Vysoká škola ekonomická v Praze

Fakulta informatiky a statistiky


# [Název práce]

DIPLOMOVÁ PRÁCE

Studijní program: Aplikovaná informatika

Studijní obor: Informační systémy a technologie

Autor: Bc. Ondřej Schrek

Vedoucí diplomové práce: Ing. Jarmila Pavlíčková, Ph.D.

Praha, prosinec 2019

## Prohlášení

Prohlašuji, že jsem bakalářskou práci [Název práce] [vypracoval/vypracovala] samostatně za použití v práci uvedených pramenů a literatury.

V Praze dne 1. prosince 2019 .....

Bc. Ondřej Schrek

## Poděkování

Děkuji Ing. Jarmile Pavlíčkové, Ph.D. za odborné směřování práce. Její vstřícnost a rady mi velmi pomohly ke zdárnému dokončení této práce.

## Abstrakt

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Aenean id metus id velit ullamcorper pulvinar. In convallis. Sed convallis magna eu sem. Maecenas lorem. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum. Ut tempus purus at lorem. Phasellus et lorem id felis nonummy placerat. Mauris elementum mauris vitae tortor. Aliquam id dolor. Vivamus porttitor turpis ac leo.

Nullam justo enim, consectetuer nec, ullamcorper ac, vestibulum in, elit. Integer vulputate sem a nibh rutrum consequat. Temporibus autem quibusdam et aut officiis debitis aut rerum necessitatibus saepe eveniet ut et voluptates repudiandae sint et molestiae non recusandae. Ut tempus purus at lorem. Neque porro quisquam est, qui dolorem ipsum quia dolor sit amet, consectetur, adipisci velit, sed quia non numquam.

Maecenas fermentum, sem in pharetra pellentesque, velit turpis volutpat ante, in pharetra metus odio a lectus. Vivamus ac leo pretium faucibus. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Nullam dapibus fermentum ipsum. In sem justo, commodo ut, suscipit at, pharetra vitae, orci. Nemo enim ipsam voluptatem quia voluptas sit aspernatur aut odit aut fugit.

## Klíčová slova

amet, consectetuer, dolor, Lorem ipsum, sit.

## JEL klasifikace

amet, consectetuer, dolor, Lorem ipsum, sit.

## Abstract

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Aenean id metus id velit ullamcorper pulvinar. In convallis. Sed convallis magna eu sem. Maecenas lorem. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum. Ut tempus purus at lorem. Phasellus et lorem id felis nonummy placerat. Mauris elementum mauris vitae tortor. Aliquam id dolor. Vivamus porttitor turpis ac leo.

Nullam justo enim, consectetuer nec, ullamcorper ac, vestibulum in, elit. Integer vulputate sem a nibh rutrum consequat. Temporibus autem quibusdam et aut officiis debitis aut rerum necessitatibus saepe eveniet ut et voluptates repudiandae sint et molestiae non recusandae. Ut tempus purus at lorem. Neque porro quisquam est, qui dolorem ipsum quia dolor sit amet, consectetur, adipisci velit, sed quia non numquam.

Maecenas fermentum, sem in pharetra pellentesque, velit turpis volutpat ante, in pharetra metus odio a lectus. Vivamus ac leo pretium faucibus. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Nullam dapibus fermentum ipsum. In sem justo, commodo ut, suscipit at, pharetra vitae, orci. Nemo enim ipsam voluptatem quia voluptas sit aspernatur aut odit aut fugit.

## Keywords

amet, consectetuer, dolor, Lorem ipsum, sit.

## JEL Classification

amet, consectetuer, dolor, Lorem ipsum, sit.

# Obsah

# Seznam obrázků

## Seznam tabulek

## Seznam zkratek

| | |
|---|---|
| BP | bakalářská práce |
| DP | diplomová práce |
| FIS | Fakulta informatiky a statistiky |
| UNESCO | Organizace OSN pro vzdělání, vědu a kulturu |
| NATO | Severoatlantická aliance |

# Úvod

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Aenean id metus id velit ullamcorper pulvinar. In convallis. Sed convallis magna eu sem. Maecenas lorem. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum. Ut tempus purus at lorem. Phasellus et lorem id felis nonummy placerat. Mauris elementum mauris vitae tortor. Aliquam id dolor. Vivamus porttitor turpis ac leo. Maecenas fermentum, sem in pharetra pellentesque, velit turpis volutpat ante, in pharetra metus odio a lectus. Phasellus et lorem id felis nonummy placerat.

# 1 Historie vývoje serverových aplikací

Server je označení pro zařízení, nebo počítačový program, který poskytuje funkcionalitu, případně služby klientům, kteří jsou většinou reprezentovány programy či zařízeními. První zmínka o serverech se datuje k roku 1969 v RFC-5, což je jeden z dokumentů popisující první globální síť ARPANET. (RFC 5: Decode Encode Language (DEL), 2005) Experimentální síť propojující univerzity, jež byla provozovaná až do roku 1990 a která se mimo jiné označuje jako předchůdce internetu, tak jak ho známe v dnešní době. V prvopočátcích byl vývoj serverových aplikací prováděn velmi ad-hoc, zejména díky tom že se tvořila pro servery nestandardizovaná API a také se využívali programovací jazyky, které byly zrovna programátorům k dispozici, jednalo se nejčastěji o jazyk C. Programovací jazyk C byl vyvinut v roce 1972, určený pro všeobecné použití, jedná se o imperativní a procedurální jazyk. V této době totiž neexistoval žádný jazyk, nebo jeho nadstavba která by se specializovala na tvorbu serverových aplikací. Každý web server měl vlastní API např. NSAPI, Microsoft ISAPI, proti kterému se programovali serverové aplikace, avšak se jednalo o proprietární a nestandardizovaná řešení, tudíž se vždy vyvíjelo pro konkrétní implementaci webového serveru daného výrobce. Tuto nejednotnost v roce 1993 vyřešil CGI (Common Gateway Interface) standart, který umožnil webovým serverům poskytovat jednotné rozhraní pro zpracování požadavků tzv. http requestů. Velkou výhodou tohoto přístupu byla technologická volnost pro vývojáře, kteří si mohli vybrat způsob implementace a případně jazyk, který jim vyhovoval pro vývoj a vyvíjet aplikace bez závislosti na rozdílné implementaci rozraní webových serverů různými výrobci.

Kromě výše zmíněného jazyka C se v té době pro serverový vývoj využíval jazyk Perl. Perl je skriptovací jazyk, který byl vyvinut v roce 1987, jedná se o dynamicky slabě typový jazyk. Jméno jazyka Perl je zkratka pro název Practical Extraction and Reporting Language. Jazyk, jak je patrné z názvu byl primárně určený pro vývoj skriptů pro systémy UNIX. Byť se jazyk Perl orientoval na jinou oblast vývoje i přesto první polovinu 90. let ve vývoji pro servery více či méně ovládl a stal se tak mezi serverovými vývojáři nepsaným standardem. Avšak stále neexistoval žádný jazyk, který by plně uspokojil požadavky vývojářů a byl primárně určen pro serverový vývoj. I přes velký rozmach trpěl jazyk Perl celou řadou nedostatků, jelikož jazyk nebyl původně zamýšlený pro vývoj serverových aplikací. Jedním z problému jazyka byla jeho velká sémantická a syntaktická volnost, dalším problémem bylo jeho primární zaměření pro vývoj v rozsahu velikosti skriptů. Perl byl tím pádem hůře použitelný pro vývoj enterprise aplikací, které jsou obvykle velmi rozsáhlé a díky výše uvedeným nedostatkům se stávali poměrně obtížně čitelné a spravovatelné pro vývojáře.

V této době se začali poprvé objevovat jazyky, které se používají i v současnosti pro vývoj serverových aplikací. V roce 1991 se objevil Python, interpretovaný, silně dynamicky typovaný objektový jazyk, který byl určen jak pro malé ale i enterprise řešení, jež se vyznačovali zejména vysokou čistotou vyvíjeného kódu aplikace. Dá se říct, že se jednalo o volného následníka jazyka Perl, který se dere velmi výrazně do popředí v žebříčcích zájmu uživatelů v průběhu tohoto desetiletí, kdy je přiřazován k nejprogresivnějším jazykům dnešní doby.

O pár let později, konkrétně v roce 1995 se objevil velmi populární skriptovací jazyk na straně serveru, jednalo se o jazyk PHP. PHP je interpretovaný jazyk, který je multi-paradigmatický. V počátcích vzniku se v PHP programovalo procedurálně, kdy díky tomuto přístupu často vznikal tzv. špagetový kód, avšak v současné době se programuje zejména objektově, s využitím frameworků, které jsou dnes při jeho využití již v podstatě nutností a vývoj na čistém PHP je velmi ojedinělý.

Ve stejném roce se poprvé představil jazyk Java, který způsobil ve světě serverového vývoje poměrně velký boom. Java přišla s v té době revolučním konceptem WORA – write once, run everywhere, což byla v té době poměrně zásadní vlastnost, díky které se odstínila platformní závislost, se kterou se kompilované jazyky v té době potýkali. Java je interpretovaný, silně staticky typovaný, objektově orientovaný jazyk, který se kompiluje do tzv. bytecode, který je následně spuštěn, přesněji řečeno interpretován v JVM – Java Virtual Machine, který zajišťuje jeho platformní nezávislost. Další vlastností díky, které se jazyk stal populárním je jeho blízká syntaktická podobnost s jazyky z rodiny C, které se v té době stále hojně využívali, avšak oproti nim nabízel jazyk Java přístup na vyšší úrovni a vývojáře odstiňoval od problémů, kterým byly nuceni čelit např. od práce s pamětí v jazycích rodiny C, který byl častým zdrojem chyb. Java oproti tomu disponuje velmi kvalitní automatickou správou paměti a tzv. Garbage collectorem, který se stará o dealokaci paměti automaticky a také širokou škálu kvalitních knihoven, které velmi zpohodlňovali práci vývojáře. Jednou z výrazných předností jazyka, je plná zpětná kompatibilita, což je na druhou stranu v současné době poměrně velkou brzdou v rozvoji jazyka.

Po miléniu na rozmach Javy zareagoval i Microsoft a vydal jazyk C#. Jazyk C# je kompilovaný, silně staticky typovaný, objektově orientovaný jazyk, který vychází z rodiny C. S jazykem C++ dokáže na programové úrovni kooperovat. Jeho primární zaměření není jako v případě Javy jen na servery, ale jeho využití je mnohostranné, navíc se v C# vyvíjejí i hry zejména pomocí populárního frameworku Unity. Jazyk umožňuje multiplatformnost díky kompilaci do MSIL code (Microsoft Intermediate Langugae), dnes nazýván CIL (Common Intermediate Langugae), který je následně just-in-time kompilován na hostiteli, což je podobný koncept jako používá jazyk Java.

Až v současném desetiletí se začali objevovat poměrně inovativní přístupy, které poměrně výrazně změnili pohled na serverový vývoj. V roce 2010 společnost Twitter představila jazyk JavaScript, který byl do té doby používán výhradně pro implementaci klientské strany, na straně serveru. První řešení vykazovalo výkonnostní problémy, avšak po mnoha optimalizacích architektury se podařilo získat velmi uspokojivé výsledky. V roce 2013 byl implementován jeden z prvních větších komerčních projektů, realizovaných právě pomocí JavaScriptu, konkrétně pomocí knihovny Node.JS, který umožňuje jednoduché použití JavaScriptu pro serverové aplikace. Jazyk JavaScript byl vydán již v roce 1995 a původně sloužil jako jazyk pro tvorbu skriptů na webové stránky. Jedná se dynamicky typovaný jazyk, který je multi-paradigmatický. Nejčastěji se využívá v kombinaci s nějakým frameworkem na klientu i na serveru, v čisté podobě se vyskytuje ojediněle, mimo jiné také díky velmi produktivní komunitě, která produkuje mnoho rozšíření postavených nad jazykem.

Po půli prvního desetiletí milénia se objevila tendence hledat alternativy pro Javu, které byly podpořeny poměrně dlouhou pauzou mezi vydávanými verzemi, nejdelší trvala 5 let od

roku 2006 kdy byla zveřejněna verze 6 do představení verze 7 v roce 2011, tedy poměrně dlouhá doba v porovnání s vývojem konkurenčních jazyků. Java díky zpětné kompatibilitě a malé aktivitě tehdejšího vlastníka jazyka společnosti Sun, nepřidávala zásadní vylepšení prostředí a jazyka jako takového. Vývojáři hledali alternativy, avšak málokdo chtěl úplně opustit svět, který se točil kolem poměrně kvalitně vybudovaného prostředí JVM. Začali se objevovat jazyky jako Clojure, Scala, Groovy, které byly postavené nad JVM, využívali plně její potenciál a v některých případech i mnohem více než Java. Toto se projevovalo v měření výkonu a alokace paměti, kdy některé jazyky dokázali předčit samotnou Javu, zejména v kódu s funkcionálními prvky. (Slesarev, 2019; Plessis, 2017) Jazyky také implementovali moderní koncepty a netradiční přístupy, které Java postrádala.

Groovy je silně dynamicky typovaný skriptovací jazyk pro JVM. Poskytuje pokročilou správu více vláknového zpracování. Scala je multi-paradigmatický jazyk postavený na kombinaci funkcionálních a objektových přístupů. Stejně jako Groovy poskytuje pokročilou správu více vláknového zpracování, avšak na rozdíl od něj je silně staticky typovaný jazyk. Clojure je v podstatě obdoba jazyka Lisp na JVM, která klade důraz na více vláknové zpracování a jedná se o čistý funkcionální jazyk.

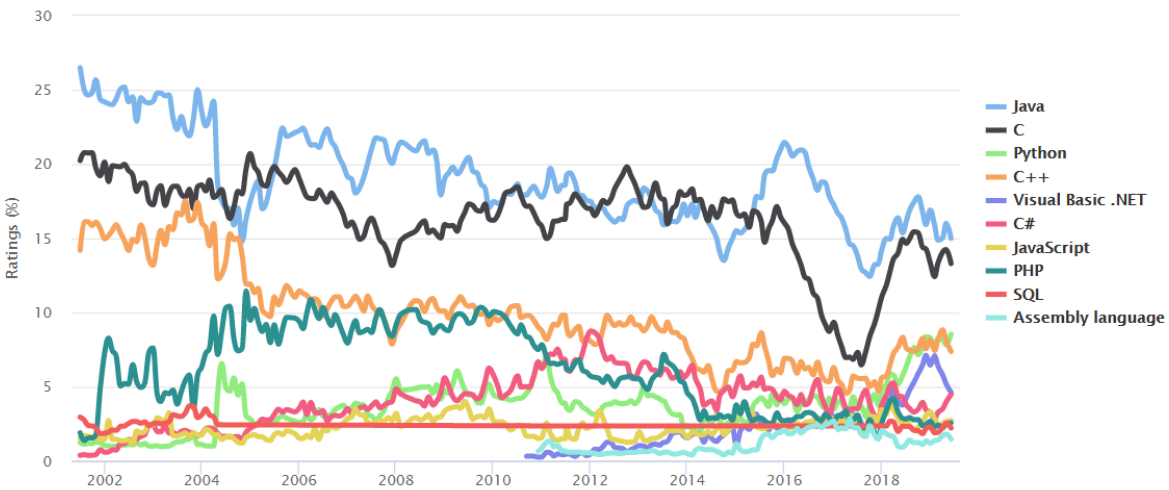
Výše zmíněné jazyky byly spíše pokusy, které využívali netradiční přístupy. Díky tomu faktu, se kterým souvisela poměrně náročná adaptace vývojáři a také na vrub ne zcela příznivé kompatibilitě s Javou stály tyto jazyky více méně v ústraní (Frankel, 2017). Aktivně je v komerčních projektech využívalo poměrně malé množství vývojářů v porovnání s původním jazykem postaveným nad JVM a to Javou.

Jeden z mála jazyků, který zaznamenal výraznější úspěch než výše uvedené a přitáhl pozornost mnoha vývojářů je Kotlin. Jazyk, který sází na plnou kompatibilitu s Javou a velmi rychlou adaptaci Java vývojáři. Kotlin je silně staticky typovaný jazyk, umožňující vývoj podle nejpoužívanější paradigmat (procedurální, objektové, funkcionální). Kotlin je plně objektový jazyk, oproti Javě v něm nenajdeme primitivní datové typy. První zmínky o Kotlinu se datují do července roku 2011.

# 2 Současnost ve vývoji serverových aplikací

Ke zmapovaní technologií, které se v současné době nejčastěji používají pro vývoj serverových aplikací byla využita veřejně dostupná měření a výzkumy. Vyhodnocení bylo provedeno kombinací vybraných zdrojů, jelikož každý zdroj pokrývá jen část domény a jen ve vzájemné korelaci poskytují směrodatný výsledek.

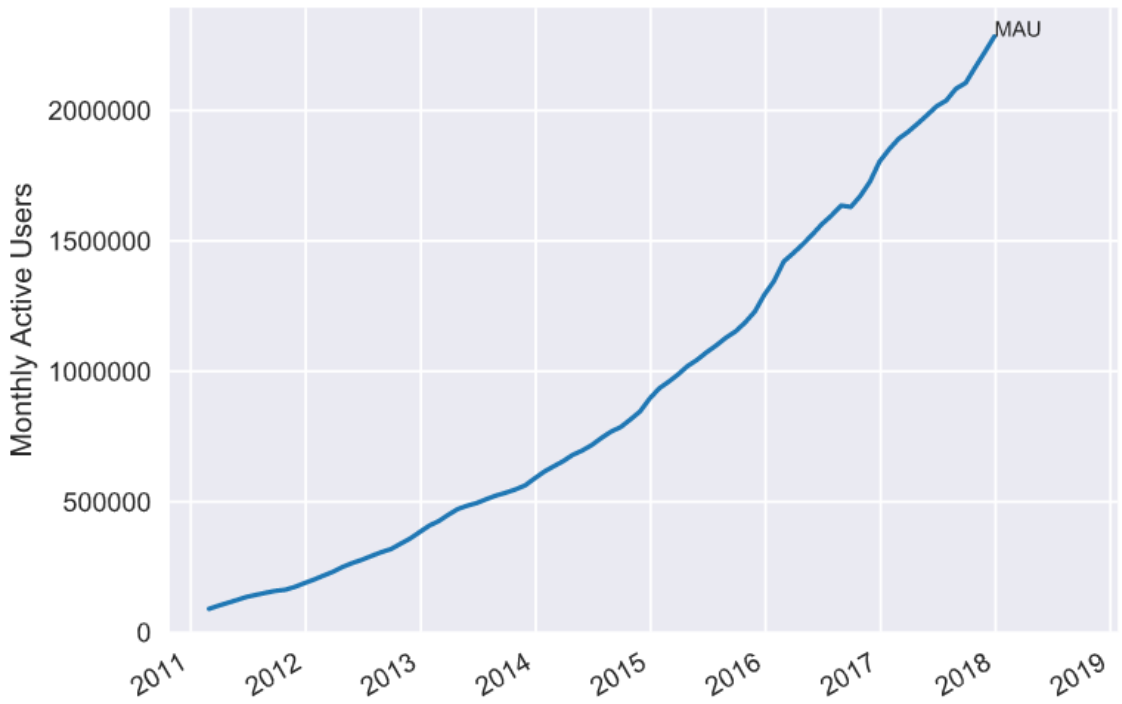
Pro zhodnocení zájmu o osvojení programovacího jazyka jsem vyhledal metriku sledující trendy v četnosti vyhledávání klíčových slov, která se vztahují k vyhledávání tutoriálů programovacích jazyků, což se dá pokládat za množství zájmu, které se technologii skrze uživatele dostává. Toto hodnocení se nazývá PYPL (Popularity of Programming Language). K červnu 2019 mluví statistiky jasně o vítězi, kterým se stává jazyk Python s 28 % podílem z celkového počtu a relativním přírůstkem o 4,7 %, na druhém místě je Java s 20 % a s úbytkem o 1,8 %, na třetím místě se umístil JavaScript s již pouze 8 % a drobným úbytkem v řádu desetin procent, na čtvrtém místě se 7 % se umisťuje jazyk C#, která má úbytek zhruba půl procenta, páté místo obsadilo PHP s necelými 7 % a poměrně vysokým úbytkem v podobně 1 %. Jazyk Kotlin obsadil třináctě místo s 1,5 % a relativním přírůstkem 0,5 %, což je v procentuálním vyčíslení největší posun v celém žebříčku. Je zřejmé že vývojáři mají tendenci se učit nové a netradiční jazyky.




Obr. 2.1: TIOBE index

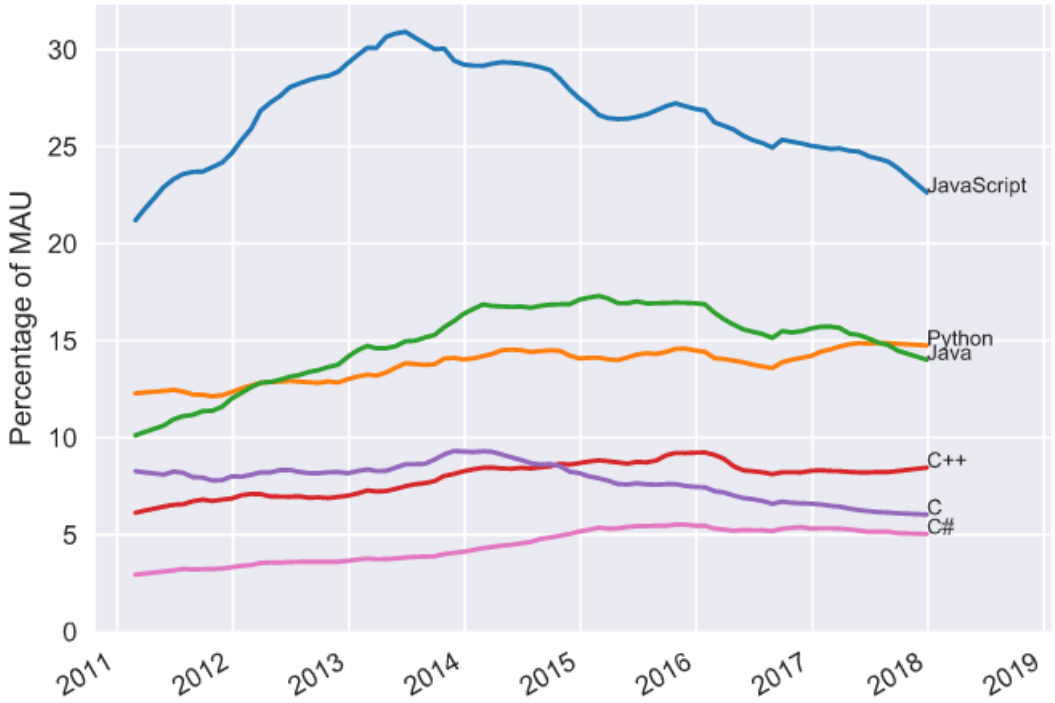
Další metrikou je TIOBE Index, který měří popularitu jazyků napříč vyhledávači, kde sleduje zájem o jazyky podle vyhledávání klíčových slov a neomezuje se pouze na tutoriály například jako výše zmíněný PYPL. Z grafu, je patrný prudký růst dotazů v posledních letech na jazyk Python, u ostatních jazyků je trend spíše klesající. Růst Pythonu podpořil také zvýšený zájmem o umělou inteligenci a strojové učení, kde se jazyk velmi často využívá. Zajímavý je výrazný propad, který Java zažila po roce 2016, který utnul její poměrně strmý růst.

Ben Fredirickson vytvořil hodnocení jazyků podle dat o struktuře repositářů na GitHub, což je veřejné uložiště systému pro verzování Git. Do úložiště přispívá celkem 37 miliónů uživatelů, kteří spravují dohromady 75 miliónů repositářů a uskutečnili již celkem 1,25 miliard transakcí. Každá transakce je spárovaná s konkrétním repositářem a uživatelem. Fredirickson sjednotil transakce každého uživatele a repositáře za měsíc a díky údajům, které poskytuje repositář, bylo možné přiřadit kolekci sjednocených transakcí, ke konkrétnímu jazyku jako jeho použití. Z toho vznikl ukazatel Monthly Active User, tedy uživatelé, kteří jsou v rozmezí jednoho měsíce aktivní. Hodnoty dosahují v roce 2019 přes 2 miliony aktivních uživatelů za měsíc. Díky tomu rozsahu mají statistiky poměrně dobrou vypovídající hodnotu.



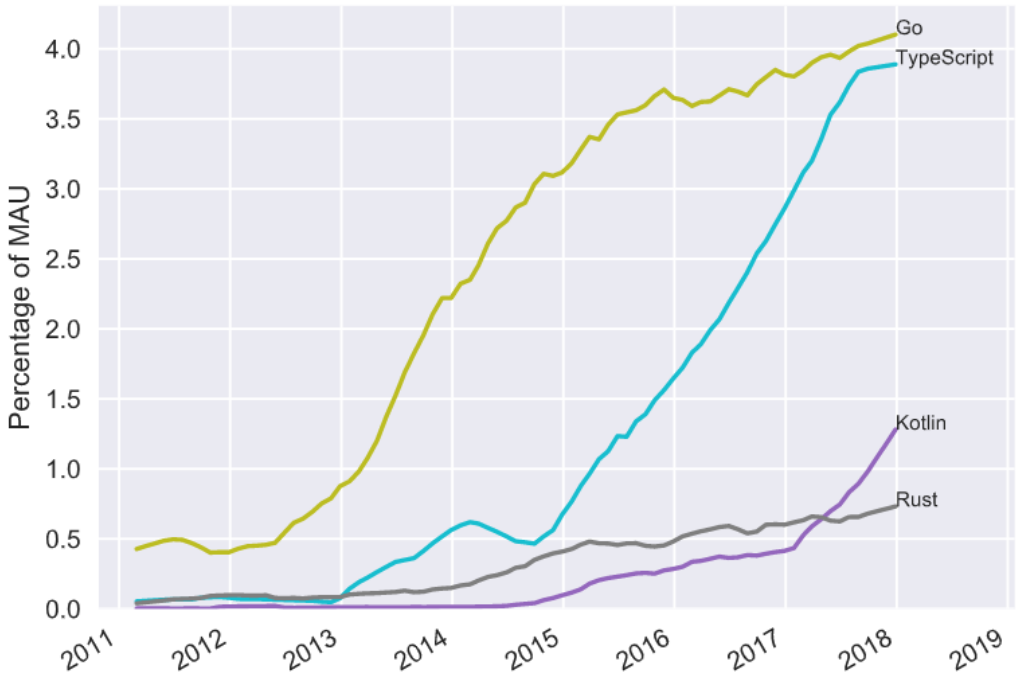

Obr. 2.2: Celkový počet aktivních uživatelů na serveru

První metrikou bylo hodnocení programovacího jazyka podle počtu aktivních uživatelů za měsíc. U této metriky vyšel jako vítěz JavaScript, který měl 23 %, byť má klesající trend a pomalu se vrací až na počáteční úroveň z roku 2011, tedy předtím, než začal expandovat díky jeho uvedení jako full-stack řešení. Druhý Python s 15 % a mírným růstem, Java třetí se 14 % a klesajícím trendem, PHP se umístilo na šestém místě s 6 % a velmi výrazným klesajícím trendem, na sedmém C# s 5 %, které se drží již delší dobu na stejné hodnotě. Kotlin se objevil na patnáctém místě s 1,3 % a velmi prudkým růstem.



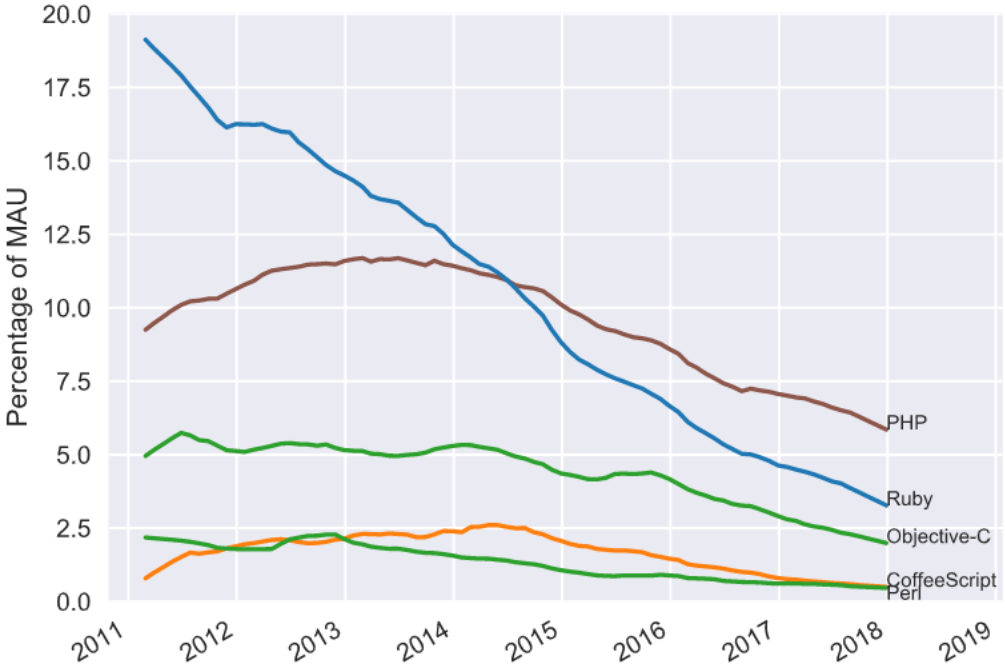

Obr. 2.3: Počet aktivních uživatelů dle jazyku

Druhá statistika zachycuje programovací jazyky, které mají největší relativní přírůstek aktivních uživatelů. Jedná se tedy o výběr nejvíce progresivních jazyků, kde rozhodujícím kritériem je vysoká míra zájmu, kterou jazyku programátoři projevují. Na vrcholu statistiky se již delší dobu drží jazyk Go, jenž je vyvíjený společností Google od roku 2007, avšak první stabilní verze byla uvedena až v roce 2012. Jedná se o kompilovaný, silně staticky typovaný, multi-paradigmatický jazyk, se syntaxí odvozenou z rodiny jazyků C. Hlavní motivací autorů byl záměr poskytnout inovovaný jazyk v mnoha aspektech podobný svému vzoru, avšak při zachování vysokého výkonu. Jazyk měl být obohacen o vlastnosti moderních programovacích jazyků, zvláštní důraz byl kladen na paralelizaci. Mezi klíčové benefity jazyka patří zvýšené zabezpečení paměti, kdy její správu převzal garbage collector a tzv. gorutiny, které umožňují paralelizaci na úrovni jazyka, díky tomu jsou o dost méně náročnější na využívání zdrojů než v případě tradičního řešení v podobě vláken. Go však nedisponuje všemi vlastnostmi, na které jsme zvyklí z jiných jazyků např. dědičnost a generika. Další nové programové konstrukty jsou do jazyka Go přidávány poměrně konzervativně, což mu někteří kritici vyčítají. Do jazyka Go jsou v současnosti nejvíce přepisovány síťové a webové aplikace na kterých je po převodu běžně zaznamenáván nárůst rychlosti v násobcích o řádu desítek a v některých případech dokonce až stovek. Druhým jazykem je TypeScript, což je nadstavba pro JavaScript, která poskytuje typovou kontrolu a další vlastnosti, které jsou běžně používány v objektových jazycích. Velkou výhodou TypeScriptu je jeho plná kompatibilita s JavaScriptem díky přímému překladu, čím si získal značnou oblíbenost v komunitě vývojářů. Třetí je jazyk Kotlin, který zažívá poměrně strmý nárust popularity od roku 2017, což se dá spojit s jeho oficiálním představením Googlem v květnu 2017 jako výchozí jazyk pro vývoj android aplikací. Avšak podle oficiálních statistik GitHub za rok 2018 je Kotlin oceněn jako nejrychleji rostoucí jazyk, který téměř ztrojnásobil počet kontributorů, v porovnání s Go, které dosáhlo pouze jeden a půl násobku.

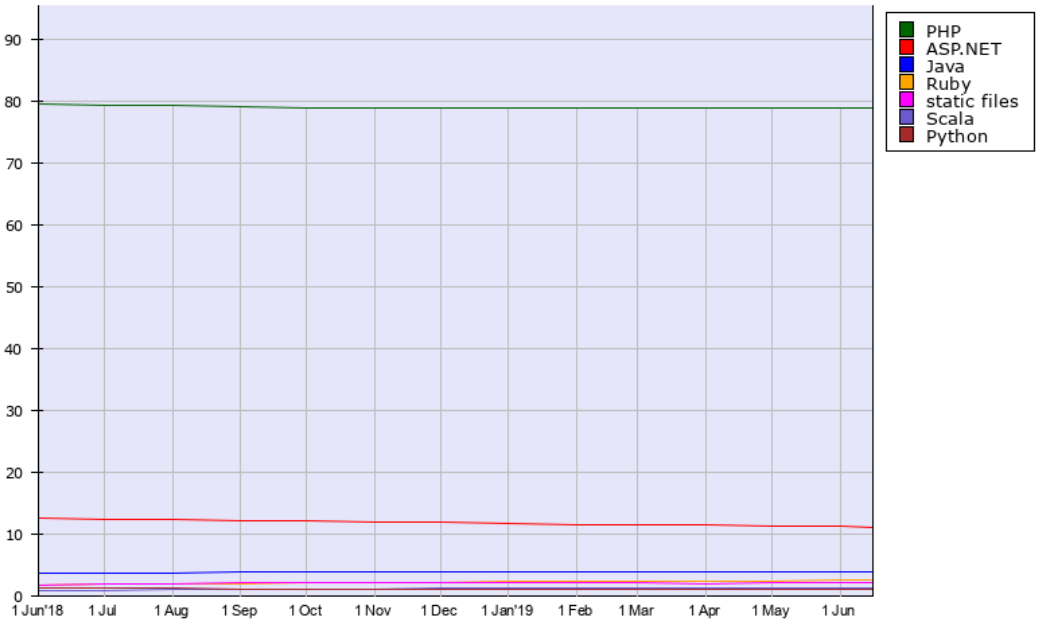



Obr. 2.4: Nejvyšší relativní přírůstek aktivních uživatelů

Třetí statistika naopak zobrazuje největší procentuální odlivy aktivních uživatelů od programovacího jazyka. Na tomto grafu můžeme sledovat, které jazyky upadají v zájmu vývojářů, jenž je méně častěji využívají, respektive nepřispívají do repositářů, kde převládá daný jazyk. Ze serverových jazyků upadá zvláště jazyk Ruby, nad kterým se používá oblíbený webový framework Rails tzv. Ruby on Rails. Dalším velmi populárním jazykem, který upadá zájmu je jazyk PHP, pokles je nejspíše způsoben růstem inovativních jazyků a rozšíření využívání JavaScriptu i na serverové straně, avšak PHP všeobecně zůstává ve středu pozornosti, co se týče využívání pro tvorbu webových aplikací. Toto potvrzuje průzkum W3Techs, který zkoumá technologie použité na webových stránkách, které se umístili v prvních 10 miliónech dle návštěvnosti. Z toho PHP využívá téměř 8 miliónů nejnavštěvovanějších stránek světa. Za zmínku stojí i pokles jazyka Perl, který byl zmíněn v kapitole o historii vývoje.

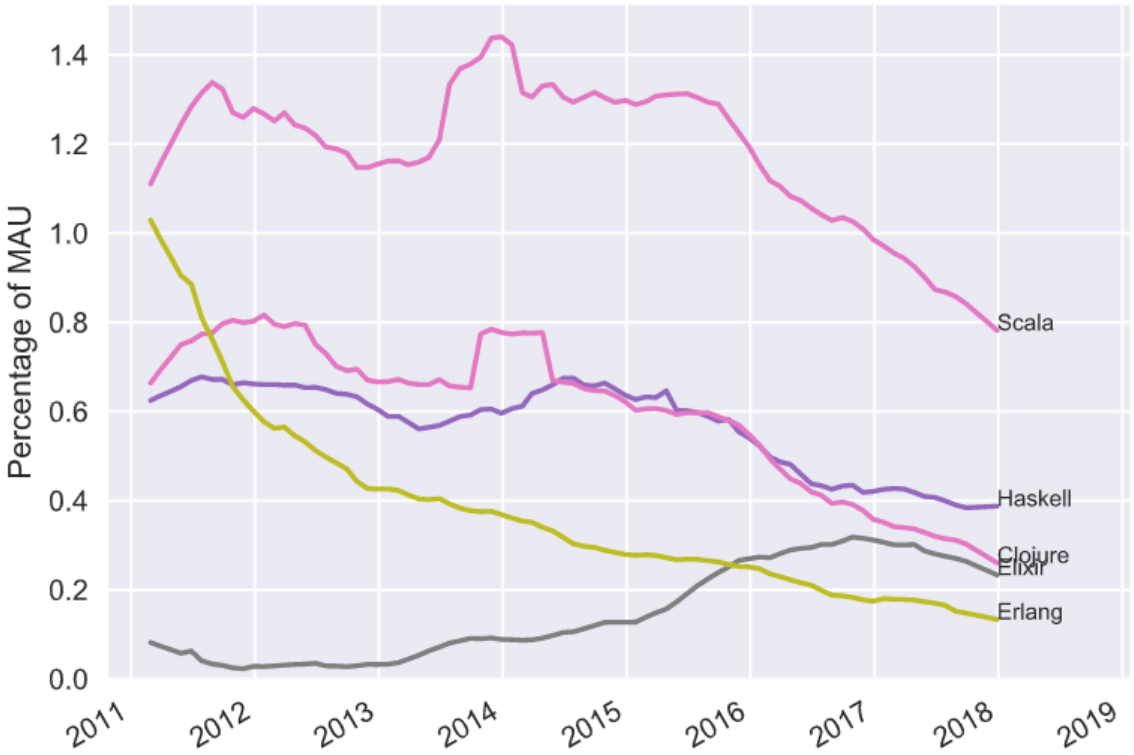



Obr. 2.5: Nejvyšší relativní úbytek aktivních uživatelů



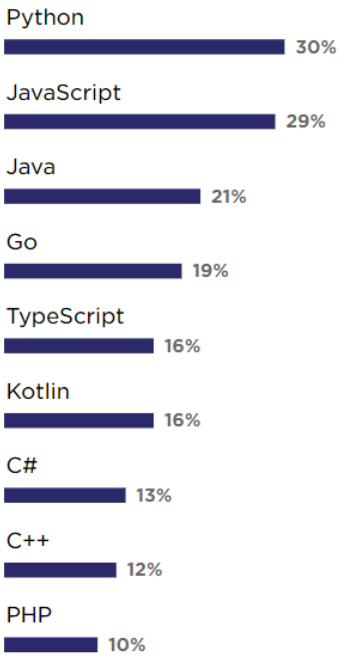

Obr. 2.6: Výsledky průzkumu společnosti W3Techs
(Usage of server-side programming languages for websites., 2019)

Poslední zajímavou statistikou je také přehled oblíbenosti funkcionálních jazyků, které byly v popředí zájmů i pro vývoj serverových aplikací. Z grafu je patrné že největší boom pro funkcionální jazyky byl mezi roky 2013–2015. Avšak současný trend pro všechny funkcionální jazyky je dlouhodobě klesající.




Obr. 2.7: Počet aktivních uživatelů funkcionálních JVM jazyků

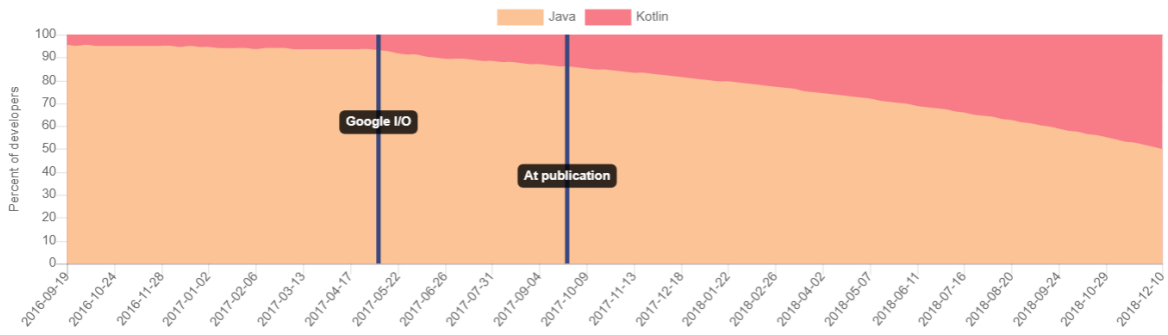
Výše zmíněné statistiky byly spíše zaměřeny na technologicky orientována, tzv. tvrdá data. Pro srovnání jsem zvolil několik průzkumů mezi samotnými vývojáři, které by měli korelovat s výše uvedenými statistikami. První statistikou je každoroční průzkum mezi vývojáři, který provádí společnost JetBrains. Ankety byly k roku 2018 a částečné výsledky za rok 2019. Celkem v ní bylo dotazováno přes třináct tisíc vývojářů a ve statistice bylo zastoupeno 58 % backend vývojářů. Jednou z otázek byl dotaz, který jazyk se začnou učit, nebo budou pokračovat v jeho osvojování v roce 2019. Výsledky korelují s výše uvedenými statistikami, oproti nim zde ale více dominuje jazyk Kotlin, který předběhl mnoho známých a populárních jazyků. Tento jev mohl být způsoben lehkým zkreslením díky zacílení sběru dotazníků v okruzích uživatelů či jinak spřízněných osoby s firmou JetBrains. Firma JetBrains vyvíjí jazyk Kotlin, tím pádem je vyšší pravděpodobnost že dotazník zasáhl právě velkou část Kotlin komunity a také Java vývojáře pro které firma vytváří IDE Idea IntelliJ, jelikož dle dat se většina příznivců Kotlinu rekrutuje právě z Java komunity. Mezi roky 2018 a 2019 v dotazníku významně vzrostl počet uživatelů Kotlinu, v roce 2018 ho aktivně používalo pouze 9 % dotazovaných a 13 % se ho chystalo využívat. Následující rok bylo již 16 % aktivních uživatelů a dalších 10 % se ho chystá v budoucnu využívat. Průzkum také ukazuje, jaké další technologie vývojáři využívající Kotlin ovládají, dominují dva jazyky, Java s 86 % a JavaScript, který ovládá 61 % uživatelů Kotlinu. Téměř 62 % uživatelů využívá Kotlin pro vývoj mobilních aplikací, tedy míří vývoj na platformu Android. Na JVM míří vývoj 57 % uživatelů, z toho celkového počtu je pro serverový vývoj využíván ve 41 % a pro ostatní použití pouze v 16 %. Kotlin je v 96 % případů nasazení využíván pro nové projekty, ve zbylém počtu se jedná o již existující projekty. Co se týká úrovně zkušeností vývojářů, tak Kotlin využívá aktivně méně jak 2 roky téměř 84 % vývojářů, pouze 1 % využívá Kotlin déle jak 4 roky, tyto data jsou k roku 2019.




Obr. 2.8: Zájem o osvojení nového programovacího jazyka
(The State of Developer Ecosystem in 2018, 2018)

Dalším uživatelským průzkumem, který probíhá již dlouhodobě a s každoroční pravidelností je dotazník internetového vývojářského fóra StackOverflow, oproti JetBrains statistice není okruh dotazovaných uživatelů tolik vymezen, jelikož fórum navštěvují různí vývojáři bez ohledu na využívané technologie. Pro porovnání jsem vybral statistiky za roky 2018 a 2019 aby se dal pozorovat meziroční rozdíl v hodnotách. Dotazník vyplnilo v roce 2018 100 tisíc vývojářů a v roce 2019 necelých 90 tisíc vývojářů po celém světě. Více než polovina vývojářů byla zaměřena na backend, stejně jako u dotazníku JetBrains. Nejvíce zastoupenou kategorií vývojářů, byli ti, kteří mají praxi 4-8 let, což odpovídá seniornímu profilu vývojáře. Co se týká technologií 67 % vývojářů ovládá JavaScript, který poklesl o 3,5 %, 41 % Javu, která poklesla o 4 %. Python 41,7 %, který si polepšil o 4 %. C# 31 % a poklesl o 4 %, PHP 26 % a pokleslo o 5 %. Naopak zaznamenal výrazný procentuální růst jazyk Kotlin 6,4 % s nárůstem o 2 %. Další sekce mapovala oblíbenost webových frameworků. Meziročně se u téměř každého produktu zvýšila obliba, avšak pořadí zůstalo stejné. Jako nejpoužívanější framework byl Node.js s 49 %, druhý byl .NET s 37 %, třetí Spring s 16 % u něj došlo k procentuálnímu úbytku, čtvrtý skončil Django s 13 %, což je framework pro jazyk Python. Ve statistice, kde se dotazovali vývojářů, v jakém jazyce by si přáli vyvíjet, zvítězil Python s 25 %, druhý JavaScript s 17 %, třetí Go s 15 % a čtvrtý Kotlin s 11 %. Pořadí zůstalo během roku poměrně beze změn a dá se říct, že žádný výrazný výkyv v přání vývojářů nenastal.

Oba průzkumy potvrdily trendy, které byly patrné z analýzy provedené na tvrdých datech, a to že vývojáři preferují nové, progresivní jazyky, které jim poskytují nové možnosti pro tvorbu serverových aplikací a nezdráhají se využít i poměrně mladé technologie pro produkční nasazení. V současné době jsou využívány hlavně jazyky PHP, Java, C#, JavaScript pro tvorbu serverových aplikací. Avšak mimo tyto zažité technologie se derou, do středu zájmu také méně rozšířené, nebo nové technologie jako je Python, Kotlin, avšak statistiky ukazují že mají potenciál konkurovat již zavedeným jazykům. V případě Pythonu se očekává že doroste, či dokonce přeroste tradiční technologie v řádu několika let, pokud bude pokračovat v současném tempu růstu. U Kotlinu je situace o něco komplikovanější, protože se jedná o jazyk úzce spojený s Javou, avšak v rámci prostředí JVM se jedná o podobný děj jako v případě Pythonu, kdy počet nových projektů zakládaných v Kotlinu se značně zvyšuje oproti počtu těch, které jsou v Javě, na platformě Android již Kotlin převyšuje Javu a má k říjnu 2018 podíl téměř 51 %.




Obr. 2.9: Využití Kotlinu a Javy v nových projektech

# 3 Trendy ve vývoji serverových aplikací

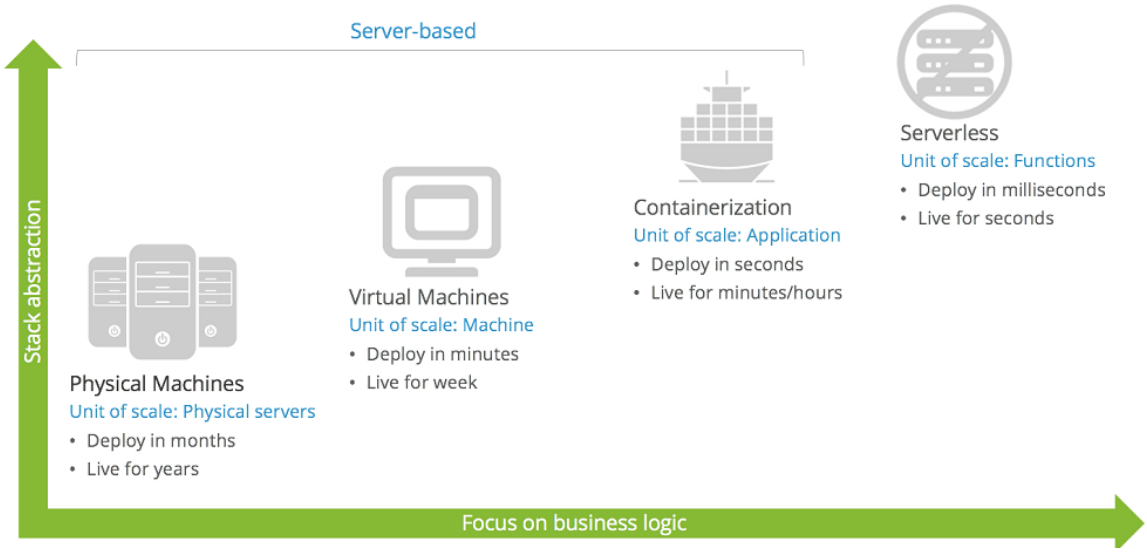
V době, kdy se již dostáváme na hranice možností křemíků a škálovat výkon aplikaci, je nutné jinými způsoby než jen pouhým navyšováním výkonu hardwaru. Ke slovu přichází paralelizace a distribuované zpracování, které se čím dál častěji implementuje právě v serverových aplikacích. Trend současnosti je stavět distribuované systémy, které umožňují paralelizaci zpracování a snadné škálování výkonu. V dnešní době toto reflektují i programovací jazyky, které se snaží podporovat a usnadňovat vývojářům paralelizaci např. Go, Kotlin mající řešení, která jsou oproti standardním jazykům a jejich vláknům poměrně nenáročná na zdroje a dají se poměrně masivně škálovat.

Zvládání komplexity velkých aplikací je hlavní výzva pro vývojáře softwaru a architekty, kteří systém navrhují. Podrobněji se touto problematikou zabývali Moseley a Marks v publikaci Out of the Tar Pit, kde navrhují nové způsoby zvládání složitosti rozsáhlých systémů. Navrhují rozdělení na základní logiku a také základní a vedlejší stav. V tomto modelu se bussines logika nezabývá stavy, ale pouze definuje relace, integritní omezení a provádí čisté funkce bez side-efektů. Naopak změny stavů neovlivňují logiku aplikace, ale pouze spouštějí akce (funkce bussines logiky) na ostatní elementy v systému. Tento přístup je v podstatě podporován funkcionálním programováním, které je ovšem v ryzí podobě velmi ortodoxní přístup, jak izolovat logiku od stavu a není v současnosti příliš populárním paradigmatem v programování. Tvůrci moderních programovacích jazyků, ale berou v potaz výhody a usnadnění, které nabízejí deklarativní paradigmata, potažmo funkcionální přístupy k řešení algoritmických problémů. V současnosti se stává populárním aplikovat je v odlehčenější formě do objektově orientovaných jazyků. Na toto reagovala i Java a ve verzi 8 z roku 2014 přidala poměrně inovativní možnosti do jazyka v podobě lambda výrazů, referencí na metody, streamové zpracování. Některé jazyky jsou více otevřené a podporují funkcionální přístupy v rámci multi-paradigmatického prostředí, což je kompromis oproti čistě funkcionálním jazykům. Současným trendem je zvyšování expresivity jazyků. Mezi roky 2013 a 2015 nastala renesance funkcionálních jazyků díky jejich imutabilitě a přístupu k paralelizaci, na JVM platformě zažil vzestup jazyk Scala, avšak neujal se tak široce. Některé frameworky pro vývoj webových aplikací zvyšují expresivitu do té míry, že se z nich postupně stávají spíše DSL jazyky, které řeší problémy na vyšší úrovni abstrakce a odstiňují vývojáře co nejvíce od technologických detailů tak aby se mohli plně soustředit na řešení vlastní aplikační logiky.

Nejen samotné vývojové technologie ovlivňují trendy, inovace zaznamenalo také prostředí do, kterého jsou serverové aplikace nasazovány a v niž následně běží. V prvopočátcích, kdy se aplikace nasazovali přímo na fyzické stroje manuálně a s postupem času také automaticky, trvalo nasazení aplikace poměrně dlouho dobu a bylo značně komplikované. Postupně se prostředí začala více odstiňovat od fyzického stroje virtualizací, kdy na jednom či více fyzických serverech běželi virtuální instance serverů. V tomto desetiletí se začal využíval běh přímo v kontejnerech, které umožnili odstínění až na úroveň jednotlivé aplikace. Kontejnerizace řeší problém s konfigurací napříč virtuálními instancemi. Díky tomu je možné aplikace nasazovat automaticky v jednotkách minut, nezávisle na prostředí

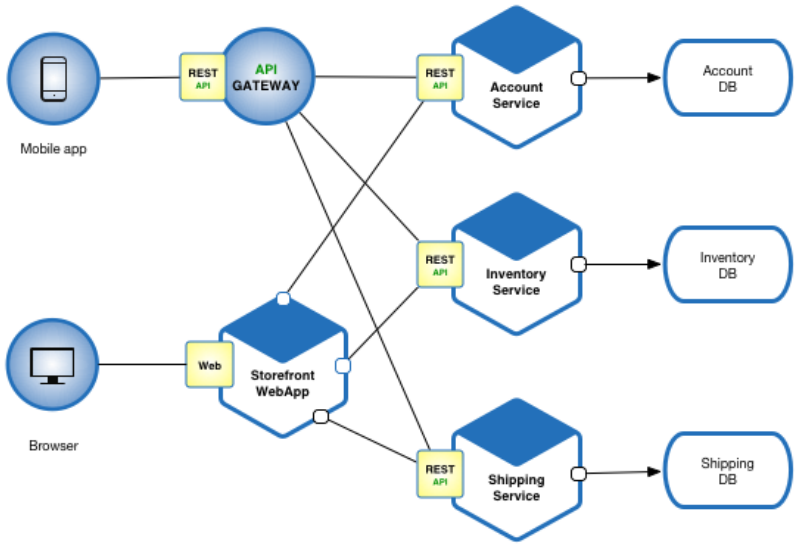
a jeho nastavení. Jelikož aplikace využívá jednotně nakonfigurovaný kontejner, ve kterém je odstíněna od složitého nastavování. Nejčastěji využívanou aplikací pro kontejnerizaci je Docker.

Ve světě serverových aplikacích se poměrně často začíná uplatňovat trend serverless přístupu, kdy jsou odstíněné hardwarové servery a kdy aplikace běží v cloudu. Tedy infrastrukturu (IaaS), platformu (PaaS), nebo funkce (FaaS) zajišťuje poskytovatel služby a uživatel platí přímo za čas běhu aplikace, případné množství využitých zdrojů. V dnešní době je na trhu několik leaderů, Amazon se svým AWS, Google s Cloud Platform a Microsoft Azure. Serverless aplikace jsou většinou stavěny jako kompozice mikroslužeb či funkcí, avšak některé aplikace jsou i monolitické.




Obr. 3.1: Evoluce prostředí pro serverové aplikace (Buchholz, 2017)

Microservices architektura je jedním ze současných trendů při vývoji serverových aplikací, kdy místo velké monolitické aplikace, vytvoříme kompozici více menších autonomních služeb, které umístíme do kontejnerů a poté mezi sebou propojíme. Vychází z konceptu SOA, tedy servisně orientované architektury. Jednotlivé mikro služby se skládají do kompozic a vzájemně mezi sebou komunikují. Pro orchestraci se dnes využívá populární nástroj Kubernetes, který dokáže zajistit kompletní management kontejnerizovaných aplikací včetně jejich monitoringu. Tuto architekturu podporují mnohé frameworky pro tvorbu serverových aplikací a poskytují také svoje proprietární nástroje pro orchestraci služeb například Spring Cloud a Data Flow. Nasazení mikroslužeb snižuje složitost systému, umožňuje snadněji upravovat a testovat aplikace. (Richardson, 2018) Dále usnadňuje škálovatelnost vytížených služeb a také podporují principy continuous delivery a principy DevOps, které se v současnosti stávají velmi populární. (Balalaie, Heydarnoori, Jamshidi, 2018, s. 42-52; Chen, 2018) Na druhou stranu sebou přináší i všechny nevýhody distribuovaných systémů, pro vývojáře se jedná o složitější opravy chyb a jejich detekci, například při debugování. Na zvyšující se oblibu JVM micro-frameworků zareagovalo i nejpoužívanější IDE mezi Java vývojáři Idea IntelliJ a ve verzi 2019.1.3, přidává podporu pro mnoho z nich, mimo jiné i pro Micronaut. (Paraschiv, 2018; Kalyuzhnaya, 2019)




Obr. 3.2: Ukázka architektury pomocí microservices (Richardson, 2018)

# 4 Webové frameworky

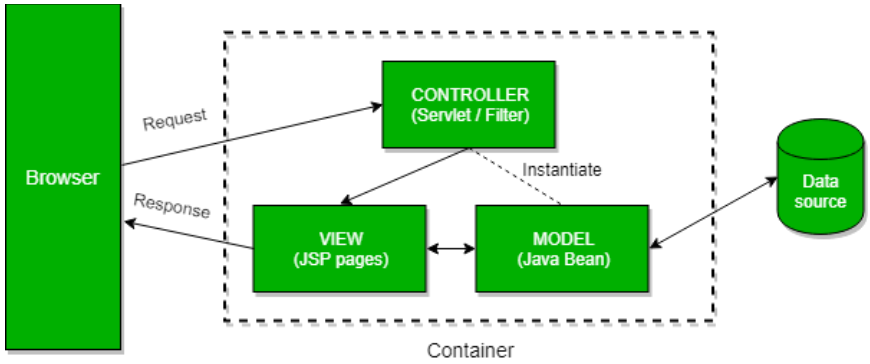
Framework je terminus technicus pro sadu softwarových nástrojů distribuovaných formou komplexního aplikačního rámce, který poskytuje vývojáři podporu při vývoji aplikací. Obecný význam existence frameworků je poskytnutí řešení pro typické problémy v oblastech, pro které je určen. Vývojář se díky nim může soustředit pouze na řešení problému na business úrovni a je odstíněn od implementace rutinních a opakujících se technických součástí aplikace, které jsou řešeny podle zavedených best-practices v dané oblasti. Díky tomu je při použití frameworku možné vývojáře snadno usměrňovat, vést jejich vývoj do stanovené architektury, odpadá velká část kontroly k dodržování best-practicies, díky defaultní implementaci, a to vše vede k celkovému zlepšení kvality aplikace jako celku. Avšak framework vývojáře zcela nezprošťuje od řešení technických úskalí, spíše poskytuje vyšší úroveň abstrakce, či DSL[1] nad danou technickou doménou. V současné době jsou frameworky poměrně pokročilé a vyvíjeny profesionálními týmy, tudíž je zaručena efektivita a kvalita kódu ve kterém je framework napsán. Poměrně mnoho článků a diskuzí se zabývá použitím frameworků při vývoji aplikací. Nejčastější argumenty pro použití frameworku, kromě těch výše zmíněných je zvláště bezpečnost, která je zaručena velkým množstvím vývojářů, kteří se podílejí na vývoji a testování. Do testování je v podstatě zahrnuta celá komunita uživatelů, tudíž u vývojově pokročilého frameworku se již zásadní chyby téměř neobjevují a pokud ano, jedná se o drobné defekty, které jsou velmi rychle odhaleny a opravovány. Oproti proprietárním řešení poskytují frameworky většinou kvalitní dokumentací a pro nově příchozí vývojáře je výrazně lepší učící křivka, navíc případně mohou využít svoje přechozí znalosti, jelikož frameworky jsou stavěny poměrně obecně aby v dané oblasti mohli pokrýt co největší část potřeb. V neposlední řadě je výhodou časová úspora téměř při všech fází vývoje aplikace, která je při dnešních nákladech poměrně zásadní. Nevýhody frameworků je jejich obecné zaměření, kdy některé specifické potřeby při vývoji nelze jednoduše provést, například díky jeho omezením. Pro některé druhy řešení může framework vykazovat nedostatečný výkon, který nesplňuje požadavky. V neposlední řadě se jedná o určitou formu softwarového vendor lock-in[2], protože změna frameworku většinou znamená zásadní úpravu kódu aplikace.

Webový framework, jak je z názvu patrné je určen pro ulehčení tvorby webových aplikací na straně serveru. První webové frameworky se začali objevovat kolem roku 1995. Pro Javu se populární webové frameworky, které jsou využívány i v současné době začali objevovat až po novém miléniu. Webové frameworky primárně poskytují rozhraní mezi serverovou stranou aplikace a klienty, kterými jsou v současnosti nejčastěji různí front-end konzumenti (web, mobilní zařízení) reprezentující získaná data uživateli. Většina frameworků je

[1] Zkratka pro doménově specificky jazyk. Tento druh programovacího jazyka se úzce zaměřuje se na konkrétní problémovou doménu.

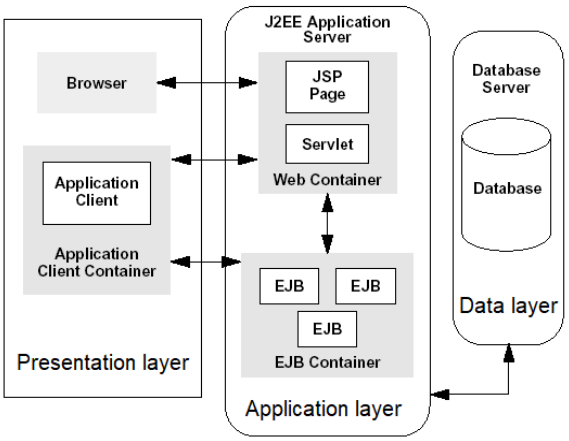
[2] Označení pro chování dodavatele a vlastností jeho produktu, které činní zákazníka závislým na tomto dodavateli a produktech, kdy změna na jiného dodavatele je značně náročná a komplikovaná.

vystavěna na jednotné architektuře, do které komponuje vývojář svoje řešení a která dotváří tzv. development guidelines a podporuje architektonické best-practicies. V současnosti se nejvíce využívá model MVC (Model-View-Controller) a jeho variace. Jedná se o léty ověřený architektonický vzor, jehož hlavním významem je oddělit busines data a logiku (Model) od jejich prezentace (View) a zpracování vstupů (Controller). Od tohoto vzoru jsou odvozeny další, které modifikují vzor do jiných kontextů např. MVP, MVVM. Tento vzor patří mez tzv. push-based architektury, nebo také nazývané jako action-based. To znamená, že framework reaguje na akce, které zpracuje a následně zasílá (push) do view aby byly výsledky prezentovány.




Obr. 4.1: Ukázka architektury MVC v Java EE
(Spring MVC Interview Questions, 2018)

Většina webových frameworku dobře zapadá do tzv. třívrstvé architektury, který separuje celou aplikaci do nezávislých vrstev, které mají slabou vazbu tzv. loose-coupled. První vrstva je prezentační (front-end), která se stará o zobrazování dat. Druhá je aplikační vrstva (back-end), ve které se nachází business logika. A poslední vrstvou je datová (RDBMS), která uchovává data a tvoří abstrakci pro aplikační vrstvu. Každá vrstva komunikuje pouze se svými sousedy. Tento model je poměrně podobný výše uvedenému vzoru MVC, avšak tento vrstevnatý model řeší uspořádání celé aplikace. Webový framework se nachází v aplikační vrstvě a vzor MVC, řeší její vnitřní strukturu. Lépe celou situaci dokresluje ilustrace na obrázku níže.



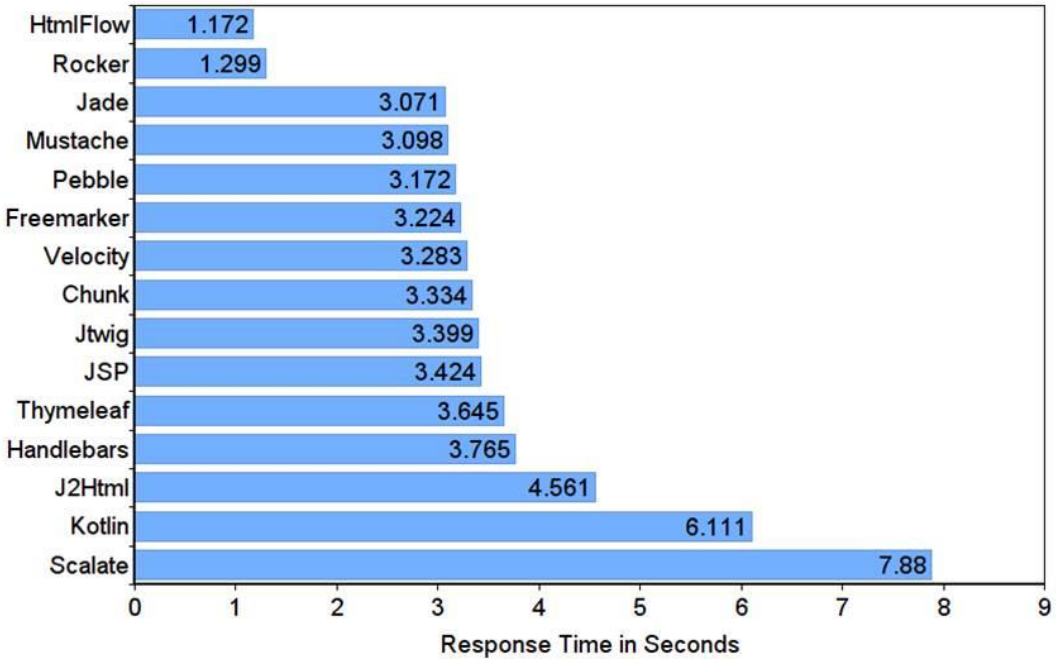

Obr. 4.2: Třívrstvá aplikace v Java EE

Jak bylo výše zmíněno, webové frameworky dnes neposkytují pouze nástroje pro tvorbu rozhraní, ale zahrnují velkou škálu dalších modulů, které pokrývají téměř celou doménu vývoje serverových aplikací. Mezi nejčastější podporovaná rozšíření patří šablonovací knihovny, které usnadňují tvorbu statických HTML stránek na straně serveru, avšak v dnešní době většina knihoven umožňuje doplnění o AJAX, který podporuje tvorbu dynamických HTML stránek, díky využití asynchronního JavaScriptu, který běží na pozadí v prohlížeči a dodává webovým stránkám dynamiku bez nutnosti jejich znovu načítání ze serveru. První šablonovací jazyk pro Javu se objevil v roce 1999 jednalo se o JSP (Java Server Page), šlo o abstrakci nad servlety, do kterých jsou stránky překládány za běhu. Všeobecně se moc velké obliby nedočkali. V zápětí po vydání JSP se objevila další technologie JSF (Java Server Faces), která poskytovala celý MVC framework pro tvorbu webových aplikací v Javě. Pro úplnost je nutné uvést, že jazyk Kotlin nelze využít pro skripty uvnitř JSP stránek, avšak lze ho použít pro JSF. Daleko větší oblibu získaly šablonovací enginy třetích stran. Mezi nejznámější patří FreeMarker, Thymeleaf, které lze použít pro Kotlin i Javu. Thymeleaf šablony jsou oproti FreeMarkeru plně validní HTML stránky. Pro jazyk Kotlin v současné době neexistuje mnoho nativních variant. Kvalitní knihovnu, která stojí za pozornost je Kotlinx.html. Je vyvíjena přímo vývojáři Kotlinu. Oproti přechozím variantám má velkou výhodu a to tím, že umožnuje psát šablony ve stylu a syntaxi Kotlinu, cože žádná jiná alternativa nenabízí. Pro srovnání jsem vyhledal výkonnostní test, který srovnává šablonovací enginy. Test byl proveden Luisem Durate a výsledek publikoval na webu Dzone. (Modern Type-Safe Template Engines, 2018) Autor článku je současně vývojářem šablonovacího enginu HtmlFlow, tudíž se dá předpokládat vysoká relevantnost testu. Autor se zaměřil na slabé stránky současných řešení, kde zmiňuje chybějící systém validací při generování HTML stránek, zvláště absenci statické validace již při kompilaci, namísto upozornění na chybu až při běhu aplikace, které jsou provázeny neočekávanými pády. Další nevýhodu spatřuje v nedostatečném výkonu, složité syntaxi, která vývojáře zbytečně rozptyluje a omezenou flexibilitu, kdy je u většiny frameworku poskytována pouze velmi omezená paleta možností pro kontrolu toku dat v šabloně a provádění komplexních akcí v šabloně.

Výpis 4.1: Ukázka tvorby HTML šablon pomocí Kotlinx.HTML (vlastní zpracování)

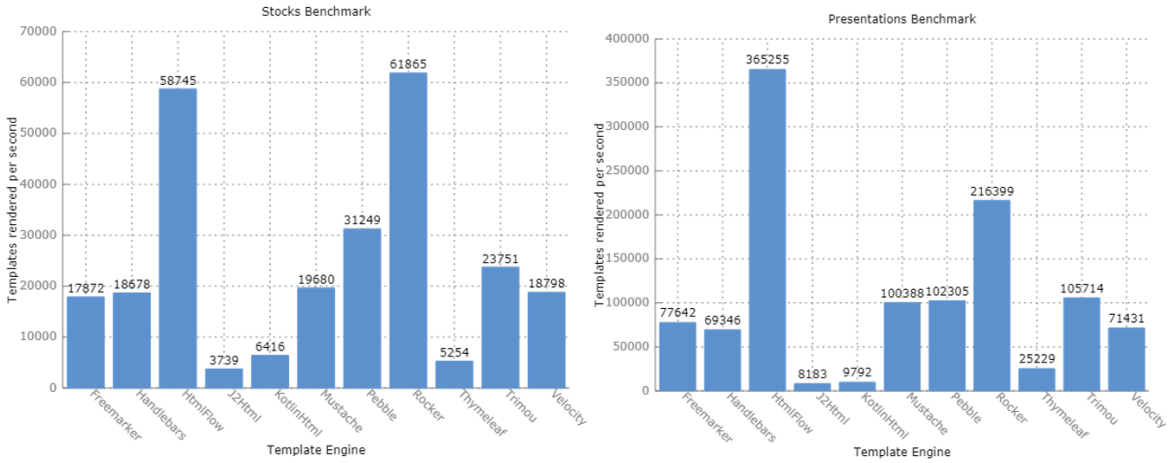
```
class HtmlTemplates {
    companion object {
        fun studentTemplate(student: Student) = generateStudentTemplate(student)
    }

  private fun generateStudentTemplate(student: Student)= createHTMLDocument().html {body {
    ul {
      li { student.name }
      li { student.number }
   }
  }}.serialize(false)
}
```

Knihovna Kotlinx.HTML značně snižuje nutnost použití textu, pro vytvoření HTML stránky, kdy je celá šablona psána přímo v Kotlinu. Knihovna je plně validní s HTML 5 a validuje správnou syntaxi, která je prováděna při kompilaci. Drobnou nevýhodou je absence validace atributů, které mohou nabývat různých hodnot a ošetření je ponecháno na vrub vývojáři. Pro testování výkonu byly použity dva nejoblíbenější testy pro šablonovací enginy. První je Spring test, který testuje engin pod náporem 25 paralelních klientů, kteří dohromady odešlou 25 tisíc requestů. (Comparing Template engines for Spring MVC, 2018) Měří se celkový čas, který zabere obsloužení tohoto množství.




Obr. 4.3: Výsledek Spring testu
(Modern Type-Safe Template Engines, 2018)

Druhý test spočívá v rychlosti vykreslování šablon, využity jsou dvě. První šablona Stocks je více zaměřena na tzv. binding (napojení datových položek do šablony a naopak) a má také více dat (20 objektů). Druhá šablona Presentation má pouze 10 objektů. První šablona tedy více ověří práci se řetězci a více prověří výkon díky mnoha voláním metod.




Obr. 4.4: Výsledky testu vykreslování
(Modern Type-Safe Template Engines, 2018)

V těchto testech nevyšla, co se týká výkonu knihovna Kotlinx.HTML nejlépe, avšak je třeba brát v potaz, že oproti vítězi ve funkčnosti zaostávala pouze v absenci validace atributů a na rozdíl od všech testovaných enginů je plně v duchu jazyka Kotlin, což velmi usnadní vývoj a vývojář se nemusí rozptylovat syntaxí a sémantikou šablonovacího enginu. Avšak pokud přechozí argumenty nedosahují kýžených přínosů a výkon je klíčovou prioritou je nejspíše vhodné poohlédnout se po jiném řešení a použít například, obecné šablonovací enginy i za cenu kompromisu v podobě menší palety funkčnosti a zvýšené složitosti vývoje komplexnějších šablon.

Při tvorbě webových aplikací se často potýkáme s problémy v rychlostí vracení odpovědí (HTTP reponses) na příchozí zprávy klienta (HTTP request). Zvláště pokud pracujeme s rozsáhlou databází, provádíme mnoho volání vzdálených služeb, či složité výpočty je efektivní znovupoužití vynaloženého úsilí na získání dat. V principu lze říci, že frekventované služby využívá mnoho klientů a mnoho z nich tvoří velké množství duplicitních požadavků. Je tedy nasnadě uchovávat výsledky v paměti, která umožňuje rychlé čtení a zápis. Díky tomu můžeme odbavit zprávy v klienta už v servisní vrstvě, aniž bychom využívaly vrstvu pro získávání dat (repository, service-client). Tato funkčnost se všeobecně nazývá cachování a paměť pro uložení cache. () Cache je oproti standardnímu uložišti určena na dočasné uchování dat. Data jsou v ní často uložena v upravené struktuře, nejčastěji ve struktuře klíč-hodnota, kdy uchovávaná hodnota může být určitá projekce modelu, či pouze primitivní datový typ. Oproti standardní paměti zvládne obsloužit větší množství požadavků ve velmi krátkém čase. Avšak oproti standardní paměti bývá značně drahá při velkých objemech dat, a navíc ke standardní paměti často přistupujeme přes mnoho kódu a infrastruktury, čímž vždy ztrácí oproti operační paměti. I v případě že je disk napojen přímo ve fyzickém stroji, díky rozdílným rychlostem sběrnice, kdy DRAM je až desetkrát rychlejší než připojení disku a dokáže přenést až 20 GB/s[3]. Pro vývojáře vystávají zásadní otázky a problémy, jak navrhnout algoritmus který bude řídit cachování, jež musí řešit integritu uložených dat v čase a uplatňovat pravidla pro uchovávání záznamů v paměti. Nejčastější pravidla pro uchovávání záznamů, které se v praxi používají jsou uchování dle četnosti použití (LFU), kdy jsou nejméně používané záznamy odstraňovány. Dále se používá mazání záznamů dle jejich posledního použití (LRU/MRU), kdy se používá buď mazání nejdéle použitých, či naopak mazání nejnověji použitých, tohoto se nejčastěji využívá v systémech, kde jsou nejstarší položky nejvíce přistupovány. Tyto pravidla se často aplikují ve vztahu k době života záznamu (TTL) či s počítadlem konkrétních akcí (např. přístupy, změny), v případě, že je nechceme limitovat časem. Velkou výzvou pro vývojáře je navrhnout robustní systém, který hlídá integritu dat. Všeobecně se používají tři politiky pro řízení integrity dat. První je přístup write-trough, kdy při každém zápisu dat do paměti je rovnou uložíme do cache, write-around obchází vkládání do cache při zápisu dat a záznam do cache vloží až při jeho prvním načtením z paměti, poslední způsob write-back při kterém jsou data zapsána do cache a až později se z cache uloží do trvalé paměti (asynchronní přístup), díky tomuto je zajištěn rychlejší zápis než v případě write-trough. Dále je nutné zajistit správu již neaktuálních záznamů, které byly upraveny a jsou nekonzistentní (např.

[3] https://superuser.com/questions/1173675/how-much-faster-is-memory-ram-compared-to-ssd-for-random-access

aktualizovány, nebo smazány). Všechny tyto způsoby však neřeší modifikaci záznamů z jiného místa v systém, ale v tomto případě se jedná o řešení na architektonické úrovni celého systému a přesahuje rozsah jedné webové aplikace. Pro rutinní použití ve webové aplikaci poskytují frameworky podporu pro cachování. Většina frameworků implementuje jednoduchá pravidla a hlídá integritu pouze v rámci záznamů v cache. Některé umožňují zaintegrovat sofistikovanější systémy třetích stran, které umožňují využití složitějších pravidel, avšak s nativním přístupem přes webový framework.

Moderní frameworky nabízejí možnost generování projektu, případně i celé vnitřní struktury a jednotlivých komponent automaticky, čímž dokáží šetřit vývojový čas, který je nutný při započetí projektu. Ušetří nás zvláště složitého komponování a konfigurování modulů frameworku a dalších knihoven, což bývá problematické, zvláště u rozsáhlých monolitických frameworků např. Spring. V případě tvorby microservices, kdy pro každou službu zakládáme projekt samostatně, může tato funkcionalita ušetřit nezanedbatelné množství času.

Většina z frameworků také usnadňuje přístup k datům, které jsou uloženy v jiném systému. Nejčastěji jsou to databáze, či jiné webové služby. Pro databáze webové frameworky nabízí API, které odstiňuje vývojáře od přímé komunikace s ní, tyto prostředky se liší v úrovni abstrakce přístupu k datům. Nejvíce je vývojář odstíněn při použití technologií založených na ORM[4], které mapují položky tabulek z databáze na entitní třídy a naopak. Při jejich použití je vývojář plně odstíněn od práce s databází, avšak nevýhodou je nižší možnost kontroly v komunikací s databází, byť jsou dnešní ORM nástroje velmi dobře přizpůsobitelné. Mezi nejčastější podporované implementace pro ORM patří Hibernate, některé frameworky dokonce nabízejí implementaci pomocí vlastních řešení. Nižší úrovní je programová podpora práce s SQL, která umožňuje manuální tvorbu operací, avšak přináší jazykové konstrukty a často i validaci. Mimo přímé podpory pro práci s databází poskytují frameworky podporu transakčního zpracování a databázových migrací.

Pro jazyk Kotlin momentálně neexistuje proprietární náhrada za Hibernate. Existuje několik pokusů o implementaci ORM přímo pro Kotlin. Jedním z pokusů, který stojí za zmínku je knihovna Ktorm, avšak stále je oproti Hibernate v poměrně ranném vývojovém stádiu, navíc je více zaměřena jako SQL DSL knihovna, kdy ORM je spíše okrajová záležitost. Všeobecně má Kotlin více knihoven, které se zabývají právě poskytnutím DSL pro relační databáze využívající jazyk SQL. Nejzralejší knihovnou a přímo od tvůrců Kotlinu je knihovna Exposed. Na základě vlastního subjektivního hodnocení mi přijde, že tato knihovna oproti Ktormu pracuje na nižší úrovni abstrakce a vyžaduje vyšší expresivitu ve zdrojovém kódu od vývojáře. Ktorm má navíc větší škálu funkčnosti a příjemnější syntaxi. Ktorm má potenciál značně přesáhnout Exposed, jelikož udělal obrovský pokrok, od první beta verze, která vyšla v prosinci roku 2018 a nyní o rok později je ve verzi 2.5, což svědčí o velké aktivitě autorů. Oproti tomu Exposed v současnosti ani není ve stabilní verzi 1.0, byť jeho vývoj probíhá od roku 2016.

[4] https://www.techopedia.com/definition/24200/object-relational-mapping--orm

Framework mají často podporu pro automatické mapování JSON z tříd a naopak. Čímž je vývojář odstíněn od zpracování a často jen deklaruje chování mapperu pomocí anotací nad položkami, či v konfiguračním souboru pro obecné nastavení chování. Toto je velmi podstatná vlastnost, kterou by měl framework umožňovat, jelikož při nejrozšířenější komunikaci pomocí REST se používají téměř výhradně data formátovaná dle JSON specifikace, jenž vytlačila XML formáty pro popis přenášených dat.

Aspektově orientované programování a scheduling jsou funkce, které frameworky vývojářům nabízí a jež usnadňují implementaci enterprise aplikací, kdy často nestačí objektové paradigma, nebo není úplně vhodné pro řešení pro business pravidel a technických řešení, která se prolínají napříč aplikací. Scheduling je funkce pro usnadnění automatických akcí, které jsou vykonávány v definovaném čase, často bývají implementovány ve stylu Linux nástroje Cron. Kdy vývojář pouze označí metodu a definuje u ní intervaly kdy se má spouštět, některé frameworky často umožňují další předdefinovaná pravidla. Výhodou je že vývojář se nemusí starat o implementaci samotného triggerování funkce, kterou přebírá framework.

Velmi významným dílem se webové frameworky podílejí také na zabezpečení aplikace. Většina z nich podporuje integraci využívaných autentizačních standardů jako je Basic, OAtuh2. Autorizaci poskytují pomocí proprietárního řešení v kombinaci s autentizačním standardem, nebo přímo integrují autorizační standard jako je například OpenId. Mimo jiné frameworky v rámci zabezpečení poskytují správu veřejné publikace a přístupy ke statickým zdrojovým souborům.

# Závěr

V teoretické části práce byla představena v první kapitole historie vývoje serverových aplikací. Ve druhé byly popsány současné trendy, kterými se vývoj těchto aplikací ubírá. Třetí kapitola poskytuje informace o využití frameworků v současné době, informace pocházejí ze syntézy více zdrojů. Tato část sloužila jako podkladová informace pro výběr technologie, kterou se má v současnosti smysl více zabývat.

Praktická část začíná kapitolou, ve které se rozebírá celá problematika webových frameworků a v této kapitole jsou také vyhrazeny všechny aspekty, které se dále v práci ověřovali a hodnotily. Ve další kapitole se řeší samotný výběr frameworků k testování. Poté následuje kapitola, ve které jsou vybrána a zdůvodněna kritéria podle kterých budou frameworky hodnoceny a potažmo co se v rámci každého kritéria hodnotí. Dále pak následují kapitoly, které se věnují podrobným popisem a hodnocením jednotlivých frameworků. Po této části následuje kapitola, ve které jsou popsány způsoby výkonnostního testování a v jejím závěru jsou uvedeny výsledky testování frameworků. Poslední kapitola praktické části se zabývá vyhodnocením všech výsledků a shrnutí nabytých poznatků.

Na základě dosažených výsledků, lze konstatovat, že práce naplnila vytyčené cíle. Podařilo se zmapovat a ohodnotit současné možnosti, které jsou při volbě jazyka Kotlin pro serverový vývoj k dispozici. Tato práce může být dále rozšířena, či upravena jednak s ohledem na rozvoj testovaných frameworku, tak i pro rozšíření o další nové frameworky. Upraveny mohou být váhy kritérií a tím pádem je možné přizpůsobit hodnocení frameworků pro individuální potřeby čtenáře

# Použitá literatura


Balalaie, Armin; Heydarnoori, Abbas; Jamshidi, Pooyan (May 2016). "Microservices Architecture Enables DevOps: Migration to a Cloud-Native Architecture". IEEE Software: 42–52.

Benchmarking Scala Against Java. Dzone.com [online]. 2012, 2012 [cit. 2019-11-17]. Dostupné z: https://dzone.com/articles/benchmarking-scala-against

BUCHHOLZ, Scott. The Many Potential Benefits of Serverless Computing. *Deloitte.com* [online]. 2017 [cit. 2019-11-17]. Dostupné z: https://deloitte.wsj.com/cio/2017/11/09/serverless-computings-many-potential-benefits/

Cache (computing). Techtarget.com [online]. [cit. 2019-11-17]. Dostupné z: https://searchstorage.techtarget.com/definition/cache

*Comparing Template engines for Spring MVC* [online]. 2018 [cit. 2019-11-17]. Dostupné z: https://github.com/xmlet/spring-comparing-template-engines#how-were-the-results-measured

Chen, Lianping (2018). Microservices: Architecting for Continuous Delivery and DevOps. The IEEE International Conference on Software Architecture (ICSA 2018).

DANILOV, Anton. Analyze programovacího jazyka Kotlin, jeho srovnání s jazykem Java 8 a praktické využití při vývoji na platformě Android [online]. Praha, 2017 [cit. 2019-06-11]. Dostupné z: https://theses.cz/id/u5c85i. Diplomová práce. Vysoká škola ekonomická v Praze. Vedoucí práce Jarmila Pavlíčková.

Developer Survey Results 2019. Stack Overflow [online]. [cit. 2019-11-17]. Dostupné z: https://insights.stackoverflow.com/survey/2019

Developer Survey Results 2018. Stack Overflow [online]. [cit. 2019-11-17]. Dostupné z: https://insights.stackoverflow.com/survey/2018

Dragos-Paul a Adam ALTAR. Designing an MVC Model for Rapid Web Application Development. 24th DAAAM International Symposium on Intelligent Manufacturing and Automation, 2013 [online]. 2013, 1-8 [cit. 2019-11-17]. Dostupné z: https://www.sciencedirect.com/science/article/pii/S187770581400352X?via%3Dihub anticscholar.org/8353/e9b61783ef7d3a06bcca207d91b624c45c6e.pdf

ERB, Benjamin. Concurrent Programming for Scalable Web Architectures. Ulm, 2012. Diploma thesis. Ulm university.

Forbes.com [online]. 2018, 2018 [cit. 2019-11-17]. Dostupné z: https://www.forbes.com/sites/forbestechcouncil/2018/10/05/15-predictions-for-the-next-big-thing-in-software-development/#30a935e0522f

FRANKEL, Nicolas. The Rise and Fall of JVM Languages. *Dzone.com* [online]. 2017 [cit. 2019-11-17]. Dostupné z: https://dzone.com/articles/the-rise-and-fall-of-jvm-languages

FREDERICKSON, Ben. Ranking Programming Languages by GitHub Users. Benfrederickson.com [online]. [cit. 2019-11-17]. Dostupné z: https://www.benfrederickson.com/ranking-programming-languages-by-github-users

Inside Vert.x. Comparison with Node.js. Cubrid.org [online]. 2017 [cit. 2019-11-17]. Dostupné z: https://www.cubrid.org/blog/inside-vertx-comparison-with-nodejs

JAROŠ, Jan. Použitelnost jazyka Kotlin při vývoji mobilních aplikací [online]. Praha, 2018 [cit. 2019-06-11]. Dostupné z: https://theses.cz/id/fps0ec. Bakalářská práce. Vysoká škola ekonomická v Praze. Vedoucí práce Rudolf Pecinovský.

Javalin [online]. [cit. 2019-11-17]. Dostupné z: https://javalin.io

Java magazine: march-april [online]. Oracle, 2019 [cit. 2019-09-28]. Dostupné z: https://blogs.oracle.com/javamagazine

KALININ, Denis. Modern Web Development with Kotlin [online]. Leanpub [cit. 2019-11-17]. Dostupné z: https://leanpub.com/modern-web-development-with-kotlin

KALYUZHNAYA, Zlata. The State of Java in 2018. *Jetbrains.com* [online]. 2018 [cit. 2019-11-17]. Dostupné z: https://blog.jetbrains.com/idea/2019/08/whats-next-intellij-idea-2019-3-roadmap/#comment-499765

KOZÁKOVÁ, Eva. Specifikace server-side technologií, jejich implementace a porovnání [online]. Hradec Králové, 2019 [cit. 2019-06-11]. Dostupné z: <https://theses.cz/id/u461g4/>. Diplomová práce. Univerzita Hradec Králové, Fakulta informatiky a managementu. Vedoucí práce doc. Ing. Filip Malý, Ph.D..

KUBICA, Tomáš. Serverless - existují aplikace bez serverů? Techbeacon.com [online]. 2017, 2017 [cit. 2019-11-17]. Dostupné z: https://www.tomaskubica.cz/post/2017/serverless-existuji-aplikace-bez-serveru

KUCHARENKO, Igor. Mastering High Performance with Kotlin: Overcome performance difficulties in Kotlin with a range of exciting techniques and solutions. 1. Packt, 2018. ISBN 978-1788998017.

Ktor [online]. [cit. 2019-11-17]. Dostupné z: https://ktor.io

Ktor for fast server prototyping. Kotlin-academy.com [online]. 2019 [cit. 2019-11-17]. Dostupné z: https://blog.kotlin-academy.com/ktor-for-fast-server-prototyping-6e7c6d2ec296

LAVENA, Zoe. Understanding the Background of Microservice Architecture. Meduim.com [online]. 2019, 2019 [cit. 2019-11-17]. Dostupné z: https://medium.com/sw-d/microservice-4609d656ccf8

MAI, Duc Anh. Transformation of Android Mobile App from Java to Kotlin [online]. Praha, 2018 [cit. 2019-06-11]. Dostupné z: https://theses.cz/id/fukc4l. Diplomová práce. Vysoká škola ekonomická v Praze. Vedoucí práce Alena Buchalcevová.

MAJEED, Abdul a Ibtisam RAUF. MVC Architecture: A Detailed Insight to the Modern Web Applications Development. Peer Review Journal of Solar & Photoenergy Systems [online]. 2018, 1-7 [cit. 2019-11-17]. Dostupné z: https://pdfs.semanticscholar.org/8353/e9b61783ef7d3a06bcca207d91b624c45c6e.pdf

Major Programming Trends to Prepare for in 2019. Hackernoon.com [online]. 2018, 2018 [cit. 2019-11-17]. Dostupné z: https://hackernoon.com/major-programming-trends-to-prepare-for-in-2019-169987cc75f4

Micronaut [online]. [cit. 2019-11-17]. Dostupné z: https://micronaut.io

Micronaut blog [online]. [cit. 2019-11-17]. Dostupné z: https://objectcomputing.com/news/tag/micronaut

Modern Type-Safe Template Engines. Dzone.com [online]. 2018 [cit. 2019-11-17]. Dostupné z: https://dzone.com/articles/modern-type-safe-template-engines

MOTTL, Jan. Moderní vývoj aplikací pro platformu Android [online]. Praha, 2018 [cit. 2019-06-11]. Dostupné z: https://theses.cz/id/tk66dq. Diplomová práce. Vysoká škola ekonomická v Praze. Vedoucí práce Jarmila Pavlíčková.

MOSELEY, Ben a Peter MARKS. [online]. 2006 [cit. 2019-11-17]. Dostupné z: http://curtclifton.net/papers/MoseleyMarks06a.pdf

NEZDARA, Vojtěch. Vývoj moderních webových aplikací [online]. Praha, 2016 [cit. 2019-06-11]. Dostupné z: <https://theses.cz/id/uqff1w/>. Diplomová práce. Vysoká škola ekonomická v Praze. Vedoucí práce Alena Buchalcevová.

Non-blocking I/O using Servlet 3.1: Scalable applications using Java EE 7 (TOTD #188). Oracle.com [online]. 2019, 2012 [cit. 2019-11-17]. Dostupné z: https://blogs.oracle.com/arungupta/non-blocking-io-using-servlet-31:-scalable-applications-using-java-ee-7-totd-188

PARASCHIV, Eugen. The State of Java in 2018. *Baeldung.com* [online]. 2018 [cit. 2019-11-17]. Dostupné z: https://microservices.io/patterns/microservices.html

PLESSIS, Corneil. *Performance Comparison JVM Languages* [online]. 2017 [cit. 2019-11-17]. Dostupné z: https://www.slideshare.net/CorneilduPlessis/performance-comparison-jvm-languages

Project Reactor [online]. [cit. 2019-11-17]. Dostupné z: https://projectreactor.io

PRONSCHINSKE, Mitch. 9 code and framework trends to watch in 2018. Techbeacon.com [online]. 2018, 2018 [cit. 2019-11-17]. Dostupné z: https://techbeacon.com/app-dev-testing/9-code-framework-trends-watch-2018

PYPL PopularitY of Programming Language. Pypl.github.io [online]. [cit. 2019-11-17]. Dostupné z: http://pypl.github.io/PYPL.html

RAO K, Raghavendra. Kotlin for Enterprise Applications using Java EE: Develop, test, and troubleshoot enterprise applications and microservices with Kotlin and Java EE. 1. Birmingham, United Kingdom: Packt, 2018. ISBN 9781788997270.

Register for the Open Source APM for an Event Management Web Application webinar todayRegister Now Trends in Enterprise Software Development – 2019 and Beyond. Dzone.com [online]. 2019, 2019 [cit. 2019-11-17]. Dostupné z: https://dzone.com/articles/trends-in-enterprise-software-development-2019-and

RELYEA, Rebecca. Choosing a Reactive Framework for the JVM. Medium.com [online]. 2019 [cit. 2019-11-17]. Dostupné z: https://medium.com/priceline-labs/choosing-a-reactive-framework-for-the-jvm-ec66f6cde552

RFC 5: Decode Encode Language (DEL). *Ietf.org* [online]. 2005, 1969 [cit. 2019-11-17]. Dostupné z: https://tools.ietf.org/html/rfc5

RICHARDSON, Chris. Pattern: Microservice Architecture. *Microservices.io* [online]. 2018 [cit. 2019-11-17]. Dostupné z: https://microservices.io/patterns/microservices.html

ROBERTS, Mike. Serverless Architectures. Martinfowler.com [online]. 2017, 2018 [cit. 2019-11-17]. Dostupné z: https://martinfowler.com/articles/serverless.html

RYABTSEV, Alexander. Web Frameworks: How To Get Started. Djangostars.com [online]. 2017 [cit. 2019-11-17]. Dostupné z: https://djangostars.com/blog/what-is-a-web-framework

SAMUEL, Stephen a Stefan BOCUTIU. Programming Kotlin: Get to grips quickly with the best Java alternative. 1. Packt, 2017. ISBN 978-1787126367.

Serverless Architectures vs. Containerized Architectures. Dzone.com [online]. 2019, 2019 [cit. 2019-11-17]. Dostupné z: https://dzone.com/articles/serverless-architectures-against-containerized-arc?edition=491306&utm_source=Daily%20Digest&utm_medium=email&utm_campaign=Daily%20Digest%202019-06-25

SCHWERMER, Patrik. Performance Evaluation of Kotlin and Java on Android Runtime. Stockholm, Sweden, 2018. Master thesis. Nada. Vedoucí práce Elena Troubitsyna.

SLESAREV, Alexander. *Benchmarks of different languages* [online]. 2019 [cit. 2019-11-17]. Dostupné z: https://github.com/kostya/benchmarks

SOSHIN, Alexey. Hands-on Design Patterns with Kotlin: Build scalable applications using traditional, reactive, and concurrent design patterns in Kotlin. 1. Packt, 2018. ISBN 978-1788998017.

Spark [online]. [cit. 2019-11-17]. Dostupné z: http://sparkjava.com

Spring MVC Interview Questions. *Baeldung.com* [online]. 2018 [cit. 2019-11-17]. Dostupné z: https://www.baeldung.com/spring-mvc-interview-questions

SUNDIN, Emil. Perception and effects of implementing Kotlin in existing projects. Falun, 2018. Bachelor thesis. Dalarna University. Vedoucí práce Wei Song.

The State of Developer Ecosystem in 2018. JetBrains [online]. [cit. 2019-11-17]. Dostupné z: https://www.jetbrains.com/research/devecosystem-2018

The State of Developer Ecosystem in 2019. JetBrains [online]. [cit. 2019-11-17]. Dostupné z: https://www.jetbrains.com/research/devecosystem-2019

Three-Tier Architecture. Technopedia.com [online]. [cit. 2019-11-17]. Dostupné z: https://www.techopedia.com/definition/24649/three-tier-architecture

TIOBE Index for November 2019. Https://www.tiobe.com/ [online]. [cit. 2019-11-17]. Dostupné z: https://www.tiobe.com/tiobe-index

Usage of server-side programming languages for websites. W3Techs.com [online]. 2019, 2019 [cit. 2019-11-17]. Dostupné z: https://w3techs.com/technologies/overview/programming_language

Vert.x [online]. [cit. 2019-11-17]. Dostupné z: https://vertx.io

VESELÝ, Luďek. Proč používat Kubernetes. Ludekvesely.cz [online]. 2019, 2017 [cit. 2019-11-17]. Dostupné z: https://www.ludekvesely.cz/proc-pouzivat-kubernetes

Vuksanovic, Irena Petrijevcanin a Bojan Sudarevic. Use of web application frameworks in the development of small applications. 2011 Proceedings of the 34th International Convention MIPRO (2011): 458-462.

WENDEL, Per. Programming APIs with the Spark Web Framework [online]. 2015 [cit. 2019-11-17].

Where Is My Cache? Architectural Patterns for Caching Microservices. Dzone.com [online]. [cit. 2019-11-17]. Dostupné z: https://dzone.com/articles/where-is-my-cache-architectural-patterns-for-cachi?edition=521353&utm_source=Weekly%20Digest&utm_medium=email&utm_campaign=Weekly%20Digest%202019-10-02

Why Kotlin for backend is a real thing? Kotlindevelopment.com [online]. 2019 [cit. 2019-11-17]. Dostupné z: https://www.kotlindevelopment.com/why-kotlin-for-backend-is-a-real-thing

# Přílohy

## Příloha A: Fullerův trojúhelník se stanovením vah kritérií hodnocení

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **1** | | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| **2** | | | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 |
| **3** | | | | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 |
| **4** | | | | | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| **5** | | | | | | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **6** | | | | | | | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 |
| **7** | | | | | | | | 1 | 1 | 1 | 1 | 0 | 0 | 0 |
| **8** | | | | | | | | | 0 | 1 | 1 | 0 | 0 | 0 |
| **9** | | | | | | | | | | 1 | 1 | 0 | 0 | 0 |
| **10** | | | | | | | | | | | 1 | 0 | 0 | 0 |
| **11** | | | | | | | | | | | | 0 | 0 | 0 |
| **12** | | | | | | | | | | | | | 1 | 1 |
| **13** | | | | | | | | | | | | | | 0 |
| **14** | | | | | | | | | | | | | | |
| **15** | | | | | | | | | | | | | | |

Zdroj: autor

Čísla řádků a sloupců v tabulce odpovídají níže uvedeným kritériím:

1. Podpora Kotlinu
2. Web
3. Security
4. Templating
5. Caching
6. Dependency injection
7. JSON
8. Integrace
9. Přístup k datům
10. AOP
11. Scheduling
12. Testovatelnost
13. Podpora a komunita
14. Praxe
15. Výkon