Česká zemědělská univerzita v Praze

Provozně ekonomická fakulta

Katedra informačních technologií

Bakalářská práce

Srovnání frameworků Django a Flask na základě výkonu vybraných komponent

Mgr. Ondřej Malina

ČESKÁ ZEMĚDĚLSKÁ UNIVERZITA V PRAZE

Provozně ekonomická fakulta

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Mgr. Ondřej Malina

Informatika

Název práce

Srovnání frameworků Django a Flask na základě výkonu vybraných komponent

Název anglicky

Django and Flask Framework comparison based on performance of selected components

---

Cíle práce

Hlavní cíl práce je srovnat výkon vybraných komponent frameworků pro vývoj webových aplikací v Pythonu Django a Flask. Vedlejší cíle práce jsou:
- Vybrat vhodné komponenty frameworků pro testování
- Naprogramovat komponenty a připravit experiment
- Provést a vyhodnotit měření

Metodika

Teoretická část práce je založena na studiu a analýze odborných a vědeckých informačních zdrojů. Praktická část práce je založena na experimentu. Data pro experiment jsou sesbírána za využití nástroje vytvářejícího zátěž na měřené komponenty frameworků. Vyhodnocení je tvořeno srovnáním výkonu daných komponent, poukázáním na nedostatky měření a následným shrnutím celého experimentu. Z teoretických poznatků a experimentu budou formulovány závěry práce.

Doporučený rozsah práce

45

Klíčová slova

Experiment, Python, Flask, Django, webové aplikace, srovnání

Doporučené zdroje informací

GRINBERG, Miguel. Flask web development. Sebastopol: O'Reilly, 2014. ISBN 9781449372620.
LUTZ, Mark. Learning Python. 5th ed. Sebastopol: O'Reilly, 2013. ISBN 9781449355739.
MARTIN, Robert C. Clean code: a handbook of agile software craftmanship. Upper Saddle River, NJ: Prentice Hall, c2009. Robert C. Martin series. ISBN 978-0132350884.
VINCENT, William S. Django for beginners: build websites with Python and Django. USA: William S. Vincent, [2018-2019]. ISBN 978-1983172663.

Předběžný termín obhajoby

2021/22 LS – PEF

Vedoucí práce

RNDr. Alexander Galba

Garantující pracoviště

Katedra informačních technologií

Elektronicky schváleno dne 10. 8. 2021

doc. Ing. Jiří Vaněk, Ph.D.

Vedoucí katedry

Elektronicky schváleno dne 5. 10. 2021

Ing. Martin Pelikán, Ph.D.

Děkan

V Praze dne 24. 10. 2021

Čestné prohlášení

Prohlašuji, že svou bakalářskou práci "Srovnání frameworků Django a Flask na základě výkonu vybraných komponent" jsem vypracoval samostatně pod vedením vedoucího bakalářské práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou citovány v práci a uvedeny v seznamu použitých zdrojů na konci práce. Jako autor uvedené bakalářské práce dále prohlašuji, že jsem v souvislosti s jejím vytvořením neporušil autorská práva třetích osob.

V Praze dne 12.03.2022 _____

Poděkování

Rád bych touto cestou poděkoval RNDr. Alexandru Galbovi za odborné vedení a konzultace při zpracování bakalářské práce.

Srovnání frameworků Django a Flask na základě výkonu vybraných komponent.

Abstrakt

Bakalářská práce se zabývá webovými aplikace, frameworky pro tvorbu webových aplikací a jejich srovnáním prostřednictvím experimentu. Cílem práce je srovnat výkon vybraných komponent frameworků Django a Flask a na základě srovnání formulovat závěr o povaze těchto frameworků.

Práce je rozdělená na dvě části. První část je teoretická a byla vypracována na základě rešerše odborné literatury. Jsou v ní představeny klíčové koncepty, hlavní argumenty pro výběr srovnávaných frameworků a technologie potřebné pro provedení experimentu. Druhá část je praktická, obsahuje detailní popis mnou provedeného experimentu. Ten se skládal z naprogramování komponent, vytvoření WSGI serveru a reverse proxy serveru, spuštění aplikací na serveru a měření latence aplikací při podrobení zátěži. Komponenty srovnávané v experimentu byly vybrány na základě frekvence jejich využití ve webových aplikací a odlišné implementaci ve frameworcích.

Výstupem teoretické části je komplexní konceptuální základ nezbytný pro provedení experimentu měření webových aplikací. Výstupem praktické části je zhodnocení naměřené rychlosti srovnávaných komponent a vyvození z toho plynoucích závěrů.

Klíčová slova: experiment, Python, Flask, Django, webové aplikace, srovnání

Django and Flask Framework Comparison Based on Performance of Selected Components

Abstract

The bachelor thesis deals with web applications, frameworks for creation of web applications and their comparison through an experiment. The goal of the thesis is to compare a performance of selected components of frameworks Django and Flask and formulate a conclusion based on the comparison.

The thesis is split into two parts. The first one is theoretical and is based on review of an academic literature. Main arguments for selection of compared frameworks together with technologies necessary for conduction of the experiment and key concepts are presented in it. The second one is practical and contains detailed description of the conducted experiment, which consisted of several parts - programming of the components, creation of WSGI server and reverse proxy server, starting of the applications on the server and measurement of applications' latency under http request load. Components compared in the experiment were chosen based on frequency of their usage in web applications and different implementation in frameworks.

The outcome of the theoretical part is a complex conceptual foundation necessary for conduction of an experiment of web applications' measurement. The outcome of the practical part is an evaluation of the compared components' measured speed and deduction of consequent conclusions.

Keywords: experiment, Python, Flask, Django, web applications, comparison

Obsah

Seznam obrázků

Seznam tabulek

1 Úvod

Webová aplikace je aplikace spuštěná na webovém serveru a zobrazená prostřednictvím webového prohlížeče. Její hladký průběh zpravidla, ale ne vždy, vyžaduje připojení k internetu (výjimkou jsou například Google Dokumenty, které fungují v rámci webového prohlížeče offline).

Popularita webových aplikací neustále roste, protože nevyžadují od uživatele instalaci a prostřednictvím reklamy vydělávají hodně peněz. Stále častěji se také můžeme setkat s převodem klasických desktopových aplikací do webových aplikací. Typickým příkladem je sada kancelářských nástrojů MS Office.

Mezi známé a často využívané webové aplikace patří Facebook nebo Youtube. Počty jejich uživatelů ve světě se počítají v miliardách a důsledky jejich aktivit jsou enormní. Proto se také stávají čím dál častěji cílem zájmu nadnárodních organizací, vládních regulatorních úřadů i univerzitních výzkumů.

Vzhledem k počtu uživatelů a významu webových aplikací je naprosto nezbytné jejich perfektní zpracování z hlediska vývoje. Jedním z nástrojů běžně využívaných pro tvorbu webových aplikací jsou webové frameworky. Ty si můžeme představit jako balíky předprogramovaných mini aplikací, které vývojář zasazuje do svého kódu, aby zbytečně neztrácel čas psaním již vymyšleného.

Čím déle existují webové aplikace, tím více vzniká webových frameworků. Ne všechny webové frameworky jsou stejně kvalitní, proto je nutné podrobovat jejich výkon pravidelnému testování. Aplikace vytvořena na základě pomalého webového frameworku může mít při zvýšené uživatelské zátěži problémy s fungováním, v některých případech vyžaduje kompletní přepsání.

Podle dostupných informací je na světě přibližně 3,7 miliardy lidí bez internetu. Vzhledem k projektům jako Starlink je pouze otázkou času, než se i oni připojí. Jejich pohybem v online světě bude zátěž na webové aplikace větší než kdy předtím, a proto poptávka po kvalitních nástrojích usnadňujících jejich vývoj naléhavější. Snad tato práce tuto poptávku alespoň částečně uspokojí.

Tato bakalářská práce je příspěvkem do tématu testování kvality webových frameworků. Poskytuje dobrý teoretický základ a prostřednictvím jednoduchého experimentu doplňuje již existující výsledky.

2 Cíl práce a metodika

2.1 Cíl práce

Hlavní cíl práce je srovnat výkon vybraných komponent frameworků pro vývoj webových aplikací v Pythonu Django a Flask. Vedlejší cíle práce jsou:

- Vybrat vhodné komponenty frameworků pro testování
- Naprogramovat komponenty a připravit experiment
- Provést a vyhodnotit měření

2.2 Metodika

V teoretické části práce jsou popsány zásadní koncepty spojené s vývojem a testováním webových aplikací. Dále jsou zde prezentovány současné trendy ve využití frameworků pro vývoj webových aplikací. Vysvětleny jsou i důvody pro volbu v práci testovaných frameworků. Krátké kapitoly jsou věnovány teoretickému popisu fungování nástrojů využitých pro uskutečnění experimentu. Nakonec jsou předloženy výsledky experimentů zaměřených na stejné téma jako tato bakalářská práce.

Praktická část práce je založena na experimentu, jehož cílem je srovnání výkonu vybraných frameworků. Pro účely této práce výkon definuji jako rychlost odezvy aplikace napsané za použití daného frameworku. Výkon je tedy tím lepší, čím nižší je latence.

Experiment se skládá ze tří částí. První části je vývoj předem vybraných, funkcionálně ekvivalentních aplikací na základě komponent z frameworků Django a Flask. V druhé části jsou tyto aplikace spuštěny na lokálním WSGI Gunicorn serveru za využití reverse proxy serveru NGINX. Ve třetí části podrobuji tyto aplikace uměle vytvořené zátěži pomocí programu WRK.

Data sesbíraná programem WRK ukazují průměrnou latenci aplikace a průměrný počet zpracovaných žádostí za vteřinu při stanovené zátěži. Dále jsou k dispozici informace o celkovém počtu zpracovaných http žádostí a směrodatných odchylkách průměrné latence a průměrného počtu zpracovaných zátěží.

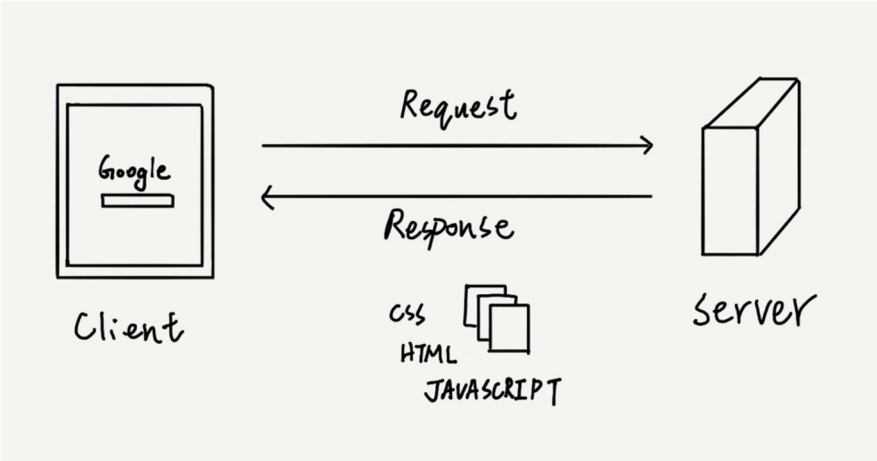
Vyhodnocení je tvořeno srovnáním metrik napočítaných programem WRK. Hlavním kritériem srovnání je latence aplikací, ale pro barvitější popis chování aplikací využívám i dalších dostupných metrik. Poznatky získané studiem zdrojů a experimentem jsou využity

k celkovému shrnutí práce. Je poukázáno na její nedostatky a zároveň navrženy metody pro možné zlepšení.

3 Teoretická východiska

3.1 Webová aplikace

Webová aplikace je program spuštěný ve webovém prohlížeči. (Indeed Editorial Team 2021) Příklady takové aplikace jsou Dropbox, YouTube nebo Google Photos. Základem fungování webové aplikace je princip klienta a serveru, viz obrázek Princip klient-server. Klient, koncový uživatel, pracuje v interakci s internetovou stránkou (front-end) ve svém webovém prohlížeči. Internetová stránka přijímá od klienta požadavky a odesílá je na server (back-end). Server požadavky vyhodnotí a na jejich základě odesílá zpět informace. V případě Google Photos je front-end galerie klientových fotografií zobrazená v prohlížeči a back-end databáze Google ukládající fotky klienta. (Molina-Ríos a Pedreira-Souto 2020; Chi An 2018)



Obrázek 1: Princip klient-server (Chi An 2018)

Webové aplikace dělíme na statické a dynamické, ačkoliv v případě statických je využití pojmu aplikace značně nepřesné. (Molina-Ríos a Pedreira-Souto 2020) Na rozdíl od dynamických webových aplikací nejsou statické webové aplikace v interakci s uživatelem ve smyslu zobrazení personalizovaného obsahu. Většinou se jedná o jednoduché, informativní stránky, zobrazené pomocí značkovacích jazyků HTML a CSS[1]. Jejich obsah i forma jsou jednotně prezentovány všem uživatelům internetu. Poskytují jednoduchý vizuální, zvukový či textový obsah, který uživatel konzumuje, ale nemůže s ním interagovat.

Webové aplikace mají oproti klasickým desktopovým aplikacím množství výhod, a proto jsou mezi vývojáři, ale i uživateli, čím dal tím populárnější.

[1] HTML a CSS nejsou programovacími jazyky ve smyslu C#, Pythonu nebo Javy. Nelze s nimi provádět elementární operace programování, například cykly.

- Nezabírají žádné místo na lokálním disku, protože se nemusí instalovat
- Nevyžadují údržbu ze strany uživatele, jsou aktualizovány centrálně vývojáři
- Jsou levnější, často zadarmo, protože nemusejí být nákladně distribuovány a vývojář peníze vydělá na reklamě
- Všichni uživatelé mají stejnou, stále aktuální verzi
- Jsou přístupné okamžitě z jakéhokoliv zařízení s připojením k internetu a webovým prohlížečem
- Nejsou spjaté s jednou platformou či operačním systémem, pro běh vyžadují pouze kompatibilní webový prohlížeč (Indeed Editorial Team 2021)

Webové aplikace vznikají v desítkách jazyků za pomoci desítky frameworků. Kromě HTML a CSS jako značkovacích jazyků, bez kterých žádná moderní webová aplikace nevznikne, patří mezi oblíbené programovací jazyky JavaScript, Python, PHP nebo C#. Oblíbené frameworky jsou jQuery, React.js nebo ASP.NET.

3.2 Framework

Framework je abstraktní softwarový konstrukt zjednodušující vývoj. Je tvořený systémem abstraktních tříd a tříd, a definuje, jak spolu instance těchto tříd komunikují. Aplikace často využívají několik frameworků. Webové aplikace minimálně dva, jeden pro back-end část, druhý pro front-end část.

Frameworky jsou užitečné pro zjednodušení vývoje v určité oblasti. Vývojář si samozřejmě může aplikaci napsat celou sám, bez využití jakéhokoliv frameworků, v praxi se to ale nestane. Existují pro to minimálně dva dobré důvody.

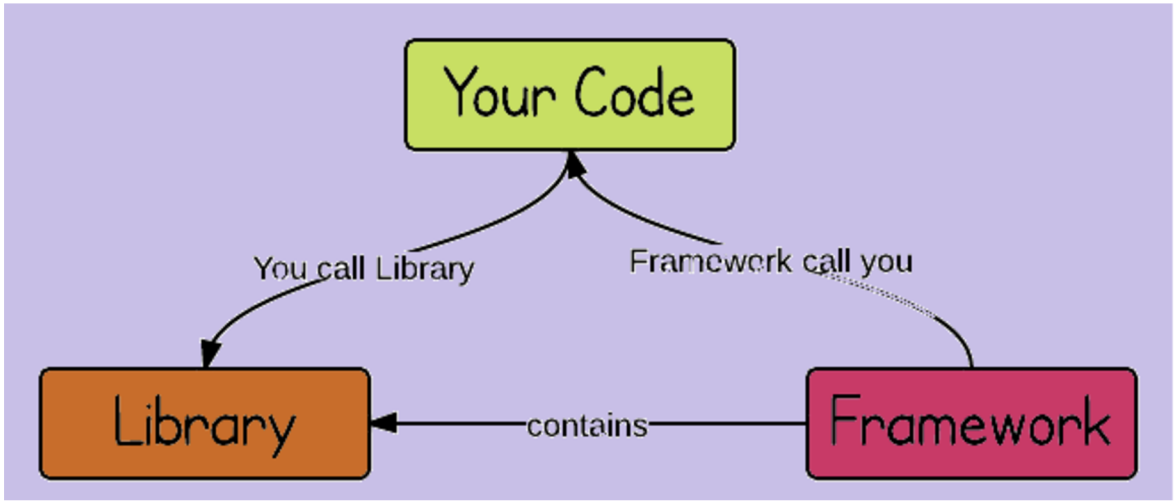
1. Neúměrně zkomplikování vývoje celé aplikace. V důsledku čas strávený vývojem aplikace exponenciálně roste, čímž rostou i náklady
2. Náchylnost k chybovosti. Běžně využívané frameworky jsou naprogramovány v kvalitních softwarových firmách (Microsoft, Google, Apple) a bývají nejen naprosto bezchybné, ale i perfektně optimalizované.

Vývoj s pomocí frameworků nese i několik rizik.

1. Některé objekty tvořené z tříd frameworků se chovají natolik komplexně, že koncept objektového rozhraní nedostatečně zachycuje všechny jejich aspekty.
2. Design frameworků je zaměřený na chování jednotlivých objektů a vztahů mezi nimi, čímž dochází ke ztrátě pozornosti k celkovému cíli.

3. Ačkoliv frameworky zajišťují plynulou komunikaci mezi vlastními třídami, komunikace s třídami definovanými vývojáři může být problematická.
4. Zneužití frameworku pro účely, ke kterým nebyl vytvořen. K tomu často dochází v důsledku nejasně specifikovaného účelu tvůrcem frameworku. Důsledkem jsou nestabilní řešení a bugy v programech (Riehle 2000).

Softwarové frameworky jsou často zaměňovány se softwarovými knihovnami, ačkoliv se jedná o dva odlišné koncepty. Rozdíl pramení z inverze kontroly. Vývojář ve svém kódu volá knihovnu a rozhoduje o tom, kdy a jak bude využitá. Rozhoduje o chodu programu. Frameworky na druhou stranu přijímají vstup a zaručují výstup. Jak ovšem dojde k získání výstupu rozhoduje samotný framework. Graficky rozdíl vztahu mezi knihovnou, frameworkem a kódem aplikace ztvárňuje obrázek Vztah knihovny, frameworku a kódu aplikace. Na něm jsou patrné výše popsané principy, ale zároveň vztah mezi frameworkem a knihovnou. Framework může obsahovat knihovnu, knihovna framework ne (Sofiene 2019).



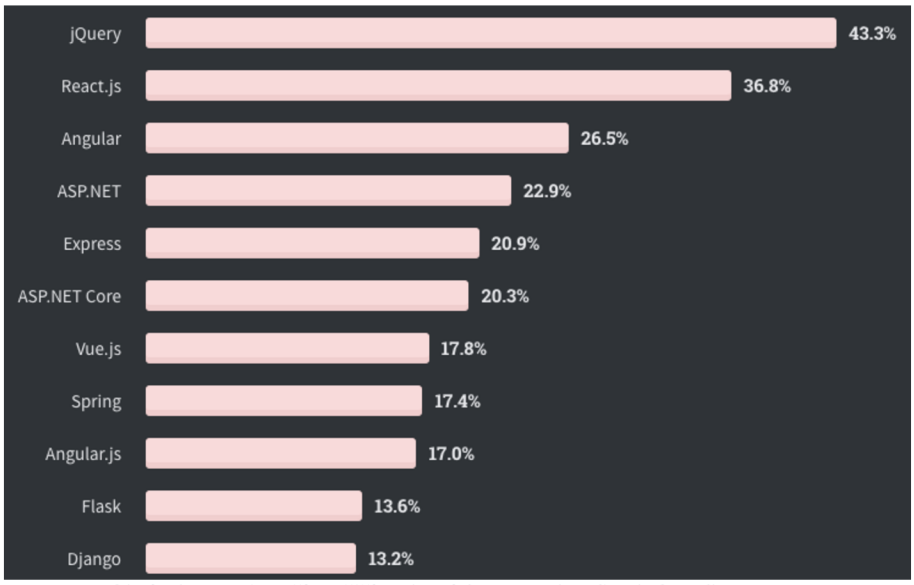
Obrázek 2: Vztah knihovny, frameworku a kódu aplikace (Sofiene 2019)

3.3 Stack Overflow Survey 2020

Webový portál Stack Overflow každoročně provádí rozsáhlý výzkum zaměřený na trendy světa programování. (Stack Overflow 2020) V zatím posledním zveřejněném výzkumu z roku 2020 dotazoval 65 000 vývojářů ze 186 zemí světa. Nejedná se o výzkum reprezentativní ve statistickém slova smyslu, vývojáři se ho účastní dobrovolně na základě žádosti portálu.

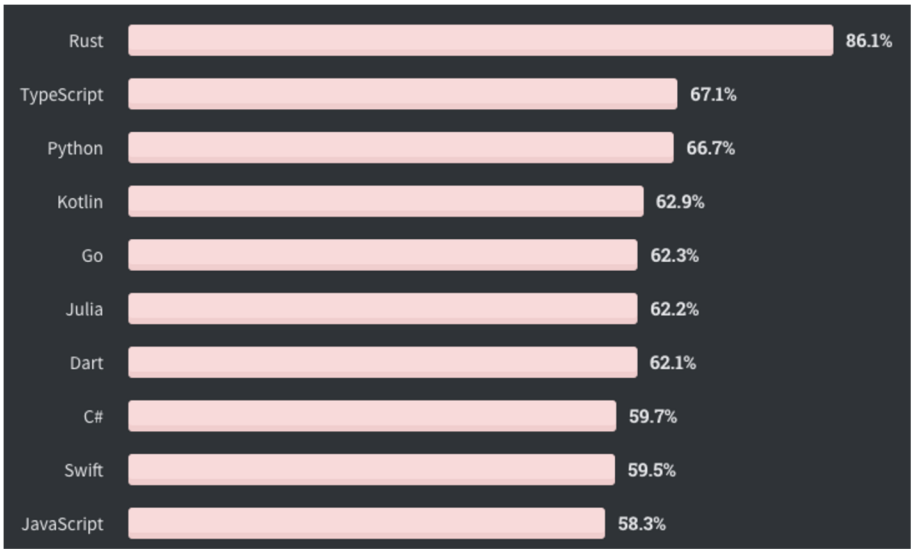
Jedna z otázek cílí na využívání webových frameworků. Z obrázku Nejvyužívanější webové frameworky je patrné, že nejvyužívanější framework je jQuery (43,3 %), druhý

React.js (36,8 %), třetí Angular (26,5 %)[2]. Flask (13,6 %) a Django (13,2 %) se umístily až na 10. a 11. pozici. Většina z frameworků v první desítce slouží pro vývoj v JavaScriptu, Python na první pohled není pro vývoj webových aplikací populární. Obrázek Nejmilovanější vývojové jazyky ukazuje podíl respondentů, kteří vyvíjí s daným jazykem a chtějí v tom pokračovat. JavaScript se umístil až na 10. místě s 58,3 %, zatímco Python drží 3. příčku s 66,7 %. Samozřejmě je nutné zmínit, že 2. místo drží TypeScript, který je nadstavbou JavaScriptu. Obrázek Nejžádanější vývojové jazyky zobrazuje podíl vývojářů, kteří v daném jazyce sice nevyvíjí, ale vyjádřili zájem vyvíjet v něm v budoucnu. Tady je Python jednoznačně nejsilnější ze všech jazyků. Pokud by tento zájem pokračoval i v dalších letech, mohl by být Python populárnější volbou než JavaScript. A z tohoto důvodu se vyplatí zkoumat dostupné Python frameworky pro vývoj webových aplikací, podrobovat je kritice. Ta slouží jejich vývojářům jako zpětná vazba, aby je mohli nadále zlepšovat. Zároveň může dobrá, veřejně přístupná kritika, sloužit potencionálním vývojářům webových aplikací k informovanější volbě správného frameworku pro práci.

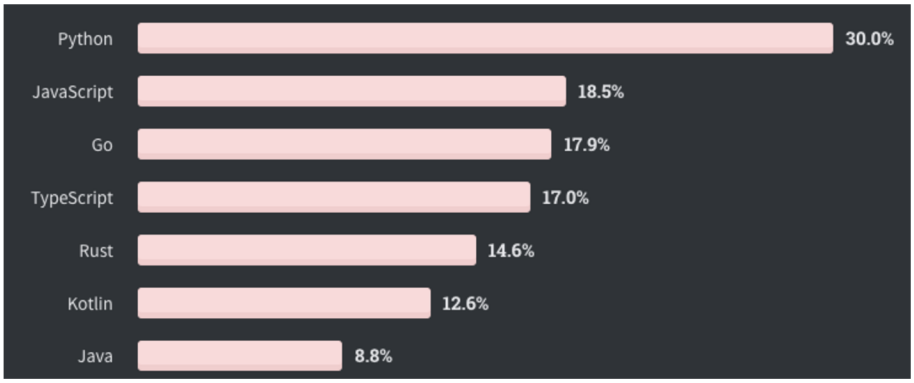


Obrázek 3: Nejvyužívanější webové frameworky (Stack Overflow 2020)

[2] Každý respondent může označit více než jednu knihovnu, proto není součet procent u jednotlivých frameworků roven 100 %.



Obrázek 4: Nejmilovanější vývojové jazyky (Stack Overflow 2020)



Obrázek 5: Nejžádanější vývojové jazyky (Stack Overflow 2020)

3.4 Rychlost webové aplikace

Rychlost webové aplikace do značné míry determinuje její úspěch, proto je nutné ji neustále optimalizovat a kontrolovat. Rychlost má přímí vliv na pozici aplikace v SERPu (search engine result page, stránka s výsledky vyhledávání), zároveň je v silné korelaci s konverzním poměrem[3].

Google[4] chce svým uživatelům zprostředkovat perfektní zkušenost, proto neustále vyvíjí svůj webový vyhledávač, aby na zadaný dotaz vrátil nejen relevantní, ale i optimalizované. Relevantní výsledky jsou v tomto případě stránky odpovídající na dotaz/potřebu uživatele. Jelikož je ale takových stránek několik pro každý dotaz, Google

[3] Konverzní poměr webové stránky je většinou definován jako podíl nějaké metriky a počtu návštěv dané stránky. Pro online obchod bude konverzní poměr definován jako počet nákupů/počet návštěv. Výsledek se udává v % a je jedním z hlavních ukazatelů výkonu dané stránky.
[4] Ačkoliv existuje mnoho webových vyhledávačů, v této práci budu mluvit především o Googlu, protože drží dominantní pozici na trhu a pro většinu webových aplikací je tak zásadní.

mezi relevantními výsledky vybírá ty, které odpovídají kritérium přístupnosti. V tomto případě se jedná o dobře optimalizované stránky, které uživateli nabízí pohodlné užívání.

Na základě potřeby optimalizace stránek vzhledem k vyhledávacím algoritmům Googlu vznikl nový marketingový obor SEO (search engine optimalization). SEO je zaměřeno na akvizici návštěv z neplacených kanálů, takzvaných organických návštěv (přicházejících z vyhledávačů). Účelem SEO týmů je optimalizace webové stránky, aby stoupla na nejvyšší možnou pozici v SEO ranku. SEO rank je ukazatel měřící, nakolik webová stránka vyhovuje vyhledávacímu algoritmu Googlu. Čím je stránka lépe hodnocena, tím výš ji Google v SERPu zobrazuje.

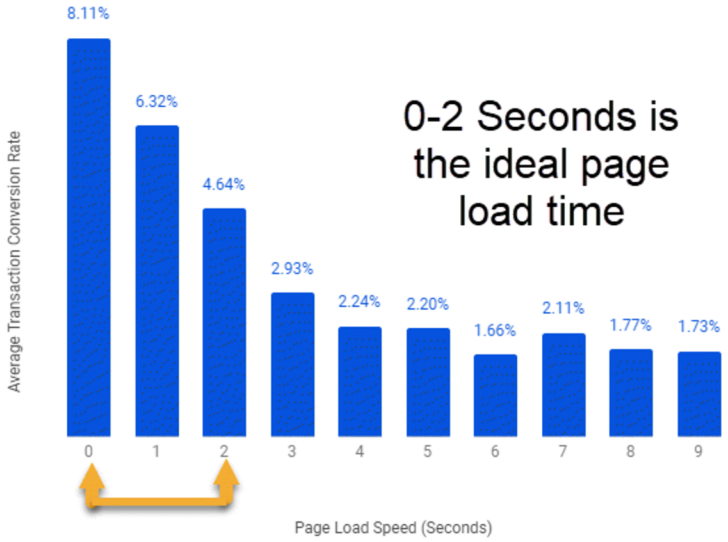
Rychlost webové stránky přímo ovlivňuje pozici webové stránky v SEO ranku. Pokud je příliš nízká, uživatelé z ní odcházejí. V opačném případě jsou spokojení a stránku využívají. Zajímavostí je, že rychlost webové stránky SEO rank ovlivňuje více v případě vyhledávání přes mobilní než desktopové zařízení. Samozřejmě je v zájmu samotného Googlu zobrazovat uživatelům relevantní a rychle stránky. V opačném případě by mohlo dojít k odlivu uživatelů ke konkurenčním vyhledávačům. Za přijatelnou rychlost načtení se obecně považují 3–4 vteřiny.

Rychlost načtení webové stránky neovlivňuje pouze neplacené vyhledávání. Firmy si u Googlu mohou zaplatit PPC (pay per click) reklamu. Ta funguje formou aukce. Čím více je firma ochotna zaplatit Googlu za přivedení uživatele prostřednictvím dobré pozice ve vyhledávání, tím výše se objeví její stránky v placených SERP výsledcích. Minimální aukční cenu, která je posléze násobena hodnotou přihozenou do aukce, určuje takzvané Quality Score. Quality Score vypočítává Google pro každé klíčové slovo, které chceme prostřednictvím reklamy v SERPu zobrazit. Hlavním faktorem určujícím Quality Score slova je CTR (click through rate)[5] odkazu, který pod daným slovem inzerent nabízí. Ihned po CTR Google hodnotí kvalitu odkazované webové stránky, ve které hraje rychlost načtení stránky klíčovou roli (Lafrenz 2018).

Vztah mezi rychlostí načtení webové stránky a konverzním poměrem je velmi silný. Marketingová agentura Portend provedla výzkum 10 webových stránek zaměřených na prodej zboží. Měření stránek probíhalo 30 dní, jednotlivé stránky byly frekventovaně navštěvované (39 000 až 718 000 návštěv), tři stránky byly zaměřený na B2B (business to

[5] Click Through Rate je počítána jako počet prokliků zobrazené URL v SERPu / počet zobrazení URL v SERPu

business) klienty, zbytek na B2C (business to client) klienty. Výsledky měření zobrazuje obrázek Vliv rychlosti načtení webové aplikace na konverzní poměr.



Obrázek 6: Vliv rychlosti načtení webové aplikace na konverzní poměr (Jeffers 2019)

Měřené aplikace s téměř okamžitým načtením měly v průměru konverzní poměr 8,11 %. Zpomalení načtení webové stránky o 1 vteřinu způsobilo pokles konverzního poměru o 2 procentní body. Pokles rychlosti o další vteřinu taktéž. Teprve při rychlosti načtení přes 3 vteřiny lze mluvit o stabilizaci chování uživatelů vzhledem k rychlosti odezvy.

Zdánlivě nízké rozdíly v konverzním poměru mají značný vliv na výdělek. Pokud firma prodává produkt s cenou 300 Kč a stránku navštíví 1000 uživatelů týdně, obrat při konverzním poměru 8 % činí 24 000 Kč, 18 000 Kč při 6 %, 6000 Kč při 2 %. Aby došlo k požadovanému efektu optimalizace na konverzní poměr, musí být zaměřena na podstatné části webové aplikace, mezi které většinou patři domovská stránka, registrace uživatele a průchod nákupním procesem (Jeffers 2019).

Autor práce z vlastní zkušenosti ví, že plně profesionální firmy závislé na online prodeji mají téměř perfektně odladěné aplikace a pouze vzácně dochází k odhalení větších chyb v optimalizaci odezvy. Důsledky optimalizace se většinou počítají v setinách či desetinách vteřin, které jsou i pro pravidelné uživatele těchto aplikací téměř nepostřehnutelné. Proto považuje výše zmíněné rozdíly relevantní spíše pro malé podniky.

3.5 Framework Flask

Flask je microframework zaměřený na vývoj webových aplikací. Jeho výjimečnost spočívá v tom, že dává vývojářům plnou kontrolu nad vývojem jejich aplikací bez zbytečných omezení. Standardní frameworky často nutí vývojáře k využití určité

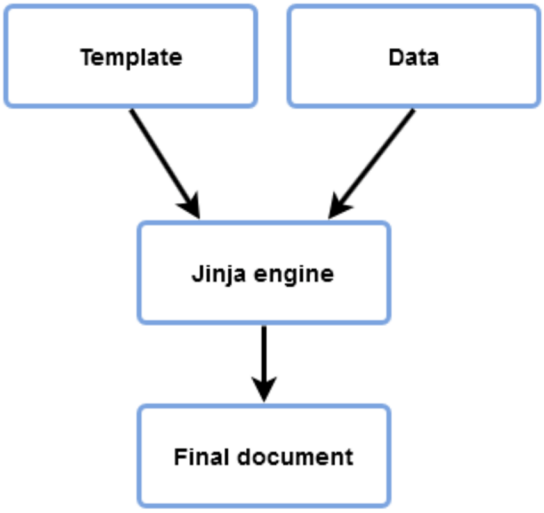
technologie. Například pracují pouze s několika málo databázemi nebo umožňují jeden konkrétní typ autentizace uživatele. Pokud chce vývojář jinou než podporovanou databázi/autentizaci využít, musí framework složitě obcházet. Flask poskytuje pouze nástroje nezbytné pro vývoj jádra webových aplikací a zbytek práce přenechává na třetích stranách. To dává vývojáři velkou svobodu, ale také zpomaluje vývoj v případě chybějících šablon či funkcionalit. Závislost Flasku na vývoji třetích stran představuje také bezpečnostní riziko.

Jádro Flasku tvoří tři závislosti.

1. Směrování (routing), ladění (debugging) a WSGI (web server gateway interface) subsystémy z Werkzeugu
2. Šablony z Jinja2
3. Integrace příkazové řádky pomocí Click (Grinberg 2018).

Werkzeug je sada knihoven, jejichž cílem je vytvoření funkčního WSGI. WSGI umožňuje komunikace mezi Pythonem a webovým server, jelikož servery neumí nativně s Python webovými aplikacemi pracovat. Flask využívá Werkzeug pro zpracování žádostí, zpracování odpovědí, směrování URL, middleware, http služby, zpracování výjimek. (Kennedy 2021)

Jinja je šablonovací systém pro Python umožňující dynamickou tvorbu textových dokumentů. Mezi textové dokumenty se v tomto případě řadí také veškeré dokumenty tvořené značkovacími jazyky, například HTML, JSON či XML. Jinja umožňuje zachytit logiku aplikace v kódu, ale nechat tvůrci šablony nástroje ke kontrole běhu a rozložení v koncovém dokumentu. Mezi nástroje přímo poskytnuté Jinjou patří mimo jiné kontrolní struktury (cykly a podmínky), filtrování, makra, dědičnost šablon. Princip fungování Jinja šablon zachycuje obrázek Princip fungování Jinja. V podstatě jsou potřeba pouze dvě věci, Jinja šablona a data. Jinja engine poté tyto ingredience spojí a vytvoří z nich finální dokument (Przemek 2020).



Obrázek 7: Princip fungování Jinja (Przemek 2020)

Click je Python knihovna pro vytváření aplikací v Command Line Interface (CLI). Pokud uživatel využívá Flask v terminálu, používá vlastně aplikaci naprogramovanou za pomoci knihovny Click. Tato aplikace odkazuje na základní funkce Flasku, rozšíření nebo definované příkazy.

3.6 Framework Django

Django je standardní framework pro vývoj webových aplikací. Vznikl v roce 2005 jako open source projekt a je zdarma. Komunita kolem Djanga je velmi aktivní a pravidelně vydává aktualizace. Django využívají miliony vývojářů po celém světě a s jeho pomocí byly vyvinuty aplikace Instagram, Pinterest nebo Bitbucket. Django nabízí mnoho před programovaných komponent, které lze téměř okamžitě nasadit do jakékoliv aplikace. Mezi tyto komponenty mimo jiné patří autentizace uživatele, šablony, směrování, admin prostředí, dobré zabezpečení, databázové připojení.

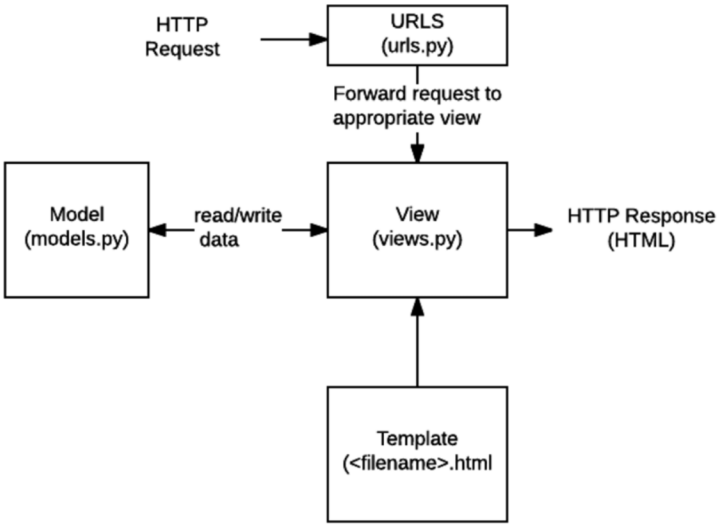
Cílem Django frameworku je maximálně usnadnit vývoj webové aplikace pomocí již zmíněných komponent tak, aby vývojář nemusel ztrácet energii při jejich programování, a naopak se mohl soustředit na logiku samotné aplikace. Výhodou tohoto přístupu není jen rychlost, ale také eliminace chyb (Vincent 2018).

Portál Mozzila Developer vyzdvihuje Django, protože umožňuje budovat software, který je kompletní, všestranný, bezpečný, škálovatelný, udržovatelný a přenosný.

1. Kompletní: Django poskytuje veškeré nástroje a komponenty potřebné pro vývoj webové aplikace. Jelikož jsou všechny komponenty z jednoho frameworku, perfektně spolu komunikují a vychází z jednotného designového principu.

2. Všestranný: Django umožňuje vývoj téměř jakéhokoliv typu webové aplikace, spolupracuje téměř se všemi front-end frameworky a poskytuje výstupy v nepřeberném množství formátů. Django je také rozšiřitelné dalšími komponentami, pokud vývojář postrádá určitou funkcionalitu.
3. Bezpečný: Django komponenty jsou naprosto bezpečné. Aplikaci napsané v Djangu nehrozí typické bezpečnostní chyby, jako ukládání informací o návštěvě do cookie nebo nezahashované uschovávání hesel v databázi.
4. Škálovatelný: Django využívá share-nothing architekturu. Její jádro spočívá v naprostém oddělení jednotlivých částí aplikace. Tento přístup má několik výhod. Za prvé umožňuje vcelku jednoduché nahrazení jednotlivých částí aplikace za jiné, za druhé dovoluje flexibilně reagovat na zvýšenou zátěž a přidávat hardwarové komponenty.
5. Udržovatelný: Django podporuje vývojáře v užívání základních designových principů a vzorů pro tvorbu udržovatelného kódu. Ty spočívají v pravidle „Neopakuj se“. Podpora je dostupná v podobě možnosti sdružovat podobné funkce do aplikací, případě kusy kódu do modulů.
6. Přenosný: Django je napsaný v Pythonu, jedním z populárním a značně rozšířeném programovacím jazyku. Díky tomu je možné jej využívat na všech klíčových platformách (Linux, Mac OS, Windows). Zároveň Django podporují téměř všichni velcí poskytovatelé webhostingu.

Typickou architekturu Django aplikace znázorňuje obrázek Architektura Django aplikace. Příchozí HTTP požadavek rozřadí URL mapper. Ten mimo jiné umožňuje parsování URL na jednotlivé části, z nichž některé vstupují do dalších funkcí. Po rozřazení putuje žádost do View, které ji zpracuje. View mimo jiné požádá v databázi o data potřebná ke správnému zobrazení odpovědi a následně deleguje nashromážděná data do šablony formátující výstupy. Mezi View a databází se nacházejí modely. Ty aplikaci slouží pro manipulování dat v databázi (přidávání, mazání, úprava) a zároveň určují jejich strukturu. Poslední část, šablony, jsem již detailně popsal v kapitole o Flasku (Schonning et al. 2020).



Obrázek 8: Typická architektura Django aplikace (Schonning et al. 2020)

3.7 Testování webových aplikací

Webové aplikace hrají ústřední roli v ekonomice západního světa. Na jejich korektním fungování závisí nejen soukromý business, ale i vládní organizace. Testování webových aplikací vede k optimalizaci jejich výkonu, který se následně promítá do spokojenosti uživatelů.

Jeden z rámců využitelný pro testování webových aplikací se nazývá QoS, což je zkratka vycházející z anglického názvu Quality of Services. Název rámce byl zvolen tak, aby odrážel především pohled uživatelů aplikace, kteří očekávají určitou úroveň kvality využívané služby. Špatná QoS může být příčinou nespokojených zákazníků nebo ztracených obchodních příležitostí.

QoS vychází z dvou základních metrik, dostupnosti a době odezvy. Dostupnost je procentuální vyjádření času, kdy je služba přístupná pro zákazníky. Požadavky na dostupnost aplikace se mohou lišit podle jejího typu. Aplikace pro online obchodování mají striktnější požadavky na dostupnost než malé online obchody. Dostupnost je také podstatně ovlivněna sezónností, popřípadě denní hodinou. Velké online obchody před Vánoci navyšují kapacitu svých serverů, aby dokázaly zpracovat všechny požadavky. Univerzitní servery čelí zvýšené zátěži před otevřením zápisu studentů na zkoušky. Dalším důležitým faktorem ovlivňujícím sezónnost je geografická poloha uživatele aplikace. Z výše uvedeného mimo jiné vyplívá, že testování webové aplikace vyžaduje dobrou znalost jejích uživatelů. Odkud pocházejí a kdy aplikaci využívají.

Doba odezvy se typicky měří jako čas mezi odesláním HTTP žádosti uživatelem a doručením odpovědi uživateli. Pro webovou aplikaci je podstatný čas odezvy z hlediska vnímání uživatele. Ačkoliv je správná optimalizace odezvy webové aplikace základem k jejímu úspěchu, existuje celá řada faktorů, které jsou mimo dosah tvůrce webové aplikace a zároveň odezvu zásadně ovlivňují. Mezi tyto faktory patří internetový poskytovatel webové aplikace, internetový poskytovatel uživatele webové aplikace, kvalita připojení uživatele aplikace, sítě zahrnuté ve směrování paketů mezi uživatelem a aplikací nebo zpoždění způsobená službami třetích stran (Menascé 2002).

Názornou ukázku možného rozpadu doby odezvy zobrazuje obrázek Příklad rozpadu odezvy aplikace. Komunikace mezi serverem a uživatelem začíná při DNS Lookup, kdy dochází ke konverzi URL adresy zadané uživatelem do IP adresy. Druhý krok je nastavení TCP spojení[6]. Následuje stažení prvního paketu, posledního paketu a celkový čas pro stažení dané části stránky. Celý proces od odeslání HTTP žádosti po zobrazení výsledku uživateli v tomto příkladu trvá 4,03 vteřiny.

Odeslání prvního HTML souboru trvá celkem 2,17 vteřiny. Jakmile dojde ke stažení první části odpovědi serveru uživatelem, CDN[7] napomáhá v odeslání obrázků. Odeslání prvního obrázku zabere v tomto případě 0,39 vteřiny, odeslání obrázku n 0,44 vteřiny.

Důležitá je rozdělení času odpovědí jednotlivých částí odezvy na síťové a síťové s dalšími komponenty (například firewall). Mezi pouze síťové komponenty neovlivněné firewallem serveru patří TCP připojení, ostatní komponenty firewall serveru ovlivňuje. Komponenty ovlivněné firewallem se mohou chovat různě při rozdílné zátěži požadavků uživatelů na server.

[6] Transportní vrstva TCP/IP je využívaná internetem pro obousměrný přenos dat
[7] Content Delivery Network zvyšuje dostupnost dat uživatelům

		Starting Time	Outside the Firewall			Inside the Firewall				Completion Time
			Network	External Server	Total Outside	Web Server	Application Server	Database Server	Total Inside	
First HTML File	DNS Lookup		0.01	0.02	0.03				0	0.03
	Initial TCP Connection		0.08		0.08	0			0	0.08
	First Packet Download		0.08		0.08	0.02	0.06	1.4	1.48	1.56
	Last Packet		0.4		0.4	0.1			0.1	0.5
	TOTAL		*0.57*	*0.02*	*0.59*	*0.12*	*0.06*	*1.4*	*1.58*	*2.17*
Image 1	DNS Lookup	2.2	0		0				0	2.2
	Initial TCP Connection	2.2	0.08		0.08	0			0	2.28
	First Packet Download	2.2	0.08		0.08	0.02			0.02	2.3
	Last Packet	2.2	0.15		0.15	0.06			0.06	2.41
	TOTAL	2.2	*0.31*	*0*	*0.31*	*0.08*	*0*	*0*	*0.08*	*2.59*
Image *n*	DNS Lookup	3.59	0		0				0	3.59
	Initial TCP Connection	3.59	0.08		0.08	0			0	3.67
	First Packet Download	3.59	0.08		0.08	0.02			0.02	3.69
	Last Packet	3.59	0.2		0.2	0.06			0.06	3.85
	TOTAL	3.59	*0.36*	*0*	*0.36*	*0.08*	*0*	*0*	*0.08*	*4.03*
External CDN Server	DNS Lookup	2.5	0.01	0.03	0.04				0	2.54
	Initial TCP Connection	2.5	0.03	0	0.03				0	2.53
	First Packet Download	2.5	0.04	0.03	0.07				0	2.57
	Last Packet	2.5	0.05	0.03	0.08				0	2.58
	TOTAL	2.5	*0.13*	*0.09*	*0.22*	*0*	*0*	*0*	*0*	*2.72*

Obrázek 9: Příklad rozpadu odezvy aplikace (Menascé 2002)

Zajištění dobré QoS lze dosáhnout pomocí srovnávání, testování a výkonnostního managementu aplikace. Srovnání se provádí na webové aplikaci a skládá se ze tří částí. První část je specifikace zátěže, kterou testujeme aplikaci. Lze využít kontrolované prostředí nebo generovat požadavky v reálném prostředí. Druhá část je specifikace metrik, které při zátěži chceme měřit. Typicky se měří rychlost odezvy, ale lze využít i jiné metriky. Třetí část je způsob vyhodnocení naměřených dat. Odezvu naměřenou z různých lokalit můžeme zprůměrovat nebo využít medián či roztptyl.

Testování je metoda, při které dochází ke spuštění sady skriptů simulujících chování uživatelů aplikace při různých úrovních zátěže a následném měření výkonu aplikace v daných úrovních. Jedno z možných využití testování je zajištění optimálního běhu aplikace při předpokládané zvýšené zátěži. Internetové srovnávače cen typu Heureka.cz nebo Zboží.cz před Vánoci navštěvuje podstatně víc uživatelů ve srovnání se zbytkem roku. Pokud by aplikace nebyly otestované a připravené na zvýšenou zátěž, mohlo by dojít k jejich zhroucení a následné ztrátě zisku (navíc dochází k odlivu uživatelů na stabilnější konkurenci zhroucené aplikace). Druhé typické využití testování je při nasazení nových funkcionalit do aplikace. Pokud vývojáři aplikace chtějí zvýšit její profitabilitu, mohou do ní nasadit bannery

s reklamou. Zobrazení reklamy na stránce aplikace může být výkonově náročné, obzvláště pro starší počítače. Proto je před samotným nasazením bannerů do produkčního prostředí aplikace nutné zjistit testem rozdíl v odezvě aplikace před nasazením bannerů a po nasazení bannerů.

Testování aplikace často probíhá v podstatě jednoduchým způsobem, kdy vývojáři vytvoří zátěž na aplikaci v podobě sady HTTP žádostí a měří rychlost odezvy aplikace. V takovém případě nelze mluvit o simulaci průchodu stránky uživatelem. Jde spíše o měření nástrojů využitých k tvorbě aplikace. Zároveň lze nalézt implementační chyby a nevhodně optimalizovaný kód.

Komplexnější testování aplikace vychází z předem připravených scénářů chování uživatelů. Scénáře jsou vytvořeny tak, aby odrážely typické průchody aplikací. Důležité je zohlednit možnost odchodu uživatele z aplikace v průběhu návštěvy, aniž by došlo k dosažení cíle návštěvy (cíle návštěvy jsou většinou definovány podstatou aplikace, např. pro internetové prodejce je typickým cílem prodej zboží). Uživatel, který z aplikace odejde v průběhu své návštěvy, aniž by dosáhl cíle, zabírá méně výpočetního času aplikace než uživatel, který aplikací projde až do konce, což může v důsledku ovlivnit odezvu aplikace. V případě komplexních testů aplikací se v rámci různých scénářů průchodu aplikací dále náhodně mění předem specifikované proměnné, mezi které patří počet návštěv započatých za hodinu, využití různých částí aplikace vytvářejících různou zátěž na aplikaci (definováno ve scénáři průchodu aplikací), další proměnné spojené se samotným uživatelem jako čas mezi jednotlivými kroky v průchodu aplikací nebo odchod z aplikace v důsledku příliš dlouhého čekání na odpověď serveru. Při komplexnějším testu lze jeho výsledek hodnotit nejen rychlostí odezvy aplikace jako v případě jednoduché testu, ale i dalšími metrikami, např. počty návštěv ukončených v průběhu průchodu aplikací nebo počtem návštěv dosahujících cíle.

Výkonnostní management aplikace spočívá v monitoringu aplikace. Může být reaktivní nebo proaktivní. V reaktivním pro daný účel určená aplikace sleduje chování programu prostřednictvím sestavy předem definovaných metrik a v případě problému alarmuje vývojáře, kteří problém opraví. V proaktivním přístupu vývojáři pravidelně udržují aplikaci skrze procesy redukující chyby způsobující kolísání odezvy a dostupnosti aplikace (Menascé 2002).

3.8 Obdobná řešení

Hodnocení frameworků se dá obecně rozlišit na kvantitativní a kvalitativní. Kvalitativní hodnocení je zaměřené na podstatu frameworku. Popisuje jeho silné a slabé stránky a podává výstupy v podobě slovních hodnocení. Dále se snaží se zachytit typické případy využití frameworku. Jednou z možných metod je popis silných a slabých stránek frameworku relativně k frameworkům zaměřených na stejnou oblast využití[8].

Kvantitativní hodnocení frameworků je založeno na empirickém měření výkonu frameworků. Výsledkem je srovnání naměřených hodnot. Při kvantitativním testování je podstatné explicitně definovat metodu a nástroje využité ke srovnání. Autoři testu většinou publikují i zdrojový kód testu, aby každý mohl výsledky snadno ověřit prostřednictvím replikace. Nadále se budu zabývat pouze kvantitativním hodnocením frameworků, protože kvalitativní hodnocení není cílem této práce.

3.8.1 Srovnání frameworků Tech Empower

Rozsáhlé srovnání webových frameworků pravidelně publikuje Tech Empower (Tech Empower 2021). Poslední srovnání proběhlo 8.2.2021. Testem prošlo 436 webových frameworků napsaných pro vývoj v několika různých jazycích.

Testování frameworků se skládá ze sedmi částí, každá cílí na jinou funkci.

1. Serializace JSON souborů: přesměrování HTTP žádostí, parsování hlavičky, serializace JSON souborů
2. Jeden databázový požadavek: výkon ORM (object-relational mapper), generátor náhodných čísel, připojení k databázi.
3. Několik databázových požadavků: variace jednoho databázového požadavku.
4. Fortunes: test ORM, databázového připojení, řazení, serverových šablon, šifrování charakterů
5. Aktualizace databáze: aktualizace databázového záznamu
6. Prostý text: elementární přesměrování žádostí
7. Mezipaměť: ukládání informací z databáze do mezipaměti

Princip testů spočívá v generování HTTP žádostí a následném měření rychlosti odezvy. Pro generování HTTP žádostí využívají autoři testu nástroj WRK, kterému se budu podrobněji věnovat později. Aplikace postavené na testovaných frameworcích byly spuštěné

[8] Typickým příkladem kvalitativního hodnocení frameworku je článek od Jigar Mistry A Detailed Comparison of 10 Popular Web Development Frameworks (Mistry 2021)

v Amazon EC2, které odpovídá reálnému produkčnímu prostředí webových aplikací. Zároveň umožňuje každému zopakovat test v totožných podmínkách. Kromě Amazon EC2 proběhly testy i na lokálním počítači s procesorem Intel Core i7 (Sandy Bridge) a operačním systémem Ubuntu. Testy proběhly v cloudu i na lokálním hardware, aby byla potvrzena jejich nezávislost na prostředí. Všechny testy také musely splnit následujících 8 kritérií.

1. Kód testu musí být napsán tak, aby byl připraven pro produkční prostředí. Kontrola naplnění požadavku probíhá přes sdílený kód v GitHubu.
2. Testují se pouze produkčně způsobilé, reálně využívané frameworky, které mají robustně implementovaný http protokol.
3. Testy musí mít vypnuté zapisování logů na disk. Počet žádostí generovaných přes WRK je tak vysoký, že by mohl disk snadno zaplnit.
4. Dodržovaní konvencí vzhledem k pojmenování proměnných či záznamů v JSON souborech.
5. Všechny testy musí obsahovat server a datum v hlavičce HTTP odpovědi.
6. Přesměrování žádostí je vyžadováno.
7. Všechny databázové dotazy musí do databáze přijít přesně tak, jak jsou napsané.
8. Všechny žádosti musí proběhnout na portu tvořeném čtyřmi číslicemi.

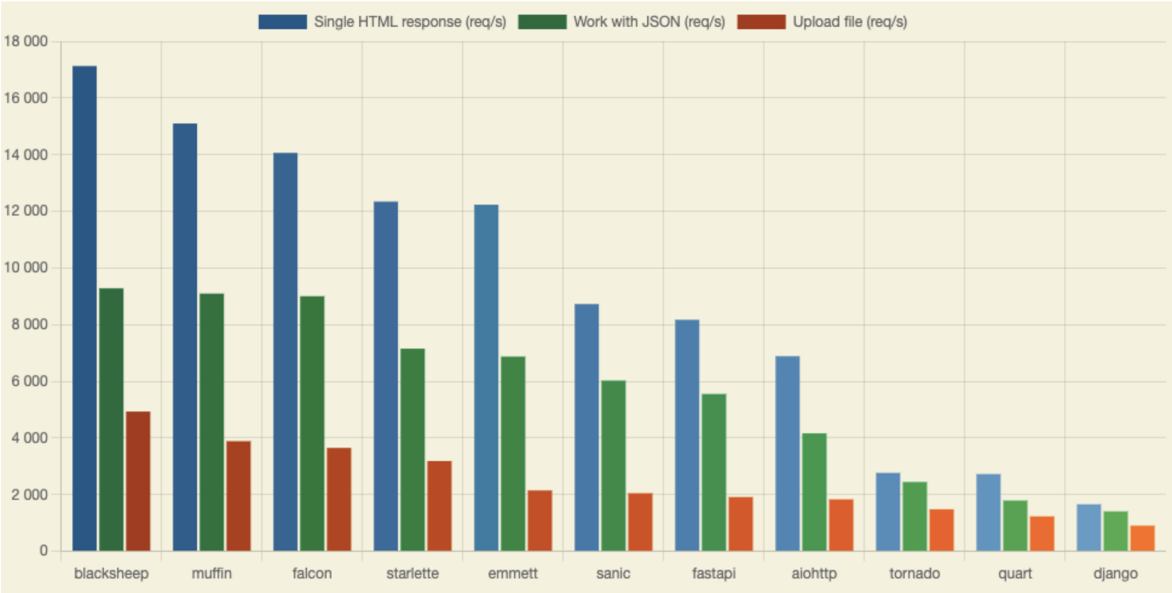
Celý projekt Tech Empower neustále roste a vyzývá vývojáře k doplnění dalších frameworků ke srovnání. K tomu stačí napsat testy odpovídající požadavkům Tech Empower a vytvořit pull request na GitHub. Autoři projektu taktéž plánují přidání dalších testů.

3.8.2 Test Python frameworků pro vývoj webových aplikací

Pravidelný test 11 asynchronních frameworků pro vývoj webových aplikací v Pythonu provádí Kirill Klenov. Test běží jako GitHub action. Hardware využitý pro běh testu je dvou jádrové CPU Intel Xeon, 7GB RAM, 14GB SSD disk a OS Ubuntu 20.04. Softwarovou základnu tvoří Docker, Gunicorn a WRK. Test se skládá ze tří částí.

1. Přijetí HTTP žádosti a navrácení HTML odpovědi s dynamickou hlavičkou. Účelem je simulace jednoduché HTML odpovědi.
2. Kontrola hlavičky, parsování parametrů, přiřazení hodnoty parametru, navrácení JSON odpovědi. Účelem je simulace JSON REST API.
3. Přijetí a uložení souboru na disk. Test simuluje procesování formulářových dat a práci se soubory (Klenov 2016).

Výsledky testu jsou na obrázku Srovnání Python frameworků pro vývoj webových aplikací. Autor testu výsledky prezentuje ve formě počtu zvládnutých požadavků za vteřinu. Nejrychlejší Python framework pro vývoj webových aplikací je Blacksheep, který v případě prvního testu dokáže odpovědět na 17 000 požadavků na vteřinu, 9 000 v případě druhého testu a 4500 požadavků u třetího testu. Ze všech 11 srovnávaných webových frameworků skončil Django na posledním místě (Flask nebyl měřen). Proč je Django i přes svoji nízkou rychlost stále tak populární? Podle mého názoru jsou klíčové tři faktory. První je široká uživatelská základna, která ostatním Django vývojářům ochotně pomůže vyřešit případné problémy. Druhý je vyspělost frameworku. Ten umí elegantně vyřešit téměř každý požadavek na webovou aplikaci. Zároveň obsahuje minimální množství chyb, protože všechny byly v minulosti opraveny. Třetí je autorita frameworku ve vývojářské komunitě. Vývojářům začínajícím s vývojem webových aplikací v Pythonu pravděpodobně dříve či později Django někdo doporučí, zatímco BlackSheep je prozatím poměrně neznámý.



Obrázek 10: Srovnání Python frameworků pro vývoj webových aplikací

3.9 Serverové technologie Gunicorn WSGI, NGINX, měření ve WRK

Kvantitativní testování webových frameworků je závislé na dvou nástrojích. První je server pro běh aplikace, druhý generuje HTTP žádosti a měří rychlost odezvy. V předešlé kapitole Obdobná řešení využívaly autoři obou testů pro generování HTTP žádostí a měření rychlosti odezvy nástroj WRK. Servery pro hostování aplikací využívaly různé. Mnou napsané testovací aplikace běží na lokálním WSGI Gunicorn serveru, žádosti ke Gunicornu

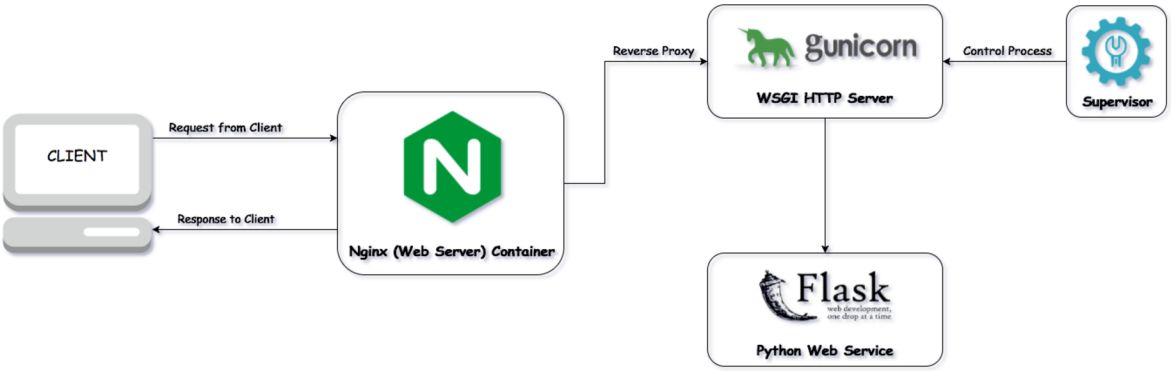
zprostředkovává webový proxy server NGINX. Pro generování HTTP žádostí a měření rychlosti latence využívám WRK.

3.9.1 Webový server Gunicorn WSGI

WSGI (Web Server Gateway Interface) je nástroj umožňující komunikaci mezi Python webovými aplikacemi a webovými servery. Jedná se ve své podstatě o middleware, který směřuje HTTP žádosti do různých částí aplikace, balancuje zátěž a umožňuje běh několika různých aplikací či frameworků zároveň. Využití WSGI serveru má tři výhody. Za prvé umožňuje zpracovat velké množství žádostí současně, za druhé zrychluje vývoj webové aplikace, protože je potřeba znát pouze základy WSGI, za třetí dovoluje flexibilně měnit různé komponenty pro vývoj webové aplikace, aniž by se musela měnit aplikace samotná (Goldberg 2016).

Funkci WSGI HTTP serverů vzhledem k cestě HTTP žádosti od klienta k webové aplikaci demonstruje obrázek Princip WSGI HTTP Serverů. Uživatel zařízení odešle HTTP žádost. Tu zpracuje webový server sloužící jako reverse proxy, a následně předá WSGI serveru, který umožňuje komunikaci mezi webovým serverem a Python webovou aplikací.

Gunicorn (Green Unicorn) je Python WSGI HTTP server. Výhodou Gunicornu je široká kompatibilita s různými webovými frameworky. Je také poměrně snadno implementovatelný a poměrně rychlý.



Obrázek 11: Princip WSGI HTTP Serverů (Nayak 2018)

3.9.2 Nástroj WRK

WRK je nástroj pro generování zátěže na webové aplikace. Výhodou WRK je možnost nastavit parametry testu, například počet otevřených HTTP spojení, doba běhu testu, počet využívaných jader, skriptování nebo přidání HTTP hlavičky.

3.9.3 Proxy server NGINX

NGINX je webový proxy server. Jeho účel spočívá v zrychlení odezvy webových stránek při vysokém zatížení. Toho dosahuje pomocí „event driven“ asynchronní architektury (DevopsCurry 2021). V případě využití NGINX společně s Gunicornem slouží NGINX webový server nejen pro již zmíněné zrychlení odezvy, ale zároveň pro hostování statických souborů (.http, .css).

3.10 Shrnutí

Webová aplikace je program spuštěný ve webovém prohlížeči. Základem její architektury je rozdělení na uživatelskou (front-end) a serverovou část (back-end). Uživatelská část přijímá vstupy od uživatele a zobrazuje výstupy serveru. Serverová část ukládá uživatelská data, připravuje výstupy a provádí veškeré potřebné výpočty. Webové aplikace se dělí na statické a dynamické. Statické nepřijímají vstupy uživatele a pouze zobrazují předem definovaný obsah. Dynamické jsou s uživatelem v interakci. Mezi oblíbené programovací jazyky pro vývoj webových aplikací se podle Stack Overflow (Stack Overflow 2020) řadí JavaScript, PHP, Python, C#.

Webové aplikace jsou vyvíjený pomocí frameworků. Framework je softwarový konstrukt tvořený systémem abstraktních tříd a tříd. Framework definuje, jak spolu instance těchto tříd komunikují. Webovou aplikaci většinou tvoří několik frameworků.

Stack Overflow každoročně provádí rozsáhlý výzkum mapující současné trendy v programování. Součástí výzkumu je měření popularity frameworků a jazyků pro vývoj webových aplikací. JavaScriptové frameworky jsou nejvyužívanější, nicméně Python je nejžádanější vývojový jazyk, proto lze předpokládat, že v budoucnu jeho frameworky v žebříčku využívanosti obsadí vyšší pozice. (Stack Overflow 2020)

Podstatnou charakteristikou webové aplikace je rychlost odezvy. Ta má přímí vliv na umístění aplikace ve výsledcích vyhledávače Googlu a koreluje s konverzním poměrem. Rychlost také ovlivňuje cenu reklamy v případě zobrazení na stránce výsledků vyhledávání již zmiňovaného vyhledávače. Obecně přijatelná rychlost odezvy je nižší než 4 vteřiny.

Nejpopulárnější frameworky pro vývoj webových aplikací v Pythonu jsou Flask a Django. (Stack Overflow 2020) Flask je microframework, jeho přednost spočívá v předání plné kontroly nad vývojem aplikace bez zbytečných omezení pro vývojáře. Flask poskytuje nástroje nezbytné pro vývoj jádra webové aplikace, zbytek nechává na třetích stranách. Django je plnohodnotný framework poskytující vývojáři veškerý potřebný servis pro vývoj

plnohodnotné webové aplikace. Cílem Django frameworku je zredukovat čas strávený programováním často využívaných komponent a umožnit tak vývojáři soustředit se na logiku samotné aplikace.

Testování webových aplikací je klíčové pro jejich optimalizaci. Jeden z rámců využitelných pro testování webových aplikací se nazývá Quality of Services. Ten zakládá na měření dostupnosti a doby odezvy. Zajištění Quality of Services umožňují procedury srovnání, testování a výkonnostního managementu aplikace.

Portál Tech Empower pravidelně testuje většinu dostupných webových frameworků. V testu měří dobu odezvy frameworků při různých procedurách, například rychlosti zpracování JSON souborů. Webové aplikace připravené pro testování hostuje server Amazon EC2. Dobu odezvy měří nástroj WRK, který na aplikace vytváří umělé HTTP žádosti. Aplikace připravené k testování musí být naprogramovány pomocí přesně daných pokynů.

4 Vlastní práce

V teoretické části jsem popsal klíčové koncepty a procesy související s vývojem a testováním webových aplikací. Tyto koncepty a procesy jsem využil pro sestavení vlastního experimentu, který se skládal z několik kroků.

Nejprve jsem hledal vhodné komponenty ke srovnání. Po jejich nalezení jsem naprogramoval jednoduché aplikace využívající dané komponenty pomocí principů z knih Learning Python (Lutz 2013) a Clean Code (Martin 2008). Po naprogramování aplikací jsem vytvořil Gunicorn WSGI server pro hostování daných aplikací a postavil před něj NGINX proxy server. Nakonec jsem na aplikace pustil zátěžový test prostřednictvím nástroje WRK a zaznamenal výsledky řešení. V následujících podkapitolách detailně popíšu jednotlivé kroky mnou provedeného experimentu.

4.1 Nastavení serverů pro experiment

Pro běh experimentu je v základní formě možné využít vývojové servery Flasku a Django. Vývojáři Flasku i Django ovšem od takové možnosti na svých webových stránkách odrazují, protože servery nejsou určené pro produkční řešení. Vtipně to shrnuje věta na webových stránkách Django „We're in the business of making Web frameworks, not Web servers". (Django nedatováno) Stejně tak varuje Flask, že vývojový server nebyl naprogramován s ohledem na stabilitu, efektivitu a bezpečí.

Samozřejmě lze namítnout, že můj experiment nevyžaduje bezpečnost, protože nepřenáší citlivá data. Otázky stability a efektivity jsou ovšem zásadní. Ačkoliv by šlo spustit zátěžové testy na vývojových serverech, výsledek samotných testů by byl ovlivněn schopnostmi vývojových serverů, na kterých jsou aplikace spuštěny. Servery Djangu a Flasku jsou naprosto odlišné, proto se mohou chovat různě a zásadně zkreslovat výsledky testů.

Kromě problematiky schopností vývojových serverů je také vzít v úvahu cíl samotného experimentu. Mým cílem bylo v rámci časových a programátorských schopností maximálně přiblížit běh aplikací během spuštění zátěžového testu produkčnímu prostředí, tedy takovému prostředí, ve kterém mohou být aplikace provozovány pro ostatní uživatele. Z těchto důvodů není možné využívat vývojových serverů, protože ty v produkci nikdy využívány nejsou.

Pro zapojení Python aplikace do produkce se standardně využívají dva servery. Jeden slouží k běhu samotné aplikace, protože tradiční servery neumí pracovat s Python aplikacemi. Druhý, takzvaný proxy server, pomáhá WSGI serveru při zvýšené zátěži (více v teoretické části). Jako WSGI server jsem si zvolil Gunicorn, ačkoliv jsou na trhu další řešení, třeba Waitress nebo uWSGI. Pro moji volbu neexistuje žádný specifický důvod. Snad jen pořadí Gunicornu v seznamu doporučených WSGI serverů na webu Flasku a Djanga.

4.1.1 Webový server Gunicorn

Gunicorn lze jednoduše nainstalovat prostřednictvím pip (konzolová aplikace pro správu a instalaci Python aplikací, knihoven apod.). Celý proces spuštění aplikace na Gunicorn serveru je intuitivní a nevyžaduje specifické znalosti fungování serverů. Pro spuštění aplikací Django a Flask na Gunicornu stačí využít jednoduchý příkaz zadaný pomocí Bashe. Pro Flask je příkaz *gunicorn --workers=4 wsgi:app*, pro Django *gunicorn --workers=4 config.wsgi*. Oba příkazy jsou velmi podobné, jediný nastavený parametr je počet workers. Doporučené nastavení jsou 2-4 workers pro každé jádro v procesoru. Aplikaci spouštím na svém na laptopu se dvěma jádry, proto jsem zvolil počet workers = 4.

Fungování Gunicorn serveru lze optimalizovat pomocí pokročilejšího nastavení, například specifikací workers. Vzhledem k mým značně omezeným znalostem v oblasti fungování serverů jsem se o to nepokoušel. Přesto nepochybuji o možnosti běh aplikace vylepšit. Cílem této práce ovšem není maximalizovat responzivitu a rychlost serveru, proto jsme tím ani netrávil nadbytečný čas.

Aplikace spuštěná na Gunicorn serveru se chová stejně jako aplikace spuštěná na vývojovém serveru. Při zadání správné URL dynamicky vykresluje text do šablony a zapisuje do databáze. Stejně tak je možné otestovat její rychlost pomocí nástroje WRK.

4.1.2 Proxy server NGINX

Proxy server pro můj test je NGINX. Stejně jako v případě výběru WSGI serveru je volba NGINX dána doporučením na stránkách Flasku, Django podle mých znalostí žádný určitý proxy server nedoporučuje.

Instalace NGINX je v případě využití operačního systému založeného na Unixu pohodlná prostřednictvím nástroje na správu aplikací Homebrew. Po nainstalování lze NGINX ovládat sadou příkazů v příkazové řádce. Mezi základní patří *sudo systemctl stop*

nginx pro stopnutí NGINX, *sudo systemctl start nginx* pro zapnutí NGINX a *sudo systemctl reload nginx* pro restartování NGINX.

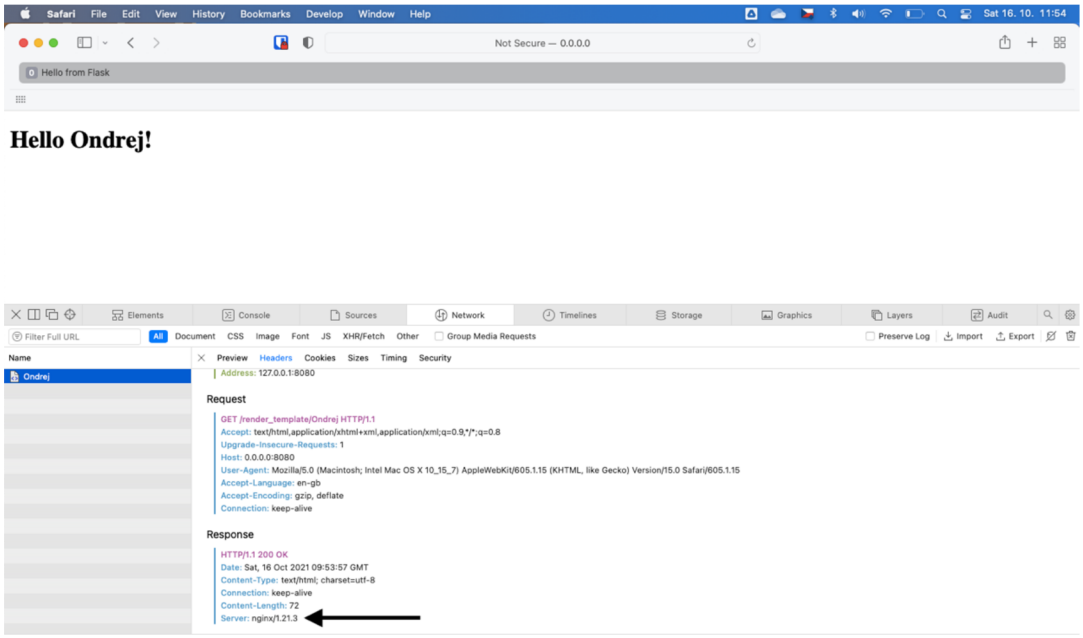
Zásadním momentem práce s NGINX je jeho konfigurace prostřednictvím konfiguračního souboru nginx.conf, který se nachází na adrese */usr/local/nginx/conf*. Nastavení je nutné provést pro minimálně tři základní parametry. Prvním a druhý parametrem jsou port a název serveru, na kterém má NGINX poslouchat. Jedná se tedy o URL adresu, kterou uživatel aplikace zadává do prohlížeče. Třetím parametrem je adresa WSGI serveru, na kterou má NGINX žádosti přeposílat.

Mnou zvolené nastavení NGINX serveru je na základní úrovni. Přesto je dobré zdůraznit, že nastavení NGINX lze upravovat a optimalizovat. V případě testování většího množství aplikací v jeden moment umožňuje NGINX vytvoření více konfiguračních souborů, každého pro jednu aplikaci, které poslouchají na různých portech a vrací různé odezvy.

V případě úspěšného nastavení NGINX lze jeho fungování ověřit prostřednictvím několika jednoduchých kroků. Prvním krokem je kontrola zapnutí samotného NGINX. Pokud je zapnutý NGINX a vypnutý Gunicorn server (+ testovací servery aplikací), vrací NGINX při návštěvě URL zvolené v konfiguraci speciální chybovou hlášku, viz Chybová hláška NGINX serveru při přesměrování na neexistují server.

Obrázek 12: Chybová hláška NGINX serveru při přesměrování na neexistují server

Pokud se nespokojíme s daným potvrzením o funkčnosti NGINX serveru, lze ve webovém prohlížeči prostřednictvím vývojářských nastavení najít zdroj zobrazené odpovědi. Pokud je vše v pořádku, zdroj odpovědi je NGINX server, viz obrázek Zdroj odpovědi ve webovém prohlížeči.



Obrázek 13: Zdroj odpovědi ve webovém prohlížeči

4.2 Nástroj WRK

Pro spuštění testu zátěže na jednotlivé komponenty využívám nástroj WRK. Stejně jako v případě NGINX je pohodlná možnost instalace WRK prostřednictvím unixového správce aplikací Brew.

Wrk je z uživatelského hlediska velmi prostý a nabízí k nastavení pouze pět parametrů. První parametr je počet spojení otevřených na každé využité vlákno. Druhý parametr je počet využitých vláken. Třetí parametr je doba trvání testu. Čtvrtý parametr je testovaná URL adresa. Tyto parametry by měly být nastaveny pro každý test. Wrk obsahuje ještě dva nepovinné parametry. První nepovinný parametry je hlavička pro vkládání hlavičky do http žádostí. Druhý nepovinný parametr je skript, který umožňuje nastavení spuštění více sady testů, aniž by vývojář musel dělat manuální změny.

Pro svůj test jsem využil dvou různých příkazů. Pro test vykreslení dynamického obsahu do šablony příkaz *wrk -t2 -c40 -d10s http://0.0.0.8080/render_template/Ondrej*, pro test zapisování do databáze *wrk -t2 -c40 -d10s http://0.0.0.8080/write_to_database*. Test tedy pracuje s dvěma vlákny, na každém vlákně 40 připojení a test běží po dobu 10 sekund.

Při testování jsem narazil na problém v případě vytvoření příliš velké zátěže na komponentu zapisující do databáze. Tento problém se objevil u Djanga i Flasku, protože nebyl spojený s jednotlivými frameworky, ale ve schopnosti SQLite rychle zapisovat data. Nutno dodat, že se nejedná o chybu, SQLite je naprogramovaný pro zpracování dat na stránkách s malou až středně velkou návštěvností, případně se využívá jako databázový

systém v mobilní aplikacích. Mimochodem je SQLite často doporučována jako nejlepší databázový systém pro iOS a Android vývojáře, protože klade důraz na efektivitu, spolehlivost, nezávislost a jednoduchost. (SQLite nedatováno) Navíc je k dispozici zdarma.

4.3 Hardware

Testování proběhlo na lokálním serveru vytvořeném na Macbook Air (13-inch, Early 2014). Macbook má k dispozici dvoujádrový procesor Intel Core i5 Dual Core, 1,4 GHz, 4 GB 1600 MHz DDR3 paměti RAM, integrovanou grafickou kartu Intel HD Graphics 5000 1536 MB. Operační systém MacBooku je macOS Big Sur, verze 11.6.

4.4 Výběr komponent ke srovnání

Pro srovnání výkonu frameworků Django a Flask jsem zvolil dvě komponenty. Počet komponent může být i vyšší, dvě jsem zvolil z důvodu omezení dané rozsahem bakalářské práce a časovými možnostmi k správnému naprogramování testů.

Komponenty samotné jsem vybral na základě dvou měřítek. Prvním je frekvence jejich využití ve webových aplikacích. Jedná se tedy o kritérium důležitosti, nebo vlivu na celkový chod aplikace. Čím využívanější je komponenta ve webových aplikacích a její rychlost pomalejší vzhledem k funkčně obdobné komponentě v jiném frameworku, tím je její vliv na celkový běh aplikace vyšší a ochota vývojáře využít příslušnou náhradu stoupá. Výběr komponent na základě prvního měřítka není podložen teorií či dalším měřením, ale zkušeností autora. Druhým je implementace daných komponent v měřených frameworcích. Pokud jsou komponenty implementovány stejným způsobem, nemá smysl hledat mezi nimi rozdíly. Každý framework měřených komponent musí využívat rozdílné řešení daného problému.

4.5 Šablony pro vykreslení dynamického obsahu

První komponentou měřenou v experimentu je šablonový systém pro zobrazení dynamického obsahu webové aplikaci, tzv. template systems. Vzhledem k měřítkům popsaným výše se jedná o ideální komponentu pro experiment.

Dynamické vykreslení obsahu do šablon je základem webových aplikací, protože je využité prakticky na každém kroku. Příkladem je vypsání přijatých emailů po přihlášení do online aplikace Gmail (netvrdím, že Gmail využívá šablonové systému Flasku či Djanga, uvádím pouze jako příklad). Gmail po přihlášení uživatele do aplikace zavolá server a

požádá o všechny přijaté emaily uživatele. Přijatá odpověď se ale pro každého uživatele liší. Uživatel č.1 může mít email zřízený týden a ve schránce pouze 5 emailů, zatímco uživatel č.2 5 let a 10000 emailů. Aby webová aplikace mohla správně vykreslit 5 či 10000 emailů, využívá šablony, které dynamicky stránku podle potřeby mění (což standardní HTML neumožňuje).

Pro vykreslení dynamického obsahu využívají Flask a Django odlišné nástroje, čímž je naplněno kritérium pro výběr komponent číslo 2. Zatímco Flask využívá systém nazvaný Jinja 2, Django v základu nabízí Django Template System. Nutno podotknout, že Django i Flask umožňují využití odlišného šablonového systému, Django na svých stránkách specificky zmiňuje Jinja 2 jako dobrou alternativu pro Django Template System.

4.5.1 Metoda vykreslení dynamického obsahu ve Flasku

Pro naprogramování testu dynamické šablony ve Flasku je nutné využít dvou Flask tříd, Flask a render_template. Třída Flask je nejdůležitější třídou celého frameworku, protože bez ní není možné Flask webovou aplikaci vytvořit. Tato třída implementuje WSGI aplikaci a chová se jako centrální objekt, její konstruktor vyžaduje pouze jeden argument, jméno hlavního modulu aplikace. Třída render_template slouží k vykreslení šablony.

Kód pro test vykreslení dynamického obsahu pomocí Jinja 2 šablony ve Flasku je na obrázku Vykreslení šablony ve Flasku. Kód se skládá ze tří částí. V první části pouze importuji podstatné knihovny, v druhé části tvořím instanci třídy Flask. Třetí části vytváří cestu ke stránce zobrazující dynamicky obsah za pomoci Jinja 2 šablony a tuto stránku propojuje s šablonou nazvanou greetings.html.

```
from flask import Flask
from flask import render_template
import sqlite3 as sql
import string
import random

app = Flask(__name__)

@app.route('/render_template/<name>')
def hello(name=None):
    return render_template('greetings.html', name=name)
```

Obrázek 14: Vykreslení šablony ve Flasku

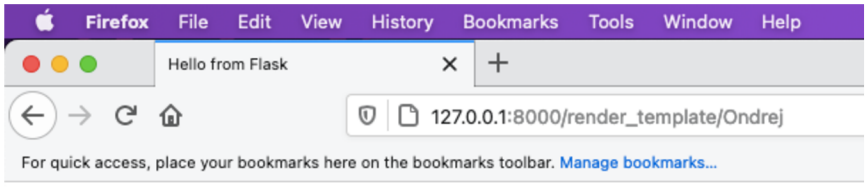
Dynamický obsah vykreslený do šablony se mění na základě URL. Cesta v URL se skládá z částí *render_template* a <*name*>. Špičaté závorky obalující name říkají Flasku, že se jedná o proměnnou. Tu lze nadále v kódu využít, například pro vykreslení v šabloně.

Pokud tedy do cesty za druhé lomítko vložíme jakýkoliv string, tak ho Flask přenese do šablony, která ho následně vykreslí. Příkladem takové URL může být: *www.somedomain.com/render_template/Petr*. Kód Jinja 2 šablony je na obrázku Jinja 2 šablona. Kód je ve své podstatě typický HTML soubor obohacený o jednu proměnnou. Skládá se z jednoduché deklarace typu textového dokumentu, názvu stránky (nepodstatné pro experiment) a nadpisu. Nadpis je tvořený slovy *Hello* a *{{ name }}*. *{{ name }}* slouží Jinje pro identifikování místa, kam chce vývojář vložit proměnnou. V případě mé experimentální aplikace se jedná o proměnnou name, kterou definuji v URL cestě.

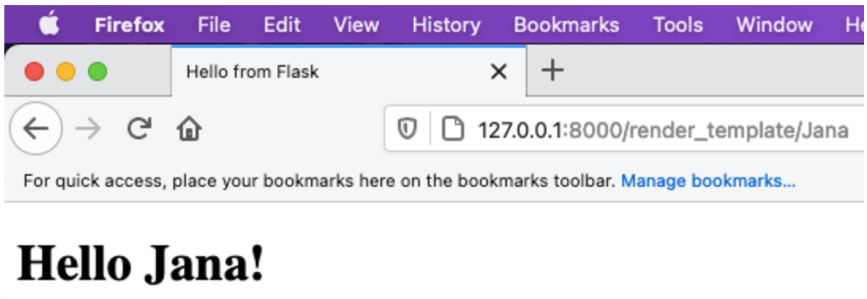
```
<!doctype html>
<title>Hello from Flask</title>
<h1>Hello {{ name }}!</h1>
```

Obrázek 15: Jinja 2 šablona

Výsledné vykreslení stránky v prohlížeči pomocí šablony je na obrázcích Vykreslení šablony v prohlížeči. Jak je z obrázků patrné, na základě URL se mění obsah zobrazený ve webovém prohlížeči. Jednou zdraví Ondřeje, podruhé Janu.



Obrázek 16: Vykreslení šablony ve Flasku příklad 1



Obrázek 17: Vykreslení šablony ve Flasku příklad 2

4.5.2 Metoda vykreslení dynamického obsahu v Djangu

V teoretické části práce jsem popsal Django jako kompletní framework, jehož cílem je maximálně usnadnit vývoj aplikace. To se projevu již od samého začátku, kdy stačí napsat přesně dva příkazy k tomu, aby Django založil nový projekt a novou aplikaci (Django třídí

veškerou práci na projekty a aplikace. Projekt je jedna webová stránka, která se skládá z aplikací. Například Google Cloud je projekt, jenž se skládá z aplikací kalendář, adresář a email). Projekt obsahuje hlavní konfigurační soubor, který se při základním programování upravuje pouze zřídka. Veškerá práce se odehrává v aplikaci.

```
ondrejmalina@MacBook-Air-3 render_template % tree
.
├── __init__.py
├── __pycache__
│   ├── __init__.cpython-39.pyc
│   ├── admin.cpython-39.pyc
│   ├── apps.cpython-39.pyc
│   ├── models.cpython-39.pyc
│   ├── urls.cpython-39.pyc
│   └── views.cpython-39.pyc
├── admin.py
├── apps.py
├── migrations
│   ├── __init__.py
│   └── __pycache__
│       └── __init__.cpython-39.pyc
├── models.py
├── templates
│   └── render_template
│       └── greetings.html
├── tests.py
├── urls.py
└── views.py
```

Obrázek 18: Strom Django aplikace

Aplikace Django se skládá z velkého množství souborů, viz obrázek Strom Django aplikace, které ale pro jednoduchý test zobrazení dynamického obsahu nejsou potřeba. Využité jsou pouze dva, urls.py a views.py. Soubor urls.py obsahuje mapu URLs dané aplikace, soubor views.py obsahuje samostatný kód aplikace. Stejně jako v případě Flasku je kód aplikace pro dynamické zobrazení obsahu za pomoci šablony prostý (viz Vykreslení šablony v Django). Skládá se z jedné knihovny a jedné funkce. Funkce sbírá proměnnou zadanou do URL a předává ji ke zpracování šabloně. Ta ji poté vykreslí do prohlížeče.

```python
from django.shortcuts import render

def greetings(request, name):
    return render(request, 'render_template/greetings.html', {'name': name})
```

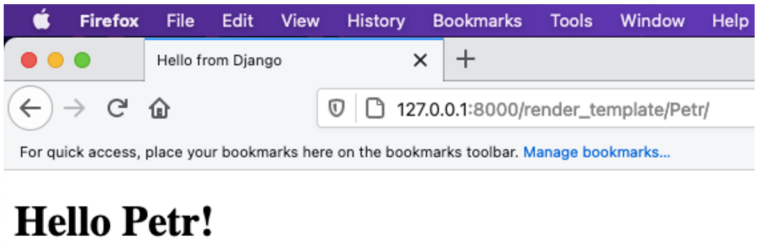
Obrázek 19: Vykreslení šablony v Django

Šablona pro Django Template System má v případě jednoduchého vykreslení obsahu totožný formát s šablonou z dílny Jinja 2. To je také jeden z důvodů, proč Django na svých webových stránkách zmiňuje Jinju jako validní náhradu svého výchozího šablonového systému.

```
1   <!doctype html>
2   <title>Hello from Django</title>
3   <h1>Hello {{ name }}!</h1>
```

Obrázek 20: Django Template System šablona

Vykreslení šablony prostřednictvím Django aplikace a Django Template System je na obrázku Vykreslení šablony v Djangu. Stejně jako v případě Flasku se na základě hodnoty zadané do URL zobrazila zpráva zdravící, v tomto případě, Petra.



Obrázek 21: Vykreslení šablony v Djangu

4.5.3 Srovnání kódu Django a Flask

Podstatná část správného provedení experimentu spočívá ve vytvoření takových podmínek, kdy jediná změna mezi sledovanými subjekty je změnou testovanou v rámci výzkumu. Srovnání aplikací napsaných v Djangu a Flasku tento požadavek bezesporu splňují. Využívají typově stejné šablony, zpracovávají stejný úkol, vykreslují stejný obsah. Jediný a hlavní rozdíl tedy spočívá ve způsobu, jakým k danému úkolu aplikace nativně přistupují.

4.6 Zápis do databáze

Druhou komponentou zahrnutou do testu je zápis do databáze. Stejně jako u první komponenty je zápis do databáze jednou z frekventovaně využívaných funkcí webových aplikací.

V Google kalendáři dojde k zápisu do databáze pokaždé, když uživatel přidá nový kontakt. U kalendáře je zápisem do databáze vytvoření nové události. V případě emailu se jedná o přidaní emailové adresy mezi spamy. Příkladů je nespočet a s nepatrnou nadsázkou lze říct, že k zápisu (samozřejmě i čtení) do databáze dochází v každé webové aplikaci. Bez funkce zápisu do databáze by webové aplikace nemohli existovat.

Pro test zápisu do databáze využívají aplikace odlišný systém. Flask se, jako vždy, spoléhá na knihovny třetích stran a schopnost uživatelně flexibilně integrovat dané knihovny do programu. Django naopak vsází na předpřipravený model, který uživatele v maximální

možné míře osvobozuje od nutnosti využít jiné knihovny, nebo dokonce psát v jiných jazycích.

Flask i Django umožňují uživateli využít několik různých databázových systémů. Já jsem si pro svůj experiment zvolil databázový systém SQLite3. Důvodem je výchozí konfigurace SQLite3 jako databázového systému v Django, což značně usnadňuje proces vývoje. Dalším důvodem je výchozí zahrnutí SQLite3 v Pythonu, čímž odpadá nutnost instalace dalších databází. Posledním důvodem je samotný koncept SQLite3, který nevyžaduje spuštění databáze na serveru.

4.6.1 Metoda vývoje testu pro zápis dat do SQLite databáze ve Flasku

Flask neumí pro uživatele vytvořit databázi, ani databázovou tabulku. Proto je nutné před vytvořením aplikace zapisující data do databáze danou databázi vytvořit. Knihovna sqlite3 k tomu vývojáři poskytuje všechny potřebné třídy. Pro účely testu jsem připravil jednoduchou tabulku, jak je z kódu na obrázku Tvorba databáze pro Flask test patrné. Nazval jsem ji random_text a skládá se z dvou proměnných. Proměnná id je primární klíč, který vzniká automaticky při každém zápisu dat. Pro jistotu jsem přidal restriktivní podmínku, že daná proměnná nesmí obsahovat nulové hodnoty. Druhá proměnná je random_text. Jedná se o string s maximální povolenou délkou 200 znaků. Nulové hodnoty opět nejsou povoleny.

```python
import sqlite3

def create_connection():
    conn = None
    conn = sqlite3.connect('database.db')
    print("Opened database successfully")
    return(conn)

def create_table(conn,sql):
    conn.executescript(sql)
    print("Table created successfully")
    conn.close()

def main():
    sql_create_table =  """
                        DROP TABLE IF EXISTS random_text;

                        CREATE TABLE random_text (
                        id INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL,
                        random_text VARCHAR(200) NOT NULL
                        );
                        """
    conn  = create_connection()
    create_table(conn,sql_create_table)

if __name__ == '__main__':
    main()
```

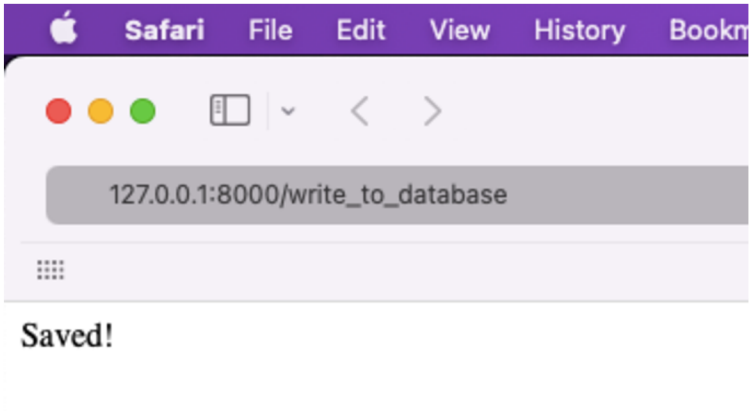
Obrázek 22: Tvorba databáze pro Flask test

Samotný kód pro test je na obrázku Kód pro test zápisu do databáze ve Flasku. Kód se skládá ze dvou funkcí, které pro své fungování využívají tři knihovny. První funkce create_random_text() je zbytná a v testu slouží jen jako drobné zpestření. Jejím výstupem je string stvořený z 10 náhodných znaků. Druhá funkce write_to_database slouží k zápisu dat do databáze. Daty zapisovanými do databáze jsou v tomto případě náhodné stringy o stejné délce generované první funkcí. V případě úspěšného vygenerování a vložení stringu do databáze zobrazí webová stránka potvrzení „Saved!“.

```
13  def create_random_text():
14      alphabet = string.ascii_lowercase
15      random_text = (''.join(random.choice(alphabet) for i in range(10)))
16      return(random_text)
17
18  @app.route("/write_to_database")
19  def write_to_database():
20      conn = sql.connect('database.db')
21      cursor =  conn.cursor()
22      cursor.execute("INSERT INTO random_text (random_text) VALUES (?)",(create_random_text(),))
23      conn.commit()
24      return "<p>Saved!</p>"
```

Obrázek 23: Kód pro test zápisu do databáze ve Flasku

Obrázek Spuštění zápisu dat do databáze ve Flasku ukazuje výstup webového prohlížeče v případě zadání cesty /write_to_database, jejímž načtením se spustí vygenerování náhodného stringu a zapsání dat do databáze. Jak vypadá samotný zápis v databázi vygenerovaný danou funkcí je na obrázku Data v SQLite3 databázi pro Flask test.



Obrázek 24: Spuštění zápisu dat do databáze ve Flasku

SELECT * FROM random_text rt

Grid | Text

	id	random_text
1	1	honeaiedpl
2	2	rwtszarkzm
3	3	eyhwpurrhc
4	4	zvyygwlgqi
5	5	mpiqamnmwl
6	6	sjmsetizac
7	7	gwnnrwbhkt
8	8	cbhvtiqemc
9	9	grorvypixk

Obrázek 25: Data v SQLite3 databázi pro Flask test

4.6.2 Metoda vývoje testu pro zápis dat do SQLite databáze v Djangu

Na rozdíl od Flasku využívá Django pro tvorbu databáze a databázových tabulek vlastní řešení, které ve své podstatě nevyžaduje po vývojáři znalost SQL. Další zajímavostí je sada předpřipravených databázových tabulek, které Django nabízí vývojáři, aby jejich definováním nebyl zdržován při práci na své aplikaci. Mezi ně se řadí tabulka logů z admin modelu nebo tabulka autorizovaných uživatelů aplikace.

Na obrázku Strom Django aplikace z kapitoly Metoda vykreslení dynamického obsahu v Django aplikaci je modul models.py. V tomto modulu vývojář definuje tabulky potřebné pro jeho aplikaci nad rámec tabulek generovaných Djangem automaticky. Kód pro vygenerování tabulky random_text je na obrázku Tvorba databáze pro Django test. Pro vytvoření tabulky je zapotřebí pouze jedné třídy z knihovny Django. Pro definici sloupců tabulek není nutné využít SQL.

```
1   from django.db import models
2
3   class RandomText(models.Model):
4       random_text = models.CharField(max_length=200)
5       class Meta:
6           db_table = "random_text"
```

Obrázek 26: Tvorba databáze pro Django test

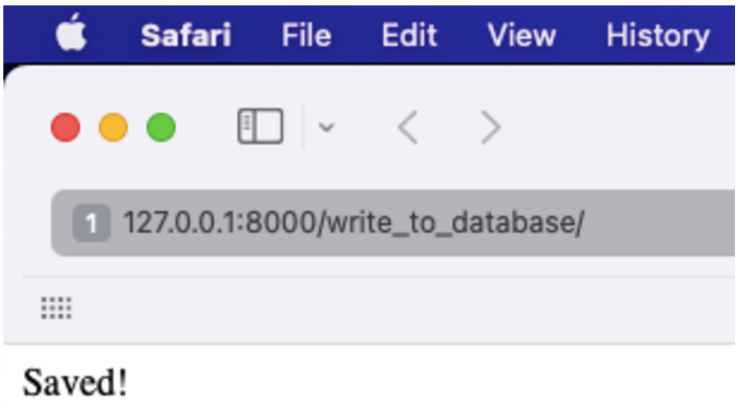
O celý proces zápisu do databáze se starají dvě funkce, viz obrázek Kód pro test zápisu do databáze v Djangu. První funkce vytváří náhodný string o délce deseti znaků.

Druhá funkce zapisuje do databáze a v případě úspěšného zápisu zobrazí potvrzení zápisu ve formě „Saved!“.

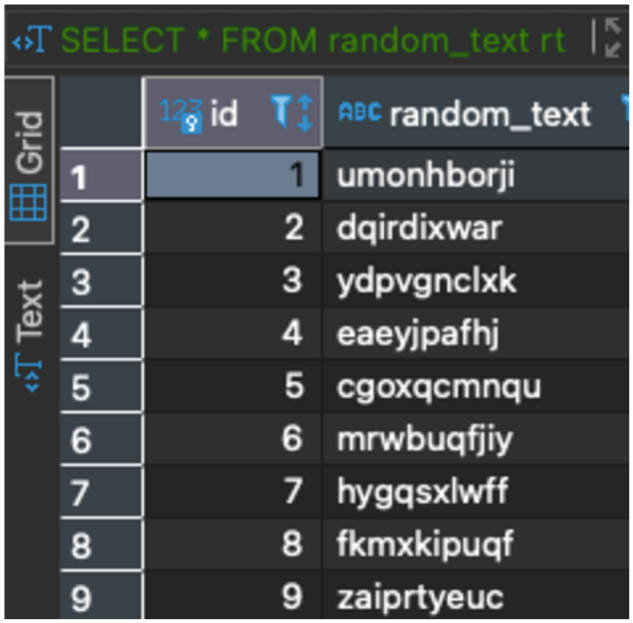
```
6   def create_random_text():
7       alphabet = string.ascii_lowercase
8       random_text = (''.join(random.choice(alphabet) for i in range(10)))
9       return(random_text)
10
11  def write_to_database(request):
12      q = RandomText(random_text=(create_random_text()))
13      q.save()
14      return HttpResponse("Saved!")
```

Obrázek 27: Kód pro test zápisu do databáze v Djangu

Na obrázcích Spuštění zápisu dat do databáze v Djangu a Data v SQLite3 databázi pro Django test je ukázka odpovědi prohlížeče v případě úspěšného zapsání dat do databáze a jejich následná podoba v databázi.



Obrázek 28: Spuštění zápisu dat do databáze v Djangu



Obrázek 29: Data v SQLite3 databázi pro Django test

4.6.3 Srovnání kódu Django a Flask

V případě testu vykreslení dynamického obsahu do šablony byly kódy aplikací vytvořených prostřednictvím frameworků Django a Flask velmi podobné. O to výrazněji působí kontrast mezi kódy v případě vytvoření databázových tabulek a aplikací pro zápis dat do databáze.

Flask nezprostředkovává vytvoření tabulek v databázi a přenechává tuto práci vývojáři aplikace. Ten si při jejím vývoji musí osvojit nejen práci s Flaskem, ale i databázovým systémem, který pro ni zvolí. Zároveň Flask neumí zapisovat do databáze. V případě webové aplikace s formulářem, jehož vstupní hodnoty chce vývojář zanést do databáze, poskytuje Flask pouze rozhrání pro přenos z uživatelského rozhraní do backendu. Jakmile jsou hodnoty v backendu, ponechá Flask zbytek práce knihovnám třetích stran, například SQLite3.

Django na rozdíl od Flasku nabízí veškerý komfort potřebný k snadnému odesílání dat z frontendu na backend. Pomocí jednoduchého příkazu vytvoří databázi včetně požadované databázové tabulky. Do této tabulky umí taktéž zapisovat. Celý proces nevyžaduje od vývojáře vystoupení z Django frameworku nebo využití jiného jazyku než Python.

Názorně rozdíl mezi přístupy frameworků demonstruje počet řádků kódu výše popsaných úkonů. V případě tabulky pro můj test si Django vystačí s 6 řádky kódu, Flask potřebuje 27 řádků kódu.

Na základě předchozích odstavců působí Flask nemotorně. Každý přístup má ovšem své klady a zápory. Django je pohodlnější z hlediska počtu úderů do klávesnice. Nemusí být ale rychlejší. Rychlost je přitom v případě vytížených aplikací klíčová. Dalším faktorem je preference vývojáře z hlediska chování frameworku. Ne každý oceňuje zprostředkování práce s databází Djangem. Proces vytváření databázových tabulek a zápisu do nich je standardním úkonem. Vývojáři mu dobře rozumí a nemožnost nastavit ho standardním způsobem může být nakonec negativem.

5 Výsledky a diskuse

Pro závěrečné srovnání frameworků jsem nastavil jednoduchou metodiku. Výkon definuji jako latenci aplikace napsané v daném frameworku, tedy rychlost její odezvy při volání klientem. Tato metrika je jediným a hlavním kritériem posouzení výsledku frameworků v jednotlivých testech. Srovnání naměřených hodnot ovšem nevyhodnocuji pouze na binární stupnici lepší/horší, ale zároveň si všímám absolutního rozdílu. Rozdíl v latenci dvou aplikací v řádu jednotek milisekund je méně závažný než rozdíl v řádu desítek, či stovek milisekund. Pro zajímavost čtenáře doplňuji metriky o průměrné latenci o související statistické ukazatele, jako průměrná a relativní směrodatná odchylka. Ty ovšem nemají na finální vyhodnocení vliv.

Každé měření jsem spustil třikrát, abych měl jistotu, že výsledky jsou validní a naměřené hodnoty příliš neoscilují. To se potvrdilo u každé z testovaných aplikací. Z tří měřících pokusů prezentuji vždy ten, jehož průměrná latence se nachází mezi zbylými dvěma pokusy.

5.1 Test vykreslení dynamického obsahu do šablony

Výstup z WRK lze rozdělit na dvě části. První, horní část, zobrazuje parametry Gaussovy křivky pro daný test. Do této části patří tabulka s řádky průměrná latence a počet volání za vteřinu (latency, req/sec) a sloupci průměr, směrodatná odchylka, maximální odchylka a relativní směrodatná odchylka. V druhé části výstupu je absolutní počet volání, přesná doba trvání testu, množství zpracovaných dat, absolutní počet zpracovaných žádostí za vteřinu a absolutní počet přenesených dat za vteřinu.

Výsledek testu vykreslení dynamického obsahu do šablony pro Flask je na obrázku Test vykreslení dynamického obsahu do šablony ve Flasku, výsledek testu Djanga na obrázku Test vykreslení dynamického obsahu do šablony v Djangu.

```
ondrejmalina@MacBook-Air-3 ~ % wrk -t2 -c40 -d10s http://0.0.0.0:8080/render_template/Ondrej
Running 10s test @ http://0.0.0.0:8080/render_template/Ondrej
  2 threads and 40 connections
  Thread Stats   Avg      Stdev     Max   +/- Stdev
    Latency    43.59ms    4.68ms  86.33ms   88.59%
    Req/Sec   459.77     38.02   535.00     70.50%
  9171 requests in 10.02s, 2.05MB read
Requests/sec:    915.21
Transfer/sec:    209.14KB
```

Obrázek 30: Test vykreslení dynamického obsahu do šablony ve Flasku

```
ondrejmalina@MacBook-Air-3 ~ % wrk -t2 -c40 -d10s http://0.0.0.0:8080/render_template/Ondrej/
Running 10s test @ http://0.0.0.0:8080/render_template/Ondrej/
  2 threads and 40 connections
  Thread Stats   Avg      Stdev     Max   +/- Stdev
    Latency    53.18ms    3.29ms  91.26ms   87.72%
    Req/Sec   376.71     17.55   404.00     80.00%
  7519 requests in 10.03s, 2.30MB read
Requests/sec:    749.91
Transfer/sec:    235.08KB
```

Obrázek 31: Test vykreslení dynamického obsahu do šablony v Djangu

Výsledky ukazují, že v případě prvního testu na základě průměrné latence je rychlejší Flask s šablonovým systémem Jinja 2. Průměrná latence Flasku je 43,6 ms, průměrná latence Djanga 53,18 ms. Flask je rychlejší o 9,58 ms v absolutních číslech, o 18 % v relativních číslech. Směrodatná odchylka Flasku je o něco vyšší než u Djanga, Django je tedy z hlediska latence konstantnější. To je vidět i na relativní směrodatné odchylce. Na druhou stranu, nejvyšší naměřené latence u Flasku je 86,33 ms, u Djanga 91,26 ms. Výsledky počtu zpracovaných volání za vteřinu zrcadlí výsledky latence. Flask zvládl v průměru zpracovat větší množství žádostí, stejně jako v případě latence má i vyšší hodnoty směrodatné odchylky a maximální hodnoty. Za 10 vteřin trvání testu vykreslil Flask šablonu 9150, průměrný počet vykreslení je 915 za vteřinu. Django vykreslilo 7490 šablon za 10 vteřin, v průměru 749 za vteřinu.

5.2 Test zápisu dat do databáze

Výsledky testu zápisu dat do databáze ve Flasku jsou na obrázku Test zápisu dat do databáze ve Flasku, výsledky pro Django na obrázku Test zápisu dat do databáze v Djangu. Ukazuje se, že ve srovnání s Flaskem je zápis dat do databáze v Djangu rychlejší. Průměrná latence při zápisu ve Flasku je 106 ms, zatímco v Djangu 46 ms. V absolutní hodnotě činí rozdíl mezi aplikacemi 59 ms, v relativní hodně se jedná o víc jak dvounásobnou rychlost.

```
ondrejmalina@MacBook-Air-3 ~ % wrk -t2 -c40 -d10s http://0.0.0.0:8080/write_to_database
Running 10s test @ http://0.0.0.0:8080/write_to_database
  2 threads and 40 connections
  Thread Stats   Avg      Stdev     Max   +/- Stdev
    Latency   105.59ms   28.16ms 252.06ms   87.56%
    Req/Sec   190.09     45.71   333.00     86.36%
  3796 requests in 10.04s, 648.73KB read
Requests/sec:    377.93
Transfer/sec:     64.59KB
```

Obrázek 32: Test zápisu dat do databáze ve Flasku

```
ondrejmalina@MacBook-Air-3 ~ % wrk -t2 -c40 -d10s http://0.0.0.0:8080/write_to_database
Running 10s test @ http://0.0.0.0:8080/write_to_database
  2 threads and 40 connections
  Thread Stats   Avg      Stdev     Max   +/- Stdev
    Latency    46.25ms    7.93ms 157.72ms   90.95%
    Req/Sec   433.90     60.41   630.00     71.00%
  8656 requests in 10.03s, 2.23MB read
Requests/sec:    863.01
Transfer/sec:    227.55KB
```

Obrázek 33: Test zápisu dat do databáze v Djangu

Flask v tomto testu ztrácí i v ostatních metrikách. Směrodatná odchylka Flask aplikace u odezvy je 28 ms, zatímco v Djangu 8 ms. Maximální naměřená latence Flasku je 252 ms, u Djanga 158 ms. Relativní směrodatná odchylka Flasku je 88 %, u Djanga 91 %.

Metriky počet volání za vteřinu vychází jednoznačně lépe pro Django. Flask zvládl v průměru 190.09 volání za vteřinu s průměrnou směrodatnou odchylkou 46, maximální směrodatnou odchylkou 333 a relativní směrodatnou odchylkou 86 U Djanga jsem naměřil víc jak dvojnásobně lepší hodnoty. V průměru Django zvládne 434 volání za vteřinu se směrodatnou odchylkou 60, maximální naměřenou hodnotou 630 a relativní směrodatnou odchylkou 71 %.

Ve výsledku Flask za 10 vteřin provedl 3796 zápisů do databáze, zatímco Django 8656 zápisů za vteřinu. Průměrný počet zápisů za vteřinu u Flasku je 378, u Djanga 863. Každou vteřinu Flask přenesl 65 KB dat, Django 228 KB dat.

5.3 Zhodnocení

Výsledky testování ještě jednou přehledně shrnuje tabulka Výsledky měření.

Test	Latence	Žádost / Vteřina
Vykreslení dynamického obsahu		
Průměr		
Flask	43,59	459,77
Django	53,18	376,61
Směrodatná odchylka		
Flask	4,68	38,02
Django	3,29	17,55
Maximalní odchylka		
Flask	86,33	535
Django	91,26	404
Relativní směrodatná odchylka		
Flask	88,63%	70,50%
Django	87,72%	80,00%
Zápis do databáze		
Průměr		
Flask	105,59	190,09
Django	46,25	433,9
Směrodatná odchylka		
Flask	28,16	45,71
Django	7,93	60,41
Maximalní odchylka		
Flask	252,06	333
Django	157,72	630
Relativní směrodatná odchylka		
Flask	87,56%	86,36%
Django	90,95%	71,00%

Tabulka 1: Výsledky měření

Na základě testování nelze prohlásit jeden z frameworků za výkonnější, protože každý z frameworků vykazoval lepší výkon v jednom z měření. Flask rychleji vykresluje dynamický obsah do šablony, Django rychleji zapisuje do databáze. Nicméně je nutné dodat, že rozdíl ve výsledcích prvního testu byl marginální, zatímco rozdíl ve výsledcích druhého testu byl markantní.

Výsledky naměřené v mém testu se v podstatě shodují s výsledky naměřenými ve srovnávacích testech Tech Empower (Tech Empower 2021), tedy alespoň při poměření podobně napsaných testů. V rámci srovnání mých výsledků s výsledky naměřenými portálem Tech Empower porovnávám pouze pozice frameworků v rámci žebříčku, nikoliv

absolutní hodnoty latence či jiné metriky. Testy Tech Empower byly měřeny na úplně jiném hardwaru v jiných podmínkách, proto považuji takové porovnání za zbytečné.

Portál Tech Empower má test nazvaný Plaintext, ve kterém měří rychlost frameworků při vykreslení prostého textu bez využití šablony (test s šablonou portál nemá). V tomto testu Flask zvítězil nad Djangem.

Srovnávacím kritériem pro test zápisu do databáze je na portálu Tech Empower test čtení z databáze. Tech Empower zápis do databáze neměří, pouze aktualizaci dat. Ta je ovšem spjatá s dalšími procesy, které mohou mít na testování značný vliv, zatímco čtení z databáze je prosté zobrazení jednoho databázového zápisu. V tomto testu zvítězilo Django, stejně jako v mém případě.

5.4 Diskuse

Flask se profiluje jako odlehčený framework poskytující uživateli pouze jádro nezbytné k napsání webové aplikace. Pro všechny ostatní účely pouze doporučuje vývojáři knihovny, ačkoliv finální rozhodnutí výběru nástroje ponechává na něm. Očekával jsem dominanci Flasku nad Djangem v obou testech. Byl jsem přesvědčený, že knihovny napsané pro jeden účel (sqlite3 v případě zápisu do databáze) musí být rychlejší než knihovny Djanga. Ty jsou také psané za jedním účelem, ale ne lidmi, které se na jeden daný účel specializují (toto je moje domněnka, která může být mylná). Výsledky měření jsou tedy pro mě do značné míry překvapením.

Určitě by bylo zajímavé a vhodné mnou napsané testy rozšířit o další komponenty, v optimálním případě o všechny srovnatelné, aby se jasně ukázalo, jak jednotlivé frameworky performují i v dalších oblastech. Kromě přidání dalších komponent by bylo zajímavé srovnat výkon frameworků v doslova produkčním prostředí při vypublikování na vzdáleném serveru, namísto lokálním.

Výkon vybraných komponent testovaných frameworků se v jednotlivých měřeních liší. V testu vykreslení dynamického obsahu do šablony o 18 %, v testu zápisu do databáze o více než 200 %. Přesto je nutné podotknout, že latence testovaných aplikací obou frameworků je velmi nízká. A když je latence velmi nízká u obou frameworků, pouze u jednoho ještě nižší, nemusí být v dnešní době silného hardwaru rychlost hlavním kritériem pro posouzení kvality frameworků. A to především v případě programování menších aplikací, které pravděpodobně nebudou muset odpovídat na velké množství požadavků zároveň.

Pokud se vývojář rozhoduje k napsání menší webové aplikace a nemůže si vybrat mezi využitím Djanga a Flasku, namísto volby řízené kritériem rychlosti se může rozhodovat i na základě dalších kritérií. Jedním z nich je například systém vývoje aplikace, který se u obou frameworků velmi liší. Jak je zřejmé ze čtvrté kapitoly této práce, Django má pro vývojáře předpřipravené téměř kompletní řešení vývoje, zatímco Flask nutí k využití knihoven třetích stran. Je již na každém, pro jaké řešení se nakonec rozhodne.

6 Závěr

Hlavním cílem práce bylo srovnat výkon vybraných komponent frameworků pro vývoj webových aplikací v Pythonu Django a Flask. Srovnal jsem výkon dvou komponent, první se využívá na vykreslení dynamického obsahu do šablony, druhá k zápisu dat do databáze. Srovnání ukázalo, že Flask umí rychleji vykreslovat dynamická data do šablony, zatímco Django umí rychleji zapisovat do databáze.

Na základě naměřených výsledků nelze tvrdit, že by jeden z frameworků byl rychlejší než druhý. Výsledek naznačuje, že každý z frameworků se hodí k jinému účelu. V případě zájmu o další rozšíření práce doporučuji doplnit srovnání o další komponenty. Větší množství srovnávaných komponent by umožnilo komplexněji charakterizovat oba frameworky a formulovat závěry o jejich silných a slabých stránkách.

Prvním vedlejším cílem práce bylo vybrat vhodné komponenty frameworků pro testování. Výběr komponent se řídil kritérii frekvence využití komponenty ve webových aplikacích a rozdílností implementace komponenty vývojáři frameworků. Na základě zvolených kritérií byly vybrány komponenty pro vykreslení dynamického obsahu do šablon a zápis dat do databáze.

Druhým vedlejším cílem práce bylo naprogramovat komponenty a připravit experiment. Komponenty byly naprogramovány tak, aby neobsahovaly žádné zbytečné kusy kódu, které by mohly mít vliv na rychlost aplikace. Detailní metoda naprogramování komponent je popsána v praktické části práce. Experiment byl připraven tak, aby v rámci možností maximálně odrážel reálné produkční nasazení komponent. Pro běh experimentu byly komponenty spuštěny na WSGI Gunicorn serveru a před ním stál NGINX reverse proxy server.

Třetím vedlejším cílem práce bylo provést a vyhodnotit měření experimentu. Měření probíhalo za pomocí nástroje WRK, který na komponenty posílá http žádosti a měří rychlost odpovědí. Vyhodnocení poté probíhalo na základě srovnání latence jednotlivých komponent.

Za hlavní přínos práce považuji komplexní spojení teorie potřebné k provedení experimentu srovnání výkonu webových frameworků a experimentu samotného. Ostatní práce většinou akcentují pouze jednu z těchto složek, ale nikdy ne oboje. Jak jsem již zmínil v kapitole Diskuze, v případě zájmu rozšířit tuto práci doporučuji doplnit více komponent ke srovnání. Poté by bylo možné formulovat přesnější závěry o povaze obou frameworků.

7 Seznam použitých zdrojů

CHI AN, Lie, 2018. *How the Puffin Browser Works. Puffin is wicked fast. Let's dig deeper… | by Liu An Chi (tigercosmos) | Coding Neutrino | Medium* [online] [vid. 2021-06-16]. Dostupné z: https://medium.com/coding-neutrino-blog/how-the-puffin-browser-works-440c91cece8f

DEVOPSCURRY, 2021. *What is Nginx? Understanding the basics of Nginx in 2021* [online] [vid. 2021-10-02]. Dostupné z: https://medium.com/devopscurry/what-is-nginx-understanding-the-basics-of-nginx-in-2021-f8ee0f3d3d54

DJANGO, nedatováno. *Writing your first Django app, part 1* [online] [vid. 2021-10-11]. Dostupné z: https://docs.djangoproject.com/en/3.2/intro/tutorial01/

GOLDBERG, Kevin, 2016. *An Introduction to Python WSGI Servers for Performance* [online] [vid. 2021-06-27]. Dostupné z: https://www.appdynamics.com/blog/engineering/an-introduction-to-python-wsgi-servers-part-1/

GRINBERG, Miguel, 2018. *Flask Web Development*. 2. vyd. ISBN 9781491991732.

INDEED EDITORIAL TEAM, 2021. *What Is a Web Application? How It Works, Benefits and Examples | Indeed.com* [online] [vid. 2021-06-16]. Dostupné z: https://www.indeed.com/career-advice/career-development/what-is-web-application

JEFFERS, Jackie, 2019. *Site Speed is (Still) Impacting Your Conversion Rate - Portent* [online] [vid. 2021-06-25]. Dostupné z: https://www.portent.com/blog/analytics/research-site-speed-hurting-everyones-revenue.htm

KENNEDY, Patrick, 2021. *What is Werkzeug?* [online] [vid. 2021-06-20]. Dostupné z: https://testdriven.io/blog/what-is-werkzeug/

KLENOV, Kirril, 2016. *GitHub - klen/py-frameworks-bench at 5ae32857c2bcb98bf068d1abe82fe94525790076* [online] [vid. 2021-06-14]. Dostupné z: https://github.com/klen/py-frameworks-bench/tree/5ae32857c2bcb98bf068d1abe82fe94525790076

LAFRENZ, Grayson, 2018. *The Importance of Site Speed in 2018 | Power Digital Marketing* [online] [vid. 2021-06-25]. Dostupné z: https://powerdigitalmarketing.com/blog/the-importance-of-site-speed-in-2018/#gref

LUTZ, Mark, 2013. *Learning Python*. 5th vyd. B.m.: O'Reilly Media. ISBN 9781449355739.

MARTIN, Robert C, 2008. *Clean Code: A Handbook of Agile Software Craftsmanship*. 1. vyd. USA: Prentice Hall PTR. ISBN 0132350882.

MENASCÉ, Daniel, 2002. *Load Testing, Benchmarking, and Application Performance Management for the Web.*

MISTRY, Jigar, 2021. *Top 10 Web Development Framework With Detailed Comparison* [online] [vid. 2021-06-26]. Dostupné z: https://www.monocubed.com/web-development-framework-comparison/

MOLINA-RÍOS, Jimmy a Nieves PEDREIRA-SOUTO, 2020. Comparison of development methodologies in web applications. *Information and Software Technology* [online]. **119**, 106238 [vid. 2021-06-16]. Dostupné z: doi:10.1016/j.infsof.2019.106238

NAYAK, Rahul, 2018. *Deploy Flask App with Nginx Using Gunicorn and Supervisor* [online] [vid. 2021-06-27]. Dostupné z: https://medium.com/ymedialabs-innovation/deploy-flask-app-with-nginx-using-gunicorn-and-supervisor-d7a93aa07c18

PRZEMEK, Rogala, 2020. *Jinja2 Tutorial* [online] [vid. 2021-06-20]. Dostupné z: https://ttl255.com/jinja2-tutorial-part-1-introduction-and-variable-substitution/

RIEHLE, Dirk, 2000. *Framework Design: A Role Modeling Approach* [online]. B.m. Swiss Federal Institute of Technology. Dostupné z: https://riehle.org/computer-science/research/dissertation/index.html

SCHONNING, Nick, Hamish WILLEE, Peter BENGTSSON, Chris MILLS, Florian SCHOLZ, Tim O'HAVER a Anton BERSHANSKIY, 2020. *Django introduction* [online] [vid. 2021-06-20]. Dostupné z: https://developer.mozilla.org/en-US/docs/Learn/Server-side/Django/Introduction

SOFIENE, Ben Khemis, 2019. *What is the Difference Between a Framework and Library? | by Sofiene Ben Khemis | Medium* [online] [vid. 2021-06-26]. Dostupné z: https://sofienebk.medium.com/what-is-the-difference-between-a-framework-and-library-2b712a1a1c41

SQLITE, nedatováno. *Appropriate Uses For SQLite* [online] [vid. 2021-10-16]. Dostupné z: https://www.sqlite.org/whentouse.html

STACK OVERFLOW, 2020. *Stack Overflow Developer Survey 2020* [online] [vid. 2021-06-14]. Dostupné z: https://insights.stackoverflow.com/survey/2020

TECH EMPOWER, 2021. *Web Framework Benchmarks* [online] [vid. 2021-06-26]. Dostupné z: https://www.techempower.com/benchmarks/#section=intro&hw=ph&test=fortune

VINCENT, William, 2018. *Django for Beginners*. B.m.: William S. Vincent.