FAKULTA MECHATRONIKY,
INFORMATIKY A MEZIOBOROVÝCH
STUDIÍ TUL

Bakalářská práce

Asynchronní programování v PHP

Studijní program:	B0613A140005 Informační technologie
Studijní obor:	Aplikovaná informatika
Autor práce:	**Adam Petříček**
Vedoucí práce:	Ing. Jana Vitvarová, Ph.D. Ústav mechatroniky a technické informatiky

Liberec 2024

FAKULTA MECHATRONIKY,
INFORMATIKY A MEZIOBOROVÝCH
STUDIÍ TUL

Zadání bakalářské práce

Asynchronní programování v PHP

Jméno a příjmení:	**Adam Petříček**
Osobní číslo:	M21000131
Studijní program:	B0613A140005 Informační technologie
Specializace:	Aplikovaná informatika
Zadávající katedra:	Ústav mechatroniky a technické informatiky
Akademický rok:	2023/2024

Zásady pro vypracování:

1. Seznamte se s principy a výhodami asynchronního programování. Porovnejte asynchronní se synchronním přístupem a analyzujte vhodné případy použití.
2. Seznamte se s asynchronním programováním v PHP a s technologiemi, které ho umožňují implementovat. Vyzkoušejte a porovnejte tyto technologie mezi sebou z hlediska funkcionality, způsobu použití, výkonu a dalších relevantních aspektů.
3. Doplňte porovnání z bodu 2 o další jazyky, které podporují asynchronní programování.
4. Proveďte analýzu bezpečnosti asynchronních aplikací v PHP a identifikujte možná rizika, která by mohla ohrozit jejich funkčnost a integritu. Navrhněte možná řešení.
5. Navrhněte a implementujte ukázkovou aplikaci v PHP, která využívá asynchronní programování pomocí některé z analyzovaných technologií.
6. Popište použitou metodiku vývoje asynchronní aplikace v PHP z bodu 5. Uveďte použité nástroje a knihovny. Popište proces testování, nasazení a další důležité kroky a aspekty vývoje.

Rozsah grafických prací:	dle potřeby dokumentace
Rozsah pracovní zprávy:	30 až 40 stran
Forma zpracování práce:	tištěná/elektronická
Jazyk práce:	čeština

Seznam odborné literatury:

[1] DOU, Bruce. Mastering Swoole PHP: Build high performance concurrent system with async and coroutines. Transfon, 2020. ISBN 978-18-381-3440-2.

[2] BUTTI, Roberto. High Performance with Laravel Octane: Learn to fine-tune and optimize PHP and Laravel apps using Octane and an asynchronous approach. Packt Publishing, 2022. ISBN 978-18-018-1940-4.

Vedoucí práce:	Ing. Jana Vitvarová, Ph.D. Ústav mechatroniky a technické informatiky
Datum zadání práce:	12. října 2023
Předpokládaný termín odevzdání:	14. května 2024

L.S.

prof. Ing. Zdeněk Plíva, Ph.D.
děkan

doc. Ing. Josef Chaloupka, Ph.D.
garant studijního programu

V Liberci dne 12. října 2023

Prohlášení

Prohlašuji, že svou bakalářskou práci jsem vypracoval samostatně jako původní dílo s použitím uvedené literatury a na základě konzultací s vedoucím mé bakalářské práce a konzultantem.

Jsem si vědom toho, že na mou bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb., o právu autorském, zejména § 60 – školní dílo.

Beru na vědomí, že Technická univerzita v Liberci nezasahuje do mých autorských práv užitím mé bakalářské práce pro vnitřní potřebu Technické univerzity v Liberci.

Užiji-li bakalářskou práci nebo poskytnu-li licenci k jejímu využití, jsem si vědom povinnosti informovat o této skutečnosti Technickou univerzitu v Liberci; v tomto případě má Technická univerzita v Liberci právo ode mne požadovat úhradu nákladů, které vynaložila na vytvoření díla, až do jejich skutečné výše.

Současně čestně prohlašuji, že text elektronické podoby práce vložený do IS/STAG se shoduje s textem tištěné podoby práce.

Beru na vědomí, že má bakalářská práce bude zveřejněna Technickou univerzitou v Liberci v souladu s § 47b zákona č. 111/1998 Sb., o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších předpisů.

Jsem si vědom následků, které podle zákona o vysokých školách mohou vyplývat z porušení tohoto prohlášení.

7. května 2024 Adam Petříček

Poděkování

Rád bych poděkoval vedoucí této bakalářské práce Ing. Janě Vitvarové, Ph.D. za poskytnuté konzultace a rady.

Asynchronní programování v PHP

Abstrakt

Tato bakalářská práce se zabývá analýzou konceptu asynchronního programování, porovnáním asynchronních PHP technologií a implementací ukázkové webové aplikace. Asynchronní přístup umožňuje webovému serveru efektivněji zpracovávat příchozí requesty a díky tomu jich odbavit více naráz. V rámci porovnání bylo vybráno 8 technologií, které umožňují implementovat asynchronicitu v PHP. Technologie byly nejprve porovnány na základě faktických rozdílů, poté byl proveden benchmark počtu zpracovaných requestů za sekundu. Vítěz PHP porovnání byl následně zařazen do druhého benchmarku s dalšími programovacími jazyky. Výhody asynchronního přístupu byly demonstrovány vytvořením ukázkové aplikace ve frameworku Laravel. Tato aplikace získává informace o Bitcoin ekosystému z různých API, zpracovává je a poskytuje uživateli v čitelné formě. Automatizované nasazení aplikace je řešeno pomocí GitHub Actions, běhové prostředí je vytvořeno přes Docker.

Klíčová slova: Asynchronní programování, Event Loop, PHP, Docker, benchmark, Bitcoin, Laravel

Asynchronous programming in PHP

Abstract

This bachelor thesis focuses on the analysis of asynchronous programming, comparison of various asynchronous PHP technologies and implementation of a demo web application. The asynchronous approach allows the web server to process incoming requests more efficiently and handle more requests at once. The comparison includes 8 selected asynchronous PHP technologies. These technologies were first compared based on factual differences, followed by a benchmark measuring number of requests processed per second. The winner of the PHP comparison was then placed in a second benchmark with other programming languages. The benefits of the asynchronous approach were demonstrated by creating a demo application in Laravel framework. This application retrieves data about the Bitcoin ecosystem from various APIs, processes it and them provides it to the user in a readable form. Automated deployment of the application is handled using GitHub Actions, and the runtime environment is created via Docker.

Keywords: Asynchronous programming, Event Loop, PHP, Docker, benchmark, Bitcoin, Laravel

Obsah

Seznam obrázků

Seznam tabulek

Seznam zkratek

HTTP	Hypertext Transfer Protocol
IO	Input/Output
HTML	Hypertext Markup Language
AJAX	Asynchronous JavaScript and XML
PHP	Hypertext Preprocessor
CGI	Common Gateway Interface
FPM	FastCGI Process Manager
RFC	Request For Comments
CLI	Command Line Interface
AWS	Amazon Web Services
PECL	PHP Extension Community Library
RPC	Remote Procedure Call
API	Application Programming Interface
RPS	Requests Per Second
BTC	Bitcoin
UX	User Experience
UI	User Interface
REST	Representational State Transfer
HATEOAS	Hypermedia as the Engine of Application State
SPA	Single Page Application
CI/CD	Continuous Integration/Continuous Deployment

1 Úvod

PHP je stále jedním z nejpoužívanějších jazyků pro webové servery. Tradiční PHP kód běží synchronně, tzn. všechny operace blokují hlavní vlákno dokud nejsou dokončeny. Jiné webové technologie (např. Node.js) umožňují kód spouštět asynchronně, díky čemuž může běžet více operací souběžně bez blokace hlavního vlákna. Nabízí se otázka, zda je asynchronní přístup možné implementovat také v PHP, případně jaké to přináší výhody a kdy je to vhodnou volbou.

Rešeršní část této práce začíná kapitolou 2, kde je koncept asynchronního programování nejprve popsán obecně a porovnán se synchronním a paralelním přístupem. Dále je asynchronní programování zasazeno do prostředí webu, včetně nastínění způsobu implementace. Rešeršní část pokračuje kapitolou 3, která už obsahuje konkrétní způsoby implementace asynchronicity v PHP, porovnání s běžným PHP přístupem a možné bezpečnostní hrozby.

Druhou částí práce je porovnání technologií, nachází se v kapitole 4. Obsahuje metodiku výběru porovnávaných technologií a seznam nezahrnutých včetně zdůvodnění. Pro každou vybranou technologii následuje odstavec textu, který ji v krátkosti popisuje. Faktické rozdíly mezi technologiemi jsou vyjádřeny pomocí tabulek. Dále je v rámci této kapitoly popsáno prostředí pro benchmark výkonu, grafy z měření a interpretace výsledků. Kromě porovnání v rámci PHP technologií obsahuje kapitola také rozšíření porovnání o další programovací jazyky.

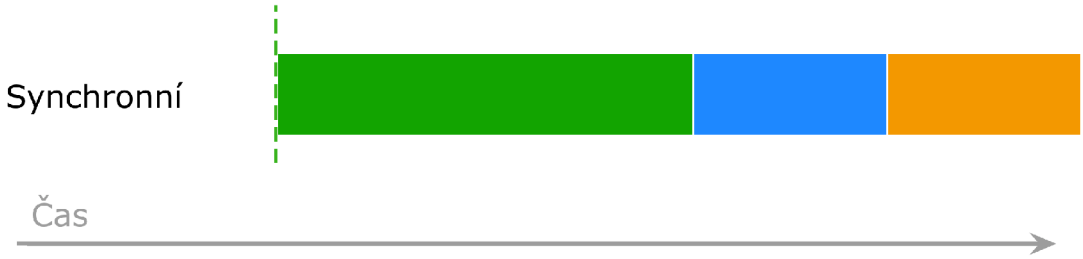
Kapitola 5 se zabývá implementací ukázkové asynchronní PHP aplikace, což je poslední částí práce. Nejprve je nastíněn obecný koncept aplikace a vysvětlen tzv. modul, ze kterých se aplikace skládá. Dále kapitola obsahuje popis grafického návrhu a ovládacích prvků. Z pohledu frontendu je popsán způsob komunikace s backendem a použité technologie. Při popisu backendu je více kladen důraz na využití asynchronního přístupu a jeho integraci do frameworku Laravel. Poslední kapitola 6 popisuje metodiku vývoje aplikace, konfiguraci prostředí aplikace přes Docker a mechanismus automatizovaného nasazení aplikace s využitím GitHub Actions.

2 Obecná teorie asynchronního programování

2.1 Synchronní zpracování

Základním modelem zpracování operací je synchronní průběh. Jde o nejjednodušší možný způsob. Operace se spouštějí postupně a dodržují předem definované pořadí. Každá další operace před jejím spuštěním čeká na dokončení té předchozí. Lze tedy očekávat, že při spuštění libovolné operace byly bez chyb dokončeny všechny operace předcházející a je možné přistupovat k jejich výsledkům. Toto chování způsobuje, že synchronní kód je zároveň deterministický.

Zpracovávané „operace“ jsou zatím jen obecný koncept, může se jednat o libovolná data (příchozí vstup od uživatele, později upřesněno na HTTP requesty). Na obrázku 2.1 jsou odlišnou barvou zobrazeny tři příchozí operace, které jsou synchronně zpracovávány. Důležitým faktem je, že když začne zpracovávání jedné z operací, zpracovává se jen ona až do dokončení.



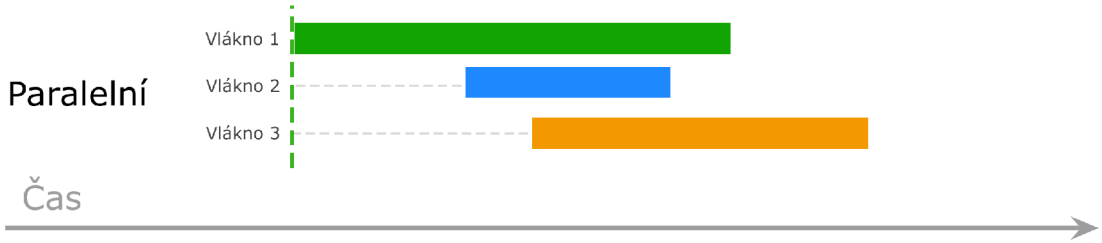
Obrázek 2.1: Synchronní model zpracování operací

2.2 Paralelní zpracování

Dalším modelem zpracování operací je paralelní přístup, což znamená, že více úloh běží současně v jednu chvíli. Spuštění probíhá v oddělených kontextech, paralelní zpracování tedy implikuje využití více vláken. Tím zároveň vzniká nutnost řešit synchronizaci dat mezi jednotlivými úlohami běžícími v totožný čas. Tento model lze nazvat jako preemptivní multitasking [1].

Na obrázku 2.2 jsou opět barevně odlišeny různé operace. Jelikož se v průběhu času zpracování operací překrývá, musí být v systému k dispozici více vláken. V tomto

zjednodušeném modelu je pro každou příchozí operaci vytvořeno nové vlákno (lze optimalizovat).



Obrázek 2.2: Paralelní model zpracování operací

2.2.1 Synchronizace mezi procesy

Pokud při paralelním zpracování přistupuje více běžících procesů ke stejnému místu v paměti, vzniká riziko, že nastane nějaký z jevů, které jsou popsány v této kapitole. **Souběh** (race condition) je situace, kdy kvůli nedeterministickému chování nezávisle běžících vláken dojde ke změně pořadí vykonávaných instrukcí a následně chybě programu. **Uváznutí** (deadlock) je stav, kdy jsou dva paralelně běžící procesy závislé navzájem na sobě. Každý z nich pro pokračování potřebuje dokončení operace v druhém procesu a ani jeden tedy nemůže pokračovat dál. Pro řešení těchto situací se používají různé programové konstrukce jako např. semafor nebo zámek (mutex).

2.3 Asynchronní zpracování

Prováděné operace v aplikaci lze rozdělit na takové, které využívají primárně výkon procesoru (CPU-bound) a takové, které jsou závislé na vstupně-výstupních operacích (IO-bound). Příkladem CPU-bound operace může být např. matematický výpočet a IO-bound operace čtení dat z disku. Pokud naše aplikace obsahuje vyšší množství právě IO-bound operací, lze ušetřit čas pomocí efektivnějšího střídání aktuálně vykonávané úlohy [2]. Když hlavní vlákno aplikace čeká na provedení časově náročné IO-bound operace (např. právě čtení z databáze/disku), může mezitím provádět jiné úkony, které čekají ve frontě. Až prováděný IO-bound úkon poskytne odpověď, procesor se k němu vrátí a získaná data dále zpracuje. Samotný asynchronní přístup tedy nijak neimplikuje použití více vláken, hovoří pouze o efektivním využívání procesorového času v rámci jednoho vlákna. Lze říci, že asynchronní zpracování značí, že probíhá více úkonů naráz, ale v jeden čas se pracuje pouze na jednom z nich. Alternativním názvem pro asynchronní model je kooperativní multitasking.

Obrázek 2.3 vykresluje takový model zpracování operací, který mezi operacemi různě v čase přepíná. Přepínání je řízeno na základě IO-bound operací (což závisí na definované logice operace). Primárním cílem modelu je neplýtvat čas procesoru.

Obrázek 2.3: Asynchronní model zpracování operací

2.4 Srovnání konceptů

Synchronní přístup představuje nejjednodušší možnost pro implementaci a nejmenší riziko vzniku dalších komplikací. Takto psaný kód je nejlépe čitelný ve smyslu identifikace pořadí prováděných operací. Jeho nevýhodou je možná neefektivita při velkém množství IO-bound operací a celkově omezenost v oblasti rychlosti, kde mu protiváhu naopak dělá paralelní přístup. Souběžné spuštění poskytuje teoreticky optimální možnost co se rychlosti zpracování týče, protože je pro každou příchozí operaci vytvořen samostatný kontext a její zpracovávání započne ihned. Za cenu ideálního výkonu ovšem vznikají další bezpečnostní rizika jako problém synchronizace dat mezi procesy. Naopak asynchronní přístup umožňuje optimalizovat výkonnostní problémy synchronního modelu a zároveň se vyhnout problémům způsobeným souběžností v čase. Aby byl naplno využit, zpracovávaná logika musí obsahovat alespoň nějaké IO-bound operace. Vyžaduje další režii v podobě softwarových konstrukcí pro řízení jeho běhu a také vyžaduje promyšlení celkové struktury aplikace - pokud je některá část prováděné logiky závislá na výsledcích předcházející části, definované callbacky se musí také odpovídajícím způsobem řetězit. Každý z uvedených konceptů má své pro a proti a hodí se pro rozdílné situace.

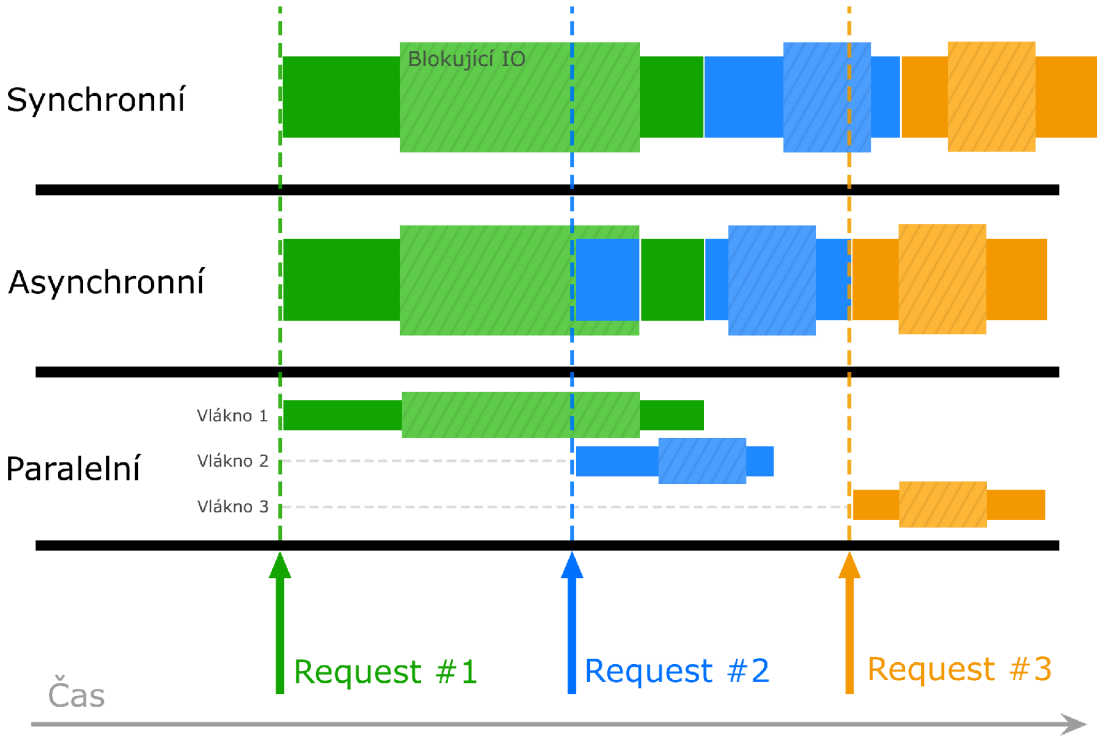
2.5 Zasazení do kontextu webu

Asynchronní přístup je natolik obecný koncept, že i ve webovém prostředí o něm lze hovořit ve více různých kontextech. Z pohledu webového frontendu se tento koncept využívá pro zachování interaktivity uživatelského rozhraní po odeslání requestu na backend. Pokud je request odeslán standardním způsobem (např. pomocí HTML formuláře), prohlížeč ho zpracovává synchronně. Celé uživatelské rozhraní zamrzne, prohlížeč čeká na odpověď od web serveru a až poté provede překreslení stránky. Naopak o asynchronním přístupu lze hovořit, pokud je request odeslán přes JavaScript s využitím technologie AJAX. Poté při zpracovávání requestu zůstává uživatelské rozhraní interaktivní a když aplikace obdrží response, pouze zavolá definovanou callback funkci, která dále pracuje s obdrženými daty.

Využívání konceptu asynchronicity v JavaScriptu popsaným způsobem je pro webové aplikace dnešní doby již v podstatě standard (v rámci knihoven jako React,

Vue, Svelte atd.) [3]. V této práci je proto popisováno využití asynchronního konceptu naopak z pohledu backendu (web serveru). Jde tedy o změnu způsobu, jakým web server zpracovává příchozí requesty. V případě synchronního přístupu probíhá zpracování requestů sekvenčně, tzn. jsou zpracovávány postupně a vždy pouze jeden naráz. Asynchronní přístup umožňuje přepínat mezi zpracovávanými requesty na základě IO-bound operací a díky tomu maximalizovat využití procesorového času. Naopak čistě paralelní přístup v kontextu přijímání requestů nedává smysl, serveru by při snaze vytvořit nové vlákno pro každý příchozí request velmi rychle došly systémové prostředky.

Na obrázku 2.4. jsou tyto tři přístupy (synchronní, paralelní, asynchronní) porovnány z pohledu web serveru, který zpracovává příchozí requesty v čase. Důležité je sledovat chování procesoru při blokujícím IO (šrafovaná oblast) a vztah příchozích requestů ke tvorbě nových systémových prostředků (vláken) u paralelního přístupu.

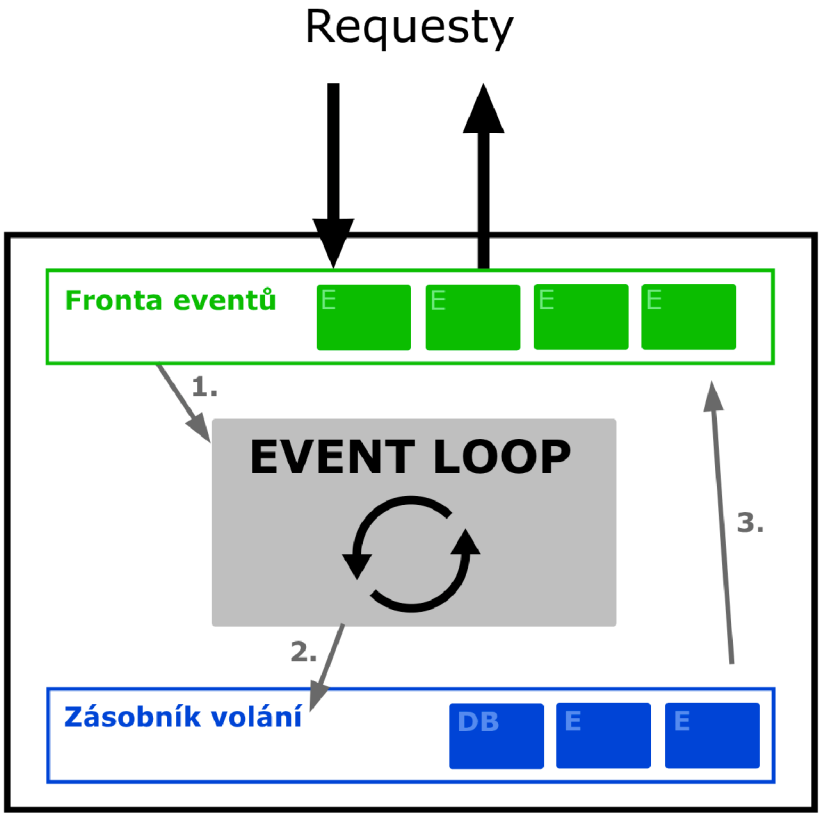


Obrázek 2.4: Srovnání třech modelů zpracování operací z pohledu webového serveru, který zpracovává requesty

2.5.1 Způsob implementace

Ve webovém kontextu se asynchronní přístup implementuje pomocí návrhového vzoru **Event Loop**. Ten poskytuje rozhraní pro řízení eventů. Jako eventy si lze představit jednotlivé příchozí HTTP requesty. Event Loop funguje tak, že si udržuje zásobník volání (call stack), který obsahuje aktuální stav vykonávaného programu

a frontu eventů (event queue), která obsahuje úlohy čekající na zpracování. Příchozí HTTP requesty se uloží do fronty eventů. Event Loop pracuje jako nekonečná smyčka, která v rámci každé iterace provádí následující úkony. Pokud je zásobník volný (prázdný), vezme první dostupný callback z fronty a vloží ho na vrchol zásobníku ke zpracování. Když je zpracovávaná úloha dokončena, její callback je vložen zpět do fronty eventů a proces se opakuje pro další čekající eventy [4]. Grafické znázornění popsaného principu se nachází na obrázku 2.5.



Obrázek 2.5: Diagram obecného fungování Event Loop

3 Asynchronní programování v PHP

3.1 Běžný PHP přístup

Jazyk PHP se standardně chová *synchronně* a *bezstavově*. Bezstavovost znamená v praxi, že se pro každý příchozí request spustí PHP kód v odděleném kontextu, který na úrovni aplikace nesdílí žádné zdroje s dalšími requesty. Pokud chce skript využít formu sdílené paměti s ostatními requesty, musí tak učinit buď pomocí systému lokální cache (jako je např. Memcached) nebo externího úložiště (libovolná databáze). Další vlastností klasického PHP skriptu je, že vnitřně funguje synchronně, tudíž při IO-bound operaci se chová jako blokující a procesor zůstává v nečinném módu než se daná operace dokončí (viz kapitola 2.1) [2].

3.1.1 Historie a vývoj architektury

Důvodem pro přítomnost synchronní architektury v jazyce PHP je jeho původní způsob implementace - byl vytvořen jako mezivrstva pro volání služeb v jazyce C. Z tohoto důvodu zdědil i přístup k řízení IO-bound operací, který je právě blokující.

Historicky se PHP skripty integrovaly do web serverů pomocí různých způsobů. Nejvíce užívané byly původně dva z nich, později přibyl třetí. První možností je využití protokolu CGI (Common Gateway Interface), který je nezávislý na konkrétním web serveru - ten ovšem způsobuje nutnost vytvořit nový proces pro každý příchozí request; pokud server poté obdrží mnoho requestů současně, může se potýkat s výkonnostními problémy. Druhou možností je rozšíření do Apache s názvem *mod_php*, které sice podporuje jistou formu „fronty requestů “ a jejich následné zpracování pomocí paralelních workerů (implementované jako další rozšíření v Apache). Jeho mechanismus je ovšem stále uzavřen uvnitř web serveru Apache (což přináší určitá omezení, jako např. nemožnost běhu více různých verzí PHP naráz) [5].

Třetí možností je vylepšený protokol FastCGI, který kromě zvýšení výkonu a stability poskytuje také možnost ovládat spouštění procesů izolovaně od web serveru. Tzn. obsluhu fronty requestů má na starost konkrétní skriptovací jazyk, nikoliv web server. **PHP-FPM** (FastCGI Process Manager) je implementací tohoto protokolu v PHP. Představuje doporučený způsob spouštění synchronních PHP skriptů v nynější době [6], typicky použit společně s web serverem Nginx, lze ho ovšem použít také s libovolným jiným web serverem.

3.1.2 Výhody a nevýhody

Hlavní výhodou bezstavové a synchronní PHP aplikace je její efektivní škálovatelnost. Všechny zdroje jsou uloženy externě (databáze, cache) a načítají se do aplikace při každém requestu znovu. Je tedy možné přidat další identické procesy a obsluhovat tak více requestů současně. Teoreticky lze takto aplikaci paralelizovat do nekonečna, právě díky bezstavové povaze každého requestu. Jediné omezení je právě externí úložiště, které může mít problém stíhat tolik současných připojení. Další výhodou tohoto přístupu je menší šance na vznik memory leaku. Jelikož si aplikace při příchozím requestu načte vše od začátku a po dokončení se všechny zdroje opět uvolní, programátor nemá tolik prostoru udělat chybu.

Primární nevýhodou je naopak výkon. Načítání všech externích zdrojů (např. připojení k databázi) do aplikace u každého requestu znovu je výpočetně drahé. Tento proces lze nazvat jako znovusestavování aplikačního kontextu. Pokud by tyto zdroje přetrvávaly načtené i mezi requesty, bylo by možné je využít přímo a ušetřit čas, kdy se musí načítat. Aplikace by tím ovšem ztrácela bezstavový charakter PHP.

3.1.3 Motivace

Pro využití plného potenciálu asynchronního přístupu musí logika aplikace obsahovat významné množství IO-bound operací. Díky mechanismu Event Loop lze několikanásobně zvýšit počet requestů, které dokáže server odbavit za jednotku času. Tento přístup je tedy primárně vhodný pro aplikace, které se potýkají s **velkým množstvím přichozích requestů naráz**. Alternativou by mohlo být využití jiného programovacího jazyka, který umožňuje asynchronní zpracování již v základu (např. Node.js). Pro nové projekty je toto definitivně jednou z možností, ovšem existující projekty (již napsané v PHP) je takto možné refaktorovat a díky využití asynchronního přístupu zlepšit jejich výkon. Kompletní změna backend jazyka může být u větších projektů příliš nákladná.

3.2 Možnosti implementace asynchronicity

V kapitole 3.1.1 byly popsány způsoby, jak integrovat PHP skripty do web serveru pro klasické synchronní fungování. To znamená, že je spuštěna nová instance aplikace pro každý přijatý request. Naopak pro vytvoření asynchronního serveru je nutné spustit PHP aplikaci v CLI módu. Aplikace tedy po spuštění zůstává běžet, naslouchá na určitém portu a web server (Apache / Nginx) v tomto modelu figuruje pouze jako reverse proxy. V rámci této kapitoly jsou popsané různé programové konstrukce, které můžou být na úrovni jazyka PHP použity pro implementaci asynchronního chování. Programátor se v praxi k těmto konstrukcím typicky vůbec nedostane, místo nich využívá abstrakce poskytnuté některou z knihoven.

3.2.1 Starší možnosti

Nejjednodušší z možností spuštění kódu neblokujícím způsobem je využití funkce `exec`. To je nativní PHP funkce, která umožňuje spustit externí program v novém procesu, který nesdílí s původní aplikací žádné zdroje. Tomuto procesu lze předat libovolný stav. Daný kus blokujícího kódu se tedy přemístí do vedlejšího PHP skriptu. Ten je poté pomocí funkce exec spuštěn a jeho výstup je přesměrován do `/dev/null`. Díky tomu hlavní vlákno aplikace nečeká na dokončení zpracování tohoto kódu a při zavolání funkce exec pokračuje ihned dál. Nevýhodou tohoto přístupu je nemožnost monitorování běhu spuštěného procesu přímo z aplikace (proces monitoringu závisí na procesu samotném).

Druhou možností je využití funkce `pcntl\_fork`. Pro přístup k této funkci je nutné PCNTL rozšíření do PHP. Tato funkce zduplikuje aktuální proces a vytvoří z něj novou strukturu, která se skládá z jednoho rodičovského procesu a dalších potomků (jejich počet je určen tím, kolikrát je funkce zavolaná). Podle detekce čísla procesu lze poté spouštět kód v různém aplikačním kontextu. Díky tomu má aplikace lepší kontrolu nad spouštěním procesů a dokáže monitorovat případnou vyvolanou chybu v některém z nich. Tento přístup lze navíc zkombinovat s metodou `exec` pro dosažení ještě optimálnějšího běhu programu [7].

3.2.2 Fibers

Před rokem 2021 byly dříve uvedené funkce jedinou možností pro dosažení asynchronního chování v jazyce PHP. V roce 2021 byla vydána PHP verze 8.1, která přidává funkcionalitu Fibers. Jedná se o full-stack přerušitelné funkce [8]. Full-stack znamená, že Fiber má ve vlastní paměti uložen jeho aktuální stav (started, suspended, running, terminated) a všechny proměnné, které využívá. Jeho stav lze libovolně ovládat (spustit, přerušit) z hlavní aplikace a i z vnitřku samotné funkce. Fibers tedy přidávají do PHP možnost jak využít koncept, který lze nazvat jako *kooperativní multitasking*. Generátory se v PHP sice nacházejí již od verze 5.5 a poskytují podobné chování (vrácení dat do místa volání a zanechání stavu). Nelze se pomocí nich ovšem vrátit do volaného bloku kódu, kde byl zavolán `yield` [9].

Ve vydaném RFC (Request For Comments) je upřesněn koncept, který byl vytvořením Fibers zamýšlen [10]. Vývojáři PHP popisují, že Fibers nejsou určeny pro přímé použití programátory při tvorbě aplikace. Naopak slouží pro tvůrce knihoven, aby mohli implementovat návrhový vzor event loop a tím poté zpřístupit ovládání asynchronního webového serveru. Z uvedených technologií jsou Fibers využívány v rámci Amphp a ReactPHP.

3.2.3 Implementace v jiném jazyce

Jelikož jazyk PHP do roku 2021 nepodporoval přímé konstrukce pro asynchronní zpracování, poslední možnou alternativou je využití jiného programovací jazyka pro implementaci mechanismu řízení requestů. Tento proces lze vysvětlit na příkladu použití programovacího jazyka Go. Jelikož obsahuje programovou konstrukci Go-

routines (konceptuálně podobné Fibers), s jejich využitím lze implementovat asynchronní HTTP server. Správu příchozích requestů tedy řeší Go. Logika aplikace je ovšem napsaná v PHP a Go s ním komunikuje využitím technologie RPC (Remote Procedure Call). Vývojáři tak zůstane možnost využívání jazykových konstrukcí PHP, ale na pozadí se používají právě např. Goroutines pro dosažení asynchronního chování.

3.3 Bezpečnost

Asynchronně implementovaná PHP aplikace mění tradiční PHP bezstavový model fungování za stavový model. Jelikož aplikace nyní sdílí svůj stav mezi více requesty, vznikají určité bezpečnostní implikace, které v bezstavovém modelu nemůžou nastat. Programátor tak za cenu lepšího výkonu (který bezstavový model nabízí) musí dávat pozor na práci s pamětí. Tento způsob vnitřního fungování aplikace lze přirovnat k tomu, jak vnitřně funguje např. Node.js.

3.3.1 Memory leak

Typickou bezpečnostní hrozbou je memory leak. Tento problém se projevuje obecně u **dlouho běžících aplikací** [11]. Jde o to, že si aplikace alokuje kus paměti, který poté až do jejího ukončení není uvolněn. V bezstavovém modelu tohle není možné, jelikož jsou PHP skripty spouštěny znovu při každém příchozím requestu a nezůstávají běžet v CLI režimu. Naopak ve stavovém režimu tento problém může nastat, protože aplikace funguje v rámci jednoho dlouho běžícího procesu.

Ukázkovou demonstrací tohoto problému by byl např. server, který při každém příchozím requestu přidá data do pole ve statické proměnné. Při lokálním vývoji tento problém nemusí být na první pohled vidět, ovšem čím déle aplikace poběží (po nasazení na produkci), tím více paměti může pole zabírat. Z tohoto důvodu je doporučeno opatřit dlouho běžící aplikace nástrojem na sledování spoteby paměti, tak lze případný memory leak detekovat. Některé dále uvedené technologie tomuto problému předcházejí tak, že běžící workery restartují jednou za čas (např. každých 500 odbavených requestů).

3.3.2 Synchronizace dat

I přes to, že v kontextu zpracování přijatých requestů lze říci, že aplikace pracuje asynchronně, uvedené technologie v rámci vnitřní implementace tohoto konceptu používají i různé konstrukce z paralelního programování. Tím by automaticky vznikaly určité problémy se synchronizací dat (popsány v kapitole 2.2.1). Programátor ovšem tyto problémy nemusí řešit, protože je za něj řeší použitá implementace pro sdílení dat. Jednou z možností je tohle vyřešit pomocí abstrakce, kterou poskytuje daná asynchronní technologie (např. Swoole Tables), další možností je využít např. systém lokální cache (Memcached, Redis).

4 Porovnání technologií

Tato kapitola popisuje proces porovnání asynchronních PHP technologií včetně vysvětlení stanovených kritérií porovnání, specifikace použitého prostředí pro benchmark a vyobrazení tabulek a grafů s výsledky. Nejprve je každá technologie stručně popsána a zasazena do kontextu s ostatními uvedenými technologiemi. Celý proces porovnání lze poté rozdělit na dvě hlavní části. Jeho první částí je porovnání vybraných PHP technologií na základě kritérií, které ukazují faktické rozdíly mezi nimi. Druhou částí je popis zkonstruovaného benchmarku, v rámci něhož byl v Docker prostředí měřen počet requestů, které dokáže daný server zpracovat za sekundu. Všechny konkrétní hodnoty v rámci porovnání (počet GitHub hvězd, commitů, kvalita dokumentace atd.) jsou aktuální ke dni 16. 4. 2024.

4.1 Výběr PHP technologií

Existuje mnoho technologií, které různým způsobem pracují s asynchronním přístupem. Aby porovnání dávalo smysl, bylo nutné zvolit konkrétní parametry, které musí každá vybraná technologie splňovat. Ty byly zvoleny takovým způsobem, aby se jednalo o technologie stejného typu s aktivním vývojem a komunitou. Jedná se o následující tři parametry:

1. technologie musí obsahovat asynchronní implementaci **HTTP serveru**
2. GitHub repozitář dané technologie musí mít minimálně 1000 hvězd
3. datum posledního commitu v repozitáři nesmí být starší než 1 rok

Na základě uvedených parametrů bylo vybráno následujících 8 technologií, které je splňují:

- Swoole
- OpenSwoole
- RoadRunner
- Amphp

- Workerman
- FrankenPHP
- ReactPHP
- Swow

4.1.1 Nezahrnuté technologie

Kromě vybraných technologií existuje v PHP ekosystému řada dalších, které do porovnání nebyly zahrnuty z důvodu porušení některého z parametrů. V následujícím seznamu je pro každou vyřazenou technologii uveden důvod jejího vyřazení a číslo parametru, který nesplňuje.

- Parametr č. 1 (asynchronní HTTP server)
  - **HprosePHP** - jedná se o middleware
  - **BrefPHP** - jde o serverless architekturu pro nasazení na AWS
  - **Revolt** - jedná se o samotnou Event Loop implementaci bez serveru (využívá ho ovšem zahrnutý Amphp)
  - **Guzzle** - jde o implementaci HTTP clienta
- Parametr č. 2 (minimálně 1000 GitHub hvězd)
  - **Pht Threading** - neaktivní repozitář, pouze 178 hvězd
  - **Moebius** - neaktivní repozitář, pouze 217 hvězd
- Parametr č. 3 (poslední commit starý méně než rok)
  - **PHPDaemon** - poslední commit v červnu 2022
  - **KrakenPHP** - poslední commit v červenci 2017
  - **Recoil** - poslední commit v březnu 2022
  - **Appserver** - poslední commit v říjnu 2021
  - **Icicle** - poslední commit v březnu 2017 (deprecated ve prospěch knihovny amphp/amp)
  - **Aerys** - poslední commit v srpnu 2020 (deprecated ve prospěch knihovny amphp/http-server)

Jedinou nesrovnalostí v rámci zahrnutých technologií je OpenSwoole, který nesplňuje 2. parametr počtu hvězd (jeho repozitář má pouze 790 hvězd). Jelikož se ovšem jedná o komunitní fork technologie Swoole (viz kapitola 4.2.1), repozitář OpenSwoole si zanechal veškerou historii vývoje. Z tohoto důvodu (a také protože je dále aktivně vyvíjen) byla pro OpenSwoole udělena výjimka a byl zahrnutý do porovnání.

4.2 Stručný popis technologií

V rámci této kapitoly jsou všechny zahrnuté technologie stručně představeny. Je kladen důraz na nastínění způsobu implementace dané technologie v PHP, popis konceptuálních odlišností od ostatních zmíněných technologií a případné doplnění o další relevantní informace. Rozdíly v konkrétních hodnotách měřených parametrů jsou vyhodnocovány v další kapitole.

4.2.1 Swoole a OpenSwoole

Swoole je asynchronní víceúčelový PHP framework. Kromě implementace HTTP serveru podporuje také ostatní síťové technologie (TCP, WebSockets atd.). Původně je napsán v jazyce C a do PHP je implementován ve formě rozšíření (extension). Lze ho nainstalovat z PECL (PHP Extension Community Library). Velký rozsah podporovaných protokolů a technologií z něj dělá univerzální nástroj pro tvorbu neblokujících PHP aplikací [12].

V říjnu roku 2021 byla vydána nová verze Swoole 4.7.1. Tato verze přidávala funkcionalitu, která stahuje soubor z privátně vlastněného serveru na základě napevno zapsané URL adresy ve zdrojovém kódu [13]. Jelikož se obsah souboru na serveru může kdykoliv změnit, toto lze považovat za bezpečnostní hrozbu. Ve vývojovém týmu Swoole toto vyvolalo neshody; nakonec se část vývojářů odpojila a vytvořila komunitní fork zvaný OpenSwoole [14]. Bezpečnostní hrozba byla nakonec odstraněna, fork ovšem zůstal zachován. Oproti Swoole nabízí kvalitněji zpracovanou dokumentaci v anglickém jazyce, ale menší tým správců a pomalejší vývoj. Jinak se konceptuálně jedná o velmi podobné technologie.

4.2.2 RoadRunner

RoadRunner je aplikační server, load-balancer a process manager. Je implementován v jazyce Go jako samostatný proces a s PHP komunikuje pomocí RPC (s využitím knihovny Goridge). Díky tomu ho lze rozšířit o další knihovny z jazyka Go. S využitím konstrukce Goroutines spouští jednotlivé PHP skripty jako tzv. workery [15].

4.2.3 Amphp

Amphp je ekosystém, který obsahuje kolekci mnoha knihoven pro implementaci event-driven přístupu v různých aspektech webu. Jednou z existujících knihoven je implementace pro HTTP server, další knihovny lze použít pro práci s technologiemi Redis, Postgres, MySQL, WebSockets a další. Konceptuálně je velmi podobný jako dále zmíněný ReactPHP. Dříve měl vlastní Event Loop implementaci, nyní využívá Revolt [16].

4.2.4 Workerman

Workerman je PHP aplikační kontejner. Poskytuje obecné nízkoúrovňové konstrukce pro práci s různými webovými službami (TCP, VPN, FTP, herní servery atd.) a samozřejmě také HTTP server. Je implementován přímo v PHP a vyžaduje mít nainstalovaná rozšíření POSIX a PCNTL. Oproti ostatním zmíněným technologiím se jedná o více generickou volbu, kterou lze použít jako stabilní základ asynchronně fungující webové služby s důrazem na výkon [17].

4.2.5 FrankenPHP

FrankenPHP je aplikační server pro PHP, který funguje jako nadstavba nad web serverem Caddy. Podobně jako RoadRunner je implementován v jazyce Go a také s PHP skripty pracuje v tzv. „worker“ módu. Může být také použit jako samostatný binární soubor pro integraci PHP kódu do Go aplikace, která využívá balíček `net/http`. Jeho spojení s web serverem Caddy způsobuje náročnější konfiguraci. Dále lze pomocí FrankenPHP zabalit PHP aplikaci společně s Caddy do jednotného binárního souboru pro snadnější distribuci [18].

4.2.6 ReactPHP

ReactPHP je PHP knihovna pro tvorbu aplikací s využitím event-driven přístupu. Hlavní součástí jejího jádra je vlastní implementace Event Loop podle návrhového vzoru reaktor. Ta využívá konstrukci Fibers. ReactPHP je tedy implementován přímo v jazyce PHP. Nad touto Event Loop jsou k dispozici další nízkoúrovňové nástroje jako HTTP server, DNS překladač nebo abstrakce pro koncept Stream. Instalace ReactPHP je dostupná přímo z balíčkovacího nástroje Composer. V ekosystému knihovny ReactPHP jsou dostupné stovky přídavných knihoven, které do něj lze doinstalovat [19].

4.2.7 Swow

Swow je multiplatformní engine, který podporuje souběžné (concurrent) IO. Je implementován v jazyce C (stejně jako Swoole) a komunikace s PHP probíhá pomocí RPC. Do PHP je implementován jako rozšíření. Jeho filozofie vývoje zahrnuje kombinaci nejmenšího možného jádra jazyka C s jazykem PHP pro vytvoření výkonného web serveru, který zvládne obsluhovat velké množství současných requestů. Nenachází se v repozitáři PECL, ovšem v balíčkovacím systému Composer je vytvořený nástroj, který umožňuje Swow automaticky zkompilovat a nainstalovat. Alternativní možností je provést kompilaci manuálně [20].

4.3 Kritéria porovnání

První část porovnání se zabývá zkoumáním faktických vlastností charakterizující vybrané technologie. Primárně byly vybírány taková kritéria, která lze exaktně změřit.

Pro neměřitelná kritéria (např. kvalita dokumentace) je uvedena konkrétní metodika, která rozlišuje mezi úrovněmi hodnocení. Dále bylo provedeno rozdělení kritérií do samostatných kategorií podle vypovídající hodnoty. V následujícím seznamu jsou uvedena všechna porovnávaná kritéria včetně jejich kategorie a v případě nutnosti také krátkého doplňujícího popisu.

- Kategorie **„komunita“**
  - **GitHub hvězdy**
  - **Forky repozitáře**
  - **Vytvořené issues** (otevřené i uzavřené)
  - **Vytvořené pull requesty** (otevřené i uzavřené)
- Kategorie **„technologie“**
  - **Způsob implementace:** jak je daná technologie implementována v kontextu k jazyku PHP
  - **Podporované protokoly:** seznam konkrétních síťových protokolů v prostředí webu, které technologie umožňuje využít
  - **Úlohy na pozadí:** existuje možnost spustit paralelní úlohy na pozadí (jobs)
- Kategorie **„ostatní“**
  - **Min. verze PHP:** minimální verze PHP, se kterou umí technologie pracovat
  - **Datum vzniku:** datum release nejstarší dostupné verze v GitHub repozitáři
  - **Licence:** druh licence, které software podléhá
  - **Kvalita dokumentace** (skóre 0 až 5; metodika definovaná níže)

V dalším seznamu jsou exaktně neměřitelná kritéria doplněny o metodiku, která definuje jejich výslednou hodnotu.

- Kritérium **„kvalita dokumentace“** (rozsah skóre 0 až 5)

  skóre 5: technologie má detailní dokumentaci s dostatkem praktických příkladů a popisem API

  skóre 4: technologie má průměrně obsáhlou dokumentaci s dostatkem praktických příkladů nebo popisem API

  skóre 3: technologie má průměrně obsáhlou dokumentaci s málo praktickými příklady

  skóre 2: technologie má málo obsáhlou dokumentaci v jednom jazyce

  skóre 1: technologie má pouze automaticky vygenerovanou dokumentaci (např. na základě komentářů u metod)

  skóre 0: technologie neobsahuje žádnou formu dokumentace

4.4 Srovnávací tabulky

Pro každou z vybraných kategorií byla vytvořena samostatná tabulka, která všem parametrům dané kategorie přiřazuje hodnotu ve vztahu ke každé z měřených technologií. Pokud to u daného parametru dává smysl, technologie s nejlepším výsledkem je zvýrazněna tučně.

4.4.1 Komunita

Tabulka 4.1: Výsledky porovnání pro kategorii „komunita“

Název	GitHub hvězdy	Forky	Issues	Pull requesty
Swoole	**18 204**	**3 159**	**3 358**	**1827**
OpenSwoole	790	48	156	188
RoadRunner	7 671	402	730	917
Amphp	4 124	248	240	191
Workerman	10 918	2 252	610	378
FrankenPHP	5 755	181	250	380
ReactPHP	8 811	722	212	262
Swow	1 117	106	115	96

4.4.2 Technologie

Tabulka 4.2: Výsledky porovnání pro kategorii „technologie“

Název	Implementace	Protokoly	ÚNP*
Swoole	rozšíření	HTTP, WebSocket, UnixSocket	Ano
OpenSwoole	rozšíření	HTTP, WebSocket, UnixSocket	Ano
RoadRunner	PM**	HTTP, WebSocket	Ano
Amphp	nativní	HTTP, WebSocket	Ne
Workerman	nativní	HTTP, WebSocket, SSL	Ne
FrankenPHP	web server	HTTP	Ne
ReactPHP	nativní	HTTP, DNS	Ne
Swow	rozšíření	HTTP, WebSocket	Ne

* Úlohy na pozadí
** Process manager

4.4.3 Ostatní

Tabulka 4.3: Výsledky porovnání pro kategorii „ostatní“

Název	Verze PHP*	Datum vzniku	Licence	Dokumentace**
Swoole	7.2+	**14. 4. 2014**	Apache-2.0	3
OpenSwoole	7.2+	17. 10. 2021	Apache-2.0	**5**
RoadRunner	7.4+	28. 1. 2018	MIT	**5**
Amphp	8.1+	22. 7. 2014	MIT	4
Workerman	5.4+	31. 1. 2015	MIT	3
FrankenPHP	8.2+	20. 9. 2023	MIT	4
ReactPHP	**5.3+**	5. 7. 2018	MIT	**5**
Swow	8.0+	8. 3. 2021	Apache-2.0	2

* Minimální podporovaná verze PHP
** Škála 0 až 5 (kde 5 je nejlepší)

4.5 Návrh benchmarku

Druhou částí porovnání technologií bylo vytvoření benchmarku pro měření výkonu serveru. Jako měřený parametr byl zvolen počet requestů, které dokáže server odbavit během jedné sekundy (RPS). Benchmark byl navržen tak, aby od sebe byly jednotlivé technologie oddělené v rámci prostředí. Pro ovládání benchmarku byl vytvořen jednoduchý CLI skript v jazyce Python, který poskytuje uživateli rozhraní pro spuštění benchmarku a generování grafů z naměřených výsledků. Dále je uživateli pro správu proměnných hodnot k dispozici konfigurační soubor a možnost specifikovat parametry v rámci volání CLI skriptu. Prostředí benchmarku bylo realizováno s využitím technologií Docker a Docker Compose.

4.6 Konkrétní popis benchmark prostředí

Všechny měřené technologie jsou v rámci Docker Compose prostředí rozděleny do samostatných kontejnerů. Každý takový kontejner spouští web server, který poslouchá na přiděleném portu. Skrze vytvořenou síť v Docker Compose jsou všechny kontejnery propojeny. Dále prostředí obsahuje jeden kontejner s názvem `benchmark\_master`. Právě v něm se nachází Python skripty pro ovládání celého benchmarku, adresář s naměřenými výsledky ve formátu JSON a adresář s vygenerovanými grafy. Prostředí je navrženo takovým způsobem, aby bylo benchmark možné rozšířit o další měřené technologie pouze pomocí vytvoření kontejneru a upravení konfiguračního souboru. Samotné měření probíhá s využitím CLI nástroje *Bombardier*, který poskytuje prostředí pro benchmark HTTP serverů. Nástroj je napsán v jazyce Go a v rámci Docker prostředí je zpřístupněn jako samostatný binární soubor.

Uvedené výsledky měření byly pořízeny na PC s následujícími hardwarovými specifikacemi. Na odlišném hardware se může výsledek lišit.

- Procesor: **AMD Ryzen 5 3600 (6 jader, frekvence 3600 MHz)**
- Grafická karta: **NVIDIA GeForce GTX 1660**
- Operační paměť: **16 GB, frekvence 1200 MHz**
- Operační systém: **Windows 10**

4.6.1 Metodika měření

Vnitřní logika serveru je totožná u všech měřených technologií. Server při obdrženém requestu vrací JSON response s aktuálním časem ve formátu UNIX timestamp, verzí programovacího jazyka a názvem měřené technologie. Serverová logika tedy neobsahuje žádné IO-bound operace. Tak je učiněno z důvodu minimalizace externích vlivů měření, doba trvání IO operace by se mohla lišit. Pro zajištění stejných podmínek ve využívání systémových prostředků je skrze Docker Compose aplikován každému kontejneru limit maximálního množství využívané RAM a procesoru. Technologie byly měřeny v jejich základní konfiguraci.

V konfiguračním souboru lze definovat parametry měření. Každý takový parametr obsahuje celkový počet requestů, které budou odeslány na server a počet souběžných requestů (concurrency). Pokud je concurrency např. 20, znamená to, že je na server odesláno 20 requestů v jeden okamžik. Po obdržení 20 odpovědí je odesláno dalších 20 requestů a takto benchmark pokračuje až do naplnění celkového počtu. V základní konfiguraci je definováno celkem 5 parametrů (tzn. výsledkem je 5 grafů). Všechny definované parametry obsahují stejný počet celkem 20000 requestů, concurrency se pro každý parametr liší, dosahuje hodnot 1, 10, 200, 500 a 1000.

4.6.2 Python skripty

Vytvořené Python skripty poskytují nástroj pro ovládání benchmarku a následnou interpretaci výsledků. Fungují na bázi klasických CLI aplikací s parametry, čehož je docíleno pomocí Python knihovny *Typer*. První skript slouží pro spuštění benchmarku a lze u něj specifikovat seznam technologií, které budou zahrnuty do měření. Po spuštění si skript načte všechny potřebné hodnoty z konfiguračního souboru. Poté skrze Python knihovnu `subprocess` postupně volá CLI příkaz benchmark nástroje *Bombardier* pro každý definovaný parametr měření v konfiguračním souboru. Výsledky jsou průběžně ukládány JSON souborů ve složce `outputs`.

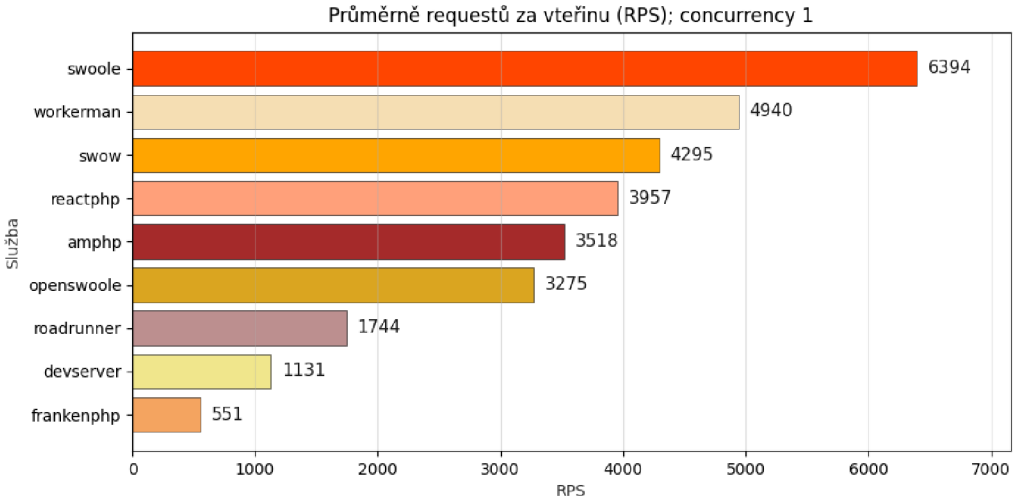
Druhý Python skript slouží pro vizualizaci naměřených hodnot. Při jeho zavolání lze opět specifikovat seznam technologií, dále existuje možnost nastavit barevené schéma grafů a parametr pro smazaní starých vygenerovaných grafů. Na základě zadaných parametrů se z naměřených hodnot nejprve sestaví tabulka, která je vypsána do konzole. Pro to se využívá Python knihovna *Tabulate*. Dále jsou z naměřených hodnot vytvořeny pomocí knihovny *Matplotlib* grafy, které jsou uloženy do složky `graphs`. Pro každý specifikovaný parametr měření je vytvořen samostatný graf obsahující všechny zahrnuté technologie. Obrázek č. 4.1 ukazuje tabulku s nápovědou pro tento skript.

```
> py visualise.py --help

Usage: visualise.py [OPTIONS] [CLEAR_GRAPHS]

Analyze measured benchmark results, generate images of graphs and visualise data to a table in
CLI.

 Arguments
   clear_graphs      [CLEAR_GRAPHS]  If true, all the old generated graphs in graphs/ folder
                                     will be removed.
                                     [default: False]

 Options
 --service           [openswoole|python|nodejs|reactphp  List of service names to be
                     |golang|swoole|roadrunner|amphp|fr  included in the graphs. If no
                     ankenphp|swow|workerman]            services are specified, graphs are
                                                         generated from  all possible
                                                         services in the output folder.
 --color-scheme      [color_warm|color_cold]             Name of color scheme to be used for
                                                         bar graph generation.
                                                         [default: color_warm]
 --help                                                  Show this message and exit.
```

Obrázek 4.1: Tabulka s nápovědou pro příkaz generující grafy

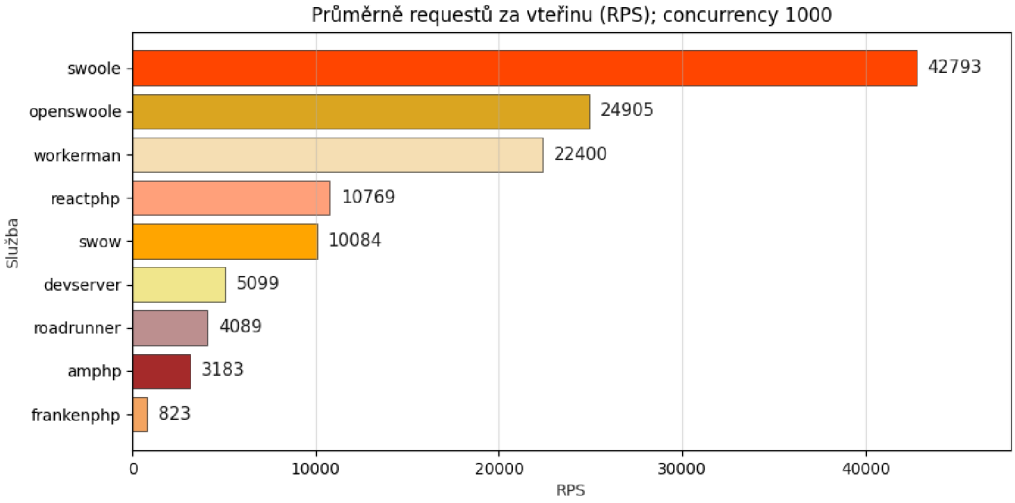
4.7 Výsledek PHP benchmarku

I přes to, že výsledkem benchmarku je celkem 5 grafů, pro demonstrační účely jsou uvedeny pouze dva z nich - graf s nejnižší a nejvyšší hodnotou concurrency. Na obrázku č. 4.2 lze vidět výsledek benchmarku pro PHP technologie s parametrem concurrency 1. Výsledné RPS se nyní pohybuje v jednotkách tisíců, jelikož výhody asynchronicity se více projeví až při vyšším množství současných requestů.



Obrázek 4.2: Výsledek PHP benchmarku pro concurrency 1

Obrázek č. 4.3 ukazuje graf s výsledkem benchmarku pro concurrency 1000. Swoole zde získal výrazně lepší výsledek než ostatní technologie, následován OpenSwoole na druhém místě. Oproti prvnímu grafu lze vidět, že při vyšší hodnotě concurrency stíhají některé z technologií vyřídit až desítky tisíc requestů během jedné sekundy.



Obrázek 4.3: Výsledek PHP benchmarku pro concurrency 1000

4.8 Doplnění o další jazyky

Dalším krokem bylo rozšíření porovnání o ostatní programovací jazyky, které umožňují asynchronní implementaci HTTP serveru. Za jazyk PHP byl vybrán zástupce na základě prvního provedeného benchmarku. S ohledem na provedené porovnání byl jako vítěz zvolen Swoole. Ostatní programovací jazyky jsou zastoupeny technologiemi umožňující nejrychlejší možnou implementaci asynchronního HTTP serveru na základě jiných, externě provedených benchmarků. V této kapitole je popsán proces výběru těchto technologií a krátké uvedení ostatních jazyků do kontextu s PHP. Porovnání s dalšími jazyky je provedeno pouze na základě výkonu technologie (benchmark), tabulkové porovnání kritérií není provedeno z důvodu příliš velkých rozdílů vnitřní implementace asynchronního přístupu mezi jazyky.

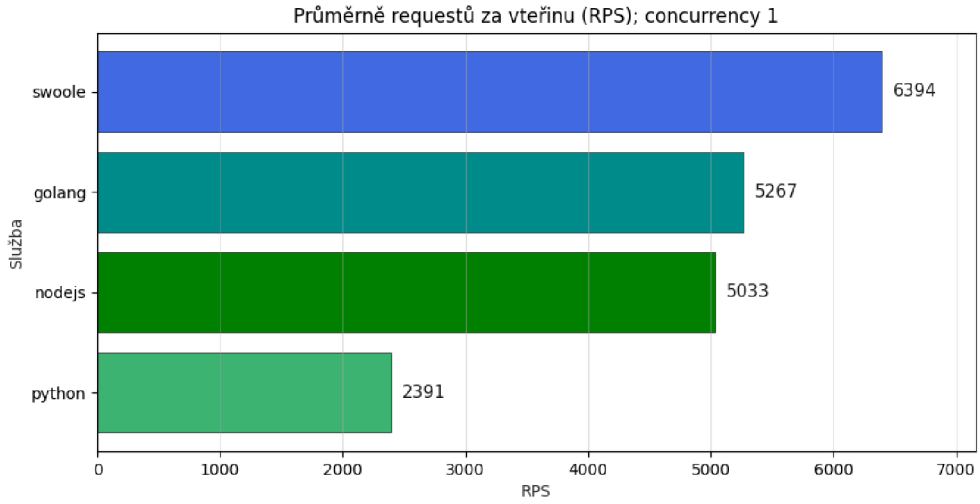
V rámci jazyka **Python** byla jako zástupce vybrána knihovna *Blacksheep*. Na základě externího benchmarku vychází jako nejrychlejší možnost [21]. Funguje na základě standardu pro webové aplikace ASGI (což je asynchronně fungující nástupce WSGI se zpětnou kompatibilitou). Do Pythonu lze tuto knihovnu přidat pomocí standardního správce balíčku *pip*.

Pro jazyk **JavaScript** bylo využito běhové prostředí Node.js s knihovnou *Hyper Express*. Jedná se o nejrychlejší možný HTTP server v rámci JavaScriptu dle dalšího externího benchmarku [22]. Knihovna poskytuje omezenou zpětnou kompatibilitu s API Express.js, ovšem je výrazně rychlejší.

V jazyce **Go** je využita knihovna *fasthttp*. Její dokumentace uvádí až desetinásobný nárůst v rychlosti při porovnání s knihovnou *net/http*, která je obsažena ve standardní instalaci jazyka Go. Stejně jako u ostatních jazyků, rychlost této knihovny je potvrzena externím benchmarkem [23].

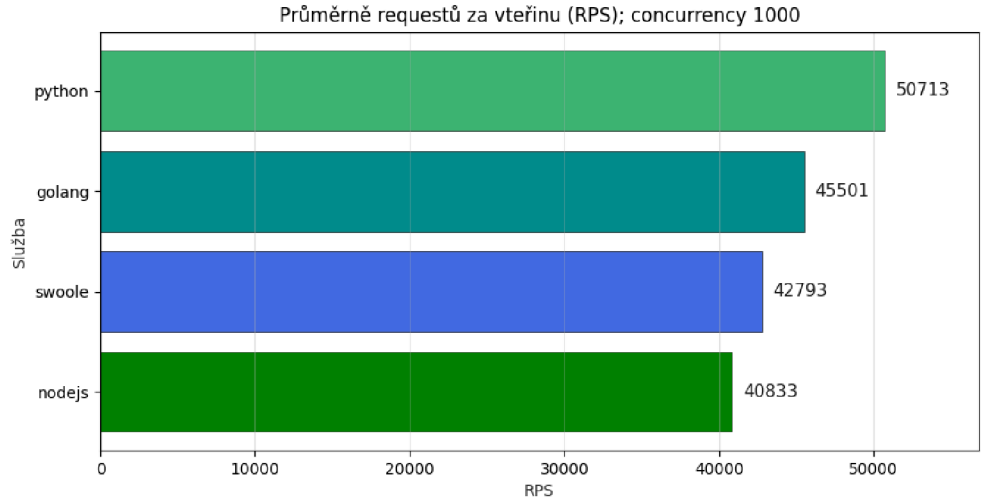
4.9 Výsledek benchmarku mezi dalšími jazyky

První měření mezi dalšími programovacími jazyky bylo provedeno opět s parametrem concurrency 1. Výsledek se nachází na obrázku 4.4 a podobně jako u prvního provedeného benchmarku se rychlost pohybuje v jednotkách tisíců requestů za sekundu.



Obrázek 4.4: Výsledek benchmarku dalších jazyků pro concurrency 1

Na obrázku 4.5 lze vidět výsledek měření pro 1000 současných requestů. Oproti PHP benchmarku mají data naměřená v této kategorii nižší odchylku, výsledky pro jednotlivé jazyky jsou relativně podobné.



Obrázek 4.5: Výsledek benchmarku dalších jazyků pro concurrency 1000

4.10 Celkové zhodnocení porovnání

Na základě provedeného benchmarku vychází Swoole ze všech testovaných asynchronních PHP technologií jako nejrychlejší možnost. Jedná se ovšem o PHP rozšíření, což může zkomplikovat instalaci. Technologie OpenSwoole je Swoole velice podobná, na základě benchmarku je pomalejší, ovšem obsahuje rozsáhlejší anglickou dokumentaci. FrankenPHP měl naopak v oblasti výkonu nejhorší výsledek ve všech kategoriích měření. Jelikož se jedná o technologii starou pouze několik měsíců, toto řešení ještě není dostatečně stabilní. Pro obecně zaměřenou aplikaci lze tedy jako vítěze porovnání označit **Swoole**. Nabízí vysokou stabilitu platformy, aktivní vývojový tým i komunitu, dostačující dokumentaci a nejlepší výkon. Ostatní porovnávané technologie jako ReactPHP, Amphp nebo RoadRunner nabízejí různé další specifické výhody (jako rozsáhlý ekosystém Amphp nebo jednoduchost instalace RoadRunneru) a můžou se hodit pro odlišné, více konkrétní use-case.

Benchmark by šel dále vylepšit pomocí sestavení vlastního Docker Image obsahujícího zkompilované PHP, ze kterého poté image jednotlivých technologií vycházejí jako základ (momentálně vycházejí z univerzálního DockerHub image *php-cli*). Další možností pro vylepšení benchmarku by bylo přidání IO-bound operací do definované logiky serveru. V momentálním stavu benchmark nijak nereflektuje výhody asynchronního přístupu v řízení IO-bound operací, oproti synchronnímu přístupu benchmark vyznačuje pouze rychlejší odpoveď serveru z důvodu spuštění aplikace ve stavovém módu, proto není nutno sestavovat kontext aplikace znovu při každém příchozím requestu.

5 Implementace ukázkové aplikace

5.1 Popis aplikace

Pro demonstraci využití asynchronního přístupu v praxi byla vytvořena aplikace s názvem **BitcoinSpotlight**. Funguje jako pomyslná „cache“ vrstva nad daty z Bitcoin API. Ve formě webové aplikace poskytuje uživateli nástroj pro zobrazení různých informací z Bitcoin ekosystému. Na straně frontendu obsahuje aplikace kompaktní uživatelské rozhraní se základními ovládacími prvky, backendová logika je implementována ve frameworku Laravel s využitím technologie *Laravel Octane*, která propojuje Laravel a asynchronní web server. Aplikace slouží pouze pro čtení a neobsahuje registraci.

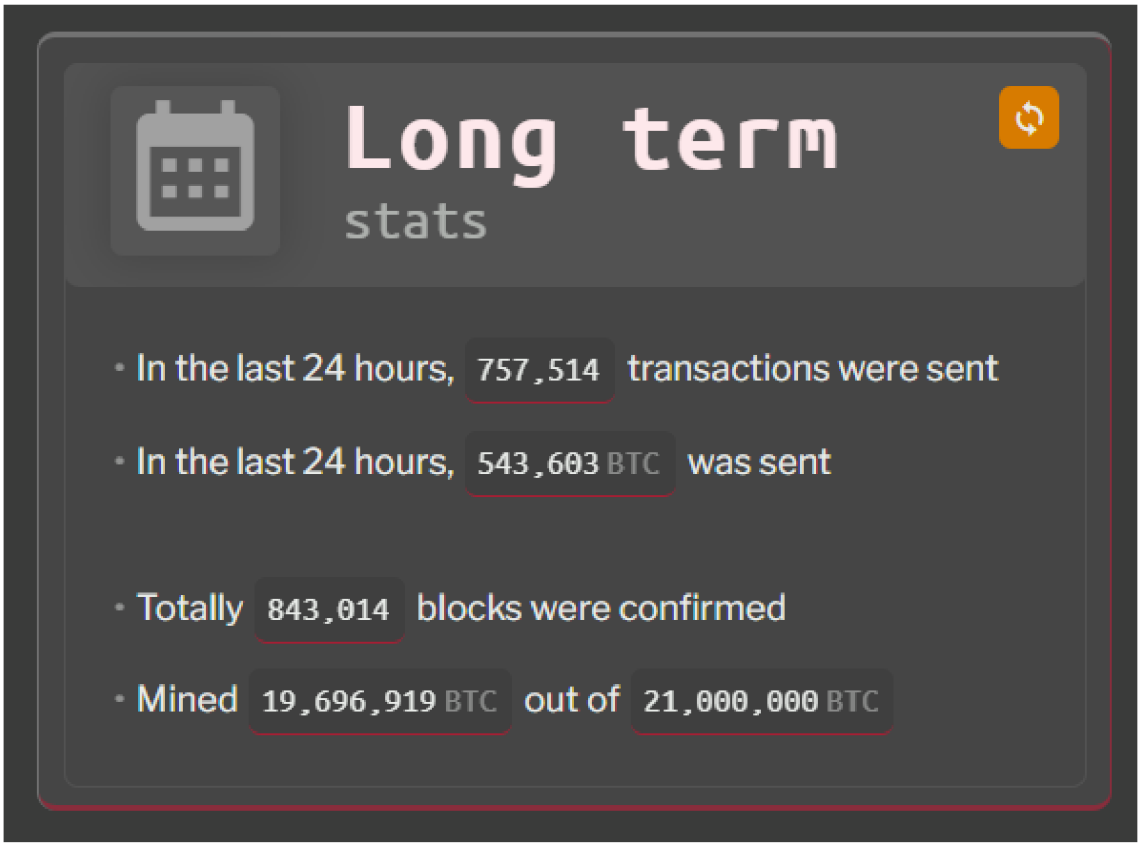
Aby aplikace vhodně demonstrovala výhody asynchronního přístupu, bylo nutné v návrhu logiky backendu využít nějaké IO-bound operace. Tento požadavek je v aplikaci zastoupen pravidelným voláním různých Bitcoin API. Aplikace dále výsledky volání API vnitřně zpracovává, ukládá do paměti a poté je poskytuje uživatelům ve formě HTML stránek.

5.2 Abstrakce konceptu

Aplikace se dělí na tzv. **moduly**. Obsahuje jich šest a každý z nich je charakterizován primárně typem dat, která zobrazuje. Podle toho je určen formát těla modulu, které může být buď bodový seznam hodnot, zvýrazněná hodnota nebo graf. Kromě těla obsahuje každý modul také následující prvky:

- název
- ikona
- nepovinný popisek (udává doplňující informace k datům)
- charakteristická barva
- způsob obnovy dat

Jedná se tedy o natolik obecný koncept, že aplikace může být rozšířena o další moduly pro zobrazení jiných dat, včetně možnosti znovuvyužití naprogramované logiky z již existujících modulů. Příklad jednoho z modulů lze vidět na obrázku 5.1.



Obrázek 5.1: Příklad modulu pro zobrazení dlouhodobých statistik

5.2.1 Detailnější popis

Pokud je způsob obnovy dat modulu nastaven na manuální režim, aktualizace dat je provedena v reakci na stisknutí tlačítka. V opačném případě modul obsahuje interní časovač, který data aktualizuje automaticky každých n sekund. Každý odeslaný požadavek na server se váže právě k jednomu konkrétnímu modulu. Obdržená data modul dále využívá pouze sám v rámci sebe a nijak nekomunikuje s ostatními moduly. Z pohledu serveru tedy modul figuruje jako samostatný prvek, který zasílá HTTP requesty pouze z důvodu aktualizace svých zobrazovaných informací.

5.3 Konkrétní moduly

Prvním a jedním z nejvýznamnějších modulů je modul žluté barvy, který informuje o ceně Bitcoinu. Cena je uživateli zobrazena v dolarech se zaokrouhlením na dvě desetinná místa. Zdrojem této informace je poslední zaznamenaná cena na burze. Ceny Bitcoinu na jednotlivých burzách nejsou vždy totožné; tzv. arbitrážní obchodníci ovšem sledují cenové rozdíly mezi burzami a nakupují příliš levné BTC [24]. To zajišťuje, že se ceny neustále přibližují k tomu, aby byly velmi podobné napříč všemi burzami. Pro zvýšení přesnosti zobrazované částky tedy aplikace cenu průměruje ze dvou různých API, které cenu vypočítávají z odlišných burz. Jelikož má cena

tendenci se měnit velmi často, modul svá data aktualizuje automaticky, každých 10 sekund.

Druhým implementovaným modulem je světle modrý graf historie ceny. Aplikace při každém čtení ceny tuto cenu také zapíše do své paměti. Pro ilustrační účely aplikace je stanoven limit počtu uložených cen na 30 minut do minulosti. Z těchto dat v paměti aplikace je poté sestavován spojnicový graf. Existuje také možnost základní manipulace s grafem (přiblížení, posun, export do obrázku). Z tohoto důvodu má modul nastavenou aktualizaci dat ručně; v opačném případě by hrozilo, že se automatická aktualizace provede ve chvíli, kdy uživatel manipuluje s grafem. Následné překreslení grafu by uvedlo graf do základní pozice a uživateli resetovalo např. nastavené přiblížení, což vede k horší UX.

Třetí modul v aplikaci obsahuje obecné informace o Blockchainu a je fialové barvy. Blockchain je datová struktura v návrhu Bitcoinu, konkrétně „append-only“ decentralizovaná a distribuovaná databáze, která řeší společně s mechanismem proof-of-work problém dvojí útraty [25]. Tato databáze a její uživatelé představují určitou síť, u níž lze měřit různé parametry. Seznam parametrů, který aplikace o Blockchainu uchovává je uveden v následujícím seznamu. Jelikož se v praxi tyto hodnoty mění s každým vytěženým blokem (což se děje průměrně jednou za 10min), modul má nastavenou automatickou aktualizaci dat každou minutu.

- hashrate (TH/s) - reprezentuje výpočetní výkon celé sítě (míru jejího zabezpeční) jako počet provedených hashů za sekundu
- průměrná doba mezi bloky (s) - aktuální průměrná čekací doba mezi vytěženými bloky, protokol Bitcoinu ji průběžně přibližuje hodnotě 600 sekund
- procentuální šance na nalezení bloku každým pokusem vypočtení hashe
- průměrný počet vypočtených hashů než je nalezen blok
- počet nepotvrzených čekajících transakcí

Čtvrtým modulem je tmavě modrý souhrn informací z posledního vytěženého bloku. Blok je vytěžen tím způsobem, že těžař uhádne takovou nonci, na kterou když je společně s daty bloku aplikována hashovací funkce, výsledek je dostatečně malé číslo (což je ovlivněno obtížností, kterou protokol Bitcoinu neustále mění). Tím se blok stává platným a všechny transakce, které jsou v něm obsaženy jsou považovány za potvrzené. Stejně jako u minulého modulu se toto děje průměrně jednou za 10min, modul si tedy svá data aktualizuje každou minutu automaticky. Zde je seznam informací, které jsou o posledním vytěženém bloku uchovávány:

- hash bloku (podle aktuální obtížnosti obsahuje různý počet nul na začátku)
- výška bloku - pořadové číslo bloku; počet bloků, které mu předcházejí
- počet potvrzených transakcí v rámci bloku
- datum a čas vytěžení

Pátým modulem v rámci aplikace je opět graf, tentokrát se jedná o rozdělení velikostí těžařských poolů s charakteristickou zelenou barvou. Jelikož výpočetní výkon jednotlivce není v rámci celé sítě dostačující, účastníci se sdružují do tzv. „těžařských poolů“, v rámci kterých sdílejí výpočetní výkon (lze přirovnat k výpočetnímu clusteru). Pokud by některý z poolů byl výrazně větší než všechny ostatní, mohlo by dojít k tzv. 51% útoku a bezpečnost sítě by byla ohrožena. Sloupcový graf tedy ilustruje velikost každého poolu pomocí zobrazení počtu potvrzených bloků během posledních 24 hodin. Modul využívá ruční možnost aktualizace dat ze stejných důvodů jako graf v druhém modulu.

Šestý a poslední modul zobrazuje dlouhodobé statistiky ekosystému Bitcoinu plus ostatní informace a má červenou barvu. Tyto statistiky se nemění často, proto aktualizace dat modulu probíhá manuálně. Obsahuje následující hodnoty:

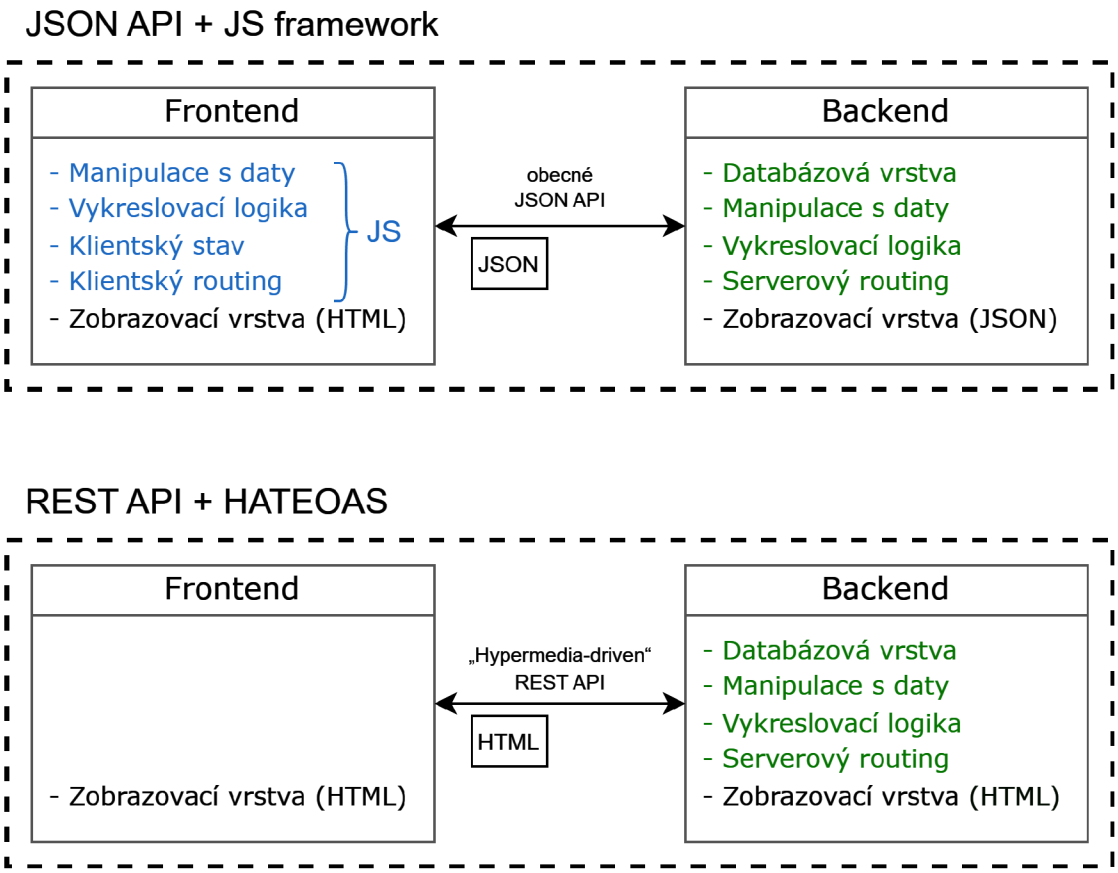
- počet transakcí za posledních 24 hodin
- počet BTC odeslaných za posledních 24 hodin
- celkový počet potvrzených bloků (nejdelší řetěz)
- celkový počet vytěžených BTC

5.4 Architektura frontendu

Častou volbou pro strukturu webové aplikace je v dnešní době využití některého z JavaScript frameworků, např. React, Vue nebo Angular pro vykreslení uživatelského rozhraní na frontendu [26]. Backend poté s frontendem komunikuje typicky přes JSON API. Tento přístup je vhodný pro aplikace, které mají velké množství funkcionality na straně klienta, ovšem přináší jednu zásadní nevýhodu - je nutné udržovat stav na dvou místech zároveň. Další nevýhodou je, že využití JavaScript frameworku v aplikaci přidává krok sestavení (build-step) a díky tomu i závislost na balíčkovací systém (např. npm). Pro menší aplikace může být tento přístup až zbytečně moc složitý.

Alternativou (která byla taktéž použita v ukázkové aplikaci) je využití principu *REST* (Representational State Transfer) v původním významu tohoto termínu, který definoval Roy Fielding ve své disertační práci z roku 2000 [27]. Aby bylo API označeno jako RESTful, musí být tzv. „hypermedia-driven“. To označuje způsob komunikace, kdy získaná odpověď z API obsahuje odkazy na další zdroje a sama je zdrojem - díky tomu je kompletně samopopisná. Klientovi pro zpracování odpovědi stačí pouze umět vykreslit dané hypermedium (v tomto případě HTML) [28]. Tento přístup lze také nazvat *HATEOAS* (Hypermedia as the Engine of Application State). Naopak při využití JSON API dostaneme pouze samotná data, která musí naše aplikace dále zpracovat a vykreslit. Tím pádem vznikají dvě samostatné aplikace a stav se udržuje jak na frontendu, tak i backendu. Přes API poté probíhá komunikace z důvodu aktualizace stavu v případě, že byl změněn druhou stranou.

Na obrázku 5.2 lze vidět rozdíl mezi těmito dvěma přístupy. HATEOAS předává veškerou zodpovědnost na server - klient v tomto návrhu pouze zobrazuje hypermedia, která mu server poskytne (je tedy užíván koncept *server-side rendering*). Nevýhodou tohoto přístupu může být ztráta obecného JSON API, které je nutno nahradit za konkrétní hypermedia API. V případě, že naše aplikace potřebuje jako klienta např. webové rozhraní, mobilní aplikaci i desktop, bylo by nutné pro každé zařízení vytvářet samostatné API.



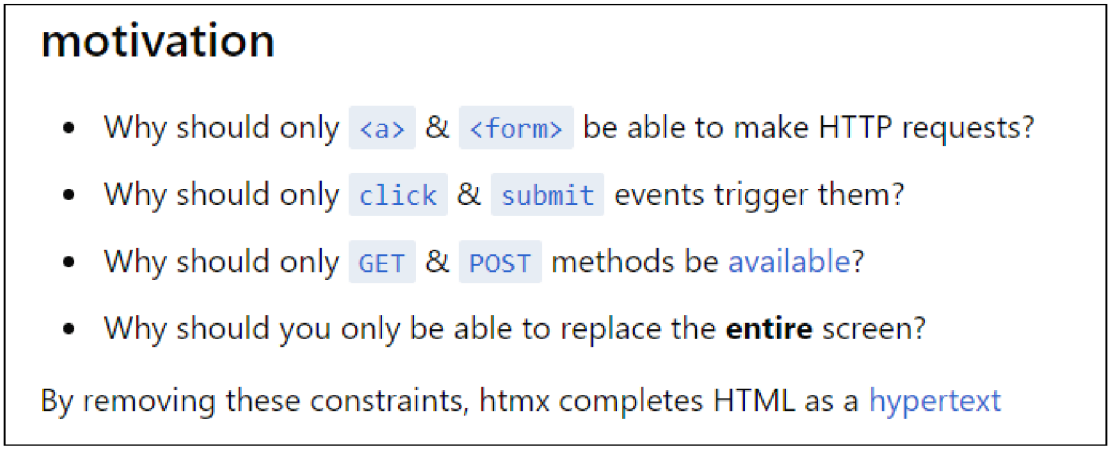
Obrázek 5.2: Diagram rozdílných přístupů webové architektury

5.4.1 HTMX

HTMX je JavaScript knihovna používaná pro tvorbu frontendu, který implementuje HATEOAS přístup. Pomocí atributů přímo v HTML umožňuje vyvolat AJAX request a získanou odpověď (ve formě hypermedia) poté vložit na libovolné místo do HTML DOM. Dále poskytuje podobnou abstrakci také pro WebSockets, Server-Sent Events a CSS přechody [29].

I přes to, že knihovna HTMX je vnitřně naprogramována v JavaScriptu, funguje jako rozšíření jazyka HTML. Při využívání HTMX není nutné psát žádný JavaScript kód. Pokud bychom chtěli vytvořit webovou aplikaci s HATEOAS přístupem pouze se samotným jazykem HTML, jsme omezeni nutností obnovit stránku při každém

requestu, můžeme využívat pouze GET a POST HTTP metody a máme omezené možnosti, jak samotný request vyvolat (viz obrázek 5.3). HTMX tyto limitace odstraňuje, díky čemuž lze pomocí něj dosáhnout vysoké interaktivity webu. HTMX lze tedy v kontextu komplexity vnímat jako kompromis mezi samotným HTML (které mnohdy není dostačující v uspokojení požadavků na interaktivitu frontendu) a JS frameworkem jako je React, Vue nebo Angular (které mohou být pro velké množství aplikací zbytečně komplikované nebo pomalé) [30].



Obrázek 5.3: Motivace vzniku knihovny HTMX (problémy, které řeší)

5.4.2 Hyperscript

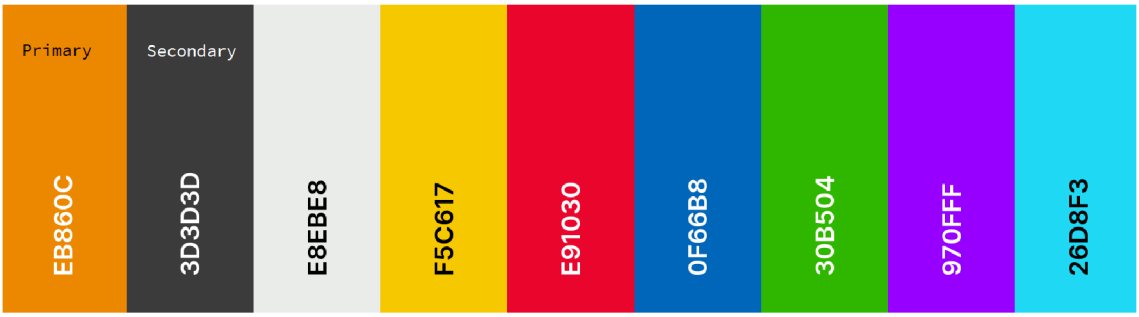
HTMX se stará o komunikaci se serverem a následné zpracovávání odpovědí. Ve webovém prostředí je ovšem často nutné implementovat funkcionalitu pouze na straně klienta, která nebude závislá na stavu aplikace. Příkladem takové funkcionality může být otevření dialogového okna nebo dočasné skrytí prvku na stránce. Pro tento účel je v aplikaci využita knihovna *Hyperscript*. Podobně jako např. známá knihovna *jQuery* slouží Hyperscript pro manipulaci s HTML DOM stromem. Hyperscript je stejně jako HTMX vnitřně implementován v JavaScriptu, ovšem z pohledu programátora figuruje jako samostatný skriptovací jazyk a pro jeho využívání není nutné psát JavaScript kód.

Základním stavebním blokem v Hyperscriptu jsou eventy, na které umí reagovat, případně je také vyvolat [31]. Díky tomu se dá velmi efektivně používat společně s HTMX (které vyvolává eventy při AJAX komunikaci). Jelikož oproti standardním JavaScript knihovnám (Vue, React) není reaktivní, pomocí eventů je nutno řídit veškerou komunikaci mezi elementy. Hyperscript definuje chování HTML elementu přímo v něm přes vlastní HTML atribut, což je další odlišení od standardních JavaScript knihoven, kde se kód píše do script tagu a daný element se spojí s definovanou logikou např. pomocí atributu „id“. Syntax Hyperscriptu se podobá anglickému jazyku a pro pochopení je velmi intuitivní. Patří do rodiny jazyků označovaných jako *xTalk*, které jsou odvozeny od jazyka *HyperTalk*. Knihovna volí tyto dva specifické přístupy z důvodu zvýšení čitelnosti kódu a zjednodušení údržby do budoucna.

V kontextu umístění kódu v rámci aplikace by se dalo tedy říci, že je potlačen princip *SoC* (Separation of Concerns) za cenu dosažení vyššího *LoB* (Locality of Behaviour) [32].

5.5 Grafický návrh

Při vytváření grafického návrhu aplikace byl kladen důraz na příjemně čitelné a kompaktní UI. Celkové barevné schéma je laděno do tmavých barev, aby byl text pohodlný pro čtení. V této kapitole jsou popsány některé grafické koncepty použité v aplikaci. Všechny využité barvy v rámci CSS jsou definovány jako proměnné a v kódu se používají pod přiřazenými názvy. Na obrázku 5.4 lze vidět paletu použitých barev. Dominantní (primární) barvou na stránce je oranžová a výchozí barva textu je bílá, stejně jako oficiální BTC logo. Vyobrazená paleta je zjednodušená, na stránce jsou kromě barev v paletě využity také jejich další odstíny (např. pro hover efekty).



Obrázek 5.4: Paleta barev použitých na stránce

Každý modul má svou unikátní charakteristickou barvu, která ho na stránce vizuálně odlišuje od ostatních. Zároveň pokud je obsahem modulu bodový seznam, jsou v něm obsaženy zvýrazněné hodnoty s jiným fontem a podtržením, které má právě charakteristickou barvu modulu. Dále je tato barva využita pro vytvoření vrženého stínu, který se nachází na pravé spodní straně modulu a imituje lehký 3D efekt. Pokud modul obsahuje graf, barvy křivek a sloupců grafu jsou také stejné barvy. V rámci celého webu jsou na psaný text využité celkem tři fonty - *Ubuntu* jako výchozí font pro text, *Ubuntu Mono* (neproporcionální písmo) pro zvýraznění hodnot a *Libre Franklin* pro text v patičce.

Pro vytvoření ikon se využívá CSS knihovna *Material Design Icons*. Každý blok má definovanou vlastní ikonu, která vyjadřuje význam daného modulu. Jedinou výjimkou je první modul s cenou, který místo ikony obsahuje rastrový obrázek s logem BTC. Každý modul má také definované různé animace textu pro plynulé zobrazení obsahu při načtení nových dat. První modul s cenou navíc obsahuje animovanou šipku nahoru nebo dolů (obarvenou zelené nebo červeně), která slouží jako indikátor, zda je cena oproti minulému měření vyšší nebo nižší. V hlavičce webu se nachází na míru vytvořené jednoduché logo aplikace (viz obrázek 5.5) s grafickým objektem šestiúhelníku.

Obrázek 5.5: Logo aplikace BitcoinSpotlight

5.6 Ovládací prvky

Návrh frontendu aplikace obsahuje prvky, u kterých bylo nutné implementovat určitou logiku. Tak bylo učiněno s využitím event-driven přístupu a knihovny *hyperscript*. Pro jednotlivé prvky byly definovány "vzorce chování" (v hypercscriptu klíčové slovo `behavior`), které je poté možno instalovat přímo do jednotlivých HTML elementů (v rámci hyperscriptem přidaného HTML atributu). Tyto vzorce chování umí také pracovat s argumenty a přistupovat ke globálním proměnným hyperscriptu.

Primárním ovládacím prvkem je nastavení režimu načítání dat, které obsahuje každý modul. Nachází se v pravém horním rohu modulu. První možností je manuální režim, tzn. uživatel musí stisknout tlačítko pro načtení dat. Druhou možností je automatické obnovení. Takový modul obsahuje časovač v sekundách, který okolo sebe má rámeček s kruhovou výsečí (sloužící jako vizualizace odpočtu). Vedle časovače se nachází tlačítko, kterým lze odpočet pozastavit / znovu spustit. Volitelnou možností na každém modulu je dále ovládací prvek s nápovědou. Ten má modrobílou barvu, nachází se v levém vrchním rohu modulu a zobrazuje krátký text při najetí myši přes něj (na mobilních zařízeních kliknutí). Tento text může např. popisovat význam dat uvnitř modulu, doplňovat grafy nebo případně vysvětlovat určité další pojmy.

Proměnné hodnoty napříč všemi moduly jsou uloženy v jednom objektu, v rámci hyperscriptové globální proměnné definované v HTML elementu, který obluje všechny moduly (wrapper). Tento objekt je nadále serializován (viz následující kapitola).

5.6.1 Persistence nastavení

Pro zlepšení UX je v rámci aplikaci implementováno ukládání dat na úrovni frontendu. Všechny proměnné hodnoty v rámci modulů (pauza automatického obnovování dat a počet sekund zbývající do obnovení) jsou uloženy v jednom společném objektu. Tento objekt existuje jako hyperscriptová globální proměnná a je definován v HTML elementu, který obklopuje všechny moduly (wrapper). Prostřednictvím hyperscriptu je také vytvořen event handler, který slouží k ukládání dat do localStorage a je

volán, pokud jsou změněny. Data se ukládají tak, že se objekt serializuje pomocî šifrovací funkce AES. Výsledná reprezentace objektu ve formě 256 bitů dlouhého řetězce se ukládá do localStorage. Při opětovném načtení stránky ze stejného zařîzení se hodnota z localStorage načte zpět do stránky a převede se zpět na původní objekt. Díky tomu zůstane aktuální frontendový stav totožný jako byl při uzavření stránky. Takto navržený mechanismus je vhodný pro rozšíření o další funkcionalitu. Do objektu lze přidat libovolné další proměnné (které se budou ukládat) a zašifrovaný objekt je možné kromě uložení do localStorage také zpřístupnit uživateli pro import a export. Díky tomu by bylo možné na libovolném zařízení obnovit konkrétní stav aplikace, který byl exportován na jiném zařízení.

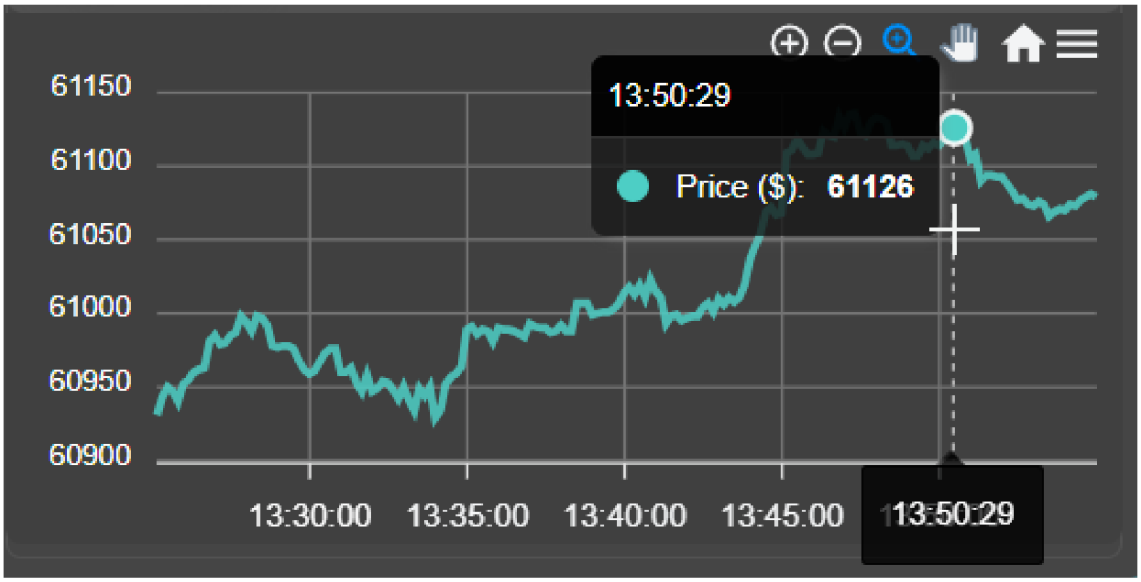
5.7 Grafy

Pro zobrazení grafů je využita JavaScript knihovna *Apexcharts*. Alternativní možností by bylo vygenerovat grafy na backendu a uživateli vrátit pouze obrázek. Touto cestou by se ovšem vytratily veškeré možnosti frontendové interaktivity. V JavaScriptu je tedy implementována funkce pro inicializaci grafu (včetně specifikace typu grafu, barev a dalšího nastavení), která jako parametr přijímá zobrazovaná data. Tato funkce je volána fragmentem kódu, který je obdržen z backendu jako odpověď na dotaz vyvolaný pomocí HTMX, samostatně pro každý modul.

Grafy vygenerované knihovnou Apexcharts poskytují uživateli možnost je interaktivně ovládat. Na stránce se vyskytují dva samostatné grafy. První z nich zobrazuje cenu BTC za posledních 30 minut ve formě spojnicového grafu. Perioda měření je 10 sekund, v grafu je tedy zobrazeno maximálně 180 vrcholů. Druhý graf zobrazuje, jak velké jsou jednotlivé těžařské pooly pomocí metriky počtu ověřených bloků za posledních 24 hodin. Jeho forma je sloupcový graf. Z důvodu přehlednosti je otočen o 90°. Interaktivita grafů spočívá v možnosti zobrazení detailu dat (hodnot vrcholů v prvním grafu, počtu ověřených bloků v druhém grafu) při najetí myší ve formě informativní bubliny s textem. Dále knihovna poskytuje dialogové menu, skrze které lze manipulovat s přiblížením grafu, posouvat aktuálně zobrazovanou oblast a exportovat graf jako obrázek na své zařízení.

5.7.1 Doplnění významu dat konkrétních grafů

V této kapitole je dále rozveden původ existence hodnot v jednotlivých grafech a jejich význam. První graf obsahuje průměrnou cenu za BTC v posledních 30min. Jelikož cena BTC závisí na poptávce a nabídce, lze ji definovat jako výše poslední provedené transakce na burze. Pro přesnější výsledek se cena vypočítává jako průměr hodnot z API poskytovaného službami *Blockchain.com* a *Coinbase*. Vizuální podobu grafu lze vidět na obrázku 5.6 včetně otevřené informativní bubliny, která zobrazuje cenu ve vybraném čase.



Obrázek 5.6: Graf vývoje ceny BTC za posledních 30min

Druhý graf obsahuje data o velikosti těžařských poolů. Těžaři jsou integrální součástí Bitcoinové sítě. Právě oni mají právo zapsat do blockchainu nové transakce a tím je potvrdit. Aby toto nemohl udělat kdokoliv, cena provedení zápisu nového bloku byla uměle zvýšena. Díky mechanismu hádání náhodných čísel s velmi malou pravděpodobností na úspěch a nutností při tom pálit mnoho elektrické energie (která je drahá) se dlouhodobě nevyplatí v síti podvádět [33]. Tento systém se nazývá proof-of-work a celá síť může být tak velká a robustní právě díky tomu, že těžařů je velký počet a jejich ekonomické incentivy jsou nastaveny tak, aby se jim vyplatilo poskytovat svůj výpočetní výkon na zabezpečení sítě. Jelikož výkon počítače samotného jednotlivce sám o sobě v síti není relevantní, začaly vznikat právě těžařské pooly, které slouží jako sdružení výkonu všech jeho účastníků dohromady. Pool poté každému účastníkovi sítě přidělí rozsah čísel, ve kterých hledá řešení a pokud se některému ze členů poolu podaří uhádnout výsledek a potvrdit blok, odměnu pool rozdělí mezi všechny členy, každému adekvátně podle toho, kolik energie spotřeboval. Díky grafu se lze ujistit, že existuje dostatek velkých poolů, které se navzájem kontrolují. Kdyby jeden pool ovládal více než 51 % hashrate celé sítě, mohlo by dojít k podvrhnutí celého proof-of-work mechanismu.

5.8 Architektura backendu

Backend ukázkové aplikace by bylo možné implementovat pouze s využitím samotného asynchronního web serveru. Tento přístup ovšem špatně škáluje a při snaze o rozšíření aplikace už by se mohlo více vyplatit použít nějaký framework. Aby se implementace co nejvíce blížila praktické situaci, jako backendový framework byl zvolen Laravel. Jedná se o nejčastěji využívaný PHP framework (statistiky udávají, že 43 % PHP webů s frameworkem má právě Laravel [34]). Vývojáři poskytuje různé funkcionality pro efektivní řešení rutinních úkolů (autentizace, routování, cache,

session, mail atd.). Celý framework operuje pod záštitou MVC architektury. Mimo standardní instalaci poskytuje Laravel také rozsáhlý ekosystém dalších nástrojů pro pokročilejší funkce (full-text vyhledávání, OAuth, WebSockets atd.) [35]. Tyto doplňující nástroje lze do aplikace integrovat skrze balíčkovací systém Composer.

V rámci implementace HATEOAS přístupu obsahuje aplikace samostatnou routu pro každý modul. Tato routa volá odpovídající metodu v controlleru, který poté pomocí helperů (ty slouží pro práci s daty) získá konkrétní hodnoty. Přes šablonovací systém Blade se následně získané hodnoty aplikují do HTML šablony a výsledek je vrácen zpět uživateli jako hypermedium.

5.8.1 Laravel Octane

Laravel v základní instalaci neposkytuje žádný způsob implementace asynchronního přístupu. Pro tento účel slouží rozšíření Laravel Octane, které v rámci aplikace funguje jako spojující vrstva mezi Laravelem a vybraným asynchronním web serverem. Programátorovi tedy zůstává přístup ke všem abstrakcím Laravelu, ovšem oproti standardnímu PHP cyklu je jeho aplikace spuštěna pouze jednou, zůstává v paměti a využívá jeden z asynchronních web serverů. Momentálně lze při instalaci Laravel Octane vybrat, zda bude použit Swoole, OpenSwoole, RoadRunner nebo FrankenPHP.

Na základě provedeného porovnání asynchronních technologií byl do aplikace vybrán web server Swoole. Ten vyžaduje přítomnost příslušeného PHP rozšíření, které lze nainstalovat např. pomocí PECL. Při využití Laravel Octane ve spojení se serverem Swoole jsou programátorovi k dispozici také nově přidané abstrakce (jejichž konkrétní využití je popsáno v následující kapitole).

5.9 Využití možností serveru Swoole

Swoole Tables zajišťují možnost sdílení dat mezi jednotlivými workery (které zpracovávají requesty) s vysokou rychlostí čtení. Data se ukládají do definovaných tabulek v RAM. V rámci ukázkové aplikace je vytvořena struktura celkem tří tabulek. Pro každý z grafů existuje jedna samostatná tabulka (pojmenovanány `prices` a `pools`). Do tabulky `prices` se zapisuje nová hodnota každých 10 sekund a tabulka má maximální kapacitu 180 záznamů, pokrývá tedy hodnoty naměřené v rámci posledních 30 minut. Každý záznam obsahuje naměřenou cenu a příslušný čas měření, který je reprezentován jako UNIX timestamp. Pro druhý graf je vytvořena tabulka `pools`. Zde se nachází pouze počet potvrzených bloků za posledních 24 hodin pro daný těžařský pool. Kapacita této tabulky je 30 záznamů. Jelikož relevantních těžařských poolů (minimálně jeden potvrzený blok za den) existuje momentálně cca 15, lze se domnívat, že jejich počet skokově nenaroste na více než dvojnásobek, tabulka by tak neměla nikdy být zaplněna. Třetí existující tabulka v rámci aplikace se nazývá `data` a slouží pro uchování dalších pomocných proměnných. Jelikož nové hodnoty do tabulky `prices` časem stále přibývají, ale její kapacita je omezená, musí existovat mechanismus, který limituje počet zapsaných prvků. To je implementováno přes

pomocnou proměnnou, která uchovává hodnotu aktuálně zapisovaného indexu. Tato proměnná se nachází právě v tabulce `data`. Pokud hodnota zapisovaného indexu do tabulky `prices` vzroste nad počet maximálních prvků, resetuje se na nulu a novými hodnotami jsou pak nahrazovány nejstarší data v tabulce.

Swoole Cache je abstrakce pro ukládání obecných dat do Swoole Tables bez nutnosti definovat pro ně samostatnou tabulku. V aplikaci se využívá pro ukládání hodnot získaných z API, které figurují jako samostatné proměnné. Příkladem můžou být všechny hodnoty v modulech „informace o posledním bloku“nebo „dlouhodobé statistiky“. Jelikož je Swoole Cache implementovaná jako nadstavba nad klasickým Laravel Cache systémem, lze definovat i čas, po který je hodnota v cache uložena. Například hodnoty z modulu dlouhodobých statistik mají definován čas uložení na jednu hodinu. Pokud tento čas vyprší a hodnoty se z cache smažou, právě uživatelův další request vyvolá obnovení těchto hodnot (naopak u modulů s grafy se hodnoty obnovují periodicky).

Pro definování periodicky volaných úkonů existuje abstrakce **Swoole Timer**. Ta umožňuje nastavit chování metody, aby se spustila znovu po uplynutí definovaného počtu sekund. V aplikaci existují dva takové časovače. První z nich se spouští každých 10 sekund a získává aktuální cenu BTC. Druhý časovač má jako periodu 1 hodinu a získává hodnoty pro graf těžařských poolů.

6 Metodika vývoje aplikace a automatizované nasazení

6.1 Metodika vývoje

V této kapitole je kompletně popsán proces vývoje ukázkové aplikace včetně vysvětlení důvodu volby jednotlivých technologií, celkového konceptu aplikace a rozvedení detailů implementace.

6.1.1 Koncept

Prvním krokem tvorby aplikace byla volba konceptu. Pro demonstraci principů asynchronního programování bylo nutné, aby výsledná aplikace obsahovala IO-bound operace. Koncepty, které tuto podmínku splňují jsou např.: chatovací aplikace, IoT řešení nebo herní server. Tyto koncepty jsou ovšem pro účely demonstrace asynchronního principu až příliš složité. Z důvodu zachování jednoduchosti byl tedy zvolen koncept takové aplikace, která volá různé API endpointy, ukládá a zpracovává získaná data a ty poté zprostředkovává uživateli. První zvažovanou možností byla sportovní aplikace, která na základě uživatelem zadaného datumu vytvoří seznam utkání (které se konají v tento den) napříč různými sporty. Tato varianta byla nakonec vyhodnocena jako nevyhovující a místo ní byla vytvořena aplikace BitcoinSpotlight. Její koncept je podobný, ovšem s tím rozdílem, že data získává z API Bitcoinu (místo sportovních API) a nemá žádný uživatelský vstup.

6.1.2 Grafický návrh

Po zvolení konceptu aplikace bylo nutné vytvořit grafický návrh a implementovat ho do frontendového kódu. Implementace je provedena s využitím CSS frameworku *Bulma* pro obecné stylování a knihovny *Animate.css* pro tvorbu animací. Primárním motivem grafického návrhu je kompaktnost zobrazení. Cílem aplikace je tedy předat data uživateli přehledně a při tom efektivně využít místo displeje. Statistiky uvádí, že téměř 60 % návštěv webových stránek pochází z mobilních zařízení [36]. Z tohoto důvodu je aplikace plně responzivní a její design je pomocí CSS Media Queries uzpůsoben pro několik možných šířek displeje. Grafický návrh jednotlivých modulů je tvořen s ohledem na možnou rozšiřitelnost do budoucna a je tedy univerzální. Aby se mezi moduly dalo rychle vizuálně orientovat, vzhled každého z nich je laděn do jednotné barvy, která je mezi ostatními moduly unikátní.

6.1.3 Frontend a backend

Dalším krokem vývoje byla tvorba frontendové logiky a implementace komunikace s backendem. Frontendová logika zahrnuje zaprvé ukládání stavu ovládacích prvků do paměti prohlížeče a jeho následné načtení při znovuotevření stránky, zadruhé funkcionalitu časovače. Komunikace s backendem probíhá pomocí JavaScript knihovny HTMX. Tento způsob je zvolen primárně z toho důvodu, že přítomné moduly často odesílají požadavky na backend; ovšem každý s jiným intervalem. Při obnově dat jednoho z modulů tak není nutno načítat celou stránku znovu, lze si z backendu vyžádat pouze fragment HTML kódu daného modulu a tím ho na stránce nahradit.

Poslední částí implementace byla tvorba backendu. Výhodou aplikace tvořené s HATEOAS přístupem a knihovnou HTMX je, že může být spárována s libovolným backendem. Aplikace využívá framework Laravel ve kterém pro každý modul definuje vlastní routu, ta je poté volána z frontendu pro obnovení dat daného modulu. Atypickou vlastností backendu této konkrétni aplikace je, že neobsahuje persistentní databázi. Tak je učiněno z tohoto důvodu, že aplikace ji nepotřebuje. Data získaná aplikací přestanou být aktuální po krátké době, není tedy nutné je trvale ukládat. Místo toho využívá Swoole Tables, což je konstrukce umožňující ukládání dat přímo do RAM.

6.2 Prostředí aplikace

Pro zajištění standardizovaného prostředí pro běh aplikace je využit Docker a Docker Compose. Aby se vytvořené kontejnery daly snadno ovládat, součástí aplikace je také Laravel Sail (dále jen Sail). Jedná se o odlehčený CLI nástroj, který slouží jako nadstavba nad funkcionalitou Dockeru [37]. Díky propojení s Laravel ekosystémem nabízí Sail také možnost přímého spouštění Artisan příkazů v běžícím kontejneru. Jelikož Sail je dostupný v rámci balíčkovacího systému Composer, po naklonování repozitáře aplikace do nového zařízení nastane problém, že nejsou lokálně nainstalovány žádné závislosti (tedy ani Sail). Instalace závislostí ovšem vyžaduje composer a PHP lokálně dostupné, což představuje určitý paradox oproti filozofii používání Dockeru. Pro zpřístupnění Sail příkazů poskytuje oficiální dokumentace tedy příkaz, který vytvoří dočasný a jednoúčelový Docker kontejner. Ten nainstaluje potřebné závislosti do aplikace (složka `vendor`) uvnitř Docker prostředí, poté je přesune do lokálního prostředí a nakonec odstraní sám sebe. Po spuštění tohoto příkazu lze využívat Sail příkazy také z lokální příkazové řádky mimo Docker kontejner.

Celá aplikace běží v rámci frameworku Laravel verze 10 a jazyka PHP verze 8.3. Kromě již zmíněného nástroje Laravel Sail by bylo možné aplikaci rozšířit také o další podobné nástroje z Laravel ekosystému. Jedním z příkladů může být **Laravel Pint**, což je minimalistický nástroj pro statickou analýzu kódu, který umožňuje kód formátovat. Je postaven jako nadstavba nástroje PHP-CS-Fixer. Ten poskytuje buď možnost využití již existujícího standardu pro formátování (např. PSR-2) nebo definování vlastního standardu. Laravel Pint je tedy vlastní definovaný standard formátovacích pravidel vytvořený specificky pro Laravel [38]. Od verze Laravel

9 se nachází v základní instalaci každé aplikace, jeho volání by tedy bylo možné integrovat do Docker prostředí, případně propojit s automatizovaným nasazením aplikace.

6.3 Automatizované nasazení

V dnešní době je proces sestavení zdrojového kódu a následného nasazení aplikace velmi často automatizován pomocí tzv. Continuous Integration (CI) a Continuous Delivery (CD). Statistiky uvádí, že až 22 % repozitářů (měřeno pouze z repozitářů s více než 1000 hvězdami) obsahují nějakou formu CI/CD [39]. Jelikož jako verzovací platforma pro tento projekt byl zvolen GitHub, CI/CD bylo implementováno pomocí vestavěného nástroje **GitHub Actions**. Ten umožňuje vytvářet tzv. „workflows“ a v nich definovat libovolné procesy, které proběhnou za určité podmínky (např. když nastane push, release, pull request atd.).

6.3.1 Konkrétní popis

Celý proces nasazení aplikace je definován v souboru `.github/workflows/ci.yml`. Podmínkou jeho spuštění je vytvoření release v repozitáři aplikace. Tento release musí být také oficiálně uvolněn (je ve stavu „published“). Pokud je tato podmínka splněna, v rámci GitHubu se spustí Docker kontejner založený na distribuci Ubuntu, který se přes SSH připojí na vzdálený server, kde aplikace běží. K připojení potřebuje IP adresu, jméno a heslo. Tyto údaje jsou uloženy ve skrytých proměnných (secrets), které se uchovávají zašifrované na GitHubu. Po připojení na server se skript přesune do adresáře s aplikací, stáhne si aktuální změny z git repozitáře a pomocí Sail příkazů aplikaci znovu sestaví a spustí. Tento proces předpokládá, že na vzdáleném serveru bylo již provedeno prvotní nastavení prostředí aplikace (pro dostupnost volání Git a Sail příkazů).

7 Závěr

Tato bakalářská práce se zabývala problematikou asynchronního programování v PHP. V rámci rešerše byl nejprve prozkoumán asynchronní přístup obecně, poté v kontextu webu a nakonec konkrétně v jazyce PHP. Byly popsány výhody a nevýhody asynchronního přístupu oproti synchronnímu a paralelnímu, mechanismus Event Loop pro použití v prostředí webu a dostupné možnosti implementace asynchronního přístupu v PHP.

V druhé části práce bylo provedeno porovnání vybraných asynchronních PHP technologií. Nejprve byla uvedena metodika výběru těchto technologií a seznam nezahrnutých. Vybrané technologie byly poté porovnány na základě stanovených kritérií, výsledky jsou kategorizované do tří tabulek. Dále bylo sestaveno prostředí pro benchmark těchto technologií s využitím Dockeru. V jazyce Python byly vytvořeny skripty, pomocí kterých lze benchmark spustit a vygenerovat grafy z naměřených dat. První provedený benchmark obsahuje pouze PHP technologie. Jeho vítěz je poté zařazen do druhého benchmarku, který rozšiřuje porovnání o další programovací jazyky podporující asynchronní programování.

Jako vítěze PHP porovnání lze označit Swoole. Z porovnávaných technologií poskytuje nejstabilnější a nejuniverzálnější platformu pro tvorbu asynchronních aplikací. Lze ho také propojit s frameworkem Laravel a v provedeném benchmarku vyšel jako nejrychlejší možnost. Swoole obstál také v benchmarku s ostatními jazyky. Lze tedy říci, že v kontextu asynchronních web serverů se při zvolení vhodné technologie dokáže jazyk PHP rovnat ostatním jazykům.

Nakonec byla vytvořena ukázková aplikace, která demonstruje principy asynchronního programování. Tato aplikace se nazývá *BitcoinSpotlight* a slouží pro čtení vybraných informací z Bitcoin ekosystému. Byla vytvořena s využitím frameworku Laravel a asynchronního web serveru Swoole. Pro komunikaci mezi frontendem a backendem využívá HATEOAS přístup, na frontendu knihovnu HTMX. Aplikace běží v prostředí Dockeru a pomocí GitHub Actions je vytvořeno automatizované nasazení na produkční server. Aplikace byla také spuštěna do ostrého provozu.

Práci by bylo možné rozšířit primárně v oblasti benchmarku. Pro přesnější výsledky měření by bylo vhodné vytvořit vlastní výchozí image pro kontejnery měřených technologií. Tento image by obsahoval zkompilované PHP a byl by pro všechny měřené technologie totožný. Dále lze do benchmarku zahrnout IO-bound operace, případně otestovat více různých konfigurací technologií. Ukázková aplikace by šla rozšířit např. přidáním dalších modulů pro jiné kategorie dat. Další možností rozšíření ukázkové aplikace je implementace exportu a importu uloženého nastavení,

případně přidat možnost přenastavení počtu sekund u časovače pro automatickou obnovu dat modulu.

Asynchronní programování v PHP je stále relativně nový koncept. Vzhledem k dlouhé synchronní historii jazyka PHP ho některé starší knihovny nemusí podporovat, problém může být také s externími nástroji na ladění (např. xdebug). Programátor si navíc musí dávat větší pozor na práci s pamětí (hrozí memory leak). Ekosystém okolo asynchronního programování v PHP ovšem stále roste a vznikají nové nástroje, jako např. FrankenPHP. Asynchronní přístup sice není univerzálním řešením pro všechny webové aplikace, ovšem u aplikací s vysokým množstvím příchozích requestů naráz a IO-bound operacemi v backendové logice může výrazným způsobem zlepšit výkon.

Seznam použité literatury

1. TERŠL, Adam. *Paralelní programování a datové struktury v C++*. 2016. Bakalářská práce. Masarykova univerzita.
2. DOU, Bruce. *Mastering Swoole PHP: Build high performance concurrent system with async and coroutines*. Transfon, 2020. ISBN 978-18-381-3440-2.
3. GOEL, Nishu. *The Web Almanac by HTTP Archive: JavaScript* [online]. 2021-12. [cit. 2024-05-08]. Dostupné z: https://almanac.httparchive.org/en/2021/javascript#ajax.
4. GOLD, Tomáš. *Asynchronní JavaScript pod pokličkou aneb Eventloop v praxi* [online]. 2018-03. [cit. 2024-05-08]. Dostupné z: https://zdrojak.cz/clanky/asynchronni-javascript-poklickou-aneb-eventloop-praxi/.
5. O'PHINNEY, Matthew Weier. *How to Use Apache HTTPD With php-fpm and mod_php* [online]. 2023-12. [cit. 2024-05-08]. Dostupné z: https://www.zend.com/blog/apache-phpfpm-modphp.
6. MILADEV95. *CGI vs FastCGI vs PHP-FPM* [online]. 2023-07. [cit. 2024-05-08]. Dostupné z: https://medium.com/@miladev95/cgi-vs-fastcgi-vs-php-fpm-afbc5a886d6d.
7. MARCHANT, Jack. *Exploring Async PHP* [online]. 2023-05. [cit. 2024-05-08]. Dostupné z: https://dev.to/jackmarchant/exploring-async-php-5b68.
8. TECHNOLOGIES, Zend. *Fibers* [online]. [cit. 2024-05-08]. Dostupné z: https://www.php.net/manual/en/language.fibers.php.
9. PHPWATCH. *PHP 8.1: Fibers* [online]. [cit. 2024-05-08]. Dostupné z: https://php.watch/versions/8.1/fibers.
10. AARON PIOTROWSKI, Niklas Keller. *PHP RFC: Fibers* [online]. 2021-03. [cit. 2024-05-08]. Dostupné z: https://wiki.php.net/rfc/fibers.
11. IBM. *Memory-leaking programs* [online]. 2023-03. [cit. 2024-05-08]. Dostupné z: https://www.ibm.com/docs/es/aix/7.2?topic=performance-memory-leaking-programs.
12. DOU, Bruce. *Open Swoole Documentation* [online]. 2023-02. [cit. 2024-05-08]. Dostupné z: https://openswoole.com/docs.
13. FOREMTEHAN. *Swoole's admin interface hot-loads code from a third-party server* [online]. 2021. [cit. 2024-05-08]. Dostupné z: https://github.com/swoole/swoole-src/issues/4434.

14. DOU, Bruce. *Open Swoole extension v4.7.1* [online]. 2021-10. [cit. 2024-05-08]. Dostupné z: https://news-web.php.net/php.pecl.dev/17446.
15. ROADRUNNER. *RoadRunner documentation* [online]. [cit. 2024-05-08]. Dostupné z: https://roadrunner.dev/docs.
16. AMPHP. *Amphp documentation* [online]. [cit. 2024-05-08]. Dostupné z: https://amphp.org/.
17. WALKOR. *Workerman documentation* [online]. [cit. 2024-05-08]. Dostupné z: https://manual.workerman.net/doc/en.
18. DUNGLAS, Kévin. *FrankenPHP: Modern App Server for PHP* [online]. [cit. 2024-05-08]. Dostupné z: https://frankenphp.dev/docs.
19. REACTPHP. *ReactPHP documentation* [online]. [cit. 2024-05-08]. Dostupné z: https://reactphp.org/.
20. TWOSEE. *Swow documentation* [online]. [cit. 2024-05-08]. Dostupné z: https://docs.toast.run/swow/en.
21. KLENOV, Kirill. *Python Async (ASGI) Web Frameworks Benchmark* [online]. 2022-03. [cit. 2024-05-08]. Dostupné z: https://klen.github.io/py-frameworks-bench.
22. CHOUBEY, Mayank. *Node.js: The fastest web framework in 2024* [online]. 2024-01. [cit. 2024-05-08]. Dostupné z: https://medium.com/deno-the-complete-reference/node-js-the-fastest-web-framework-in-2024-fa11e513fa75.
23. SMALLNEST. *Go web framework benchmark* [online]. 2020-05. [cit. 2024-05-08]. Dostupné z: https://github.com/smallnest/go-web-framework-benchmark.
24. BINANCE. *Vysvětlení likvidity* [online]. 2018-12. [cit. 2024-05-08]. Dostupné z: https://academy.binance.com/cs/articles/liquidity-explained.
25. NAKAMOTO, Satoshi. *Bitcoin: A Peer-to-Peer Electronic Cash System* [online]. 2008-10. [cit. 2024-05-08]. Dostupné z: https://bitcoin.org/bitcoin.pdf.
26. RAVAL, Nihar. *Top JavaScript Statistics You Must Know to Kickstart Your Business in 2024* [online]. 2024-02. [cit. 2024-05-08]. Dostupné z: https://radixweb.com/blog/top-javascript-usage-statistics.
27. FIELDING, Roy T. *Architectural Styles and the Design of Network-based Software Architectures*. 2000. Dis. pr. University of California, Irvine.
28. GROSS, Carson. *Hypermedia APIs vs. Data APIs* [online]. 2021-07. [cit. 2024-05-08]. Dostupné z: https://htmx.org/essays/hypermedia-apis-vs-data-apis/.
29. SOFTWARE, Big Sky. *HTMX documentation* [online]. 2024-05. [cit. 2024-05-08]. Dostupné z: https://htmx.org/docs/.

30. PETROS, Alexander. *Is htmx Just Another JavaScript Framework?* [online]. 2024-01. [cit. 2024-05-08]. Dostupné z: https://htmx.org/essays/is-htmx-another-javascript-framework/.

31. SOFTWARE, Big Sky. *Hyperscript documentation* [online]. 2024-05. [cit. 2024-05-08]. Dostupné z: https://hyperscript.org/docs/.

32. GROSS, Carson. *Locality of Behaviour (LoB)* [online]. 2020-05. [cit. 2024-05-08]. Dostupné z: https://htmx.org/essays/locality-of-behaviour/.

33. ANKALKOTI, Prashant; SANTHOSH. A Relative Study on Bitcoin Mining. *Imperial Journal of Interdisciplinary Research (IJIR)*. 2017.

34. DEVABIT. *An expert guide to PHP framework popularity in 2023* [online]. 2023. [cit. 2024-05-08]. Dostupné z: https://devabit.com/blog/php-framework-popularity-2023/#php-framework-popularity-top-options.

35. OTWELL, Taylor. *Laravel documentation* [online]. 2024-05. [cit. 2024-05-08]. Dostupné z: https://laravel.com/docs/10.x.

36. STATCOUNTER. *Desktop vs Mobile vs Tablet Market Share Worldwide* [online]. [cit. 2024-05-08]. Dostupné z: https://gs.statcounter.com/platform-market-share/desktop-mobile-tablet.

37. BUTTI, Roberto. *High performance with Laravel Octane: learn to fine-tune and optimize PHP and Laravel apps using Octane and an asynchronous approach*. Birmingham: Packt Publishing, 2022. ISBN 978-1-80181-904-6.

38. OTWELL, Taylor. *Laravel Pint* [online]. [cit. 2024-05-08]. Dostupné z: https://laravel.com/docs/10.x/pint.

39. CHEN, Tingting; ZHANG, Yang; CHEN, Shu; WANG, Tao; WU, Yiwen. Let's Supercharge the Workflows: An Empirical Study of GitHub Actions. In: *2021 IEEE 21st International Conference on Software Quality, Reliability and Security Companion (QRS-C)*. 2021.

A Odkaz na zdrojový kód

- Zdrojové kódy benchmarku a aplikace (aktuální ke dni 14. 5. 2024) si lze zobrazit zde:
  https://github.com/arbyys/tul-bp