Univerzita Hradec Králové
Fakulta informatiky a managementu
Katedra informatiky a kvantitativních metod

Porovnání webových frameworků pro jazyk Java

Bakalářská práce

Autor: Vít, Řehák
Studijní obor: Aplikovaná informatika

Vedoucí práce: doc. Mgr. Tomáš Kozel, Ph.D.

Hradec Králové 04.2023

Prohlášení:

Prohlašuji, že jsem bakalářskou práci zpracoval samostatně a s použitím uvedené literatury.

V Hradci Králové dne 25.4.2023 Vít Řehák

Poděkování:

Děkuji vedoucímu bakalářské práce doc. Mgr. Tomáši Kozlovi, Ph.D. za metodické vedení práce, vstřícnost a ochotu, se kterou mi vždy vyhověl. Dále bych chtěl poděkovat rodině za veškerou podporu.

Anotace

Tato práce se zabývá porovnáním Java frameworků pro tvorbu webových aplikaci. Nejdříve je zde stručný úvod do webových technologií. Následuje porovnání běžného způsobu kompilace jazyku Java a jejího alternativního způsobu prostřednictvím GraalVM. V hlavní části jsou představené porovnávané technologie Spring, Quarkus a Micronaut. Tyto frameworky jsou popsány z hlediska možností tvorby webových aplikací, testů, zpracování dat a bezpečnosti. Dále jsou uvedeny testy, které se týkají především výkonu a spolehlivosti zmiňovaných technologií. Tyto testy jsou rozděleny na zátěžové testy a testy běžného provozu. Praktická část se věnuje návrhu informačního systému a jeho následné implementaci v jedné vybrané technologii. Poslední část práce se věnuje samotné tvorbě aplikace, technologiím, které byly při ní využity, a problémům, se kterými se autor práce při implementaci setkal.

Annotation

Title: Comparison of Web Frameworks for Java Language

This work deals with a comparison of Java frameworks for web application development. Firstly, there is a brief introduction to web technologies. Next, a comparison of the standard way of compiling Java language and its alternative way through GraalVM is presented. In the main part, the compared technologies Spring, Quarkus, and Micronaut are introduced. These frameworks are described in terms of their capabilities for creating web applications, testing, data processing, and security. Furthermore, tests are presented that mainly relate to the performance and reliability of the mentioned technologies. These tests are divided into stress tests and tests of regular operation. The practical part focuses on the design of an information system and its subsequent implementation in one selected technology. The last part of the work deals with the actual creation of the application, the technologies used during its development, and the problems encountered by the author during implementation.

Obsah

1 Úvod ... 1
2 Cíl práce ... 2
3 Webová aplikace ... 3
3.1 Frontend ... 3
3.1.1 HTML (HyperText Markup Language) ... 4
3.1.2 CSS (Cascading Style Sheets) ... 4
3.1.3 JavaScript ... 5
3.1.4 ReactJs ... 7
3.2 HTTP Komunikace ... 8
3.2.1 Typy Metod ... 9
3.2.2 Status Kódy ... 9
3.3 Backend ... 10
3.3.1 Architektura mikroslužeb ... 10
3.4 Databáze ... 10
3.4.1 PostgreSQL ... 11
4 Java ... 12
4.1 Java kompilace interpretace a běh aplikace ... 12
4.1.1 JDK (Java Development Kit) ... 13
4.1.2 JRE (Java Runtime Environment) ... 13
4.1.3 JVM (Java Virtual Machine) ... 14
4.1.4 Java Interpret ... 14
4.1.5 JIT (Just In Time) ... 14
4.2 GraalVM kompilace interpretace a běh aplikace ... 14
4.2.1 GraalVM JIT ... 15
4.2.2 AoT (Ahead of Time) ... 15

4.2.3 Native Image 15
5 Porovnání Spring X Quarkus X Micronaut 16
5.1 Framework vs. knihovna 16
5.2 GraalVM native frameworky 16
5.3 Spring (Spring boot) 16
5.3.1 Spring principy fungování 17
5.3.2 Spring Web 18
5.3.3 Spring Security 20
5.3.4 Spring Data 20
5.3.5 Spring Test 21
5.3.6 Spring výhody a nevýhody: 21
5.4 Quarkus 22
5.4.1 Quarkus principy fungování 22
5.4.2 Quarkus Web 24
5.4.3 Quarkus Security 24
5.4.4 Quarkus Data 25
5.4.5 Quarkus Test 25
5.4.6 Quarkus výhody a nevýhody 25
5.5 Micronaut 26
5.5.1 Micronaut principy fungování 27
5.5.1.2 CDI (Context and Dependenci Injection) 27
5.5.2 Micronaut Web 28
5.5.3 Micronaut Securtity 28
5.5.4 Micronaut Data 29
5.5.5 Micronaut Test 30
5.5.6 Micronaut výhody a nevýhody 30

5.6 Testování Spring X Quarkus X Micronaut 31
5.6.1 Popularita Java frameworků 32
5.6.2 Kompilace a start aplikace 33
5.6.3 Testované scénáře 35
5.6.4 Diskuse nad výsledky testů 41
5.7 Shrnutí výsledků porovnání 42
6 Praktická část: Informační systém 44
6.1 Funkcionality 44
6.2 Vybrané technologie 45
6.2.1 Backend 45
6.2.2 Frontend 46
6.3 Návrh databáze 47
6.4 Návrh API 49
6.4.1 Endpointy 49
6.5 Problémy 50
7 Shrnutí výsledků 52
7.1 Budoucí pokračování 52
8 Závěry a doporučení 53
9 Seznam použité literatury 54
10 Přílohy 58

Seznam obrázků

OBRÁZEK 1 SCHÉMA FUNGOVÁNÍ WEBOVÉ KOMUNIKACE 3
OBRÁZEK 2 HTML ELEMENT <STRONG> 4
OBRÁZEK 3 CSS NASTAVENÍ BARVY 5
OBRÁZEK 4 QUERYSELECTOR A LISTENER 6
OBRÁZEK 5 TŘÍDNÍ KOMPONENTA 7
OBRÁZEK 6 FUNKCIONÁLNÍ KOMPONENTA 8
OBRÁZEK 7 UKÁZKA HTTP POŽADAVKU 8
OBRÁZEK 8 SCHÉMA JAVA KOMPILACE 13
OBRÁZEK 9 SCHÉMA JAVA KOMPONENT 13
OBRÁZEK 10 SCHÉMA ZPRACOVÁNÍ HTTP POŽADAVKU 19
OBRÁZEK 11 POPULARITA JAVA FRAMEWORKŮ 32
OBRÁZEK 12 PRŮMĚRNÁ DOBA KOMPILACE NATIVE IMAGE (S) 33
OBRÁZEK 13 PRŮMĚRNÁ DOBA STARTU NATIVE IMAGE (S) 34
OBRÁZEK 14 VELKOST SPUSTITELNÉHO SOUBORU 34
OBRÁZEK 15 PRŮMĚRNÉ VYUŽITÍ CPU (%) 35
OBRÁZEK 16 PRŮMĚRNÉ VYUŽITÍ RAM (MIB) 36
OBRÁZEK 17 POČET ÚSPĚŠNÝCH A NEÚSPĚŠNÝCH TESTOVACÍCH SCÉNÁŘŮ 36
OBRÁZEK 18 PRŮMĚRNÁ DOBA ODPOVĚDI (MS) 37
OBRÁZEK 19 PRŮMĚRNÉ VYUŽITÍ CPU (%) 37
OBRÁZEK 20 PRŮMĚRNÉ VYUŽITÍ RAM (MIB) 38
OBRÁZEK 21 POČET ÚSPĚŠNÝCH A NEÚSPĚŠNÝCH TESTOVACÍCH SCÉNÁŘŮ 38
OBRÁZEK 22 PRŮMĚRNÁ DOBA ODPOVĚDI (MS) 39
OBRÁZEK 23 PRŮMĚRNÉ VYUŽITÍ CPU (%) 40
OBRÁZEK 24 PRŮMĚRNÉ VYUŽITÍ RAM (MIB) 40
OBRÁZEK 25 PRŮMĚRNÁ DOBA ODPOVĚDI (MS) 41
OBRÁZEK 26 SCHÉMA DATABÁZE 47
OBRÁZEK 27 PŘÍKLAD REPOSITÁŘE A NATIVNÍHO SQL DOTAZU 48
OBRÁZEK 28 ENDPOINT PŘIDĚLENÍ ROZPOČTU 49
OBRÁZEK 29 ENDPOINT ZOBRAZENÍ VYTVOŘENÝCH SKUPIN 50

Seznam tabulek

TABULKA 1 PODPOROVANÉ TECHNOLOGIE 43

1 Úvod

V současné době se téměř každý z nás pohybuje v prostředí internetu a je obeznámen s pojmy, jako jsou e-shop, email, web, prohlížeč, popřípadě cloud, ale už málo lidí ví, co se vlastně pod těmito pojmy skrývá za logiku. V dnešní době se všechny firmy předhánějí o naši pozornost, které nám už moc nezbývá. Dříve jsme byli schopni čekat na spuštění aplikace systému nebo načtení obrázku i celé minuty. Naštěstí tato doba už je dávno pryč a naše trpělivé čekání také. A tak mají nyní nejvíce pozornosti aplikace, které se načtou jako první. A proto vznikají stále nové technologie a nástroje pro zrychlení běhu aplikací a načítání webu. Tato práce se snaží představit způsoby, jak toho dosáhnou především s pomocí GraalVM a nativních frameworků. Nejprve je ale důležité si představit fungování již zmíněných webových technologií.

2 Cíl práce

Cílem této práce je přiblížit čtenáři problematiku webu a webových technologií a následně porovnat frameworky pro tvorbu REST API v jazyce Java. Vybrané frameworky podporují možnost kompilace do nativního (strojového) kódu a tvorbu spustitelného souboru. Technologie budou nejdříve popsány a následně srovnány podle těchto kritérií:

- doba zpracování požadavku
- maximální zátěž serveru
- doba spouštění aplikace
- doba kompilace

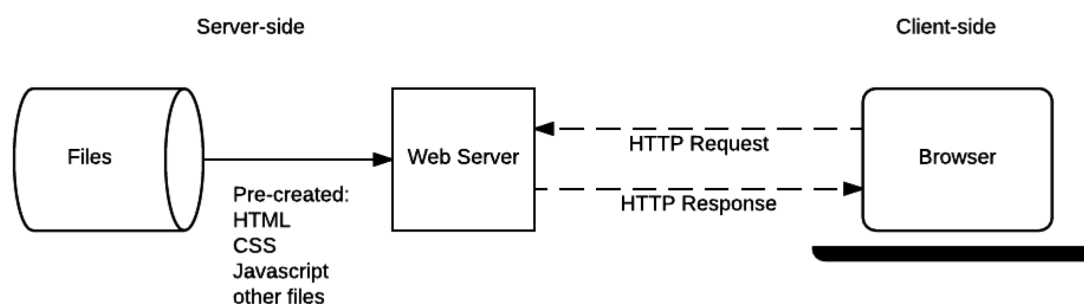
Poté bude následovat návrh a implementace informačního systému. Pro perzistenci dat bude využit databázový server typu PostgreSQL. Objektový návrh databáze bude navržen v modelovacím nástroji Enterprise Architect. Backend bude vytvořen v jedné z později porovnávaných technologií. Pro interpretaci dat na frontendu bude použito HTML, CSS, JavaScript a ReactJS.

3 Webová aplikace

Webová aplikace se vždy skládá ze dvou základních částí frontendu a backendu. Mezi těmito částmi probíhá komunikace prostřednictvím HTTP protokolu. V dnešní době se snaží většina webový aplikací dodržovat architekturu MVC (model view controller), která rozděluje aplikaci do několika logických celků.

Příklad zpracování požadavku:
Klient pošle dotaz na server prostřednictvím webového prohlížeče. Server rozpozná adresu a metodu, poté deleguje dotaz na příslušný controller a endpoint. Zde se zpracovává, například se manipuluje s daty v databázi. Pokud vše proběhlo v pořádku, data jsou odeslána v podobě odpovědi zpět na klienta. V opačném případě klient obdrží odpověď v podobě chybového kódu.

Běžně je tato jednoduchá komunikace rozšířena například o autentifikaci a autorizaci uživatele. Velmi často jsou požadovány dynamické stránky, které jsou ještě před odesláním klientovi sestaveny z dat uložených v databázi. Tato data bývají poskládána pomocí šablonovacích nástrojů, jako jsou Handlebars, Razor, nebo Thymeleaf.



Obrázek 1 Schéma fungování webové komunikace
Zdroj: [1]

3.1 Frontend

Frontend je část aplikace, která je každému uživateli nejbližší, protože se jedná o uživatelské převážně grafické rozhraní. V případě webů je frontendem stránka,

která se zobrazí v prohlížeči po odeslání požadavku na server. Skládá se především z HTML, CSS, popřípadě z dalších technologií, pokud je zapotřebí vyšší stupeň interakce, jako třeba odesílání objednávek na e-shopu nebo posílání emailů.

3.1.1 HTML (HyperText Markup Language)

„HTML (HyperText Markup Language) je základním stavebním kamenem webu. Definuje význam a strukturu webového obsahu. Ostatní technologie kromě HTML se obecně využívají k popsání vzhledu/prezentace webové stránky (CSS) nebo funkčnosti/chování (JavaScript).“ (přeloženo z [2]) Skládá se ze značek (tagů), kterými obaluje data samotné stránky a přidává jim význam, jako třeba odkaz označení kurzívou nebo tučným písmem. Ale mimo to udává i znakovou sadu, kterou bude stránka zobrazena. V závislosti na typu tagu můžou obsahovat dodatečné údaje v podobě atributů.

- href – udává cíl odkazu
- src – udává cestu například k obrázku
- type – udává například typ tlačítka nebo vstupu

Specifickými typy atributů jsou class a id, které se používají pro odkazování na elementy v CSS nebo JavaScriptu.

```
<strong id="toto je atribut"> tento text bude zobrazen tučně </strong>
```

Obrázek 2 HTML element <strong>
Popis: tento text bude zobrazen tučně a bude obsahovat id „toto je atribut“

3.1.2 CSS (Cascading Style Sheets)

CSS je nástroj, který slouží k návrhu a grafickému uzpůsobení stránky, protože HTML určuje pouze strukturu a význam jednotlivých částí stránky, zatímco CSS nastavuje barvu, velikost písma, orámování, šířku elementu, animace a tak dále. [3]

Tyto styly se dají zapisovat třemi způsoby podle priority:

- Řádkový zápis – CSS v atributy style samotného elementu
- Interní zápis – CSS v tagu style ve hlavičce
- Externí zápis – CSS v externím souboru spárovaný tagem link

Kaskádové styly jsou takto nazývány proto, že vždy specifičtější vyjádření stylu má větší prioritu:

- * – všechno
- tag – určitý typ tagu
- #identifikátor – tag s konkrétním id
- .třída – tagy s konkrétní třídou

Aby bylo možné zaměřit pouze tagy v konkrétní části stránky, dají se CSS zanořovat.

```
footer a{
    color: blue;
}
```

Obrázek 3 CSS nastavení barvy
Popis: CSS nastaví modrou barvu všech odkazů, které se nacházejí v patičce.

3.1.3 JavaScript

JavaScript taky nazývaný jako JS se stará o dynamičnost stránky společně s uživatelskými interakcemi. Nejčastěji je možné se s ním setkat na frontendu, kde se používá například pro získávání dat z formulářů, validace, zasílání dotazů na server a další. Ale v kombinaci s Node.js ho lze použít pro tvorbu backendu. JS využívá velké množství frameworků a knihoven, jako třeba ReactJS (tvorba uživatelského rozhraní), axios (odesílání HTTP požadavků), JQuery (zjednodušení práce s HTML a JS), Express.js (tvorba backendu) a mnoho dalších. Některé z nich pouze zpříjemňují práci vývojáře, ale třeba ReactJS se pokouší o zlepšení výkonu samotné stránky.

Jak u CSS, tak i u JS je možné scripty zapisovat více způsoby:

- Interní zápis – JS v tagu script
- Externí zápis – JS v externím souboru spárovaný atributem src v tagu script

I přestože JS neobsahuje standartní datové typy, manipulace s komplexnějšími daty probíhá prostřednictvím zápisu objektů ve formátu JSON (Java Script Object Notation), který vznikl z JS a poté se rozšířil jako jeden z nejpopulárnějších datových formátů. Pro práci s DOM (Dokument Object Model) se používají metody/funkce jako ve standartních vyšších jazycích. Pro přístup ke konkrétním elementům je potřeba nejprve získat jejich instanci. Toho lze dosáhnout pomocí queryselectoru, to je funkce, která je schopna zaměřit konkrétní element například pomocí atributu ID nebo name. Poté co queryselector vrátí referenci na element, je možné upravovat jeho hodnotu, vnitřní HTML, přidat potomka (zanořit nový element do již existujícího) nebo přidat listener. Listener je speciální druh funkce, která čeká na výskyt přidělené události, například kliknutí myši nebo odeslání formuláře. Po detekování příslušné akce provede přidělený blok kódu. [4]

HTML:

```
<p id="odstavec">Příklad</p>
```

JS:

```
let odstavec = document.getElementById("odstavec ");
odstavec.addEventListener("click", () => Console.log("Toto je odstavec"));
```

Obrázek 4 Queryselector a Listener

Popis: Získání reference na odstavec s id odstavec pomocí queryselectoru, a následné zaregistrovaní listeneru, který kontroluje událost kliknutí na element.

Výsledek: Po kliknutí na odstavec se do konzole vypíše: „Toto je odstavec“.

3.1.4 ReactJs

„React je deklarativní, efektivní a flexibilní JavaScript knihovna pro tvorbu uživatelského rozhraní. Umožňuje skládat komplexní uživatelská rozhraní z malých izolovaných kousků kódu, zvaných "komponenty"." (přeloženo z [5])

Na rozdíl od JS a HTML, které při sebemenší změně dokumentu provádějí změnu tím, že aktualizují celou stránku, React přichází s nápadem použít tzv. virtuální DOM. Ten organizuje komponenty a elementy na stránce do stromové struktury. Při každé změně stránky se změny zapisují do virtual DOM, který se nachází pouze v paměti. Ten se poté porovnává s předešlou verzí. Pokud jsou objeveny nějaké změny, neaktualizuje se celý DOM, ale pouze příslušná větev. Tato funkcionalita nejvíce prospívá single page webům, protože díky React roteru je možné měnit pouze hlavní komponentu stránky v závislosti na URL adrese, a předejít tak opětovnému načtení patičky navigace a podobně. [5]

Komponenty se dají tvořit dvěma způsoby:

1. Třídní komponenta (Class component) – vznikají dědičností z React Component s funkcí render. Pro práci se stavovými proměnnými se používají předdefinované funkce jako componentDidMount a componentWillUnmount.

```
class Ukazka1 extends React.Component {
  render() {
    return <h1>Hello, World!</h1>;
  }
}
```

Obrázek 5 Třídní komponenta

2. Funkcionální komponenta (Function component) – vznikají jako JS funkce s návratovou hodnotou. Pro práci se stavovými proměnnými se používají tzv. hooky – předdefinované funkce jako useState, useEffect, useRef, atd. Je také možnost tvorby vlastních hooků.

```
function Ukazka2() {
  return <h1>Hello, World!</h1>;
}
```

Obrázek 6 Funkcionální komponenta

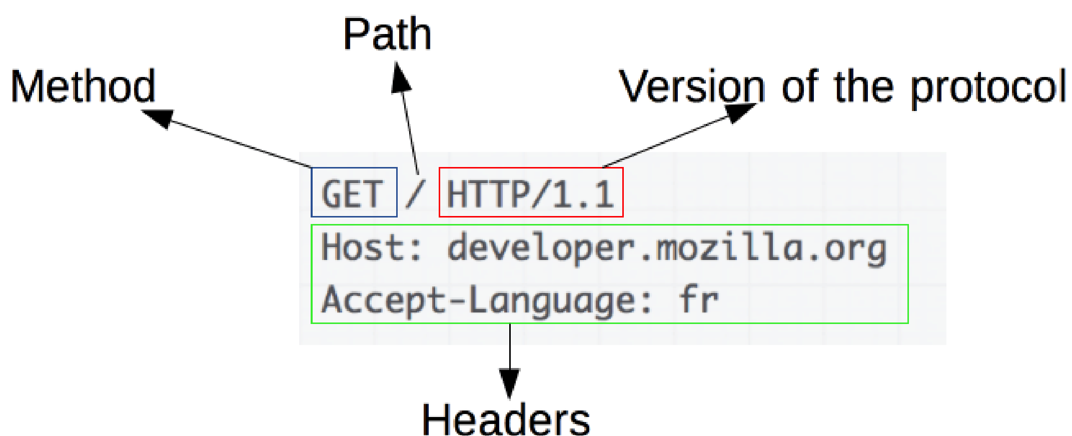
3.2 HTTP Komunikace

„HTTP je protokol pro získávání obsahu, jako jsou HTML dokumenty. Je to základ pro jakoukoli výměnu dat na webu a jedná se o klient-server protokol. To znamená, že požadavky jsou iniciovány příjemcem, obvykle webovým prohlížečem. Celý dokument je zkonstruován ze získaných dílčích dokumentů např. text, popis rozložení, obrázky, videa, skripty a další.“ (přeloženo z [6]) Data jsou předávána v různých formátech, ale nejčastější formát je JSON a XML. Komunikace je realizována pomocí zpráv, které se rozlišují na:

Požadavek – Zasílán uživatelem, obsahuje informace o metodě, cestě, verzi protokolu a dodatečné informace v podobě hlaviček.

Odpověď – Zpráva odeslána serveremframe jako reakce na požadavek, obsahuje informace o verzi protokolu, status kód, popis statusu a dodatečné informace v podobě hlaviček.

V době zpracování této práce platí tyto standardy HTTP/1.1, HTTP/2, HTTP/3.



Obrázek 7 Ukázka HTTP požadavku
Zdroj: [6]

3.2.1 Typy Metod

Nejčastěji používané metody protokolu HTTP:

- GET – Získání dat
- POST – Odeslání dat
- PUT – Aktualizace dat
- DELETE – Smazání dat
- OPTIONS – Popisuje možnosti komunikace pro cíl požadavku

3.2.2 Status Kódy

Status kódy vyjadřují, zda byl požadavek úspěšný či nikoliv a poskytuje základní přehled o odeslaném požadavku.
Rozdělují se do pěti kategorií:

- 100–199 – Informační zprávy
- 200–299 – Zprávy o úspěchu
- 300–399 – Zprávy o přesměrování
- 400-499 – Zprávy o problému na straně uživatele
- 500-599 – Zprávy o problému na straně serveru

Nejčastěji využívanými kódy jsou:

- 200 – OK – záleží na metodě, např. data byla doručena nebo úspěšně vrácena
- 401 – Unauthorized – uživatel se musí autentizovat pro obdržení odpovědi
- 404 – Not found – požadavek nedoručen, stránka nenalezena
- 503 – Service unavailable – server může být přetížen nebo v údržbě

3.3 Backend

Backendem se rozumí serverová část aplikace, na kterou jsou zasílány dotazy z klienta. Tato komunikace probíhá pomocí HTTP protokolu. Server se stará o zpracování požadavků a tvorbu odpovědí. Pokud dotaz vyžaduje autentifikaci uživatele, je to právě backend, kdo ho ověří na základě informací zaslaných z frontendu. Backend obsahuje celou businesse logiku zpracování dat a jejich následné uložení do databáze.

3.3.1 Architektura mikroslužeb

Mikroslužby jsou architektonickým a organizačním přístupem k vývoji softwaru, kde se software skládá z malých nezávislých služeb, které komunikují pomocí dobře definovaných API. Tyto služby jsou vlastněny malými, samostatnými týmy. Architektury založené na "mikroslužbách" usnadňují škálování aplikací a zrychlují vývoj, umožňují inovace a zrychlují čas uvádění nových funkcí na trh. (přeloženo z [7])

3.4 Databáze

Databázový systém je část aplikace běžící na backendu, která slouží k bezpečnému uchovávání dat. Samotná databáze (data) může být umístěna buď na stejném serveru jako aplikace, nebo na vzdáleném serveru. Databázové systémy lze rozdělit do několika skupin na základě architektury. Mezi nejčastější typy patří relační databáze a NoSQL databáze.

- Relační databáze – V tomto případě jsou data sdružována do tabulek, které jsou následně provázané pomocí primárních a cizích klíčů. Mají jasně definovanou strukturu a datové typy, což umožnuje efektivní dotazování nad daty pomocí jazyka SQL (Structured Query Language). Na druhou stranu je obtížné měnit jejich strukturu z důvodu vysoké provázanosti dat. Nejčastěji používanými relačními databázemi jsou MySQL, Oracle, SQL server, PostgreSQL.

- NoSQL databáze – NoSQL (not only SQL) je přizpůsobený pro práci s daty, která nejsou atomická. To znamená, že je možné ukládat například celé JSON objekty. Často se využívají při tvorbě systémů, které budou do budoucna rozšiřované. Mezi nejznámější zástupce patří Apache Cassandra a MongoDB.

3.4.1 PostgreSQL

PostgreSQL je open source RDBMS (Relational Database Management System), který používá SQL, a dále ho rozšiřuje. [8]

- Podporuje komplexní datové typy, jako jsou pole a JSON.
- PostgreSQL je dostupný na velkém množství platforem jako Linux, Windows, macOS.
- Podporuje replikaci dat.
- Možnost využití transakčního zpracování, které přistupuje k sadě příkazů jako k atomické operaci.
- PostgreSQL dovoluje tvorbu uživatelských funkcí v programovacích jazycích, jako jsou C, Java, Python.

4 Java

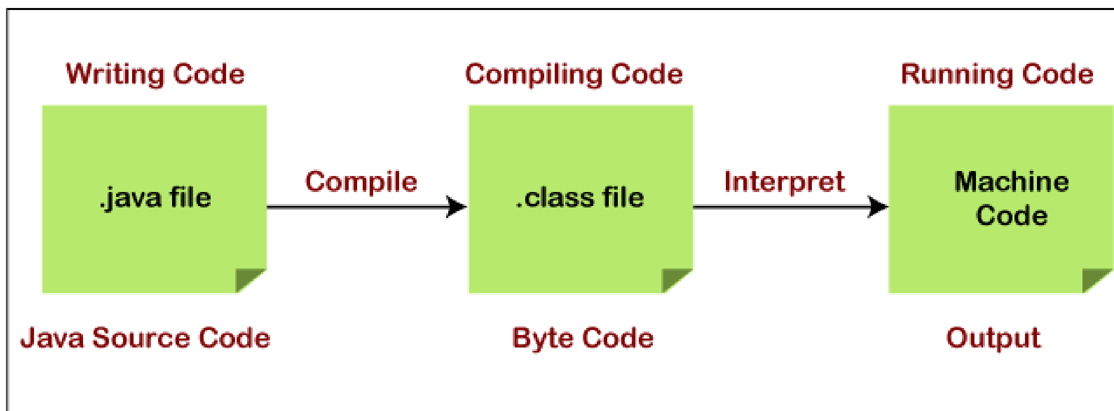
Java je objektově orientovaný programovací jazyk vyšší úrovně, který získal velkou oblibu zejména díky své přenositelnosti mezi platformami, což je zapříčiněno způsobem kompilace a spouštění programu. Java je vyvíjena pod společností Oracle. Vychází z rodiny jazyku C, C++, se kterými má mnoho společných rysů, jako např. syntaxi, ale přesto nepodporuje některé funkcionality svých předchůdců, jako jsou pointery a struktury. Podobně jako je tomu u jiných jazyků, disponuje širokou škálou různých API, knihoven a frameworků. Paměť je spravována automaticky pomocí garbage collectoru, který má za úkol vyhledat a uvolnit části paměti, které už nejsou potřeba.

4.1 Java kompilace interpretace a běh aplikace

Kompilační proces je pro Javu stěžejní, zejména kvůli její platformní přenositelnosti. *„Program musí být převeden do formy, které rozumí JVM, aby jakýkoliv počítač s JVM byl schopen interpretovat a spustit program. Kompilování Java programu znamená vzít programátorsky přívětivý kód v souborech programu (také nazýván zdrojový kód) a převést ho na bytecode, který představuje platformě nezávislé instrukce pro JVM. Poté, co byl program úspěšně zkompilován na Java bytecode, může být interpretován a spuštěn na jakékoliv JVM. Interpretace a běh Java programu znamená vyvolání JVM interpreta, který převede Java bytecode na platformě závislý strojový kód, kterému počítač rozumí, a je tak schopen jej spustit.“ (přeloženo z [9])*

Příkazy:

- javac nazevAplikace.java – vyvolání kompilace
- java nazevApikace – vyvolání interpret



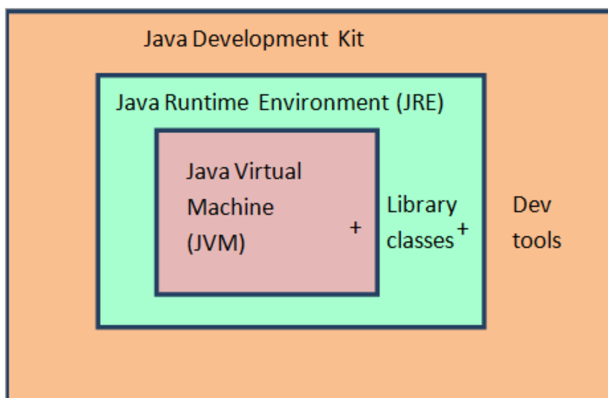
Obrázek 8 Schéma Java kompilace
Zdroj: [10]

4.1.1 JDK (Java Development Kit)

JDK se skládá z JRE a vývojářských nástrojů, mezi které se řadí debuger, compiler a javaDoc. Tyto nástroje se využívají např. ke kontrole kódu, odchytávání běhových chyb a zkompilování tříd typu .java na bytecode. JDK není pro normální běh aplikace potřeba.

4.1.2 JRE (Java Runtime Environment)

JRE je minimální běhové prostředí, které se využívá pro spouštění Java aplikace. Skládá se z JVM a standartní knihovny. Na rozdíl od JVM jde o software běžící v operačním systému. Z tohoto důvodu je na platformě závislý a je potřeba pro každý OS správná implementace.



Obrázek 9 Schéma Java komponent
Zdroj: [11]

4.1.3 JVM (Java Virtual Machine)

JVM je virtuální komponenta, díky které je Java platformě nezávislá. Obsahuje interpret a JIT, které se starají o čtení a překlad bytecodu uloženého v souborech typu .class na spustitelné instrukce v podobě strojového kódu.

4.1.4 Java Interpret

Interpret při běhu převádí bytecode na instrukce v podobě strojového kódu, kterému konkrétní OS rozumí. Nicméně interpretované aplikace jsou pomalejší než nativní spustitelný soubor. Především z důvodu nutné režie, která je spojena se zpracováním bytecodu. Z tohoto důvodu vznikl JIT.

4.1.5 JIT (Just In Time)

JIT optimalizuje běh aplikace tím, že za běhu překompiluje bytecode, a šetří tak práci interpretovi. To ale hned neznamená, že se všechny zkompilují, protože kompilace by vyšla náročněji než pouhá interpretace. Děje se tak např. při startu aplikace, když se volá spousta metod vícekrát a pokud některá přesáhne určitý počet volání, pak se místo interpretace zkompiluje, aby se ušetřil výkon při jejím dalším volání. Ve výchozím nastavení je tato funkce povolena. [12]

4.2 GraalVM kompilace interpretace a běh aplikace

„GraalVM je vysoko výkonnostní JDK navržený pro zvýšení výkonu Java aplikace při používaní méně zdrojů. Nabízí dvě možnosti běhu aplikace: na Hotspot JVM s Graal JIT kompilátorem nebo jako AOT kompilovaný nativní spustitelný soubor. Kromě Javy nabízí běhové prostředí pro JavaScript, Ruby, Python a mnoho dalších populárních jazyků. GraalVM poskytuje schopnost kombinovat programovací jazyky v jedné aplikaci, přičemž eliminuje jakékoliv týkající se volání jiných jazyků.“ (přeloženo z [13])

4.2.1 GraalVM JIT

GraalVM JIT na rozdíl od standartního JIT obsaženého v JVM přináší několik výhod, mezi které patří například:

- Možnost práce s větší škálou programovacích jazyků z důvodu možnosti jejich kombinování v jedné aplikaci.
- Používá pokročilejší techniky optimalizace, jako třeba odhadování výsledků podmínek a připravování na jejich vyhodnocení, což má za následek zrychlení běhu u předpokládaných výsledků, ale ne špatné vyhodnocení podmínek.
- Menší paměťovou stopu díky efektivnější alokaci paměti.

4.2.2 AoT (Ahead of Time)

AoT je technologie kompilování zdrojového kódu s předstihem přímo na strojový kód konkrétního OS, což má za následek zrychlení startu aplikace, která na rozdíl od native image stále může být závislá na VM nebo jiné kompilační infrastruktuře. Nicméně AoT vyústí ve zpomalení kompilace aplikace a nemusí být vhodný pro všechny případy, jako je například použití reflexe.

4.2.3 Native Image

Native image je samostatně stojící spustitelný soubor, který je možné použít bez jakýchkoliv závislostí včetně použití GraalVM, čímž se stává rychlejším než samotné AoT. Start se může pohybovat v řádech milisekund. Součástí kompilovaného souboru je pouze kód, který je potřeba za běhu aplikace, jako jsou aplikační třídy, standartní knihovny a jejich závislosti. Samotný native image je pak tvořen pomocí Native Image builder. Ten provede analýzu aplikační třídy a metadat. Následně určí, které metody má aplikace možnost využít za běhu, aby finální soubor neobsahoval zbytečná data. Nakonec tyto metody zkompiluje a vytvoří spustitelný soubor, který obsahuje i potřebné části obsažené v VM jako garbage collector. Navzdory všem výhodám se stává výsledný soubor platformě závislý. [14]

5 Porovnání Spring X Quarkus X Micronaut

"Framework neboli softwarový rámec je platforma, která poskytuje základ pro vývoj softwarových aplikací. Můžete si ho představit jako šablonu pracujícího programu, který lze selektivně upravovat přidáním kódu. Využívá sdílených prostředků, jako jsou knihovny, obrázkové soubory a referenční dokumenty, a skládá je do jednoho balíčku. Tento balíček lze upravovat tak, aby vyhovoval specifickým potřebám projektu. S pomocí frameworku může vývojář přidávat nebo nahrazovat funkce, aby aplikace získala novou funkcionalitu." (přeloženo z [15]) Framework typicky může obsahovat předepsané části kódu a ukázky nejlepších postupů.

5.1 Framework vs. knihovna

Knihovny jsou daleko flexibilnější a představují spíše soubor předepsaných tříd a metod, které řeší určitou problematiku, jako je zpracování textu nebo matematické operace. Oproti tomu frameworky jsou komplexnější nástroje pro vývoj aplikace. A mnohdy v sobě obsahují spousty knihoven.

5.2 GraalVM native frameworky

Tato práce se zabývá porovnáním tří frameworků Micronaut, Quarkus a Spring (resp. jeho nadstavbou Spring Boot), a to v kombinaci s GraalVM a tvorbou nativních souborů. Tyto technologie se řadí k nepopulárnějším (viz. kapitola 5.6.1 Popularita Java frameworků) možnostem v Javě, pokud přijde na tvorbu webového API nebo mikroslužeb obecně.

5.3 Spring (Spring boot)

Spring je open-source framework, který poskytuje obsáhlou paletu možností konfigurace s využitím knihoven třetích stran. Dále zahrnuje Spring MVC framework, který poskytuje nástroje pro tvorbu webových stránek, jako jsou zpracování HTTP požadavků, model-view-controler architekturu a další technologie s tím spojené. Také lze využít AOP (Aspect Oriented Programming).

AOP je programovací paradigma, které pomáhá zaintegrovat funkcionality potřebné napříč více vrstvami aplikace. Tyto funkcionality zapouzdří do samostatných modulů (aspektů), které lze následně použít, kde je potřeba. Tento přístup abstrahuje businesse logiku od tzv. cross-cutting concerns (společné zájmy), kterými muže být například logování nebo jiné funkce používané napříč aplikací. Dýky AOP lze tvořit přehlednější a modulárnější kód. [16]

5.3.1 Spring principy fungování

Spring framework poskytuje takzvaný kontejner, který je zodpovědný za správu Java objektů (tzv. beanů) a jejich závislostí. Využívá IoC (viz. níže), což za má za následek volnější vazby mezi třídami, dále pak čistší kód a modulárnější aplikaci. Díky tomu se aplikace lépe testují, škálují a udržují. [16]

5.3.1.1 IoC (Inversion of Control)

IoC je návrhový vzor pro volné spojování tříd v aplikaci. V tradičním programování by si třídy vytvářely a spravovaly svoje závislosti samy, což může vést k úzce provázané hierarchii, kterou je těžké testovat a udržovat. Díky IoC je tato odpovědnost přenesena ze tříd na externí kontejner a ten pak provazuje třídy podle potřeby. Ve Springu je realizován pomocí aplikačního kontextu (Aplication Context). Jeho práce spočívá ve tvorbě, konfiguraci a správě bean. Pro jejich tvorbu a provázání se používají anotace, XML konfigurační soubory nebo Java kód. Když se kontejner spustí, přečte si konfiguraci a vytvoří potřebné objekty. Tyto objety se dále nazývají beany a jsou následné dostupné napříč celou aplikací. [17]

5.3.1.2 DI (Dependency Injection)

DI je metoda úzce spojená s IoC, jedná se o způsob vkládání závislostí objektům namísto toho, aby je získaly samy. Toto dovoluje větší flexibilitu a modularitu aplikace, protože závislosti se dají snadno nahradit nebo upravit bez zásahu do samotného objektu. DI je dosaženo prostřednictvím konstruktorů setrů nebo metod. Spring kontejner zjistí z konfiguračních souborů, které závislosti mají být doplněné a přímo za běhu je doplní. [17]

5.3.1.3 Bean

„Ve Springu se objekty, které tvoří páteř aplikace a jsou spravovány kontejnerem Spring IoC, nazývají beany." (přeloženo z [18]) Jinými slovy bean je jakákoliv instance třídy, která byla nakonfigurována. Aby se s objektem mohlo nakládat jako s beanem, musí splňovat určitá kritéria. Zaprvé instance musí být vytvořena pomocí Spring kontejneru a ne samotnou aplikací. Zadruhé musí být nadefinována ve Spring konfiguračním souboru nebo označena anotací Component. Nakonec musí mít unikátní identifikátor, aby se na ni dalo v celé aplikaci odkazovat. Beany mají různé typy, podle kterých se rozlišuje jejich délka života v Spring kontejneru. Nejběžnější typy jsou singleton (existuje pouze jeden napříč celou aplikací) a prototype (při každém zavolání se vytvoří nová instance). Ještě existují další typy, které se používají při tvorbě webových stránek. Jedná se například o request (závisí na HTTP požadavku), session (závisí na HTTP session) a application (závisí na aplikačním kontextu).

5.3.2 Spring Web

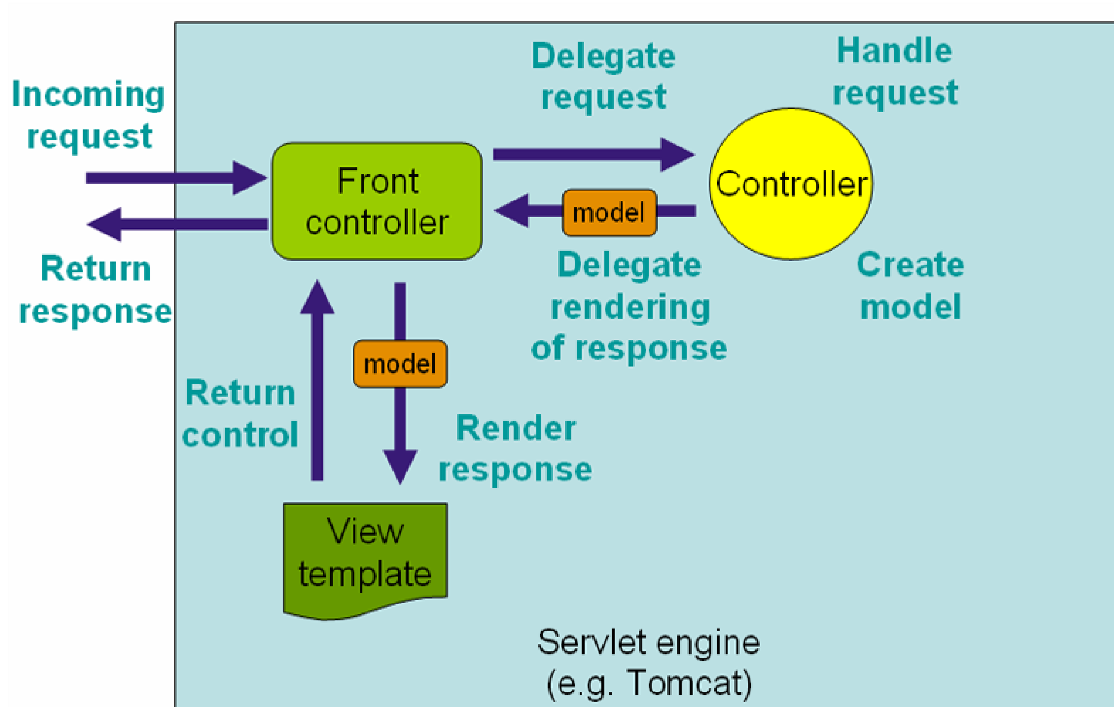
Spring MVC (Model View Controller) je framework, který je součástí Springu a využívá se pro tvorbu webových aplikací. Dodržuje architekturu MVC, která se skládá z několika komponent. [19]

- Model – Zapouzdřuje data a businesse logiku ve formě Java tříd.
- View – Zodpovědný za renderování uživatelského rozhraní, to může být realizováno pomocí JSP, Thymeleaf nebo jinými enginy[1].
- Controller – Zpracovává HTTP požadavky a odpovědi. Obdrží uživatelský podnět, který vyvolá příslušnou metodu v modelu, na jejímž základě se vybere příslušné view pro vygenerování odpovědi.
- Dispatcher Servlet – Přijímá všechny příchozí požadavky a deleguje je na konkrétní controller.

---

[1] V rámci této práce bude backend tvořen jako REST API, proto bude view generováno, až na klientské straně, bez použití nástrojů, které poskytuje Spring.

- Handler Mapping – Získává požadavky od Dispatcher servletu a vrací mu informaci, kam má být požadavek přesměrován.
- View resolver – Je zodpovědný za rozlišení view na základě požadavku na view od controlleru.
- Interceptors – Slouží k zachytávání příchozích i odchozích požadavků.



[2]

Obrázek 10 Schéma zpracování HTTP požadavku
Zdroj: [19]

[2] Front controller je pouze obecný návrhový vzor pro řízení toku požadavků. Dispatcher Servlet je jednou z možných implementací.

5.3.3 Spring Security

Spring Security je framework pro implementaci zabezpečení aplikací založených na Springu. Poskytuje flexibilní přístup k bezpečnosti webových stránek s širokou podporou mechanismů pro autentifikaci a autorizaci.

- Autentifikace – ověření identity uživatele
- Autorizace – ověření, zda uživatel má potřebná oprávnění

Mimo standardní přihlašování nabízí Spring další způsoby ochrany:

- HTTPS
- Ochranu před útoky, jako je SQL injection
- Session management
- Ochranu proti Cross-site request forgery (CSRF)

Spring Security může být implementována buď konfiguračním souborem, nebo anotacemi. Lze ji nakonfigurovat podle potřeb konkrétní aplikace.

5.3.4 Spring Data

Spring Data je framework vestavěný ve Springu a umožňuje přístup k datům, která mohou být uložena pomocí různých uložišť. Podporuje relační databáze (MySQL, PostgreSQL, Oracle), NoSQL (MongoDB) i cloudová uložiště (AWS, Azure). Má za cíl zjednodušit vývoj tím, že abstrahuje aplikaci od datové vrstvy a zpřístupní uložená data pomocí jednotného rozhraní. [20]
Výhody:

- Jednotné API napříč různými uložišti umožňuje snadné přecházení mezi uložišti, bez nutnosti změn v aplikaci.
- Redukuje nutnost psát opakující se kód, např. tvorbu jednoduchých dotazů nebo ruční převádění dat na objekty díky mapování.
- Tvorba dotazů na základě metod objektů nebo anotací.
- Podpora transakčního zpracování dat.

Je rozdělen do modulů, kde každý poskytuje různé sady funkcionality pro specifický typ uložiště, například:

Spring Data JPA – Umožnuje práci s relačními databázemi implementací JPA (Java Persistence API).

Spring Data MongoDB – Poskytuje nástroje pro práci s NoSQL databází MongoDB.

5.3.5 Spring Test

Spring Test je modul ve Springu, který poskytuje spoustu nástrojů pro testování aplikací. Obsahuje například: Web Test Client, MockMVC a Spring TestContext. [21]

- Web Test Client: umožnuje testování webového API, poskytuje metody pro posílání a ověřování testovacích požadavků.
- MockMVC: je framework pro testování Spring MVC aplikace. Obsahuje nástroje pro testování controllerů a jejich interakcí s požadavky. Umožňuje simulaci běhu celé aplikace.
- Spring TestContext Framework: jedná se o rozšíření Springu pro testování, které má v sobě zaintegrováné nástroje, jako jsou JUnit a TestNG.

5.3.6 Spring výhody a nevýhody:

Výhody:

- Nenáročný, nevyžaduje velké množství zdrojů pro svoji funkci, což ho dělá vhodným pro zařízení s omezenými možnostmi.
- Modulární, to znamená, že programátor může využít pouze potřebné moduly pro svou aplikaci a ne celý framework.
- Podporuje dependency injection, která je jednou z největších výhod. Jedná se o návrhový vzor, který pomáhá oddělit komponenty aplikace. Zvyšuje jejich znovupoužitelnost, usnadňuje údržbu a testování.
- Spring dovoluje programátorům nakonfigurovat jejich aplikaci použitím XML, anotací tříd a samotným Java Kódem.
- Integrace Spring spojuje a podporuje spousty frameworků a technologií, jako jsou Hibernate a JUnit, což ho dělá vhodným nástrojem i pro enterprice aplikace. [22]

Nevýhody:

- Spring dokáže být se všemi svými možnostmi až přehnaně komplexní, díky čemuž se stává poměrně těžký na pochopení.
- Díky jeho širokým možnostem a s využitím mnoha knihoven může být výsledná aplikace pomalejší, než by tomu bylo u konkurenčních frameworků. [22]

5.4 Quarkus

„Quarkus vznikl, aby Java vývojářům umožnil vytvářet aplikace pro moderní cloudový svět. Quarkus je Kubernetes-native Java framework ušitý na míru pro GraalVM a HorSpot, sestavený z nejlepších knihoven a standardů.“ (přeloženo z [23]) Je rychlý, nenáročný a uzpůsobený pro kontejnerové nasazení nejčastěji v prostředí Kubernetes (platforma pro spravování kontejnerových aplikací více zde [24]). Obsahuje komplexní systém konfigurace.

Podporuje:

- reaktivní programování – jde o model, který umožňuje efektivnější práci s událostmi a asynchronními voláními, což je vývodné v architektuře mikroslužeb, kde se zpracovává velké množství požadavků najednou bez blokování hlavního vlákna. [25]
- nástroje pro zvýšení produktivity a zpříjemnění práce vývojáře, jako je live cooding, hot reloading a obsahuje zabudované nástroje pro Docker nebo třeba Git.

5.4.1 Quarkus principy fungování

Quarkus nabízí několik možností konfigurace CDI kontejneru (viz. níže), jako např. anotacemi, pomocí XML nebo konfiguračním souborem. Obsahuje mnoho vestavěných technologií, mezi které patří JAX-RS, Hibernate ORM nebo Apache Camel. Jádro má postavené na technologii mikroslužeb. To dovoluje rozpad na

menší nezávislé služby, kde každá z nich může být vyvíjena a nasazena bez ohledu na ostatní, což umožňuje snadné rozšíření aplikace. Díky této architektuře jsou jednotlivé služby (moduly) schopny běžet za pomoci minimálních zdrojů, například v kontejnerovém prostředí, jako je Kubernates. [26]

5.4.1.1 CDI (Contexts and Dependency Injection)

CDI je implementace DI používaná ve frameworku Quarkus. Umožňuje vkládání závislostí za běhu aplikace, správu tvorby a životního cyklu aplikačních komponent, které nazývá beany. Jedná se o komponenty spravované DI kontejnerem. Mohou obsahovat metody, tzv. metody životního cyklu. Ty vznikají pomocí anotací například @PostConstruct (metoda zavolána po vytvoření beanu), nebo @PreDestroy (metoda zavolána před smazáním beanu). Bean může být také svázána s určitou částí aplikace pomocí scope. Toto všechno má za následek efektivnější sdílení a opakované použití zdrojů napříč aplikací. K tomu přispívá i fakt, že se Quarkus snaží snížit používání reflexe za běhu na minimum, a aby toho dosáhl, využívá AoT kompilaci, která optimalizuje bytecode už při kompilaci aplikace. [26]

5.4.1.2 Microprofile

Microprofile je sada API a návrhových vzorů pro tvorbu mikroslužeb v Java aplikacích. Jejich součástí je sada specifikací a doporučení, která pokrývají různé aspekty mikroslužeb. Jsou implementovány jako Quarkus rozšíření.

Příklady:

- Config – Jak konfigurovat aplikaci
- Health – Jak zjistit zdravotní stav aplikace
- Metrics – Jak sbírat a zjistit data o výkonu aplikace
- JWT Propagatino – Jak zajistit odolnost proti chybám

5.4.2 Quarkus Web

Quarkus poskytuje mnoho API a rozšíření pro tvorbu webových aplikací včetně servletů, HTTP serveru, websocketů, Apache Camelu a jiných. Dalším velmi často využívaným frameworkem je RESTEasy. Ten je implementací JAX-RS API, které je součástí Java EE a obsahuje nástroje pro tvorbu REST API. Poskytuje anotace pro mapování HTTP požadavků na Java metody nebo třeba mapování výjimek, které převádějí běhové chyby na odpovědi s příslušným kódem, jako je 404 atd. Všechny tyto technologie mohou být přidány v podobě rozšíření, která poskytují potřebné závislosti a konfiguraci pro fungování s Quarkusem. [27]

Příklady anotací:

- Nastavení metody požadavku: @GET, @POST, @PUT, @DETELE
- Nastavení cesty požadavku: @Path

5.4.3 Quarkus Security

„Quarkus Security je framework, který poskytuje architekturu, několik mechanismů pro autentizaci a autorizaci a další nástroje pro vývoj bezpečných a kvalitních Java aplikací pro produkční prostředí „(přeloženo z [28]). Podporuje následující nástroje: OAuth 2.0, JWT, RBAC a SSL/TLS.

- Toto rozšíření má vestavěnou podporu pro autorizační a autentifikační mechanismy jako JWT (JSON Web Token), OAuth 2.0 a další.
- RBAC (Role Based Access Control) se stará o přístup uživatele do určitých částí aplikace na základě jeho role.
- Obsahuje podporu pro SSL/TLS šifrování, pro zajištění bezpečné komunikace mezi serverem a klientem.
- Quarkus Security poskytuje možnosti pro správu uživatelů, jako je bezpečné ukládání přihlašovacích údajů, zamykání účtů nebo restartování hesla.

5.4.4 Quarkus Data

Toto rozšíření poskytuje nástroje pro práci s databází a datovou vrstvou aplikace. Obsahuje několik podrozšíření pro přístup ke specifickým typům databází a databázovým technologiím. [29]

- Hibenate ORM
- Quarkus poskytuje podporu JDBC, což je standardizovaná Java knihovna pro přístup k relačním databázím. S tímto rozšířením je možné se připojit do jakékoliv databáze, která poskytuje JDBC driver.
- Panache je knihovna, která využívá Hibenate. Toto spojení usnadní psaní kódu pro přístup a manipulaci s daty. [30]
- Flyway je nástroj sloužící pro migrace databáze.

5.4.5 Quarkus Test

Quarkus test je rozšíření, které poskytuje podporu tvorby testů. Obsahuje několik utilit, anotací a tříd pro testování různých částí aplikace. Mimo toto rozšíření je možnost využít technologie jako JUnit nebo TestNG. [31]

- TestHTTPResource – utilita pro testování HTTP požadavků a odpovědí
- @QuarkusTest – anotace pro označení třídy jako Quarkus test, díky tomu se aplikace spustí v testovacím módu před provedením testu.
- @Mock – anotace sloužící pro mockování (simulaci) CDI bean nebo jiných zdrojů, nicméně tento přístup je zastaralý.

5.4.6 Quarkus výhody a nevýhody

Výhody:

- Quarkus používá AoT kompilaci, což mu umožňuje start v řádech milisekund, čímž se stává skvělou možností pro kontejnerové prostředí. A minimalizuje tím nutnost použití reflexe za běhu.
- Je nenáročný, to mu dovoluje běžet v prostředí s omezenými prostředky.
- Jeho výkon a modularita ho činí výhodným pro architekturu mikroslužeb.
- Poskytuje řadu vývojářských nástrojů a pluginů pro efektivní tvorbu kódu.

- Podporuje cloud-native technologie jako Kubernetes, OpenShift a další.

Nevýhody:

- Jedná se o relativně nový framework, a tak v porovnání frameworky, jako je Spring, nemá tak velkou podporu komunity, neboli disponuje menším množstvím pluginů a knihoven třetích stran.
- Osahuje svoji vlastní sadu API a paradigmat, což může vyžadovat nějaký čas k naučení.
- Byl navržen pro práci ve specifickém prostředí, čímž může být méně flexibilní ve spolupráci s jinými typy prostředí a architektur.

5.5 Micronaut

Micronaut je moderní full-stack Java framework založený na JVM, který je navrhovaný pro tvorbu modulárních a lehce testovatelných JVM aplikací s podporou pro Java, Kotlin a Groovy. (přeloženo z [32]) Micronaut podporuje mnoho nástrojů pro tvorbu aplikací, jako jsou DI, IoC, AoP, auto-konfigurace. Další technologie se týkají webové tvorby a mikroslužeb, například HTTP routování, load balancing (rozkládání zátěže), Service Discovery (zprostředkovává komunikaci mezi servery).

Cíle Micronautu:

- Minimalizovat použití reflexe.
- Vyhnout se proxy.
- Optimalizovat start a běh aplikace.
- Snížit paměťovou náročnost aplikace.
- Poskytnout jasně pochopitelné chybové hlášky.

5.5.1 Micronaut principy fungování

Vzhledem k tomu, že Micronaut je framework založený na JVM, tak využívá spoustu metod, aby se odbouralo zpomalení spojené s interpretací bytecodu pomocí JVM. Používají se funkce, jako jsou AoT kompilace, CDI a další optimalizační metody pro poskytnutí nenáročného a efektivního běhu aplikace.

- AoT redukuje čas potřebný ke startu aplikace
- Podpora reaktivního programování dovoluje odbavit větší počet požadavků najednou. Nabízí podporu pro RxJava a Reactor.
- Používá anotace pro validaci dat, konfiguraci a HTTP routování, čímž odstraňuje potřebu psát spoustu opakujícího se kódu.

5.5.1.1 IoC (Inversion of Control)

„Na rozdíl od ostatních frameworků, které se spoléhají na reflexi během běhu aplikace a proxy, Micronaut využívá data získaná v době kompilace pro implementaci dependenci injection." (přeloženo z [33])

5.5.1.2 CDI (Context and Dependenci Injection)

CDI je implementační metoda IoC, která se využívá ve frameworku Micronaut. Slouží ke snížení paměťové zátěže a zvýšení rychlosti startu aplikace. CDI už při kompilaci aplikace analyzuje závislosti a generuje optimalizovaný kód pro jejich propojení. Tímto se odbourává potřeba použití reflexe za běhu a také se snižuje s tím spojená režie. [33]

1. Micronaut skenuje kód a hledá třídy a rozhraní s podporovanými anotacemi. Mezi ně se řadí @Inject, @Singleton, @Bean a tak dále.
2. Po jejich identifikování sestaví graf závislostí, který reprezentuje vztahy mezi třídami a jejich propojení. Pro identifikaci potřebných závislostí se používá konstruktor a metody typu SET.
3. Následně je vygenerován bytecode, který reprezentuje tato propojení. Díky tomu se eliminuje potřeba již zmiňované reflexe.

4. Při startu aplikace je tento bytecode použit k instancování tříd a jejich provázání pomocí DI frameworku (Spring, Guice nebo vlastním Micronaut DI frameworkem). Protože graf závislostí byl vytvořen už při kompilaci, start a inicializace je mnohem rychlejší než v případě použití reflexe.

5.5.2 Micronaut Web

„Design HTTP serveru v Micronautu byl optimalizován pro přeposílání zpráv mezi mikroslužbami typicky v JSONu a nebyl zamýšlen jako plně funkční server-side MVC framework. Například momentálně nepodporuje žádné server-side views nebo některé další funkce tradičně spojované se server-side MVC frameworky." (přeloženo z [34]) To znamená, že se hodí ke tvorbě REST API. Aby toho dosáhl, obsahuje mnoho technologií, jako například:

- Obsahuje flexibilní routovací engine. Ten umožňuje konfigurace endpointů pomocí anotací, DSL (Domain-Specific language) a konfiguračních souborů.
- Micronaut má zabudovaného HTTP klienta.
- Podporuje SSE (Server-Send Events), který umožnuje serveru zasílat informace o událostech v reálném čase. Spojení probíhá prostřednictvím dlouho trvající HTTP komunikace, ale pouze ve směru ze serveru ke klientovi.
- WebSockety jsou další možností navázání komunikace v reálném čase. V tomto případě je možné posílat data i z klienta na server. Celá komunikace probíhá prostřednictvím jednoho TCP spojení.
- Možnost využití reaktivního programování umožnuje přijímat velké množství požadavků a je dobrým předpokladem pro budoucí rozšiřování aplikace.

5.5.3 Micronaut Securtity

Součástí Micronautu je i security framework, který disponuje širokou škálou možností, jak se bránit útokům. Například XSS, CSRF, SQL injection a další. K tomu využívá následující technologie. [35]

- Podporuje různé mechanismy autentifikace a autorizace. Například OAuth 2.0, JWT, LDAP a další. Mimo jiné je zde možnost ověření identity pomocí služeb jako Google nebo Facebook.
- Micronaut umožňuje konfiguraci přístupových práv k jednotlivým zdrojům aplikace. Lze tak učinit buď prostřednictvím anotací, nebo konfiguračního souboru.
- Micronaut security framework kooperuje s ostatními security frameworky, jako je Spring Security nebo Shiro.
- Ochrana proti XSS (Cross-Site Scripting) útokům je řešena zavedením různých bezpečnostních opatření. Například validací vstupů, kódováním výstupů nebo pomocí CSP (Content Security Policy). Ty specifikují data, která mohou být stránkou načtena. Tato politika je nastavena serverem při posílání odpovědi a nachází se v HTTP headers.

5.5.4 Micronaut Data

„Micronaut Data je nástrojová sada pro přístup k databázi. Používá AoT způsob kompilace k předzpracování dotazů pro rozhraní repositářů, které jsou poté spuštěny tenkou běhovou vrstvou." (přeloženo z [36]) Micronaut Data se inspirovalo v GORM a Spring Data, přičemž se je snaží vylepšit pomocí následujících postupů.

- Micronaut na rozdíl od Spring Data a GORM nevytváří žádný běhový model vztahů mezi entitami, neboť toto zabírá velké množství paměti, zvláště pak při použití Hibenatu. Ten si udržuje svůj vlastní model, a tím se zátěž zdvojnásobuje. Tento problém je řešen pomocí AoT kompilace.
- Pomocí AoT je také řešena tvorba dotazů. Dříve zmíněné technologie tvoří dotazy až za běhu aplikace pomocí regulárních výrazů a shodujících se vzorů (pattern matching).
- Při kompilaci se kontroluje implementace metod repositářů.

5.5.5 Micronaut Test

„Jedním z cílů Micronautu je eliminovat umělé rozdělení, které tradiční frameworky dávají mezi unit a funkcionální testy, z důvodu pomalého startu a spotřeby paměti" (přeloženo z [37]) Toto rozdělení vzniká, protože v případě unit testů se testuje pouze malá izolovaná část programu, například metoda. Ale u funkcionálních testů se testuje celá aplikace nebo její velká část. Je tedy vyžadován běh celé aplikace. Micronaut tento problém řeší tím, že přistupuje k funkcionálním testům tak, jako k rozsáhlým unit testům. Tímto způsobem je schopen spouštět pouze ty části, které se testu týkají, čímž snižuje nároky na paměť a zrychluje start testu.

Micronaut se také pokouší snížit potřebu jiných testovacích frameworků a knihoven na minimum. Nicméně pokud je potřeba použít některé funkce, které sám Micronaut nepodporuje, je zde možnost použít nástroje jako JUnit, Spock, Kotest a další.

5.5.6 Micronaut výhody a nevýhody

Výhody:

- Rychlé spuštění aplikace s nízkou zátěží systému, především paměti.
- Předkompilované dotazy pro repositáře pomocí AoT. Následkem je snížení paměťové náročnosti.
- Obsáhlá a dobře zpracovaná dokumentace.
- Možnost využít ostatní frameworky jako jsou Spring Data nebo Hibernate.

Nevýhody:

- Nepodporuje některé funkce MVC, zejména pak server-side views. Využívá se tedy spíše pro tvorbu mikroslužeb nebo REST API.
- Z důvodu nedávného vzniku frameworku je dostupný pouze omezený počet knihoven třetích stran.
- Tento framework ještě není moc rozšířený a existuje pouze malá komunita programátorů.

5.6 Testování Spring X Quarkus X Micronaut

Když přijde na porovnávání, vždy je nejdůležitější upřesnit, jaká jsou klíčová kritéria, podle kterých se budou dané technologie porovnávat. Je to čistě výkon, podporované technologie a knihovny nebo snad fakt, se kterou technologií se lépe pracuje? Dále je důležité upřesnit, co je se systémem zamýšleno do budoucna, jestli se bude rozšiřovat, nebo ne. Toto všechno závisí především na prostředí, ve kterém aplikace bude operovat. Pokud se bude jednat například o bankovní systém, musí být kladen velký důraz především na bezpečnost a spolehlivost systému. Na druhou stranu, pokud jde třeba o streamovací služby nebo práci v reálném čase, je žádoucí, aby odpověď byla doručena co nejdříve. Pokud se jeden z 0,1 % dotazů ztratí u přetížených služeb, není to takový problém, pokud stačí provést opětovné načtení stránky. Ale důležité je vědět, kolik dotazů ta desetina procenta představuje, i velmi malé procento může představovat velký unikající zisk.

V rámci této práce budou technologie z důvodu objektivity porovnány především na základě výkonu. Mezi posuzovaná kritéria se řadí:

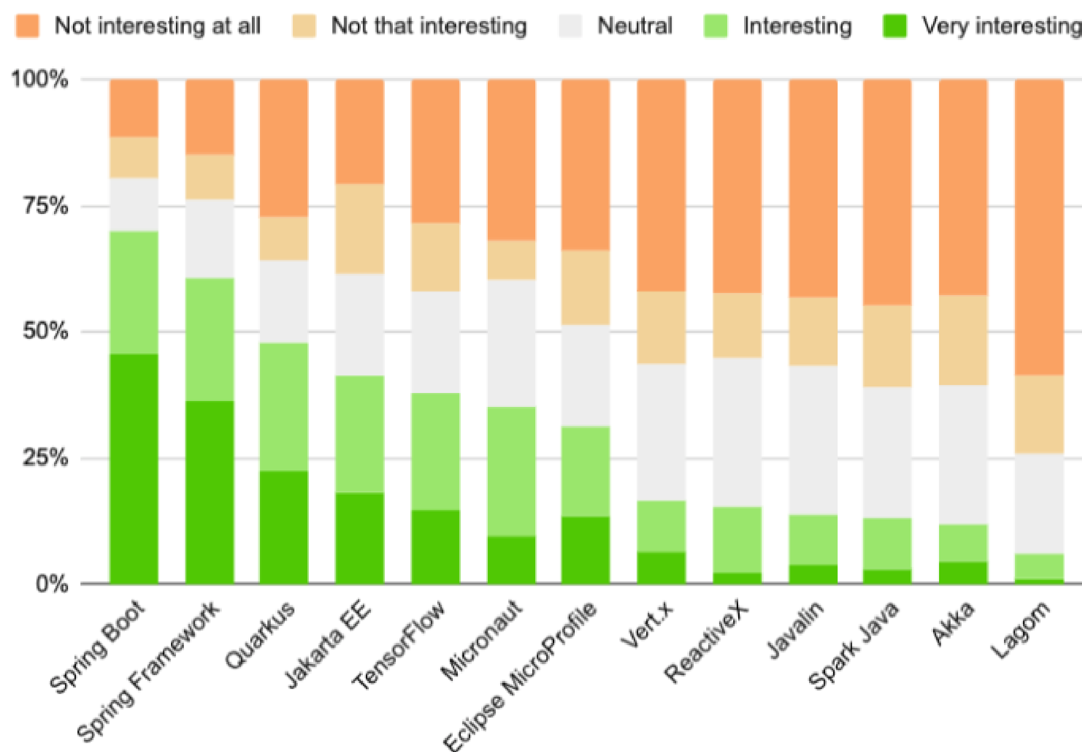
- Popularita technologie
- Průměrná doba kompilace
- Průměrný čas spouštění aplikace
- Průměrný čas odpovědi
- Počet úspěšných a neúspěšných provedení testovacího scénáře
- Využití CPU a RAM
- Počet provedených scénářů za minutu

Při konečném výběru technologie bude přihlédnuto k potřebám tvořeného informačního systému.

5.6.1 Popularita Java frameworků

Na obrázku níže lze vidět, že nejpopulárnějším frameworkem je Spring (Spring boot), a nelze se tomu divit, když původní verze Springu vyšla už v roce 2004. To ho dělá po Jakarta EE (1999) druhým nejstarším frameworkem z tohoto seznamu. Právě z důvodu jeho popularity se stal v průběhu let nejrozšířenějším, a to mělo za následek vznik velkého množství knihoven třetích stran a rozsáhlou komunitu. Tento fakt může být klíčový k řešení některých krajních případů. Na druhou stranu Quarkus (2019) a Micronaut (2018) jsou nejnovější frameworky na tomto seznamu a také se umístily velmi vysoko. Za to může především jejich nenáročnost a moderní architektura, která je přizpůsobena trendu mikroslužeb a kontejnerovému nasazení.

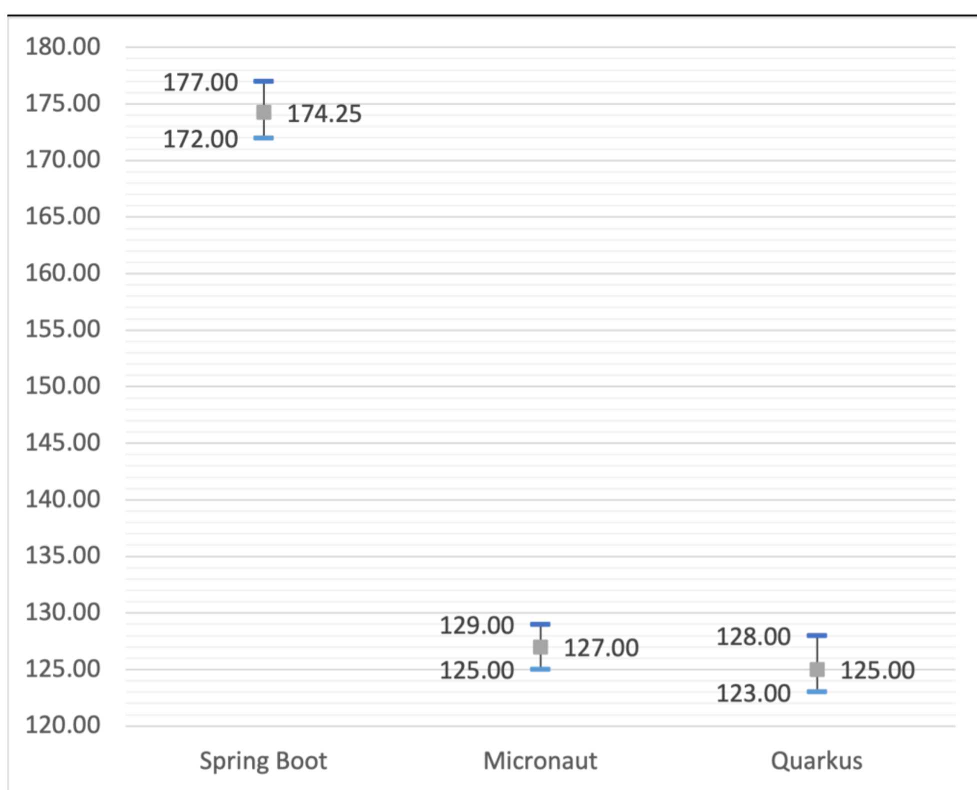
Následující graf vychází z dotazníku provedených společností JAXenter z roku 2020. Lidé hlasovali, jak je každá z následujících technologií zajímá na škále 1–5.



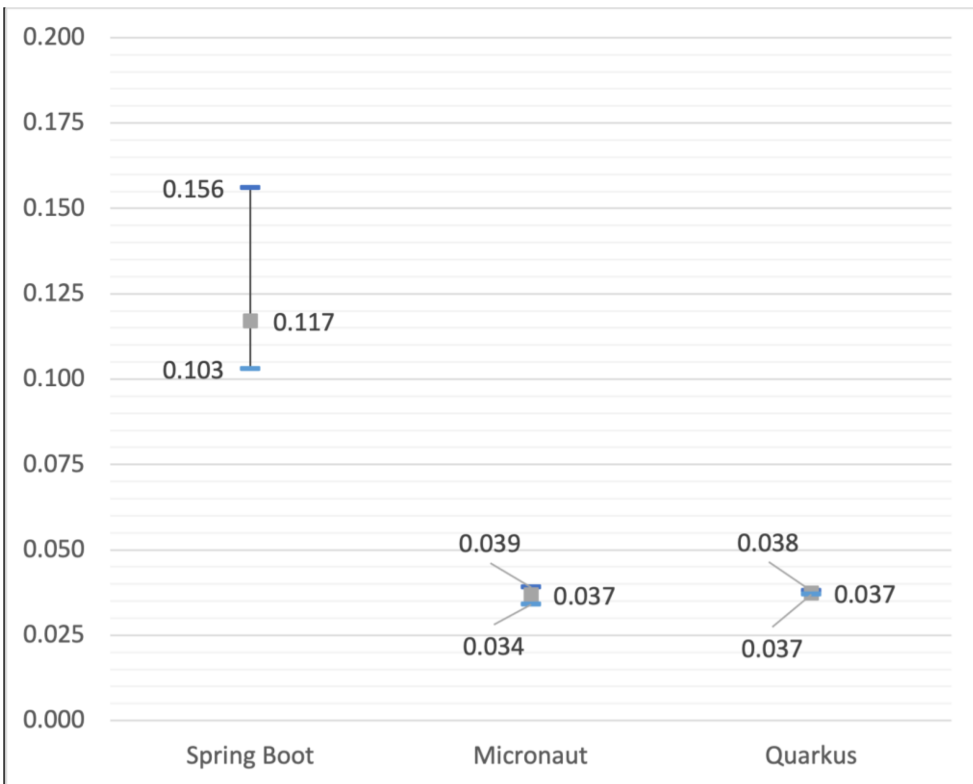
Obrázek 11 Popularita Java frameworků
Zdroj: [38]

5.6.2 Kompilace a start aplikace

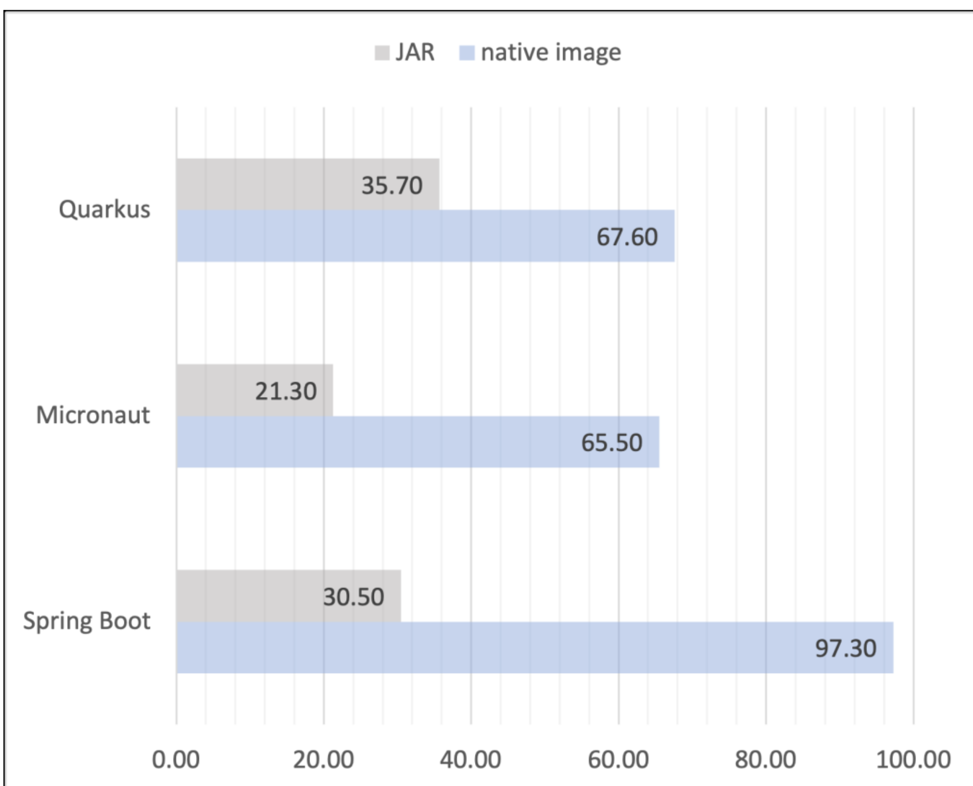
Pro testování kompilace aplikace byl použit nástroj Gradle building tool. Testů bylo provedeno celkem 5. Doba startu byla odvozena na základě logů uvedených v jednotlivých frameworcích. Taktéž bylo provedeno 5 testů. Na grafech je možné pozorovat lepší výsledky ve všech třech kategoriích u Micronautu a Quarkusu, který byly zpočátku navrhovány a optimalizovány pro kontejnerové použití, tvorbu nativních aplikací a cloud computing.[38][39]



Obrázek 12 Průměrná doba kompilace native image (s)
Zdroj: [39]



Obrázek 13 Průměrná doba startu native image (s)
Zdroj: [39]



Obrázek 14 Velkost spustitelného souboru
Zdroj: [39]

5.6.3 Testované scénáře

Pro test výkonu byly vytvořeny scénáře, které jsou rozdělené do dvou kategorií. Zátěžové testy a testy normálního provozu. Při každém dotazu jsou použita unikátní náhodně generovaná data. [39]

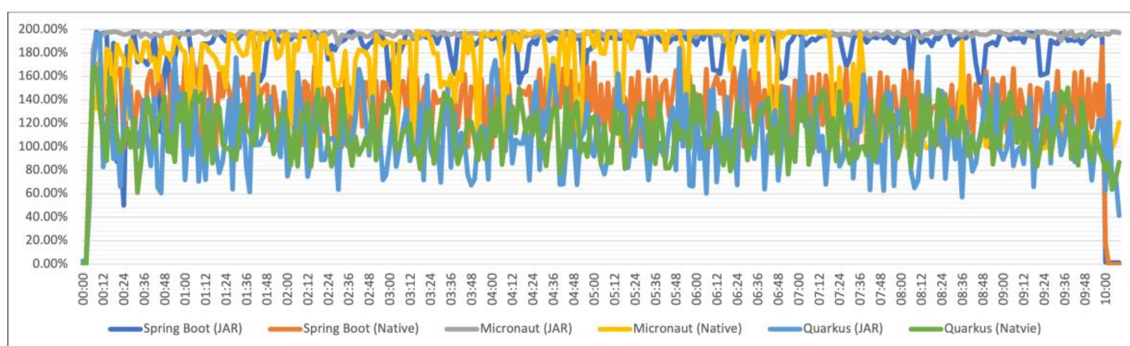
5.6.3.1Zátěžové testy

Zátěžové testy pomáhají odhalit chování systému za extrémních podmínek. Cílem těchto testů je zjistit maximální kapacitu systému, případně odhalit problémové oblasti jako úzká hrdla, která mohou způsobit přetěžování serveru. V takovém případě můžou být následkem pomalejší odpovědi, nedostupnost a nespolehlivost serveru nebo dokonce bezpečnostní rizika. Takový server je mnohem náchylnější k útokům typu DoS, který zasílá velké množství požadavků a tím dále přetěžuje server.

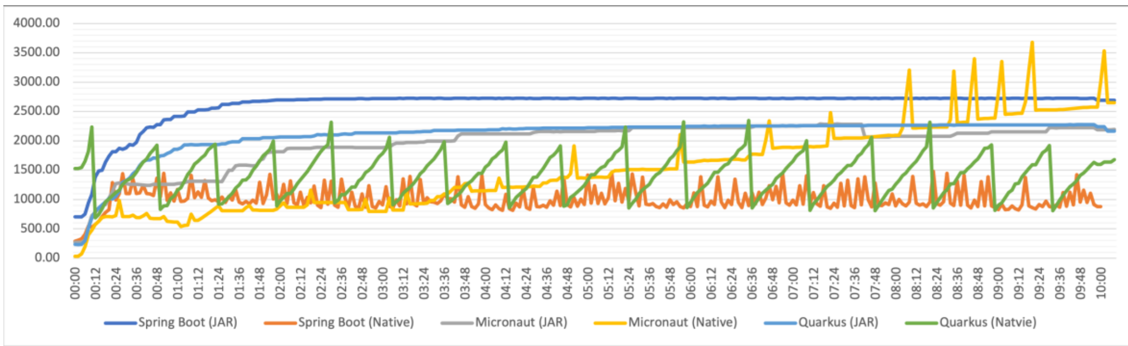
5.6.3.1.1 Tvorba hledání a mazání uživatele

V následujícím scénáři bylo vytvořeno 20 000 virtuálních uživatelů, kteří měli za úkol:

V systému vytvořit uživatele → zkontrolovat, zda byl vytvořen → načíst uživatele ze systému → zkontrolovat jeho data → odstranit uživatele ze systému → zkontrolovat, zda byl odstraněn pokusem o jeho opětovné načtení. [39]

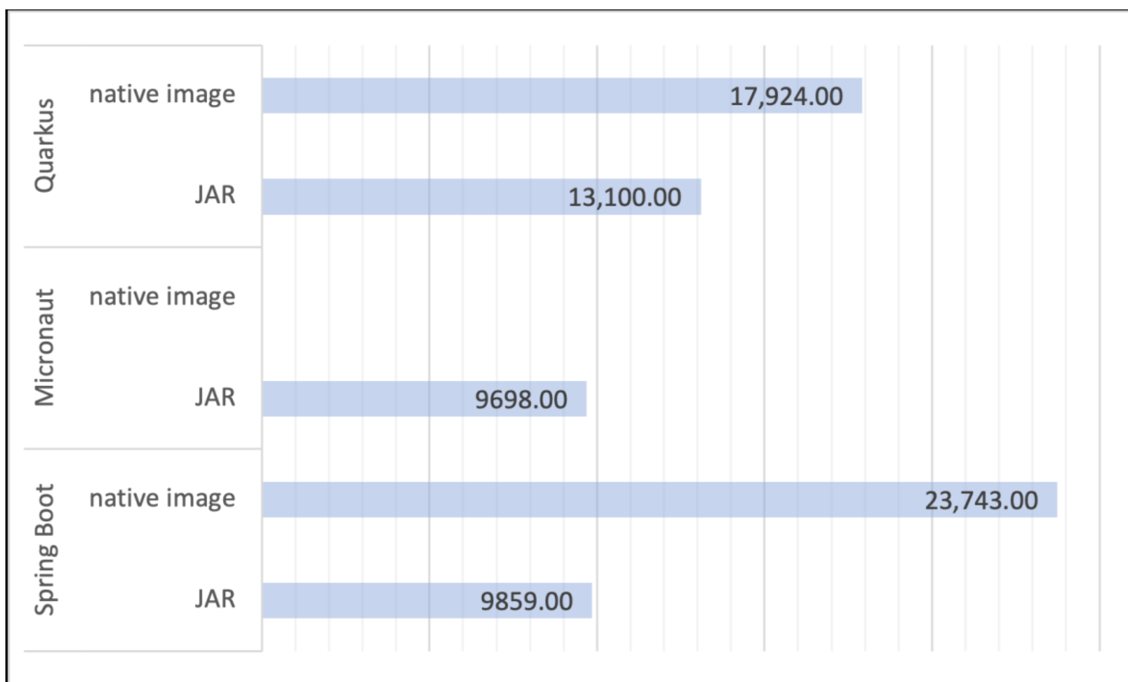


Obrázek 15 Průměrné využití CPU (%)
Zdroj: [39]



Obrázek 16 Průměrné využití RAM (MiB)
Zdroj: [39]

Obrázek 17 Počet úspěšných a neúspěšných testovacích scénářů
Zdroj: [39]



Obrázek 18 Průměrná doba odpovědi (ms)
Zdroj: [39]

5.6.3.1.2 Test zpracování velkého množství dat

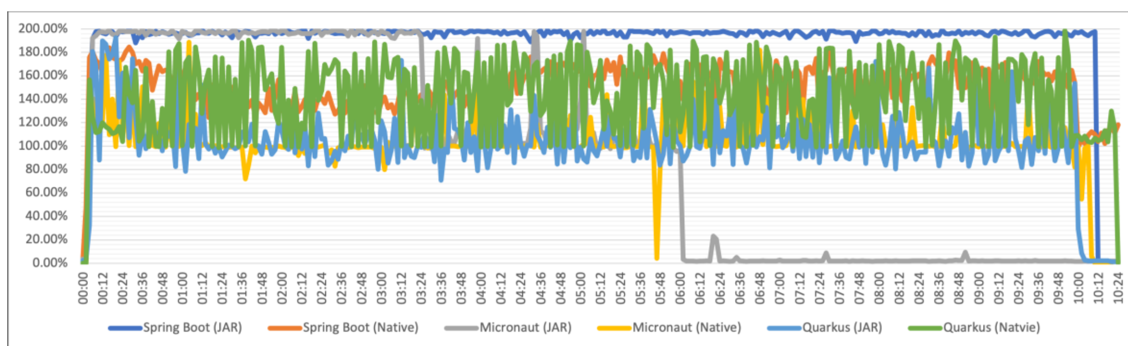
V následujícím scénáři bylo vytvořeno 5 000 virtuálních uživatelů, kteří měli za úkol:

Seřadit 10 000 prvků pomocí merge sortu → zkontrolovat výsledek

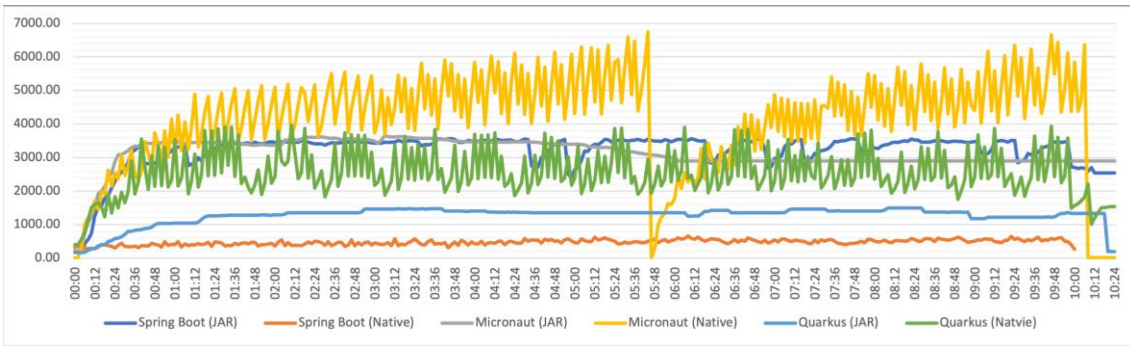
Seřadit 10 000 prvků pomocí quick sortu → zkontrolovat výsledek

Seřadit 10 000 prvků pomocí heap sortu → zkontrolovat výsledek

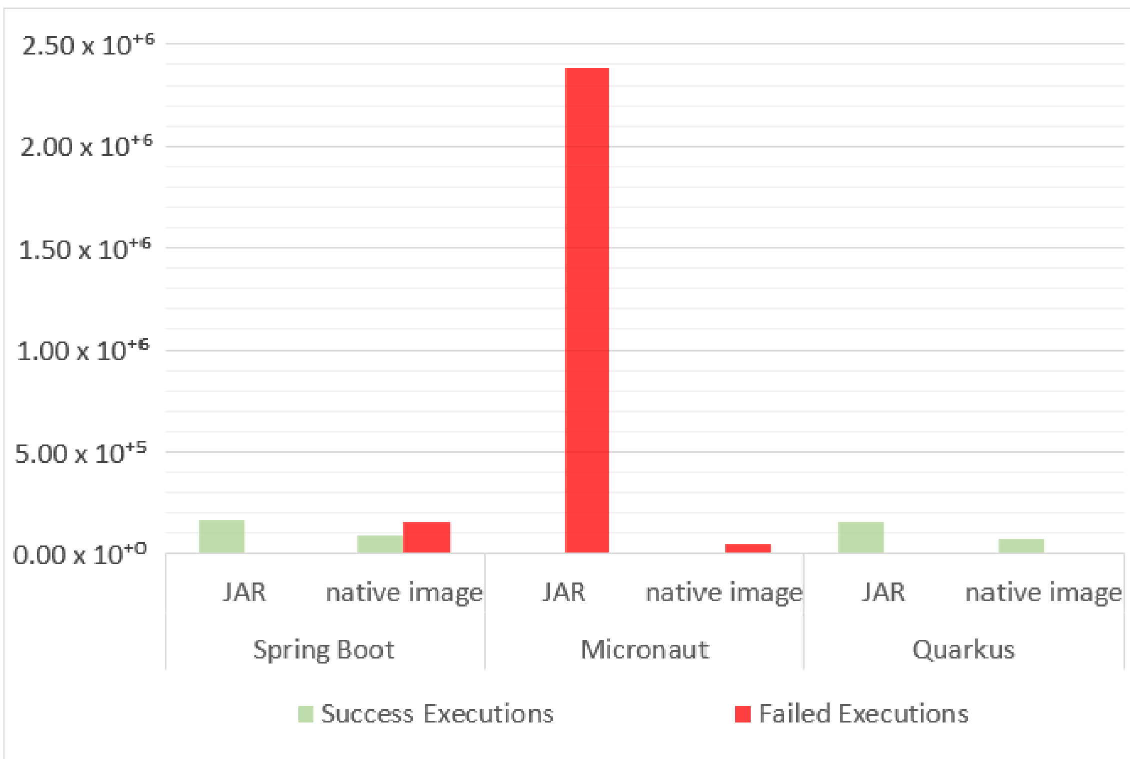
[39]



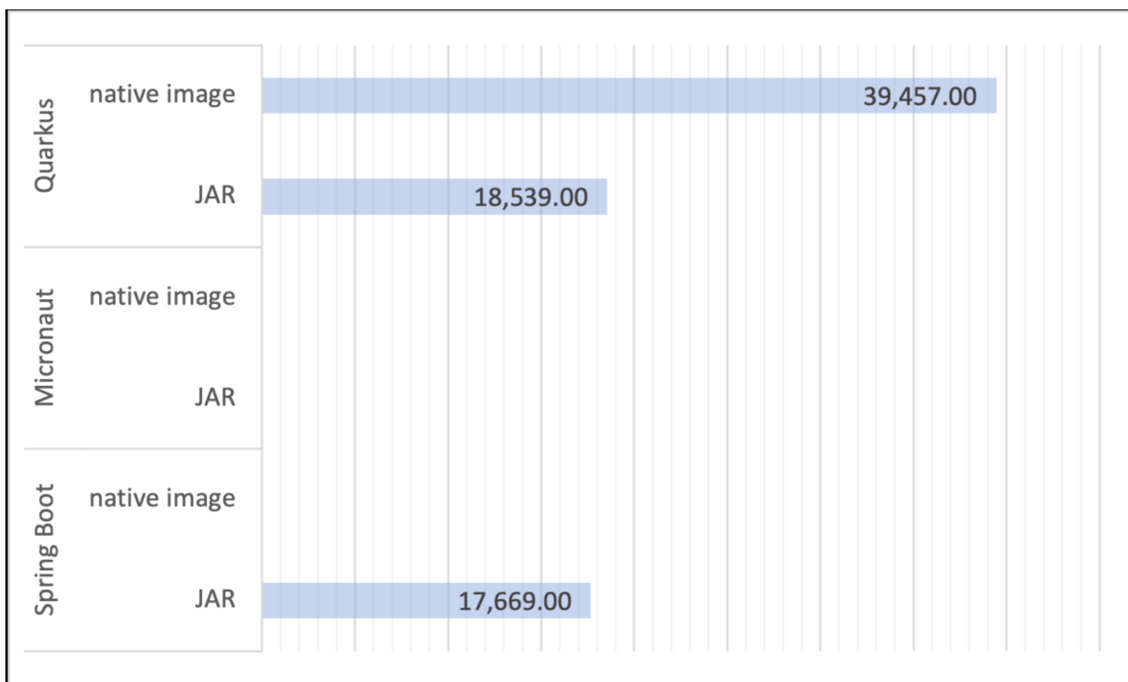
Obrázek 19 Průměrné využití CPU (%)
Zdroj: [39]



Obrázek 20 Průměrné využití RAM (MiB)
Zdroj: [39]



Obrázek 21 Počet úspěšných a neúspěšných testovacích scénářů
Zdroj: [39]



Obrázek 22 Průměrná doba odpovědi (ms)
Zdroj: [39]

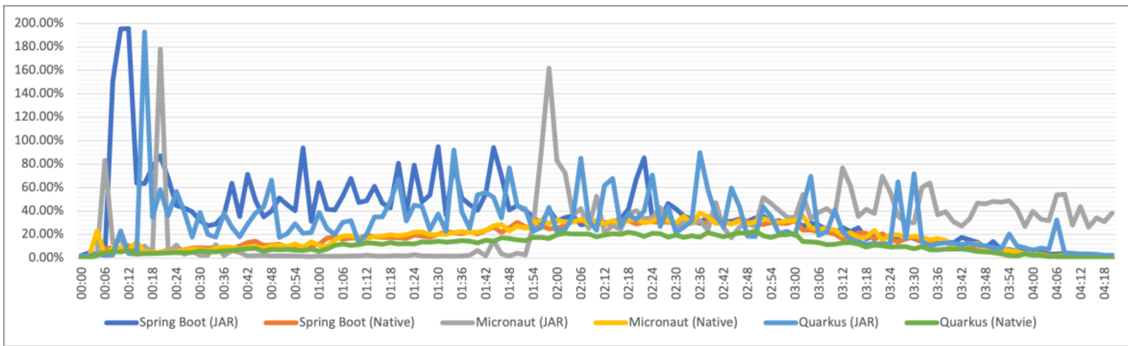
5.6.3.2 Test běžného provozu

Tyto testy simulují běžný provoz serveru. Na základě těchto testů se může společnost rozhodnout, jestli by nebylo vhodné zvýšit výkon například přidáním serverů a využít například funkce load balancing (vyrovnávání zátěže). Tato funkce má za úkol rozesílat požadavky na několik dalších serverů se stejnou funkcionalitou. Uživatel nic nepozná, protože se stále bude odkazovat na stejný server tzv. gateway. Zátěž ale bude rozdělena mezi více serverů, čímž dojde ke snížení zátěže na jednotlivých serverech. Stejně jako u zátěžových testů i test běžného provozu může odhalit problémové sekce v systému.

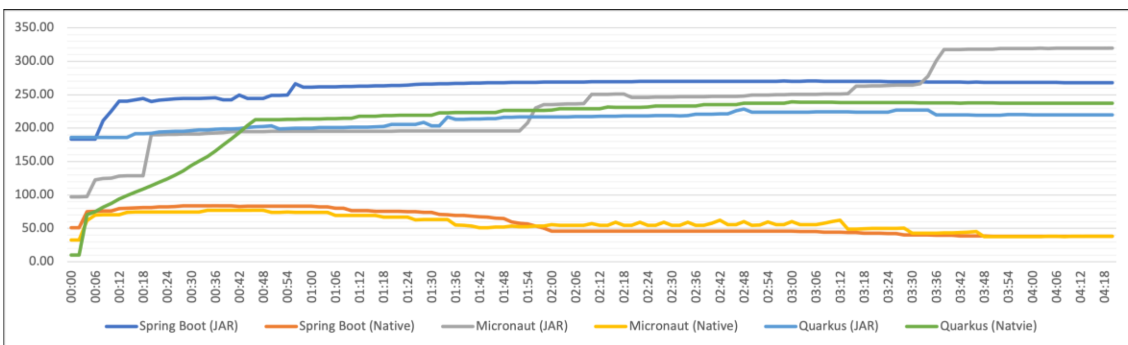
5.6.3.2.1 Tvorba hledání aktualizování a mazání uživatele

úkol:

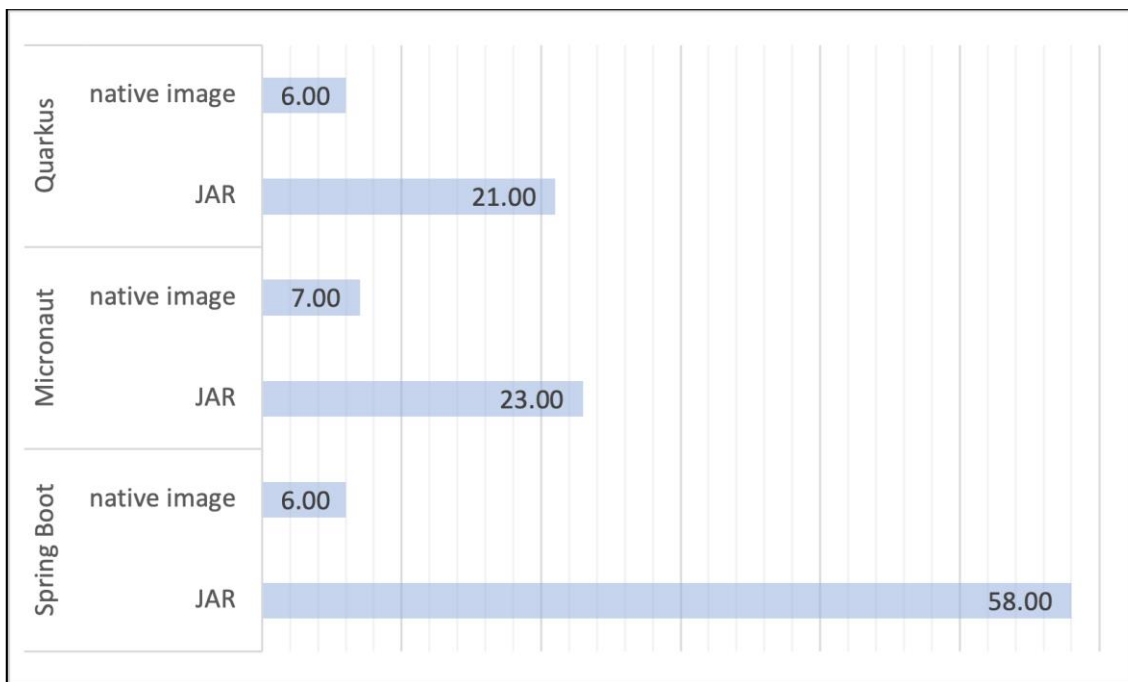
V systému vytvořit uživatele → zkontrolovat, zda byl vytvořen → načíst uživatele ze systému → zkontrolovat jeho data → aktualizovat uživatelova data → opět je zkontrolovat → odstranit uživatele ze systému → zkontrolovat, zda byl odstraněn pokusem o jeho opětovné načtení. [39]



Obrázek 23 Průměrné využití CPU (%)
Zdroj: [39]



Obrázek 24 Průměrné využití RAM (MiB)
Zdroj: [39]



Obrázek 25 Průměrná doba odpovědi (ms)
Zdroj: [39]

5.6.4 Diskuse nad výsledky testů

Na základě výše uvedených skutečností testů lze vyvodit závěry:

Při pohledu na obrázky 7–9 je vidět, že lepších výsledů dosáhly Micronaut společně s Quarkusem, které byly primárně přizpůsobeny pro tvorbu mikroslužeb a nativních aplikací. Co se týče doby spouštění aplikace je vidět, že Spring zaostává, zejména protože využívá pro správu závislostí reflexi přímo za běhu aplikace. Obě konkurenční technologie se snaží minimalizovat tuto potřebu za použití AoT, které připravuje závislosti už při kompilaci. Podporu AoT se Spring dočká s příchodem verze 6.0. Na druhou stranu by Spring měl mít kratší dobu kompilace právě díky absenci AoT, což jak je vidět není pravda. Především z důvodu toho, že Spring je starší nástroj s robustnějším jádrem a více závislostmi. Ale toto všechno závisí především na použitých modulech a knihovnách.

Při pohledu na obrázky 10–17 je vidět, že z hlediska rychlosti odpovědi vyhrává Quarkus. Při velké zátěži je ale patrné, že novější nativní technologie proti zaběhlým JAR souborům stále zaostávají jak v rychlosti odpovědi, tak především ve

spolehlivosti. V těchto testech nejvíce pohořel Micronaut s minimální úspěšností odpovědi.

Při testování normální zátěže viz. Obrázky 18, 19 a 20 je vidět, že při dostatku systémových zdrojů je možné dosáhnout prostřednictvím nativního souboru více než trojnásobné rychlosti odpovědi u všech testovaný technologií. Co se týče nativní technologie, všechny frameworky dosahují téměř totožné rychlosti. Pokud bude brána v potaz i rychlost odpovědi u JAR souboru, vítězem se stává Quarkus.

5.7 Shrnutí výsledků porovnání

Po prozkoumání všech tří frameworků jak z hlediska výkonu, spolehlivosti i podporovaných technologií stojí za zmínku, že neexistuje univerzální způsob, jak vybrat nejlepší technologii. Vždy je nutné uvažovat, k čemu bude využita. Na základě výše provedených testů lze určit, že nejrychlejší je Quarkus společně s Micronautem, který, jak bylo zjištěno, velmi trpí při velké zátěži, což ho nedělá vhodným pro podnikové aplikace. Na druhou stranu Spring je velmi populárním spolehlivým a široce rozšířeným nástrojem s rozsáhlou škálou knihoven a velkou komunitou.

Tabulka 1 Podporované technologie

	Spring	**Quarkus**	**Micronaut**
Web	Spring Web, Spring MVC	RESTEasy, Vert.x,	Micronaut Servlet, Netty
Testing	JUnit, Mockito	JUnit 5, Mockito	JUnit 5, Spock
Data Access	Spring Data, Hibernate ORM	Hibernate ORM, Panache	Micronaut Data, JOOQ
Security	Spring Security, Spring LDAP	Quarkus Security, Keycloak, SmallRye JWT	Micronaut Security, Micronaut is JWT
Výhody	Rozsáhlá dokumentace, Robustní platforma, Velká komunita, Mnoho knihoven	Rychlí start aplikace, Nízká spotřeba paměti, Možnost tvorby native image	Rychlí start aplikace, Nízká spotřeba paměti, Možnost tvorby native image
Nevýhody	Pomalejší spuštění aplikace, Paměťová náročnost	Limitovaná podpora některých Java EE standardů, Malá komunita	Limitovaná podpora některých Java EE standardů, Malá komunita

Zdroj: Autor práce

Autor práce se rozhodl pro tvorbu informačního systému za použití frameworku Spring (Spring boot). Společnost, pro kterou je systém tvořen, má se Springem také zkušenosti, i předešlý sytém byl v něm vytvořen. Nejzásadnějším kritériem při rozhodování byl účel. Systém bude operovat s financemi, a tak je žádoucí použít technologii, která je za dobu své existence osvědčená s vysokou mírou spolehlivosti a možnostmi zabezpečení.

6 Praktická část: Informační systém

Tvořený systém je určen pro vnitrofiremní použití. Jeho hlavní funkcionalitou je rozdělování odměn zaměstnancům. Jelikož je tvořen jako nástupce již existujícího systému, byl také sepsán seznam nových funkcionalit, které zaměstnanci postrádají.

V rámci této práce byl zpracován pouze backend. Frontend bude realizován mimo rozsah této práce. S ním spojené funkcionality, jako import a export dat, budou v budoucnu také dopracovány. Technologie, které jsou v plánu použít při implementaci frontendu jsou uvedeny v kapitole 6.2.2. Frontend.

6.1 Funkcionality

Základní funkcionalita:

Jednou za určité období je každému ze zaměstnanců přidělen obnos peněz, který mají za úkol rozdělit mezi své spolupracovníky. Rozdělování probíhá tak, že zaměstnanec vybere svého kolegu, kterého chce odměnit, určí obnos ze svého rozpočtu a přiloží zprávu pro příjemce. Po skončení období administrátor vyexportuje soubor s celkovými odměnami, který zašle na příslušné oddělení k zúčtování.

Nové funkcionality:

- Aby si zaměstnanci nemuseli pamatovat další přihlašovací údaje, nový systém podporuje přihlašování lidí pomocí jejich google účtů, které mají v rámci firmy.
- Možnost tvorby skupin, pro jednodušší přidělení odměn. Člověk vytvoří skupinu, přidá do ní kolegy a následná odměna je rozpočítána mezi všechny její členy.
- Nový systém umožní vytvářet koncept odměn již v probíhajícím časovém úseku. Tyto odměny jsou ukládány jako rozepsané a je možné jejich následné upravení nebo smazání.
- Možnost importovat odměny ve formě CSV souboru. To umožní efektivnější tvorbu většího počtu odměn.

- Graf se zobrazením všech zaměstnanců a vazeb (odměn) mezi nimi.
- Okno se statistikami, jako například, kdo byl nejvíce odměňovaný zaměstnanec a podobně.
- Zasílání připomínkových emailů před koncem odměňovacího období.

6.2 Vybrané technologie

Pro tvorbu systému bylo využito velké množství frameworků a knihoven, které jsou dále rozděleny na frontend a backend.

6.2.1 Backend

Jak již bylo zmíněno, pro tvorbu backendu byl vybrán Java framework Spring. Serverová část aplikace je řešena jako REST API, tedy neobsahuje žádné pohledy a slouží pouze pro získání dat, která jsou žádaná na frontendu. Bylo také použito několik technologií, mezi které patří Lombok, Spring Web, Open Authorization 2.0, Model Mapper, Postgresql (myšleno driver) a JPA (Java Persistence API).

- Spring Web – Poskytuje základní nástroje pro tvorbu webových aplikací.
- Open Authorization 2.0 – Je autorizační protokol používaný pro zabezpečení webových API. Umožnuje autorizaci uživatelů prostřednictvím služeb jako Google nebo Github. Díky tomu je možné získat informace o přihlášených uživatelích bez nutnosti, aby museli sdílet své přihlašovací údaje.
- Lombok – Je nástroj pro generování stále se opakujícího kódu tzv. „boilerplate code“ v jazyce Java. Tím snižuje trvání vývoje a dává prostor věnovat se aplikační logice. Příkladem boilerplate kódu jsou gettery, settery nebo konstruktory.
- Model Mapper – Byl použit na mapování objektů mezi vrstvami aplikace.
- Postgresql (driver) – Je knihovna, která zprostředkovává komunikaci mezi aplikací a Postgresql databázovým systémem.
- Java persistence API – Jedná se o standartní rozhraní pro komunikaci s databází, které nabízí možnosti mapování objektů na databázové tabulky.

Jeho součástí jsou i předpřipravené nejpoužívanější metody pro manipulaci s daty, jako jsou SAVE, FIND a DELETE. Další pokročilejší dotazy musí být tvořeny ručně.

6.2.2 Frontend

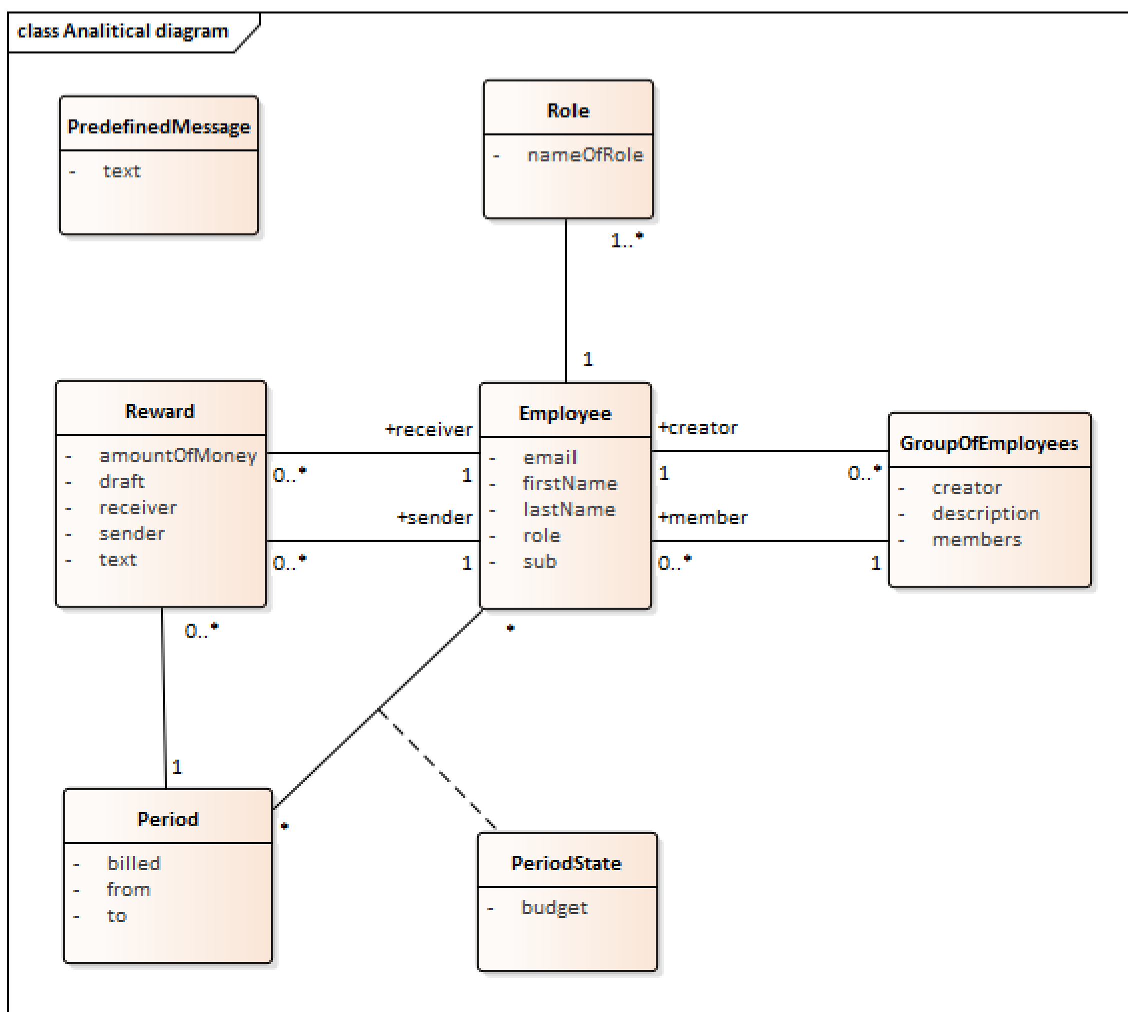
Frontendová část aplikace bude tvořena pomocí knihovny ReactJS, především z důvodu možnosti tvorby znovupoužitelných komponent a manipulace s nimi. Protože Javascript ani ReactJS nepodporují specifikování datových typů proměnných, bude pro doplnění této funkcionality použit framework Typescript. A nakonec pro tvorbu vzhledu aplikace bude použita technologie Material 3. Mezi další použité technologie patří například Axios, React Router a Formik, D3js.

- Typescript – Je framework používaný na frontendu pro zavedení struktury dat. Na výslednou aplikaci nemá žádný vliv, ale je nápomocný při tvorbě aplikace. S jeho pomocí lze například definovat datové typy parametrů a návratových hodnot metod.
- Axios – Je knihovnou, která usnadňuje práci s dotazy zasílanými na serverové API. Využívá pro zpracování asynchronní komunikace objekty typu „Promis“. Tyto objekty mohou nabývat hodnot pending (čekající), fulfilled (splněná) a rejected (odmítnutá).
- React Router – Rozšiřuje ReactJS a umožnuje jednoduchou tvorbu navigace pomocí toho, že zobrazuje nebo skrývá jednotlivé komponenty v závislosti na právě vybrané URL adrese. Přidává komponenty typu „Link“ pro konkurování mezi jednotlivými pohledy. Tyto komponenty na rozdíl od standartního tahu <a> nenačítají celou stránky znovu, ale pouze načte komponentu, která má být vykreslena na konkrétní URL.
- Formik – Jedná se o knihovnu pro správu formulářů a manipulaci s daty v nich. Poskytuje řadu funkcí a komponent, které umožnují tvořit a validovat vstupní data formulářů. Umožnuje tvorbu vlastních validátorů, tyto validátory mohou obsahovat počáteční hodnoty, vlastní chybové hlášky a další.

- D3.js – Pro tvorbu již zmiňovaného grafu bude použita knihovna D3.js, která poskytuje mnoho způsobů vizualizace dat. Data mohou být vizualizována pomocí grafů, diagramů a map. Dále obsahuje funkce pro interaktivní manipulaci s těmito daty.

6.3 Návrh databáze

Databáze byla navržena podle systémových požadavků v aplikaci Enterprise Architect. Jak již bylo zmíněno, jako databázový sytém byl vybrán Postgresql, ke kterému je aplikace připojena pomocí Postgresql driveru.



Obrázek 26 Schéma databáze
Zdroj: Autor práce

Na obrázku výše lze vidět návrh databáze se všemi databázovými entitami společně s jejich atributy.

- Tabulka zaměstnanec obsahuje email, jméno, příjmení, výčet rolí a unikátní identifikátor (sub), který je získán po přihlášení prostřednictvím služby Google.
- Tabulka skupina zaměstnanců obsahuje popis, seznam členů a zaměstnance, který ji vytvořil.
- Tabulka odměna obsahuje zprávu, odesílatele, příjemce, stav (rozepsaná/odeslaná) a její výši.
- Tabulka období obsahuje status stavu zaúčtování a jeho časový úsek od-do. Období je tvořeno administrátorem.
- Tabulka role je číselník obsahující výčet existujících rolí.
- Tabulka předdefinovaná zpráva slouží pouze jako výčet před generovaných zpráv, které mohou být použity při odesílání odměny.
- Tabulka stav období obsahuje rozpočet zaměstnance pro dané období.

Pro manipulaci s daty byly využity dotazy definované rozhraním JPA. Tvorba a zasílání složitějších dotazů bylo realizováno prostřednictvím nativních dotazů SQL.

```
@Repository
public interface PeriodRepository extends JpaRepository<Period,Long> {

    @Query(value = "SELECT * FROM period WHERE now() BETWEEN start_of_period AND end_of_period;", nativeQuery = true)
    Optional<Period> currentPeriod();
}
```

Obrázek 27 Příklad repositáře a nativního SQL dotazu
Zdroj: Autor práce

Na obrázku výše je vidět JPA repositář, který byl využit pro ukládání dat o období. Samotný dotaz vrací právě probíhající období na základě jeho začátku a konce.

6.4 Návrh API

Backendová část aplikace byla tvořena jako REST API, to znamená, že nevrací žádné kompletní stránky (pohledy), ale pouze posílá data, která jsou požadována klientem. Celý backend se skládá z několika controllerů pro správu zaměstnanců, odměn, skupin a podobně.

6.4.1 Endpointy

Při tvorbě endpointů byly využívány HTTP metody GET, POST, PUT a DELETE. V rámci odpovědí byly použity následující status kódy:

- 200 – Při kladném zpracování požadavku o zaslání dat.
- 204 – Při kladném zpracování požadavku o smazání dat.
- 403 – Při vstupu neoprávněného uživatele na zabezpečený endpoint.
- 404 – Při nenalezení entity v databázi, se kterou má být manipulováno.

Ukázka vytvořených endpointů:

6.4.1.1Přidělení rozpočtu

Controller: /periodState

URL: /assignBudget

Metoda: POST

Potřebná role: ADMIN

Ukázka:

```
//ASSIGN BUDGET
@PostMapping(path = "/assignBudget", consumes = "application/json", produces = "application/json")
@Transactional
@PreAuthorize("@permissionEvaluator.adminRole(principal)")
public ResponseEntity<List<PeriodStateOut>> assignBudget(@RequestBody List<PeriodStateIn> periodStateIn){
    return ResponseEntity.ok(periodStateService.assignBudget(periodStateIn));
}
```

Obrázek 28 Endpoint přidělení rozpočtu
Zdroj: Autor práce

Popis: tento endpoint slouží administrátorům pro přidělení rozpočtu zaměstnancům. Vstupní data jsou tvořena jako dvojice zaměstnanec – částka. Na základě těchto dvojic jsou poté tvořeny objekty, které představují vztahy mezi zaměstnanci a obdobím.

6.4.1.2 Zobrazení všech skupin vytvořených zaměstnancem

Controller: /groupOfEmployees

URL: /myGroups

Metoda: GET

Potřebná role: libovolný přihlášený zaměstnanec

Ukázka:

```
//MY GROUPS
@GetMapping(path = "/myGroups", produces = "application/json")
public ResponseEntity<List<GroupOfEmployeesOut>> myGroups(@AuthenticationPrincipal OAuth2AuthenticatedPrincipal principal) {
    return ResponseEntity.ok(groupOfEmployeesService.getMyGroups(principal.getAttribute( name: "sub")));
}
```

Obrázek 29 Endpoint zobrazení vytvořených skupin
Zdroj: Autor práce

Popis: tento endpoint slouží zaměstnancům pro výpis všech skupin, které si vytvořili. Výpis obsahuje název, popis skupiny a informace o jejích členech.

6.5 Problémy

Tato sekce se věnuje problémům, se kterými se autor práce setkal při tvorbě API.

1. Tvorba nativního souboru – Při tvorbě nativního souboru nastal problém s nastavováním systémových proměnných, jako je JAVA _HOME, protože aplikace měla problém nalézt cestu k potřebným příkazům uloženým v Javě a GraalVM. Následně bylo nutné doinstalovat Windows 10 SDK, aby bylo možné spustitelný soubor vytvořit (v tomto případě pro Windows 10). Nakonec bylo zjištěno, že příkaz lze provést pouze za použití 64bitového

příkazového řádku, který byl následně doinstalován prostřednictvím Visual Studia. Podrobný návod ke tvorbě nativního souboru je uveden zde [40].

2. Autentifikace prostřednictvím služby Google – Ještě před začátkem programování je nutné registrovat projekt na stránkách Console Cloud Google. Přihlašování probíhalo v pořádku, problém nastal při odhlašování, protože žádná dokumentace se nezmiňuje o tom, že požadavek o odhlášení je nutné zaslat metodou POST.
3. Nestandardní chování – Při testování API nastal problém, kdy uživatel mohl být členem své vlastní slupiny, neboli mohl jejím prostřednictví odměňovat sám sebe.

7 Shrnutí výsledků

Úspěšně se podařilo navrhnout a implementovat backendovou část informačního systému. Jako framework pro jeho tvorbu byl vybrán Spring (Spring boot), protože je nejvíce rozšířený a nejspolehlivější z výše porovnávaných. Při pohledu na porovnání je patrné, že Quarkus a Micronaut také dosahují velmi dobrých výsledků, ve většině testů byly schopny překonat Spring, ale problém u nich nastal při zátěžových testech. V tomto případě obzvláště Micronaut nebyl schopen odbavit téměř jediný požadavek. To je nepřípustné, a to zejména, když má aplikace pracovat s financemi.

Backend funguje na principu REST API a pouze zasílá data na prohlížeč. Přihlašování bylo na základě požadavků zprostředkováno prostřednictvím služby Google. Samotná data aplikace jsou pak ukládána pomocí databázového systému Postqresql.

7.1 Budoucí pokračování

Tento systém a především frontend bude ještě dopracováván, protože pro jeho zpracování ještě nebyly dodány podklady od zadavatele. Prvního praktického využití se systém dočká koncem roku 2023. Poté bude sbírána zpětná vazba od uživatelů a na jejím základě bude systém dokončen.

8 Závěry a doporučení

Při výběru frameworku je vždy nutné zohlednit, za jakým účelem je aplikace vyvíjena. Dalšími důležitými vlastnostmi jsou podporované technologie a knihovny. Na základě uvedených testů je vidět, že frameworky se liší především ve spolehlivosti, kde vyhrává Spring. Mimo jiné se s ním lépe pracuje především z důvodu množství již existujících návodů a dokumentací. To neznamená, že zbylé frameworky (Quarkus a Micronaut) jsou špatné, ale autor práce se domnívá, že by bylo rozumné je využít spíše pro experimentální účely a vyčkat na budoucí verze, které budou stabilnější. Co se týče samotného porovnání, frameworky byly porovnávány především na základě výkonu. Zde by jistě bylo možné detailnější prozkoumání vnitřní architektury a samotného fungování frameworků. Nebo popřípadě vytvořit stupnici, na které se porovnají jednotlivé aspekty aplikace (testování, přístup k datům, atd.), a následně vybrat technologii na jejich základě.

9 Seznam použité literatury

[1] Introduction to the server side [online]. San Francisco: Mozilla Corporation, ©1998–2023 [cit. 2023-04-06]. Dostupné z: https://developer.mozilla.org/en-US/docs/Learn/Server-side/First_steps/Introduction

[2] MDN Web Docs: HTML: HyperText Markup Language [online]. San Francisco: Mozilla Corporation, ©1998–2022 [cit. 2022-08-19]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/HTML

[3] MDN Web Docs: CSS: Cascading Style Sheets [online]. San Francisco: Mozilla Corporation, ©1998–2022 [cit. 2022-08-21]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/CSS

[4] MDN Web Docs: JavaScript [online]. San Francisco: Mozilla Corporation, ©1998–2022 [cit. 2022-08-21]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/JavaScript

[5] Reactjs: Tutorial: Intro to React [online]. Kalifornie: Meta Platforms, ©2022 [cit. 2022-08-21]. Dostupné z: https://reactjs.org/tutorial/tutorial.html

[6] MDN: An overview of HTTP [online]. San Francisco: Mozilla Corporation, ©1998–2022 [cit. 2023-03-18]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview

[7] Microservices [online]. Seattle: Amazon Web Services, © 2023 [cit. 2023-04-13]. Dostupné z: https://aws.amazon.com/microservices/

[8] Documentation [online]. Santa Barbara (kalifornie): PostgreSQL Global Development Group, © 1996-2023 [cit. 2023-04-14]. Dostupné z: https://www.postgresql.org/docs/

[9] Java: Essentials, Part 1, Lesson 1: Compiling Running a Simple Program [online]. Austin (Texas): Oracle Corporation, © 2023 [cit. 2023-03-18]. Dostupné z: https://www.oracle.com/java/technologies/compile.html

[10] Java Interpreter: Essentials, Part 1, Lesson 1: Compiling Running a Simple Program [online]. Noida (Indie): JavaTpoint, © 2011-2021 [cit. 2023-03-18]. Dostupné z: https://www.javatpoint.com/java-interpreter

[11] All Java components Diagram. In: Software Testing Help: Java Components: Java Platform, JDK, JRE, & Java Virtual Machine [online]. Bengalúru (Indie): SOFTWARETESTINGHELP, © 2022, © 2022 [cit. 2023-03-17]. Dostupné z: https://www.softwaretestinghelp.com/java-components-java-platform-jdk/

[12] The JIT compiler [online]. armonk (USA): IBM Corporation, ©1993-2019 [cit. 2023-03-19]. Dostupné z: https://www.ibm.com/docs/en/ztpf/1.1.0.15?topic=reference-jit-compiler

[13] GraalVM: Get Started with GraalVM [online]. Austin (Texas): Oracle Corporation, © 2018-2023 [cit. 2023-03-19]. Dostupné z: https://www.graalvm.org/latest/docs/getting-started/#run-java

[14] GraalVM: Native Image Basics [online]. Austin (Texas): Oracle Corporation, © 2018-2023 [cit. 2023-03-20]. Dostupné z: https://www.graalvm.org/latest/reference-manual/native-image/basics/

[15] What is a Framework? Why We Use Software Frameworks [online]. Dublin: Code Institute, © 2023 [cit. 2023-04-13]. Dostupné z: https://codeinstitute.net/global/blog/what-is-a-framework/

[16] Spring Boot Reference Documentation: Getting Started [online]. Palo Alto California: Spring, © 2012-2023 [cit. 2023-03-21]. Dostupné z: https://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/#getting-started

[17] The IoC container [online]. Palo Alto California: Rod Johnson, © 2004-2016 [cit. 2023-03-27]. Dostupné z: https://docs.spring.io/spring-framework/docs/3.2.x/spring-framework-reference/html/beans.html

[18] Bean [online]. Palo Alto California: Rod Johnson, © 2008 [cit. 2023-03-27]. Dostupné z: https://docs.spring.io/spring-javaconfig/docs/1.0.0.M4/reference/html/ch02s02.html

[19] Web MVC framework [online]. Palo Alto California: Rod Johnson, © 2004-2016 [cit. 2023-03-27]. Dostupné z: https://docs.spring.io/spring-framework/docs/3.2.x/spring-framework-reference/html/mvc.html

[20] Spring Data [online]. Palo Alto California: VMware, © 2023 [cit. 2023-03-28]. Dostupné z: https://spring.io/projects/spring-data#overview

[21] Testing [online]. Palo Alto California: VMware, © 2023 [cit. 2023-03-28]. Dostupné z: https://docs.spring.io/spring-framework/docs/current/reference/html/testing.html

[22] Java Spring Pros and Cons [online]. Noida (Indie): JavaTpoint, © 2011-2021 [cit. 2023-03-28]. Dostupné z: https://www.javatpoint.com/java-spring-pros-and-cons

[23] What is Quarkus? [online]. Raleigh (Severní Karolína): Red Hat, © 2019-2023 [cit. 2023-03-28]. Dostupné z: https://quarkus.io/about/

[24] Kubernetes [online]. San Francisco: Linux Foundation, © 2023 [cit. 2023-04-14]. Dostupné z: https://kubernetes.io/

[25] QUARKUS REACTIVE ARCHITECTURE [online]. Raleigh (Severní Karolína): Red Hat, © 2019-2023 [cit. 2023-03-28]. Dostupné z: https://quarkus.io/guides/quarkus-reactive-architecture

[26] CONTEXTS AND DEPENDENCY INJECTION [online]. Raleigh (Severní Karolína): Red Hat, © 2019-2023 [cit. 2023-03-28]. Dostupné z: https://quarkus.io/guides/cdi-reference

[27] USING THE REST CLIENT [online]. Raleigh (Severní Karolína): Red Hat, © 2019-2023 [cit. 2023-03-28]. Dostupné z: https://quarkus.io/guides/rest-client

[28] QUARKUS SECURITY OVERVIEW [online]. Raleigh (Severní Karolína): Red Hat, © 2019-2023 [cit. 2023-03-28]. Dostupné z: https://quarkus.io/guides/security-overview-concept

[29] DATASOURCES [online]. Raleigh (Severní Karolína): Red Hat, © 2019-2023 [cit. 2023-03-28]. Dostupné z: https://quarkus.io/guides/datasource

[30] SIMPLIFIED HIBERNATE ORM WITH PANACHE [online]. Raleigh (Severní Karolína): Red Hat, © 2019-2023 [cit. 2023-03-28]. Dostupné z: https://quarkus.io/guides/hibernate-orm-panache

[31] TESTING YOUR APPLICATION [online]. Raleigh (Severní Karolína): Red Hat, © 2019-2023 [cit. 2023-03-28]. Dostupné z: https://quarkus.io/guides/getting-started-testing#testhttpresource

[32] Micronaut: 1 Introduction [online]. Saint Louis (Missouri): Micronaut Foundation, © 2022 [cit. 2023-03-29]. Dostupné z: https://docs.micronaut.io/latest/guide/

[33] Micronaut: 3 Inversion of Control [online]. Saint Louis (Missouri): Micronaut Foundation, © 2022 [cit. 2023-03-29]. Dostupné z: https://docs.micronaut.io/latest/guide/#ioc

[34] Micronaut: 6 The HTTP Server [online]. Saint Louis (Missouri): Micronaut Foundation, © 2022 [cit. 2023-03-29]. Dostupné z: https://docs.micronaut.io/latest/guide/#httpServer

[35] Micronaut Security [online]. Saint Louis (Missouri): Micronaut Foundation, © 2022 [cit. 2023-03-30]. Dostupné z: https://micronaut-projects.github.io/micronaut-security/latest/guide/

[36] Micronaut Data [online]. Saint Louis (Missouri): Micronaut Foundation, © 2022 [cit. 2023-03-31]. Dostupné z: https://micronaut-projects.github.io/micronaut-data/latest/guide/

[37] Micronaut Test [online]. Saint Louis (Missouri): Micronaut Foundation, © 2022 [cit. 2023-03-31]. Dostupné z: https://micronaut-projects.github.io/micronaut-test/latest/guide/

[38] PLECINSKI, Piotr, Nataliia BOKLA, Tamara KLYMKOVYCH, Mykhailo MELNYK a Wojciech ZABIEROWSKI. Comparison of Representative Microservices Technologies in Terms of Performance for Use for Projects Based on

Sensor Networks. Sensors [online]. 2022, 22(20) [cit. 2023-04-03]. ISSN 1424-8220. Dostupné z: doi:10.3390/s22207759

[39] WYCIŚLIK, Łukasz, Łukasz LATUSIK a Anna Małgorzata KAMIŃSKA. A Comparative Assessment of JVM Frameworks to Develop Microservices. Applied Sciences [online]. 2023, 13(3) [cit. 2023-04-04]. ISSN 2076-3417. Dostupné z: doi:10.3390/app13031343

[40] Installation on Windows Platforms [online]. Austin (Texas): Oracle Corporation, © 2018-2023 [cit. 2023-04-25]. Dostupné z: https://www.graalvm.org/latest/docs/getting-started/windows/

[41] DOWNEY, Tim. Guide to Web Development with Java [online]. Cham: Springer International Publishing, 2021 [cit. 2022-07-31]. Texts in Computer Science. ISBN 978-3-030-62273-2. Dostupné z: doi:10.1007/978-3-030-62274-9

10 Přílohy

1) Zdrojový kód backendové části aplikace je prostřednictvím GitHub z odkazu: https://github.com/VitRehak/RewaringSystem-Backend

Univerzita Hradec Králové
Fakulta informatiky a managementu

Zadání bakalářské práce

Autor: **Vít Řehák**

Studium: I2000408

Studijní program: B1802 Aplikovaná informatika

Studijní obor: Aplikovaná informatika

Název bakalářské práce: **Porovnání webových frameworků pro jazyk Java**

Název bakalářské práce AJ: Comparison of Web Frameworks for Java Language

Cíl, metody, literatura, předpoklady:

Cíl: Prozkoumat, popsat a porovnat vybrané frameworky na platformě Javy a vytvožit vzorovou aplikaci demonstrující vlastnosti vybraného frameworku.
Osnova:

1. Úvod
2. Webové technologie pro frontend a backend
3. Popis vybraných frameworků
4. Porovnání a výběr frameworku pro tvorbu IS
5. Návrh a implementace vzorové aplikace
6. Shrnutí a závěr

Zadávající pracoviště: Katedra informatiky a kvantitativních metod, Fakulta informatiky a managementu

Vedoucí práce: doc. Mgr. Tomáš Kozel, Ph.D.

Datum zadání závěrečné práce: 26.1.2021