovací scénáře

3.2.1 Výhody

DOMjudge má veřejně dostupné zdrojové kódy s licencí GNU a je tedy možné jej nasadit pro vlastní použití. Zároveň obsahuje kompletní API podle specifikace pro ICPC soutěže. Systém má také jednoduchou možnost přidat další jazyk pro možnou validaci.

3.2.2 Nevýhody

Není stavěné na jiné soutěže, které nejsou podle ICPC, systém dokáže pouze binární vyhodnocování úloh.

3.2.3 Tvorba vlastní soutěže

3.2.3.1 Možnosti

Systém umí nastavit jméno, datum a čas začátku soutěže, jeho dobu, čas penalizace za špatné odevzdání a dobu, jak dlouho před koncem dojde ke zastavení aktualizace výsledkové tabulky. Zároveň je možné všechny data k soutěži importovat ze souboru.

3.2.3.2 Omezení

Nemožnost konfigurace jiných pravidel pro řazení týmů. Nemožnost nastavení povinných úloh pro otevření další úlohy.

3.3 ICPC Live Archive a Online Judge

Dva projekty běžící na stejném systému, jde o velmi zastaralý systém, bez aktivního vývoje, z roku 2005. V dnešní době si drží relevanci mezi ostatnímy díky zásobě úloh které mají.

3.3.1 Výhody

Velký archiv a databáze různých problémů s validátorem.

3.3.2 Nevýhody

Funguje pouze jako validátor jednotlivých úloh, systém není veřejný a není možné vytvořit vlastní soutěž. Zároveň podporuje pouze zlomek jazyků oproti ostatním řešením. Jde o jeden z nejstarších systémů a v dnešní době s malou podporou ze strany provozovatele, tuto situaci se provozovatel snažil řešit

několika pokusy o fundraising (Indiegogo[2] a Patreon[3]), které bohužel byli neúspěšné. Kvůli stáří celé platformy a nedostatku financí pro podporu se stává, že je systém nefunkční, popřípadě nějaké úlohy jsou validovány špatně a oprava ze strany provozovatele trvá delší dobu.

[2] `https://www.indiegogo.com/projects/uva-online-judge-new-platform/`
[3] `https://www.patreon.com/onlinejudge`

4 Návrh

Tato kapitola se věnuje vyslovení obecných požadavků na možný validační systém, specifikaci implementované funkčnosti, návrhu celé aplikace - rozdělení na logické celky a popisu komunikace mezi jednotlivými částmi.

4.1 Obecné požadavky na validační systém

4.1.1 Část pro soutěžící

Tato část systému musí umět uživatele přihlásit, zobrazovat jednotlivá zadání, dát uživateli možnost odevzdat řešení a zobrazit výsledek validace.

Dalšími možnostmi, jak vylepšit tuto část je možnost zobrazovat celkové výsledky, přidání editoru zdrojového kódu přímo v rámci rozhraní. Pro otevřené soutěže by neměla chybět možnost registrace popřípadě správa týmu.

4.1.2 Část pro organizátory

Od části pro organizátory se očekává nastavitelnost celé soutěže. Základem je vytvoření soutěžního kola, zadání problémů pro toto kolo a následná správa uživatelů.

V rámci nastavení kola je nutností možnost nastavení času spuštění a konce daného kola. V případě problémů, je potřeba možnost vložení zadání a také testovacích scénářů. Pro správu uživatelů je potřeba mít možnost uživatele vytvořit, editovat a smazat, přihlásit na jednotlivá kola, popřípadě resetovat heslo uživatelů.

Některé systémy zároveň mají možnost importu jednotlivých částí nastavení soutěže, popřípadě umí nastavit parametry skórování problémů. Dalšími schopnostmy je, porovnávání týmů v případě bodové remízy, možnost revalidovat odevzdané řešení společně s možností editace zadání v průběhu soutěže.

4.1.3 Validační část

Podstatnou částí systému je samotná validační část, která přijme řešení odevzdané od soutěžících a zkontroluje jejich validitu proti zadané kontrole. Většina validátorů dokáže zpracovávat tři typy úloh.

- **Statické sady vstupů a výstupů** - Do odevzdaného programu jsou předány vstupy a porovnány s referenčními výstupy
- **Dynamická validace** - Řešení je spuštěno oproti scriptu, který kontroluje jeho správnost
- **Veřejné vstupy** - Všechny vstupy jsou k dispozici veřejně a soutěžící si své řešení spouští na svém počítači a odevzdávají pouze jednotlivá řešení

Dobrá validační část zároveň umí prioritizovat validaci úloh pro týmy, které neodevzdávají špatná řešení. Zároveň dokáže validovat úlohy paralelně.

4.2 Specifikace

Specifikace popisuje všechny potřebné vlastnosti, které aplikace musí splňovat.

- Aplikace bude navenek fungovat jako webová služba
- Klientská část bude zpracována jako Single Page Aplikace[1]
- Klientská část bude komunikovat se serverem přes REST API[2]
- Administrační část bude zpracována jako klasická webová aplikace s vykreslením celé stránky na straně serveru
- Klientská část a administrační část v rámci serveru budou používat pro předávání dat společnou SQL databází (MySQL) a také key-value[3] databází (Redis).
- Klientská část a validační část serveru bude rozdělena na mikroslužby[4], které díky tomu půjdou škálovat dle potřeby
- Klientský front-end se bude jednotlivým mikroslužbám autorizovat pomocí tokenu obsahujícího nejdůležitější data o stavu
- Mikroslužby budou implementované za použití mikroframeworku Lumen

[1]Single Page Aplikace (SPA) je aplikace která na klientské straně pouze dynamicky překresluje stránku na základě dat, která získá z API

[2]REST API - aplikační rozhraní oddělující jednotlivé zdroje (resources) do jednotlivých koncových bodů

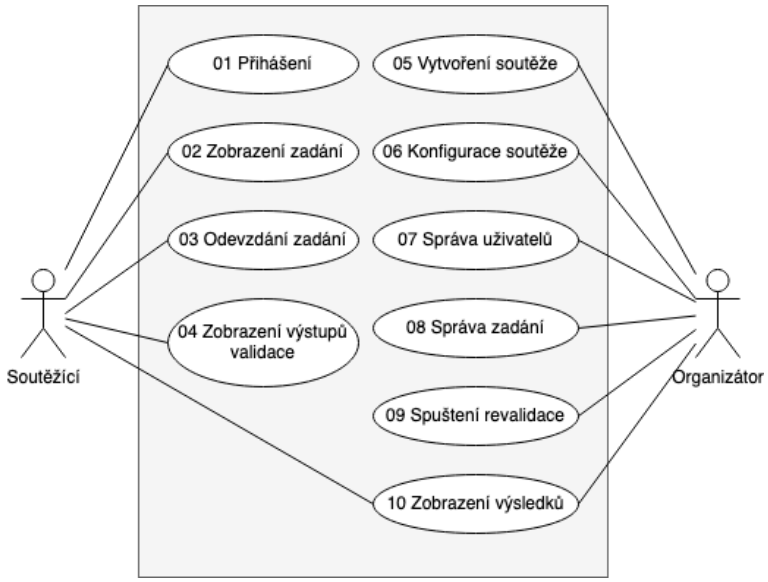
[3]Key-value databáze je databází, která k danému klíči uloží jakoukoliv hodnotu

[4]Mikroslužby jsou samostatné části systému obsluhující malou část aplikace

- Administrační část bude implementována za použití frameworku Laravel

4.3 Funkcionalita

Systém jako celek musí splňovat alespoň minimální možnost pro průběh soutěže, v diagramu na obrázku 4.1 jsou znázorněni uživatelé systému a jejich případy použití. Aplikaci využívají dva typy osob **Soutěžící** a **Organizátor**.



Obrázek 4.1: Případy užití aplikace

4.3.1 Popis případů užití

01 Přihlášení

Soutěžící se může přihlásit do systému za pomocí svého e-mailu a hesla.

02 Zobrazení zadání

Soutěžící během běžícího kola může zobrazovat zadání jednotlivých úloh.

03 Odevzdání zadání

Soutěžící během běžícího kola může odevzdat své řešení daného problému. Systém následně pošle řešení k validaci.

04 Zobrazení výstupů validace

Soutěžící během běžícího kola může zobrazovat validační výstupy pro svá odeslaná řešení.

05 Vytvoření soutěže

Organizátor může vytvořit soutěž a kolo dané soutěže.

06 Konfigurace soutěže

Organizátor může konfigurovat soutěžní parametry.

07 Správa uživatelů

Organizátor může spravovat uživatele, vytvořit nový účet, smazat účet, editovat jakékoliv informace. Zároveň může přihlásit uživatele do kola soutěže.

08 Správa zadání

Organizátor může tvořit a editovat zadání jednotlivých problémů a zadávat jeho testovací sady.

09 Spuštění revalidace

Organizátor může nad vybranou množinou odevzdaných řešení spustit validaci znovu. Tato funkcionalita je důležitá například při opravě testovacích sad, pádu validačního serveru nebo jeho opravy.

10 Zobrazení výsledků

Soutěžící i Organizátor může zobrazovat výsledky soutěžního kola.

4.4 Zvažované a použité modely architektury

Následující podkapitoly popisují jednotlivé možnosti pro architekturu aplikace a jejich následné použití v rámci aplikace.

4.4.1 Klient - Server Architektura

Architektura na bázi sdíleného výkonu pro jednotlivé klienty na straně serveru, který poskytuje klientovi data a služby na bázi žádost-odpověď [1]. Tato architektura je vhodná v případě potřeby existence a kontroly jednoho zdroje pravdy, což je i případ této práce. Nevýhodou je potřeba dostatku výkonu pro každého klienta, kdy náročnost a cena s každým klientem roste.

4.4.2 Fronta zpráv

Fronta zpráv umožňuje uložit zprávu pro její další zpracování jinou službou nabízí tedy asynchronní zpracování úloh [9]. Vzhledem k době zpracování validace jednotlivých úloh však není možné čekat synchronně na validaci úlohy během klientského požadavku na server, a tak je v tomto případě vhodné pouze uložit informaci o potřebě validace a klientovi vrátit odpověď o zařazení do fronty.

4.4.3 Monolitická aplikace

Monolitická aplikace je v dnešní době stále běžný způsob tvoření softwaru, existuje jedna nedělitelná část, která pokrývá všechny požadavky na celý systém [14].

Výhodou monolitické aplikace je sdílený zdrojový kód a jednoduchost provazovat jednotlivé části aplikace. Naopak nevýhodou je její nedělitelnost pro případné škálování. Ve chvíli, kdy je potřeba škálovat monolitickou aplikaci, je jedinou možností aplikaci spustit několikrát a všechny žádosti na aplikaci rozdělovat na jednotlivé instance.

V rámci vytvářeného systému, kde je potřeba škálovat služby pro soutěžícího, je tato architektura nevhodná, ale pro administrační rozhraní, které obsahuje komplexnější logiku a zároveň není taková potřeba tuto část škálovat, je tato architektura vhodná.

4.4.4 Mikroslužby

Naopak architektura mikroslužeb aplikaci dělí na mnoho menších služeb, kde každá sama o sobě obsluhuje pouze malou podmnožinu funkčnosti. Každá z

mikroslužeb může fungovat jako služba odpovídající na požadavky klienta a také jako služba pro ostatní mikroslužby. [14]

Výhodou této architektury je jednoduchá možnost škálovat jednotlivé části systému nezávisle na ostatních. Další výhodou je možnost mít jednotlivé služby implementované v jiném jazyce. Nevýhodou je horší správa zdrojových kódů a složitost nasazování systému jako celek.

Tato architektura je vhodná pro použití v rámci soutěžní části serveru, vzhledem k potřebě škálovat jednotlivé pod-části.

4.4.5 Single Page Application (SPA)

Single Page Application neboli jednostránková aplikace je moderní přístup k tvorbě webových aplikací, kdy dochází k dynamickému překreslování stránky. Na začátku si klient jednou[5] stáhne celou logiku front-end aplikace. Následně komunikuje se serverem pouze přes API [12].

Výhodou tohoto přístupu je omezení přenesených dat a ulehčení práce serveru pro generování celé stránky. Nevýhodou tohoto přístupu je omezená funkčnost stránky s vypnutým JavaScriptem v prohlížeči a indexování stránky vyhledávači.

Vzhledem k tomu, že jde kompletně o aplikaci, u které stejně není co indexovat (vše je až za přihlášením) a zároveň je žádoucí, aby serveru zbylo co nejvíce výkonu na samotnou klientskou část, tak tento přístup je pro navrhovaný systém vhodný.

4.4.6 REST API

REST API je architekturou bez stavového rozhraní mezi jednou webovou službou a klientem (nebo další webovou službou). Aplikační rozhraní podle REST rozděluje jednotlivé části rozhraní do skupin podle jednotlivých zdrojů, kde v každé skupině definuje akce pro čtení, vytváření a úpravu. [4] Příklad rozdělení podle standardu je ukázán v tabulce 4.1

REST API je v dnešní době jedním z nejvyužívanějších standardů pro tvoření rozhraní webových aplikací, jeho využitím lze tedy získat přehlednost rozhraní, díky všeobecné znalosti. Pro vytvářenou aplikaci je tedy tento návrh vhodný pro rozhraní v rámci části aplikace pro soutěžícího, mezi již zmíněnou Single Page Aplikací a jednotlivými mikroslužbami.

[5] V případě přístupu k aplikaci znovu bude načtena aplikace z cache prohlížeče

Akce	HTTP metoda	URI
Získání seznamu	GET	/uzivatel
Získání záznamu	GET	/uzivatel/:id
Vytvoření záznamu	POST	/uzivatel
Úprava záznamu	PUT	/uzivatel/:id
Smazání záznamu	DELETE	/uzivatel/:id

Tabulka 4.1: Příklad skupiny pro zdroj uživatel

4.4.7 Model Vew Controller (MVC)

Jde o rozdělení aplikace na tři vrstvy [14].

- Model - Modelová vrstva, komunikuje s daty převážně databází
- View - Zobrazovací vrstva starající se o naplnění dat do šablon
- Controller - Logická vrstva propojující datovou a zobrazovací vrstvu

V rámci této architektury je zároveň možné v případě potřeby ještě mezi Controller a Model vložit další vrstvy a tvořit tak čtyř/pěti vrstvou architekturu. Čtvrtou vrstvou aplikace může být například vrstva repozitářů, které obalují konkrétní databázovou vrstvu a oddělují jednotlivé operace od samotné databázové vrstvy popřípadě logické vrstvy. Dalším příkladem může být vrstva služeb, kam lze extrahovat společná logika z konkrétních kontrolerů.

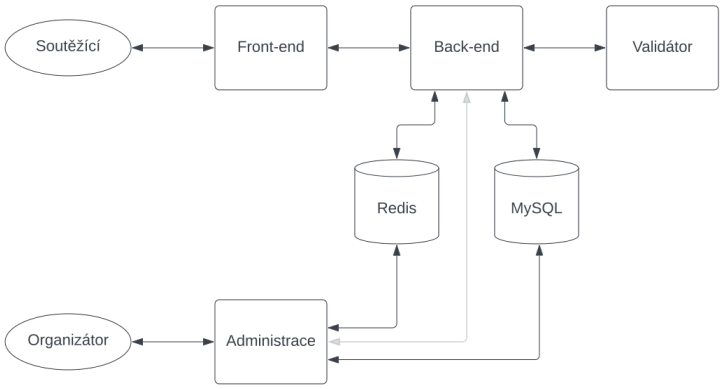
Pro většinu případů v praxi je tento způsob vhodným rozdělením aplikace a při jeho dodržování si aplikace udržuje jasnou strukturu. Tento přístup může být pro vývojáře v některých případech svazující a motivovat je psát delší nepřehledné metody v rámci jednotlivých vrstev nebo extrahovat logiku mimo, je tedy potřeba dbát na včasné přidání konzistentní vrstvy pro extrakci logiky pro udržení jasné struktury.

Vzhledem k velikosti aplikace je tento přístup vhodný pro celou část administrační aplikace. Na část pro soutěžní API je tento model nevhodný, protože API neobsahuje šablony, ale předává pouze data ve formátu JSON[6].

4.5 Návrh architektury aplikace

Aplikace bude rozdělena do několika částí podle diagramu na obrázku 4.2 popsaných v následujících podkapitolách.

[6]JSON - JavaScript Object Notation



Obrázek 4.2: Pohled na aplikaci jako celek

4.5.1 Front-end - Klientská aplikace

Klientská aplikace je hlavní rozhraní, se kterým komunikuje soutěžící. Tato část je napsaná jako Single Page Aplikace, která komunikuje s klientskou částí serveru.

4.5.2 Back-end - Klientská část serveru

Klientská část serveru je část pro soutěžícího, která se skládá z několika mikroslužeb blíže popsaných v části 4.6. Navenek tato část dává k dispozici API a interně komunikuje s validátorem. Data ukládá a čte z MySQL a Redis databáze. Pro komunikaci s validátorem používá zprávy v Redis frontě.

4.5.3 Administrace

Administrace je jedna monolitická služba pro správu celé soutěže. Konfiguraci soutěže ukládá do MySQL databáze a zároveň ukládá data pro Back-end do Redisu.

4.5.4 Validátor

Validátor čte zprávy z Redis fronty, na základě kterých pak provede validaci odevzdaného řešení a výsledek zapíše do MySQL databáze.

4.5.5 Ostatní služby

Kromě předchozích služeb systém obsahuje dva typy databází – MySQL databázi pro permanentní data soutěže a key-value databázi Redis, která

slouží jako rychlá cache dat a zároveň jako fronta pro odevzdaná řešení od soutěžících určená k validaci.

4.6 Rozdělení mikroslužeb Back-end služby

Následující sekce popisuje rozdělení Back-end části na jednotlivé mikroslužby, které je možná následně nezávisle škálovat.

4.6.1 Služba uživatele

Služba uživatele se stará o registraci a přihlášení uživatele. Během přihlášení získá všechny potřebná data o uživateli a vrátí je zakódovaná a podepsaná v rámci JWT tokenu (více o JWT v podkapitole 5.2.8).

4.6.2 Služba týmu

Služba týmu se stará o vytvoření týmu, zvaní jiných uživatelů do týmu, akceptování žádosti o přidání se do týmu a v neposlední řadě i o opuštění týmu.

4.6.3 Služba stavu

Služba stavu aktualizuje stav přihlášeného uživatele a vrátí aktualizovaný JWT token.

4.6.4 Služba zadání

Služba, která se stará o výpis zadání a zároveň o výpis a stav odevzdaných řešení, které se vážou ke konkrétnímu zadání.

4.6.5 Služba odevzdávání

Služba, která se stará o uložení odevzdaného řešení do databáze a jeho předání validační části.

4.6.6 Služba výsledků

Služba, která se stará o výpis aktuálního stavu žebříčku společně se stavem odevzdaných úloh každého týmu.

4.7 Kombinace do jednoho systému

Systém obsahuje několik služeb a mikroslužeb, které musí v rámci jedné domény existovat jako celek. Zároveň je potřeba zařídit distribuci klientských požadavků mezi instance jednotlivých služeb v případě jejich škálování.

4.7.1 Dockerizace

Na jednotlivé části budou využité Docker kontejnery, které následně dovolí jednoduché nasazení na škálovací architektury cloudových služeb jako např. Google App Engine, Microsoft Azure, AWS a jiné.

4.7.2 Vstupní bod / Vstupní služba

Aby systém mohl vystupovat navenek jako jedna služba je potřeba jednotlivé služby postavit za jednotnou proxy. Jako proxy bude využita služba Nginx, který na základě celé URL rozhodne na jakou službu má požadavek pokračovat.

4.7.3 Load balancing

Load balancing nebo-li vyvažování zátěže je technika rozdělování požadavků na jednotlivé instance dané služby. Pro realizaci v rámci vyvíjeného systému byly zvažované dvě níže popsané možnosti. viz 4.7.3.2. Pro jednoduchost byla vybrána druhá varianta.

4.7.3.1 HAProxy

Výkonné a široce konfigurovatelné řešení pro load balancing, Dovoluje například na základě hlaviček určovat cílovou instanci, přepisovat URL. Ale zároveň vyžaduje složitou konfiguraci.

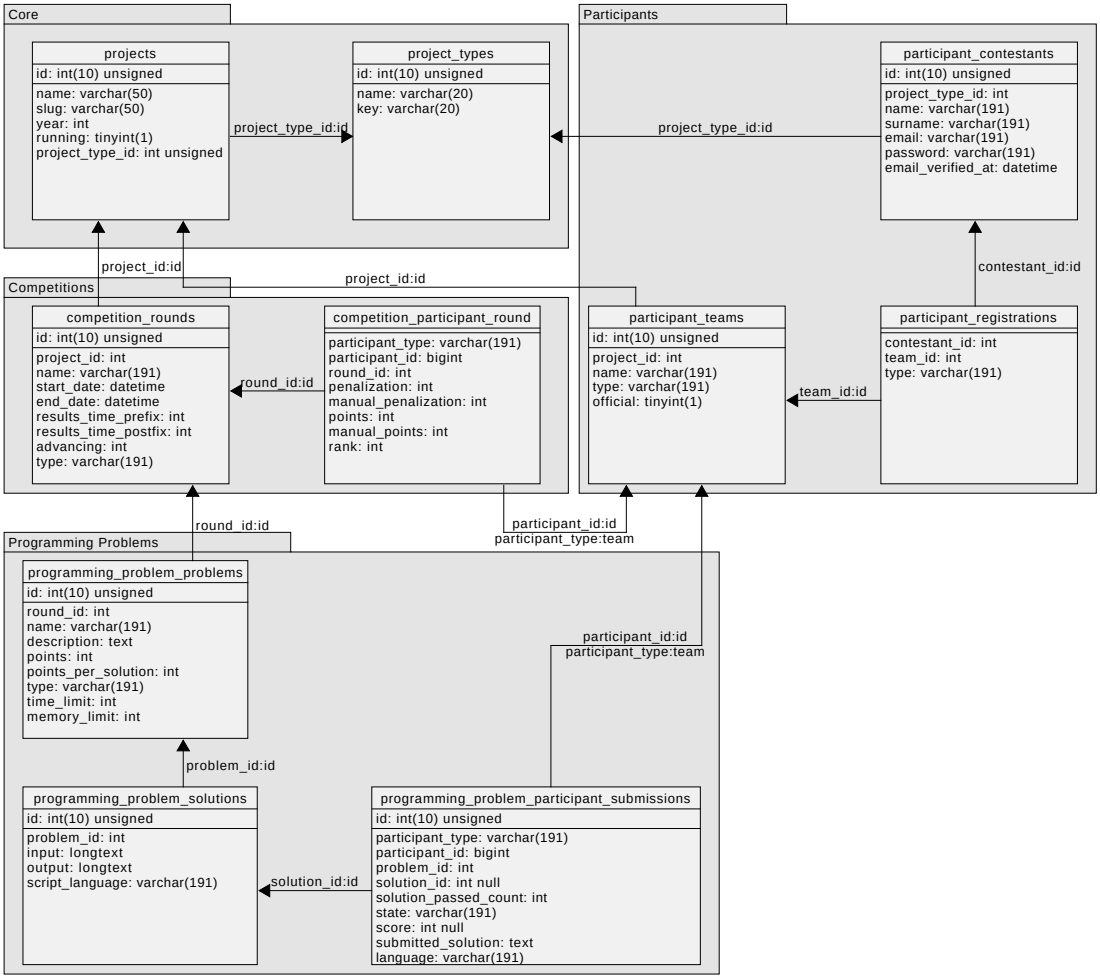
4.7.3.2 Docker DNS Round Robin

Docker interně obsahuje vlastní DNS server, díky kterému jsou k dispozici jednotlivé služby podle jména. Samotný DNS server vrací interní IP k jednotlivé službě v náhodném pořadí a může tak fungovat jako náhodný load balancing [3].

4.8 Návrh databáze

Na obrázku 4.3 je znázorněn zjednodušený model databáze. Na modelu chybí například tabulky pro administrátorské uživatele a definicí jejich práv a rolí, model se soustředí pouze na důležité části týkající se této práce.

Databáze je rozdělena do logických celků, podle kterých je následně i rozdělena administrační část do jednotlivých modulů.



Obrázek 4.3: Zjednodušený ERA model databáze

Jádro - Core

Základem aplikace je správa projektů pro umožnění v jednom systému mít více soutěží a v rámci každé ze soutěží jednotlivé ročníky soutěže. Samotnou soutěž reprezentuje tabulka `project_types` na kterou je navázán ročník soutěže v tabulce `project`

Účastníci - Participants

Prvním logickým celkem je část pro účastníky, která zajišťuje data pro správu uživatelů a týmů. Tabulka `participant_contestants` ukládá samotné uživatele, kteří jsou navázáni na jednotlivé soutěže. V rámci ročníků soutěže jsou zakládány týmy (tabulka `participant_teams`, které jsou s účastníky propojeni `M:N` vazbou přes tabulku `participant_registrations`, která určuje typ náležitosti v týmu (například správce, člen, pozvánka)

Soutěž - Competitions

Druhý celek je samotná soutěž, definující soutěžního kola a jejich nastavení (tabulka `competition_rounds`). V rámci svého nastavení ukládá začátek a konec kola, čas zmražení výsledků ze začátku a konce, počet postupujících a typ daného kola (primárně určeno na označení vstupního kola pro automatickou registraci nových týmů). Zároveň je v této části tabulka `competition_participant_round` fungující jako `M:N` spojující jednotlivé týmy (připraveno i na vazbu s účastníky, díky polymorfické vazbě) se soutěžními koly a ukládá jejich stav v rámci daného kola.

Úlohy - Programming Problems

Poslední částí jsou samotné úlohy, jejich validační řešení a odevzdaná řešení. Samotné úlohy jsou definované tabulkou `programming_problem_problems`, která ukládá název a zadání úlohy, její body za kompletní a částečné řešení, její typ, časový a paměťový limit. Typem úlohy můžou být skryté vstupy, veřejné vstupy nebo dynamická validace. Na tuto tabulku je navázaná tabulka s jednotlivými vzorovými řešeními `programming_problem_solutions`, v případě že jde o problém s dynamickou validací, je zde uložen i jazyk ve kterém je napsaný validační script.

Poslední tabulkou je tabulka `programming_problem_solutions`, která eviduje jednotlivé odevzdané řešení, jejich napojení na tým (učastníka), na úlohu a v případě veřejných vstupů taky na konkrétní validační řešení, zároveň ukládá stav validace a získané body.

5 Implementace

Tato kapitola popisuje samotnou implementaci postavenou na základě návrhu popsaného v rámci kapitoly 4. V kapitole jsou popsané využité technologie a důvody jejich využití v aplikaci. Zároveň je popsáno, jak je kód aplikace členěn.

5.1 Monorepozitář - Monolitický repozitář

Pro vývoj byl využit verzovací systém git[1]. Vzhledem k tomu, že aplikace se dělí na několik částí, dalo by se uvažovat o rozdělení každé části do svého samostatného repozitáře, avšak nastává problém jak udržet jednotlivé části aplikace sjednocené. Například při úpravě API v rámci klientské části serveru, je potřeba tuto změnu reflektovat i v rámci klientské aplikace, tato skutečnost pak nutí zavádět označování jednotlivých verzí v rámci repozitářů a definování jejich kompatibilit. Tento problém se dá vyřešit takzvaným monolitickým repozitářem, který má v sobě všechny části systému.

Každý přístup má samozřejmě své výhody a nevýhody. Hlavní výhoda monorepozitáře byla nastíněna a je jí konzistence verze mezi jednotlivými částmi. Z této výhody vyplývá i zajištění jednoho bodu pravdy a eliminace řešení závyslostí. Další výhodou je daleko jednodušší implementace CI/CD[2].

Naopak výhodou rozdělených repozitářů je kontrola nad přístupem k jednotlivým částem aplikace, například pokud je pouze klientská aplikace open-source. Další výhodou mohou být obrovské systémy, kde celý monorepozitář bude mít velikost v řádech GB a vývojáři pracují pouze na jedné části, tak může být komfortnější mít systém rozdělen.

Vzhledem k tomu, že aplikace nepotřebuje mít oddělené přístupy do jednotlivých částí systému a nebude mít obrovskou velikost, tak bylo pro vývoj jednodušší využít monorepozitář a zajistit tak jednoduchou konzistenci mezi částmi aplikace.

5.1.1 Hlavní rozdělení systému v rámci repozitáře

- **backend** - klientská část serveru

[1] git - systém ukládání rozdílových změň umožnující tvořit stromovou historii, zdrojových kódů

[2] Continuous integration a Continuous delivery - automatická kontrola stavu kódu a jeho automatické nasazování

- **checker** - kompilační a spouštěcí scripty pro validaci
- **core** - administrační část s definicí databáze
- **enums** - společný kód obsahující číselníky
- **frontend** - klientská aplikace
- **orchestration** - konfigurace nasazení aplikace

5.2 Použité technologie

Následující podkapitola ukazuje hlavní technologie použité pro vývoj celého systému a jejich využití.

5.2.1 Docker

Docker je software, který virtualizuje operační systém, dovoluje tvořit konzistentní prostředí pro běh aplikace a zároveň dokáže rozdělovat celý systém do jednotlivých kontejnerů, které běží nezávisle na sobě. [2] Jednotlivé kontejnery mohou zároveň běžet ve více instancích a starat se tak o load-balancing (viz sekce 4.7.3.2). Pro nastavení jednotlivých kontejnerů je využita nadstavba `Docker compose`, která na základě konfiguračního souboru připraví celé prostředí pro běh systému.

5.2.2 Nginx

Nginx je web server, sloužící pro rozdělování jednotlivých HTTP požadavků na jednotlivé služby nebo přímo na vracení statických souborů. V rámci celého systému figuruje jako jednotný vstupní bod pro všechny požadavky.

5.2.3 Redis

Redis je open-source key-value databáze. Celá databáze je optimalizovaná pro rychlost a svá data má za běhu v paměti (i když pro případ pádu je ukládá i na disk). Zároveň tím, že funguje pouze jako úložiště na bázi klíčů, dokáže ukládat a vyhledávat data mnohem rychleji. Díky tomu nabízí rychlost, která je mnohem větší než klasické relační databáze. Tato architektura s sebou nese i nevýhody a není vhodné takovouto databázi používat na dlouhodobé ukládáni strukturovaných dat. V běžném využití se používá ve spojení s klasickými relačními databázemi jako cache nebo krátkodobé úložiště dat.

V rámci vyvíjeného systému je Redis použit pro ukládání krátkodobých informací pro komunikaci mezi administrační částí a klientskou částí serveru a jako fronta zpráv pro validaci odevzdaných řešení.

5.2.4 MySQL

MySQL je nejpopulárnější open-source relační databázový server. Systém tuto databázi používá jako hlavní úložiště veškerých dat.

5.2.5 Laravel

Laravel je jeden z největších PHP frameworků, který existuje. V základu poskytuje řešení pro autentizaci, napojení a abstrakci nad souborovými systémy, cachovacími systémy, logovacími systémy. V základu rozděluje aplikaci podle MVC modelu, s vlastní databázovou a šablonovací vrstvou. A stejně jako každý větší framework má řešení pro IoC[3] kontejner. Programátor se tedy může věnovat implementování samotné logiky dané aplikace a je odstíněn od řešení běžných věcí, které řeší každá aplikace.

Laravel je v systému využit jako framework pro administrační část systému.

5.2.6 Lumen

Lumen je mikroframework vytvořený stejným autorem jako Laravel a společně s Laravelem sdílí některé ze svých částí. Narozdíl od Laravelu obsahuje jen nejdůležitější části, je tedy nevhodný pro větší aplikace, ale díky tomu je i rychlý a vhodný na tvorbu maličkých aplikací.

V sytému je tedy použit jako framework pro jednotlivé mikroslužby.

5.2.7 Swoole

Swoole je technologie pro umožnění asynchronního chování PHP. Více v podkapitole 5.3.1

5.2.8 JWT - JSON Web Token

JWT neboli JSON Web Token je standardem pro uchování a bezpečné předávání autentizačních dat, které jsou podepsány. Token se dělí na tři části, hlavičku, data a podpis. Všechny části jsou zakódovány pomocí base64. Klient je schopen si data přečíst, ale není schopen je modifikovat tak, aby

[3]Inversion of Control - inverze kontroly, zajištění správy závislostí mezi službami

následně server nepoznal, že nedošlo k modifikaci, na straně serveru dochází k ověření, že podpis sedí k zaslaným datům. [5]

V rámci vyvíjeného systému jsou do tohoto tokenu zakódovány data o uživateli a jeho stavu v rámci aktuální soutěže. Jednotlivé mikroslužby následně nemusí provádět autentizaci na své straně a jen ze zaslaného tokenu ví, že například soutěžící může odevzdávat úlohy v rámci běžícího kola. Toto řešení šetří potřebnou logiku a dotazy na databázi. Příklad rozkódované datové části tokenu ve výpisu 5.1.

```
{
  "iss": "lumen-jwt",
  "sub": 1,
  "iat": 1650968693,
  "exp": 1650972293,
  "id": 1,
  "projectTypeId": 1,
  "name": "John",
  "surname": "Thomas",
  "email": "thomas@gmail.com",
  "team": "Teestovaci tym",
  "teamRole": "captain",
  "teamId": 1,
  "currentRoundStart": "2022-04-25 00:40:00",
  "currentRoundEnd": "2022-04-26 23:30:00",
  "currentRoundId": 1,
  "currentRoundName": "Nominacni kolo"
}
```

Výpis 5.1: Ukázka dat uložených v tokenu

5.2.9 Vue.JS

Vue.JS je frontendový JavaScript framework pro tvorbu interaktivních webových aplikací. Framework se stará o deklarativní zobrazování vnořených komponent a reaktivitu jejich stavu. [13]

Framework je možné využít v různých mírách

- Samostatné reaktivní komponenty v rámci klasické statické stránky
- Kompletní Single Page Aplikace (4.4.5)

- Single Page Aplikace s podporou SSR[4]
- Aplikace mimo webový prohlížeč, zejména mobilní a desktopové aplikace

Základ frameworku se stará pouze o již zmíněné věci. Je tak jednoduchý a dává vývojáři svobodu si aplikaci navrhnout podle sebe. Vzhledem k popularitě má kolem sebe plno dalších knihoven, které doplňují jeho funkci.

Použité knihovny

- **Vue Router** - řeší zobrazování jednotlivých komponent podle URI
- **Pinia** - Úložiště stavu aplikace - standardizuje práci se stavem aplikace, definuje akce, které stav upravují a dává komponentám možnost se napojit na jakoukoliv část stavu, kdy při změně stavu, dojde k automatickému překreslení komponent.
- **PrimeVUE a PrimeIcons** - Sada komponent pro stavbu grafického rozhraní aplikace a sada ikon
- **Axios** - Jednoduchá abstrakce pro tvorbu asynchronních requestů
- **Vite** - Sestavovací nástroj postavený pro rychlejší a optimalizovanější sestavení celé aplikace, nabízející rychle HMR[5]
- **Vite Pages** - Nadstavba nad `Vue Router`, definuje jednotlivé stránky celé aplikace a dovoluje při sestavování aplikace automaticky vytvořit kusy kódu podle stránek, tak aby se nemusela načítat celá aplikace najednou

V rámci vyvíjeného systému je Vue.JS spolu se zmíněnými knihovnami použito na celou klientskou aplikaci.

5.3 Rychlost požadavků na mikroslužby

V rámci celého systému lze očekávat dvě místa, která budou namáhána, a to validační část a klientská část serveru, jak už bylo zmíněno, validační část je

[4]SSR - Server Side Rendering - Vykreslení požadované stránky na serveru, její přenesení a následné zinteraktivnění jako by šlo o klasickou Single Page Aplikaci, výhodami je dostupnost pro vyhledávače a kratší čas načtení stránky

[5]HMR - Hot Module Replacement - při vývoji v prohlížeči vymění pouze část zdrojových kódů aplikace, pro svižnější možnost vývoje

vyřešena asynchronním zpracování úloh. Co se týká klientské části serveru, tak částečné řešení bylo zmíněno v rámci ukládání dat do autentizačního JWT (5.2.8) a také využitím mikroframeworku Lumen (5.2.6). Další možností jak pro klientskou, tak validační část je škálování jednotlivých služeb v rámci Dockeru.

Největší zbývající překážkou je způsob, jakým PHP aplikace fungují a to potřeba s každým požadavkem nastartovat celou aplikaci od začátku, načíst program do paměti, načíst konfiguraci, připojení k potřebným službám. V rámci klasických webových aplikací je toto zanedbatelný výkon, ale i přes to, že mikroslužby využívají mikroframework, je tato operace náročná. Nabízí se tedy otázka, zda neexistuje způsob, jakým aplikaci jednou nastartovat a následně do ní pouštět požadavky za sebou. Tuhle možnost v základu PHP nemá, ale existuje řešení třetí strany Swoole.

5.3.1 Swoole

Swoole nabízí možnost asynchronního chování PHP a udržení běžícího procesu v paměti, během kterého se do procesu posílají jednotlivé požadavky a vrací odpovědi.

Z ukázkových měření (5.2 a 5.3) je vidět, že díky technologii Swoole, je možné se stejnou aplikací v průměru dosáhnou čtyřnásobné propustnosti.

```
wrk -t4 -c10 http://lumen-swoole.local

Running 10s test @ http://lumen-swoole.local
  4 threads and 10 connections
  Thread Stats   Avg      Stdev     Max   +/- Stdev
    Latency     6.41ms    1.56ms  19.71ms   71.32%
    Req/Sec   312.99     28.71   373.00     72.00%
  12469 requests in 10.01s, 3.14MB read
Requests/sec:   1245.79
Transfer/sec:    321.12KB
```

Výpis 5.2: Testovací měření bez Swoole. Převzato z [7]

```
wrk -t4 -c10 http://lumen-swoole.local:1215

Running 10s test @ http://lumen-swoole.local:1215
  4 threads and 10 connections
  Thread Stats   Avg      Stdev     Max   +/- Stdev
    Latency     2.39ms    4.88ms 105.21ms   94.55%
    Req/Sec     1.26k   197.13     1.85k    68.75%
  50248 requests in 10.02s, 10.88MB read
Requests/sec:   5016.94
Transfer/sec:      1.09MB
```

Výpis 5.3: Testovací měření se Swoole. Převzato z [7]

Vzhledem k tomu, že celý proces zůstává v paměti, tak i uložená data v rámci služeb se nemění. To je zároveň výhodou i nevýhodu. Nevýhodou je, že je potřeba si dávat pozor na vývoj celé aplikace a s každým požadavkem vyčistit potřebná data. Zároveň tato skutečnost nabízí výhodu v podobě cache v paměti procesu. Tato možnost využita při vývoji nebyla, ale systém je připraven kdykoliv tuto možnost jednoduše doimplementovat, když by bylo potřeba v rámci mikroslužeb aplikaci zrychlit.

5.4 Struktura aplikace

Celý systém je rozdělen do několika celků podle 5.1.1, tato sekce popisuje funkčnost a další členění v rámci samotných částí.

5.4.1 Administrační část

Administrační část neboli také `core`, je zdrojem pravdy dat a jejich správy. Tato část je rozdělena do několika modulů zmíněných dříve v podkapitole 4.8. Každý modul sídlí ve složce `Modules\{Název modulu}`, jedinou výjimkou je Core module, který sídlí ve složce `App`. Definice databáze se nachází ve složce `database`, která obsahuje databázové migrace sloužící k vytvoření schématu a také scripty na naplnění databáze základními daty.

5.4.1.1 Dělení modulu

Každý z modulů se dělí na jednotlivé části

- **Enums** - Číselníky pro daný modul

- **Forms** - Definice formuláře pro založení a upravování záznamů
- **Http\Controllers** - Kontrolery fungující jako logická vrstva aplikace
- **Jobs** - Asynchronní operace, tuto část neobsahuje každý modul
- **LiweTables** - Definice seznamových tabulek
- **Models** - Modelová vrstva definující entity
- **Policies** - Definice práv k jednotlivým entitám
- **routes** - Směřovací pravidla z URI na akce jednotlivých kontrolerů
- **views** - zobrazovací vrstva

5.4.2 Klientská aplikace

Další částí systémů je VueJS aplikace (5.2.9). Zdrojové kódy aplikace se nachází ve složce `src` a dělí se na několik částí.

- **components** - samostatné komponenty, které se využívají jako části jednotlivých stránek. Každá komponenta obsahuje svou šablonu, styly a vlastní logiku.
- **models** - složka s definicí TypeScript typů, které jsou předávány pomocí API s klientskou částí serveru
- **pages** - jednotlivé stránky aplikace, každá stránka má svou vlastní šablony, styly a logiku podobně jako komponenty, ale má také parametry pro práva k jednotlivým URI
- **scss** - základní styly pro celou aplikaci
- **services** - služby pro komunikaci s API, přijímají a vracejí převážně definované modely
- **store** - definice globálního stavu aplikace a jednotlivých akcí, které jsou schopný stav upravovat

Vstupním bodem celé klientské aplikace je soubor `main.ts`.

5.4.3 Klientská část serveru

Samotná implementační část API není nijak složitá a obsahuje jen pár důležitých částí. Vzhledem k jednoduchosti jednotlivých služeb, zde není využita databázová vrstva a s databází komunikuje přímo kontroler, který se zároveň stará o formátování dat pro odpověď ve formátu JSON.

- **app\Http\Controllers** - Kontrolery pro jednotlivé mikroslužby
- **app\Http\Middleware\Authenticate** - Jde o společnou vrstvu pro kontrolery, které potřebují ověření, JWT. Stará se o jeho parsování a kontrolu podpisu
- **app\Http\Middleware\UserChanged** - Jde o společnou vrstvu pro kontrolery, které můžou způsobit změnu stavu autentizace a automaticky posílá v případě potřeby v hlavičce nový token s obnoveným časem expirace.
- **app\Http\Controllers** - Kontrolery pro jednotlivé mikroslužby
- **app\Jobs\ValidateJob** - Kód pro samotnou validaci řešení
- **app\Modes\CustomUser** - Obálka nad daty dříve ukázaného JWT (5.2.8)
- **routes\web.php** - Směřovací pravidla v rámci mikroslužeb

5.4.4 Sdílené číselníky

Sdílené číselníky jsou společnou částí mezi klientskou částí serveru a administrační částí. Definují několik číselníků pro společné konzistentní používání. Vzhledem k počtu potřebných sdílených číselníků není potřeba dalšího dělení a je tak každý z číselníků přímo ve složce `enums`. Tato složka je při nasazení aplikace připojena do obou částí.

5.4.5 Validátory

Samotná validační část obsahuje bashové skripty, starající se o kompilaci a spuštění validované úlohy. Skript pro kompilaci (`compile.sh`) přijímá jako své parametry jazyk pro kompilaci, a cílovou složku.

Skript pro spuštění (`run.sh`) přijímá jako parametr cestu k souboru se vstupními daty, jazyk úlohy, složku, ve které bylo řešení zkompilované, časový a paměťový limit. Oba tyto skripty mají své jazykové části umístěné ve složce `language\{jazyk}\{compile|run}.sh`

Pro přidání nového jazyku do systému stačí vytvořit složku s klíčem jazyku a implementovat tyto dva scripty.

5.4.6 Konfigurace nasazení aplikace

Tato část obsahuje konfiguraci jednotlivých prostředí pro každou ze služeb v podobě `Dockerfile`, což je specifikace kontejneru pro `Docker` (5.2.1). Obsahuje v krocích způsob, jak takový kontejner připravit. Příklad ve výpisu 5.4

```
FROM phpswoole/swoole:php7.4

RUN \
    pecl update-channels        && \
    pecl install redis          && \
    docker-php-ext-enable redis && \
    docker-php-ext-install pcntl pdo pdo_mysql
RUN set -ex \
    && apt update \
    && apt install fswatch \
    && rm -rf /var/cache/apt/* /tmp/* /usr/share/man /usr/src/php.tar.xz*

ENTRYPOINT ["/code/artisan", "swoole:http", "start"]
```

Výpis 5.4: Přiklad Dockerfile pro mikroslužby

Systém obsahuje několik typů vlastních kontejnerů

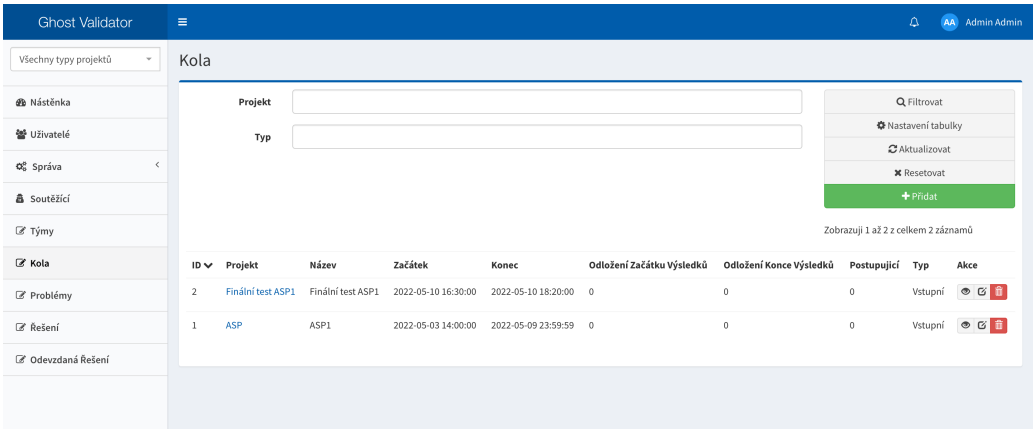
- **checker** - Kontejner pro běh samotné validace, obsahuje sestavovací nástroje pro jednotlivé jazyky
- **core** - Kontejner pro běh administrační části systému
- **nginx** - Kontejner pro běh webserveru, který slouží jako vstupní bod celého systému
- **web_container** - Kontejner s podporou Swoole (5.3.1) pro běh jednotlivých mikroslužeb

Zároveň v této části systému se nachází soubor `docker-compose.yml`, jde o konfiguraci, která popisuje nastavení jednotlivých služeb, jméno dané služby (využívá se následně jako doménové jméno v rámci interní DNS), linkování složek, jaký typ kontejneru se pro službu má využít a také počet replikací jednotlivé služby, popřípadě ještě nastavení automatické obnovy služby.

6 Vizuální průchod aplikací

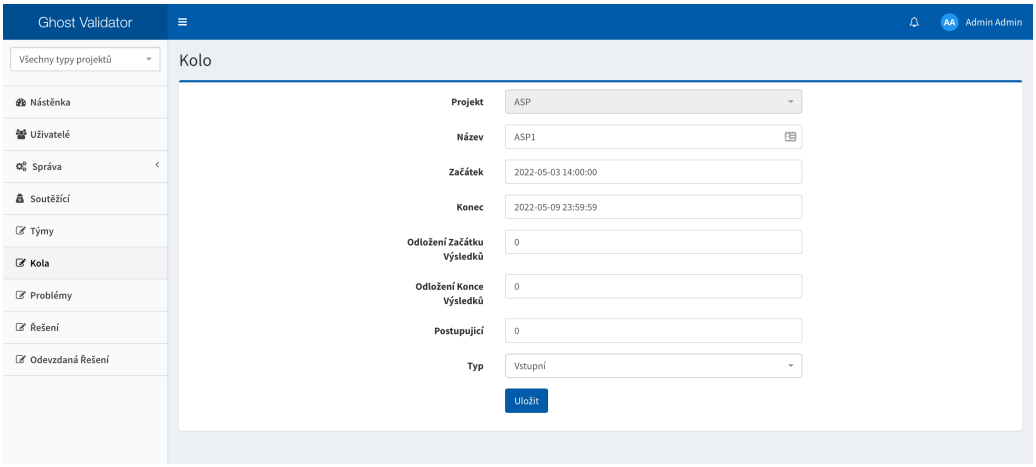
6.1 Tvorba soutěže

Tvorba soutěže začíná vytvořením kola (obrázek 6.1), v levém menu se nachází záložka `Kola` pod kterou se skrývá seznam jednotlivých kol, kolo jde přidat tlačítkem `Přidat` nebo editovat, kliknutím na ikonku s tužkou vedle daného kola.



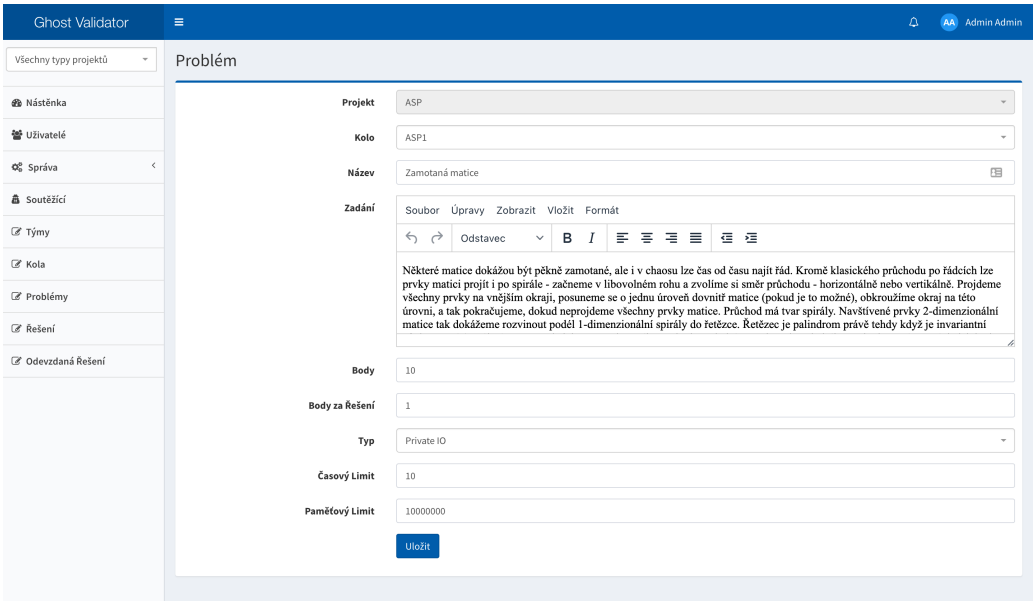
Obrázek 6.1: Seznam jednotlivých kol soutěže

Přidání nebo editace kola má totožný formulář (obrázek 6.2) dovolující specifikovat jeho nastavení.



Obrázek 6.2: Přidání/Úprava kola

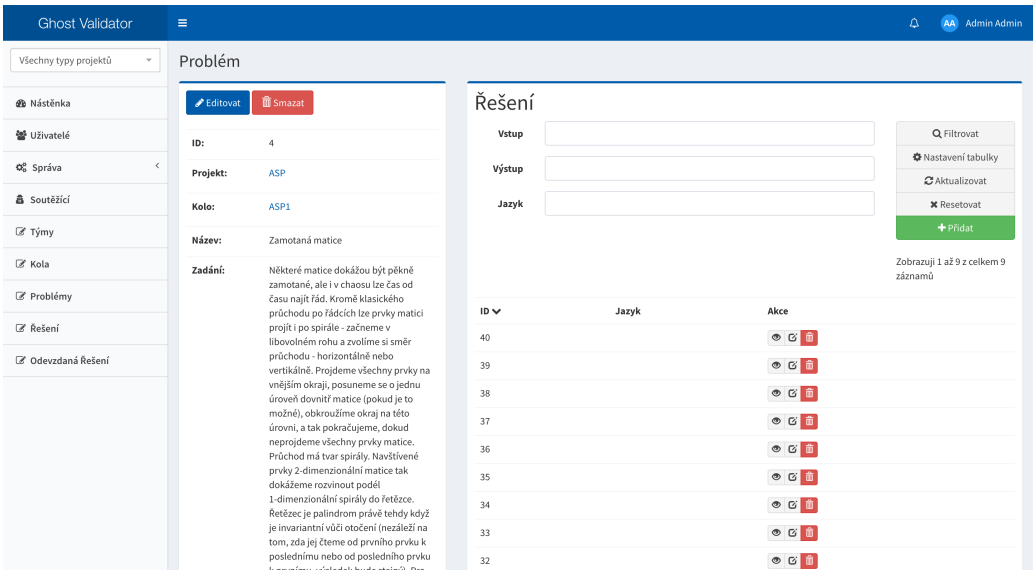
Dalším krokem je vytvoření úloh pro dané kolo, což je možné přes záložku `Problémy`. Seznam vypadá obdobně jako seznam kol. V rámci formuláře (obrázek 6.3) je možné zadat textaci dané úlohy za pomoci WYSIWYG[1] editoru. Nastavit limity a bodování.



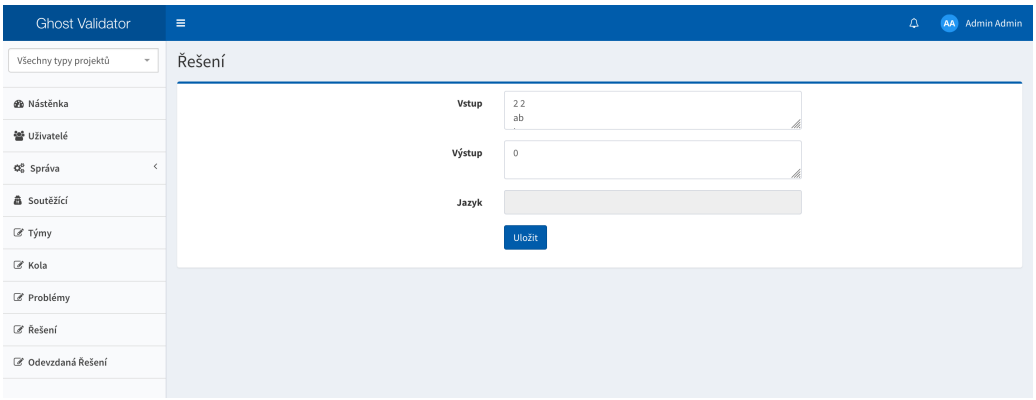
Obrázek 6.3: Přidání/Úprava úlohy

Po uložení dané úlohy dojde k zobrazení detailu (obrázek 6.4), ze kterého je možné přidávat/upravovat/odebírat jednotlivé testovací případy (obrázek 6.5)

[1]WYSIWYG - What You See Is What You Get - podoba editovaného textu je stejná jako výsledná



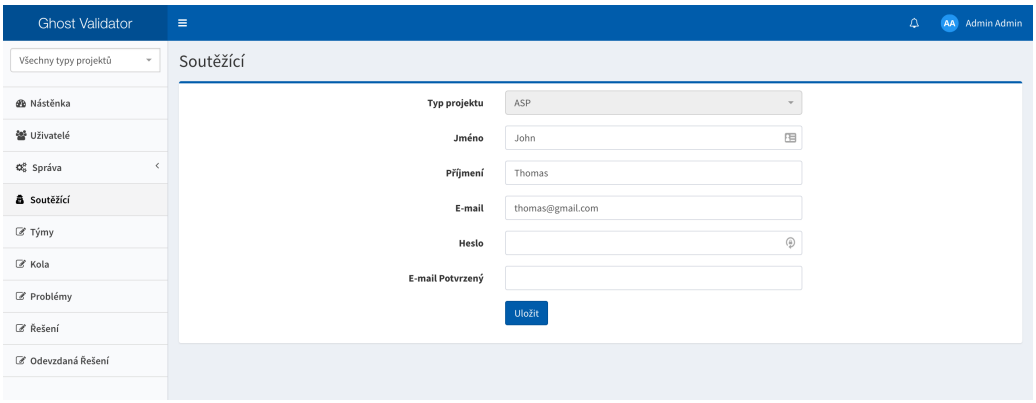
Obrázek 6.4: Detail úlohy



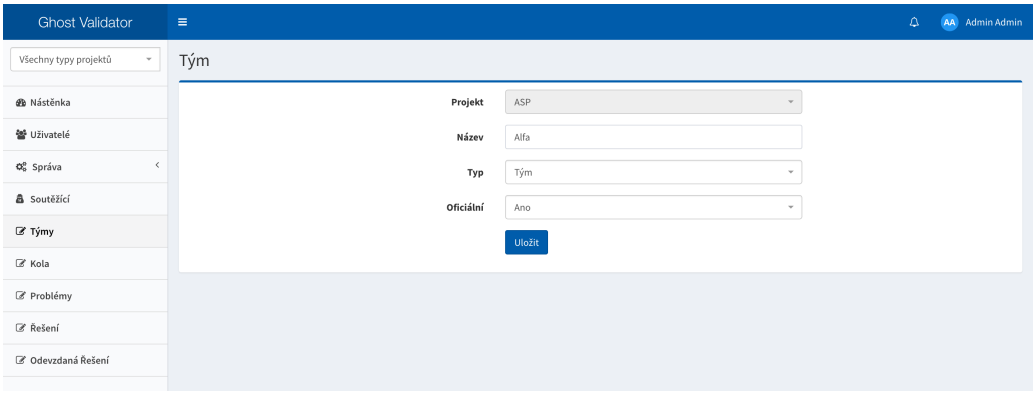
Obrázek 6.5: Přidání/Úprava řešení

Posledním krokem je vytvoření soutěžících a jejich týmů. Soutěžící (obrázek 6.6) je možné spravovat ze záložky `Soutěžící`. Týmy (obrázek 6.7) ze záložky `Týmy`. Po uložení týmu se zobrazí detail (obrázek 6.8), na kterém je možné k týmu přidat soutěžícího a editovat jeho roli v týmu.

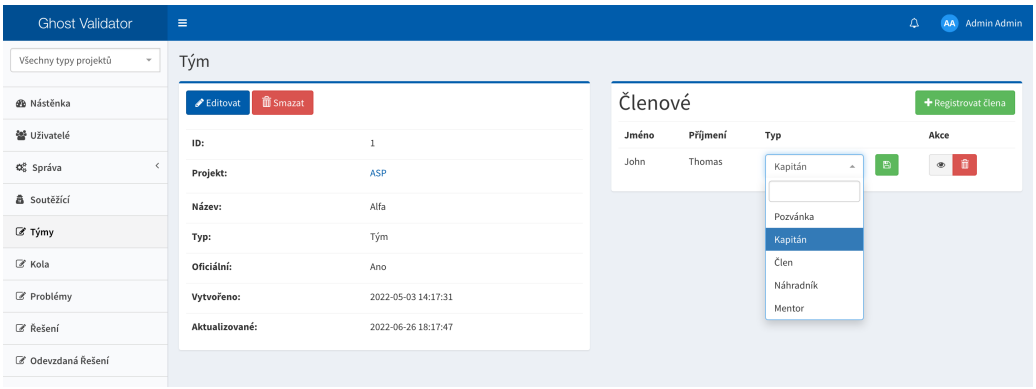
API je částečně připraveno pro práci s uživateli a týmy a rolemi uživatelů, ale aktuální verze klientské aplikace s touto skutečností nepracuje.



Obrázek 6.6: Přidání/Úprava uživatele

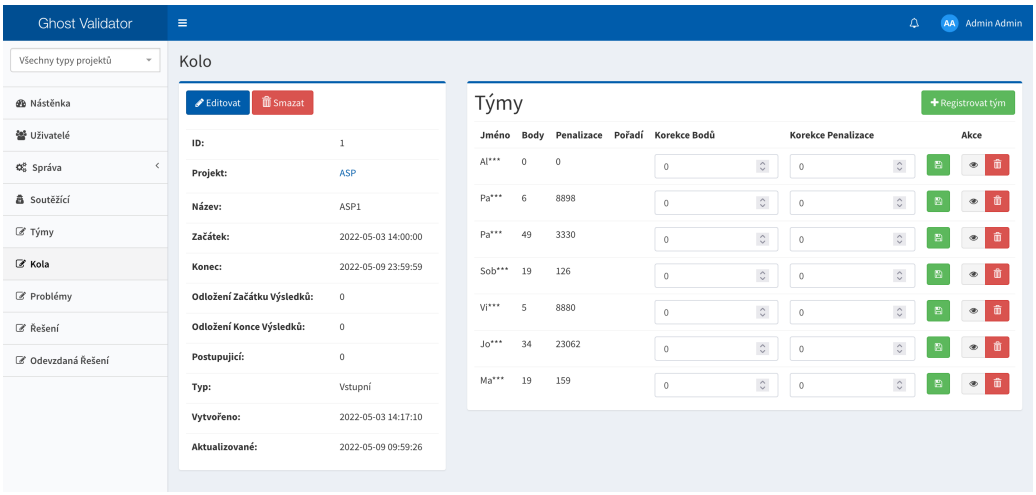


Obrázek 6.7: Přidání/Úprava týmu



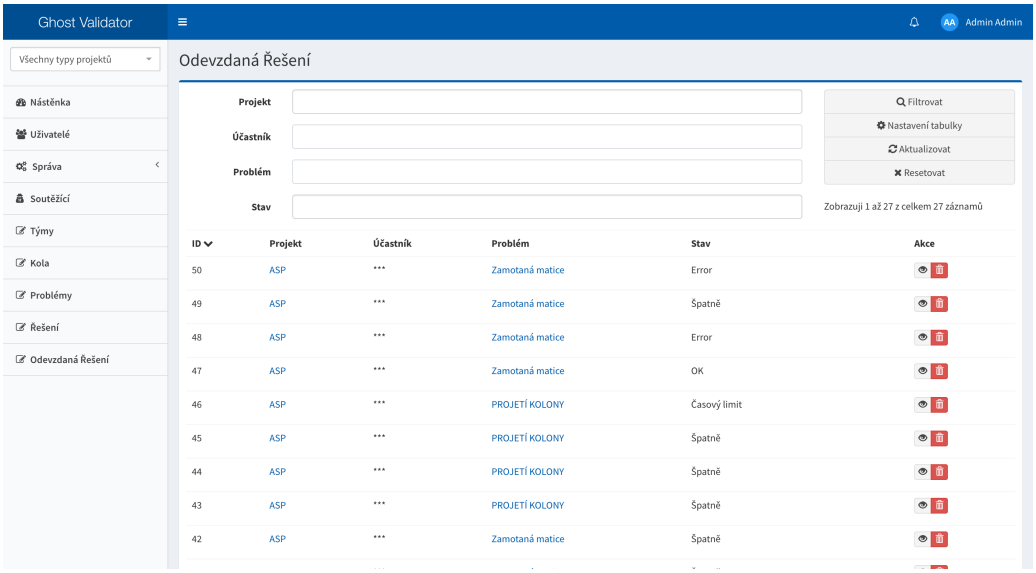
Obrázek 6.8: Detail týmu

Poslední, co je potřeba, je samotné týmy zaregistrovat na kolo soutěže, to je možné z detailu kola (obrázek 6.9), kde je také možnost během soutěže provést korekci bodů a penalizace.



Obrázek 6.9: Detail kola

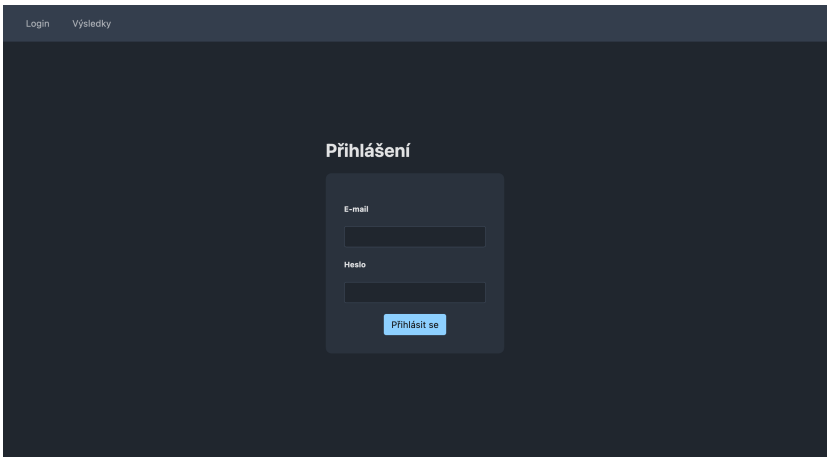
V průběhu soutěže je možné nahlížet na jednotlivé odevzdání řešení a jejich validační stav. (obrázek 6.10)



Obrázek 6.10: Seznam odevzdaných řešení

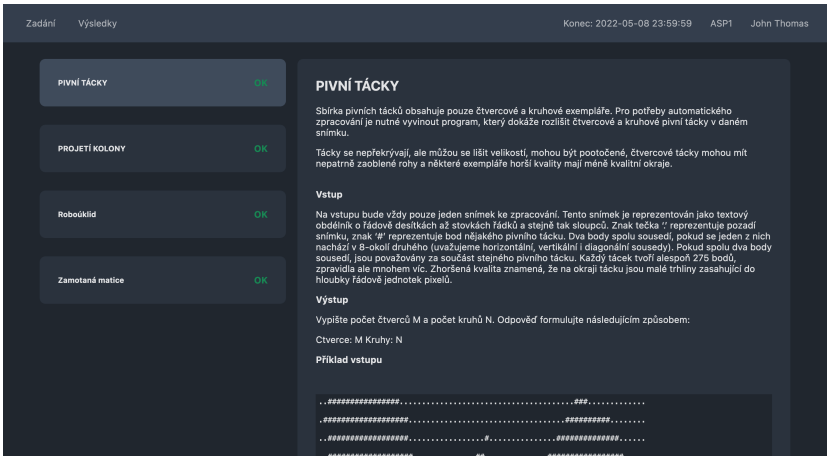
6.2 Soutěžní část

Soutěžní část má daleko jednodušší uživatelské rozhraní a obsahuje pouze několik obrazovek. Soutěžící začne na přihlašovací obrazovce (obrázek 6.11)

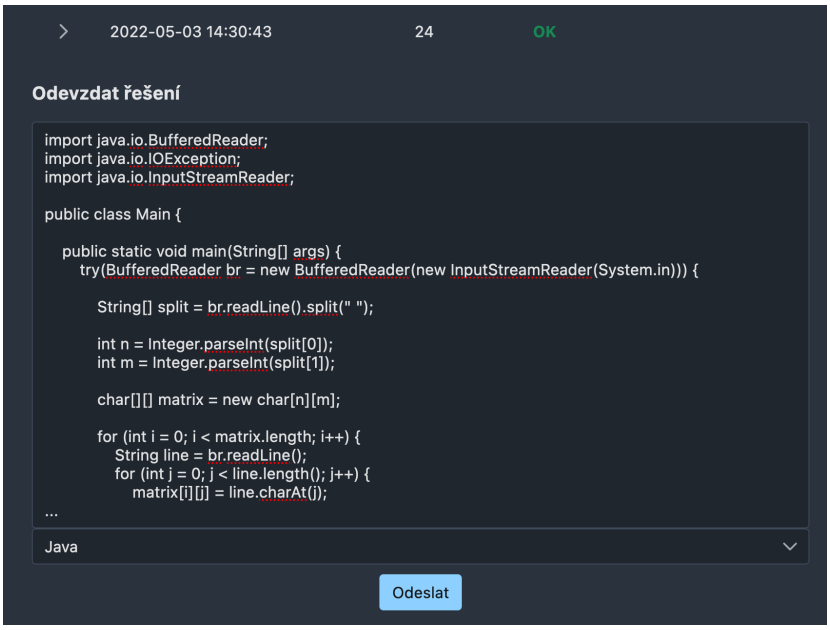


Obrázek 6.11

Nejdůležitější obrazovkou pro uživatele je obrazovka se zadáním (obrázek 6.12), kde pod každým zadání nalezne odesílací formulář pro zdrojový kód (obrázek 6.13). Po odeslání, jsou na stejné obrazovce vidět jednotlivé pokusy o odevzdání (obrázek 6.14), které jdou rozkliknout, a je zde vidět odeslaný kód, popřípadě i chyba z kompilátoru.



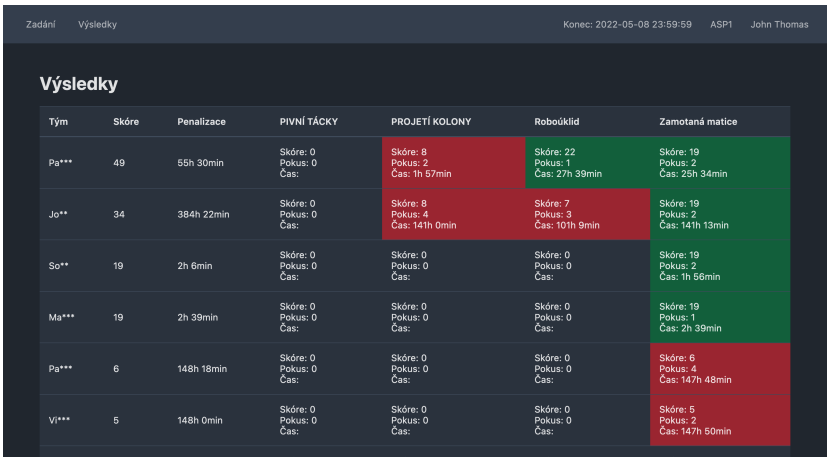
Obrázek 6.12



Obrázek 6.13

Obrázek 6.14

Poslední obrazovkou klientské aplikace je obrazovka s výsledky (obrázek 6.15).


Zadání
Výsledky
Konec: 2022-05-08 23:59:59
ASP1
John Thomas
Výsledky
Tým
Skóre
Penalizace
PIVNÍ TÁCKY
PROJETÍ KOLONY
Roboúklid
Zamotaná matice
Pa***
49
55h 30min
Skóre: 0
Pokus: 0
Čas:
Skóre: 8
Pokus: 2
Čas: 1h 57min
Skóre: 22
Pokus: 1
Čas: 27h 39min
Skóre: 19
Pokus: 2
Čas: 25h 34min
Jo**
34
384h 22min
Skóre: 0
Pokus: 0
Čas:
Skóre: 8
Pokus: 4
Čas: 141h 0min
Skóre: 7
Pokus: 3
Čas: 101h 9min
Skóre: 19
Pokus: 2
Čas: 141h 13min
So**
19
2h 6min
Skóre: 0
Pokus: 0
Čas:
Skóre: 0
Pokus: 0
Čas:
Skóre: 0
Pokus: 0
Čas:
Skóre: 19
Pokus: 2
Čas: 1h 56min
Ma***
19
2h 39min
Skóre: 0
Pokus: 0
Čas:
Skóre: 0
Pokus: 0
Čas:
Skóre: 0
Pokus: 0
Čas:
Skóre: 19
Pokus: 1
Čas: 2h 39min
Pa***
6
148h 18min
Skóre: 0
Pokus: 0
Čas:
Skóre: 0
Pokus: 0
Čas:
Skóre: 0
Pokus: 0
Čas:
Skóre: 6
Pokus: 4
Čas: 147h 48min
Vi***
5
148h 0min
Skóre: 0
Pokus: 0
Čas:
Skóre: 0
Pokus: 0
Čas:
Skóre: 0
Pokus: 0
Čas:
Skóre: 5
Pokus: 2
Čas: 147h 50min

Obrázek 6.15

7 Ověření systému v praxi

Během samotného vývoje byl systém manuálně testován po jednotlivých částech i jako celek.

7.1 Nasazení systému pro soutěž v rámci ASP1

Pro otestování v praxi byla využita hodina v rámci předmětu ASP1[1].

7.1.1 Nasazení

Připravený systém, který běžel na lokálním stroji, bylo nejprve potřeba nasadit na veřejný server. Pro toto nasazení byl využit server běžící na Microsoft Azure. Vzhledem k běžícím službám na daném serveru bylo potřeba celý systém schovat za proxy, tak aby služby existující na daném serveru fungovaly i vedle validačního systému. Prvním krokem bylo upravení doménových jmen v rámci nasazovací konfigurace v souboru `orchestration/nginx/nginx.conf`, a následná úprava použitého portu v rámci `orchestration/docker-compose.yml`. A vytvoření proxy konfigurace v rámci serveru.

Zdálo se, že tohle je jediné, co je pro fungování potřeba změnit, opak byl ale pravdou. Po debuggingu se přišlo na to, že se přes proxy nepředávají správně hlavičky. To bylo napraveno přidáním konfigurace proxy, viz výpis 7.1

```
proxy_set_header   X-Real-IP $remote_addr;
proxy_set_header Host $host;
proxy_pass_request_headers      on;
```

Výpis 7.1: Přidaná konfigurace hlaviček

Další problém nastal v rámci CORS[2] ochrany. Vzhledem k tomu, že klientská aplikace a klientská část serveru, byla navržena tak aby běžela na různých subdoménách, je potřeba, aby klientská část serveru správně reagovala na `OPTIONS`[3] dotaz a v hlavičkách vrátila povolenou doménu ze které

[1] ASP1 - Algoritmy a soutěžní programování 1

[2] CORS - Cross-Origin Resource Sharing - Moderní ochrana prohlížečů proti vyvolání dotazů z nepovolených doménových jmen

[3] Jde o standartní HTTP(S) požadavek metodou `OPTIONS` na danou URL bez jakéhokoliv těla

je možné dotazy směřovat. Ač systém na dané požadavky reagovat umí, z nějakého, v té chvíli nezjistitelného důvodu, proxy na serveru na tyto požadavky reagovat neuměla a nedokázala je správně předávat do systému. Pro vyřešení tohoto problému bylo nutné unifikovat doménová jména mezi klientskou aplikací a klientskou částí serveru.

7.1.2 Testování

Před samotnou soutěží bylo provedeno nahraní úloh do systému. Při nahrávání testovacích sad došlo k odhalení chyby(omezení) velikosti datového typu v rámci databáze, kde v tabulce `programming_problem_solutions`, byly sloupečky `input` a `output` nastaveny na datový typ `text`, který dokáže v sobě uložit data pouze do velikosti `64 kilobytů`, sloupeček byl tedy změněn na typ `longtext` podporující až `4 gigabyty`. Po této úpravě již proběhlo nahrání úloh bez dalšího problému.

Následovalo testování samotné validace. Korektní řešení byla otestována s úspěchem. Stejného úspěchu se dočkala i řešení nesplňující časový nebo paměťový limit v rámci běžícího procesu.

Problém nastal v případě, že nevalidní řešení generovalo na výstup násobný výstup oproti očekávanému výstupu, kdy daný výstup se načítá celý najednou do paměti validátoru, kam se kvůli paměťovému omezení serveru nevešel. Kontejner s validátorem pak celý havaroval a systém tak přestal validovat i jakékoliv další odeslané řešení. Správným řešením je proudové zpracování výstupu na straně validátoru, ale tato úprava byla před soutěží příliš riskantní. Vzhledem k návrhu systému existovalo jednoduché rychlé řešení pokrývající potenciální případy i jiných možností pádu a to službu validátoru pustit ve více replikách, tak pokud jedna replika spadla, mohlo dojít k odbavení validace ostatními replikami.

Během reálného testování došlo také k nálezu minoritních designových chyb, například se zobrazováním odeslaného kódu v rámci administrace, které byly na místě odstraněny.

7.2 Testování mimo soutěž

Důležitou částí systému je jeho robustnost validace řešení. To je tedy hlavním záměrem testování. Během reálného testování jsme objevili případ, kdy validační systém padá, to je tedy jeden ze scénářů, které je potřeba po opravě otestovat. Další scénář, který je potřeba otestovat, je zahlcení validačního systému dlouho běžícími úlohami. Všechny scénáře byly spojeny do jednoho testování, kdy za pomocí scriptu (Výpis 7.2) se do systému poslalo 1000

jednotlivých odevzdání, kde každé odevzdání mělo náhodně jeden z tří typů řešení. Prvním bylo již zmíněné zahlcení výstupu, druhým časově náročný script, který musí být zastaven časovým limitem a posledním řešením bylo správné řešení testovací úlohy.

```ruby
#!/usr/bin/env ruby
require 'uri'
require 'net/http'

url = URI("http://backend.validator.loc/1/problems/1")

http = Net::HTTP.new(url.host, url.port)
@solutions = [
        "{\"solution\": \"...\",\"language\": \"python3\"}",
        ...
]

1000.times do
        request = Net::HTTP::Post.new(url)
        request["Content-Type"] = 'application/json'
        request["Authorization"] = 'Bearer eyJ0eXAiO...'
        request.body = @solutions.sample
        http.request(request)
end
```

Výpis 7.2: Testovací script

Testování běželo na pěti replikách validačního systému. Na zobrazení odevzdaných řešení bylo vidět, jak se postupně mění stav z `Čeká na validaci` na `Probíhá validace` (kterých bylo očekávaně pokaždé 5) a následně na správný výsledný stav. Všech 1000 odevzdání se do systému zapsalo během 16 vteřin a jejich validace proběhla kompletně bez chyb během 26 minut a 45 vteřin. Testování ukázalo, že systém je stabilní a zvládne zpracovat i velký objem testovacích dat. Ale vzhledem k časové náročnosti validace je potřeba připravit pro validační službu na velké soutěže silný server, který bude mít více replik.

8 Závěr

Na základě analýzy existujících řešení byly nadefinovány požadavky systému. Podle těchto požadavků byl vyvinut funkční systém pro pořádání soutěží za pomocí mikroservisní architektury.

Systém byl úspěšně rozdělen do několika částí obsahující nutné minimum závislostí mezi sebou. V rámci práce byla kompletně implementována možnost validovat úlohy s fixními skrytými vstupy a výstupy. Datová vrstva byla navržena i pro budoucí podporu dynamické validace.

V rámci klientské aplikace bylo dosaženo vytvoření jednoduchého moderního rozhraní. Aplikace provádí vykreslování na straně klienta a server se stará pouze o předávání dat. To vede k velmi rychlé reaktivitě na straně klienta a také ke zlepšení výkonu na straně serveru, který je tak využit na samotnou validaci úloh.

Systém byl experimentálně vyzkoušen na soutěži v rámci předmětu ASP1, kde se ukázala chyba v samotné validaci. Ale ta samá chyba zároveň ukázala, jakou má výhodu zvolená mikroservisní architektura. Službu, která občas padala v průběhu ostrého nasazení, stačilo pouze pustit v několika replikách a uživatelé v případě pádu jedné z replik mohli bez problémů systém používat dál. Chyba tak mohla být plně opravena až po skončení soutěže.

Zároveň bylo na systému otestované zahlcení velkým množstvím odeslaných řešení úloh v různých limitních podobách. Bylo ověřeno, že systém je dostatečně robustní i na korektní validaci takového objemu práce a že dokáže v rámci replikace služby zpracovávat jednotlivé úlohy paralelně.

Výsledkem práce je robustní systém založený na moderních technologiích, který dokáže konkurovat starším zavedeným řešením pro realizaci programátorských soutěží a zároveň odstraňuje jejich zásadní nedostatky. Navržené řešení lze dále snadno rozvíjet pro nové funkcionality a postupně vylepšovat uživatelské rozhraní.

Literatura

[1] *Client-Server Architecture* [online]. Teach Computer Science, 2021. [cit. 2022/06/20]. Dostupné z: https://teachcomputerscience.com/client-server-architecture/.

[2] *What is Docker* [online]. Amazon, 2022. [cit. 2022/06/20]. Dostupné z: https://aws.amazon.com/docker/.

[3] *Poor man's load balancing with Docker* [online]. Medium, 2016. [cit. 2016/05/16]. Dostupné z: https://medium.com/@lherrera/poor-mans-load-balancing-with-docker-2be014983e5.

[4] Gupta, L. *What is REST* [online]. Restfulapi.net, 2022. [cit. 2022/06/20]. Dostupné z: https://restfulapi.net.

[5] Jones, M. *JSON Web Token (JWT)* [online]. Internet Engineering Task Force, 2015. [cit. 2022/06/20]. Dostupné z: https://datatracker.ietf.org/doc/html/rfc7519.

[6] *Kasiopea* [online]. Matematicko-fyzikální fakulta Univerzity Karlovy, 2022. [cit. 2022/06/20]. Dostupné z: https://kasiopea.matfyz.cz/pravidla/.

[7] *Laravel-Swoole* [online]. GitHub, 2019. [cit. 2022/06/20]. Dostupné z: https://github.com/swooletw/laravel-swoole.

[8] Maňák, M. *KIV ZČU Plzeň – Soutěž ICPC* [online]. Katedra informatiky a výpočetní techniky ZČU Plzeň, 2022. [cit. 2022/05/02]. Podpora soutěžních aktivit na webu Katedry informatiky a výpočetní techniky Západočeské univerzity v Plzni. Dostupné z: http://www.kiv.zcu.cz/cz/pro-studenty/souteze/acm-icpc/acm-icpc.html.

[9] *Message Queues* [online]. Amazon, NA. [cit. 2022/06/20]. Dostupné z: https://aws.amazon.com/message-queue/.

[10] *Matematická Olympiáda kategorie P* [online]. Matematicko-fyzikální fakulta Univerzity Karlovy, 2022. [cit. 2022/06/20]. Dostupné z: https://mo.mff.cuni.cz/p/osoutezi.html.

[11] *PilsProg* [online]. Katedra informatiky a výpočetní techniky ZČU Plzeň, 2022. [cit. 2022/06/20]. Dostupné z: https://pilsprog.fav.zcu.cz/index.php?language=cz&page=information.

[12] Skólski, P. *Single-page application vs. multiple-page application* [online]. neoteric, 2016. [cit. 2022/06/20]. Dostupné z: https://neoteric.eu/blog/single-page-application-vs-multiple-page-application/.

[13] *What is Vue?* [online]. VueJS, 2022. [cit. 2022/06/20]. Dostupné z: https://vuejs.org/guide/introduction.html.

[14] Wilson, C. *How to Design a Web Application: Software Architecture 101* [online]. Matematicko-fyzikální fakulta Univerzity Karlovy, 2020. [cit. 2022/06/20]. Dostupné z: https://www.educative.io/blog/how-to-design-a-web-application-software-architecture-101#mono.

Přílohy

A Spuštění systému

A.1 Požadavky

- Docker - `https://www.docker.com`
- PHP v7.4
- NodeJS v14 a NPM v6 - `https://nodejs.org/en/`
- Composer - `https://getcomposer.org`

A.2 Samotné spuštění

```
cd core
composer install
npm install
npm run prod
cp .env.example .env

cd ../backend
cp .env.example .env

cd ../frontend
npm install
npm run build

cd ../orchstration
./../core/vendor/bin/sail up

../core/vendor/bin/sail artisan key:generate
../core/vendor/bin/sail artisan migrate --seed
```