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

KIV-VSS Výkonnost a spolehlivost prog. systémů

Benchmarking - REST API

Hana Hrkalová

21. 1. 2023 Plzeň

Obsah

Specifikace a rešerše

První část dokumentace

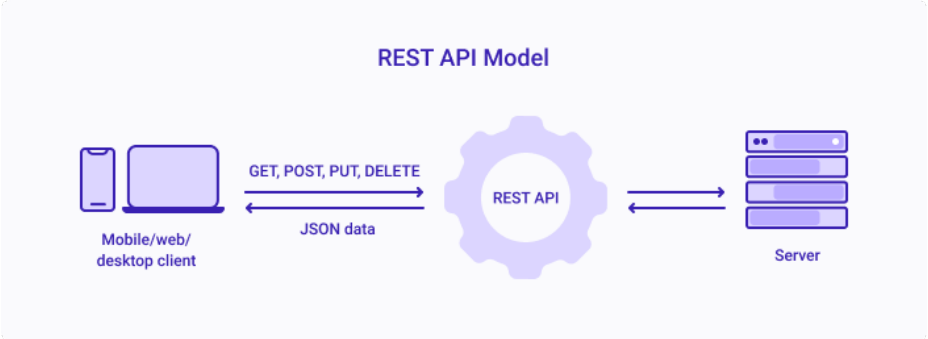
Specifikace a rešerše

1 Rest API

1.1 API

Obecně API slouží pro komunikaci s počítačem nebo systémem a za účelem zprostředkování informace nebo provedení nějaké funkce, rozhraní API pomáhá sdělit systému co chce uživatel, aby mohl požadavek pochopit a splnit.

Rozhraní API si lze představit jako prostředníka mezi uživateli (klienty) a prostředky nebo webovými službami, které chtějí získat.



Obrázek 1: Schéma REST API

1.2 REST

Pro RESTful API je nutné splnit následující kritéria:

- Architektura klient-server: rozhraní se skládá z klientů, serverů a zdrojů, přičemž požadavky jsou spravovány pomocí protokolu HTTP.
- Bezstavová komunikace klient-server: mezi požadavky nejsou ukládány žádné informace o klientovi, každý požadavek je samostatný a nespojitý.
- Ukládání dat do mezipaměti: mezipaměť zefektivňuje interakci mezi klientem a serverem.
- Jednotné rozhraní mezi komponentami: informace jsou přenášeny ve standardní podobě, což vyžaduje:
  - identifikovatelné a oddělené požadované zdroje od reprezentací zasílaných klientovi

- umožnit klientovi manipulovat se zdroji prostřednictvím reprezentace, kterou obdržel
- poskytnout zprávy vrácené klientovi s dostatečnými informacemi k popisu toho, jak je má klient zpracovat

Obecně REST (Representational State Transfer) API je tedy typ webového rozhraní, které umožňuje komunikaci mezi různými systémy pomocí standardních HTTP metod jako GET, POST, PUT a DELETE. REST API je založeno na architektuře klient-server, kde klient posílá požadavky na server a ten mu odpovídá zpět s daty ve formátu, který si klient vyžádal (nejčastěji ve formátu JSON nebo XML).

2 Popis problému

Tvorba benchmarků pro REST je proces měření výkonnosti rozhraní API. Srovnání lze provést mezi různými frameworky, poskytujícími totožné Rest API, hlavním cílem je zjistit, jak framework funguje a který se vyplatí využívat například pro malá řešení a který například pro robustní aplikace, kde bude velká zátěž na API.

2.1 Kritéria REST API

Benchmarky pro API mohou být různé, dle toho co je pro nás u implementovaného API zásadní. Obecně můžeme říct, že existuje šest základních kritérií, které určují jak dobré je naše API.

- Rychlost – jak rychlé je rozhraní API? (toto kritérium je z hlediska klienta asi nejzásadnější, jelikož je tato vlastnost vidět jako první)
- Jednoduchost používání – jak snadné je API používat? (zde také záleží jaký druh klientů bude API využívat, zda jde o vývojáře či nikoliv)
- Stabilita – Nestabilní rozhraní API má časté prostoje nebo vrací chybová hlášení (zvyšuje pravděpodobnost, že naše API nebude využíváno klienty)
- Škálovatelnost – Škálovatelné API zvládne zpracovávat rostoucí počet požadavků, když ho začne používat více uživatelů. (Pokud API není škálovatelné, pravděpodobně se zhroutí nebo vrátí chybu, když se ho pokusí použít příliš mnoho lidí současně)
- Bezpečnost – Bezpečné API je takové, které chrání uživatelská data a zabraňuje neoprávněnému přístupu. (Někdy se REST API používá jako způsob, jak může organizace sdílet zdroje a informace při zachování zabezpečení, kontroly a ověřování)
- Dokumentace – Pokud je rozhraní API špatně zdokumentováno, bude pravděpodobně obtížné ho používat a lidé ho budou používat méně.

2.2 Typy výkonnostních testů

Můžeme mít také několik druhů testů, které naše API otestují.

- Standardní testování – Testování zátěže se používá ke zjištění, jak dobře API funguje při běžném zatížení. Simuluje reálné scénáře použití, aby bylo možné posoudit výkon rozhraní API a identifikovat případná úzká místa.
- Zátěžové testování – Zátěžové testování určuje bod zlomu rozhraní API tak, že zvýší zátěž nad rámec běžných podmínek a dostane se na hranici, kdy dojde k jeho zlomu. To pomáhá určit maximální zátěž, kterou rozhraní API zvládne, což je užitečné pro plánování škálovatelnosti.
- Testování odolnosti – Testování odolnosti určuje, jak dobře API funguje po delší dobu, tj. ověřuje výdrž aplikace nebo odolnost systému. Tento typ testování pomáhá identifikovat případné snížení výkonu v průběhu času a případné úniky paměti.
- Spike testování – Spike testování hodnotí schopnost aplikace zvládnout náhlé zvýšení objemu. Provádí se náhlým zvýšením požadavků generovaných velkým počtem uživatelů. Cílem je posoudit, zda výkon aplikace utrpí, zda systém selže, nebo zda bude schopen zvládnout náhlé dramatické změny uživatelských požadavků.

Pro zajištění různých podmínek prostředí API lze využít celá řada nástrojů, které omezují výkon a simulují různé zátěže. Například `ab` nebo `wrk`.

3 Popis řešení

3.1 Kroky pro srovnávání výkonu rozhraní API

Obecně se pro testování výkonnosti definuje několik kroků.

- Definujte výkonnostní cíle – Prvním krokem při srovnávání výkonu je definování výkonnostních cílů pro vaše API, jako je doba odezvy, propustnost a počet současně pracujících uživatelů.
- Výběr nástroje pro testování výkonu – Vyberte si správný nástroj pro své konkrétní potřeby na základě faktorů, jako je typ rozhraní API, velikost projektu..
- Příprava testovacího prostředí – Nastavte testovací prostředí, vytvořte testovací data a nakonfigurujte nástroj pro testování výkonu. Zajistěte, aby se testovací prostředí věrně podobalo produkčnímu prostředí, abyste získali přesné výsledky.
- Proveďte testy – Proveďte testy vícekrát, abyste získali přesný obraz o výkonu rozhraní API.
- Analyzujte výsledky – Analyzujte výsledky, včetně doby odezvy, propustnosti a počtu současně pracujících uživatelů, a porovnejte je s výkonnostními cíli definovanými v kroku 1. Identifikujte případné bottlenecky a oblasti pro zlepšení.
- Optimalizujte výkon – Na základě analýzy v případě potřeby optimalizujte výkon rozhraní API změnou kódu nebo úpravou konfigurace.

3.2 Konkrétní popis řešení

3.2.1 Co bude testováno

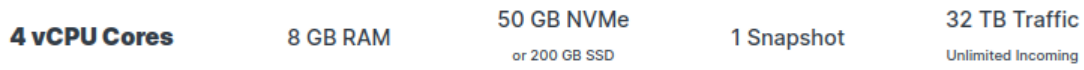
Pro otestování mého REST API bude zásadní první kritérium zmíněné v popisu problematiky, tedy rychlost. Dále se zaměřím na škálovatelnost a snadnost použití. Konkrétně budu porovnávat 5 technologií (FrameWorků) pro vytvoření REST API. Technologie jsem se snažila vybrat s ohledem na odlišnost (druh jazyka v kterém je implementován) aby bylo testování rozmanitější (typované-/netypované jazyky...).

- Spring Boot (Java)
- .NET Core
- Laravel 5 (PHP)
- Flask (Python)
- Ruby on Rails (Ruby)

Frameworky jsou vybrány i s ohledem na na mé zkušenosti. Vybrala jak jazyky s kterými jsem dobře obeznámena (Java, Python..), tak ale také jazyky, v kterých jsem nikdy neprogramovala (Ruby). Jde jak o pozorování zda mají různé typy jazyků na testování vliv, tak také o rozšíření mých znalostí v tomto směru.

3.2.2 Prostředí

Jednotlivé REST API musí mít stejné podmínky testování, aby byly výsledky relevantní. To lze zařídit jednotným prostředím, kde budou jednotlivé severy implementovány. Využiji pro to soukromý server se stálými parametry.



Obrázek 2: Parametry serverového prostředí

Pro vetší rozmanitost benchmarků bude možné využít nástroj pro limitaci hardwaru (kubernetes).

3.2.3 Typ dat

Většina benchmarků je tvořena pouze pro jeden typ dat, proto chci vytvořit webové rozhraní REST API, které vystavuje 4 endpointy, z nichž každý má jinou složitost:

- Hello World - jednoduše odpoví řetězcem JSON obsahujícím Hello World.
- Požadavkem bude matematický výpočet
- Jednoduchý výpis - výpis výčtu hodnot z databáze MySQL
- Složitý výpis - výpis s pomocí databáze MySQL s požadavky na databázi

Pro jejich testování použiji pravděpodobně nástroj `wrk` pro HTTP benchmarking, abych ověřila, zda dostanu podobné výsledky, a jak již bylo zmíněno pro možnost měnit souběžnost požadavků, aby každá technologie mohla využít svůj maximální potenciál.

Druhá část dokumentace

Popis řešení

4 Implementace a příprava

4.1 Příprava prostředí

Na prostředí bylo třeba nainstalovat všechny balíčky potřebné pro zvolené programovací jazyky, volání REST a práci s databází.

1. Spring Boot (Java)
    - java - verze 17+
    - gradle - verze 7+
2. .NET Core
    - apt-transport-https
    - dotnet-sdk-3.1, dotnet-runtime-3.1, aspnetcore-runtime-3.1
3. Express JS
    - nodejs, npm, node
    - express
4. Flask (Python)
    - python3 python3-pip python3-venv
    - flask
5. Ruby on Rails (Ruby)
    - rbenv and Dependencies
    - Ruby with ruby-build
    - Rails

4.2 Vytváření testovacích dat

Pro každý testovaný jazyk bylo třeba napsat zdrojové kódy, které spouští požadované operace (Hello Word, Fibbonacci, práce s DB).

```java
package cz.zcu.fav.kiv.vss.spring_boot_benchmark;
import org.springframework.data.repository.query.Param;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RestController;

import java.math.BigDecimal;
import java.util.Set;

@RestController
public final class Controller {

    private final TestEntityDao dao;

    public Controller(TestEntityDao dao) {
        this.dao = dao;
    }

    @GetMapping("/greet")
    public String greet() {
        return "hello";
    }

    @GetMapping("/fibonacci/{number}")
    public BigDecimal getFibonacci(@PathVariable("number") long number) {
        BigDecimal first = BigDecimal.ZERO, second = BigDecimal.ONE, temp;
        for (long i = 0; i < number; i++) {
            temp = first.add(second);
            first = second;
            second = temp;
        }
        return first;
    }

    @GetMapping("/name_list")
    public Set<String> listAllNames() {
        return dao.listAllNames();
    }


    @GetMapping("/find_by_name")
    public TestEntity findByName(@Param("name") String name) {
        return dao.findByName(name);
    }
}
```

Zdrojový kód 4.1: Ukázka programu pro Spring Boot

Kromě hlavní třídy jsou navíc vytvořené třídy reprezentující data a jejich atributy v databázi. O spuštění a sestavení programu se pro `Spring Boot` stará Gradle. Kromě testovacích programů a potřebných balíčků na serveru nic jiného není, abych pojistila izolaci operací při jednotlivém volání (ve smyslu čisté instalace).

4.3 Nastavení databáze

Databáze je nastavena dle následujícího návodu.
https://www.cherryservers.com/blog/how-to-install-and-start-using-mariadb-on-ubuntu-20-04

Dále je třeba naplnit databázi testovacími daty. To uděláme tak, že z hlavního adresáře spustíme:

```
mariadb -u root -p < create.sql
mariadb -u root -p < sub_create.sql
```

Zdrojový kód 4.2

4.4 Spuštění testovacích dat

Nejprve je třeba na serveru spustit implementaci konkrétní REST služby.

```
cd "Dot Net/Dot Net"
dotnet run --urls "http://ip_addr:port"

cd "Spring Boot"
gradle bootRun


cd Flask
flask run --host=0.0.0.0 --port=cislo_portu

cd "Express js"
node controller.js

cd "Ruby on Rails"
bin/rails server -p cislo_portu -b 0.0.0.0
```

Zdrojový kód 4.3: Způsoby spuštění REST

Následně můžeme posílat jednotlivé dotazy.

```
wrk -t2 -c5 -d5s --latency --timeout 10s  + :
    "http://ip_addr:port/find_by_name?name=Bruno%20Duke"
    "http://ip_addr:port/name_list"
    "http://ip_addr:port/fibonacci/20000"
    "http://ip_addr:port/greet"
```

Zdrojový kód 4.4: Příkazy pro jednotlivé dotazy na služby

Wrk je moderní nástroj pro benchmarkování HTTP, který je schopen generovat značnou zátěž při běhu na jednom vícejádrovém procesoru. Kombinuje vícevláknový design se škálovatelnými systémy oznamování událostí, jako jsou epoll a kqueue. Příkaz `wrk` zařizuje že dle parametrů:

- t2: Použít dvě samostatná vlákna
- c5: Otevře šest spojení (první klient je nulový)
- d5s: Test bude probíhat po dobu pěti sekund
- `-timeout 2s`: Definuje dvousekundový timeout
- `-latency`: vypíše podrobnější informace o datech latence

```
hana@Hana:~$ wrk -t 4 -c 100 -d 18s --latency "http://ip_adresa:port/greet"
 Running 18s test @ http://ip_adresa:port/greet
   4 threads and 100 connections
   Thread Stats   Avg      Stdev     Max   +/- Stdev
     Latency    16.53ms   20.81ms 323.86ms   98.46%
     Req/Sec     1.75k   242.28     2.31k    77.97%
   Latency Distribution
      50%   13.89ms
      75%   16.20ms
      90%   18.93ms
      99%  112.43ms
   123374 requests in 18.04s, 18.47MB read
  Requests/sec:   6839.80
  Transfer/sec:      1.02MB
```

Zdrojový kód 4.5: Ukázka výpisu příkazu `wrk`

5 Stanovení benchmarků

Při benchmarkování je třeba testy provést vícekrát. Stanovila jsem proto počet pokusů na sedm, pro každou operaci. Ze sedmi hodnot se odstraní maximum a minimum a zprůměrují se zbývající hodnoty.

Pro jejich testování použijeme nástroje wrk, abychom ověřili, zda dostaneme podobné výsledky, a také abychom měnili souběžnost požadavků, aby každá technologie mohla využít svůj maximální potenciál.

Výsledkem bude tedy průměr hodnot, budeme sledovat latenci, počet požadavků za sekundu a přenos dat za sekundu.

6 Naměřené hodnoty

V měření se ukládaly hodnoty latence, požadavků za vteřinu a velikost dat za vteřinu.

6.1 DotNet

DotNet	Latency	Requests/sec	Transfer/sec
Greet	4,21 ms	940,0	144,13KB
Fibbo	757 ms	4,91	20,82KB
DB - ListName	23,27 ms	171,34	439.99KB
DB - FindByName	29,84 ms	146,89	378,84KB

Obrázek 3: Tabulka hodnot pro DotNet

6.2 Spring

Spring	Latency	Requests/sec	Transfer/sec
Greet	4,72 ms	829,62	95,74KB
Fibbo	133,19 ms	29,69	124,87KB
DB - ListName	9,33 ms	527,71	457,77KB
DB - FindByName	41,72 ms	97,15	6,32KB

Obrázek 4: Tabulka hodnot pro Spring

6.3 Flask

Flask	Latency	Requests/sec	Transfer/sec
Greet	11,47 ms	163,83	28,18KB
Fibbo	197,12 ms	18,67	79,39KB
DB - ListName	40,18 ms	84,16	37,6KB
DB - FindByName	40,25 ms	78,04	36,45KB

Obrázek 5: Tabulka hodnot pro Flask

6.4 Express JS

Express JS	Latency	Requests/sec	Transfer/sec
Greet	4,99 ms	802,58	179,85KB
Fibbo	202,66 ms	19,26	83,27KB
DB - ListName	8,57 ms	462,25	454,45KB
DB - FindByName	33,63 ms	117,9	7,74MB

Obrázek 6: Tabulka hodnot pro Express JS

6.5 Ruby

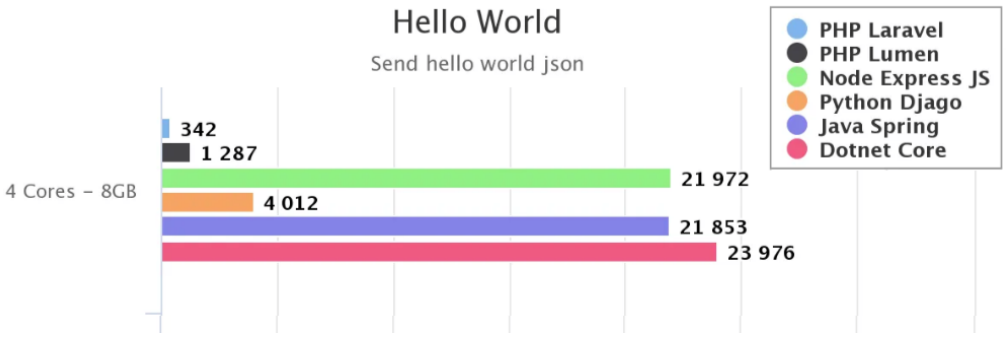
Ruby	Latency	Requests/sec	Transfer/sec
Greet	62,12 ms	63,01	35,98KB
Fibbo	501,25 ms	7,44	35,21KB
DB - ListName	94,76 ms	41,69	57,56KB
DB - FindByName	174,85 ms	22,39	1,51MB

Obrázek 7: Tabulka hodnot pro Ruby

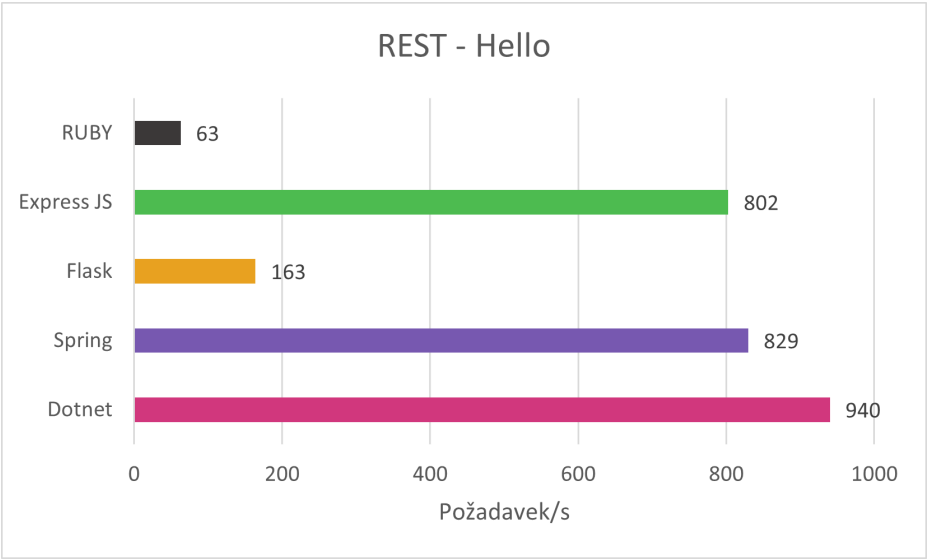
7 Porovnání výsledků

Dle replikační studie není porovnání výsledků ideální. Mé výsledky mají odlišné měřítko, aby bylo značně vidět rozdíl pro vlastní naměřené hodnoty. Snažila jsem se zachovat podobnost barev u stejných (podobných) jazyků, které jsou používány na REST API.

7.1 Greet

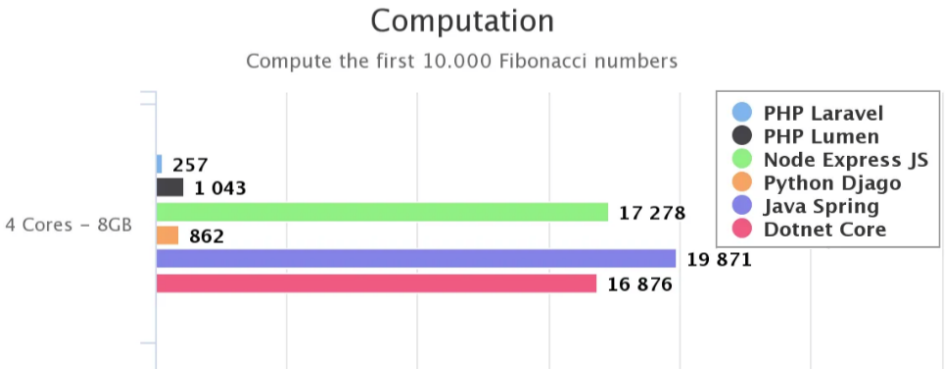


Obrázek 8: Výsledky původní studie

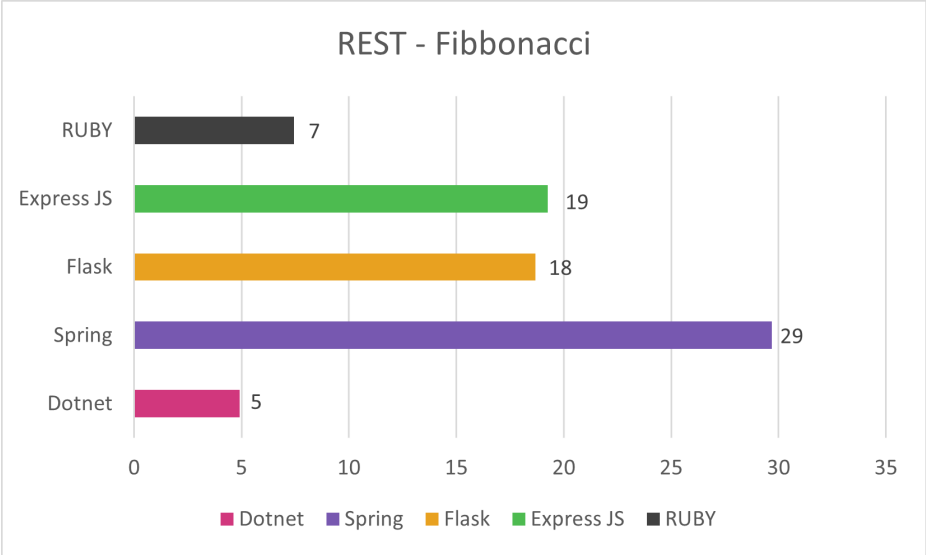


Obrázek 9: Mé aktuální výsledky

7.2 Fibbonacci

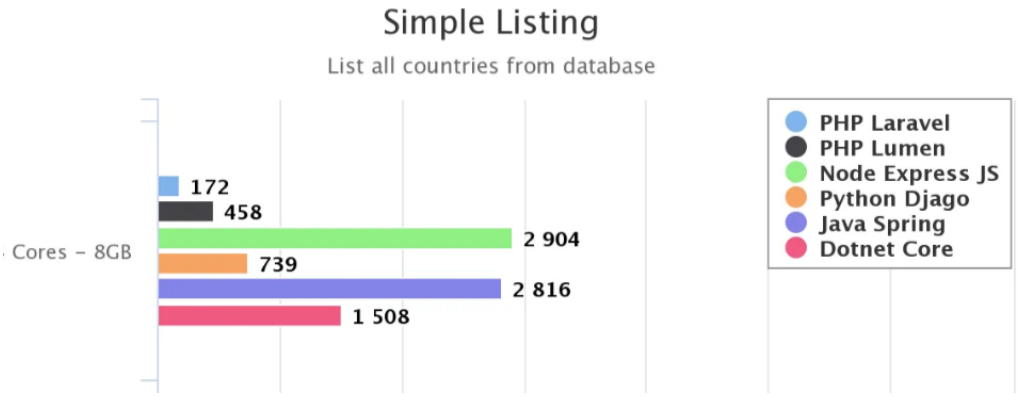


Obrázek 10: Výsledky původní studie

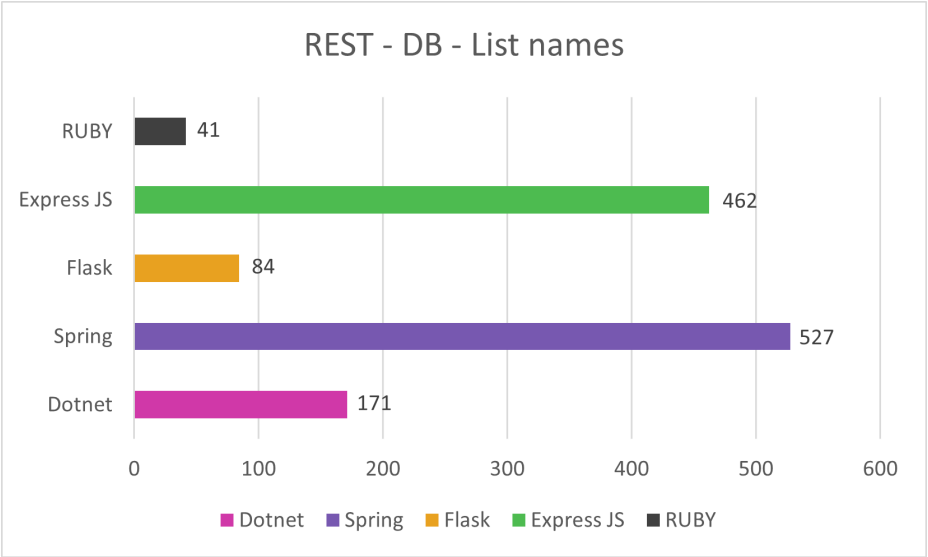


Obrázek 11: Mé aktuální výsledky

7.3 Jednoduchá práce s DB

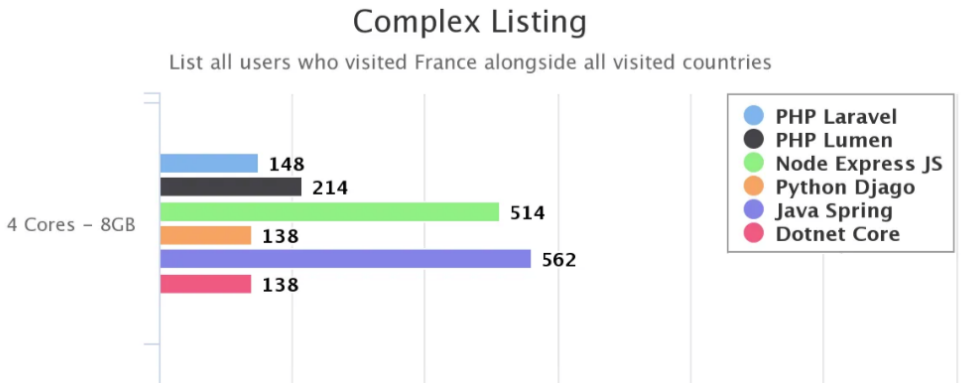


Obrázek 12: Výsledky původní studie

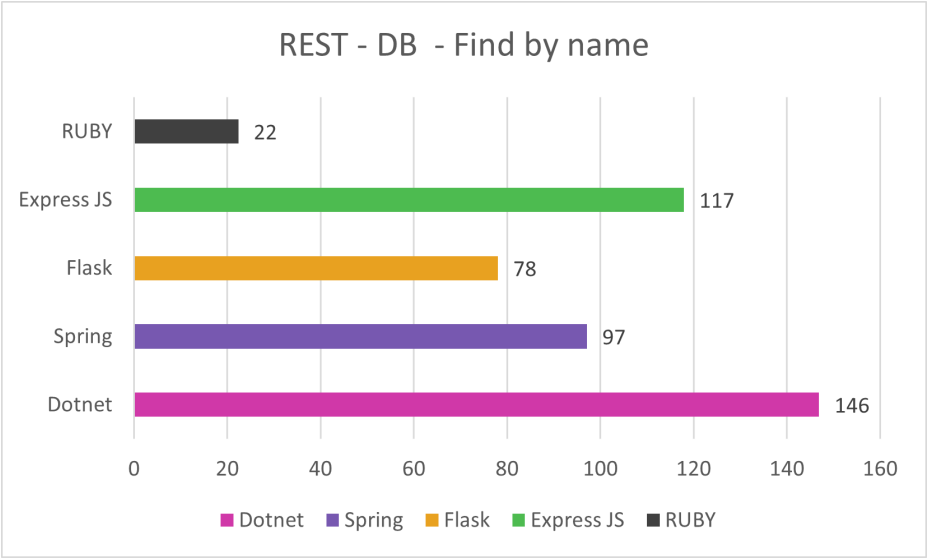


Obrázek 13: Mé aktuální výsledky

7.4 Složitější operace nad DB



Obrázek 14: Výsledky původní studie



Obrázek 15: Mé aktuální výsledky

8 Závěr

V porovnání byly v replikační studii podobné, či stejné jazyky pro REST API. Měřila se jejich latence, počet požadavků za vteřinu a objem dat za vteřinu. Ve vizualizaci výsledků v porovnání s předchozí studií se výsledky značně nemění. Měřily se požadavky na vypsání zprávy ("hello"), výpočet fibbonacciho posloupnosti a práce s databází (žádost o vypsání hodnot z tabulky a vyhledání v databázi). Z Vizualizace výsledků vyplývá že nynějším nejlepším adeptem, pokud bychom chtěli přes REST volání dělat více operací je dle rychlosti požadavků Spring. Dalších možných rozšíření práce je několik, šlo by udělat podrobnější a osbáhlejší vizualizaci stávajících naměřených dat, bylo by možné přidat další jazyky na otestování REST, nebo by šlo měření opakovat s jinými parametry příkazu `wrk`, abychom získali robustnější výsledky.

9 Zdroje

https://www.redhat.com/en/topics/api/what-is-a-rest-api
https://github.com/denji/awesome-http-benchmark
https://www.programsbuzz.com/article/why-you-should-benchmark-api-and-what-benchmarks-have
https://statusneo.com/performance-benchmarking-of-apis-a-quick-guide/
https://thechief.io/c/editorial/top-10-http-benchmarking-and-load-testing-tools
https://kubernetes.io

Hlavní podklad pro mou replikační studii
https://medium.com/@mihaigeorge.c/web-rest-api-benchmark-on-a-real-life-application-ebb743a5d7a3
https://www.moesif.com/blog/api-product-management/api-analytics/10-Most-Popular-Frameworks-For-Building-RESTful-APIs/
https://gochronicles.com/benchmark-restful-apis/
Odsud jsem vycházela z nastavení parametrů pro nástroj `wrk`
https://medium.com/@_oleksii_/how-to-benchmark-http-latency-with-wrk-a-z-guide-e5b185bd4cdc