FAKULTA APLIKOVANÝCH VĚD
ZÁPADOČESKÉ UNIVERZITY
V PLZNI

KATEDRA INFORMATIKY
A VÝPOČETNÍ TECHNIKY

Diplomová práce

# Webový portál umožňující izolované soutěžní simulace

Tomáš Ott

PLZEŇ

2024

FAKULTA APLIKOVANÝCH VĚD
ZÁPADOČESKÉ UNIVERZITY
V PLZNI

KATEDRA INFORMATIKY
A VÝPOČETNÍ TECHNIKY

## Diplomová práce

# Webový portál umožňující izolované soutěžní simulace

Bc. Tomáš Ott

**Vedoucí práce**
Ing. Martin Úbl

**PLZEŇ** **2024**

**Citace v seznamu literatury:**
OTT, Tomáš. *Webový portál umožňující izolované soutěžní simulace*. Plzeň, 2024. Diplomová práce. Západočeská univerzita v Plzni, Fakulta aplikovaných věd, Katedra informatiky a výpočetní techniky. Vedoucí práce Ing. Martin Úbl.

ZÁPADOČESKÁ UNIVERZITA V PLZNI
Fakulta aplikovaných věd
Akademický rok: 2023/2024

# ZADÁNÍ DIPLOMOVÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

| | |
|---|---|
| Jméno a příjmení: | **Bc. Tomáš OTT** |
| Osobní číslo: | **A21N0062P** |
| Studijní program: | **N3902 Inženýrská informatika** |
| Studijní obor: | **Softwarové inženýrství** |
| Téma práce: | **Webový portál umožňující izolované soutěžní simulace** |
| Zadávající katedra: | **Katedra informatiky a výpočetní techniky** |

## Zásady pro vypracování

1. Seznamte se s programátorským rozhraním systému SmartCGMS, s nemocí diabetes mellitus a soutěží Blood Glucose Prediction Challenge (BGLP).
2. Analyzujte možnosti bezpečného a izolovaného vzdáleného sestavení a vyhodnocení zdrojových kódů vzhledem k využití skrze webové rozhraní.
3. Navrhněte webový portál, který umožní správcům definovat komplexní testovací scénář, a běžným uživatelům umožní nahrávat zdrojové kódy entit systému SmartCGMS, bude je sestavovat a následně je vyhodnotí v izolovaném prostředí v rámci definovaného scénáře.
4. Implementujte navržené řešení, zejména s důrazem na efektivitu kódu, bezpečnost a izolovanost vyhodnocení.
5. Ověřte implementované řešení z hlediska funkčnosti a výkonu.
6. Zhodnoťte dosažené výsledky.

Rozsah diplomové práce: **doporuč. 50 s. původního textu**
Rozsah grafických prací: **dle potřeby**
Forma zpracování diplomové práce: **tištěná/elektronická**

Seznam doporučené literatury:

dodá vedoucí diplomové práce

Vedoucí diplomové práce: **Ing. Martin Úbl**
Katedra informatiky a výpočetní techniky

Datum zadání diplomové práce: **8. září 2023**
Termín odevzdání diplomové práce: **16. května 2024**

L.S.

**Doc. Ing. Miloš Železný, Ph.D.**
děkan

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

V Plzni dne 11. října 2023

# Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně a výhradně s použitím citovaných pramenů, literatury a dalších odborných zdrojů. Tato práce nebyla využita k získání jiného nebo stejného akademického titulu.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona v platném znění, zejména skutečnost, že Západočeská univerzita v Plzni má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona.

V Plzni dne 16. května 2024

. . . . . .
Tomáš Ott

V textu jsou použity názvy produktů, technologií, služeb, aplikací, společností apod., které mohou být ochrannými známkami nebo registrovanými ochrannými známkami příslušných vlastníků.

# Abstrakt

Tato diplomová práce je zaměřena na navržení a sestavení soutěžního systému, který umožňuje definování komplexních testovacích scénářů a jejich správu skrze webové rozhraní. Rozhraní systému dovoluje uživatelům vkládat řešení předem stanovených problémů z oblasti onemocnění diabetes mellitus v podobě zdrojových kódů entit systému SmartCGMS. Tyto entity jsou přeloženy, spuštěny a následně vyhodnoceny podle definovaného scénáře a to v distribuovaném prostředí s důrazem na bezpečnost a izolovanost vyhodnocení. Celý soutěžní systém v podobě oddělené ASP.NET aplikace je integrován do již existujícího webového portálu diabetes.zcu.cz, který je společně se systémem SmartCGMS vyvíjen na Katedře informatiky a výpočetní techniky.

# Abstract

This master's thesis is focused on design and implementation of competition system that allows its administration though web interface and defining complex testing scenarios. Interface of the system then allows users to upload diabetes mellitus related solutions in form of SmartCGMS entity source codes. These entities are compiled, executed a subsequently evaluated via the defined scenario in distributed environment with emphasis on its security and isolation. This competition system is implemented in a form of detached ASP.NET application and integrated to existing web diabetes.zcu.cz which is together with SmartCGMS developed by Department of Computer Science and Engineering.

## Klíčová slova

diabetes mellitus • SmartCGMS • soutěžní systém • distribuovaný systém • Docker Swarm • bezpečnost • izolovanost • ASP.NET • Nette

# Poděkování

Děkuji panu Ing. Martinovi Úblovi za odborné vedení diplomové práce a za velmi cenné rady při jejím vypracování.

*Tomáš Ott,*
(květen 2024)

# Obsah

# Úvod 1

Soutěže v oblasti informačních systémů představují nejen možnost ukázky technických dovedností a znalostí, ale také rozvíjí kreativitu, technické myšlení a mohou být i v zaměřené oblasti průlomové. Soutěžní programování se koná i v medicínské oblasti. Jelikož se jedná o poměrně náročné a komplexní prostředí, nemusí tak být vždy zajištěno, že se soutěž koná dostatečně transparentně a spravedlivě. Jako pokus o zajištění alespoň části těchto vlastností by mohl být použit soutěžní systém v podobě přístupného a přívětivého webového portálu.

Cílem této práce je přivést do kombinované oblasti soutěžního programování a onemocnění diabetes mellitus nový férový systém pro vyhodnocení soutěží, který umožní soutěžícím ověření funkčnosti svých navržených modelů nad reálnou datovou sadou pacientů. Tato datová sada, schována za vrstvou abstrakce poskytované novým soutěžním systémem, by tak nemusela u každého soutěžícího podléhat dlouhému procesu schvalování. Absence takového procesu by umožnila přístup do oblasti i širší veřejnosti a unifikovaný vyhodnocovací systém by zároveň umožnil zpětné sestavení modelů i po skončení soutěže. V budoucnu by takto implementovaný soutěžní systém mohl být stanoven jako standard při vytváření soutěží tohoto typu.

Před návrhem a implementací samotného soutěžního systému je však potřeba se seznámit jak s obecnou problematikou soutěžního programování, tak i tou specializovanou v oblasti onemocnění diabetes mellitus (dále kapitola 2). Na problematiku tohoto onemocnění navážeme v kapitole 3 cílovým systémem SmartCGMS, který se onemocněním zabývá. V neposlední řade jsou v kapitole 4 na základě získaných znalostí existujících soutěžních systémů analyzovány přístupy a metody vzdáleného a izolovaného sestavení zdrojových kódů.

# Soutěže a soutěžní systémy

# 2

Soutěžení a rivalita jsou neodmyslitelnou součástí naší historie a do dnešního dne jsou patrné v širokém spektru oblastí lidské činnosti. Patří mezi ně například sport, umění, zdravotnictví, ale i počítačové vědy a technologie. V každém odvětví si lze rivalitu či soutěživost představit jako platformu pro ověřování a porovnávání schopností a dovedností, ale také jako prostředek pro motivaci účastníků soutěže k dosažení těch nejlepších výsledků. Soutěže slouží tedy jako prostředek dosažení konkrétních cílů a v nejlepším případě i naleznutí optimálního řešení. Pokrok a inovace, kterou soutěže podněcují, jsou důležité jak pro pokrok v daném oboru, tak mohou být užitečné pro lidstvo samotné. [1]

V oblasti počítačových věd se soutěže staly skvělým nástrojem pro testování, porovnávání algoritmů, metod a technologií při řešení konkrétních problémů. Aby ale tyto soutěže zachovaly svoji důvěryhodnost a efektivitu, musí být jejich vyhodnocování spravedlivé, transparentní a pro všechny účastníky shodné. Otevřenost, jako tomu je i v jiných odvětvích, je důležitá především v oblasti postupu vyhodnocování a to z toho důvodu, aby bylo postupováno objektivně a bez známek jakékoliv diskriminace.

## 2.1 Soutěžní programování

Soutěžní programování je sportovní odvětví s obecným cílem naleznutí nejefektivnějších a co nejvíce výkonných postupů a datových struktur pro specifické algoritmicky nebo matematické problémy. Soutěžící hledají řešení zadaného problému v omezeném časovém intervalu, přičemž samotné zadání soutěže bývá často ve formě několika různých úloh seřezených na základě obtížnosti. Cílem soutěžícího je vyřešit co největší množství různě obtížných úkolů v co nejkratším čase. Existuje několik světových organizací zabývajících se soutěžním programováním. A právě jedním kvalitně dokumentovaným zástupcem se bude tato práce zabývat v následující sekci. Poskytne nám tak přehled o tom, čím se soutěže obecně zabývají, jak jsou soutěžní systémy budovány a jakým způsobem přistupují k vyhodnocování řešení.

, jejímž kvalitně dokumentovaným zástupcem se bude tato práce v následující sekci zabývat a poskytne nám tak přehled o tom čím se soutěže obecně zabývají, jak jsou soutěžní systémy budovány a jakým způsobem přistupují k vyhodnocování řešení [2].

### 2.1.1 ICPC

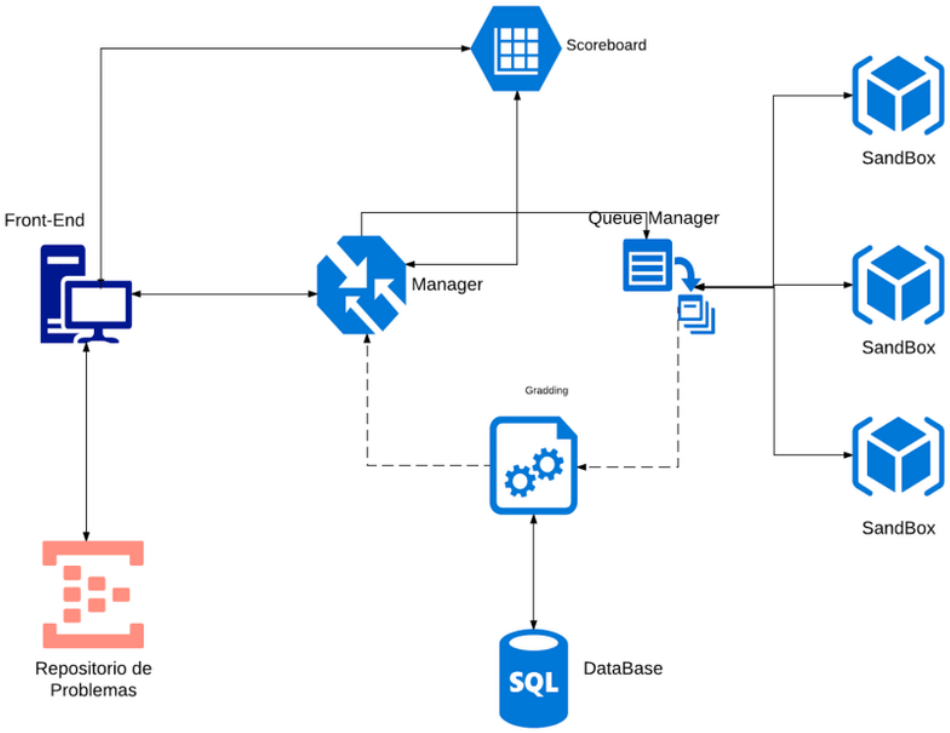
International Collegiate Programming Contest (ICPC) je programátorská soutěž pro vysokoškolské studenty zaměřená především na algoritmizaci. Soutěžící jsou rozděleni do skupin po třech, reprezentující svoji univerzitu za cílem vyřešit co nejvíce úloh v co nejkratším čase a ověřit si tak svojí spolupráci, kreativitu, inovaci a schopnost pracovat pod tlakem. Účastníci si mohou vybrat z několika programovacích jazyků jako je C, C++, Python a nebo Java. Samotná soutěž je rozdělena na několik úrovní - od lokálních a regionální soutěží, až po regionální a světové mistrovství. Zadané úlohy jsou uměle nasazeny do reálných a světových problémů a převedeny tak, aby se daly vyřešit jedním nebo kombinací algoritmických postupů [3].

#### 2.1.1.1 Principy vyhodnocování soutěže

Proces sestavení řešení a jeho následného vyhodnocení soutěže ICPC je z architektonické stránky zveřejněn pro výzkumné účely. Skládá se z několika komponent splňující vlastnosti Service Oriented Architektury (SOA)[4]. Mezi ně patří uživatelské rozhraní, manažer, vyhodnocovatel, výsledkové listiny a sandbox (viz obrázek č.2.1).

**Sandbox.** Tato komponenta spouští zadané řešení úlohy účastníka v kontrolovaném a bezpečném prostředí. Vyhodnocování je limitováno časem spuštění, maximální spotřebou paměti a počtem systémových volání. Nejdůležitější vlastností sandbox komponenty je zajištění stavu, kdy spuštěním přeloženého řešení nedojde k poškození systému nebo hostitelského počítače, dále musí aplikovat časové, paměťové a síťové omezení. Toho soutěž ICPC dosahuje dvěma způsoby. Prvním ze způsobů je využití podprocesů, pomocí kterých dosahují aplikační izolace a docker kontejnerů (dále v sekci 4.2.4). Je definováno několik typů sandboxů, které se inicializují podle typu použitého programovacího jazyka jako je C, C++, Python a nebo Java. Tato komponenta je architektonicky oddělená od ostatních především z důvodu velké proměnlivosti a nepředvídatelnosti chování přeloženého řešení, která mimo jiné otevírá i možnost externího útoku [4].

**Manažer fronty.** Tento modul, jak již název napovídá, je zodpovědný za vytváření nových sandbox komponent na základě vstupu od soutěžících. Zároveň si drží in-




Obrázek 2.1: Architektura evaluačního systému soutěže ACM ICPC [4]

formaci o počtu dostupných sandbox zařízení a určuje podle zvoleného jazyka o jaký typ se bude jednat. Hotové řešení s dostupným výstupem je pak dále předáno vyhodnocovači [4].

**Vyhodnocovač.** Tento modul je zodpovědný za vyhodnocení vygenerovaného výstupu chodu přeloženého řešení s předpokládaným výstupem. Výsledek vyhodnocení je zařazen do jednoho ze stavů, seznam všech možných stavů je znázorněn v tabulce 2.1. Zároveň vyhodnocuje jestli došlo ke kompilačním chybám, které společně se zmíněnými stavy jsou prezentovány soutěžícímu a uloženy do databáze [4].

**Centrální manažer.** Jedná se o centrální modul celé architektury, který složí především pro přeposílání zprávy mezi jednotlivými komponentami systému. Konkrétně přijímá zadané řešení od soutěžících, předává je následně manažeru fronty a od vyhodnocovače následně získá upozornění, že došlo k vyhodnocení. O výsledku celého procesu pak notifikuje výsledkovou listinu a i uživatelské rozhraní.

| | |
|---|---|
| CE | Nastala chyba překladače při překladu |
| TLE | Byl překročen časový limit řešení. Pravděpodobně z důvodu neoptimálního řešení nebo je jeho komplexita moc vysoká |
| MLE | Pamětový limit byl překročen. |
| WA | Odpověd nesprávná. Vygenerový výstup neodpovídá požadovanému výstupu. |
| AC | Řešení bylo přijato. Byly splněny všechny limity |
| RTE | Chyba za běhu přeloženého řešení. |

Tabulka 2.1: Možné stavy vyhodnocení zadaného řešení v systému ICPC [4]

**Výsledková listina.** Modul slouží pro zobrazování výsledků všech zadaných řešení a účastníků v sestupném pořadí. Pro příklad si můžeme uvést obrázek 2.2, který obsahuje výsledkovou listinu soutěže CONEISC z roku 2016, jež je založená na ACM ICPC stylu vyhodnocování soutěží. Aktivita soutěžícího je ohodnocena ve třech dimenzích, které jsou konkrétně celkový počet vyřešených úkolů, čas potřebný pro vyřešení a počet pokusů, kde za každý neúspěšný pokus je udělena penalizace 20 minut [4].

| Rank | Equipo | A | B | C | D | E | F | G | H | Solved | Time |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | UPC - INSANECodeOverride | 1 | 2 | 1 | 5 | 1 | 1 | | 4 | 6 | 498 |
| | UNI - 1000KB | 6 | 8 | 1 | 4 | 1 | 1 | 2 | 1 | 6 | 537 |
| | UNI - 2000KB | | 8 | 1 | 2 | 1 | 1 | 5 | 1 | 5 | 249 |
| 4 | UNI - MDEVELOP | 3 | 2 | 1 | | 1 | 1 | | 1 | 5 | 588 |
| 5 | UNI - AhoritaNoJoven | 1 | 13 | 1 | 11 | 1 | 1 | | 1 | 4 | 131 |
| 6 | UNMSM - JavaHaters | 3 | 3 | 1 | 2 | 1 | 2 | | 1 | 4 | 213 |
| 7 | UNJFSC - 2MYR | | | 2 | 8 | 1 | 1 | | 1 | 4 | 249 |
| 8 | UNMSM - SP11 | | | 2 | | 1 | 1 | 1 | 3 | 4 | 331 |
| 9 | UNU - TheBuggers | 2 | | 4 | | 2 | 1 | 1 | 1 | 3 | 397 |
| 10 | UL - DrinkTeam | 2 | | 2 | | 1 | 5 | | | 0 | 0 |
| 11 | UNP - sout | 1 | 4 | | | | | 11 | 4 | 0 | 0 |

Obrázek 2.2: Výsledková listina soutěže CONEISC z roku 2016 [4]

## 2.2 Soutěže v oblasti diabetes mellitus

I v oblasti onemocnění diabetes mellitus se pravidelně koná specializovaná soutěž, jejímž cílem může být například naleznutí nejlepšího algoritmu pro regulaci, nebo predikci určitých hodnot podstatných právě pro toto onemocnění. Toto onemocnění, kterým se mimo jiné zabývá i systém SmartCGMS, podrobněji rozebraného v kapitole 3, je potřeba z fyziologické stránky prozkoumat, a tím i čtenáře do problematiky samotného onemocnění diabetes mellitus uvést. Po jejím uvedení bude v sekci 2.2.2 představena soutěž, která se v tomto specializovaném odvětví pohybuje.

### 2.2.1 Onemocnění diabetes mellitus

Diabetes mellitus je heterogenní skupina onemocnění, která je způsobena relativním nebo absolutním nedostatkem hormonu inzulin (dále v sekci 2.2.1.2), ale v českém jazyce je spíše známá pod názvem cukrovka. Samotné onemocnění nabývá několika forem a může se projevovat různě. Absence inzulinu se dá kompenzovat jednorázovou injekcí nebo kontinuální stabilizací například pomocí inzulínové pumpy (sekce 2.2.1.2). Z důvodu různých život ohrožujících komplikací nesmí být toto onemocnění podceněno [5].

#### 2.2.1.1 Komplikace diabetu

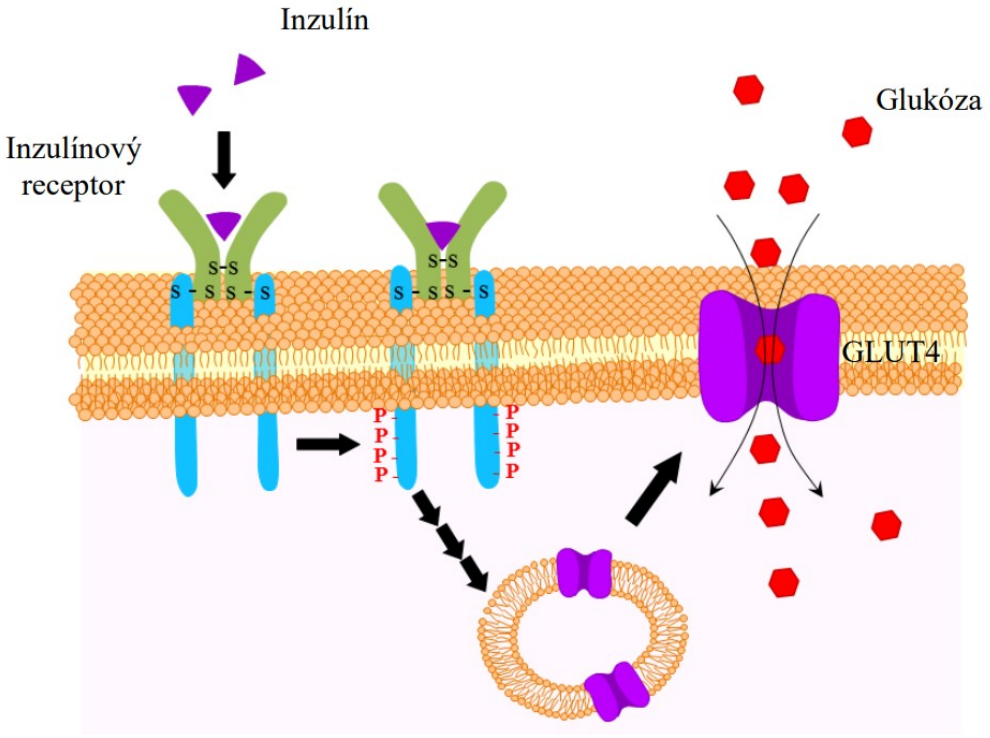
S onemocněním diabetes mellitus přichází i řada komplikací. Jednou z těchto komplikací je stav, kdy má pacient vyšší hladinu cukru v krvi než je v normě. Tento stav se nazývá **hyperglykémie**. Naopak stav, kdy je hladina nižší, označujeme jako **hypoglykemie**. Oba stavy jsou dosažitelné i u zdravého jedince, ale zdravá funkce těla je zvládne dostatečně regulovat. Pokud ale v opačném případě nebudou dlouhodobě dostatečně kompenzovány, hrozí několik rizik jako je selhání ledvin, kardiovaskulárního systému, cévní mozkové příhody, nebo i v případně dalších akutních situacích kóma či smrt. Vliv na hladinu cukru v krvi má však nejen samotný inzulin, ale i fyzická aktivita, nedostatečný nebo nadměrný příjem vysoko glykemických potravin, ale i požívání alkoholových nápojů [5]. To je jedním z důvodů, proč pro naleznutí optimálního modelu musí být senzor hladiny glukózy doprovázen i jinými senzory.

#### 2.2.1.2 Inzulin

Pomocí důležitého hormonu inzulin, jenž je ve zdravém těle vytvářen vytvářený v $\beta$-buňkách Langerhansových ostrůvcích slinivky břišní, reguluje tělo hladinu glukózy v krvi. Činí tak zpřístupněním glukózového kanálu jednotlivých buněk, který je důležitý pro přechod glukózy z krve do buněk skrze buněčnou membránu (viz obrázek 2.3). Inzulin také ovlivňuje syntézu glukózy v jaterních buňkách během hladovění

tak, aby nedošlo v krevním oběhu k jeho přesycení. Mimo jiné také reguluje uvolňování většího množství volných mastných kyselin, k němuž dochází mimo jiné při odbourávání tuku. Inzulin je tedy velmi nepostradatelným hormonem, na jehož funkci jsou buňky plně závislé [5].

**Inzulinová pumpa.** Inzulinová pumpa je elektronické zařízení, jenž funguje především na principu kontinuální inzulinové infuze zmíněného inzulinu. Je většinou složena z inzulinového zásobníku, infuzní kanyly, katetru a především řadiče, který se skládá z mikroprocesoru, motoru, baterie, displeje a tlačítek. Jelikož se jedná o elektronické zařízení s mikroprocesorem, může být jakákoliv funkce pumpy řešena a automatizována pomocí softwarového řídícího systému a může tak pacientovi nejen lépe kompenzovat diabetes, ale také mu do jisté míry zjednodušit život. Sestavit inzulinovou pumpu a k tomu najít perfektní algoritmus, který může naplno autonomně regulovat hladiny inzulinu, a to za jakýchkoliv podmínek, není lehký úkol. Avšak i malé objevy získané ze specializovaných soutěží zabývajících se onemocněním diabetes mellitus, mohou k tomuto cíli do určité míry přispět.




Obrázek 2.3: Ilustrace funkce inzulinu na glukózový kanál (**GLUT4**) u svalové a tukové buňky [5]

#### 2.2.1.3 Typy diabetu

Diabetes mellitus se rozděluje především podle příčiny vysoké hladiny glukózy v krvi. Samotných typů je několik, ale v následujících sekcích budou rozebrány pouze dva nejčastěji vyskytující se.

**Diabetes mellitus 1. typu.**
Tento typ diabetu je typický poškozením, nebo kompletním zničením $\beta$-buněk Langerhansových ostrůvků, které produkují hormon inzulin. Příčiny mohou být různorodé, ale uvádí se, že je to nejčastěji kvůli autoimunitní reakci kombinované s genetickými predispozicemi. Pacient s tímto typem onemocnění je většinou plně závislý na kompenzaci pomocí inzulínové stříkačky, pumpy, nebo například pera [5].

**Diabetes mellitus 2. typu.** Druhý typ onemocnění diabetes mellitus je způsobený z velké části špatným stravováním, ale může být ovlivněn i genetickou predispozicí jedince. Postiženi jsou převážně lidé s nadváhou či obezitou, jejichž buňky si vybudovaly tak silnou rezistenci na příchozí inzulin, že nadbytečná glukóza v krvi není dostatečně odváděna z krevního oběhu. Zároveň může tento jev významně přetížit $\beta$-buňky Langerhansových ostrůvků, což má za následek jejich postupné odumírání. Proto tedy pacient může být následně diagnostikován i s onemocněním 1. typu [5].

### 2.2.2 Soutěž BGLP

Soutěž Blood Glucose Level Prediction (BGLP) je zaměřen na soutěžním programování v oblasti predikce glukózy v krvi. Jedná se o náročný úkol ze zdravotní oblasti a nemoci diabetes mellitus určená pro výzkumníky využívající metody datové analýzy, strojového učení a umělé inteligence. Hlavním motivem soutěže je co nejspolehlivěji a nejpřesněji odhadovat a předpovídat stav glukózy v krvi. Tato predikce může mít několik využití, mezi které je možné zařadit i proaktivní vyvarování se hazardních stavů, jako je hypo a hyperglykémie (viz sekce 2.2) a nebo dalších průvodních komplikací. Cílem tak je přiblížit se běžné funkci zdravé slinivky břišní, která hladinu glukózy u zdravého jedince normálně reguluje [6].

Výzkumníci pro své pokusy v prvních ročnících soutěže museli používat uměle vytvořenou datovou sadu z důvodu absence dostupných reálných pacientských dat. Pro ročník 2020 již byla poskytnuta datová sada OhioT1DM obsahující 8 týdnů záznamů od dvanácti pacientů s onemocněním diabetes mellitus 1. typu. Zmíněné osoby procházely terapií inzulinové pumpy s kontinuálním monitorováním glukózy. Datová sada tak obsahuje samotné záznamy o aktuální glukóze v krvi, bazální i bolusový inzulín a také i nahlášené aktivity pacienta a záznamy z nositelných fitness zařízení [7].

| | 30 min | | 60 min | | | | |
|---|---|---|---|---|---|---|---|
| Paper ID | RMSE | MAE | RMSE | MAE | **Overall** | Online | Personalized |
| 13 | 18.22 | 12.83 | 31.66 | 23.60 | 86.31 | No | Yes |
| 6 | 19.21 | 13.08 | 31.77 | 23.09 | 87.15 | No | Yes |
| 16 | 18.34 | 13.37 | 32.21 | 24.20 | 88.12 | No | Yes |
| 15 | 19.05 | 13.50 | 32.03 | 23.83 | 88.41 | No | Yes |
| 1 | 18.23 | 14.37 | 31.10 | 25.75 | 89.45 | No | No |
| 14 | 19.37 | 13.76 | 32.59 | 24.64 | 90.36 | Yes | Yes |
| 7 | 19.60 | 14.25 | 34.12 | 25.99 | 93.96 | No | Yes |
| 9 | 20.03 | 14.52 | 34.89 | 26.41 | 95.85 | Yes | Yes |

Obrázek 2.4: Příklad výsledkové listiny soutěže BGLP [9]

**Limitace soutěže.** Jelikož se jedná o data reálných pacientů, jsou data chráněna pravidly soukromí DUA, které musí být účastníkem podepsány, aby je mohl při hledání řešení v podobě predikčního modelu použít. Podle informací uvedených na stránkách soutěže, jsou zmíněná data dostupná pouze dva týdny před uzavřením výsledků [6]. Po skončení soutěže jsou výsledky vyhodnoceny manuálně stanoveným výborem, který hodnotí přesnost řešení pomocí několika stanovených metrik (viz obrázek 2.4). To nejen že přináší určité riziko lidského faktoru při samotném hodnocení, ale i případnou zpětnou replikaci řešení. Pokud je zároveň datová množina dostupná účastníkům v momentě odevzdávání, může nastat i stav, kdy účastník specializuje učení modelu umělé inteligence na daný problém, a tím dojde k overfittingu modelu. Overfitting je pojem v odvětví umělé inteligence, popisující stav, kdy je algoritmus natolik specializovaný na trénovací sadu, až nemůže dělat přesné předpovědi nad jinou vstupní sadou [8].

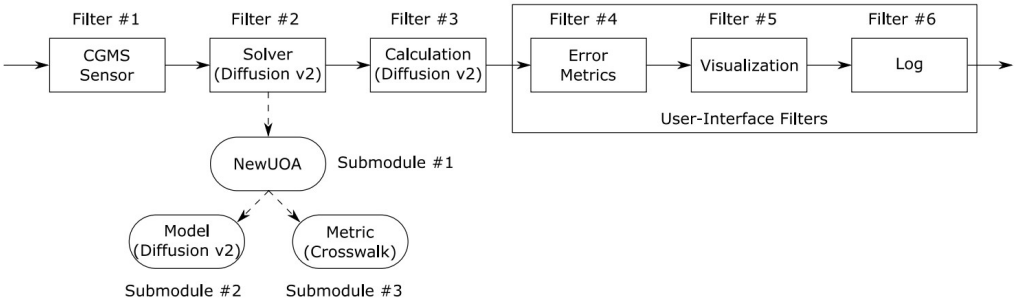
**Využití SmartCGMS.** Systém SmartCGMS (dále v kapitole 3), spouštěný přes nové webové rozhraní soutěží, by mohl přinést jednotný způsob vyhodnocování výsledků a umožnil by v případě úspěšného prvního sestavení provádět replikaci vložených řešení i po skončení soutěže, a to i po libovolné prodlevě. Zároveň by tyto pokusy mohly být vykonávány nad daty reálných pacientů. Nové rozhraní by tak představovalo vrstvu abstrakce, která by nedovolovala běžnému účastníkovi přímému přístupu k pacientským datům.

# SmartCGMS 3

SmartCGMS je softwarový framework a architektura určená pro analýzu signálů. Tento framework je složen z několika za sebou jdoucích entit, která jsou reprezentovány reálným nebo simulovaným zařízením. Mezi takové patří například senzor pro měření glukózy, model dynamiky glukózy a také případně inzulinová pumpa. Pomocí SmartCGMS je možné vytvářet simulace pro ladění specializovaných metod a v momentě kdy je metoda připravená, může být lehce nasazena do reálného prostředí využívající fyzická zařízení s použitím stejného kódu [10]. V následujících sekcích bude probrána architektura systému, vyjmenovány dostupné entity i způsob jakým bude soutěžící k implementaci přistupovat a v neposlední řadě i metoda sestavování a konfigurování dějícího se uvnitř soutěžního systému.

## 3.1 Architektura systému

Systém SmartCGMS je silně inspirován architekturou **HLA** [11], která je standardem vhodným pro sestavování a vývoj distribuovaných simulací velkých rozměrů. Nepozměněná forma HLA je však příliš složitá a náročná na zdroje pro nízkopříkonové zařízení. Proto byla architektura SmartCGMS pozměněna do *fall-through* architektury (viz obrázek 3.1), uchovávající hlavní výhody zmíněného standardu [12].




Obrázek 3.1: Diagram znázorňující *fall-through* architekturu systému SmartCGMS na příkladu sady entit předpovídající hladinu cukru v těle [12]

***Fall-through* architektura.** Jak již název napovídá, data ze vstupních modulů, jako je libovolný vstupní senzor, postupně propadávají definovanými entitami, které tyto data dále podle potřeby zpracují a pošlou dále ve formě zprávy následující entitě. Entity systému také označovány jako filtry, jsou implicitně vykonávány synchronně ve vlákně původního zdroje události [13].

## 3.2 Entity SmartCGMS

SmartCGMS může být složen ze širokého spektra entit, ty jsou postaveny podle konceptu Component Object Model (COM)[14], který udává, že jsou nezávislé na platformě či programovacím jazyce. Primárním jazykem je však moderní C++ a to především díky přednostem v oblasti optimalizace, typové a paměťové bezpečnosti, které jsou u nízkonapěťových zařízení velmi důležité. Tyto entity můžeme rozřadit do několika následujících skupin:

- **Filtr** – jedná se o základní entitu systému, která mívá typicky jeden specifický izolovaný úkol a obaluje ostatní entity.
- **Metriky** – specializovaná entita sloužící k vyhodnocování rozdílů mezi dvěmi signály. K procesu vyhodnocování dochází v momentě, kdy si o to nějaký filtr formou zprávy zažádá.
- **Diskrétní model** – entita udržující si stav, měnící se v postupných diskrétních krocích závislých na vstupních událostech ze zařízení.
- **Signálový model** – podobně jako diskrétní model generuje signál, ale v případě této entity nedochází k udržování stavu a je tak přímo závislá na vstupních datech.
- **Solver** – tato entita optimalizuje vstupní parametry modelu pomocí definované funkce.
- **Aproximátor** – aproximuje diskrétní signál v požadovaných časech, a pokud to zvolená metoda podporuje, zvládne extrapolovat i do budoucnosti.

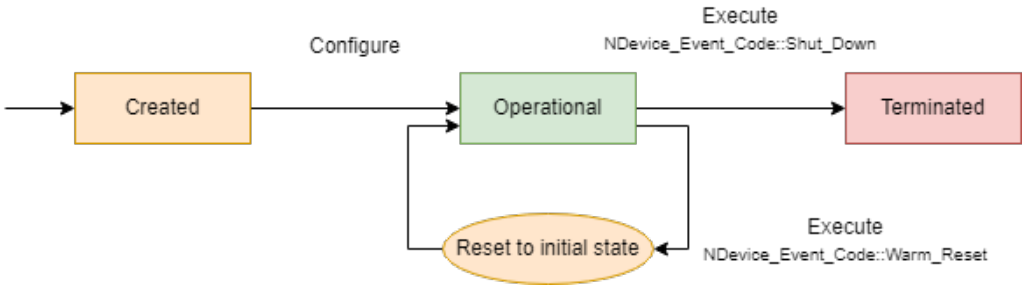
V případě, že se vrátíme k představě vytváření soutěží využívajících entit SmartCGMS, je potřeba analyzovat i způsob jakým budou soutěžící tyto entity implementovat. I když systém nabízí velké množství možností, je dost pravděpodobné, že účastník využije spíše prostý filtr, byť takové rozhodnutí může záviset na charakteru soutěžní úlohy.

### 3.2.1 Implementace filtru

Každý filtr SmartCGMS musí implementovat rozhraní `scgms::IFilter`, které definuje základní metody `Configure` a `Execute`. První metoda je volána před provozem celé sady filtrů a je určena ke konfiguraci a parametrizaci filtru. Důležité je, aby implementovaná metoda vracela stav `S_OK` při úspěchu, nebo `S_FALSE` pokud došlo ke konfiguraci s varováním. Jakýkoliv jiný stavový kód bude totiž označen za fatální

chybu, jenž vede k selhání konfigurační fáze a přeruší celý proces sestavování sady filtrů.

Jakmile je filtr úspěšně v provozu, je při datovém vstupu události volána metoda `Execute`. Zde by se měla nacházet celá funkcionalita filtru, která v případě zmíněné soutěže, může například predikovat hladinu glukózy v krvi na základě vstupních dat. Jelikož se jedná o *fall-though* architekturu, je povinností každého filtru předat vstupní událost spolu s libovolným množstvím vlastních dalšímu filtru, voláním metody `Execute` a nebo jej uvolnit metodou `Release`. Je vyžadováno, aby filtr naslouchal všem vstupním událostem a na základě obsahu zapouzdřené informace rozhodoval o další akci (viz obrázek 3.2). Především z důvodu, že může obsahovat signály pro ukončení a restart celého řetězce, kde při jejich přijetí je filtr povinen uvolnit své alokované zdroje.




Obrázek 3.2: Životní cyklus filtru SmartCGMS [10]

To jestli bude v simulaci filtr použit a v jakém pořadí, je závislé na nastavení konfigurace .ini souboru, která je rozebrána dále v sekci 3.3. Aby konfigurace rozeznala jednotlivé entity musí být popsány deskriptorem. Konkrétně filtry jsou popsány pomocí deskriptoru `scgms::TFilter_Descriptor`. Ten nese především informace o GUID filtru, jeho textovém popisu a definuje počet a typ vstupních parametrů [10].

## 3.3 Sestavení a konfigurace

SmartCGMS je open-source projekt chráněný Apache licencí, který je volně dostupný ke stažení z oficiálního git repozitáře [15]. Základ systému je složen ze čtyř repozitářů:

- **common** - repozitář obsahující rozhraní všech entit a podpůrných implementací usnadňující jejich použití.
- **core** - tento modul obsahuje implementaci entit systému jako jsou filtry, modely a nebo například metriky, které jsou podporovány systémem SmartCGMS a jsou potřebné pro vybrané základní funkce.

- **console** - jedná se o konzolové rozhraní pro framework SmartCGMS, které slouží pro spouštění předem konfigurované sady filtrů, zpracování dávkových příkazů a další.
- **desktop** - poslední modul představující grafické rozhraní pro ovládání SmartCGMS frameworku. Vhodné je především pro efektivní i rychlé konfigurování jednotlivé sady filtrů a sledování výsledků.

**Závislosti.** Pro sestavení samotného SmartCGMS a jeho spuštění je vyžadováno následujících prerekvizit:

- překladač s C++17 - například gcc 7 nebo vyšší
- CMake
- Qt 6 - použito na grafické rozhraní SmartCGMS.

**Adresářová struktura.** Adresářová struktura sestaveného řešení systému SmartCGMS obsahuje několik podstatných souborů, jejichž vlastnosti budeme využívat při návrhu soutěžního systému. Mezi základní soubory zahrnujeme následující:

- scgms-desktop - spustitelný soubor pro grafické rozhraní pro ovládání frameworku, kterého ale v nově navrhovaném soutěžním systému nebude využito.
- scgms-console - spustitelný soubor umožnující spouštění různých předem definovaných pokusů pomocí konzolového rozhraní.
- scgms - soubor ve formě dynamické knihovny obsahující všechny jádrové funkce a samotné rozhraní pro komunikaci mezi moduly SmartCGMS.
- filters - složka obsahující všechny použitelné entity systému. Zde se nachází i entity vytvořené externím uživatelem.

**Konfigurační .ini soubor.** V případě, že spouštíme konzolovou aplikaci SmartCGMS je potřeba v argumentech specifikovat cestu ke konfiguraci popisující řetězec použitých filtrů. SmartCGMS ukládá konfiguraci v lidsky čitelném formátu .ini souboru, která je měnitelná jak ručně v textové formě, tak i v GUI, které ho zvládne jak importovat tak i exportovat. Konfigurace je rozdělená do několika sekcí, kde každá sekce představuje jeden filtr. Začátek sekce musí splňovat následující formát:

```
1 [Filter_###_{XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX}]
```

Kde symboly ### značí celočíselné označení pořadí filtru z celé sady doplněné nulami, tak aby mělo číslo tři cifry. Poté sekvence znaků křížků oddělené pomlčkami uzavřené složenou závorkou představují identifikátor GUID. GUID se skládá z 32 hexadecimálních číslic, které jsou obvykle zobrazeny jako 8 skupin po 4 číslicích oddělených pomlčkami, například: "8FAB525C-5E86-AB81-12CB-D95B1588530A".

Filtry mají možnost v deskriptoru definovat sadu požadovaných parametrů. Ty pak mohou být instancovány uvnitř vyhrazené konfigurační sekce. Každý parametr odpovídá standardnímu zápisu .ini souboru klíčů a hodnot oddělených znakem rovnítka. Pro příklad definice jednoho filtru (jedné sekce) a dvou vstupních parametrů můžeme uvést následující[10]:

```
[Filter_001_{8FAB525C-5E86-AB81-12CB-D95B1588530A}]
Offset=1.5
Reference_Signal={09B16B4A-54C2-4C6A-948A-3DEF8533059B}
```

# 4 Izolované a vzdálené sestavení zdrojových kódů

Jak již bylo zmíněno v předešlých kapitolách, v oblasti soutěžního programování je schopnost izolovat proces kompilace zdrojového kódu a jeho následného spuštění klíčová. Celkový proces sestavení a vyhodnocení může být poměrně výpočetně náročný a měla by být zvážena distribuce zodpovědné komponenty. Vzdálená a distribuovaná komponenta by tak přímo neovlivňovala další funkci webového rozhraní, přes které mohou být obsluhovány i jiné méně náročné procesy. V následujících sekcích bude v této kapitole probrána problematika překladu zdrojových kódů, pojem izolace a způsob, jakým se jí dá dosáhnout, ale také zajištění distribuce a zabezpečení sdílených souborů. Na problematiku síťového zabezpečení v distribuovaném systému pak dále naváže sekce 4.4.

## 4.1 Sestavení a spuštění zdrojových souborů

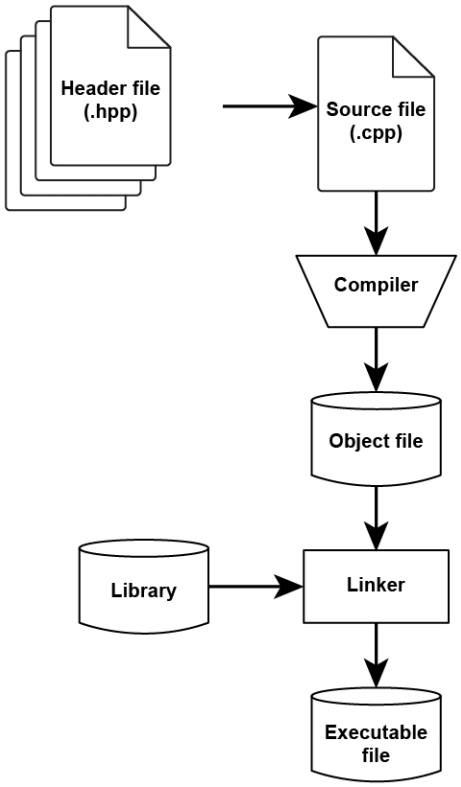
Účastníci programátorských soutěží mají možnost využívat soutěžní rozhraní, pomocí kterého mají umožněno vkládat softwarová řešení ve formě zdrojových kódů a tím demonstrovat své schopnosti a dovednosti v praktické rovinně. Jelikož se tato diplomová práce zabývá návrhem soutěžního systému, bude potřeba definovat co vlastně zdrojový kód v tomto odvětví představuje. Zdrojový kód je statický popis výpočtu počítače pomocí sady instrukcí zvoleného programovacího jazyka v lidsky čitelné podobě [16]. Tato podoba je definována programovacím jazykem, který si účastník soutěže zvolil. Z pohledu problematiky této práce se jedná o zdrojový kód zvolené entity systému SmartCGMS, jež byla definována v sekci 3.2. Primárním a zároveň jediným použitelným jazykem je ale v tento moment pouze jazyk C++, byť širší spektrum použitelných jazyků by mohlo být pro účastníky soutěže atraktivnější. Jazyk C++ je univerzální programovací jazyk stvořen jako rozšíření jazyka C, které jej rozšiřuje o objektově orientované paradigma, ale také typovou a paměťovou

bezpečnost. Jedná se o multiparadigmatický jazyk, což znamená, že soutěžící pro dosažení žádaného cíle musí definovat sekvenci kroků, které pak program vykonává. Zdrojový kód napsaný v jazyce C++ musí být přeložen pomocí překladače a linkeru (dále sekce 4.1.1) do strojového kódu, aby mohl být následně vyhodnocovacím systémem spuštěn a proveden [16]. Systém SmartCMGS, jak již bylo zmíněno v předešlé kapitole, využívá pro správu překladu a sestavování nástroj CMake, který bude dále analyzován v sekci 4.1.2.

### 4.1.1 Překladač jazyka C++

Překladač je program, obecně sloužící k transformování zdrojových kódů v textové do jiné formy. Tento proces transformace je rozdělitelný na dvě fáze (viz obrázek č.4.1). V první kompilační fázi dochází ke zpracování zdrojových kódů do formy objektových souborů, které obsahují jak strojový kód, tak také metadata. Metadata nesou informace o programových symbolech, konstantách a také závislostech mezi jednotlivými moduly. Během kompilační fáze dochází od preprocessingu, tokenizaci až k samotnému zpracování. Jakmile jsou objektové soubory připraveny, dochází k druhé fázi označována jako linkovací. Při linkovací fázi dochází ke spojování všech objektových souborů do jednoho spustitelného programu [16].

V prostředí vyhodnocování soutěže by měl být transparentně použit standardizovaný překladač, aby nedocházelo během překladu vyhodnocovacím systémem k netypickým chybám. Překladačů je několik, liší se především v podporovaných platformách, vyžadovaných standardech jazyka C++ a licencí. Pro účel této diplomové práce se nabízí překladač GCC, který je sice založený na Unix operačních systémech, ale pomocí nástrojů MinGW nebo Cygwin je možné ho využívat i na systému Windows. Před začátkem soutěže tak může být soutěžícímu doporučen překladač GCC, ale při dodržení standardů a pravidel tohoto překladače může být bezproblémově použít i jiný překladač.




Obrázek 4.1: Schéma překladu zdrojových kódů jazyka C++ [17]

### 4.1.2 CMake

CMake je multiplatformní nástroj, který slouží pro konfiguraci, sestavení a testování softwarových projektů. Sestavovací proces CMake je definován pomocí souboru `CMakeLists.txt`, ten obsahuje sbírku příkazů a instrukcí popisující strukturu zdrojových souborů. Také tento textový soubor popisuje artefakty, jako jsou výstupní binární soubory a knihovny. Při sestavování projektu pomocí nástroje CMake dochází, pokud není specifikováno jinak, ke zvolení jednoho z dostupných platformově závislých generátorů. Generátor je zodpovědný za vytváření nativní konfigurace pro sestavení projektu. Tato konfigurace pak slouží k překladu pomocí zvoleného překladače, který ji zvládne zpracovat [18]. Tento nástroj je využíván systémem Smart-CGMS a bude použit i při překládání zdrojových kódů vložených soutěžícími přes webové rozhraní.

## 4.2 Izolace prostředí

Zdrojové kódy, jejich sestavení a následné spouštění je možné označit jako velmi proměnlivý a nepředvídatelný proces. Může být totiž ovlivněn několika faktory

jako jsou například externí závislosti, specifické hardwarového vlastnosti, optimalizační triky a chyby překladačů. Z tohoto důvodu je potřeba zvážit izolaci procesu překladu od ostatních části již při návrhu soutěžního systému, aby nedošlo kvůli zmíněnému problému k jeho nečekanému selhání a případnému poškození. Izolaci si můžeme představit jako znemožnění přístupu procesu ke zbytku systému tím, že je omezen pouze na její redukovanou část [19]. V textu této sekce budou prozkoumány možnosti zmíněného omezení pomocí systémového volání `chroot`, virtualizace a kontejnerizace. Prozkoumána bude i oblast síťové izolace pomocí nástroje Docker v sekci 4.2.4.

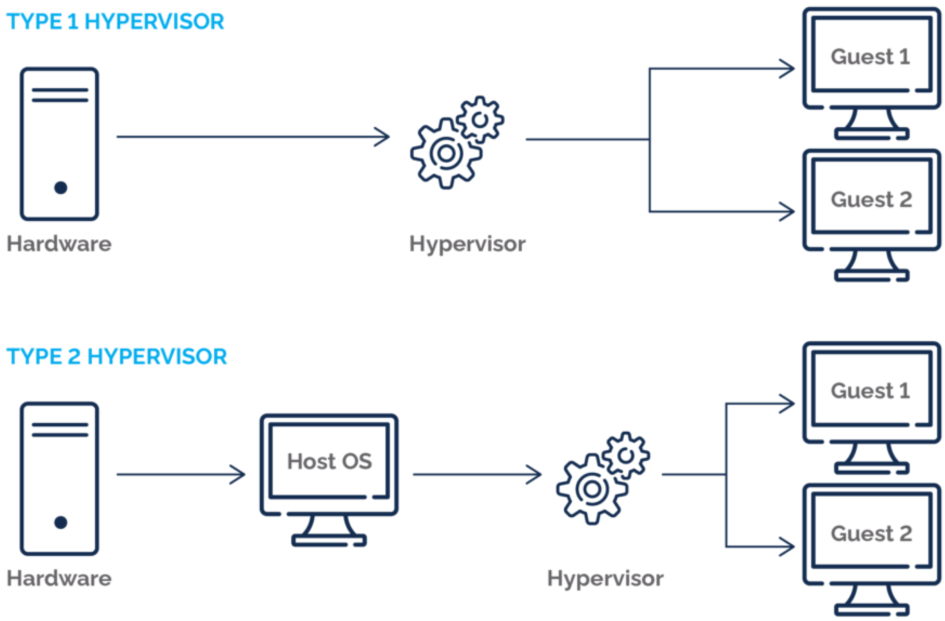
### 4.2.1 Funkce chroot

Chroot (Change Root) je systémová funkce dostupná na operačních systémech založených na systému Unix, která slouží ke změně kořenového adresáře a spuštění zadaného příkazu. Pro spuštěný příkaz a všechny jeho potomky vzniklé rozdvojením pomocí speciálního systémového volání `fork` dojde k oddělení od reálného kořenového adresáře a limitaci možných škod pouze pro vyhrazený adresář. Tato praktika se nazývá **chroot jailing**. Ikdyž byl originální záměr systémové funkce jiný a byla do systému přidána už roku 1979, je stále tato technologie použitelná pro izolaci procesů. Je tomu tak především z důvodu, že mají systémoví administrátoři plnou kontrolu nad specifikováním souborů dostupných pro spouštěnou službu. Správa vlastní webové služby pomocí čisté funkce `chroot` může být však zdlouhavá a neefektivní práce [19]. Kromě toho poskytuje chroot pouze procesní izolaci, která může být rozšířena pomocí virtualizace i o další oblasti. Z tohoto důvodu není nástroj pro účel této diplomové práce dostačující a tak bude v následující sekci prozkoumána možnost virtualizace.

### 4.2.2 Virtualizace

Virtualizace je technologie sloužící pro vytváření virtuálních systémů, které mají napodobovat schopnosti a prostředí fyzického hardwaru. Technologie také poskytuje softwarové řešení pro oddělení hardwarových komponent jednoho počítače jako je procesor, operační paměť a disk mezi několik virtuálních strojů. Virtualizované prostředí se chová jako nezávislá, oddělená entita, byť jak již bylo řečeno sdílí část prostředků s hostujícím fyzickým systémem. [20] Fyzické zařízení tak může obsluhovat několik virtuálních verzí různých serverů s jejich vlastním operačním systémem. Virtualizace tak poskytuje flexibilní způsob využití zdrojů, bez nutnosti manuální obsluhy. Fyzické zařízení nazýváme **hostitelem**, na jehož pozadí je spuštěn takzvaný hypervizor, což je softwarová nebo hardwarová komponenta umožnující vytváření a správu virtuálních strojů. Hypervizor řídí přiřazování zdrojů a zároveň zaručuje, že operace virtualizovaných strojů nezasahují do chodu ostatních. Existují

dva typy hypervizorů znázorněných na obrázku 4.2. První typ hypervizoru nahrazuje hostitelský operační systém a je tak bez další vrstvy abstrakce přímo spuštěn na poskytnutém hardwaru. Využitelný je především v datových, výpočetních a webových centrech a ve většině cloud služeb. Druhým případem je hypervizor druhého typu, který je hostován operačním systémem ve formě procesu na pozadí. Nalezne tak využití spíše v desktopových a vývojových prostředích, kde nedochází k tak vysoké pracovní zátěži. Mimo jiné je tento hypervizor preferovaný v případě, kdy se uživatel rozhodne souběžně využívat několik operačních systémů, ale zároveň mít přístup k hostitelskému systému, což v případě soutěžního systému nevyžadujeme [21].

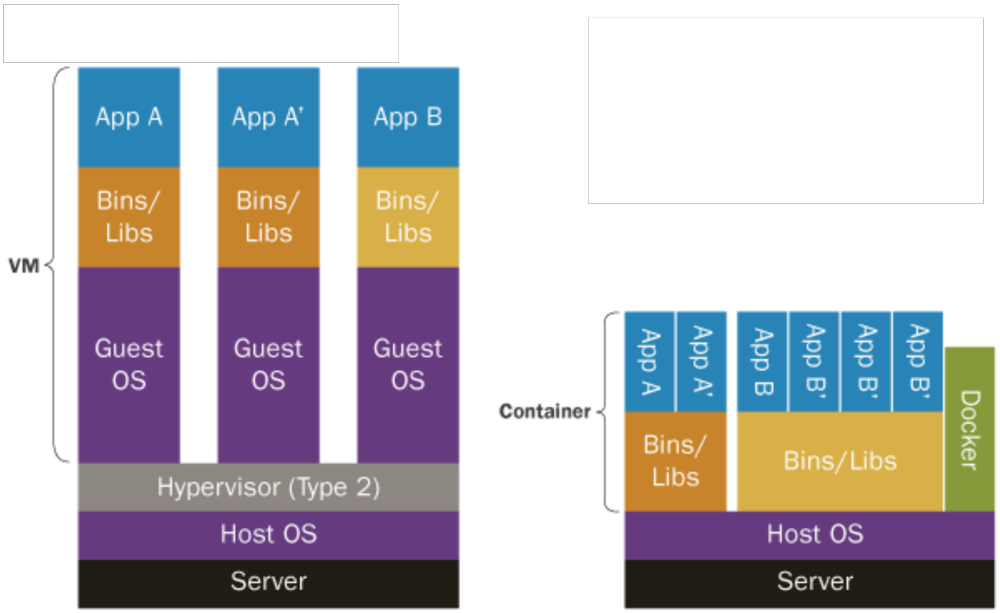



Obrázek 4.2: Znázornění dvou typů hypervizorů virtuálních strojů[21]

Virtualizace pomocí hypervizoru prvního typu se tak tváří jako výhodnější kandidát z pohledu výkonu. Posloužil by pro spouštění kompletně izolovaných simulací v případě, že tomu bude odpovídat i fyzické rozložení a nastavení sítě. Omezení však nastává v horizontální škálovatelnosti, jelikož na každém fyzickém stroji mají jednotlivé virtualizované stroje vlastní instanci operačního systému, která kromě zabírání sdílené paměti jejím jádrem, jednotlivých ovladačů a procesů na pozadí, také zatěžuje stroj z ohledu plánovače a kontextového přepínání.

### 4.2.3 Kontejnerizace

Kontejnerizace je podobou již zmíněné virtualizace, kde aplikace využívající tuto technologii běží v izolovaném uživatelském prostředí také nazývaném jako kontejner. Jednou z hlavních předností kontejnerů je, že se jedná o plně sestavené přenositelné výpočetní prostředí, které obsahuje celou vyžadovanou sadu nástrojů jako jsou knihovny, spustitelné a konfigurační soubory, tak že jsou všechny při jejím nasazení zapouzdřeny uvnitř. Oproti běžné virtualizaci nepotřebují vlastní operační systém, který místo toho sdílí s hostitelským strojem (viz obrázek 4.3). Místo hypervizora, který je přítomný u virtualizační technologie, využívá kontejnerizace kontejnerové jádro, které slouží jako rozhraní mezi operačním systémem, jeho uživateli a spravuje běh kontejnerů. Kontejner je tak za vrstvou abstrakce omezen od přímého přístupu ke zdrojům hostujícího operačního systému, tedy podobně jako by se jednalo o úspornější verzi virtuální stroje. Při zachování této abstrakce tak mohou být kontejnery spuštěny na libovolném typu infrastruktury bez vyžadování specializovaných úprav [22].



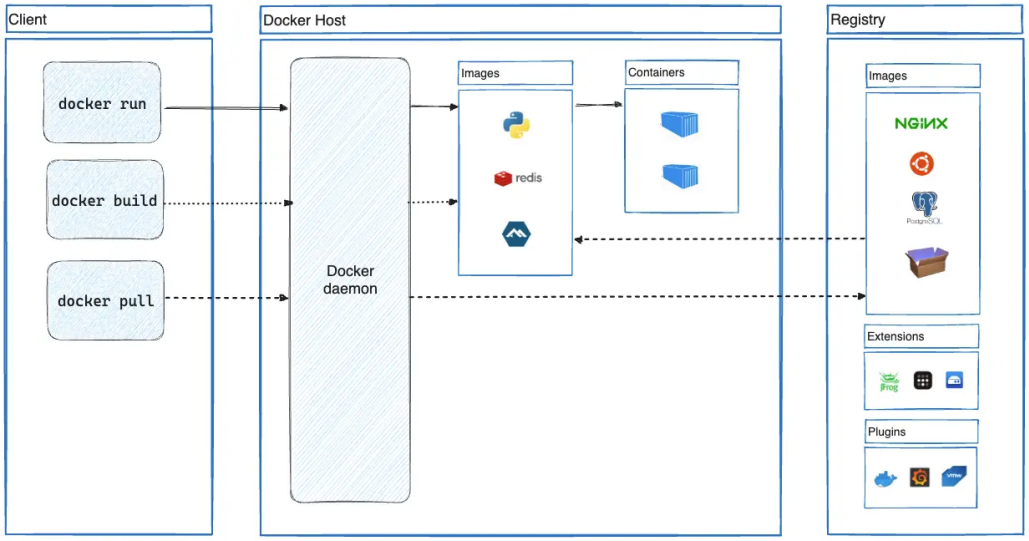

Obrázek 4.3: Porovnání vrstev abstrakce virtuálních strojů s hypervizorem 2. typu oproti přístupu kontejnerizace [23]

Kontejnerizace stejně jako tradiční virtualizace poskytuje plnou izolaci aplikací a dovoluje jim běžet nezávisle na ostatních. Připravený kontejner je však při nasazování oproti virtuálním strojům mnohem méně objemný a proces nasazení tak trvá v jednotkách vteřin. Odstraněním vrstvy nutného hypervizora a virtualizovaných operačních systémů zároveň dochází k vyšší serverové výkonnosti a cenové efek-

tivitě. Soutěžní systém by tak mohl využít vlastností kontejnerizace pro překlad a vyhodnocování zdrojových kódů, které představují hrozbu z hlediska bezpečnosti a vyžadují vysoký výkon k překladu i spuštění [22].

### 4.2.4 Docker

Docker je moderní nástroj pro vývoj, nasazení a spouštění aplikací ve formě kontejnerů. Poskytuje širokou sadu funkcí sloužící mimo jiných i pro izolaci aplikací, definici vlastní infrastruktury a díky využívání výhod kontejnerizaci nabízí také způsob rychlého nasazení vyvinutých softwarových komponent. Ke kontejnerům mohou být pomocí tohoto nástroje připojeny vlastní virtuální sítě, diskové oddíly a úložiště. Virtuální síť tak může poskytnout další úroveň izolace od okolního světa. Docker využívá principů client-server architektury, která je zobrazena na obrázku 4.4. Klientskou aplikací rozumíme jakékoliv rozhraní pro interagování s Docker ekosystémem na pozadí, mezi které patří například Docker CLI, Windows Docker Desktop, Portainer, nebo také Docker Compose. Serverovou část pak obsluhuje Docker hostitel v podobě daemona, který pro komunikaci poskytuje několik koncových bodů ve formě REST API [24, 25].




Obrázek 4.4: Architektura nástroje Docker [25]

#### 4.2.4.1 Docker daemon

Docker daemon také známý jako `dockerd` je stěžejní komponentou celého docker prostředí. `Dockerd` je spuštěn jako služba na pozadí operačního systému, sloužící pro správu Docker objektů. Jedná se jak o správu celého životního cyklusu kontejnerů,

který zahrnuje jeho vytváření, spouštění a následné monitorování, tak i spravování definovaných virtuálních sítí, obrazů, pluginů a mnoho dalších [25]. Stará se také o komunikaci a využívání zdrojů operačního systému hostitele, tak aby spuštěné kontejnery byly schovány za vrstvou abstrakce a nebylo jim z důvodu bezpečnosti umožněno přímého přístupu. Tato vrstva abstrakce také poskytuje svým uživatelům bezproblémový způsob vývoje, nasazení a správy vyvíjených aplikací v izolovaném kontejnerovém prostředí [25].

Uživatel interaguje s docker daemon pomocí HTTP požadavků zaslaných na REST API. Tyto požadavky následně zpracuje a provede. Požadavky umožňují vytváření nových kontejnerů, jejich spouštění, vypínání, správu sítě a také správu operací úložiště. Jelikož se jedná o síťové rozhraní může být k němu přistupováno i ze samotných kontejnerů. To však přináší určité riziko, kdy útočník získá kontrolu nad kontejnerem a mohl by poškodit jak ostatní kontejnery, tak i hostitelský systém [25].

#### 4.2.4.2 Docker virtuální síť

Nástroj docker umožňuje definovat komplexní virtuální síťový systém pro komunikaci mezi kontejnery, hostitelským zařízením a vnější sítí. V dnešní době existuje pět druhů virtuálních podsítí poskytující široké spektrum možností. Jmenovitě se jedná se o následující bridge, host, overlay, ipvlan, macvlan. Kromě zmíněných podporuje docker i využívání sítí třetí strany [26].

**bridge.** Virtuální docker síť typu `bridge` vytváří softwarový ekvivalent mostu, který stejně jako v hardwarové terminologii spojuje dvě části sítě na linkové vrstvě modelu ISO/OSI [27]. Ke spojení dojde mezi virtuální docker sítí s lokální, což znamená, že kontejnery zůstávají součástí virtuální sítě a mohou spolu komunikovat pomocí interních IP adres a DNS jmen, ale zároveň přistupovat k internetu [26].

**host.** Kontejner využívající síťový režim `host` sdílí síť svého hostitele bez jakékoliv izolace. Nemá sice v lokální podsíti vlastní přiřazenou IP adresu, ale jeho používaný port je přímo uveřejněn na síťovém rozhraní hostitele. Kontejner se tak neschovává za žádnou vrstvou síťové abstrakce a může s okolím komunikovat jako jakákoliv jiná síťová služba [26].

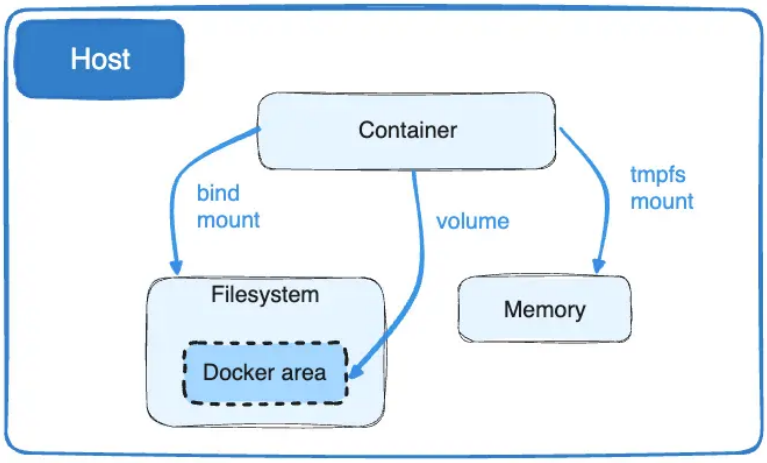
**overlay.** `Overlay` je ovladač pro distribuované síťě, který umožnuje sdílení přes několik docker daemonů (viz sekce 4.2.4.1). Kontejnery v této síti tak mohou mezi sebou komunikovat bez jakékoliv další směrovací služby. Tento typ je důležitý především při orchestraci kontejnerů (dále v sekci 4.2.4.4) umožnující zachování izolace na linkové vrstvě i v distribuovaném řešení [26].

**ipvlan.** Pokročilá virtuální síť `ipvlan` umožňuje totální kontrolu nad adresováním protokolů IPv4 a IPv6. Tento ovladač je vhodný v integrování kontejnerizovaných služeb do již existující fyzické sítě. Pro sítě tohoto typu si je možné definovat vlastní rozhraní, což umožňuje určité benefity z pohledu výkonu oproti běžným `bridge` sítím [26].

**macvlan.** Poslední typem docker sítě je `macvlan`, která umožňuje přiřadit kontejneru MAC adresu, který se pak tváří jako přímo připojené zařízení k fyzické síti, byť se stále jedná hostovaný proces. Kromě MAC adresy musí být kontejneru definována maska podsítě i výchozí brána fyzické sítě. Rizikem tohoto přístupu může být však neúmyslné degradování reálné sítě z důvodu vyčerpání rozsahu IP adres a definování nevhodného množství MAC adres, které může dále zpomalit přepínání[26].

### 4.2.4.3 Persistence dat

Nástroj Docker využívá pro problematiku persistence dat kontejnerů mechanizmu nazveného jako **volume**. Ten funguje na základě funkce `mount` souborového systému, která umožňuje připojení části jiného souborového systému do zvolené složky označované jako **mount point**. V kontextu docker kontejnerů se tak jedná o složku existující i mimo jeho životní cyklus a neovlivňuje tak ani celkovou velikost kontejneru (viz obrázek 4.5) [25]. V kombinaci vzdáleného souborového systému (dále sekce 4.3) tak přestavuje možné řešení persistence dat distribuovaného soutěžního systému spuštěného přes několik docker daemonů.




Obrázek 4.5: Ukázka použití mechanizmu připojeného Docker volume v kontextu hostitelského souborového systému [25]

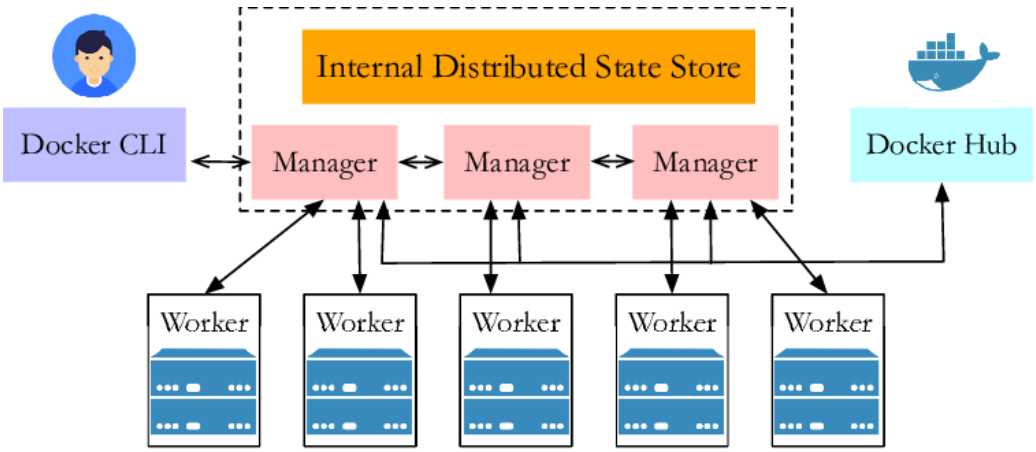
#### 4.2.4.4 Orchestrace kontejnerů

Docker kontejnery díky svým vlastnostem zaručují, že zabalené aplikace poběží stejným způsobem jako v prostředí ve kterém byly vyvíjeny, což umožňuje jejich rychlé a snadné nasazení. Tato vlastnost také otevírá i možnosti horizontálního škálování pomocí automatizovaných nástrojů využívajících funkcí několika Docker daemonů. Pro tyto účely existuje několik nástrojů, v následujícím textu se však budeme zabývat pouze aktuálně nejznámějšími jako je Docker Swarm a Kubernetes [25].

**Kubernetes.** Kubernetes je open-source nástroj pro orchestraci a správu životního cyklu kontejnerizovaných aplikací. Kubernetes umožňuje definovat komplexní infrastrukturu distribuovaných služeb, kterou mohou být podle libosti škálovány, postupně aktualizovány, přepínány mezi jednotlivými verzemi aplikací a spoustu dalších. Může být také definováno kolik výpočetních zdrojů je pro chod systému vyžadováno a nástroj podle dostupných možností jednotlivé služby a aplikace rozloží mezi fyzické uzly. Stará se mimo jiné i o restartování aplikací v případě pádu [28]. Kubernetes je tedy nástroj zjednodušující spolehlivé spuštění a správu komplexních systémů, mezi které ale nově navrhovaný soutěžní systém rozsahově nepovažujeme.

**Docker Swarm mód.** Swarm mód je rozšířenou funkcí nativně obsažené v aktuálních verzích Dockeru sloužící pro kontejnerovou orchestraci. Orchestrace může být vykonána přes několik hostujících daemonů pouze v momentě kdy jsou přepnuty do swarm módu. Nad touto skupinou daemonů pak může být distribuován shluk vybraných aplikací a služeb. Každý Docker daemon v orchestrovaném shluku komunikuje s ostatními daemony přes nativní Docker REST API [28].

Na obrázku 4.6 je možné spatřit architekturu Docker Swarm módu několika fyzických uzlů. Docker daemoni spuštěný na jednotlivých fyzických uzlech jsou zařazeni do jedné ze dvou skupin. První skupina uzlů obstará roli manažer, zodpovědného za zpracování specifikací uživatele, stahování závislostí z Docker Hub, rozdávání pracovních úloh a udržování aktuálního stavu shluku. Druhá skupina jsou pak worker uzly přijímající a vykonávající zadané úlohy [28]. Mezi takové úlohy může mimo jiné patřit i nasazení samotného soutěžního systému, který tak bude rozdistribuován přes celý definovaný Docker Swarm.




Obrázek 4.6: Schéma worker-manažer rozložení Docker Swarm módu [28]

## 4.3 Distribuce a zabezpečení souborů

Využití distribuovaného řešení soutěžního systému přináší problém v oblasti sdílení souborů obsahující zdrojové kódy entit systému SmartCGMS. Musí být proto specifikován způsob, jakým může soutěžní systém předávat tyto soubory svým vzdáleným pracovním uzlům. Abychom mohli využít vlastností Docker volume popsaného v sekci 4.2.4.3, je vyžadováno, aby se zvolený způsob distribuce souborů během souborových operací choval stejně jako běžný souborový systém. Pro tyto účely se nabízí využiti síťových souborových systémů. Než ale budou v následujících sekcích rozebrány možné technologie síťových souborových systémů, je potřeba se seznámit se souborovými právy operačních systémů založených na systému Unix, které valná většina Docker kontejnerů používá [29].

### 4.3.1 Oprávnění v souborovém systému

Souborové oprávnění je bezpečnostním prvkem, jenž je součástí téměř všech systémů. Určuje kteří uživatelé a jakým způsobem mohou přistoupit k souborům a adresářům souborového systému. Z pohledu na souborový systém operačních systémů jádra Unix je každý soubor a adresář označen sadou bezpečnostních atributů. Bezpečnostní atribut pro nastavení bezpečnosti definuje tři skupiny oprávnění. První skupina stanovuje oprávnění pro vlastníka souboru, druhý označuje skupinu vlastníků a třetí uděluje oprávnění všem ostatním účtům. Tato oprávnění jsou v případě číselného módu reprezentována agregací čísel 4, 2 a 1, nebo v případě symbolického módu znaky r, w a x. Konkrétní význam čísel a znaků je následující [29]:

- 4 nebo r – uživatel může obsah souborů číst a záznamy adresářů procházet.

- 2 nebo w – oprávnění k modifikaci obsahu souboru a adresářů. Toto oprávnění mimo jiné umožňuje i přidávat nové soubory.
- 1 nebo x – jedná se oprávnění ke spuštění a manipulaci se soubory nebo adresáři. Toto oprávnění svým způsobem ovlivňuje i chování předešlých oprávnění, především při průchodu adresáři a nebo získávání dodatečných informací o souboru.

Vlastníkem souboru může být jakýkoliv uživatel systému označený pomocí **UID**. Ty jsou rozděleny a obecně identifikováni následně [29]:

- 0 – správce systému také známý jako root
- 0-500 – systémové účty jako je například dockerd (Docker daemon)
- 500+ – účty běžných uživatelů

Naopak vlastnické skupiny jsou označovány pomocí čísla **GID**. Skupiny jsou stejně jako je tomu u uživatelů obecně identifikováni následovně [29]:

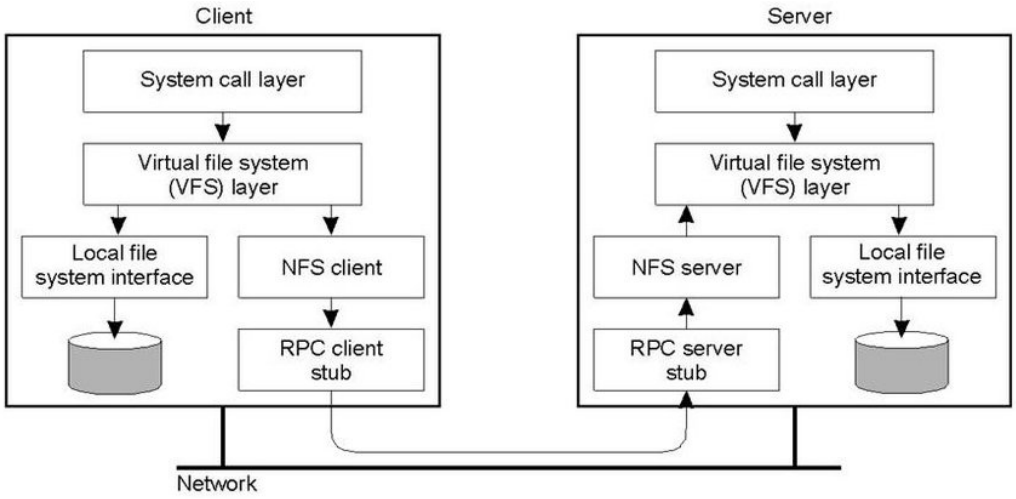
- 0 –- je skupina vyhrazená pro root uživatele
- 1-99 – skupiny pro systémové aplikace.
- 100+ – skupiny pro běžné uživatele.

**Využití soutěžním systémem.** Pokud bychom zmíněné znalosti použili v kontextu soutěžního systému, můžeme tímto způsobem omezovat přístup distribuovaných pracovních uzlům k adresářům síťového souborového systému, ke kterým v určitý moment nesmí přistupovat. Naopak pokud by zadání bylo předáno právě tomuto uzlu, je možné nastavit patřičná práva a vlastníky, tak aby mohl obsah adresáře editovat jenom on a samozřejmě root uživatel.

### 4.3.2 NFS

Jedním v dnešní době rozšířeným síťovým souborovým systémem je NFS, který se sice začal vyvíjet už v polovině 80. let, ale za léta získal několik aktualizací ve formě verzí NFSv3 a nejnovější NFSv4.2. NFS poskytuje transparentní přístup ke vzdáleným souborům, což znamená, že s daty uloženými na vzdáleném serveru je možné pracovat tak, jako by se jednalo o soubory lokální. Byť na začátku navržen jako součást souborových systémů založených na systému Unix, je nyní zaměřen na to, aby ho bylo možné jednoduše přenést na různé operační systémy a architektury [30, 31].

Běžné souborové systémy mohou obsahovat širokou škálu různorodých dat. Pro transformaci těchto dat využívá NFS standard **XDR**, popisující jakým způsobem mají být data před samotným přenosem serializována. V momentě transportu jsou tedy ve standardem definované binární podobě, což zaručuje interoperabilitu jak mezi libovolnou distribucí operačního systému, tak i různými procesorovými architekturami [30]. Samotná transportace dat je řešena pomocí volání Remote procedure call (**RPC**), umožnující spouštění procedur v jiném adresním prostoru než volající proces běží. Toto volání funguje obecně na způsobu request–response, při kterém klient zažádá o nějaká data a server podle svých možností zašle zpět. Tento proces výměny zajistí, že uživatelé a aplikace používající souborové operace přistupují k síťovému souborovému systému stejně, jako by se jednalo o jakýkoliv jiný souborový systém (viz obrázek 4.7) [30, 31].




Obrázek 4.7: Diagram systémového volání síťového souborového systému NFS [31]

Od verze NFSv3 je pro přenos dat využit protokol TCP a dále od verze NFSv4.2 je tento protokol standardizován na port 2049, což umožňuje určité bezpečnostní omezení nastavení firewallu. Nejnovější verze také poskytuje autentizační možnosti a opatření [30, 31]. Jedná se o mechanizmy AUTH_SYS s kombinací Export Controls a AUTH_GSS v kombinaci s Kerberos [32]. Další způsob jak zvýšit zabezpečení dat síťového souborového systému, v momentě kdy je klientský uzel připojen, je nastavením přístupových práv již probraných v sekci 4.3.1.

### 4.3.3 AFS

AFS je další příklad distribuovaného souborového systému, který využívá modelu klient/server, kde jsou veškerá data uložena na serveru. Klientské zařízení pak tato data podle potřeby stahují a po jejich stažení ukládají do své interní cache umístěné

na lokálním disku. Pokud klient aktualizuje obsah souboru, pošle po jeho uzavření kopii na server. Naopak pokud došlo k editaci jiným uzlem v připojené síti a klient má uloženou cache souboru, je informován serverem o jeho změně [33].

V oblasti bezpečnosti využívá AFS sadu pravidel definovanou standardem Access Control List (**ACL**), která v tabulce přístupů definuje, kteří uživatelé/stroje mohou přistupovat k souborovému systému i k jednotlivým souborům. Každý soubor souborového systému odkazuje do ACL tabulky, ve které je seznam uživatelů, kteří jsou označeny přiřazenými oprávněními. Existuje několik přiřaditelných oprávnění, které rozdělujeme následovně [33]:

- Lookup (l) – dovoluje vypsat obsah adresáře, přiřazené tabulky ACL a přístupu k podadresářům.
- Insert (i) – umožňuje vytvářet nové soubory nebo adresáře.
- Delete (d) – oprávnění mazat soubory a adresáře.
- Administer (a) – povolení pro změnu obsahu ACL tabulky a tedy změnu oprávnění.
- Read (r) – umožnění čtení obsahu souborů a adresářů.
- Write (w) – dovoluje modifikaci souborů .
- Lock (k) – oprávnění pro zamykání souborů například systémovým voláním flock().

Oproti běžnému přístupu k zabezpečení souborových systémů založených na systému Unix (viz. sekce 4.3.1), tak poskytuje ACL v této oblasti vyšší flexibilitu.

Způsobem jakým byl distribuovaný souborový systém AFS navržen, je především pro použití na uzlově objemných sítích a to díky zmíněnému systému centralizovaného ukládání dat na serveru a cache systému jednotlivých klientských uzlů [33]. Soutěžní systém menších rozměrů, tak tyto výhody pravděpodobně nevyužije a je tak pravděpodobně pro tyto účely příliš komplikovaný, byť pokročilejší systém oprávnění by byl naopak do jisté míry žádanější.

## 4.4 Zabezpečení komunikace

Jelikož komunikace distribuovaného řešení soutěžního systému je vedena skrze Internet, je třeba z důvodu jistých rizik zvážit její zabezpečení. Zabezpečení komunikace může nabývat několika forem, ale standardizovaným protokolem pro bezpečnost přenosu dat je uváděn protokol TLS [34], který bude dále v sekci 4.4.1 popsán.

### 4.4.1 TLS

Transport Layer Security (TLS) je široce rozšířeným kryptografickým protokolem navržen tak, aby během komunikace kolující po Internetu usnadnil zajištění soukromí a bezpečnost dat. Běžným případem použití tohoto protokolu je pro zašifrování komunikace webových aplikací a serverů. Bývá často z historických důvodů zaměňován svým již neaktualizovaným protokolovým předchůdcem Secure Sockets Layer (SSL)[34]. TLS zaručuje, že komunikace pomocí tohoto protokolu je šifrovaná a nedojde během ní k porušení integrity [34].

**Certifikát.** Webový portál, server a nebo jakákoliv jiná entita se při komunikaci ověřuje pomocí certifikátu podepsaného certifikační autoritou. Certifikát se vydává pro specifické doménové jméno pod danou organizací, zařízením, nebo uživatelem. Kromě vydané domény se pro certifikát stanovují i související poddomény a expirační čas. Každý certifikát má přidělený soukromý klíč, pomocí kterého dochází k zašifrování a tím také k podepsání zasílané zprávy pomocí asymetrického šifrování. Příjemce této zprávy pak veřejným klíčem certifikátu může zprávu dešifrovat a ověřit si tím, že se jedná o předpokládaného odesílatele [34].

Certifikát může být vytvořen a podepsán i vlastní, jednoúčelovou certifikační autoritou. Takové certifikáty označované jako self-signed obsahují sice všechny informace jako běžný certifikát, ale jsou podepsány soukromým klíčem serveru, který je vytvořil. Takovýto certifikát však není možné protější stranou ověřit, jelikož certifikační autorita v tomto případě není veřejně známá. Obecně se nedoporučuje z důvodu bezpečnosti takovéto certifikáty používat, ale pro účely vývoje a testování, tedy v momentě kdy je uživatel jistý, že se jedná o požadovaný server, může postačit. [34]

**TLS Handshake.**

Proces předávání informací komunikujících entit pomocí protokolu TLS se nazývá TLS Handshake. Během tohoto procesu dochází ke specifikování verze protokolu, šifrovacího algoritmu, ověření identity pomocí veřejného klíče a digitálního podpisu certifikační autority a také vygenerování relačních klíčů určených pro zajištění šifrování následujících zpráv [34].

# Návrh soutěžního systému

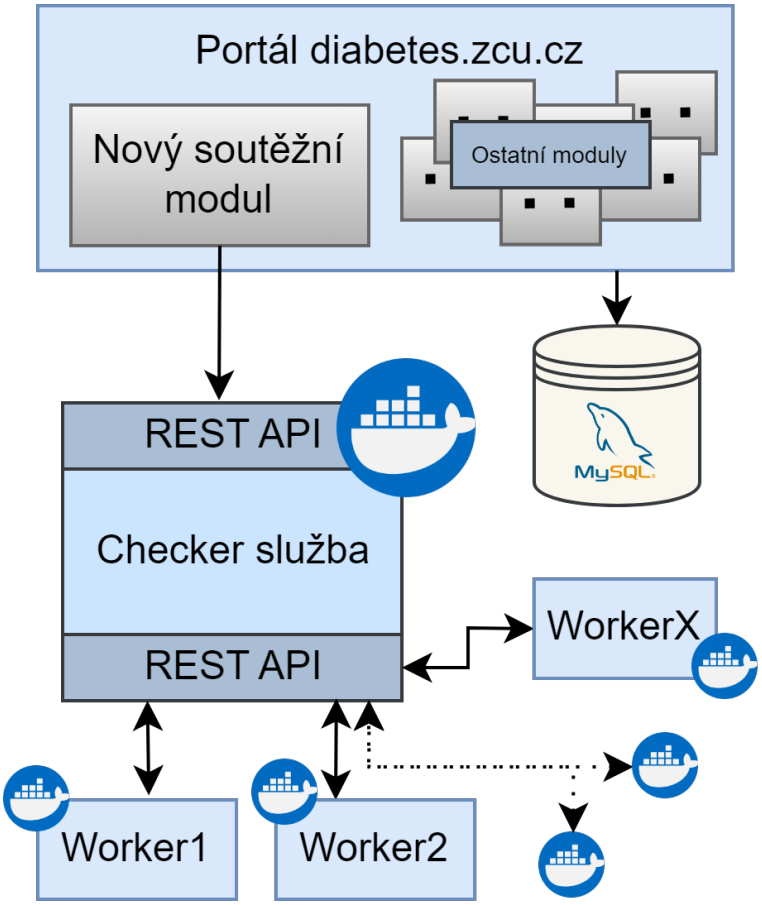
# 5

V této kapitole bude na základě informací z předešlých analytických kapitol navržen soutěžní systém, který umožní pomocí webového rozhraní definovat komplexní soutěže a vyhodnocovat nad nimi řešení soutěžících ve formě zdrojových kódů entit SmartCGMS. V sekci 5.1 bude uvedena kompletní architektura navrhovaného systému, která bude dále v následujících sekcích popsána do větších detailů. První navazující sekce 5.2 se bude následně zabývat, kromě navrhnutí architektury komponenty, také zabývat jednotlivými pohledy webového rozhraní z doménové stránky. Na tyto poznatky bude navazovat sekce 5.3, ve které dojde k navržení samotné soutěžní služby.

## 5.1 Koncept soutěžního systému

Nově navrhovaný systém bude obsahovat několik od sebe oddělených komponent. Jedná se o webové rozhraní diabetes.zcu.cz a soutěžní systém, dále v textu jako Checker služba. Pro sestavování vložených řešení bude Checker služba využívat izolovaných pracovních uzlů ve formě docker kontejnerů, které v následujících sekcích a kapitolách budeme označovat jako Worker. Komponenty takto oddělíme, aby splňovaly architektonický návrhový vzor SOA (viz sekce 2.1), umožňující nasazení Checker služby na oddělený fyzický stroj, čímž docílíme toho, že nebude webové rozhraní brzděno vyhodnocovacím procesem. Implementace soutěžního systému pomocí tohoto vzoru by umožnila komunikaci i s jinými klientskými aplikacemi jako je například mobilní aplikace. Navrhovaná architektura celého soutěžního systému je vyobrazena na obrázku 5.1.

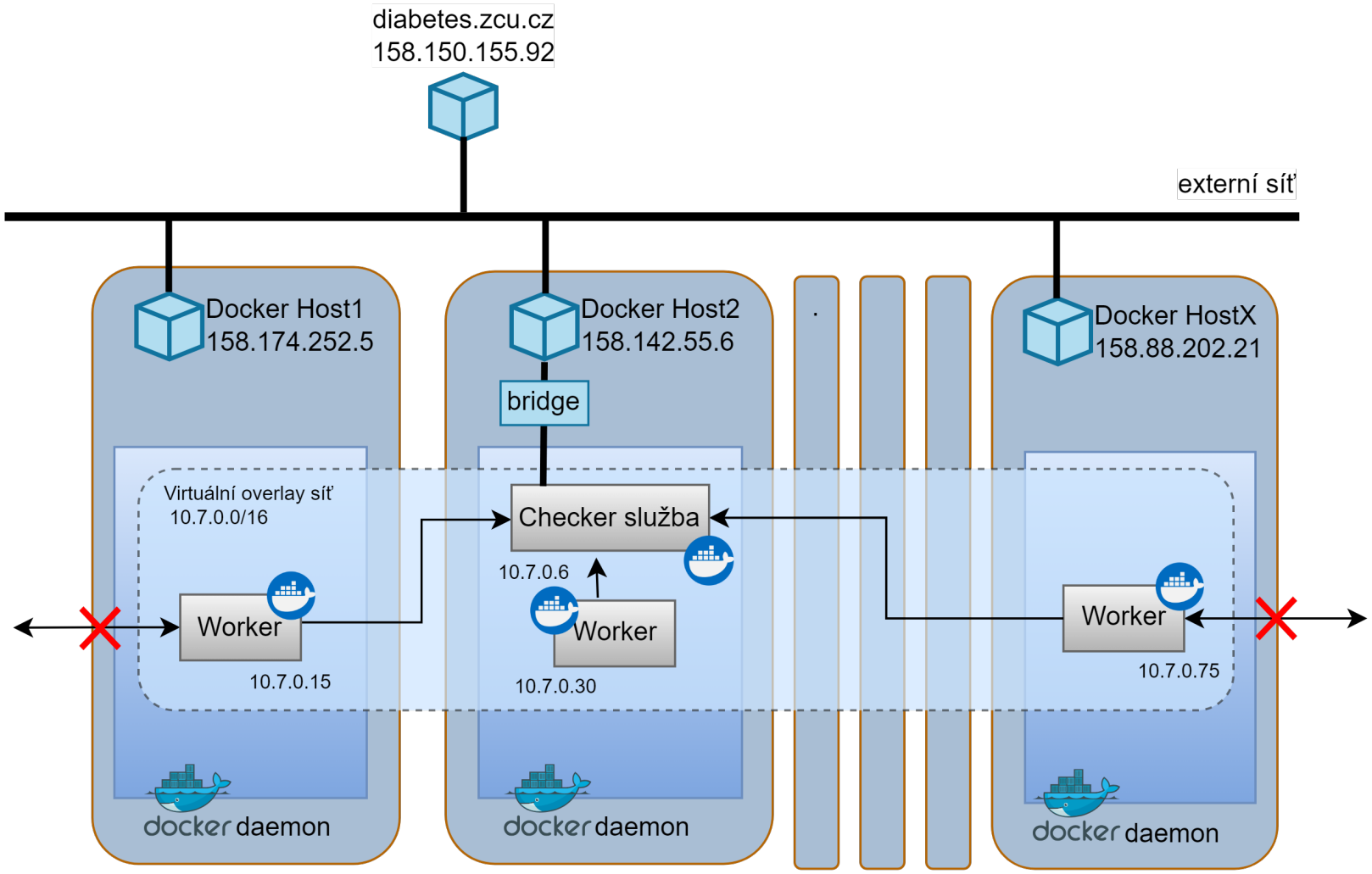
První zmíněnou komponentou celého systému je již existující webový portál diabetes.zcu.cz, což je webová aplikace napsaná ve frameworku Nette a nasazená v reálném prostředí. Pro ukládání potřebných dat využívá portál databázi MySQL. V rámci předmětu OPSWI byl v minulosti autorem této diplomové práce portál také rozšířen o rozhraní klinických studií a správy lékařů, které ale s nově navrhovanou funkcionalitou úzce nesouvisí. V rámci diplomové práce bude tato komponenta




Obrázek 5.1: Architektonický pohled na nově navrhovaný soutěžní systém integrovaný k již existujícímu webovému portálu diabetes.zcu

rozšířena o soutěžní modul (dále v sekci 5.2), který bude zasílat požadavky na REST API [24] Checker služby.

Checker služba bude společně se svými Worker uzly kontejnerizována pomocí nástroje Docker. Aby byla dodržena síťová izolace worker uzlů v distribuovaném prostředí přes několik fyzických uzlů, budou tyto kontejnery omezeny jednou virtuální overlay síťí (viz sekce 4.2.4.2), pomocí které budeme schopni zabránit přímému přístupu do vnější sítě. Jelikož ale potřebujeme, aby Checker služba byla přístupná požadavkům z webového portálu, bude z virtuální docker sítě propojená pomocí virtuálního mostu (viz sekce 4.2.4.2). Síťové rozložení webového portálu a zmíněné Checker služby je možné spatřit na obrázku 5.2. Dalšími detaily, které se týkají čistě návrhu Checker služby se bude tato diplomová práce zabývat v sekci 5.3.




Obrázek 5.2: Návrh síťového rozložení portálu diabetes.zcu.cz a Checker služby přes X oddělených fyzických uzlů spojených skrze overlay virtuální síť

### 5.1.1 Zabezpečení komunikace

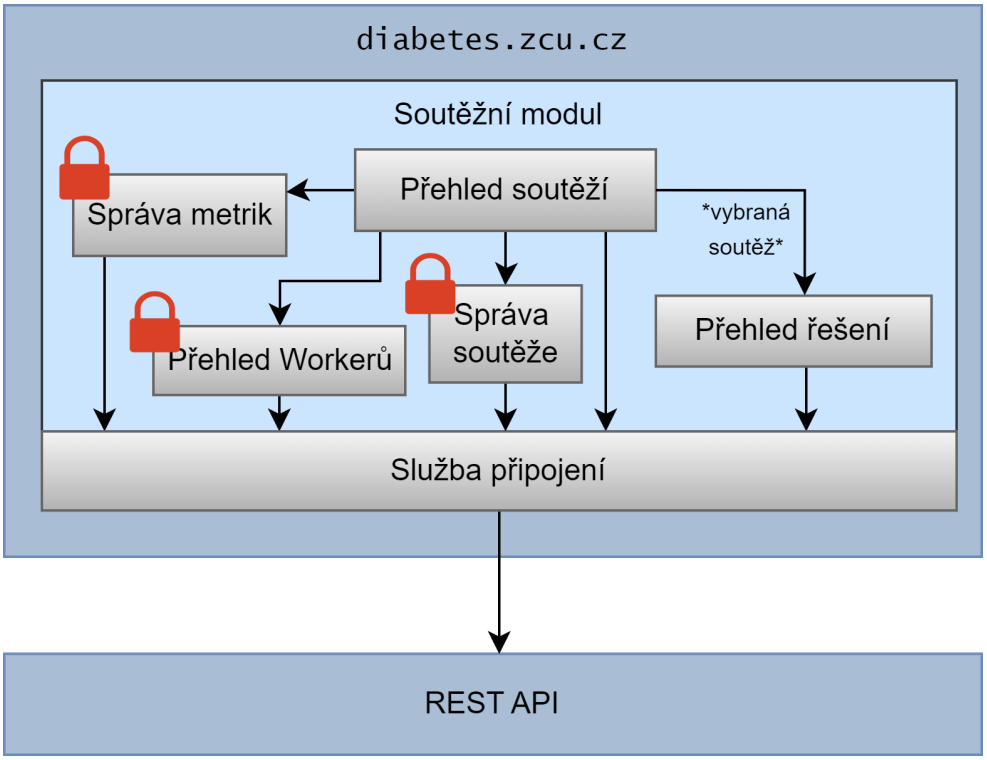
Již existující webový portál má zabudovaný autorizační systém, pomocí kterého ověřuje, zda má uživatel právo přístupu na určité stránky portálu. Tato vlastnost bude využita i u nově navrženého modulu, jelikož správa soutěžního systému může být přístupná pouze uživateli s administrátorským oprávněním (dále v sekci 5.2).

Jelikož Checker služba svůj vlastní autorizační systém mít nebude, bude záviset na webovém portálu, na kterém bude docházet k autorizačnímu procesu. Checker službě tak budou zasílány údaje o uživatelích jako je jejich ID a role. Aby mohl systém věřit, že se při přístupu na API jedná o webový portál, který tyto údaje zasílá, bude autentizován pomocí klientského certifikátu (viz. sekce 4.4). Celá komunikace pak bude zabezpečena pomocí SSL certifikátu.

Jak je znázorněno na obrázku 5.1 Worker uzly se budou dotazovat přes jiné API než webový portál. Aby bylo určeno, zda se jedná o Worker uzel bude také autentizován pomocí vlastního klientského certifikátu (viz. sekce 4.4). Samotná komunikace s Worker uzlem při předávání zdrojových kódu se pak koná v izolovaném síťovém prostředí.

## 5.2 Webový portál diabetes.zcu

Jak již bylo zmíněno v předešlé sekci, webový portál diabetes.zcu je již existující aplikace napsaná ve frameworku Nette, která bude dále rozšířena o rozhraní soutěžního systému. Schéma navrhovaného modulu je zobrazeno na obrázku 5.3, které . V následující sekci budou popsány možné případy použití, na kterých budou založeny jednotlivé pohledy.




Obrázek 5.3: Schéma navrhovaných pohledů soutěžního modulu a síťového konektoru, kde ikonka červeného zámečku označuje administrátorský pohled

### 5.2.1 Přehled Worker uzlů

Checker služba si bude udržovat frontu Worker uzlů (dále sekce 5.3.2). Informace o těchto Worker uzlech budou administrátorovi zobrazeny a poslouží především pro obecný přehled o procesu kompilace na pozadí Checker služby. Také tento pohled poslouží v momentech, kdy dojde v jednom z Worker uzlů k nečekané chybě a pomocí něj tak dokáže administrátor identifikovat kontejner, ve kterém došlo k chybě. Mezi důležité informace můžeme považovat přiřazené ID, interní adresu, aktuální stav, poslední čas komunikace a posledního přiřazeného řešení.

### 5.2.2 Správa a přehled metrik

Systém SmartCGMS bude v momentě vyhodnocování obsahovat množinu předem definovaných metrik, které budou použity během evaluačního procesu (dále v sekci 5.3.4). Administrátor tyto metriky bude mít možnost pomocí webového rozhraní v soutěžním systému zaregistrovat. Metrika bude registrována s unikátním jménem, krátkým popisem její funkce a GUID, kterým se v SmartCGMS označuje. Metriky budou dále odkazovány při zakládání a editaci soutěží.

### 5.2.3 Správa a přehled soutěží

Prvním případem použití dostupného i běžnému uživateli je zobrazení a správa soutěží. Administrátor webového portálu musí mít možnost vytvářet nové soutěže a definovat nad nimi komplexní scénáře, ale v případě nutnosti soutěž i upravovat. Naopak běžný uživatel bude mít možnost ze seznamu zvolit jednu z již definovaných a dostupných soutěží, do kterých byl pozván, a nebo se připojit do jedné z veřejně dostupných.

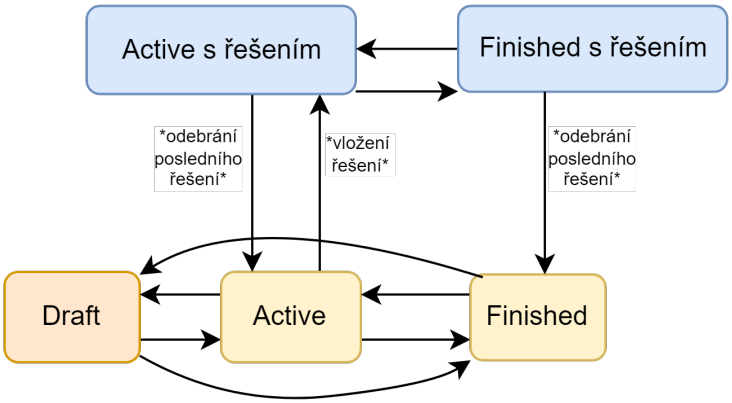
#### 5.2.3.1 Vlastnosti soutěže a její vytvoření

Pro definování soutěže s komplexním scénářem bude muset administrátor vyplnit velké množství údajů. Každá soutěž bude potřeba pojmenovat unikátním jménem a přidat k ní popisek, který může nést informaci o pravidlech soutěže. Aby měl administrátor kontrolu nad tím, zda mají soutěžící povoleno do soutěže přidávat řešení, označíme soutěž jedním ze tří stavů a přidáme datum uzavření, po kterém bude možné prohlížet jenom výsledky. Jedná se o následující stavy:

- Draft – Soutěž se nachází v návrhovém stavu. Je viditelná pouze administrátorům a tak do ní není možné ani přidávat nové řešení.
- Active – Soutěž je v aktivním stavu, je viditelná cílovým uživatelům a dovoluje přidávat nové řešení soutěžícím.
- Finished – Soutěž je označena jako dokončená, není možné do ní přidávat nové řešení a může být zobrazena pouze výsledková listina.

Přechod mezi stavy znázorněný na obrázku 5.4 je neomezený do momentu, do kterého některý ze soutěžících nevloží řešení. Po této události bude možný pouze přechod mezi stavy **Active** a **Finished**. **Draft** bude omezen, aby nedocházelo k omylům, kdy administrátor tímto způsobem soutěž zablokuje.

Administrátor bude mít možnost stanovit seznam pozvaných účastníků z již zaregistrovaných uživatelů v portále diabetes.zcu. Tento seznam bude možné mě-




Obrázek 5.4: Stavový diagram životního cyklu soutěže

nit i během chodu soutěže a pozvaným uživatelům bude při každé změně zaslána pozvánka v emailové formě odkazující na definovanou soutěž.

Důležité je i definování viditelnosti. Soutěž, kterou označíme statusem private, bude viditelná pouze pro skupinu lidí, kteří byli do soutěže pozváni. Naopak status **public** bude značit, že je soutěž veřejně přístupná a bude tedy soutěžícím umožněno vypsat seznam veřejných soutěží, do kterých se bude možné přidat.

Jelikož bude soutěž dovolovat uživatelům vkládat řešení ve formě zdrojových kódů, které se budou následně překládat a vyhodnocovat, je potřeba tyto procesy časově omezit. Toto omezení bude především z důvodu ochrany proti zaseknutí, ale také pro stanovení přijatelné doby chodu. V případě, že by byla soutěž časově kritického charakteru, mohla by být doba chodu také omezena z důvodu časové závislosti, tedy například v momentě, kdy se nejlepší řešení má nasadit do prostředí inzulinové pumpy.

Administrátor vytvářející novou soutěž bude muset vložit vstupní data, která budou použita na vstupu pro zadané řešení soutěžícího. K datům se soutěžící přímo nedostane, protože budou za vrstvou abstrakce vyhodnocovacího systému a proto se může jednat i o utajená data pacientů. Bude vyžadováno, aby administrátor definoval GUID signálu, kterým jsou v zadaném souboru všechny jeho záznamy označeny.

Nad vstupními daty a výsledky z filtru soutěžícího bude v evaluačním procesu (dále v sekci 5.3.4) vyhodnocováno pomocí zvolených metrik. Metriky budou moci být změněny i za chodu soutěže, ale každá taková změna bude vyžadovat nový průchod všech řešení evaluačním procesem.

Součástí definice soutěže budou i konfigurační soubory SmartCGMS (viz sekce 3.3) pro proces kompilace a následné evaluace, které budou do detailu rozebrány v následujících sekcích 5.3.3 a 5.3.4.

#### 5.2.3.2 Detailní pohled a výsledky soutěže

Při zvolení soutěže ze seznamu dostupných soutěží bude uživatel převeden na detailní pohled soutěže. Ta bude zobrazovat základní informace o soutěži jako je její název, popis, stav, GUID referenčního signálu a přiřazené metriky. Tyto informace jsou důležité pro soutěžícího, aby podle nich mohl nastavit své řešení, pokud tak neučiní přes konfiguraci. V detailním zobrazení budou ke spatření i výsledková listina, která bude obsahovat záznamy všech řešení a ponese údaje o jméně soutěžícího, výsledné hodnoty jednotlivých metrik a číslo pokusu.

### 5.2.4 Správa a přehled řešení

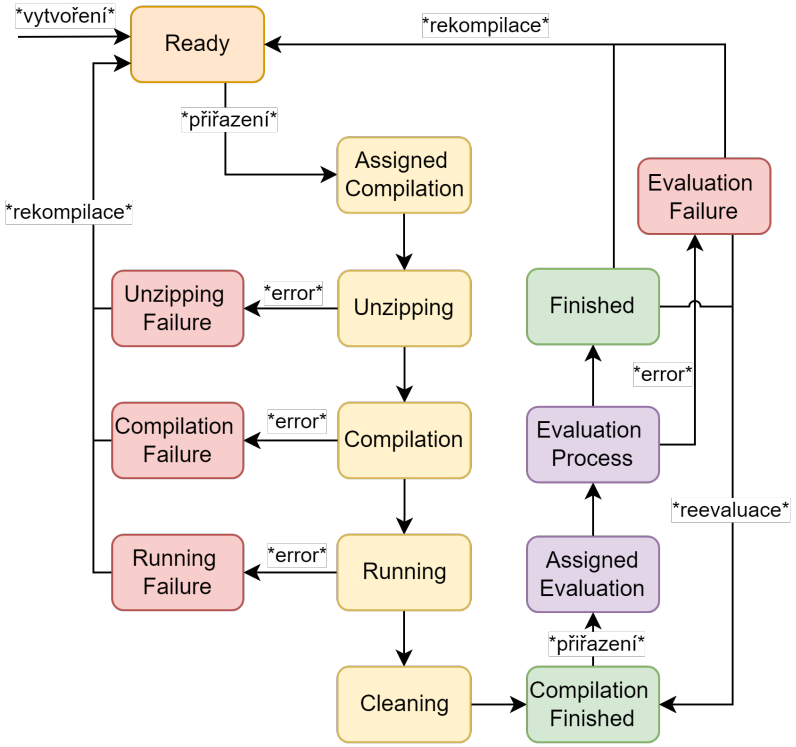
Zásadní částí soutěžního systému je i rozhraní pro přehled a vytváření nových řešení. Administrátor bude mít možnost zobrazit všechna řešení, která byla pro zvolenou soutěž přidána. Běžný uživatel však musí být omezen pouze na své dříve přidané řešení. Běžný uživatel, pokud byl pozván do soutěže a soutěž je ve stavu kdy přijímá nová řešení, by měl mít možnost vložení nového řešení. Vklad nového řešení bude vyžadovat archivní soubor obsahující zdrojové kódy entity SmartCGMS a příslušnou konfiguraci pro sestavení. Také musí obsahovat typ postupu překladu, označující sadu kompilačních příkazů a argumentů, která má být v kompilačním procesu použita. V neposlední řadě musí soutěžící definovat GUID entity SmartCGMS a vstupních parametrů potřebných pro jeho chod, následně vložené do konfiguračního .ini souboru (viz sekce 3.3). Jako poslední parametr vloženého řešení bude GUID výstupního signálu, potřebného pro evaluační proces (dále v sekci 5.3.4).

Vložené řešení bude možné si detailně prohlédnout. Detailní pohled bude obsahovat informace jako je jméno autora, čas vytvoření, poslední čas aktualizace, definované GUID entity SmartCGMS a výstupního signálu. Nesmí chybět ani aktuální stav vyhodnocování vloženého řešení. Řešení bude možné opětovně rekompilovat i reevaluovat v případě, že to aktuální stav dovoluje (viz. obrázek 5.5). Mezi všechny stavy patří následující:

- **Ready** – řešení je vložené do fronty řešení a je připravené na jeho zpracování.
- **Assigned compilation** – řešení bylo předáno zvolenému Worker uzlu a bude zpracováno.
- **Unzipping** – řešení prochází fází extrakce archivu a úpravou adresáře.
- **Unzipping Failure** – během extrakce archivu došlo k chybě. Pravděpodobně kvůli chybnému adresáři, nebo chybějícím konfiguračním souborů.
- **Compilation** – zdrojové kódy prochází kompilací.

- **Compilation Failure** – kompilace zdrojových kódů selhala.
- **Running** – přeložené řešení je spuštěno v SmartCGMS simulaci.
- **Run Failure** – SmartCGMS simulace selhala.
- **Cleaning** – sběr dat a čištění Sandboxu.
- **Compilation Finished** – Worker uzel dokončil kompilační proces.
- **Assigned Evaluation** – Řešení bylo předáno evaluátoru.
- **Evaluation Process** – evaluátor vyhodnocuje výsledky řešení vyhodnotí nad stanovenýma metrikama.
- **Evaluation Failure** – došlo k chybě během evaluačního procesu.
- **Finished** – řešení bylo vyhodnoceno.

Celý stavový diagram zmíněných stavů a jejich přechodů je možné vidět na obrázku 5.5.




Obrázek 5.5: Stavový diagram životního cyklu řešení

Soutěžícímu bude také umožněno stáhnout si kompilační logy, aby mohl v případě chyby kompilace poupravit své řešení tak, aby v dalším pokusu kompilace prošla bez chyby. Soutěžící by určitě ocenil i logy z běhu samotné simulace SmartCGMS, ty ale kvůli riziku odhalení citlivých dat pacientů poskytnuty být nesmí. Po úspěšném sestavení řešení budou zpřístupněny statistiky běhu a po dokončení evaluace i výsledné hodnoty jednotlivých metrik.

Jelikož jsou zdrojové kódy autorským dílem, musí mít soutěžící možnost své řešení kdykoliv odstranit, pokud si to přeje. Z důvodu bezpečnosti bude však nejdříve kontaktován administrátor webového portálu formou e-mailu, který poté smazaní dokončí. Během doby, kdy bude řešení označené k smazání, nebude zobrazováno ve výsledkové listině a ani nebude zpracováváno Checker službou.

V případě, že bude potřeba aby vložené řešení prošlo kompilačním nebo evaluačním procesem znovu, bude administrátorovi pomocí tlačítka umožněno tyto procesy zopakovat.

### 5.2.5 Služba připojení

Pohledy zmíněné v předešlých sekcích budou závislé na datech získaných ze vzdálené Checker služby. Pro unifikování přístupu k této službě budou jednotlivé pohledy přistupovat přes jednotné rozhraní ve formě konektoru. Tímto způsobem zajistíme nejen oddělení logiky konektoru, ale také nám to umožní mít kontrolu nad datovým tokem.

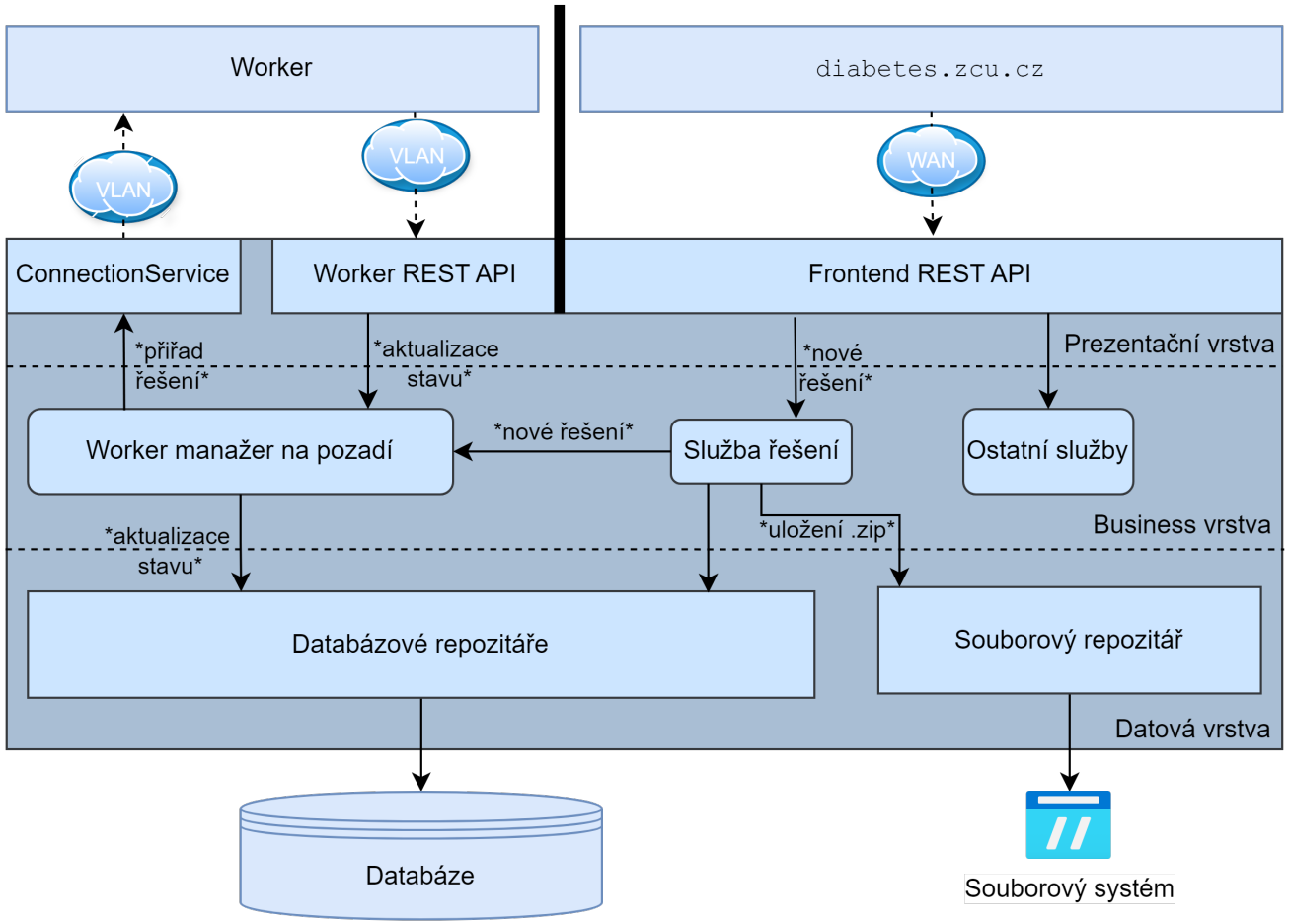
## 5.3 Checker služba

Nová aplikace také nazvaná jako Checker služba obsluhující soutěžní systém bude nasazena na oddělený stroj od stroje spravující webový portál, abychom zamezili jeho případné blokaci. Checker bude poskytovat API pro vytváření a správu soutěží a dalších entity, které byly již popsány v sekci 5.2. Data z těchto entit bude ukládat do zvolené databáze a vložené a vygenerované soubory do administrátorsky přístupného souborového systému (dále v sekci 5.3.5), aby v případě nečekané chyby vyhodnocení mohl administrátor přistoupit ke všem důležitým souborům vzniklých během samotného vyhodnocení.

Na pozadí Checker aplikace bude spuštěn jak manažer pro správu Worker uzlů zabývající se kompilačním procesem (dále v sekci 5.3.2), tak i Evaluátor určen pro vyhodnocovaní výsledků řešení během evaluačního procesu (dále sekce 5.3.4).

### 5.3.1 Architektura Checker služby

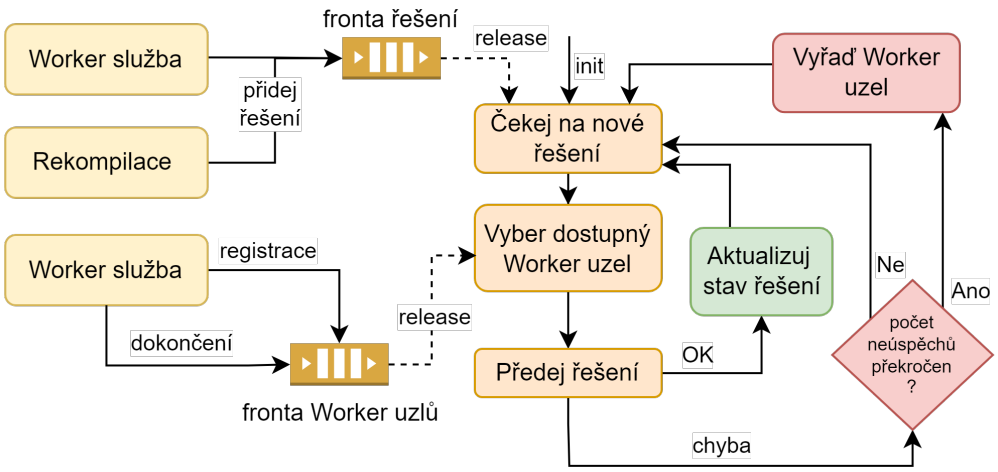
Vše bude integrováno do podoby třívrstvé architektury [35], aby došlo oddělení funkcí zpracování dat od funkcí pro ukládání a prezentaci. Každá vrstva tak bude mít vlastní sadu objektů, které nebudou mezi vrstvami sdílené. Toto rozhodnutí do budoucna umožní poměrně lehkou změnu na datové vrstvě systému, v případě, že se zvolená databázová nebo souborová technologie ukáže jako nedostačující.




Obrázek 5.6: Návrh třívrstvé architektury Checker systému z pohledu vytváření nových řešení obecné soutěže

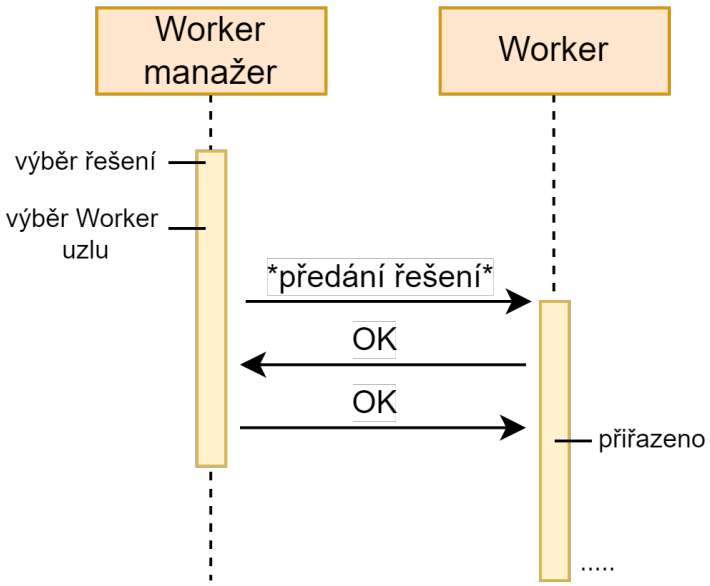
## 5.3.2 Worker manažer

Worker manažerem označujeme proces na pozadí Checker služby, obsluhující přiřazování zdrojových kódů (řešení) určených pro kompilaci na Worker uzlech. Součástí manažeru nesmí chybět fronta řešení, udržující si řešení určených pro kompilaci a frontu Worker uzlů, kteří budou schopní řešení zpracovat. Řešení bude do fronty vloženo při jeho vytvoření a nebo při příkazu rekompilace. Naopak do fronty Worker uzlů bude přidáváno registrací jednotlivých uzlů a nebo při jejich dokončení. Po vybrání zadaného řešení a dostupného Workera dojde k zaslání zdrojových kódů a metadat řešení příslušnému Worker uzlu. V momentě kdy je Worker uzel úspěšně dokončen, je zpracované řešení předáno pro evaluační proces (dále v sekci 5.3.4). Celý tento proces aktivit je k zahlédnutí na obrázku 5.7.




Obrázek 5.7: Diagram aktivit Worker manažera při přiřazování řešení zvolenému Worker uzlu

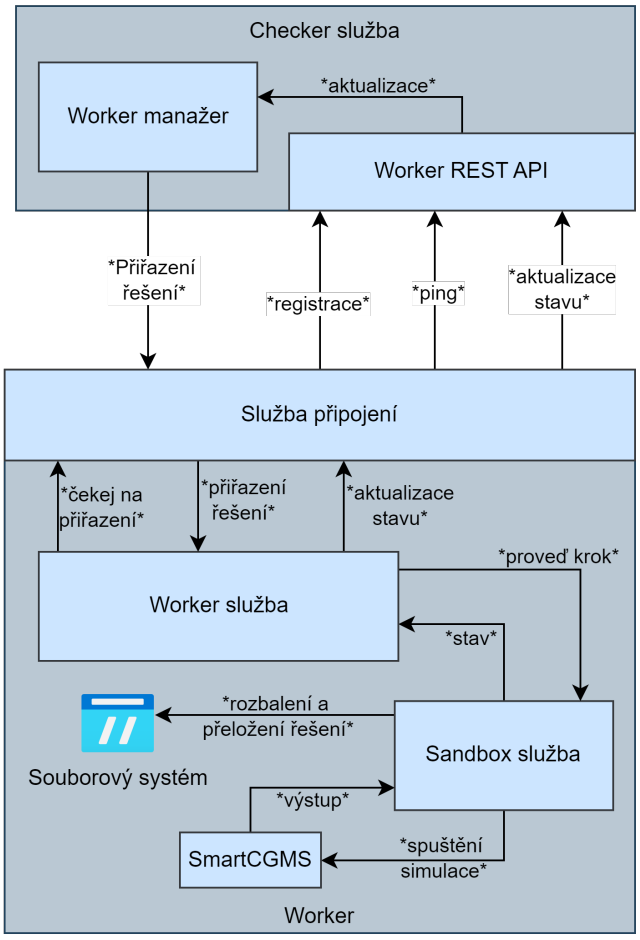
Aby bylo zajištěno, že se řešení ve formě zdrojových kódů předalo úspěšně, bude očekávána potvrzovací zpráva (viz obrázek 5.8).




Obrázek 5.8: Navržené komunikační schéma Worker manažera a Worker uzlu při předávání řešení

### 5.3.3 Kompilační proces

Vložené řešení bude po přiřazení příslušnému Worker uzlu procházet kompilačním procesem. Kompilační proces je označení průchodu uvnitř Worker uzlu specializované na přeložení zdrojových kódů entity SmartCGMS a jeho následné spuštění ve formě simulace. Podobně jako tomu je u webového portálu, bude funkcionalita oddělená konektorem, zodpovědného za komunikaci s Checker službou. Mezi jeho funkce patří registrace, průběžná ping zpráva a aktualizace stavu kompilačního procesu. Celé schéma navrhovaného toku dat je zobrazeno na obrázku 5.9.

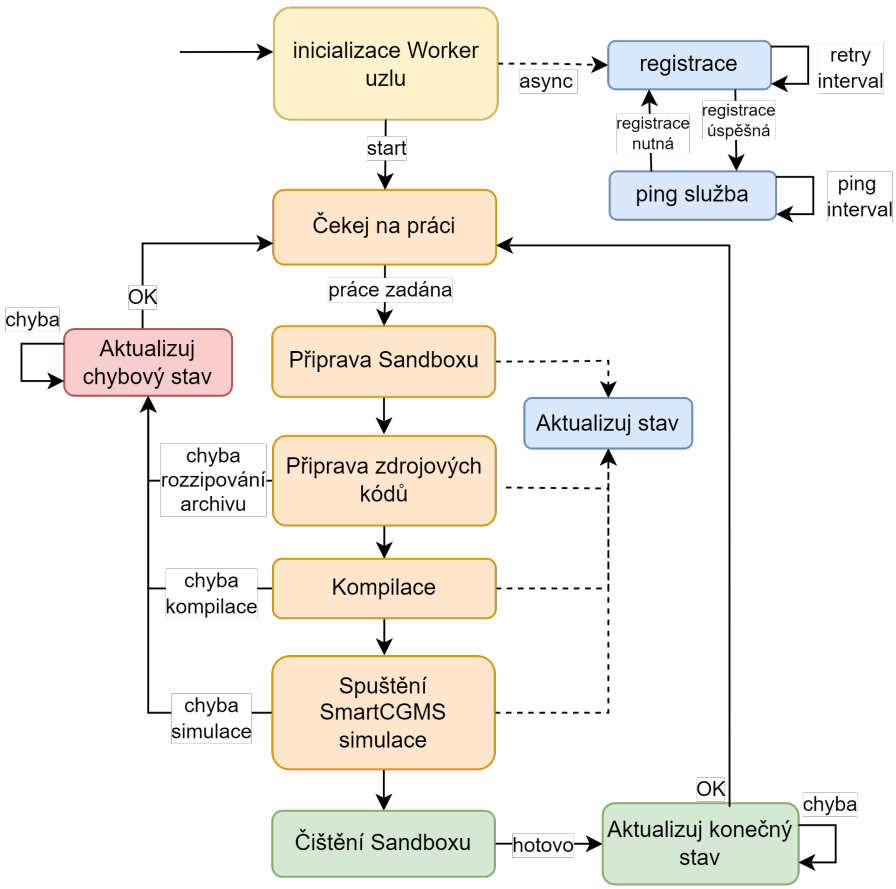



Obrázek 5.9: Navržené schéma toku dat Worker uzlu

Kompilační proces bude během své činnosti procházet několika stavy (viz obrázek 5.10). Jmenovitě se jedná o následující:

- **Čekání na práci** – Worker uzel v tomto stavu čeká na přiřazení řešení Worker manažerem.
- **Příprava sandboxu** – tento stav založí Sandbox adresář nad kterým bude dále spuštěna kompilace a samotná simulace.
- **Příprava zdrojových kódů** – stav, kdy bude docházet ke kontrole adresářové struktury vložených zdrojových kódů, kde bude ověřena i přítomnost konfiguraci pro sestavení.
- **Kompilace** – kompilační stav, kdy dochází k překladu zdrojových kódů entity SmartCGMS a vytváření spustitelné knihovny. Také během tohoto stavu budou vygenerované uživatelsky přístupné kompilační logy.

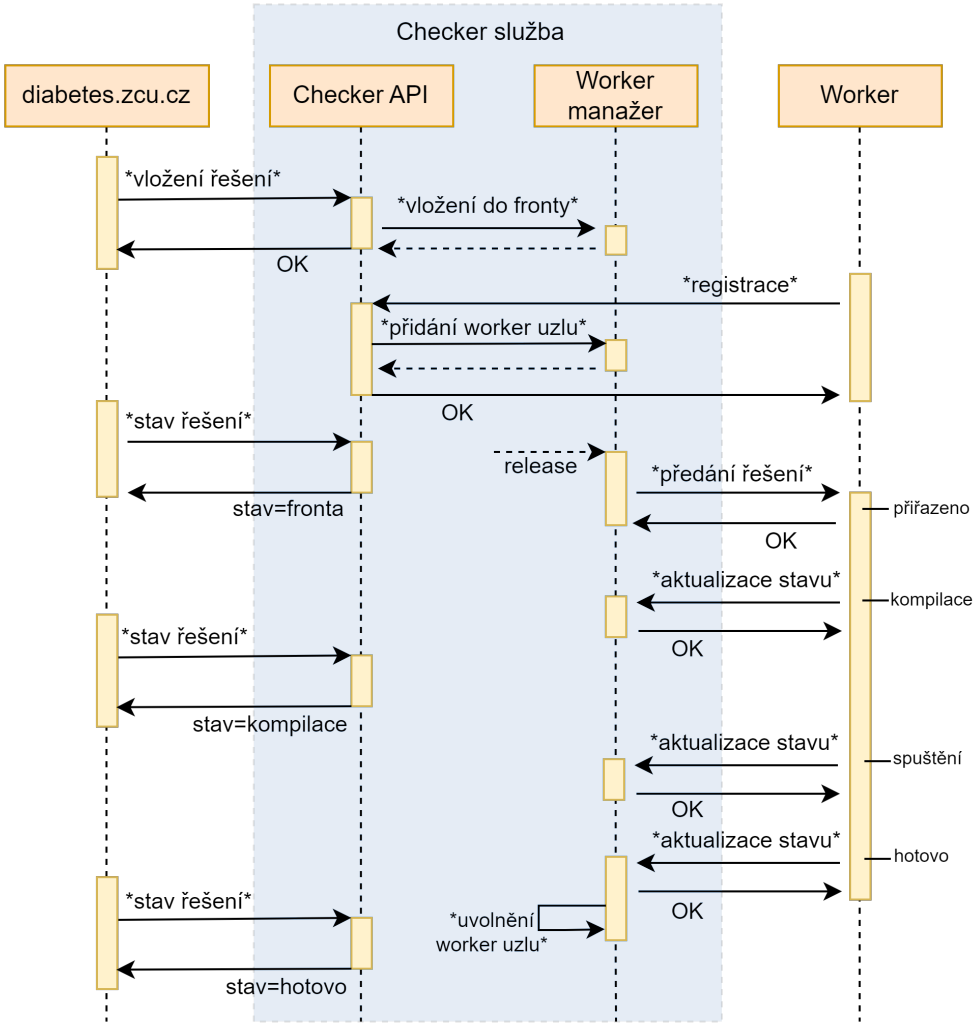
- **Spuštění SmartCGMS simulace** – stav, kdy dochází při spuštěné SmartCGMS simulaci ke konzumaci vstupních dat a ke generování výsledků, potřebných pro evaluační proces.
- **Čištění Sandboxu** – stav, kdy dochází ke sběru dat vygenerovaných při simulaci a smazání použitého Sandboxu.



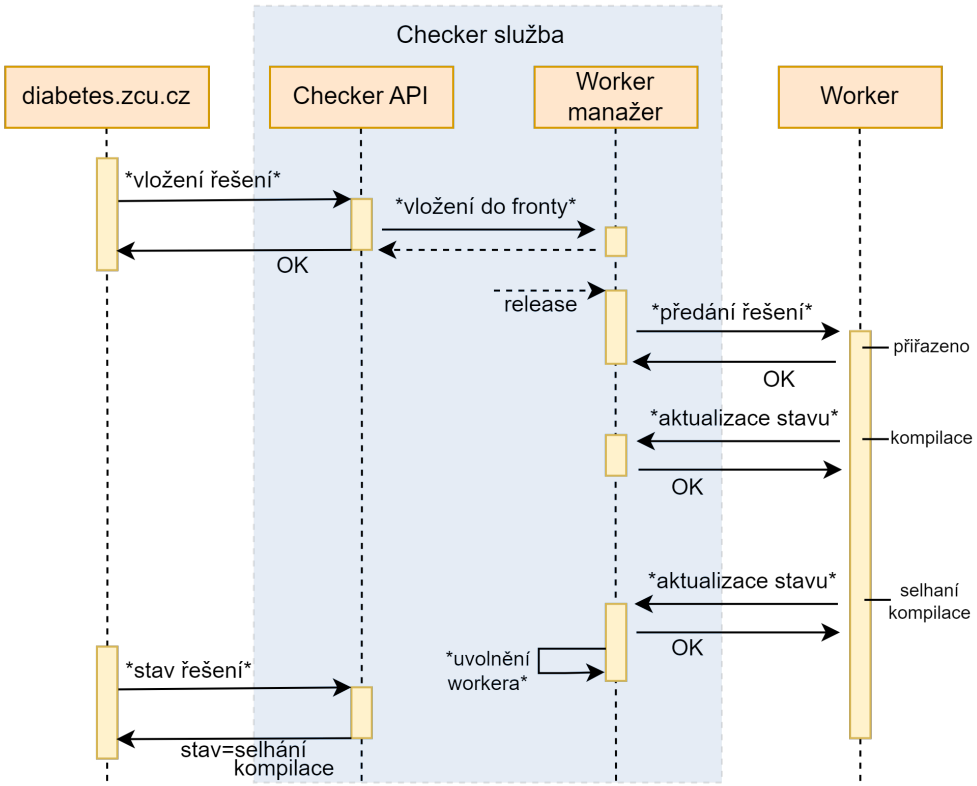

Obrázek 5.10: Diagram aktivit Worker uzlu před a během přiřazení řešení ve formě zdrojových kódů

Při každé změně stavu bude Checker službě přes REST API zaslána aktualizační zpráva, která poté poslouží uživateli ke sledování průběhu svého řešení. Také budou při aktualizaci zasílány statistiky běhu, které budou měřit čas jednotlivých průběhů a paměťové náročnosti.

Celé komunikační schéma zobrazující vytvoření řešení, registraci Worker uzlu, předání zadaní Worker manažerem a samotného kompilačního procesu, je ke spatření na obrázku 5.11. Pokud během kompilačního procesu dojde k nějaké chybě, je stejným způsobem aktualizován i chybový stav (viz obrázek 5.12).



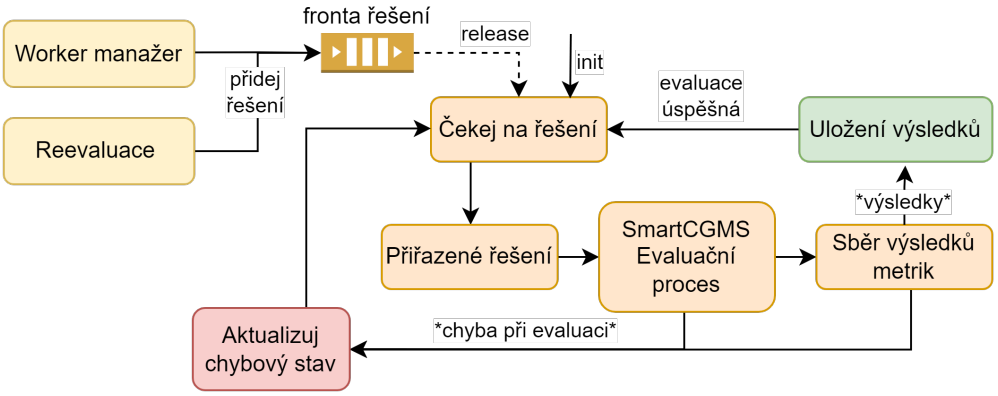

Obrázek 5.11: Komunikační schéma celého systému při vytvoření nového řešení, jeho zpracování a následného dotazování jeho stavu




Obrázek 5.12: Komunikační schéma celého systému při vytvoření nového řešení, u kterého dojde k selhání během kompilační fáze

### 5.3.4 Evaluační proces

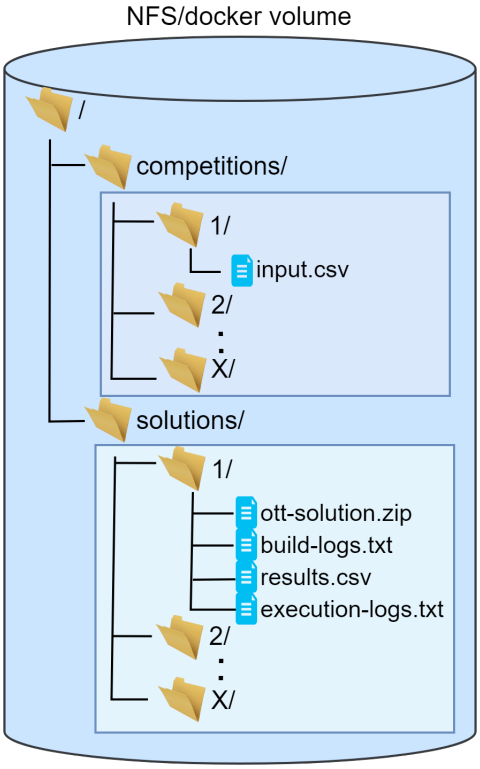
Po dokončení kompilačního procesu a přijmutí aktualizační zprávy Worker manažerem následuje evaluační proces. Evaluační proces je označení průchodu obsluhované službou na pozadí známou jako Evaluátor. Evaluátor bude aktivován přidáním řešení do interní fronty Worker manažerem, nebo příkazem pro reevaluaci řešení. Následně Evaluátor, který je součástí Checker služby, bude sloužit k připravení a spuštění simulace SmartCGMS za účelem ohodnocení výstupních dat. Výstupní data, která byla vygenerována během kompilačního procesu, budou ohodnocena pomocí soutěží stanovených metrik dostupných během simulace SmartCGMS. Po dokončení simulace musí následovat sběr podobně jako tomu je u kompilačního procesu. Diagram aktivit Evaluátoru je k představuje obrázek 5.13.




Obrázek 5.13: Diagram aktivit Evaluátoru před a během přiřazení řešení určeného pro vyhodnocení

### 5.3.5 Struktura souborového úložiště

Jak již bylo v předešlých sekcích zmíněno, během kompilačního procesu, vytváření soutěží a vkládání řešení bude využíváno datového úložiště pro ukládání jak zdrojových kódů, vstupních a výstupních dat, tak ale i log souborů. Datové uložiště, v našem případě jako síťový souborový systém napojený uvnitř Docker kontejnerů jako volume, bude potřeba strukturovat takovým způsobem, aby mohl být čitelný i administrátorem. Pro tyto účely se nabízí rozdělení adresářů na podadresáře `competitions` a `solutions`. Ty pak budou dále obsahovat podsložky, které budou pojmenovány přiřazeným ID číslem řešení nebo soutěže. Navrhovaná struktura je zobrazena na obrázku 5.14.




Obrázek 5.14: Návrh struktury síťového adresáře připojeného v Docker kontejneru jako volume

# Implementační detaily soutěžního systému

# 6

V této kapitole dojde k popsání implementačních detailů navrženého soutěžního systému, který byl architektonicky oddělen do tří komponent. Oproti návrhu bude nejdříve uveden samotný Checker systém, po kterém bude následovat webový portál diabetes.zcu, především z logické a chronologické návaznosti.

## 6.1 Checker systém

Checker komponenta společně s její sdruženou Worker komponentou představující Checker systém, jsou implementovány do podoby C# aplikace využívající framework ASP.NET. Tyto komponenty jsou zaštítěni pomocí Microsoft Visual Studio řešení, které je rozděleno na tři projekty, jmenovitě *CompetitionCheckerService*, *CompetitionCheckerWorker* a *Shared*. *CompetitionCheckerService* je projekt pro řešení veškeré správy soutěžního systému. Dále spravuje a zprovozňuje Worker manažera (dále v sekci 6.1.3) a Evaluátor (dále v sekci 6.1.4). Projekt *CompetitionCheckerWorker* obstarává logickou část kompilačního procesu dále rozebraného v sekci 6.1.3) a je oddělen od Checker služby především z důvodu umožnění jeho horizontálního škálování. Poslední zmíněný projekt *Shared* obsahuje sdílené objekty a funkce, které jsou použity při komunikaci Checker komponenty s Worker komponentou. Tento projekt tak umožňuje zmíněným komponentám pracovat nad stejnou sadou objektů za pomocí serializace a deserializace.

Podle návrhu byla v Checker komponentě dodržena třívrstvá architektura, oddělující funkce zpracování dat, jejichž objekty jsou v implementovaném řešením výhradně v podadresáři *Services*. Prezentační vrstva je pak obsažena v podadresáři *Controllers* (dále v sekci 6.1.1) a datová vrstva v *Repository* (dále v sekci 6.1.1). Vrstvy mezi sebou pro zachování oddělení jednotlivých vrstev nesdílí datové objekty, kde tak při přechodu každé vrstvy dochází k procesu mapování pomocí nástroje *AutoMapper*[36] konfigurovatelný třídou *MappingProfile*. Tímto způsobem je pojištěn

případ, kdy jedna vrstva potřebuje pracovat s daty v podobě objektů jinak než druhá vrstva. Dále z architektonického pohledu bylo pro oddělení závislostí implementovaných tříd využito návrhového vzoru Dependency injection [37] tak, že byla . Toto rozhodnutí tak zjednodušuje proces otestování pomocí unit testů a výměnu jednotlivých implementací komponent.

Protože jsou logy v oblasti distribuovaných řešení velmi důležité, je celá Checker služba s Worker uzlem obalená logovacími třídami, které vytváří log zprávy během průběhu důležitých algoritmů a přístupů. Pro tyto účely je využito knihovny Serilog [38], umožnující zapisování následujících úrovní: *Verbose*, *Debug*, *Information*, *Warning*, *Error* a *Fatal*.

### 6.1.1 Prezentační vrstva - REST API

Checker služba poskytuje pro ovládání funkcí systému přístupové body HTTP REST API. Aby mohly být tyto přístupové body, také označované jako endpointy, webovou aplikací použity, vyžadují aplikací registrovaný klientský certifikát. Pokud se komponenta při síťové komunikaci bude ověřovat certifikátem, jenž je specifický pro Worker uzly, bude mít přístup pouze k endpointům sloužících pouze pro správu Worker uzlů zmíněných v sekci 6.1.1.4. Doplněk těchto endpointů je naopak přístupný pro jakýkoliv jiný registrovaný klientský certifikát.

Jelikož nemá Checker služba implementovanou žádnou vlastní autorizační metodu, je závislá na uživatelských datech poskytnutých webovým portálem diabetes.zcu. Při používání jakéhokoliv endpointu webovou aplikací je vyžadováno, aby ke každému HTTP požadavku byly přidány dvě hlavičky *UserId* a *IsAdmin*. Tyto hlavičky ponesou údaj o identifikačním čísle a administrátorským oprávnění uživatele.

Jakýkoliv požadavek na toto rozhraní musí být proveden s ohledem na rychlost odpovědi. Proto všechny endpointy, které vyžadují nějakou náročnou akci na pozadí, jsou odděleny na počáteční požadavek a požadavek na stav akce. Příkladem je například požadavek pro vytvoření řešení, během kterého by v opačném případě musel vyčkávat na celý vyhodnocovací proces. Dalším příkladem je i aktualizace stavu Worker uzlu, která by tak ve Worker manažeru byla blokována synchronizačními akcemi.

Všechny endpointy, u kterých je vyžadováno vlastnit administrátorská práva, navrací HTTP stavový kód **409 Unauthorized**. Většina endpointů vracející data má v hlavičce odpovědi nastavený **Content-type** na `application/json`.

V následujících sekcích budou popsány jednotlivé *Controller* třídy, jejich endpointy a funkce z pohledu soutěžního systému. Celé REST API Checker služby představuje prezentační vrstvu třívrstvé architektury.

#### 6.1.1.1 CompetitionsController

Skupina endpointů sloužící pro správu soutěží.

##### GET /api/Competitions.

Endpoint slouží pro získání seznamu soutěží, kterých je uživatel součástí. Také jsou v odpovědi zaslány i soutěže, které jsou veřejně dostupné a může se jich po přidání zúčastnit. Pokud se jedná o uživatele s administrátorskými právy jsou zaslány všechny soutěže nezávisle na jejich viditelnosti a stavu. Očekávaná struktura odpovědi je následující:

```
{
    "UserCompetitions":[
        {
            "Id":int,
            "Name":string,
            "Description":string,
            "State":int,
            "Visibility":int,
            "DueDateTime":DateTime,
            "CreationTime":DateTime,
            "CreatorId":int
        },
        .....
    ],
    "OpenCompetitions":[
        ....
    ]
}
```

##### PUT /api/Competitions.

Endpoint slouží pro vytváření nových soutěží. Předané data jsou validována a přenesena do dalších vrstev, kde jsou uložena do databáze a vložený vstupní log soubor uložen do souborového systému. Po celém procesu vytvoření je navráceno číslo pro identifikaci nově vložené soutěže. V případě, že došlo v procesu validace k chybě je navrácen HTTP kód 400 a nebo 409 v případě, že soutěž se zadaným názvem již funguje. K této funkci má přístup pouze administrátor. Očekávaná struktura vstupních formulářových dat je následující:

```
{
    "Name": string,
    "Description": string,
    "State": int,
    "Visibility": int,
    "DueDateTime" DateTime,
    "Metrics": string?, //nepovinná proměnná
```

```
    "AssignedCompetitors": string?, //nepovinná proměnná
    "SCGMSCompilationIniDefinition": string,
    "SCGMSEvaluationIniDefinition": string,
    "CompilationLimit": int,
    "ExecutionLimit": int,
    "InputLogFile": IFile,
    "InputLogFileName": string,
    "ReferenceSignalGuid", string
}
```

### POST /api/Competitions.

Endpoint slouží pro aktualizaci informací o soutěži uložených v databázi. Pokud soutěž, kterou se uživatel snaží aktualizovat v systému neexistuje, je mu navrácen stavový HTTP kód 404. Očekávaná struktura je stejná jako u HTTP metody PUT, ale tentokrát bez vloženého vstupního souboru a s parametrem *Id* navíc. K této funkci má přístup pouze administrátor.

### GET /api/Competitions/{competitionId}.

Endpoint slouží pro získání všech konfigurovatelných proměnných informací o soutěži. Pokud žádaná soutěž v systému neexistuje je navrácen HTTP stavový kód 404. K této funkci má přístup pouze administrátor. Očekávaná struktura výstupních dat je následující:

```
{
    "Id": int,
    "Name": string,
    "Description": string,
    "State": int,
    "Visibility": int,
    "DueDateTime" string?, //nepovinná proměnná
    "Metrics" string, //příklad "[5,6,7]"
    "AssignedCompetitors": string, //příklad "[5,6,7]"
    "SCGMSCompilationIniDefinition": string,
    "SCGMSEvaluationIniDefinition", string,
    "CompilationLimit", int,
    "ExecutionLimit", int,
    "ReferenceSignalGuid", string,
}
```

### DELETE /api/Competitions/{competitionId}.

Endpoint slouží pro odstranění soutěže identifikované číslem *competitionId* ze systému. K této funkci má přístup pouze administrátor.

### GET /api/Competitions/{competitionId}/results.

Endpoint slouží pro získání základních informací o zvolené soutěži, přiřazených metrik, přiřazených soutěžících a výsledkovou listinu, která obsahuje výsledky vyhodnocení z evaluačního procesu. Očekávaná struktura výstupních dat je následující:

```
{
    "Name": string,
    "Description": string,
    "State": int,
    "Visibility": int,
    "AssignedMetrics" [
        {
            "Id": int,
            "Name": string,
            "Description": string,
            "Guid": string
        },
        ....
    ],
    "AssignedCompetitors": [
        5, 6, .....
    ],
    "ScoreBoard": [
        {
            "UserId": int,
            "Result": {
                "<metric-id>": double,
                "<metric-id>": double,
                ....
            }
        },
        .....
    ],
    "ReferenceSignalGuid": string
}
```

Tag <metric-id> v ukázce je nahrazen identifikačním číslem metriky.

### GET /api/Competitions/{competitionId}/core.

Endpoint slouží pro získání základních informací o zvolené soutěži. Očekávaná struktura výstupních dat je následující:

```
{
    "Id": int,
    "Name": string,
    "Description": string,
    "State": int,
    "Visibility": int,
    "DueDateTime" string?, //nepovinná proměnná
    "CreationTime": string,
    "CreatorId": string,
    "AssignedUsers", string
}
```

#### POST /api/Competitions/{competitionId}/join.

Endpoint slouží pro připojení uživatele k soutěži identifikované číslem *competitionId*. Na pozadí je ověřeno, že se doopravdy může přidat, tedy že je soutěž viditelná, v aktivním stavu a nedošlo k vypršení časového limit. Pokud tomu tak není je navrácen HTTP stavový kód **403 Forbidden**. V případě, že je uživatel už připojen je navrácen stavový kód **204 No Content**.

### 6.1.1.2 MetricsController

Skupina endpointů sloužící pro správu SmartCGMS metrik.

#### GET /api/Metrics.

Endpoint slouží pro vypsání všech dříve definovaných metrik uložených v databázi. K této funkci má přístup pouze administrátor. Předpokládaný výstup je následovný:

```
[
   {
    "Id": int,
    "Name": string,
    "Description": string,
    "Guid": string,
    "CreatedDateTime": DateTime
   },
   ...
]
```

#### PUT /api/Metrics.

Endpoint slouží pro vytvoření nové metriky. K této funkci má přístup pouze administrátor. Předpokládaný vstup formulářových dat je následovný:

```
{
    "Name": string,
    "Description": string,
    "Guid": string
}
```

#### GET /api/Metrics/{competitionId}.

Endpoint slouží pro získání všech metriky, které jsou přiřazené pod soutěží identifikované pomocí *competitionId*. K této funkci má přístup pouze administrátor. Očekávaný výstup je následující:

```
[
   {
    "Id": int,
    "Name": string,
    "Description": string,
    "Guid": string,
    "CreatedDateTime": DateTime
   },
   ...
]
```

#### GET /api/Metrics/{metricId}/detail.

Endpoint slouží pro získání metriky identifikované pomocí *metricId*. K této funkci má přístup pouze administrátor. Očekávaný výstup je následující:

```
{
    "Id": int,
    "Name": string,
    "Description": string,
    "Guid": string,
    "CreatedDateTime": DateTime
}
```

### 6.1.1.3 SolutionsController

Skupina endpointů sloužící pro správu řešení.

#### GET /api/Solutions/{competitionId}.

Endpoint slouží pro získání všech řešení, které jsou přiřazené pod soutěží identifikované pomocí *competitionId*. V případě běžného uživatele jsou zaslány pouze řešení, které zadal a v opačném případě tak navrací všechny záznamy. Očekávaný výstup v těle odpovědi je následující:

```
[
    {
        "Id": int,
        "CompetitionId": string,
        "CompetitorId": string,
        "State": int,
        "CompilationApproach": int,
        "FileName" string?, //nepovinná proměnná
        "FilterGuid" string, //příklad "[5,6,7]"
        "OutputGuid": string, //příklad "[5,6,7]"
        "Parameters": {
            "key": "value"
            ....
```

```
        },
        "WorkerId", long?, //nepovinná položka
        "CreationTime", DateTime,
        "LastUpdateTime", DateTime,
        "AttemptNumber", int,
        "DeleteFlag", bool
    },
    ....
]
```

### GET /api/Solutions/{solutionId}/result.

Endpoint slouží pro získání všech informací o vloženém řešení identifikovaného pomocí *solutionId*. Mezi tyto informace patří základní metadata o samotném řešení, ale i statistiky běhu kompilačního procesu a výsledky získané během evaluačního procesu. Přístup k těmto informacím je poskytnut jenom uživatelům, kteří řešení zadali a všem administrátorům. Pokud se jedná o nepověřenou osobu je navrácen HTTP stavový kód **404 Not Found**. Očekávaný výstup v těle odpovědi je následující:

```
{
    "Solution":{
        "Id": int,
        "CompetitionId": string,
        "CompetitorId": string,
        "State": int,
        "CompilationApproach": int,
        "FileName" string?, //nepovinná proměnná
        "FilterGuid" string,
        "OutputGuid": string,
        "Parameters": {
            "key": "value"
            ....
        },
        "WorkerId", long?, //nepovinná položka
        "CreationTime", DateTime,
        "LastUpdateTime", DateTime,
        "AttemptNumber", int,
        "DeleteFlag", bool
    },
    "RunStatistics": RunStatistics,
    "Result": MetricResult
}
```

### GET /api/Solutions/{solutionId}/download/build-logs.

Endpoint sloužící pro stažení výstupních logů vytvořeného během kompilace. Content-type v hlavičce odpovědi je v tomto případě nastaven na "application/octet-

stream"a podporuje HTTP stavový kód **206 Partial Content** v případě většího souboru. Pokud soubor neexistuje je navrácen HTTP stavový kód **404 Not Found**.

### GET /api/Solutions/{solutionId}/download/file.

Endpoint sloužící pro stažení archivovaných zdrojových souborů vložených během vytváření nového řešení. Content-type v hlavičce odpovědi je v tomto případě nastaven na "application/octet-stream"a podporuje HTTP stavový kód **206 Partial Content** v případě většího souboru. Pokud soubor neexistuje je navrácen HTTP stavový kód **404 Not Found**.

### PUT /api/Solutions.

Endpoint slouží pro vytváření nového řešení. Vstupní data definující řešení jsou validována a metadata následně uložena do databáze, vložený archiv obsahující zdrojové kódy je následně uložen na souborový systém a celé řešení je vložené do fronty Worker manažera, kde bude následně zpracován. Očekávaná vstupní data v těle požadavku jsou následující:

```
{
    "CompetitionId": int,
    "CompilationApproach": int,
    "ZippedSolution": binary,
    "FilterGuid": string,
    "OutputGuid": string,
    "Parameters": string,
    "FileName": DateTime
}
```

### DELETE /api/Solutions/{solutionId}.

Endpoint umožnující administrátorovi odstranění řešení identifikované pomocí *solutionId* jak z databáze, souborového systému, tak i z celého procesu vyhodnocování. Pokud o toto odstranění zažádá běžný uživatel tedy vlastník řešení, je pouze označeno k odstranění, ale již se nevyskytuje ve výsledkových listinách a ve vyhodnocovacím procesu. Pokud řešení není v systému nalezeno je navrácen HTTP stavový kód **404 Not Found**.

### POST /api/Solutions/recompile/{solutionId}.

Endpoint umožnující administrátorovi znovu provedení kompilačního i evaluačního procesu. Použitím této funkce je řešení identifikované pomoci *solutionId* vloženo do fronty řešení vedené Worker manažerem. Pokud řešení není v systému nalezeno je navrácen HTTP stavový kód **404 Not Found**.

### POST /api/Solutions/reeevaluate/{solutionId}.

Endpoint umožnující administrátorovi znovu provedení evaluačního procesu. Použitím této funkce je řešení identifikované pomocí *solutionId* vloženo do fronty Evaluátoru. Pokud řešení není v systému nalezeno je navrácen HTTP stavový kód **404 Not Found**.

### 6.1.1.4 WorkerController

Skupina endpointů sloužící pro správu a získávání informací o stavu Worker manažera.

#### GET /api/Worker.

Endpoint slouží pro získání informací o jednotlivých Worker uzlech držených ve frontě Worker manažeru. K této funkci má přístup pouze administrátor. Očekávaný výstup je následující:

```
[
    {
        "Id": int,
        "Address": string,
        "Port": string,
        "State": int,
        "LastHealthCheck": DateTime,
        "FailedConnectionTries" int,
        "IsActive": bool,
        "SolutionId": int,
        "SolutionName": string,
        "CompetitorId", int,
        "Attempt", int
    },
    .....
]
```

#### POST /api/Worker/health/{idWorker}.

Endpoint slouží pro zasílání průběžných ping zpráv, důležitých pro monitorování zdraví jednotlivých Worker uzlů. IdWorker zadaný jako parametr v URL slouží k identifikaci uzlu, který tento endpoint volá. V případě, že není Worker evidován, nebo byl označen jako neaktivní je vrácen HTTP stavový kód **400 Bad Request** a bude jej vracen dokud se uzel nezaregistruje. K této funkci má přístup pouze uzel autentizovaný jako Worker.

#### POST /api/Worker/state/{idWorker}/{state}/{solutionId}.

Endpoint slouží pro aktualizaci stavu (*state*) Worker uzlu (*idWorker*) nad řešením (*solutionId*). V případě, že je řešení odstraněno, nebo označeno ke smazání, nebo není

Worker vedený ve frontě je navrácen HTTP stavový kód **400 Bad Request**. K této funkci má přístup pouze uzel autentizovaný jako Worker.

`PUT /api/Worker/register.`

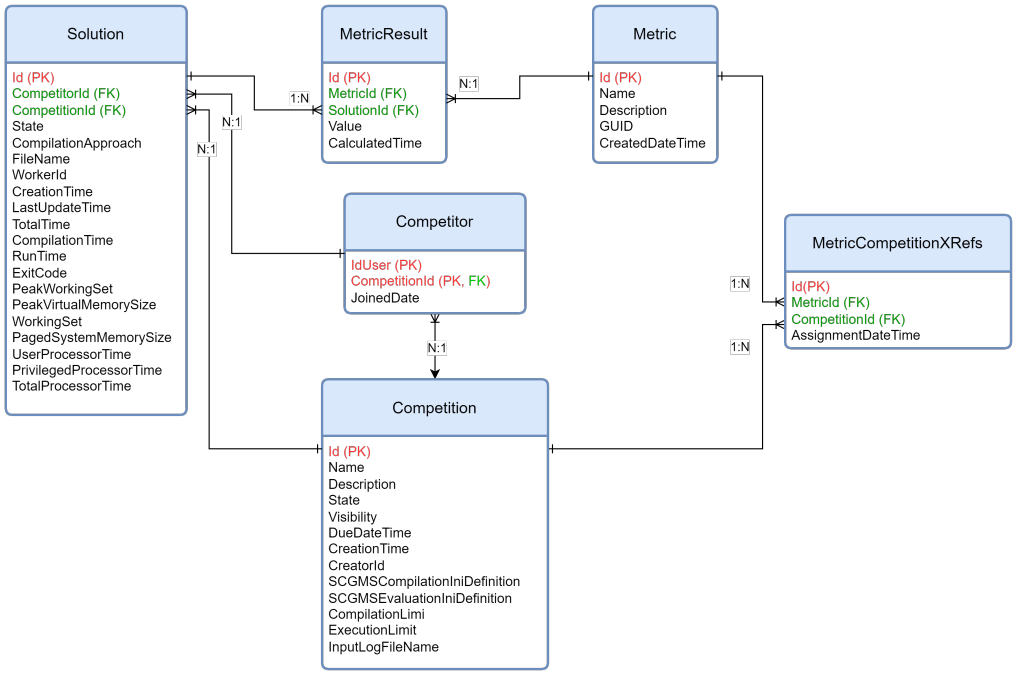
Endpoint slouží pro registraci Worker uzlu, který navrací v případě jeho úspěšné registraci identifikační číslo Worker manažerem. Worker zároveň při registraci předává adresu a port na kterém bude poslouchat pro přiřazování řešení. K této funkci má přístup pouze uzel autentizovaný jako Worker. Očekávaný datový vstup těla požadavku je následující:

```
{
    "IpAddress": int,
    "Port": string
}
```

### 6.1.2 Datová vrstva

Datová vrstva Checker služby je řešená pomocí specializovaných tříd nazvaných jako *Repository*. Ty pracují nad injektovaným databázovým rozhraním, které je implementováno pro databázi SQLite. Dotazy do databáze, mapování tabulek do objektů a ostatní operace nad touto databází jsou řešeny pomocí nástroje Entity Framework [39]. Tento nástroj spravuje také seed a migrace dat. Doménové objekty jsou uloženy v databázi podle schématu znázorněného na obrázku 6.1.

Součástí datové vrstvy je i repozitář pro přístup k souborovému systému, ten je na pozadí vyřešen pomocí připojeného NFS serveru a tak se soubory pracuje tak, jako by se jednalo o soubory na lokálním souborovém systému.




Obrázek 6.1: Schéma doménových objektů uložených v databázi SQLite

## 6.1.3 Kompilační proces

Kompilační proces je označení průchodu uvnitř Worker uzlu (dále v sekci 6.1.3.3), ale z implementačního pohledu je úzce spjatý i s funkcí Worker manažera. Worker manažer s Worker uzlem totiž během celého procesu ale i před ním komunikuje.

### 6.1.3.1 Worker manažer

Kompilační proces spravovaný Worker manažerem je modulárně součástí Checker služby ve třídě *WorkerService* spuštěný jako *BackgroundService* frameworku *ASP.NET*. Worker manažer pracuje na principu producent/konzument. Jak bylo navrženo v předešlé kapitole, Worker manažer slouží pro správu Worker uzlů a je funkce jsou rozděleny do tří vláken. Všechny vlákna přistupují ke stejné frontě Worker uzlů (*WorkersPool*) a proto jsou v kritických sekcích synchronizovány pomocí semaforu *WorkersKey*.

**Vlákno StartJobAssignment.** První vlákno implementované metodou *StartJobAssignment* slouží pro vybírání řešení, které byli vloženy do fronty při jejich vytvoření a nebo při příkazu rekompilace. Po vybrání z této fronty dojde dále ke zvolení prvního volného Worker uzlu, jemuž je následně řešení přes implementovaný konektor zasláno. V případě, že dojde během zasílání dat k chybě, je zvýšeno počítadlo ne-

úspěchů o jedna a řešení je vložené zpátky do fronty. Pokud byl přesažen počet neúspěšných pokusů, je Worker uzel vyřazen a bude se muset při následovné komunikaci znovu zaregistrovat. Celý algoritmus je zobrazen ve zjednodušené formě ve přiloženém zdrojovém kódě 6.1.

Zdrojový kód 6.1: Zjednodušená ukázka kódu přiřazování řešení zvoleným Worker uzlů uvnitř Worker manažera

```
while (running){
    SolutionPool.WaitOne(); //čekej na vložení řešení
    SolutionQueue.TryDequeue(out var solution);
    WorkersPool.WaitOne(); //čekej na aktivní Worker uzel
    WorkersKey.WaitOne();  //klíč pro vstup do kritické sekce
    var worker = Workers.First(worker => worker is
                { State: State.InQueue, IsActive: true });

    //přiřaď řešení Worker uzlu
    var success = _connService.AssignWorkload(worker,
                                                solution);
    if (success) {
        //přiřazení proběhlo úspěšně
        worker.State = State.AssignedCompilation;
        worker.LastSolutionId = solution.Id;
        worker.FailedConnectionTries = 0;
    }
    else {
        //došlo k chybě během přiřazení
        ReturnSolutionToQueue(solution);
        WorkersPool.Release(1);
        worker.FailedConnectionTries++;
        if (worker.FailedConnectionTries >
                                _maxConnectionRetries) {
            //Worker byl vyřazen z aktivní množiny
            worker.IsActive = false;
        }
    }
}
```

**Vlákno HandleUpdates.** Druhé vlákno implementované metodou `HandleUpdates` pracuje nad frontou *WorkerUpdateQueue*. Do této fronty jsou připojenými Worker uzly přidávány požadavky o změně stavu. Při každém požadavku je uvolněn semafor *UpdateKey*, nad kterým je toto vlákno zablokované. K oddělení této logiky došlo z důvodu, aby Worker uzly nemuseli čekat nad kritickou sekcí v momentě kdy požadavek zadávají a byl mohl být proveden později. K aktualizaci dochází například v oblasti stavu vyhodnocování řešení a nebo změny hodnoty času poslední komunikace. Aby nedošlo k vyhladovění ostatních vláken Worker manažera, je ma-

ximální počet zpracovávaných požadavků omezen na konfigurovatelnou hodnotu (viz příloha C). Celý algoritmus je zobrazen ve zdrojovém kódu 6.2.

Zdrojový kód 6.2: Zjednodušená ukázka kódu zpracovávání Worker požadavků

```
while (running){
    UpdateKey.Wait(); //vyčkej na jakýkoliv požadavek o změně
    WorkersKey.WaitOne(); //klíč pro vstup do kritické sekce
    int checkCount = 0;
    while(UpdateQueue.Count > 0 || checkCount > _checkLimit){
        var success = UpdateQueue.Dequeue(out var updateReq);
        ++checkCount;
        var worker = Workers.First(wrk =>
                            wrk.Id == updateReq.WorkerId);
        //zpracování řešení dokončeno
        if (updateReq.State == State.CompilationFinished) {
            worker.State = State.InQueue;
            WorkersPool.Release(1); //Worker vrácen do fronty
        }
        //došlo k chybě během vyhodnocení řešení
        else if (updateReq.State is State.UnknownFailure
                                or State.CompilationFailure
                                or State.RunFailure
                                or State.UnzippingFailure) {
            worker.State = State.InQueue;
            WorkersPool.Release(1); //Worker vrácen do fronty
        }//prostá aktualizace stavu
        else {
            worker.State = updateReq.State;
        }
    }

    checkCount = 0;
    while (HealthCheckQueue.Count > 0
                || checkCount > _checkLimit) {
        HealthCheckQueue.Dequeue(out var workerId);

        var worker = Workers.First(wrk => wrk.Id==workerId);
        worker.LastHealthCheck = DateTime.Now;
    }
    Thread.Sleep(0); //umožni ostatním vláknům pracovat
}
```

**Vlákno StartWorkerHandling.** Poslední vlákno implementované metodou *StartWorkerHandling* prochází v pevně stanoveném intervalu jednotlivé Worker uzly a v případě, že nedošlo během stanoveného času ke komunikaci, je uzel označen jako neaktivní a musí být při další komunikaci znovu zaregistrován.

#### 6.1.3.2 Konektory

Při komunikaci Worker manažeru s Worker uzlem dochází k využívání dvou druhů konektorů. Následující konektory oddělené mezi dva projekty, využívají pro komunikaci kombinaci protokolů HTTP a čistého TCP.

**Worker konektor.** Worker uzel využívá konektoru **ConnectionService**, který je součástí projektu *CompetitionCheckerWorker*. Tento konektor využívá třídu HttpClient k posílání požadavků na REST API Checker služby (viz sekce 6.1.1.4). *HttpClient* je inicializován při startu aplikace, kde je mu kromě cílové URL také přiřazen klientský certifikát ve formátu .pfx, který se používá při každém HTTP požadavku.

**Registrace** Prvním zaslaným HTTP požadavkem je registrace Worker uzlu, při kterém je uzlu přiřazeno unikátní ID vzniklé agregací adresy a portu. Tímto číslem se pak dále prokazuje při každém volání API. V případě selhání registrace, například z důvodu zatím neběžící instance Checker služby, je registrační proces opakován dokud nebude proveden úspěšně. Po registraci začne být tímto konektorem v pravidelných intervalech zasílána ping zpráva.

**Přiřazení řešení** Po úspěšné inicializaci Worker uzlu začne program naslouchat v roli TCP serveru na přiřazené interní adrese Docker overlay sítě a konfigurovaném portu. Jakmile se Worker manažer na tento server připojí, jsou jím zaslaná data načtena z bufferu a deserializována do podoby objektu. Po úspěšné deserializaci je zpátky zaslána zpráva "OK"indikující úspěch přenosu. V případě chyby je celý zmíněný proces restartován.

**Aktualizace stavu** Během kompilačního procesu dochází k průběžnému aktualizování stavu, které je realizována přes konektoru využívající třídu *HttpClient*. Součásti každé aktualizace stavu je kromě samotného stavu, poslána také průběžná statistika kompilačního procesu. Při odesílání tohoto požadavku jsou rozlišovány dva typy stavů. U prvního typu v případě, že dojde k selhání komunikace, není požadavek opakován, aby nedocházelo ke zbytečnému zdržování. Například z důvodu, že mezikrok není natolik důležitý, aby o něm musel manažer vědět. Pokusy o odeslání u druhého typu jsou naopak opakovány do jeho případného úspěchu.

**Worker manažer konektor.** Pro komunikaci Worker manažera je použit konektor *WorkerConnectionService*, pomocí kterého dochází k připojení na TCP server Worker uzlu. Po připojení je tomuto serveru v serializované formě zaslán datový objekt **Workload**, obsahující metadata o řešení a soutěži. Po zaslání dat je očekávána zpráva "OK", aby mohl Worker manažer označit kompilační proces za zahájený. V

případě selhání je proces přiřazování zrušen a musí být znovu opakován Worker manažerem. Data jsou zasílána v následující serializované json struktuře:

```
{
    "Competition":{
        ....
        "SCGMSCompilationIniDefinition": string,
        "CompilationLimit": int,
        "ExecutionLimit": int,
        "InputLogFileName": string,
        ....
    },
    "Solution":{
        "Id": int,
        "CompilationApproach": int,
        "FileName": string,
        ....
        "Parameters": string,
        ....
    }
}
```

### 6.1.3.3 Worker

Worker je zodpovědný za praktickou část průchodu kompilačním procesem. Pomocí Docker swarm režimu je Worker distribuován mezi několik na sobě nezávislých instancí. Jednotlivé instance jsou pomocí vlastního konektoru spojovány s Checker službou (viz sekce 6.1.3.2). Tento konektor také naslouchá na pracovní příkazy Worker manažera. V momentě, kdy dojde k přijmutí pracovního příkazu ve formě metadat soutěže a řešení, je spuštěn kompilační proces. Během kompilačního procesu je procházeno navrženými stavy (viz zdrojový kód 6.3), kde při každé změně stavu je přes konektor stav aktualizován. Logiku jednotlivých stavů obstarává třída *SandboxService*.

Zdrojový kód 6.3: Zjednodušená ukázka kódu průchodu všemi stavy kompilačního procesu

```
while (running){
    //začni poslouchat Worker manažera
    //na přiřazené interní adrese a TCP portu 6969
    var workload = await connector.StartListening();
    if (workload == null) {
        continue;
    }
    bool sandboxPrepared = sandbox.PrepareSandbox(workload);
    if (!sandboxPrepared) {
        await connector.UpdateState(State.UnknownFailure);
        break;
    }
    connector.UpdateState(State.Unzipping);
    bool unzipDone = sandbox.UnzipSolution(workload);
    if (!unzipDone) {
        await connector.UpdateState(State.UnzippingFailure);
        continue;
    }
    connector.UpdateState(State.Compilation);
    bool compilationDone = sandbox.CompileSolution(workload);
    if (!compilationDone) {
        await connector.UpdateState(State.CompileFailure);
        continue;
    }
    connector.UpdateState(State.Running);
    bool executionDone = sandbox.StartScgms(workload);
    if (!executionDone) {
        await connector.UpdateState(State.RunFailure);
        continue;
    }
    connector.UpdateState(State.Cleaning);
    if (!cleaningDone) {
        await connector.UpdateState(State.UnknownFailure);
        break;
    }
    //vše bylo úspěšně dokončeno
    await connector.UpdateState(State.CompilationFinished)
}
```

### SandboxService.

Třída *SandboxService* obsahuje zdrojové kódy sloužící pro vykonání jednotlivých kroků kompilačního procesu navržených v sekci 5.3.3.

**Příprava Sandboxu** V tomto stavu dochází k překopírování vzorové Sandbox složky, která obsahuje předkompilované zdrojové kódy systému SmartCGMS. Následně jsou ze vzdáleného souborového systému nakopírovány vstupní data soutěže a archiv do specifikovaného adresáře uvnitř Sandboxu.

**Rozbalení archivu** Během tohoto stavu dojde k rozbalení archivu přiřazeného řešení, ověření kompilačního přístupu (momentálně dostupný jenom C++17 cmake) a přítomnost potřebných artefaktů. Po rozbalení je možné nad celou překopírovanou složkou spustit systémovou funkci `chown`, sloužící pro udělení oprávněni přístupu a editace nad celou složkou. Oprávnění je nastaveno konfigurovatelnému uživateli, pod kterým budou v následujících stavech spuštěny procesy kompilačních nástrojů a samotné simulace SmartCGMS. V případě, že nějaká z těchto oblastí selže, je kompilační proces ukončen jako *UnzippingFailure*.

**Kompilace zdrojových kódů** Během tohoto stavu jsou postupně spuštěny všechny nakonfigurované příkazy potřebné pro kompletní kompilaci zdrojových kódů. Spuštění funkcí systému je řešeno vytvořením nového procesu pod konfigurovaným uživatelem a omezeným adresářem. Při doběhnutí každého kompilačního příkazu je na vzdáleném souborovém systému aktualizován log sestavení (později dostupného soutěžícímu). V případě, že při nějakém z kompilačních příkazů dojde k selhání nebo vypršení času je kompilační proces ukončen jako *CompilationFailure*.

**Spuštění SCGMS simulace** V tomto stavu dochází k vytvoření SmartCGMS konfiguračního souboru config.ini úpravou řetězce získaného při definování soutěžního scénáře. Jedná se jmenovitě o náhradu následujících tagů:

- <input-log-file> – tag je nahrazen relativní cestou ke vstupním datům soutěže.
- <competitor-guid> – tag je nahrazen vloženým GUID soutěžícím během vytváření řešení.
- <competitor-parameters> – tag je nahrazen seznamem parametrů vložený soutěžícím během vytváření.
- <output-log-file> – tag je nahrazen standardním jménem pro výstupní soubor.

Po vytvoření konfiguračního souboru, je stejně jako při procesu kompilace, vytvořen nový proces spuštěný pod konfigurovaným uživatelem. Proces je spuštěn následovným způsobem:

```
1 scgms-console config.ini
```

Také během chodu procesu dochází k měření několika důležitých systémových proměnných, které měří kvalitu vloženého řešení z technického hlediska. Jedná se o následující:

- **PeakWorkingSet64** – maximální množství použité fyzické paměti v bytech
- **PeakPagedMemorySize64** – maximální množství paměti ve virtuálním souboru pro stránkování paměti v bytech.
- **PeakVirtualMemorySize64** – maximální množství použité virtuální RAM v bytech.
- **WorkingSet64** – poslední záznam aktuálně využité fyzické paměti v bytech.
- **PagedSystemMemorySize64** – množství stránkovatelné paměti použité během běhu operačním systémem.
- **UserProcessorTime** – množství času stráveného na procesoru procesem bežícího v uživatelském režimu.
- **PrivilegedProcessorTime** – množství času stráveného na procesoru procesem běžícího v režimu jádra.
- **TotalProcessorTime** – celkový čas stráveného na procesoru procesem.

Po dokončení měření a celého běhu je výstup procesu uložen do log souboru, který nebude běžnému uživateli z důvodu rizika úniku pacientských dat dostupný. V případě, že v jakémkoliv z těchto stavů dojde k selhání, nebo dojde k vypršení času je kompilační proces ukončen jako *RunFailure*.

**Čištění Sandboxu** Poslední stav je zaměřen na překopírování výstupních dat na vzdálený souborový systém a smazání použité Sandbox složky.

### 6.1.4 Evaluační proces

Evaluační proces spravovaný Evaluátorem je součástí Checker služby, spuštěný jako *BackgroundService* frameworku ASP.NET. Evaluátor pracuje na principu *producent/konzument*, který si vede frontu řešení čekajících na zpracování. Do fronty dojde k vložení při dokončení kompilačního procesu, nebo uživatelem zavoláné funkce pro reevaluaci dříve vyhodnoceného řešení. Fronta je ohraničena synchronizačním primitivem semafor. Po zvolení řešení dojde k aktualizaci jeho stavu a k samotnému procesu vyhodnocení, kde je pro každou metriku soutěže inicializována a spuštěna SmartCGMS simulace. Ta vygeneruje na výstupu log soubor obsahující

signály metriky, jenž jsou dále zpracovány. Zjednodušený algoritmus pro vyhodnocování výsledků řešení během evaluačního procesu je k zahlédnutí v přiloženém zdrojovém kódu 6.4.

Zdrojový kód 6.4: Zjednodušená ukázka kódu procházejícího všemi stavy evaluačního procesu

```
while (running){
    SolutionPool.WaitOne(); //čekej na řešení
    SolutionQueue.TryDequeue(out var solution);
    solutionRepo.UpdateState(State.Evaluation);
    bool success = false;
    //pro každou metriku přiřaznou v soutěži
    foreach (var metric in competition.metrics) {
        //připrav prostředí pro SmartCGMS simulaci
        var prepOk = PrepareEvaluation(solution, metric);
        if (!preparationOk) {
            break;
        }
        //začni SmartCGMS simulaci za účelem evaluace metriky
        var evaluationOk = StartEvaluation(solution, metric);
        if (!evaluationOk) {
            break;
        }
        //zpracuj výstupní data ze simulace a ulož výsledek
        var cleanOk = CleanEvalFolder(solution, metric);
        if (!cleanOk) {
            break;
        }
        success = true;
    }

    if (!success) {
        //process evaluace selhal
        solutionRepo.UpdateState(State.EvaluationFailure);
        continue;
    }
    //vše proběhlo v pořádku
    solutionRepo.UpdateState(State.AllFinished);
}
```

**Příprava Evaluátoru.** Ještě než dojde k samotné evaluaci, musí být adresář se spustitelným systémem SmartCGMS připraven na spuštění vyhodnocovací simulace. V tomto kroku dojde k vyčištění zbylých souborů, pokud tak v minulém průchodu nebylo učiněno, například z důvodu chyby. Po případném vyčištění dojde k nakopírování nového výstupního log souboru získaného během kompilační fáze. V dalším

kroku pak musí být připraven konfigurační soubor SmartCGMS, jehož definice (šablona) je součástí zadaného scénáře soutěže v podobě řetězce. V tomto řetězci jsou nahrazeny následující tagy:

- <input-log-file> – tag je nahrazen relativní cestou ke vstupním datům vygenerovaných během kompilační fáze.
- <reference-signal> – tag je nahrazen GUID vstupního/referenčního signálu soutěže.
- <solution-input> – tag je nahrazen výstupním signálem z entity SmartCGMS soutěžícího přeloženého během kompilační fáze.
- <error-output> – tag je nahrazen relativní cestou k výstupnímu chybovému souboru.
- <metric-guid> – tag je nahrazen GUID aktuálně procházené metriky.
- <output-log-file> – tag je nahrazen relativní cestou k výstupnímu souboru později obsahující výsledky zvolené metriky.

**Evaluace.** Samotná evaluace zvolenou metrikou začíná v momentě, kdy je spuštěn systémový proces simulace SmartCGMS nad předanou sadou následujícím příkazem:

```
1 scgms-console config.ini
```

Oproti kompilačnímu procesu, ale nemusí být z bezpečnostních důvodů omezen na uživatele, což dovoluje spuštění systémového procesu ve stejném prostředí jako je spuštěna Checker služba.

**Sběr výsledků metriky.** Po každém průchodu evaluací metriky dochází ke sběru dat a čištění vygenerovaných souborů. Metriky mají tu vlastnost, že během simulace průběžně zasílají své výsledky následujícím filtrů a proto je potřeba získat z výstupního souboru poslední záznam, který tak bude značit finální výsledek evaluace zvolené metriky. Jelikož je formátování výstupních logů vždy stejné, můžeme předpokládat, že jsou data posledního záznamu ve čtvrtém sloupečku a jsou ve formátu primitivního typu *double*. Zpracovaná hodnota je pak pod evaluovanou metrikou uložena do databáze.

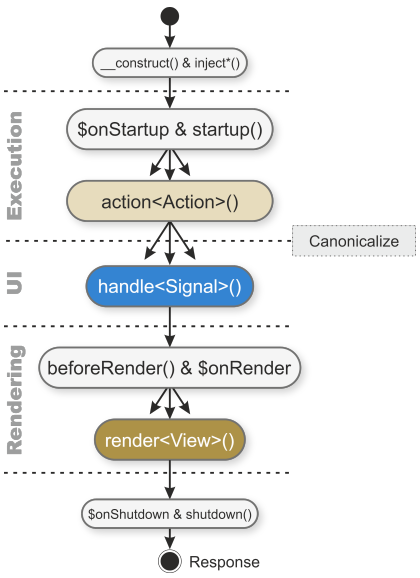
## 6.2 Webový portál diabetes.zcu

Do již existujícího webového portálu diabetes.zcu byl nově integrován soutěžní modul, představující soutěžní rozhraní mezi jeho uživateli a samotnou Checker službou běžící na pozadí. Webový portál je implementován pomocí jazyka PHP a frameworku Nette. Jednotlivé funkce a pohledy systému jsou implementovány podle vzorového modelu *Model–View–Presenter*.

V nově vzniklém soutěžním modulu jsou funkce systému definovány v prezenteru *CompetitionCheckerPresenter*. Tento prezenter je úzce spjatý s Checker službou, od které získává data z definovaných endpointů pomocí HTTP požadavků (viz. sekce 6.1.1). Chyby nastávající během chybného spojení jsou propagovány standardizovanému NETTE handleru zodpovědný za zobrazování chybové stránky. V případě očekávané chybové HTTP zprávy je uživateli zobrazen chybový dialog.

### 6.2.1 CompetitionCheckerPresenter

Prezenter frameworku Nette definuje sadu funkcí známých jako *action*, které obsluhují logiku takových stránek jejichž název odpovídá nazvu funkce. Nad *action* mohou být dále volány podfunkce označované jako *handle* (viz. obrázek 6.2). Pro příklad akce *actionMetricAdd* prezenteru *CompetitionCheckerPresenter* je v URL přeložena jako `competition-checker/metric-add`.




Obrázek 6.2: Životní cyklus prezenteru [40]

Nově implementovaný prezenter *CompetitionCheckerPresenter* obsluhuje v případě soutěžního modulu několik stránek a definuje nad nimi akce, které budou vyjmenovány a popsány v následujících odstavcích.

**actionDefault.** Jedná se o výchozí funkci definovanou na stránce `competition-checker`, která slouží k zobrazení přiřazených a veřejnosti dostupných soutěží získaných z endpointu Competitions/ metodou HTTP GET.

**actionCompetitionAdd.** Akce využívá třídu *CompetitionForm* pro vytvoření, správu a validaci formuláře sloužící pro vytváření nových a editaci již existujících soutěží. Na tento formulář je navázána funkce *onCompetitionEditForm*, která formulářová data zašle v podobě požadavku na endpoint `/api/Competitions` metody PUT nebo POST podle typu formuláře. Tato akce je přístupná pouze pro administrátora.

**actionCompetitionDetail.** Akce slouží pro zobrazení detailů soutěže a výsledkové listiny získaných pomocí endpointu GET Competitions/<*competitionId*>/results/. Akce přijímá z query dva parametry *competitionId* (identifikační číslo soutěže) a *join*. Při nastavení parametru *join* bude zaslán požadavek o připojení k soutěži na endpoint POST Competitions/<competitionId>/join/.

**actionMetric.** Akce obsluhuje zobrazení seznamu metrik získaných z endpointu GET /api/Metrics. Tato akce je přístupná pouze pro administrátora.

**actionMetricAdd.** Akce využívá třídu *MetricForm* pro vytvoření, správu a validaci formuláře pro vytváření a editaci metrik. Na tento formulář je navázána funkce *onMetricEditFormSuccess*, která formulářová data zašle v podobě požadavku na endpoint `/api/Metrics` metody PUT. Tato akce je přístupná pouze pro administrátora.

**actionSolution.** Akce slouží pro zobrazování seznamu do soutěže vložených řešení získaných z endpointu GET /api/Solutions/<competitionId>.

**actionSolutionAdd.**

Akce využívá třídu *SolutionForm* pro vytvoření, správu a validaci formuláře pro vytváření nových řešení. Na tento formulář je navázána funkce *onSolutionAddForm*, která formulářová data zašle v podobě požadavku na endpoint `/api/Solutions` metody PUT.

**actionSolutionDetail.**

Akce slouží pro zobrazení detailů řešení, statistik běhu a výsledků z evaluace získaných pomocí endpointu GET Solutions/<solutionId>/result. Akce přijímá z query parametr *solutionId* (identifikační číslo řešení).

Tato akce zároveň umožnuje spustění několika specializovaných funkcí skrze tlačítko. Jedná se následující:

- **handleDownloadSolutionFile** – funkce provede stažení nahraného řešení skrze endpoint GET Solutions/<solutionId>/download/file. Tento soubor je po jeho stažení uložen v dočasných souborech systému.
- **handleRecompileSolution** – funkce umožnuje skrze endpoint Solutions/recompile/<solutionId> znovu provést proces kompilace a evaluace řešení.
- **handleReevaluateSolution** – funkce umožňuje skrze endpoint Solutions/reevaluate/<solutionId> znovu provést proces evaluace řešení.
- **handleDownloadCompilationLogs** – funkce provede stažení kompilačních logů skrze endpoint GET Solutions/<solutionId>/download/file. Tento soubor je po jeho stažení uložen v dočasných souborech systému.

Šablony jednotlivých stránek a struktury emailů jsou řešeny pomocí šablonovacího systému *latte*. Ty se v případě zmíněného prezenteru podle definice projektu nachází pod adresářem `templates/CompetitionChecker`.

Vstupní parametry jednotlivých formulářů, ovládací prvky, validace vstupní dat a vizuální příklady jsou popsány v uživatelské příručce nacházející se v příloze C.

### 6.2.2 Konektor

Oddělenou komponentou určenou pro řešení síťové komunikace je konektor **CompetitionHttpService**. Tato služba využívá knihovny Symphony HTTPClient verze 5.4 poskytující nízko úrovňovou obalovací třídu pro práci s PHP streamem a nástrojem cURL. Pomocí této třídy jsou na API Checker služby (6.1.1) zasílány požadavky. Typ použité HTTP metody, žádaný endpoint a obsah dat v těle požadavku jsou konektoru podle typu použití předávány prezenterem **CompetitionCheckerPresenter**. Konektor tato data zabalí do přenositelné formy a přiloží i všechny požadované hlavičky. HttpClient je před zavoláním příkazu cURL inicializován následujícími parametry:

- CURLOPT_URL – agregace konfigurovatelné adresy Checker služby se zadaným endpointem
- CURLOPT_RETURNTRANSFER – očekávání výstupu požadavku ve formě textové řetězce. Vždy nastaveno na true
- CURLOPT_CUSTOMREQUEST – stanovení HTTP metody.
- CURLOPT_POSTFIELDS – tělo požadavku v JSON formátu.
- CURLOPT_HTTPHEADER – všechny předávané hlavičky požadavku.

  - UserId – id uživatele, pod kterým se posílá požadavek
  - IsAdmin – označení zdali je uživatel administrátor

- CURLOPT_CONNECTTIMEOUT – maximální možný interval před navázáním úspěšného připojení. Argument je stanovitelný konfigurací.
- CURLOPT_TIMEOUT – maximální možný interval průběhu celého požadavku. Argument je stanovitelný konfigurací.

Pro autentizaci webového portálu pomocí klientského TLS certifikátu je HttpClient inicializován i s následujícími parametry:

- CURLOPT_SSL_VERIFYPEER – přepínač zdali se má ověřovat identita serverového certifikátu. Pro použití self-signed certifikátu je ověření vypnuto.
- CURLOPT_SSLKEY – cesta k privátnímu klíči TLS certifikátu. Nastavitelná pomocí konfigurace.
- CURLOPT_SSLCERT – cesta ke klientskému TLS certifikátu. Nastavitelná pomocí konfigurace.

## 6.3 Docker

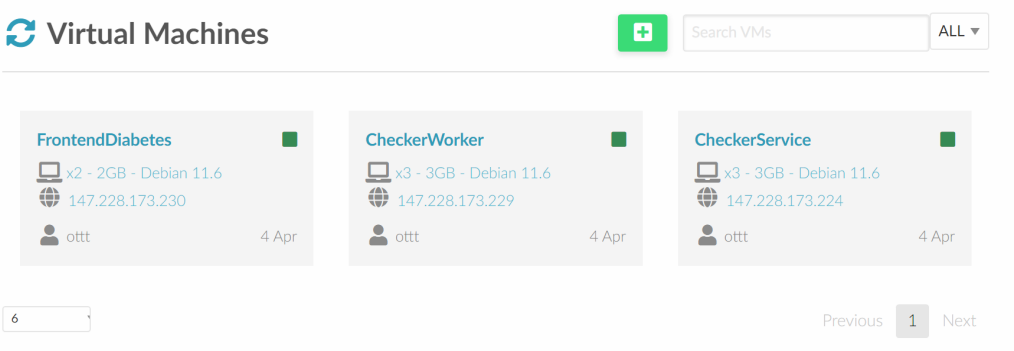
Jak již bylo navrženo v předešlé kapitole jsou komponenty Checker systému kontejnerizovány pomocí nástroje Docker, síťově omezeni na L2 vrstvě ISO/OSI modelu pomocí virtuální overlay sítě a distribuovány v Docker swarm módu. Checker služba poskytující API pro webový portál je spojena s externí síti pomocí virtuálního mostu. Jako preferovanou klientskou aplikací pro nástroj Docker byl zvolen Docker Compose, především kvůli umožnění vytvoření sady konfigurací, pomocí které je možné jedním příkazem sestavit kontejnerovou infrastrukturu. Pro umožnění orchestrace Worker uzlů je Docker compose definice oddělena do souborů `docker-compose-worker.yml` a `docker-compose-service.yml` (viz příloha B). Nástroj totiž nepodporuje při procesu orchetrace kombinaci overlay sítě a virtuálního mostu, který je ale potřebný pro chod Checker služby. Pro distribuci dat je během sestavování infrastruktury připojen ke kontejnerům Docker volume, který odkazuje na složku spravovanou pomocí NFS. Cesta k této složce je v jednotlivých kontejnerech stanovená konfigurací (viz příručka C). Stejným způsobem dochází k připojení složek obsahující klientské certifikáty, které jsou používány pro zabezpečení komunikace.

Jelikož jsou binární soubory SmartCGMS potřebné pro kompilační i evaluační proces, tak Docker image Checker služby i Worker instance vyžaduje sestavený image souboru `ScgmsNetBase.DockerFile` pojmenovaný jako *scgms*. Tento obraz

slouží primárně ke stažení zdrojových kódů systému SmartCGMS, všech potřebných závislostí a následně je přeloží. Samotné obrazy Checker služby a Worker uzlu jsou pak sestaveny využitím inkrementálního buildu, během jehož částí dochází k překladu zdrojových kódů implementovaného řešení.

# Ověření funkčnosti a výkonu systému — 7

V této bude popsáno jakým způsobem byl soutěžní systém nasazen, otestován a ověřen z hlediska výkonu. Pro účely nasazení aplikace byla využita cloudová služba poskytovaná oddělením CIV na Západočeské Univerzitě. Cloudová služba je implementována platformou OpenNebula, pomocí které je možné sestavit až čtyři virtuální stroje. Každý virtuální stroj má přiřazenou hostname a veřejnou IP adresu [41]. Celý soutěžní systém byl s využitím zmíněné časově omezené služby nasazen na tři oddělené virtuální stroje (viz obrázek 7.1).




Obrázek 7.1: Přehled použitých virtuálních strojů [41]

Jedná se o následující:

- FrontendDiabetes – slouží pro hostování webového portálu diabetes.zcu.cz pomocí webového HTTP serveru Apache. Virtuální stroj běží na operačním systému Debian 11.6, má přiřazené 2 fyzické CPU, 2GB RAM a 20GB přiděleného diskového místa.

- CheckerService – hostuje manažerský Docker daemon ve Swarm módu, na kterém běží Checker služba ve virtuální overlay síťi společně s jedním Worker uzlem. Virtuální síť je sdílená s virtuálním strojem CheckerWorker. Virtuální

stroj běží na operačním systému Debian 11.6, má přiřazené 3 fyzické CPU, 3GB RAM a 20GB přiděleného diskového místa.

- CheckerWorker – hostuje pracovníka Docker daemon ve Smarm módu, na kterém běží jeden Worker uzel ve virtuální overlay síťi sdílené s virtuálním strojem CheckerService. Virtuální stroj běží na operačním systému Debian 11.6, má přiřazené 3 fyzické CPU, 3GB RAM a 10GB přiděleného diskového místa.

Po nasazení nového Checker systému na zmíněné virtuální servery, byla pomocí webového rozhraní (viz. příloha C) vytvořena soutěž. Scénářem této soutěže byla predikce vývoje krevního cukru v krvi na datech získaných z modelového signálu entity SmartCGMS představující virtuálního pacienta. Konfigurace simulace SmartCMGS pro kompilační a evaluační procesy byla při definici soutěže nastavena podle přílohy A. Soutěži byly dále přiřazeny dvě SmartCGMS metriky. Ty počítají průměrnou absolutní odchylku a maximální odchylku mezi vstupními pacientskými daty a výstupem z predikce zadaného řešení.

Po definování všech důležitých parametrů soutěže byly pod ní vloženy dvě stejná řešení. Tyto řešení se snaží predikovat glukózu v krvi pomocí váženého průměru posledních dvou hodnot. Řešení přijímá skrze konfiguraci SmartCGMS parametr *Offset*, což je číslo přičítané při každé predikaci. Jedno z řešení bylo úmyslně tímto nastavitelným parametrem znehodnoceno nastavením na hodnotu 100, aby bylo ověřeno, že tímto krokem bude ovlivněn i výsledek evaluačního procesu. V tabulce výsledků vyhodnocení 7.1 je, jak bylo očekáváno, znázorněno, že první neznehodnocené řešení dosahuje lepších výsledků, jelikož se jedná o výpočet odchylky a je tak cílem být ohodnocen co nejmenší hodnotou.

Tabulka 7.1: Výsledek vyhodnocení v jednotkách mmol/L standardního a znehodnoceného řešení predikující úroveň glukózy v krvi pomocí váženého průměru

| **Metrika** | **Řešení 1** | **Řešení 2** |
|---|---|---|
| Průměrná absolutní odchylka | 0,481405 | 100.067 |
| Maximální odchylka | 12.2051 | 109.191 |

Další pokus byl proveden vložením řešení obsahující syntaktickou chybu ve vkládaných zdrojových kódech. Průchod Checker systémem byl očekávaně zastaven na stavu *CompilationFailure* označující chybu během kompilace. Logy z kompilace bylo možné pomocí webového rozhraní stáhnout. Obsah těchto logů je zobrazen ve výpisu 7.1.

Výpis 7.1: Ukázka chybového výpisu při vložení řešení se syntaktickou chybou

```
[ 18%] Built target scgms
[ 22%] Built target approx
[ 27%] Built target controllers
[ 36%] Built target data
[ 36%] Building CXX object
   core/dp_ott_competition_filters/CMakeFiles/dp_ott_competiti
   on_filters.dir/src/competition_filters.cpp.o
/home/app/sandbox/core/dp_ott_competition_filters/src/competit
   ion_filters.cpp: In member function 'virtual HRESULT
   CCompetition_Filter_2::Do_Execute(scgms::UDevice_Event)':
/home/app/sandbox/core/dp_ott_competition_filters/src/
   competition_filters.cpp:127:35: error: 'chybaJakoHrom' is
   not a member of 'std'
  127 |  if (!std::chybaJakoHrom(mLast_IG)) {
      |           ^~~~~~~~~~~~~
make[2]: *** [core/dp_ott_competition_filters/CMakeFiles/
   dp_ott_competition_filters.dir/build.make:76:
   core/dp_ott_competition_filters/CMakeFiles/
   dp_ott_competition_filters.dir/src/competition_filters.cpp.
   o] Error 1
make[1]: *** [CMakeFiles/Makefile2:12046: core/
   dp_ott_competition_filters/CMakeFiles/dp_ott_competition_f
   ilters.dir/all] Error 2
make: *** [Makefile:136: all] Error 2
```

Pro otestování výkonu bylo pod definovanou soutěž do systému postupně vloženo pět řešení. Časová náročnost celého průchodu a jeho klíčových oblastí byly naměřeny a zapsány do tabulky 7.2.

Tabulka 7.2: Časová náročnost celého průchodu řešení a jeho jednotlivých oblastí Checker systému

| **Měřená oblast** | **Průměr** | **Min** | **Max** |
|---|---|---|---|
| Celkový čas průchodu | 7s 565ms | 7s 433ms | 7s 731ms |
| Příprava Sandboxu | 0s 833ms | 0s 784ms | 0s 950ms |
| Rozbalování řešení | 0s 098ms | 0s 077ms | 0s 142ms |
| Kompilace | 5s 011ms | 4s 946ms | 5s 090ms |
| Běh SmartCGMS | 0s 175ms | 0s 162ms | 0s 191ms |
| Čištění Sandboxu | 0s 053ms | 0s 010ms | 0s 073ms |
| Kompilační proces | 6s 210ms | 6s 088ms | 6s 336ms |
| Evaluační proces | 1s 325ms | 1s 292ms | 1s 364ms |
| Metrika č.1 simulace | 0s 611ms | 0s 602ms | 0s 626ms |
| Metrika č.1 sběr dat | 0s 019ms | 0s 012ms | 0s 035ms |
| Metrika č.2 simulace | 0s 633ms | 0s 602ms | 0s 662ms |
| Metrika č.2 sběr dat | 0s 027ms | 0s 024ms | 0s 030ms |

K měření jednotlivých oblastí kompilačního procesu docházelo uvnitř Worker manažeru. To znamená, že časové odchylky v jednotlivých oblastech mohou být způsobené synchronizačními akcemi, síťovým přenosem, nebo vytížením databáze.

# Zhodnocení 8

V rámci této diplomové práce vznikl distribuovaný soutěžní systém, který umožňuje administrátorům definovat komplexní scénáře soutěží, využívat nad nimi metriky systému SmartCGMS a mít přehled o množině pracovních uzlů. Běžnému uživateli je pak umožněno této soutěže se zúčastnit, vložit vlastní řešení ve formě zdrojových kódů entity systému SmartCGMS a zobrazit výsledky vyhodnocovacího procesu porovnatelných s ostatními účastníky soutěže v přehledné výsledkové listině. Z pohledu autora této diplomové práce tak poskytuje uživateli přehledné, přívětivé a intuitivní rozhraní umožnující zúčastnění se soutěže. Z administrátorského pohledu však může být zprvu náročné definovat všechny parametry komplexní soutěže, byť to bylo i zároveň cílem této práce.

Implementace celého a spolehlivého soutěžního systému je však náročný úkol obsahující velké množství výzev. Některé výzvy nebyly kompletně vyřešeny a pokryty, čímž dochází k určitým limitacím. Jednou z možných limitací implementované Checker služby je neschopnost evaluační proces škálovat a rozdělit mezi několik distribuovaných strojů. To sice v aktuální formě nepředstavuje hrozbu, ale v momentě kdy bude soutěž dosahovat mnohem větších rozměrů, může dojít ke zpomalování odpovědí na požadavky webového portálu. Další limitací z výkonnostního hlediska je aktuálně použitá souborová databáze SQLite, která při dosažení vyšších rozměrů využívání systému může představovat úzké hrdlo. Systém je však díky třívrstvému architektonickému návrhu možné jednoduše migrovat na jiné druhy databází.

V současné době je v plánu implementovaný soutěžní systém příští rok využít v rámci reálné soutěže z oblasti onemocnění diabetes mellitus. Během přípravy na tuto soutěž by tak mohly být odhaleny i další podstatné nedostatky uživatelského rozhraní.

## 8.1 Budoucí rozšíření

**Unikátní Worker klientské certifikáty.** Jak již bylo zmíněno, Worker uzel využívá při komunikaci s Checker službou klientského certifikátu, který je ale sdílen

mezi všemi jeho instancemi. Nově navrhované řešení by tak využívalo unikátních klientských certifikátů pro odlišení jednotlivých instancí.

**Oddělení Evaluátoru.** Jedním z možných rozšíření implementovaného soutěžního systému je oddělení evaluačního procesu od Checker služby do vlastního kontejneru. Toto rozšíření by tak umožnilo horizontální škálování i z pohledu evaluačního procesu. Jelikož je ale samotná evaluace natolik rychlá, není v aktuální podobě tento problém potřeba řešit.

**Uživatelské rozhraní uvnitř mobilní aplikace.** Jelikož je Checker služba oddělená od webového portálu pomocí REST API, může být toto rozhraní využito i jinou klientskou aplikací. Mezi ně můžeme zařadit i mobilní aplikaci, která oproti běžnému webovému portálu má několik předností. Informace o aktualizaci stavu soutěže, řešení, nebo výsledkové listiny by tak mohla být předávána uživateli například pomocí notifikací.

**Správa celého systému pomocí Kubernetes.** Checker služba v aktuálním stavu využívá distribuci pomocí Docker swarm módu, která je pro případ distribuce pracovních uzlů adekvátní. Celý systém počínaje webovým portálem, databází, Checker službou i jednotlivými instancemi Worker uzlů by ale mohl být nasazen pomocí nástroje Kubernetes, který se pro nasazení komplexních systém v technické praxi používá.

# Závěr 9

Tato diplomová práce čtenáře seznamuje s problematikou soutěžního programování, které má přesah i do medicínské sféry. Tento přesah byl prozkoumán z oblasti soutěže zaměřené na onemocnění diabetes mellitus. Návazně na toto onemocnění byl čtenář seznámen se systémem SmartCGMS vyvíjeného na Katedře Informatiky a Výpočetní techniky, který se mimo jiné danou problematikou zabývá. Na základě těchto znalostí byla dále prozkoumána oblast vzdáleného a izolovaného sestavení zdrojových kódů, jež může být kritickou části každého soutěžního systému.

Nad zvolenými technologiemi a metodami byl navržen a implementován soutěžní systém, který byl rozdělen do tří oddělených komponent. Webové rozhraní v podobě již existujícího webového portálu diabetes.zcu bylo rozšířeno o soutěžní modul, který slouží jako rozhraní pro komunikaci s oddělenou soutěžní službou. Tato služba, kromě definice rozhraní sloužícího pro správu komplexních soutěží, dále řídí distribuovanou pracovní množinu uzlů zaměřených na sestavení a spuštění zdrojových kódů entit systému SmartCGMS. Služba nad výsledky z kompilační fáze provádí dále vyhodnocení pomocí metrik systému SmartCGMS. Statistiky a výsledky celého chodu překladu a vyhodnocení jsou prezentovány uživateli pomocí webového rozhraní.

Všechny části výsledného soutěžního systému byly nasazeny mezi tři virtuální stroje, pomocí kterých byla ověřena jejich funkčnost v distribuovaném prostředí. Jednotlivé oblasti průchodu kompilačním a vyhodnocovacím procesem byly naměřeny a tím byly zvýrazněny kritické sekce systému. K průchodu celého systému vloženého řešení ve formě zdrojových kódů entity SmartCGMS tak dojde v řádu jednotek vteřin.

Tato diplomová práce tak splnila zadání v plném rozsahu, ale výsledný soutěžní systém může být nad rámec dále rozšířen v oblasti bezpečnosti a uživatelských funkcí. Přístup k webovému rozhraní může být vylepšen specializovanou mobilní aplikaci. Celý systém by také mohl být spravován a ovládán pomocí nástroje Kubernetes.

# Testovací konfigurace soutěžního scénáře — A

## A.1 Konfigurace systému SmartCGMS pro kompilační proces

Konfigurace definuje tři filtry, první a poslední slouží pro vstup a výstup .csv log souborů. Druhý filtr a jeho parametry jsou doplněny na základě zadaných informací při vkládání řešení.

```
; CSV File Log Replay
[Filter_001_{172EA814-9DF1-657C-1289-C71893F1D085}]
Log_File = <input-log-file>
Emit_Shutdown = true
Filename_as_segment_id = false
Reset_segment_id = false
Emit_All_Events_Before_Shutdown = true

; Competition filter 2
[Filter_002_<competitor-guid>]
<competitor-parameters>

; Log
[Filter_003_{C0E942B9-3928-4B81-9B43-A347668200BA}]
Log_File = <output-log-file>
Log_Segments_Individually = false
Reduce_Log = false
Second_Threshold = 0
```

## A.2 Konfigurace systému SmartCGMS pro evaluační proces

Konfigurace je rozdělena na tři sekce, první a poslední slouží pro vstup a výstup .csv log souborů. Druhá sekce pak nastavuje SmartCGMS metriku, která vstupní data ohodnotí.

```
; CSV File Log Replay
[Filter_001_{172EA814-9DF1-657C-1289-C71893F1D085}]
Log_File = <input-log-file>
Emit_Shutdown = true
Filename_as_segment_id = false
Reset_segment_id = false
Emit_All_Events_Before_Shutdown = true

; Signal error
[Filter_002_{690FBC95-84CA-4627-B47C-9955EA817A4F}]
Description = error_evaluation
Reference_Signal = <reference-signal>
Error_Signal = <solution-input>
Metric = <metric-guid>
Levels_Required = 30
Relative_Error = false
Squared_Diff = false
Prefer_More_Levels = false
Metric_Threshold = 0
Emit_Metric_As_Signal = true
Emit_Last_Value_Only = false
Output_CSV_file = <error-output>

; Log
[Filter_003_{C0E942B9-3928-4B81-9B43-A347668200BA}]
Log_File = <output-log-file>
Log_Segments_Individually = false
Reduce_Log = false
Second_Threshold = 0
```

# Sestavení a instalace systému — B

## B.1 Webové rozhraní diabetes.zcu

Pro chod a sestavení webového portálu `diabetes.zcu` je vyžadován jazyk **PHP nejméně verze 7.1** a nástroj **composer verze nejméně 2.4**, jenž slouží pro správu závislostí projektu. V kořenovém adresáři pak stačí spustit příkaz pro stáhnutí všech potřebných knihoven:

```
composer install
```

Pro samotný chod je také vyžadována **MySQL databáze**. Ta může být zprovozněna pomocí příkazu `docker compose up` díky `.yml` souboru nacházejícího se v kořenovém adresáři a nebo může být manuálně nainstalována přímo na zařízení, na kterém dochází ke spouštění webového portálu.

Po úspěšném získání závislostí a zvoleném způsobu nasazení databáze, je již možné server spustit a to například v kořenovém adresáři následovně:

```
php.exe -S localhost:3000 -t www
```

Nebo se nabízí využití jakéhokoliv **HTTP serveru** jako je například **Apache** a celý projekt nasadit do adresáře podle manuálu zvoleného serveru.

## B.2 Checker služba

Pro zprovoznění celé Checker služby potřebujeme několik nástrojů. Patří mezi ně **Docker** verze optimálně 26 obsahující rozšíření Docker swarm módu a **docker-compose verze 1.25**. Při použití distribuovaného řešení pak také na master uzlu nfs-kernel-server pro zprovoznění NFS serveru a **nfs-common** tedy NFS klienta pro Worker uzel.

Pokud tyto prerekvizity splňujeme, můžeme na našem hlavním uzlu inicializovat Docker swarm mód, jenž je potřebný pro distribuované řešení. V následujících konzolových výpisech budou zmíněny dva virtuální stroje `sulis224` označující hlavní stroj a `sulis229` jako připojený pracovní uzel.

Výpis B.1: Ukázkový výpis inicializace Docker swarm módu.

```
root@sulis224:\$ docker swarm init
Swarm initialized: current node (mv4o7gvqqpk6og4ykuubpz82d)
    is now a manager.

To add a worker to this swarm, run the following command:

    docker swarm join --token
    SWMTKN-1-16csex01501hz90y0bvi896n9cxez5hcfsg1xtmv6qgwsgbsz7
    -69nv17t4dpu0iwwqsoa9k4ba0 147.228.173.224:2377

To add a manager to this swarm, run 'docker swarm join-token
    manager' and follow the instructions.
root@sulis224:\$
```

Jak výpis z konzole B.1 zobrazuje, tak je v aktuální chvíli na stroji s funkčním a spuštěným `Docker daemonem` možné se k nově vytvořenému `Docker swarm` manažer uzlu za pomocí vygenerovaného tokenu připojit.

Aplikace pro svůj distribuovaný chod také vyžaduje sdílený NFS adresář. Pro zprovoznění tohoto adresáře je potřeba editovat konfigurační soubor `/etc/exports` a to tak, že do něj přidáme řádku s adresou pracovního uzlu. Pro příklad se jedná o následující řádku:

```
/mnt/nfs {147.228.173.229}(rw,sync,crossmnt)
```

Po přidání je potřeba provést aktualizaci nastavení NFS serveru (viz výpis B.2).

Výpis B.2: Příkazy potřebné pro změnu nastavení NFS serveru.

```
root@sulis224:\$ exportfs -a
exporting 147.228.173.229:/mnt/nfs
root@sulis224:\$ systemctl restart nfs-kernel-server
```

Po úspěšném nastavení NFS serveru, můžeme začít s inicializací samotné Checker služby. Prvním krokem je vytvoření overlay sítě, která je distrubuovaná přes všechny připojené Docker daemony.

Výpis B.3: Příklad vytvoření docker overlay sítě.

```
root@sulis224:\$ docker network create dp-ott-inner-network\
    --opt encrypted -d overlay --attachable\
    --subnet=10.7.0.0/16

txkx80gg0r9qaq2vs70oi6i36
root@sulis224:\$
```

Po vytvoření docker overlay sítě je možné spustit nástroj Docker compose pro sestavení samotné Checker služby (viz výpis B.4).

Výpis B.4: Příklad spuštění Checker služby pomocí nástroje Docker compose.

```
root@sulis224:\$ docker-compose -f docker-compose-service.yml
    up -d
WARNING: The Docker Engine you're using is running in swarm
    mode.
Starting service_checker_service_1 ... done
root@sulis224:\$
```

Na dalším fyzickém či virtuálním uzlu s funkčním Docker daemonem provedeme připojení k master uzlu pomocí tokenu, jenž byl vygenerován dříve při inicializaci (viz výpis B.5).

Výpis B.5: Příklad připojení fyzického zařízení do existující Docker swarm sítě.

```
root@sulis229:\$ docker swarm join --token <token>
    147.228.173.224:2377
This node joined a swarm as a worker.
root@sulis229:\$
```

Také nesmíme zapomenout na připojení samotného NFS adresáře dostupného z adresy našeho master uzlu (viz výpis B.6).

Výpis B.6: Příklad připojení sdíleného NFS adresáře.

```
root@sulis229:\$ mount -F nfs 147.228.173.224:/mnt/nfs /mnt/nfs
root@sulis229:\$
```

Po ověření, že došlo k připojení do `Docker swarm` (viz výpis B.7), následuje už jen distribuované vytvoření jednotlivých Worker uzlů. Toho dosáhneme pomocí inicializace `docker service` na libovolném master (viz výpis B.8).

Výpis B.7: Zobrazení aktuálního stavu docker swarm uzlů.

```
root@sulis224:\$ docker node ls
ID           HOSTNAME  STATUS  AVAILABILITY  MANAGER STATUS
xx0ix7h60l*  sulis224  Ready   Active        Leader
wz1rr8o7v0   sulis229  Ready   Active
root@sulis224:\$
```

Výpis B.8: Příklad nasazení dvou Checker Worker uzlů.

```
root@sulis224:\$ docker stack deploy -c \
    docker-compose-worker.yml dp-ott --detach=false
Creating service dp-ott_competition_checker_worker
overall progress: 2 out of 2 tasks
1/2: running
2/2: running
verify: Service 86wrxewkdlyfqb7ihlwnue0pk converged
root@sulis224:\$
```

Pokud počet použitých uzlů nestačí může být pak kdykoliv změněn následovným příkazem:

```
docker service update --replicas <pocet-workeru> \
    dp-ott_competition_checker_worker
```

### B.2.1 Visual Studio řešení

Součástí finalního balíčku je i .sln soubor, pomocí kterého je možné aplikaci spustit na operačním systému Windows v programu MSVC. Program vyžaduje předinstalovaný balíček **ASP.NET Core Runtime 8.0 SDK**. Poté si již při prvním sestavování všechny potřebné závislosti podle potřeby stáhne. Tento přístup spuštění je užitečný především z pohledu ladění a vývoje nové funkcionality. Je však potřeba nastavit absolutní cesty k důležitým souborům v konfiguracích *appsettings.Development.json* zmíněných v sekcích C.4 a C.5.

Jelikož aplikace využívá v aktuální podobě self-signed SSL certifikáty, musí být přidány do seznamu důvěryhodných certifikátů a v případě potřeby k nim dodány cesty samotné asp.net aplikaci [42]. Přidání certifikátu do seznamu z lokálního adresáře může být provedeno následujícím příkazem:

```
dotnet dev-certs https --trust
```

Také je před prvním spuštěním vyžadována inicializace a migrace dat SQLite databáze pomocí příkazu zadaného do příkazové řádky:

```
dotnet ef --startup-project CompetitionChecker/Service \
    migrations add Initial
```

# Uživatelská příručka C

Aplikace je rozdělená na webové rozhraní a službu na pozadí, které správce musí ovládat, aby celý systém fungoval tak jak má. V následujících sekcích bude systém uveden jak z pohledu běžného uživatele, tak i z pohledu administrátora a budou vysvětleny možnosti konfigurace systému.

## C.1 Webové rozhraní diabetes.zcu

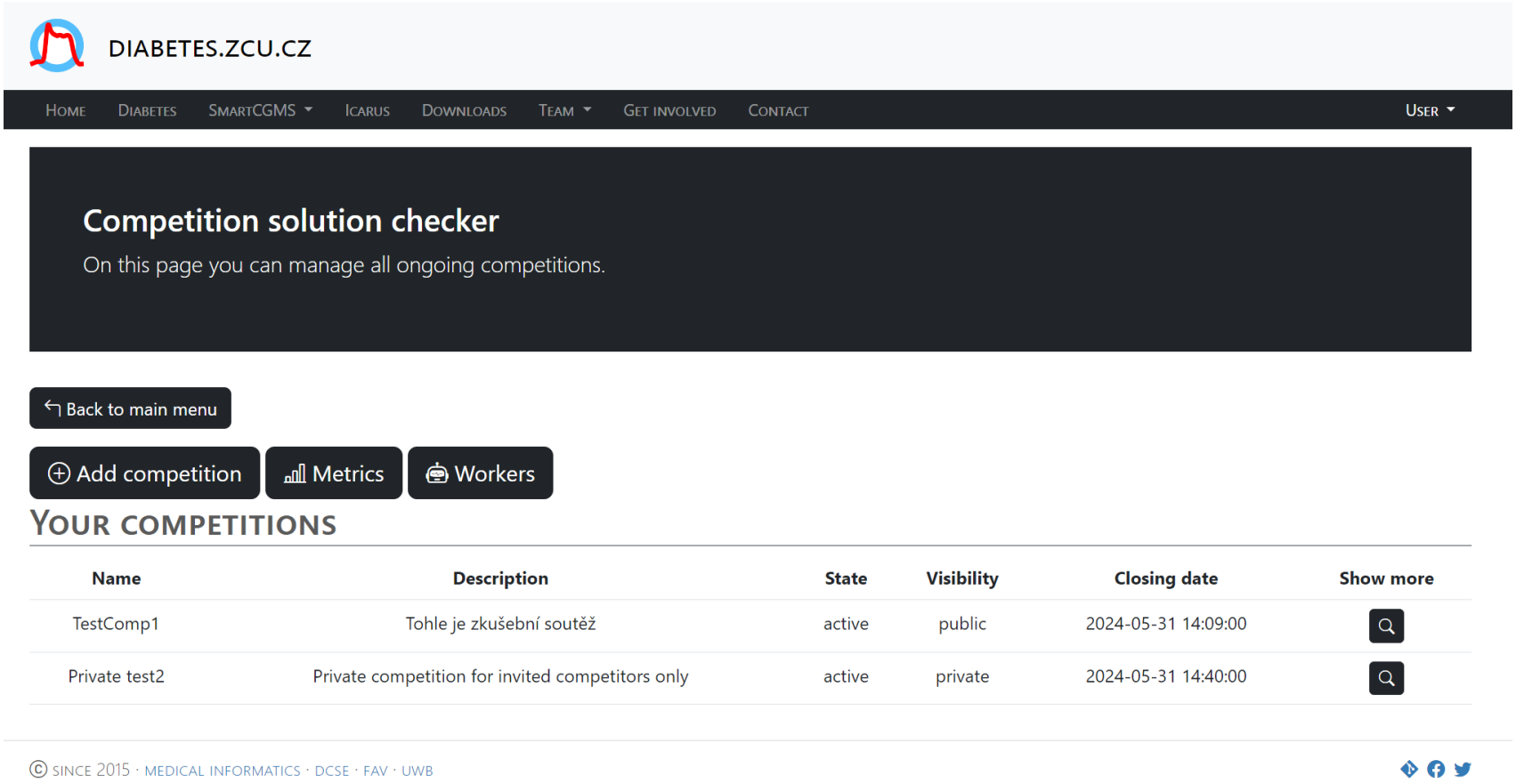
Již existující portál `diabetes.zcu` je nově rozšířen o modul vyhodnocování soutěží. Na úvodní stránku je možné se dostat pomocí tlačítka '**Checker**' nacházející se na horní liště stránky a nebo z uživatelské či administrátorské desky. Umožňuje jak správu komplexních soutěží, používaných metrik a řešení, tak i kontrolování stavu připojení jednotlivých Worker uzlů.

### C.1.1 Soutěže

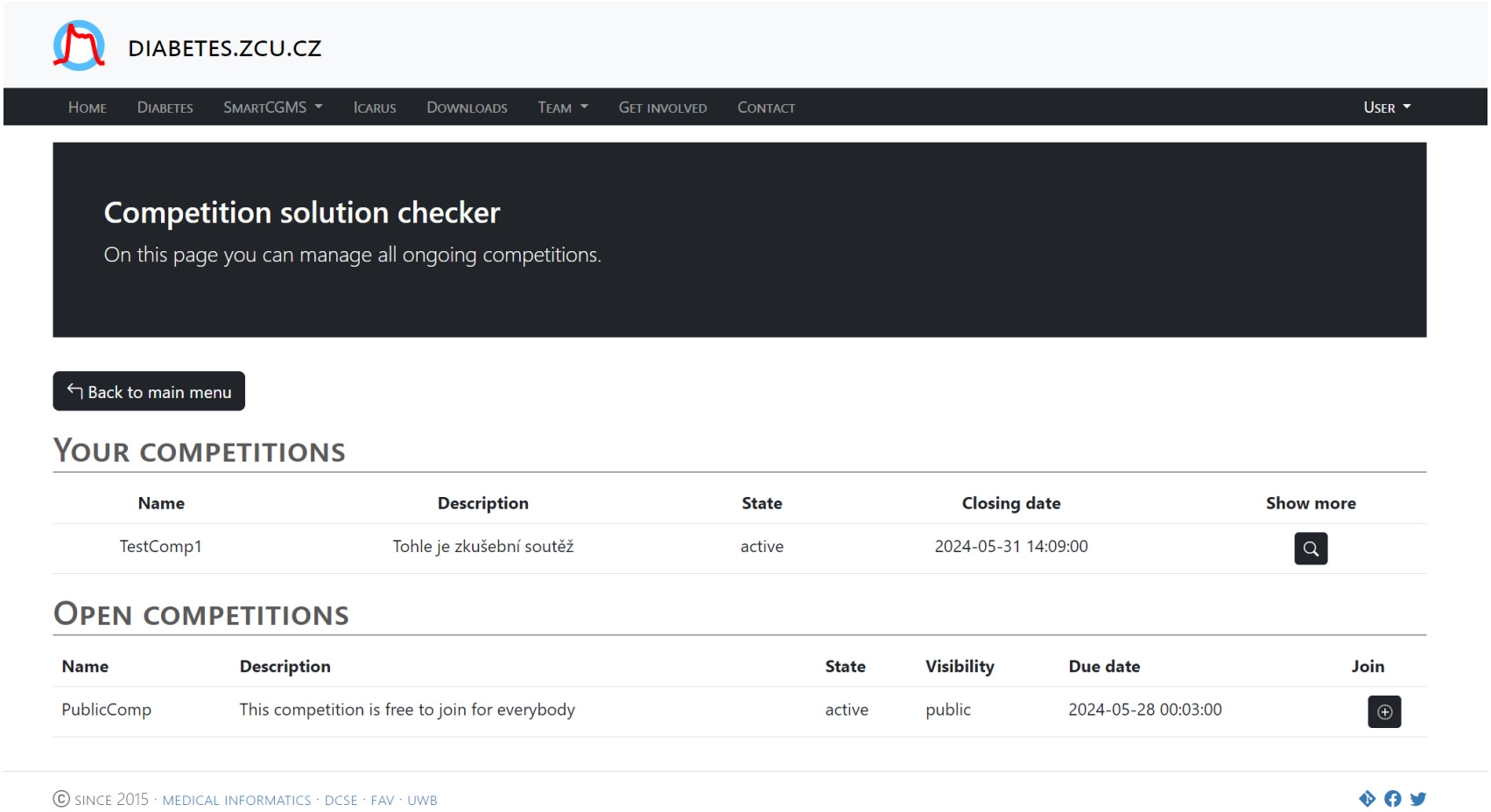
Úvodní stránka Checker systému slouží jako rozcestník pro rozhraní metrik (dále sekce C.1.3), Worker uzlů (sekce C.1.2), ale také samozřejmě pro zobrazení seznamu soutěží (viz obrázek C.1). V případě běžného uživatele jsou zobrazeny pouze soutěže, ke kterým je uživatel přiražen ('Your competitions') a nebo jsou volně přístupné pod nadpisem 'Open competitions' (viz obrázek C.2). K otevřeným soutěžím se může uživatel po stisku tlačítka se znakem plus přidat. Administrátor zde může také pomocí tlačítka 'Add competition' přejít na vytváření nových soutěží.

Ve formuláři zobrazeném na obrázku C.5 je několik editovatelných polí, kde na samém začátku může uživatel vložit název a popisek soutěže. Následují dva rozevírací seznamy slouží pro určení stavu a viditelnosti (význam stavů popsán v sekci č.6.2). Nesmí chybět ani pole pro stanovení časových limitů překladu a následně samotné exekuce sestaveného řešení. Důležité pro soutěž jsou i vstupní data. Ty mohou být vložené skrze rozhraní ve formátu .csv pro vložení souboru v sekci '**Input log file**'. Záznamy v tomto souboru se mezi sebou identifikují pomocí GUID a i zde je potřeba zadat referenční GUID ohraničené složeními závorkami. Poté je

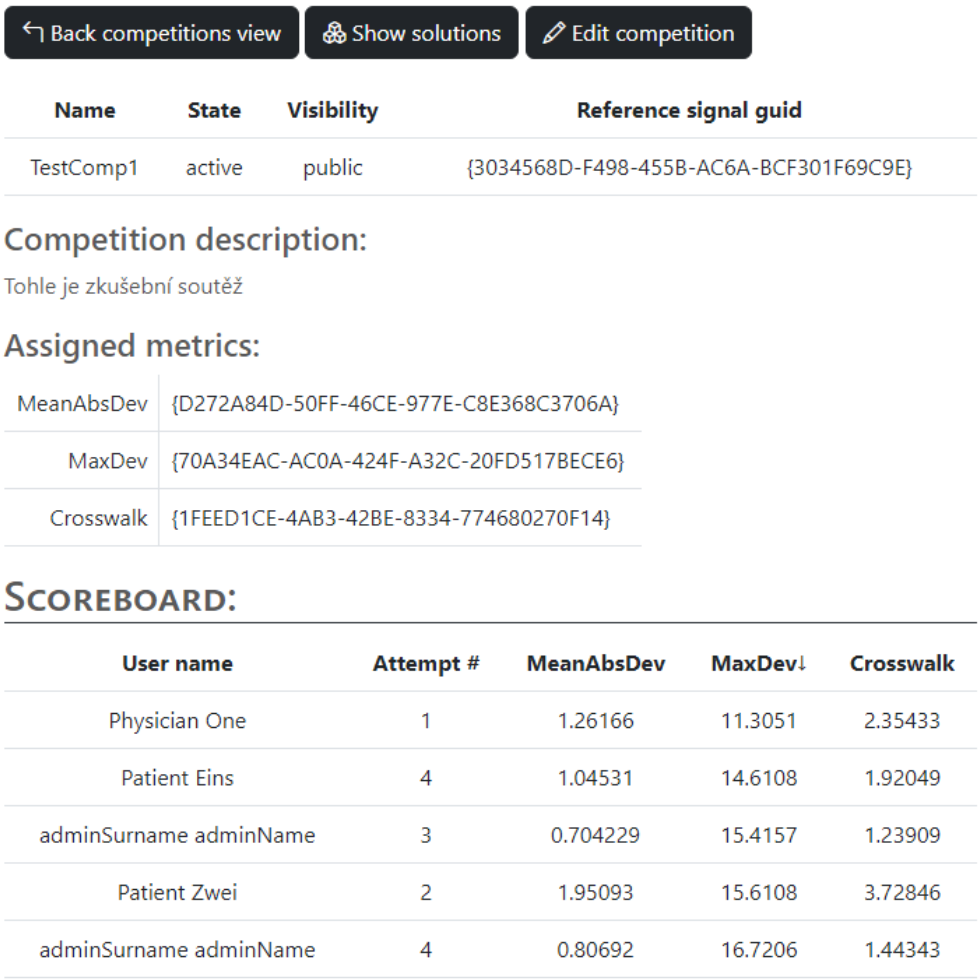
možné přiřadit soutěži metriky (dále sekce C.1.3) a pozvané účastníky. V případě, že chcete ještě omezit přidávání řešení po určiném datum a času, tak je možné využít datepicker '**Due date**'. Poslední dvě textové pole '**SCGMS compilation ini**' a '**SCGMS evaluation ini**' slouží pro zadání .ini souboru potřebného systémem SmartCGMS pro kompilaci a následnou evaluaci (vysvětleného v sekci č.6.2).



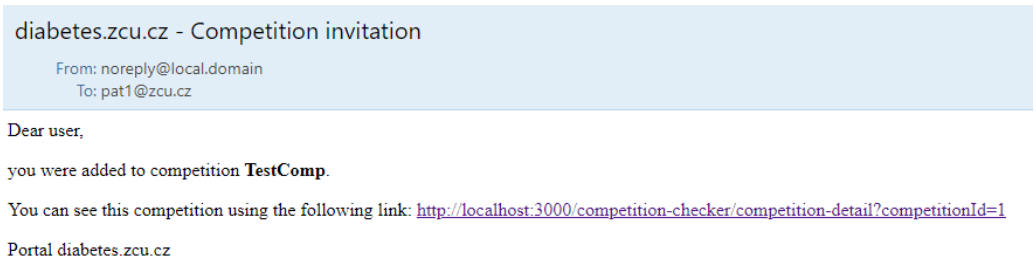

Obrázek C.1: Pohled na všechny soutěže pro uživatele s administrátorskými právy



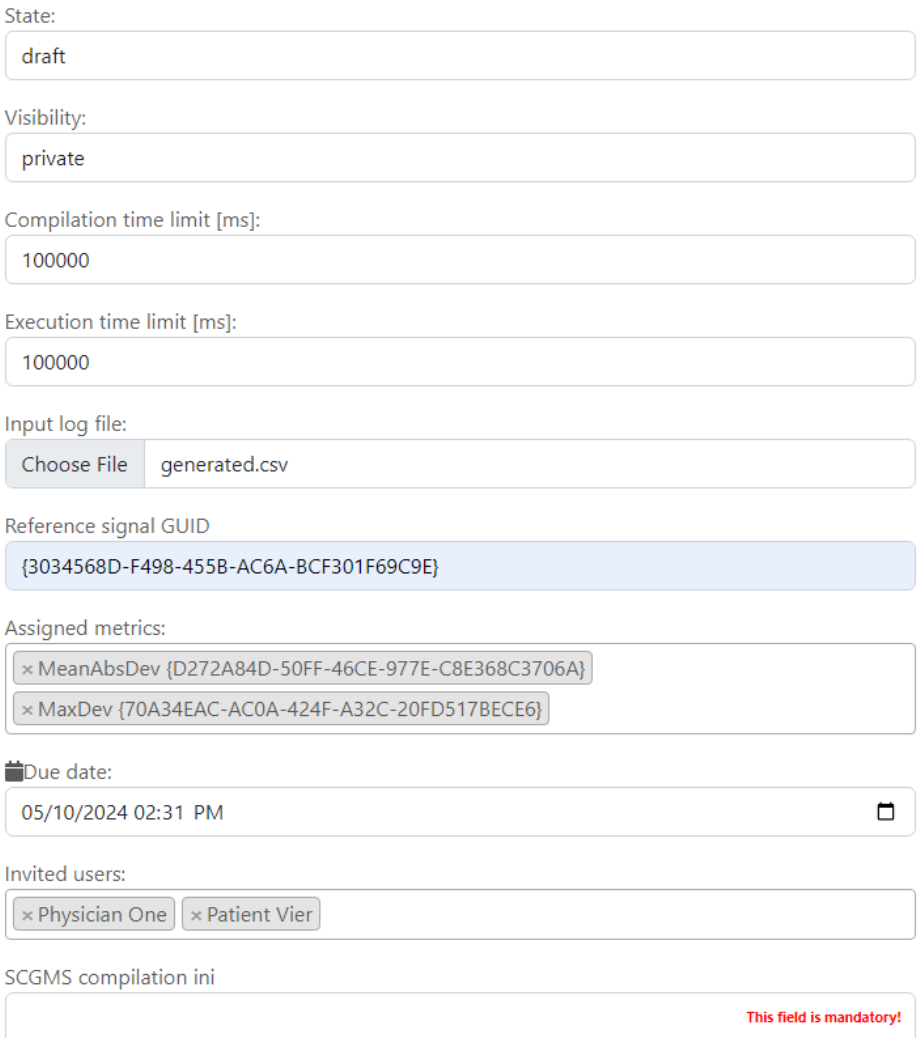

Obrázek C.2: Pohled na všechny dostupné soutěže pro běžného uživatele




Obrázek C.3: Detailnější pohled na soutěž obsahující nejen její hlavní informace ale i výsledkovou tabulku



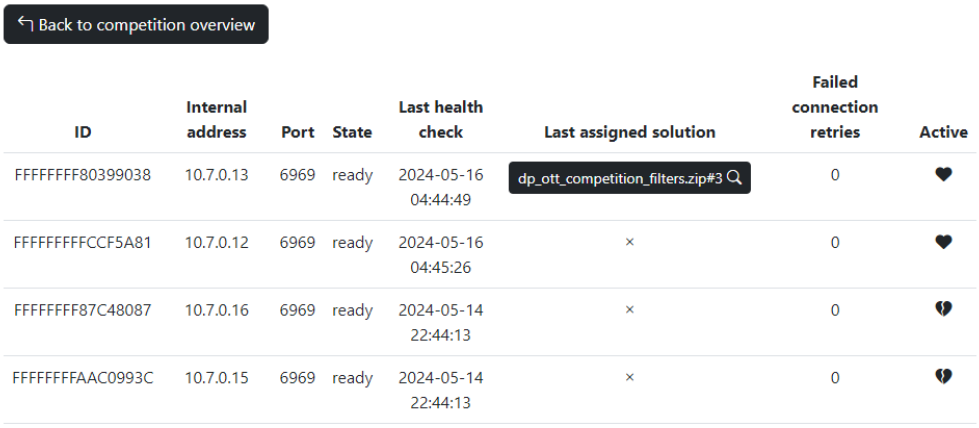

Obrázek C.4: Ukázkový obsah emailu oznamující uživateli, že byl přidán do soutěže




Obrázek C.5: Ukázka části formuláře pro přidávání nových soutěží. Na obrázku z důsledku velikosti chybí pole pro název soutěže, popis a .ini konfiguraci Smart-CGMS

### C.1.2 Worker uzly

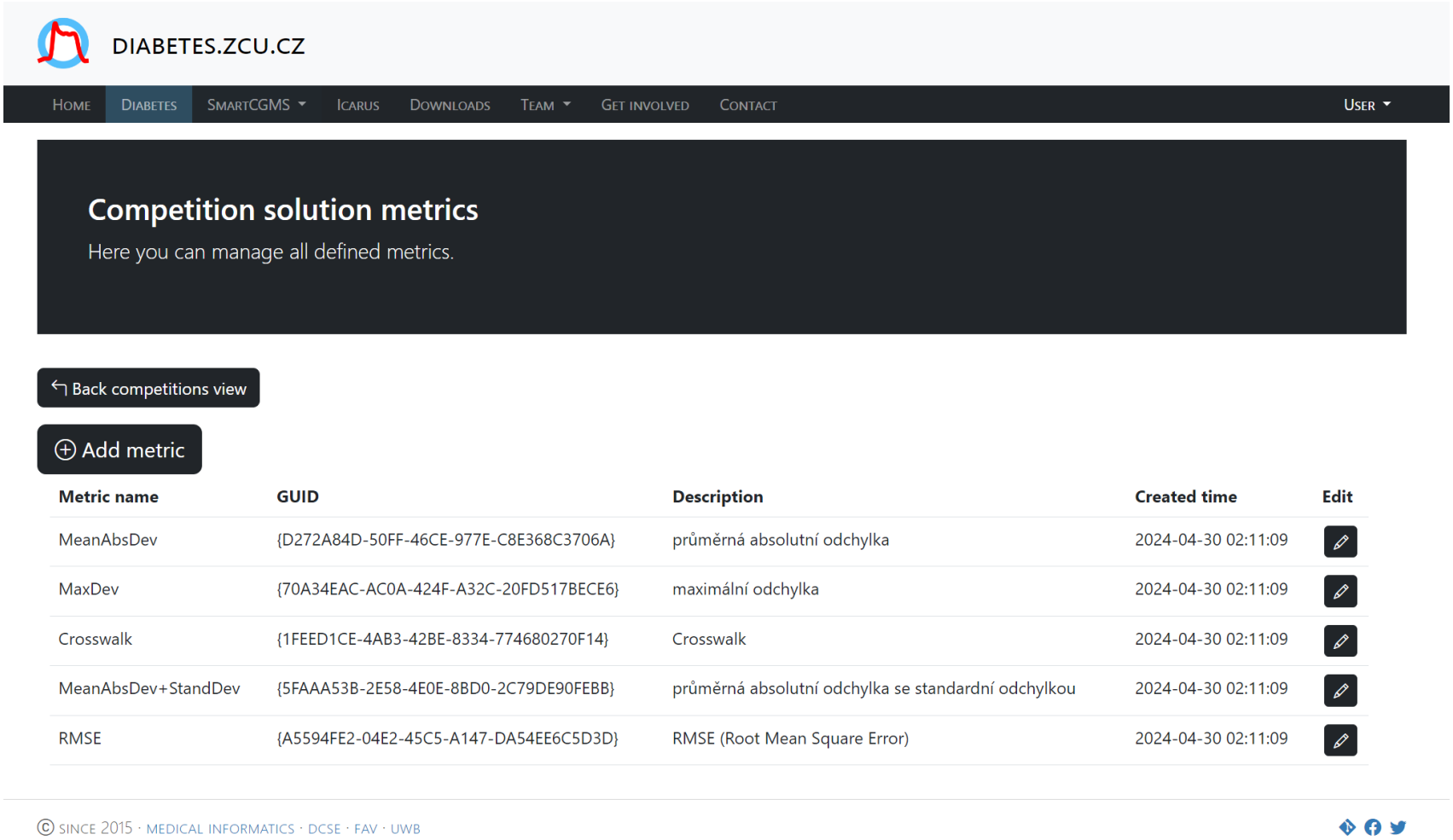
Pro přehled o aktuálně připojených zařízení a jejich stavů běhu je ve webovém rozhraní dostupná pro administrátory stránka přístupná přes tlačítko '**Workers**' dostupného na úvodní obrazovce (obrázek C.1). K zahlédnutí jsou informace o přiřazeném id, interní adresy izolované docker sítě, poslouchající port, aktuální stav a zdali je Worker uzel úspěšně připojený. Také je možné se prokliknout na poslední zpracovávané řešení (dále v sekci C.1.4) skrze tlačítko v kolonce 'Last assigned solution'.



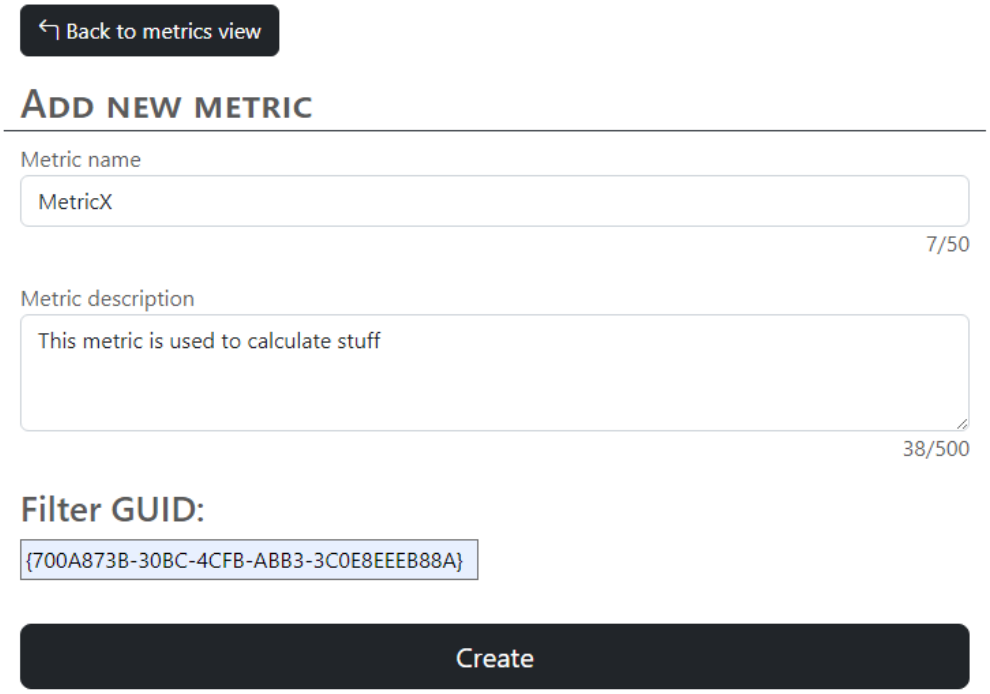

Obrázek C.6: Pohled na všechny připojené Worker uzly

## C.1.3 Metriky

Součástí každé soutěže je sada metrik sloužící jako ukazatel kvality řešení. Samotné entity jsou definovány ve zdrojovém kódu SmartCGMS, ale aby mohli být použity i checker službou, musí být dodatečně přes webové rozhraní zaregistrovány. Seznam již zaregistrovaných metrik je k zahlédnutí na obrázku C.7. Z této stránky je možné přejít pomocí tlačítka '**Add metric**' na formulář určený k definování nových metrik zobrazený na obrázku C.8.




Obrázek C.7: Pohled na všechny dostupné metriky systému SmartCGMS




Obrázek C.8: Formulář pro vytváření nových metrik

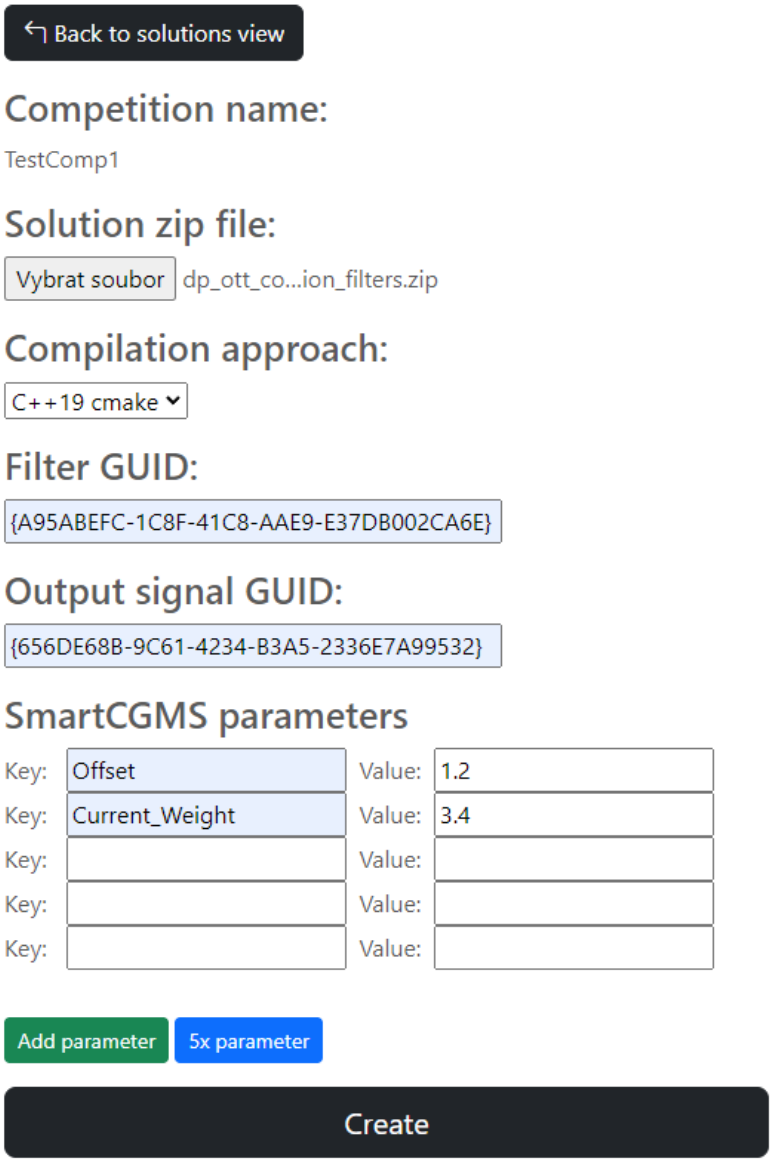
### C.1.4 Uživatelské řešení

V případě, že je uživatel zaregistrován do soutěže může nad touto soutěží vytvářet řešení v podobě archivovaných zdrojových souborů, které jsou předané checker službě ke kompilaci a následnou evaluaci. Při stisknutí tlačítka '**Show solutions**' je uživatel přesměrován na seznam svých vložených řešení zobrazeno na obrázku C.12. V případě, že se jedná o uživatele s administrátorskými právy může nahlédnout na libovolné řešení v soutěži.

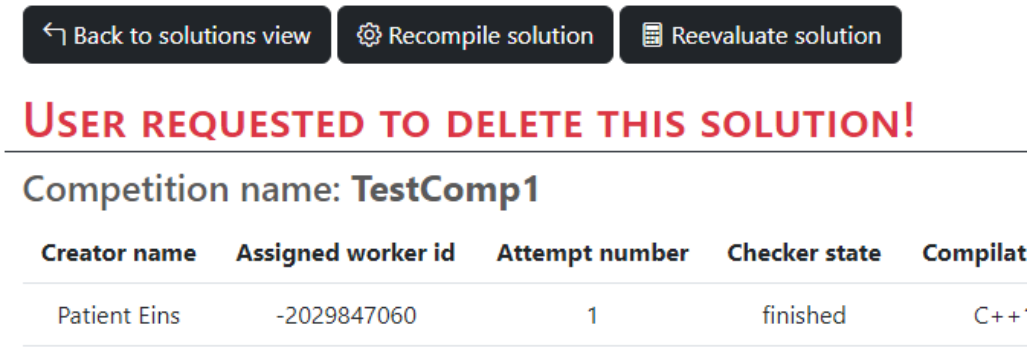
Zde při stisknutí tlačítka '**Add new solution**' je uživatel přesměrován na formulář pro vytvoření nového řešení, které k zahlédnutí na obrázku C.9.

Při zmáčknutí tlačítka s ikonkou lupy u již existujícího řešení, je uživatel přesměrován na detailnější pohled. Zde kromě zobrazení zadaných informací a aktuálního stavu řešení v systému, má tu možnost si stáhnout nahraný soubor spolu s logy ze sestavení. Po úspěšném dokončení výpočtu jsou zde vidět i kompletní statistiky, spolu s výslednými hodnotami z přiřazených metrik soutěže. Uživatel může své řešení smazat, ale pokud není vlastníkem administrátorských práv, je řešení pouze označeno jako smazané (obrázek C.10) a administrátor je požádán formou emailu (obrázek C.11), aby toto řešení smazal.

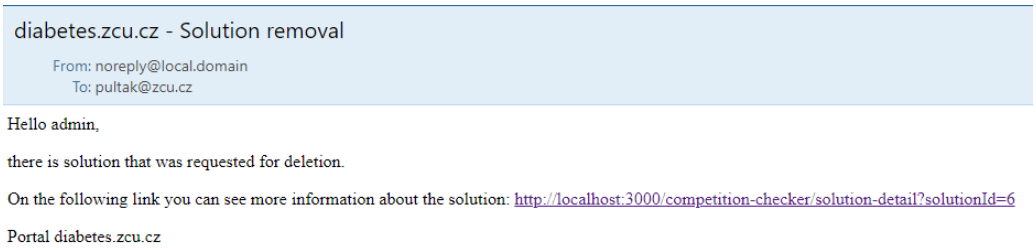
Hodnoty statistik chodu jsou detailněji vysvětleny v sekci č.6.1.



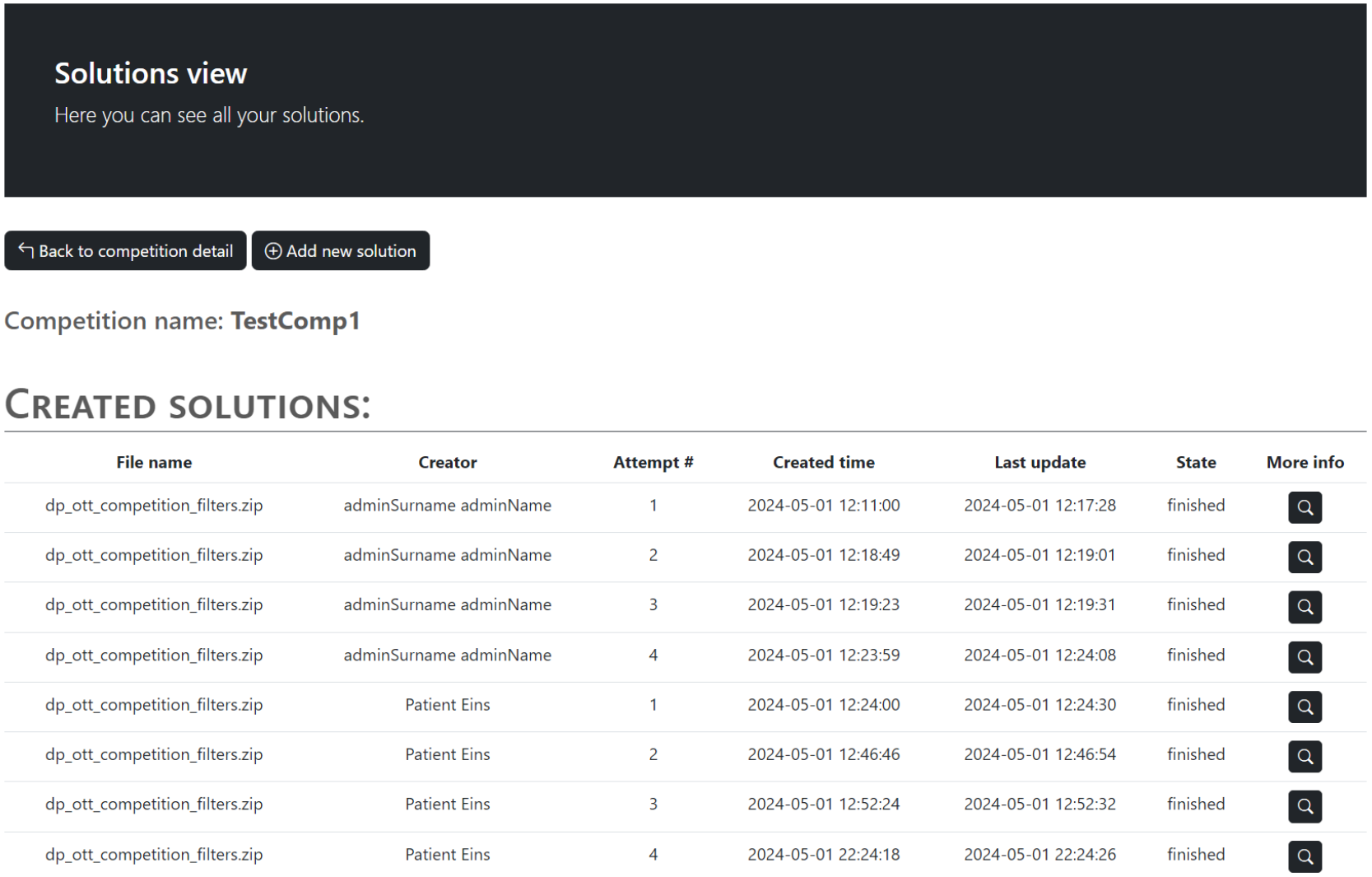

Obrázek C.9: Formulář pro vytváření nových řešení



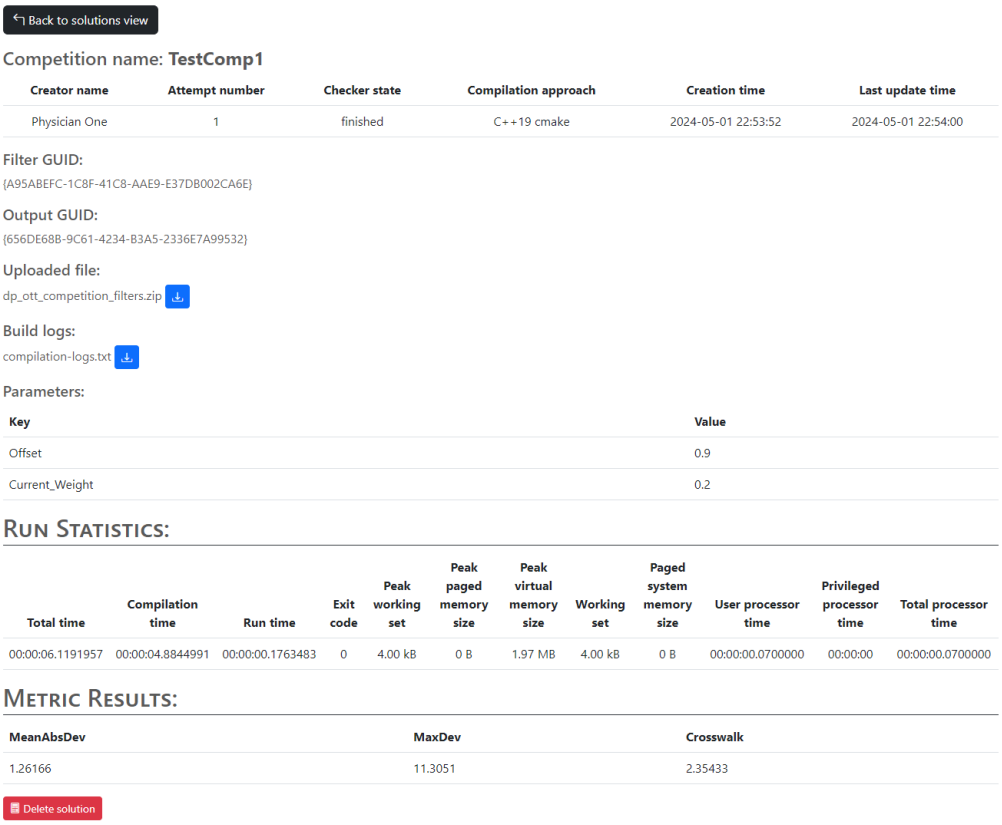

Obrázek C.10: Označení řešení z pohledu administrátora pokud si jeho vlastník vyžádal jeho smazání




Obrázek C.11: Ukázkový obsah emailu oznamující administrátorovi, že si uživatel přeje smazat své vložené řešení




Obrázek C.12: Pohled na všechny vložené řešení pod zvolenou soutěží z pozice uživatele s administrátorskými právy



Back to solutions view

Competition name: **TestComp1**

| Creator name | Attempt number | Checker state | Compilation approach | Creation time | Last update time |
|---|---|---|---|---|---|
| Physician One | 1 | finished | C++19 cmake | 2024-05-01 22:53:52 | 2024-05-01 22:54:00 |

Filter GUID:
{A95ABEFC-1C8F-41C8-AAE9-E37DB002CA6E}

Output GUID:
{656DE68B-9C61-4234-B3A5-2336E7A99532}

Uploaded file:
dp_ott_competition_filters.zip

Build logs:
compilation-logs.txt

Parameters:

| Key | Value |
|---|---|
| Offset | 0.9 |
| Current_Weight | 0.2 |

Run Statistics:

| Total time | Compilation time | Run time | Exit code | Peak working set | Peak paged memory size | Peak virtual memory size | Working set | Paged system memory size | User processor time | Privileged processor time | Total processor time |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 00:00:06.1191957 | 00:00:04.8844991 | 00:00:00.1763483 | 0 | 4.00 kB | 0 B | 1.97 MB | 4.00 kB | 0 B | 00:00:00.0700000 | 00:00:00 | 00:00:00.0700000 |

Metric Results:

| MeanAbsDev | MaxDev | Crosswalk |
|---|---|---|
| 1.26166 | 11.3051 | 2.35433 |

Delete solution

Obrázek C.13: Detailnější pohled vloženého řešení uživatele

## C.2 Rozšířená konfigurace diabetes.zcu

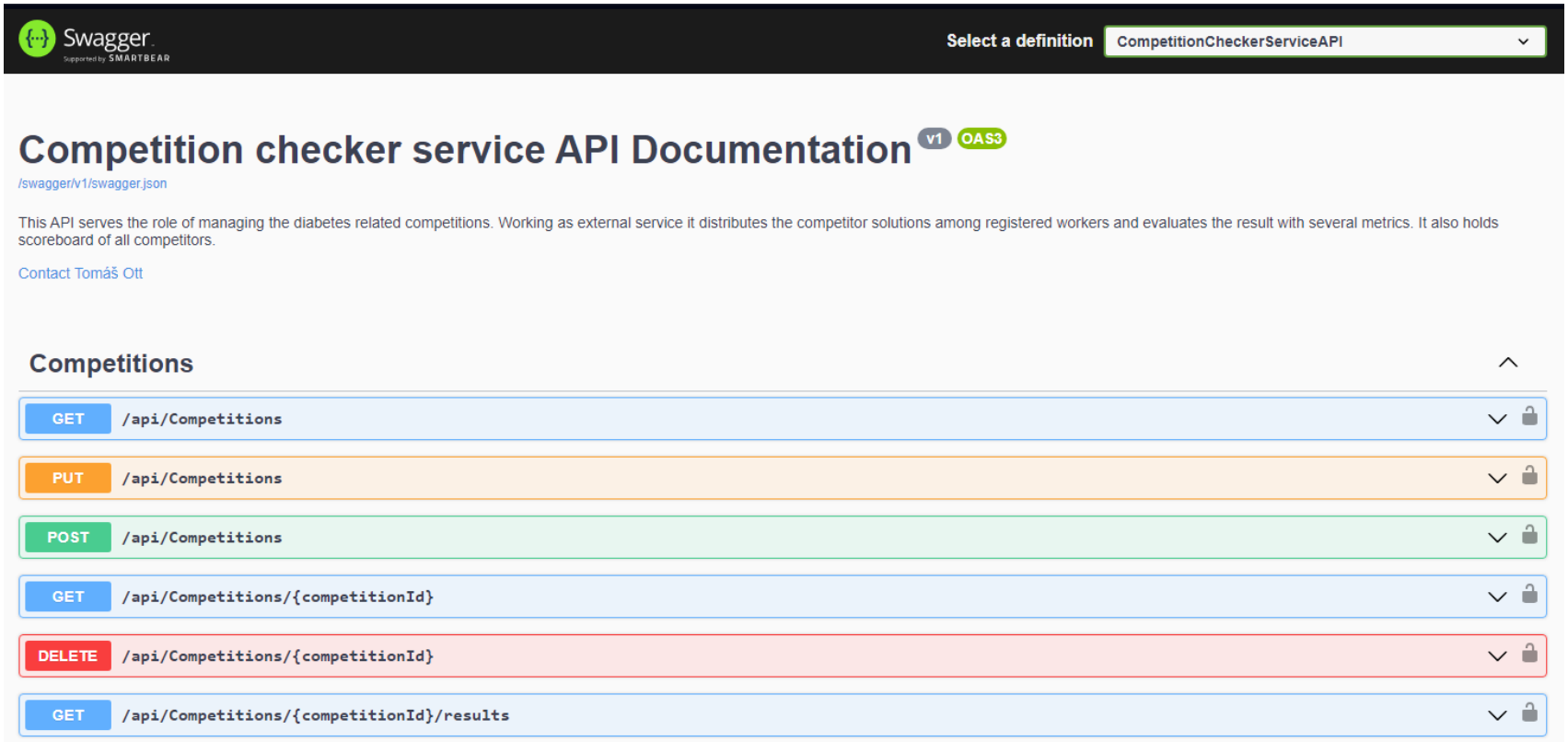
K již existující konfiguraci `local.neon` webového portálu `diabetes.zcu` bylo přidáno několik nových parametrů. Jedná se o následující:

```
.....
competition_checker:
    base_url: https://147.228.173.224:7004/api/
    connection_timeout: 500
    request_timeout: 5000
    check_ssl_certificate: true
    cert_key_file: '/usr/certificate/private_key.pem'
    cert_file: '/usr/certificate/public_key.crt'
    solution_file_pattern: ".+.zip$"
    ......
```

- `competition_checker.base_url` - základ adresy každého požadavku frontend aplikace odkazující na checker službu.
- `competition_checker.connection_timeout` - interval ve kterém pokud nedojde k navázaní komunikace, tak pokus o připojení přeruší.
- `competition_checker.request_timeout` - interval stanovující maximální možnou dobu vykonávání požadavku.
- `competition_checker.check_ssl_certificate` - přepínač stanovující zdali se má certifikát checker služby ověřovat z pohledu digitálního podpisu certifikační autority. Pro využití self-signed certifikátu, musí být tato proměnná nastavena na `false`.
- `competition_checker.cert_file` - cesta k certifikačnímu souboru ve formátu `.pem` přiloženého k požadavku.
- `competition_checker.cert_key_file` - cesta k privátnímu klíči certifikátu potřebného pro podepsání obsahu požadavku.
- `competition_checker.solution_file_pattern` - regex vzor pro vložitelné soubory.

## C.3 OpenAPI

Rozhraní checker služby je definováno veřejně rozšířeným standardem OpenAPI. Uživatel v případě, že se prověřil nastaveným uživatelských certifikátem může provolávat jednotlivé přístupové body přes webové prostředí, podobně jako kdyby tak činila frontend aplikace. Kromě endpointů na obrázku C.14 je možné zahlédnout všechny další zmíněné ze sekce č.6.1.




Obrázek C.14: Ukázka několika dostupných endpointů API checker služby skrze standard OpenAPI

## C.4 Konfigurace Checker služby

Systém vyžaduje za určitých podmínek několik konfigurovatelných proměnných. I checker služba jich vyžaduje několik ve standardním `asp.net` konfiguračním souboru `appsettings.json`. Pro spuštění samotné checker služby je využito souboru `docker-compose-service.yml`, který jak již název napovídá je důležitou konfigurací pro nástroj `docker compose`.

### C.4.1 docker-compose-service.yml

Pro sestavení a konfiguraci uzlu checker služby je využito nástroje `docker compose` a také tedy i jeho konfiguračním souborem `docker-compose-service.yml`. Obsahuje spoustu proměnných ohledně potřebných sítí, použitého docker image, konfigurace pro sestavení v podobě `Dockerfile` a připojených adresářů. Z administrátorského pohledu jsou však především důležité následující proměnné prostředí:

```
.....
services:
  checker_service:
    ......
    environment:
      - ASPNETCORE_URLS=https://0.0.0.0:7004
      - DOTNET_URLS=https://0.0.0.0:7004
      - WORKER_CERTIFICATE_ISSUER=CN=Checker-worker
      - ASPNETCORE_Kestrel__Certificates__Default__Password=
                                                    password
      - ASPNETCORE_Kestrel__Certificates__Default__Path=
                                    /usr/cert/aspnetapp.pfx
    ......
```

- `ASPNETCORE_URLS` a `DOTNET_URLS` - nastavení adresy na které bude checker služba poslouchat HTTP požadavky.
- `WORKER_CERTIFICATE_ISSUER` - certifikační autorita pod kterou je Worker uzel podepsán. Umožňující připojení k jinak nedostupným API endpointům.
- `ASPNETCORE_Kestrel__Certificates__Default__Path` - cesta k certifikačnímu souboru potřebného pro zprovoznění ssl komunikace API.
- `ASPNETCORE_Kestrel__Certificates__Default__Password` - heslo potřebné k přístupu k certifikátu.

### C.4.2 appsettings.json

```
{
  "ConnectionStrings": {
    "DefaultConnection": "DataSource=/var/data/compCheck.db"
  },
  "Serilog": {
    "Using": [ "Serilog.Sinks.Console" ],
    "MinimumLevel": {
      "Default": "Verbose",
      "Override": {
        .......
      }
    },
    "WriteTo": [.......],
    "Enrich": [.......],
    "Destructure": [.......],
    "Properties": {.......}
  },
  "AllowedHosts": "147.228.173.*",
  "WorkerService": {
    "DefaultWorkerCheckInterval": 5000,
    "ReceiveTimeout": 100000,
    "SendTimeout": 100000,
    "UnhealthyInterval": 600000,
    "UpdateCheckLimit": 100,
    "WorkerHandlingInterval": 100000,
    "WorkerMaxConnectionRetries": 5
  },
  "FileSystemProperties": {
    "VolumePath": "/var/nfs/",
    "SolutionsSubFolder": "solutions/",
    "CompetitionsSubFolder": "competitions/"
  },
  "SCGMS": {
    "BuildRootPath": "/home/app/scgms/build",
    "Executable": "scgms-console",
    "ErrorOutputGuid":
            "{E0875A1D-3388-4466-BADF-A24A84D778C1}"
  }
}
```

- `ConnectionStrings.DefaultConnection` - definuje inicializační řetězec pro připojení k libovolné databázi. V aktuálním stavu systému je podporována pouze databáze SQLite, ale při menší konfiguraci a ladění EF, může být použita i jakákoliv libovolná databáze.
- `Serilog` - poměrně rozsáhlá sekce konfigurace aktuálně používaného loggeru `Serilog`. [38]

- `AllowedHosts` - nastavení povolených adres mechanizmu `CORS`.
- `WorkerService.ReceiveTimeout` - časový interval po kterém dojde k vypršení vstupní síťové transakce.
- `WorkerService.SendTimeout` - časový interval po kterém dojde k vypršení výstupní síťové transakce.
- `WorkerService.UnhealthyInterval` - časový interval po kterém dojde k označení připojeného uzlu jako neaktivní. Pokud se uzel pokusí ke službe připojit bude tak muset učinit skrze novou registraci.
- `WorkerService.UpdateCheckLimit` - maximální počet vkuse zpracovávaných požadavků změn stavů uzlů. Poté dojde k umožnění práce ostatním uzlům.
- `WorkerService.WorkerHandlingInterval` - časový interval po kterém dochází k ověřování zdraví připojených worker uzlů.
- `WorkerService.WorkerMaxConnectionRetries` - maximální počet neúspěšných pokusů o předání pracovní nálože.
- `FileSystemProperties.VolumePath` - cesta k NFS mount adresáři ve formě docker volume obsahující důležité soubory k překladu a následně pro jeho samotné výsledky.
- `FileSystemProperties.SolutionsSubFolder` - název adresáře obsahující vstupní data soutěží.
- `FileSystemProperties.CompetitionsSubFolder` - název adresáře obsahující podsložky zdrojové soubory a výsledky jednotlivých řešení.
- `SCGMS.BuildRootPath` - cesta k adresáři sestaveného balíčku systému SmartCGMS.
- `SCGMS.Executable` - definuje název spustitelného programu SmartCGMS konzole, jenž byl v rámci překladu vytvořen.
- `SCGMS.ErrorOutputGuid` - stanovený GUID metrik výstupního SmartCGMS .csv logu.

## C.5 Konfigurace Checker worker

Nedílnou součástí Worker uzlu je i jeho konfigurace. Jedná se o konfiguraci předávanou při samotném vytváření uzlu pomocí souboru `docker-compose-worker.yml` a také před každým spuštěním programu ve standardním `asp.net` konfiguračním souboru `appsettings.json`. Podobně jako je tomu u konfigurace checker služby, je možné spustit obě služby najednou přes definici docker-compose.yml, ale tu nepodporuje `docker swarm` pro distribuované řešení.

### C.5.1 docker-compose-worker.yml

Pro orchestraci Worker uzlů mezi několika oddělenými fyzickými či virtuálními uzly v síti je využito docker swarm za pomoci definice docker compose. Ta může obsahovat spoustu proměnných co se týče nastavení samotného kontejneru, ale v rámci nastavení funkcionality Worker uzlů jsou podstatné pouze proměnné prostředí a to následující:

- `CHECKER_WORKER_PORT` - stanovení portu, na kterém bude Worker uzel poslouchat při čekání na práci přicházející od checker služby.
- `CHECKER_WORKER_ADDRESS` - stanovení adresy a nebo pouze jejího sufixu, na kterém bude možné poslouchat pracovní příkazy. Adresy musí souhlasit se záznamy z `dns` nastavení uzlu jinak nebude aplikace spuštěna. Hodnota může být i localhost nebo null, použitelná je však pouze ve vývojovém prostředí.
- `CHECKER_SERVICE_ADDRESS` - adresa na kterou se pokusí Worker připojit a poté dále komunikovat s checker službou.

Dále je možné nastavit způsob distribuovaného nasazení v sekci `deploy` a

- `mode` - způsob jakým bude aplikace nasazena. Standardní je mód `replicated` (pool uzlů náhodně rozložen), ale může být nastaven jako `global` (jeden nový uzel pro každý reálný uzel), ale bez stanovení následujícího parametru `replicas`. [43]. Může být však změněn i za chodu například příkazem `docker service update --replicas 6 nazev_service`
- `replicas` - počet uzlů, který mají být rozpoloženy přes všechny připojené docker swarm uzly.

```
.....
services:
 competition_checker_worker:
    .......
    environment:
        - CHECKER_WORKER_PORT=6969
        - CHECKER_WORKER_ADDRESS=10.7.
        - CHECKER_SERVICE_ADDRESS=10.7.0.6:7004
    deploy:
       mode: replicated
       replicas: 2
    .....
```

### C.5.2 appsettings.json

```
{
  "SCGMS": {
    "CoreFolder": "core",
    "Executable": "scgms-console"
  },
  "FileSystemProperties": {
    "VolumePath": "/var/nfs/",
    "SolutionsSubFolder": "solutions/",
    "CompetitionsSubFolder": "competitions/"
  },
  "Checker": {
    "HealthCheckInterval": 60000,
    "ClientCertificatePath": "/usr/cert/worker.pfx",
    "ClientCertificatePassword": "password",
    "SandboxPath": "/var/app/sandbox",
    "SandboxTemplatePath": "/var/app/scgms",
    "SandboxExecutableLocation": "build/",
    "CheckerUser": "app",
    "CleanSandboxAfterExecution": false,
    "RestartOnUnknownFailure": true,
    "CompilationCommands": {
      "Cpp17Cmake": {
        "ExecutableNames": [ "cmake", "make" ],
        "ExecutableArguments": [
          "-S .. -DBUILD_TESTS=OFF -DBUILD_EXAMPLES=OFF ....",
          "-j4"
        ]
      },
      "anotherApproach": {.......}
    }
  }
}
```

- `SCGMS.CoreFolder` - definuje core adresář obsahující další SmartCGMS filtr entity ke kterým bude přikopírováno a následně přeloženo vložené uživatelské řešení.
- `SCGMS.Executable` - definuje název spustitelného programu SmartCGMS konzole, jenž byl v rámci překladu vytvořen.
- `FileSystemProperties.VolumePath` - cesta k NFS mount adresáři ve formě docker volume obsahující důležité soubory k překladu a následně pro jeho samotné výsledky.
- `FileSystemProperties.SolutionsSubFolder` - název adresáře obsahující vstupní data soutěží.
- `FileSystemProperties.CompetitionsSubFolder` - název adresáře obsahující podsložky zdrojové soubory a výsledky jednotlivých řešení.
- `Checker.HealthCheckInterval` - interval po kterém Worker uzel dokazuje svoje zdraví běhu.
- `Checker.ClientCertificatePath` - cesta ke klientskému certifikátu ve formátu `.pfx` použitý ke authorizaci u checker služby.
- `Checker.ClientCertificatePassword` - heslo potřebné k využití klientského certifikátu.
- `Checker.SandboxPath` - cesta k neexistujícímu adresáři nad kterým se provádí process kompilace a spouštění programu SmartCGMS.
- `Checker.SandboxTemplatePath` - cesta k šabloně sandboxu jenž je nakopírována před každým spuštěním celého procesu.
- `Checker.SandboxExecutableLocation` - pracovní adresář ve kterém se nachází spustitelný program SmartCGMS konzole.
- `Checker.CheckerUser` - uživatel nad kterým se budou spouštět procesy kompilace a vyhodnocení. V případě, že je zanechán prázdný jsou procesy spouštěny s root právy.
- `Checker.CleanSandboxAfterExecution` - přepínač zdali má být sandbox adresář smazát po dokončení procesu a nebo zdali se bude mazat až před dalším spuštěním.
- `Checker.RestartOnUnknownFailure` - přepínač zdali se má Worker uzel vyřadit z provozu při neznámé chybě a nebo pokračovat dál v běžném chodu.

Může být užitečné především z důvodu odladění nové funkcionality, aby nedocházelo ke zbytečnému znehodnocení již nějakým způsobem porušených dat.

- `Checker.CompilationCommands` - sada překládacích přístupů použité sekvečně při samotné kompilaci
  - `Cpp17Cmake` - hlavní a momentálně jediný použitelný způsob kompilace. Je definován sadou přikazů `ExecutableNames` a jejich argumentů `ExecutableArguments`.
  - Další způsoby mohou být přidány z důvodu bezpečnosti až po další změně kódu

# Obsah odevzdaného adresáře

# D

Součástí elektronického odevzdání diplomové práci je adresář s elektronickou podobou textu, zdrojovými kódy, spustitelným Checker systémem, a se sadou testovacích souborů pro otestování funkčnosti soutěžního systému. Také je v adresáři přiložený poster a výsledky získaných při naměření včetně vygenerovaných dat Checker systému.

- **Aplikace_a_knihovny** – adresář obsahující zdrojové kódy aplikací
  - **diabetes-zcu-cz-dev** – zdrojové kódy a inicializační data webového portálu diabetes.zcu.cz. Struktura je dále popsána v sekci D.1.
  - **checker-service** – zdrojové kódy komponent Checker služby. Struktura dále popsána v sekci D.2.
  - **checker-service-bin** – spustitelná verze Checker služby vyžadující konfiguraci lokálního prostředí podle příloh B a C.
  - **checker-worker-bin** – spustitelná verze Worker uzlu vyžadující konfiguraci lokálního prostředí podle příloh B a C.
- **Poster** – složka obsahující poster ve formátech .pdf a .pub
- **Text_prace** – adresář obsahující zdrojové kódy LaTex a z něho vygenerovaná elektronická podoba textu.
- **Vstupni_data** – adresář obsahující sadu vstupních data určených pro vložení do Checker systému přes webové rozhraní.
- **Vysledky** – adresář obsahující výsledky měření a vygenerované soubory Checker systémem.

## D.1 Checker služba

- **certificate/** – certifikáty a privátní klíče potřebné jak pro zprovoznění HTTPS komunikace, tak i autentizace připojovaných uzlů pomocí klientského certifikátu.
- **CompetitionChecker/** – zdrojové kódy všech komponent Checker systému. Je rozdělena na tři projekty Service(Checker služba), Shared(sdílené objekty) a Worker (Worker uzel).
- **test-data/** – testovací data určená pro vklad do systému přes webové rozhraní.
- **.dockerignore** – soubor definující jaké soubory nemají být kopírovány během vytváření docker kontejnerů.
- **CompetitionCheckerService.sln** – Visual studio řešení definující jak celý systém složen a jak má být sestaven.
- **docker-compose.yml** – docker-compose konfigurace pro spuštění lokálního řešení Checker systému.
- **docker-compose-service.yml** – docker-compose konfigurace pro spuštění distribuovaného řešení části Checker služby.
- **docker-compose-worker.yml** – docker-compose konfigurace pro spuštění distribuovaného řešení části Work uzlu.
- **readme.md** – textový soubor obsahující sadu užitečných příkazů pro správu a sestavení distribuovaného systému.
- **ScgmsNetBase.DockerFile** – definice docker image sloužící jako základ pro definice Service.DockerFile a Worker.DockerFile.
- **Service.DockerFile** – definice docker image pro vytvoření kontejneru Checker služby.
- **Worker.DockerFile** – definice docker image pro vytvoření kontejneru Workera.

## D.2 Webový portál diabetes.zcu.cz

V následujícím seznamu je popsána struktura zdrojových kódů webového portálu diabetes.zcu.cz. Jelikož se jedná o již existující aplikaci, jsou jednotlivé soubory a adresáře označeny znakem * indikující změnu provedenou během implementace této diplomové práce.

- app/ – zdrojové kódu portálu diabetes.zcu
  - config/
    - ***common.neon** – obecná definice struktury konfiguračního souboru
    - ***local.neon** – konfigurační souboru obsahující všechny konfigurovatelné proměnné systému.
  - Presenters/
    - templates/ – Všechny latte šablony pro definované prezentery
      - ***CompetitionChecker/** – Latte šablony pro soutěžní modul
    - ***CompetitionCheckerPresenter.php** – Prezenter zodpovědný za funkci soutěžního modulu.
  - Services/
    - ***CompetitionHttpService.php** – Konektor pro spojení s REST API Checker služby.
    - templates/ – všechny latte šablony pro mailovací službu.
      - ***competition-invitation.latte** – email šablona pro vytvoření pozvánky do soutěže.
      - ***solution-removal.latte** – email šablona určená pro oznámení žádost o odstranění řešení.
  - Utils/ – adresář obsahující podpurné funkce a třídy.
    - **CheckerCompilationApproach.php** – Enum třída pro správu možný kompilačních přístupů.
    - **CheckerState.php** – Enum třída pro správu možných stavů checker systému.
    - **CompetitionForm.php** – Třída pro vytváření formulářů pro vytváření/editaci soutěží.
    - **CompetitionState.php** – Třída pro vytváření formulářů pro vytváření/editaci soutěží.
    - **CompetitionVisibility.php** – Enum třída pro správu možný viditelností soutěže.

    - **DateUtils.php** – Třída pro ovládání a validaci datum. Nově přidána metoda pro úpravu času.
    - **MetricForm.php** – Třída pro vytváření formulářů pro vytváření/editaci metrik.
    - **SolutionForm.php** – Třída pro vytváření formulářů pro vytváření řešení.
    - **StringUtils.php** – Třída pro pomocné funkce pro práci s řetězci.
- bin/ – Spustitelné php skripty.
- ***certificate/** – Vzorový klientský certifikát potřebný pro komunikaci s API Checker služby (self-signed pouze pro vývoj)
- db/ – Sada databázových skriptů pro inicializaci.
  - ***dp_data_init.sql** – Inicializační sql script pro databázi naplněnou základními uživatelskými daty.
- ***docker-compose.yaml** – docker-compose konfigurace pro nasazení databáze a mail catcheru.
- img/ – Adresář se systémem využitých obrázků.
- log/ – Výstupní složka pro logování chybových zpráv.
- www/ – Veřejná složka při spuštěném webovém serveru.
- .htaccess – Konfigurace přístupových práv Apache serveru.
- composer.json – Specifikace závislostí projektu pomocí nástroj Composer.

# Bibliografie


1. SMITH, Ken G.; FERRIER, Walter J.; NDOFOR, Hermann. Competitive Dynamics Research. In: *The Blackwell Handbook of Strategic Management*. John Wiley  Sons, Ltd, 2005, kap. 11, s. 309–354. ISBN 9781405164023. Dostupné z DOI: https://doi.org/10.1111/b.9780631218616.2006.00012.x.
2. MILLER, Stephan. *What Is Competitive Programming?* Code academy, 2021. Dostupné také z: https://www.codecademy.com/resources/blog/what-is-competitive-programming/. [citováno 06.05.2024].
3. *About ICPC*. ICPC, 2024. Dostupné také z: https://icpc.global. [citováno 06.05.2024].
4. VILCA, HDavid Calderon. Evaluation of source code in ACM ICPC style programming and training competitions. *ClbSE*. 2018, s. 208–223.
5. STANISLAV, Trojan. *Lékařská fyziologie*. Grada Publishing, 2003. ISBN 80-247-0512-5.
6. BORLE, Neil C. *Blood Glucose Level Prediction Challenge*. ECAI, 2020. Dostupné také z: https://sites.google.com/view/kdh-2020/bglp-challenge. [citováno 04.05.2024].
7. MARLING, Cindy; BUNESCU, Razvan. The OhioT1DM Dataset for Blood Glucose Level Prediction: Update 2020. *CEUR Workshop Proc.* 2020.
8. *Overfitting*. IBM, 2024. Dostupné také z: https://www.ibm.com/topics/overfitting. [citováno 06.05.2024].
9. HEALTHCARE INFORMATICS, IEEE International Conference on. *Blood Glucose Level Prediction Results*. SmartHealth Ohio. Dostupné také z: http://smarthealth.cs.ohio.edu/bglp/bglp-results.html. [citováno 01.05.2024].
10. ÚBL, Martin. *Officiální dokumentace SmartCGMS*. Faculty of Applied Sciences Western Bohemian University, 2022. Dostupné také z: https://diabetes.zcu.cz/smartcgms/. [citováno 01.05.2024].

11. KUHL, Frederick; WEATHERLY, Richard; DAHMANN, Judith. *Creating Computer Simulation Systems: An Introduction to the High Level Architecture*. Prentice Hall, 1999. ISBN 0130225118.

12. KOUTNÝ, Tomáš; ÚBL, Martin. Parallel software architecture for the next generation of glucose monitoring. 2018, "279–286". Dostupné z DOI: "https://doi.org/10.1016/j.procs.2018.10.197".

13. KOUTNÝ, Tomáš; ÚBL, Martin. SmartCGMS as an Environment for an InsulinPump Development with FDA-Accepted In-Silico Pre-Clinical Trials. 2019, "322–329". Dostupné z DOI: "https://doi.org/10.1016/j.procs.2019.11.084".

14. MICROSOFT. *Component Object Model (COM)*. Microsoft. Dostupné také z: https://learn.microsoft.com/en-us/windows/win32/com/component-object-model--com--portal. [citováno 07.05.2024].

15. KOUTNÝ, Tomáš; ÚBL, Martin. *SmartCGMS github*. Github, 2023. Dostupné také z: https://diabetes.zcu.cz/smartcgms/. [citováno 01.05.2024].

16. LIPPMAN, Stanley B.; LAJOIE, Josée; MOO, Barbara E. *C++ Primer (5th Edition)*. Addison-Wesley Professional, 2012. ISBN 9780321714114.

17. MALLIA, Antonio; ZOFFOLI, Francesco. *C++ Fundamentals*. Packt Publishing, 2019. ISBN 978-1-78980-149-1. Dostupné také z: https://subscription.packtpub.com/book/programming/9781789801491/1/ch01lvl1sec03/the-c-compilation-model.

18. HOFFMAN, Vill. *CMake*. Kitware, 2024-05-01. Dostupné také z: https://cmake.org/. [citováno 09.05.2024].

19. LESSARD, Paul. *Linux Process Containment – A practical look at chroot and User Mode Linux*. SANS Institute, 2003. Dostupné také z: https://www.giac.org/paper/gsec/2860/linux-process-containment-practical-chroot-user-mode-linux/104829.

20. MOHANAN, Remya. *What Is Virtualization? Meaning, Types, and Software*. SpiceWorks, 2023. Dostupné také z: https://www.spiceworks.com/tech/devops/articles/what-is-virtualization/.

21. MACPHERSON, Jordan. *What is a Hypervisor? Types, benefits  how does it work*. ParkView Managed Services, 2022. Dostupné také z: https://www.parkplacetechnologies.com/blog/what-is-hypervisor-types-benefits/. [citováno 06.05.2024].

22. MCCARTY, Scott. *A Practical Introduction to Container Terminology*. Red Hat Developer, 2018. Dostupné také z: https://developers.redhat.com/blog/2018/02/22/container-terminology-practical-introduction#containers_101. [citováno 10.05.2024].

23. *Lesson 18: Application Isolation*. OSU DevOps BootCamp, 2017. Dostupné také z: https://devopsbootcamp.osuosl.org/application-isolation.html. [citováno 10.05.2024].
24. IBM. *What Is REST API?* IBM, 2024. Dostupné také z: https://www.ibm.com/topics/rest-apis. [citováno 08.05.2024].
25. DOCKER, inc. *Docker overview*. Docker, 2024-05-01. Dostupné také z: https://docs.docker.com/get-started/overview/. [citováno 01.05.2024].
26. DOCKER, inc. *Network drivers overview*. Docker, 2024-05-01. Dostupné také z: https://docs.docker.com/network/drivers/. [citováno 01.05.2024].
27. *OSI Model A Complete Guide - 2021 Edition*. The Art of Service - OSI Model Publishing, 2021. ISBN 1867440288.
28. MAO, Ying et al. Resource Management Schemes for Cloud-Native Platforms with Computing Containers of Docker and Kubernetes. 2020. Dostupné z DOI: 10.36227/techrxiv.13146548.v1.
29. MCBRIEN, byScott. *Linux file permissions explained*. RedHat, 2023. Dostupné také z: https://www.redhat.com/sysadmin/linux-file-permissions-explained.
30. SANDBERG, Russel. The Sun Network Filesystem: Design, Implementation and Experience. *The Computer Journal*. 1984.
31. DE, Suman; PANJWAN, Megha. A Comparative Study on Distributed File Systems. *ResearchGate*. 2021 4.
32. SUCHÁNEK, Marek; BHIDE, Apurva; NAVRÁTIL, Milan; EAST, Jacquelynn; DOMINGO, Don. *Securing NFS*. Red Hat, 1984. Dostupné také z: https://learn.microsoft.com/en-us/aspnet/core/security/enforcing-ssl. [citováno 01.05.2024].
33. TOBBICKE, R. Distributed file systems: focus on Andrew File System/Distributed File Service (AFS/DFS). In: *Proceedings Thirteenth IEEE Symposium on Mass Storage Systems. Toward Distributed Storage and Data Management Systems*. 1994, s. 23–26. Dostupné z DOI: 10.1109/MASS.1994.373021.
34. *What is Transport Layer Security (TLS)?* CloudFlare, 2024. Dostupné také z: https://www.cloudflare.com/learning/ssl/transport-layer-security-tls/. [citováno 12.05.2024].
35. *What is three-tier architecture?* IBM, 2024. Dostupné také z: https://www.ibm.com/topics/three-tier-architecture.
36. BOGARD, Jimmy. *AutoMapper*. 2024. Dostupné také z: https://github.com/AutoMapper/AutoMapper. [citováno 10.05.2024].

37. *Dependency injection in ASP.NET Core*. Microsoft. Dostupné také z: `https://learn.microsoft.com/en-us/aspnet/core/fundamentals/dependency-injection`.
38. BLUMHARDT, Nicholas; CROPP, Simon; ERBS, Matthew; HELLANG, Kristian. *Serilog*. 2024. Dostupné také z: `https://github.com/serilog/serilog`. [citováno 01.05.2024].
39. *Entity Framework*. Microsoft, 2024. Dostupné také z: `https://learn.microsoft.com/en-us/aspnet/entity-framework`. [citováno 14.05.2024].
40. *Presentery*. Nette, 2024. Dostupné také z: `https://doc.nette.org/cs/application/presenters`. [citováno 12.05.2024].
41. *Open Nebula*. Západočeská univerzita. Dostupné také z: `https://nuada.zcu.cz`.
42. MICROSOFT. *Enforce HTTPS in ASP.NET Core*. 2024-05-21. Dostupné také z: `https://learn.microsoft.com/en-us/aspnet/core/security/enforcing-ssl`. [citováno 01.05.2024].
43. DOCKER, inc. *Compose docker specification*. Docker, 2024-05-01. Dostupné také z: `https://docs.docker.com/compose/compose-file/deploy/`. [citováno 01.05.2024].

# Přehled použitých zkratek

**A.**

`ACL` - Access Control List
`ACM` - Association for Computing Machinery
`API` - Application programming interface
`ASP.NET` - Active Server Pages Network Enabled Technologies

**B.**

`BGLP` - Blood Glucose Level Prediction

**C.**

`CORS` - Cross-Origin Resource Sharing

**D.**

`DNS` - domain name system
`DUA` - Data Usage Agreement

**G.**

`GCC` - GNU Compiler Collection
`GUI` - Graphical User Interface

**E.**

`EF` - entity framework

**G.**

`GUID` - Global Unique Identifier

**H.**

`HLA` - High Level Architecture

**I.**

`ICPC` - International Collegiate Programming Contest

`ISO/OSI` - International Organization for Standardization/Open Systems Interconnection
`IP` - Internet Protocol

**M.**

`MSVS` - Microsoft Visual Studio

**N.**

`NFS` - Network File System

**R.**

`RPC` - Remote Procedure Call

**S.**

`SCGMS` - Smart Continuous Glucose Monitoring System
`SDK` - Software Development Kit
`SSL` - Secure Sockets Layer

**T.**

`TCP` - Transmission Control Protocol
`TLS` - Transport Layer Security

**U.**

`UDP` - User Datagram Protocol
`UNIX` - Uniplexed Information Computing System

**X.**

`XDR` - External Data Representation

# Seznam obrázků

# Seznam tabulek

# Seznam výpisů