

Evaluace vlivů vývojových frameworků na efektivitu webového vývoje

Ivan Zaitsau

Bakalářská práce
2025



Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
Ústav informatiky a umělé inteligence

Akademický rok: 2024/2025

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení: Ivan Zaitsau
Osobní číslo: A21085
Studijní program: B0613A140020 Softwarové inženýrství
Forma studia: Prezenční
Téma práce: Evaluace vlivů vývojových frameworků na efektivitu webového vývoje
Téma práce anglicky: Evaluation of the Impact of Development Frameworks on the Efficiency of Web Development

Zásady pro vypracování

- Vypracovat literární studii o aktuálním stavu a nejnovějších aktualizacích v rámci technologií Java, HTML, CSS, JS.
- Analyzujte aktuálně dostupné a používané vývojové platformy pro vytváření webových aplikací pomocí Javy.
- Implementujte svůj vlastní prototyp webové aplikace ve dvou variantách-bez použití rámců a s jejich zapojením do procesu vývoje. Obě verze aplikace by měly mít stejnou funkčnost.
- Porovnejte výkonové ukazatele měřené na základě funkčnosti obou implementovaných verzí aplikace. Prezentujte výsledky odpovídajícím způsobem.
- Zhodnoťte proces vývoje, výsledky provedených měření a formulujte svá vlastní doporučení s ohledem na získané výsledky.

Forma zpracování bakalářské práce: **tištěná/elektronická**

Seznam doporučené literatury:

1. WALLS, Craig a BREIDENBACH, Ryan. Spring in action. 2nd ed. Greenwich: Manning, c2008. ISBN 978-1-933-98813-9.
2. SESHADRI, Shyam. Angular Up & Running: Learning Angular Step by Step. 2rd edition. Sebastopol, CA: O'Reilly Media, 2018. ISBN 978-1-491-99983-7.
3. HAVERBEKE, Marijn. Eloquent JavaScript: A Modern Introduction to Programming. 3rd edition. San Francisco, CA: No Starch Press, 2018. ISBN 978-1-593-27950-9
4. STEFANOV, Stoyan. JavaScript patterns. Sebastopol: O'Reilly, 2010. ISBN 978-0-596-80675-0.

Vedoucí bakalářské práce: **Ing. Jozef Kováč**
Ústav informatiky a umělé inteligence

Datum zadání bakalářské práce: **27. října 2024**
Termín odevzdání bakalářské práce: **2. června 2025**



L.S.

doc. Ing. Jiří Vojtěšek, Ph.D. v.r.
děkan

prof. Mgr. Roman Jašek, Ph.D., DBA v.r.
ředitel ústavu

Ve Zlíně dne 17. listopadu 2024

PROHLÁŠENÍ AUTORA BAKALÁŘSKÉ PRÁCE

Beru na vědomí, že

- odevzdáním bakalářské práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb., v platném znění bez ohledu na výsledek obhajoby;
- bakalářská práce bude uložena v elektronické podobě v univerzitním informačním systému a bude dostupná k nahlédnutí;
- jedno vyhotovení bakalářské práce v listinné podobě bude ponecháno Univerzitě Tomáše Bati ve Zlíně k uložení;
- na moji bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
- podle § 60 odst. 1 autorského zákona má Univerzita Tomáše Bati ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- podle § 60 odst. 2 a 3 mohu užít své dílo – bakalářskou práci – nebo poskytnout licenci k jejímu využití jen s předchozím písemným souhlasem Univerzity Tomáše Bati ve Zlíně, která je oprávněna v takovém případě ode mne požadovat přiměřený příspěvek na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše);
- pokud bylo k vypracování bakalářské práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tj. k nekomerčnímu využití), nelze výsledky bakalářské práce využít ke komerčním účelům;
- pokud je výstupem bakalářské práce jakýkoliv softwarový produkt, považují se za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se projekt skládá; neodevzdání této součásti může být důvodem k neobhájení práce.

Prohlašuji, že

- jsem na bakalářské práci pracoval(a) samostatně a použitou literaturu jsem řádně citoval(a); v případě publikace výsledků budu uveden(a) jako spoluautor;
- odevzdaná verze bakalářské práce a verze elektronická nahraná do IS/STAG jsou obsahově totožné.

Ve Zlíně, dne

.....
podpis autora

ABSTRAKT

Tato práce se zabývá evaluací vlivů moderních vývojových frameworků na efektivitu webového vývoje. Cílem práce je zhodnotit, jak integrace frameworků ovlivňuje klíčové aspekty tvorby webových aplikací, jako jsou rychlost vývoje, výkon a údržba kódu. V teoretické části je proveden literární přehled aktuálního stavu technologií Java, HTML, CSS a JavaScript a analyzovány jejich trendy a inovace. Praktická část zahrnuje implementaci dvou variant webové aplikace – první verze je vytvořena bez využití frameworků a druhá s jejich zapojením. Obě varianty jsou porovnány na základě metrik, jako je doba odezvy, spotřeba systémových zdrojů a vývojová efektivita. Na základě získaných výsledků práce hodnotí přínosy a omezení používání frameworků a formulují doporučení pro optimalizaci procesu webového vývoje.

Klíčová slova: frameworky, CSS, JavaScript (programovací jazyk), analýza, software, web, HTML (značkovací jazyk).

ABSTRACT

This paper deals with the evaluation of the effects of modern development frameworks on the effectiveness of web development. The aim of the work is to evaluate how the integration of frameworks affects key aspects of web application development such as development speed, performance and code maintenance. In the theoretical part, a literature review of the current status of Java, HTML, CSS and JavaScript technologies is conducted and their trends and innovations are analyzed. The practical part includes the implementation of two variants of the web application - the first version is created without the use of frameworks and the second one with their involvement. Both variants are compared based on metrics such as response time, system resource consumption and development efficiency. Based on the results obtained, the paper evaluates the benefits and limitations of using frameworks and formulates recommendations for optimizing the web development process.

Keywords: software frameworks, cascading style sheets, JavaScript (programming language), analysis, software, World Wide Web, HTML (document markup language).

Rád bych poděkoval vedoucímu mé bakalářské práce, Ing. Jozefu Kováčovi, za jeho odborné vedení, konstruktivní připomínky, neúnavnou ochotu pomoci a čas, který mi během tvorby bakalářské práce věnoval. Zároveň děkuji své rodině a blízkým přátelům za jejich neustálou podporu a víru ve mě, která mi pomáhala překonávat všechny nástrahy studia.

OBSAH

ÚVOD.....	13
I TEORETICKÁ ČÁST	15
1 ZÁKLADY WEBOVÉHO VÝVOJE	16
1.1 ÚVOD DO WEBOVÉHO VÝVOJE	16
1.2 KLÍČOVÉ TECHNOLOGIE WEBOVÉHO VÝVOJE	17
1.2.1 HTML, CSS a JavaScript: vlastnosti, možnosti a omezení	18
1.2.2 Java jako serverová technologie: možnosti a využití ve webovém vývoji.....	20
2 FRAMEWORKY WEBOVÉHO VÝVOJE.....	23
2.1 POJEM A KLASIFIKACE FRAMEWORKŮ	23
2.1.1 Historie frameworků ve webovém vývoji.....	25
2.1.2 Kritéria výběru frameworku pro projekt	26
2.1.3 Trendy ve vývoji frameworků.....	27
2.2 FRAMEWORKY PRO SERVEROVOU ČÁST S VYUŽITÍM JAVY	29
2.2.1 Přehled Java technologií – od Java Servlets a JSP po moderní Java EE / Jakarta	31
2.2.2 Hlavní Java frameworky pro vývoj serverových aplikací.....	33
2.2.3 Srovnání Java frameworků – kritéria výkonnosti, bezpečnosti a snadnosti údržby.....	37
2.2.4 Best practices a architektonické přístupy při využití Java frameworků.....	40
2.2.5 Nástroje pro správu a orchestraci aplikací v Java prostředí.....	43
2.3 FRAMEWORKY PRO KLIENTSKOU ČÁST	45
2.3.1 Přehled a srovnání populárních řešení (Angular, React, Vue).....	46
2.3.2 Rozšířená řešení a novinky (Svelte, Ember.js)	48
2.3.3 Nástroje pro optimalizaci a testování klientských frameworků	50
3 EFEKTIVITA WEBOVÉHO VÝVOJE: KRITÉRIA A METODY HODNOCENÍ.....	53
3.1 DEFINICE EFEKTIVITY VE WEBOVÉM VÝVOJI	53
3.2 HLAVNÍ METRIKY A KRITÉRIA HODNOCENÍ.....	54
3.3 VLIV VYUŽITÍ FRAMEWORKŮ NA KVALITU A VÝKONNOST WEBOVÝCH APLIKACÍ	55
4 SROVNÁVACÍ ANALÝZA VÝVOJE S VYUŽITÍM FRAMEWORKŮ A BEZ NICH.....	58
4.1 METODOLOGIE SROVNÁVACÍ ANALÝZY	58
4.1.1 Výběr metrik a rozsah studie.....	58
4.1.2 Nástroje pro měření a pozorování	59
4.1.3 Experimentální design: vývoj dvou verzí aplikace	59
4.2 VÝHODY A NEVÝHODY VÝVOJE S FRAMEWORKY	60
4.2.1 Výhody	60

4.2.2	Nevýhody	61
4.3	PRAKTICKÉ PŘÍKLADY A PŘÍPADOVÉ STUDIE Z REÁLNÉHO PROSTŘEDÍ.....	62
4.3.1	Případová studie: Migrace na architekturu mikroslužeb pomocí Spring Boot	62
4.3.2	Případová studie: „vanilla JS“ vs. React frontend development	63
4.3.3	Ukázka, jak začlenit Vue do již existujícího projektu	63
II	PRAKTICKÁ ČÁST	65
5	UVOD DO PRAKTICKE CASTI.....	66
6	ARCHITEKTURA A TECHNOLOGIE.....	67
6.1	ARCHITEKTURA BEZ RÁMCE	67
6.2	ARCHITEKTURA SPRING BOOT + ANGULAR	70
6.3	SROVNÁNÍ ARCHITEKTUR	73
7	DATOVÝ MODEL A PŘÍSTUP K DATŮM.....	75
7.1	PODROBNOSTI O MODELU DAT A PŘÍSTUPU VE VARIANTĚ BEZ RÁMCE	75
7.2	PODROBNOSTI O MODELU DAT A PŘÍSTUPU VE VARIANTĚ SPRING BOOT + ANGULAR	81
7.3	VÝSLEDKY	86
8	TOKY POŽADAVKŮ A ZABEZPEČENÍ.....	88
8.1	TOK POŽADAVKŮ A ZABEZPEČENÍ V APLIKACI MINIBLOG BEZ FRAMEWORKU.....	88
8.2	TOK POŽADAVKŮ A ZABEZPEČENÍ V APLIKACI MINIBLOG SPRING BOOT + ANGULAR	90
8.3	VÝSLEDKY	92
9	FRONTEND.....	93
9.1	FRONTEND JSP	93
9.2	FRONT-END ANGULAR.....	94
9.3	VÝSLEDKY	97
10	FUNKČNÍ PREZENTACE MINIBLOGU	99
10.1	PŘIHLAŠOVACÍ STRÁNKA.....	99
10.2	REGISTRAČNÍ STRÁNKA	100
10.3	HLAVNÍ STRÁNKA.....	101
10.4	STRÁNKA UŽIVATELE	103
10.5	PŘIDÁNÍ PŘÍSPĚVKU.....	104
10.6	UPRAVA PŘÍSPĚVKŮ	105
10.7	ZMĚNA BAREVNÉHO SCHÉMATU	106
11	NÁKLADY NA MONTÁŽ, NASAZENÍ A PRÁCI.....	108
11.1	PRACOVNÍ NÁKLADY MINIBLOGU (BEZ FRAMEWORKU).....	108

11.1.1	Pozorování.....	110
11.1.2	Výsledky	110
11.2	PRACOVNÍ NÁKLADY MINIBLOGU (S FRAMEWORKEM)	110
11.2.1	Pozorování.....	113
11.2.2	Výsledky	113
12	ZÁTĚŽOVÉ TESTOVÁNÍ	115
12.1	MINIBLOG BEZ FRAMEWORKU	115
12.2	MINIBLOG SPRING BOOT	116
12.3	VÝSLEDKY	116
	ZÁVĚR	118
	SEZNAM POUŽITÉ LITERATURY.....	119
	SEZNAM OBRÁZKŮ	122
	SEZNAM TABULEK.....	123
	SEZNAM UKÁZEK KÓDU	124
	SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK	125
	SEZNAM PŘÍLOH.....	126
	PŘÍLOHA P I: CD-ROM.....	127

Úvod

V posledních letech se vývoj webových stránek stal nedílnou součástí moderní informační infrastruktury. S rostoucí složitostí webových aplikací a rostoucími požadavky na jejich výkon, bezpečnost a škálovatelnost musí vývojáři používat stále efektivnější nástroje a technologie. Jedním z takových nástrojů jsou moderní vývojové rámce, které zjednodušují proces tvorby, testování a údržby aplikací a umožňují výrazně snížit časové i zdrojové náklady.

I přes zjevné výhody používání frameworků zůstává aktuální otázka jejich vlivu na celkovou efektivitu vývoje. S rychlým rozvojem technologií a nástupem nových nástrojů je zásadní provést systematické hodnocení toho, jak používání frameworků ovlivňuje klíčové aspekty vývoje webových aplikací, jako je rychlost implementace, výkonnost výsledného produktu a snadnost údržby kódu.

Cílem tohoto článku je vyhodnotit vliv moderních vývojových rámců na efektivitu vývoje webových aplikací. K dosažení tohoto cíle jsou stanoveny následující cíle:

- Analyzovat a přezkoumat současný stav technologií Java, HTML, CSS a JavaScript a moderních frameworků používaných při vývoji webových stránek.
- Prozkoumat teoretické aspekty používání frameworků, identifikovat jejich výhody, omezení a dopad na proces vývoje webových aplikací.
- Vyvinout dvě verze webové aplikace s identickou funkcí: jednu verzi bez použití frameworků a druhou verzi s jejich použitím.
- Provést srovnávací analýzu výsledných prototypů na základě takových ukazatelů, jako je doba odezvy, spotřeba systémových prostředků a celková efektivita procesu vývoje.
- Formulovat doporučení, jak optimalizovat vývoj webových stránek s ohledem na získané výsledky.

K realizaci stanovených úkolů práce využívá komplexní přístup zahrnující teoretický výzkum, praktickou implementaci prototypů a experimentální porovnání jejich výkonnosti. Praktická část práce zahrnuje vývoj dvou verzí webové aplikace, což umožňuje objektivně posoudit vliv frameworků na různé aspekty vývoje.

Struktura práce je uspořádána následovně: v úvodu je zdůvodněn význam výzkumu a formulován jeho cíl a úkoly; v přehledu literatury jsou diskutovány současné trendy ve vývoji webových aplikací a používání frameworků; v metodologické části jsou popsány

použité metody výzkumu; praktická část obsahuje popis vývoje a výsledky srovnávací analýzy prototypů; závěrečná část obsahuje diskusi výsledků, závěry a doporučení pro další výzkum.

Předkládaný výzkum si tedy klade za cíl komplexně zhodnotit vliv moderních frameworků na efektivitu vývoje webových aplikací a poskytuje praktická doporučení pro optimalizaci procesu vývoje webových aplikací v kontextu rychle se měnících technologií

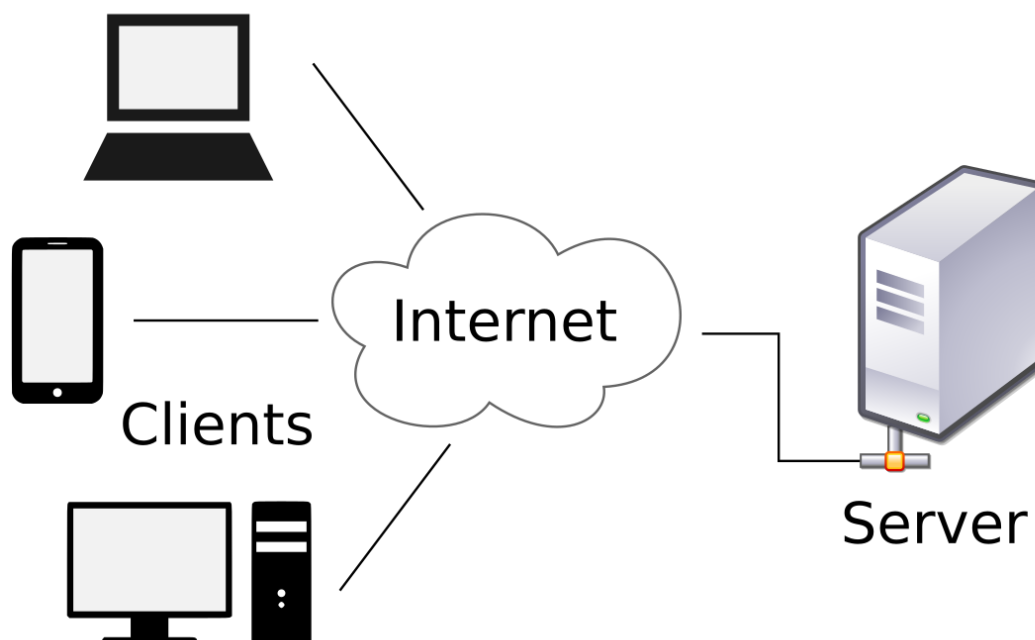
I TEORETICKÁ ČÁST

1 ZÁKLADY WEBOVÉHO VÝVOJE

Historický a technologický vývoj webu od jeho vzniku až po současnost je hlavním tématem této kapitoly. Nejprve si nastíníme základní součásti webu, včetně jazyka HTML, kaskádových stylů CSS a jazyka JavaScript na straně klienta, a jejich vzájemnou spolupráci. Dále probereme základy architektury webových aplikací, včetně funkcí prohlížeče, webového serveru a aplikační vrstvy, a také komunikační paradigma klient-server. Cílem je nabídnout pevný teoretický základ pro pochopení každé z následujících kapitol.

1.1 Úvod do webového vývoje

Od samého počátku World Wide Webu se vývojáři potýkají s neustálými problémy při vytváření webových aplikací, které jsou efektivní a uživatelsky přívětivé. Webové stránky začínaly jako nekomplikované statické dokumenty HTML, které nabízely pouze základní informace a postrádaly interaktivitu. S nástupem technologií CSS a JavaScript bylo možné vyvíjet dynamická a adaptivní rozhraní, čímž se otevřely nové možnosti interakce uživatelů s webovými systémy [1].



Obrázek 1 - Clients communicating with a server via the Internet [27]

Integrace klientských a serverových technologií definuje základní principy moderního vývoje webových stránek. Klientská komponenta se stará o vizuální reprezentaci a

interaktivitu, zatímco serverová komponenta řídí zpracování dat, zabezpečení a škálovatelnost aplikace. Toto oddělení umožňuje vytvářet systémy, které dokáží vyhovět rostoucím nárokům na rychlost a funkčnost a zároveň zajistit kvalitní uživatelský zážitek [2].

Vytváření webových aplikací výrazně usnadňují současné frameworky a knihovny jako Angular, React a Vue.js. Dosahují toho díky modulárnímu návrhu, opakovanému použití kódu a automatizaci opakujících se úloh. Díky těmto nástrojům se mohou vývojáři soustředit na implementaci obchodní logiky, snížit časové a zdrojové výdaje a zvýšit udržovatelnost a kvalitu výsledného produktu [3].

Při vývoji na straně serveru zajišťují technologie jako Node.js, PHP, Python a Java spolehlivé zpracování dotazů a interakci s databází - což je zásadní součást vytváření robustních vícevrstevných webových aplikací. Klientská a serverová řešení lze integrovat s cílem dosáhnout vysokého výkonu a škálovatelnosti moderních systémů [2].

Dodržování standardů pro vývoj webových aplikací, jako jsou HTML5, CSS3 a ECMAScript, navíc zaručuje kompatibilitu napříč různými platformami, zařízeními a prohlížeči. To zlepšuje použitelnost a bezpečnost webových systémů, což je v rychle se měnícím digitálním prostředí obzvlášť důležité [1].

1.2 KLÍČOVÉ TECHNOLOGIE WEBOVÉHO VÝVOJE

Moderní vývoj webových aplikací je založen na souboru technologií, které umožňují vytvářet funkční, interaktivní a škálovatelné webové aplikace. Tento komplex zahrnuje nástroje a jazyky, které zajišťují práci na straně klienta i serveru, a také řešení pro ukládání a zpracování dat.

Na straně klienta jsou základními technologiemi např:

HTML je značkovací jazyk, který vytváří strukturu webové stránky;

CSS je technologie pro tvorbu stylů a adaptivní design;

Programovací jazyk JavaScript má na starosti dynamiku a interaktivitu uživatelského rozhraní.

Zpracování dotazů, správa dat a obchodní logika aplikace jsou implementovány na straně serveru. Robustní infrastruktura pro zpracování informací je vytvořena pomocí různých jazyků a platform, včetně jazyků PHP, Python, Java a Node.js.JS.

Stejně zásadní jsou i systémy správy databází, které usnadňují přístup k datům a jejich ukládání. V tomto kontextu se využívají jak relační databáze (jako MySQL a PostgreSQL), tak nerelační řešení (jako MongoDB).

Současné frameworky a knihovny (například Express.js, Django a Spring na straně serveru a React, Angular a Vue.js na straně klienta) navíc výrazně urychlují proces vývoje. Poskytují škálovatelnost, modularitu a snazší údržbu kódu - což je pro vytváření rozsáhlých online aplikací klíčové [1][2][3].

Klíčové technologie pro vývoj webových aplikací tak představují vzájemně propojený soubor nástrojů potřebných k vytváření moderních webových aplikací, které splňují vysoké požadavky na výkon, použitelnost a bezpečnost.

1.2.1 HTML, CSS a JavaScript: vlastnosti, možnosti a omezení

HTML:

The fundamental markup language used to create web pages is called HTML (HyperText Markup Language). It enables you to construct headers, paragraphs, lists, tables, forms, and other features while defining the format and content of a document.

- **Možnosti:**
 - Generování sémantické struktury webové stránky.
 - Podpora různých typů médií (obrázky, video, zvuk).
 - Integrace s dalšími technologiemi (např. CSS a JavaScript).
- **Omezení:**
 - Nedostatek prostředků pro implementaci dynamického chování - HTML nemůže samostatně měnit obsah stránky po jejím načtení.

- Závislost prohlížeče při interpretaci sémantického značení, která může ovlivnit zobrazení v různých prostředích [1].

CSS:

CSS (Cascading Style Sheets) je zodpovědný za design a vizuální prezentaci webových stránek. Umožňuje oddělit obsah od designu a poskytuje flexibilitu při změně vzhledu bez změny kódu HTML.

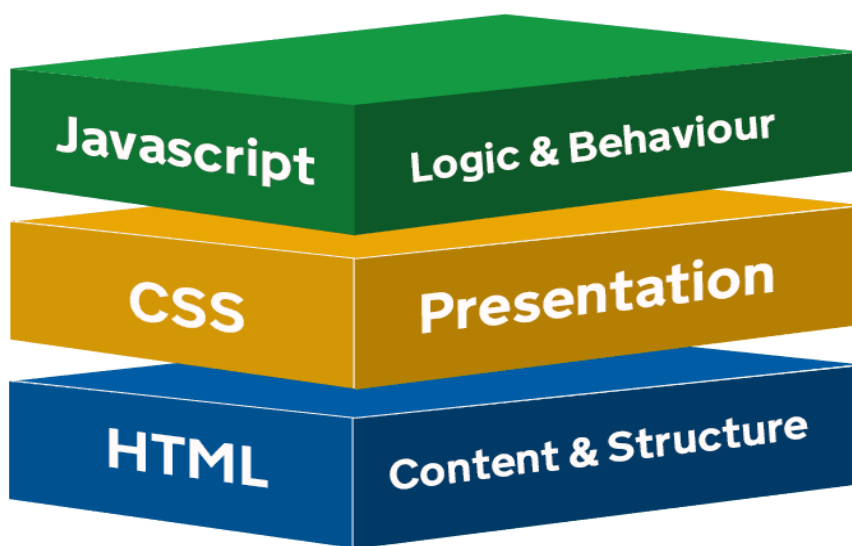
- Možnosti:
 - Nastavení stylů pro různé prvky: barvy, písma, odrážky atd.
 - Vytvoření responzivního designu pomocí dotazů na média, díky čemuž se web snadno.
 - Podpora animací a přechodů pro zvýšení vizuálního zážitku
- Omezení:
 - Omezená možnost implementace složité logiky interakce s uživatelem, která vyžaduje použití jazyka JavaScript.
- Potíže se správou rozsáhlých souborů stylů, zejména při práci na velkých projektech, bez použití preprocesorů nebo modulárního přístupu [2].

JavaScript:

Umožňuje měnit obsah stránky za běhu, zpracovávat události uživatele a komunikovat se serverem prostřednictvím požadavků AJAX.

- Možnosti:
 - JavaScript se vyznačuje různými funkcemi, které jsou v něm obsaženy, např: Dynamická aktualizace obsahu a interakce s uživatelem.
 - Manipulace s DOM pro změnu struktury a stylů stránky v reálném čase.
 - Práce s externími rozhraními API a implementace složité obchodní logiky na straně klienta.
- Omezení:

- Možné problémy s výkonem při operacích náročných na prostředky, zejména ve starších prohlížečích.
- Absence striktního typování (v nejčistší podobě), což může vést k chybám, pokud se nepoužívají moderní nástroje (např. TypeScript).
- Bezpečnostní rizika při nesprávném zpracování uživatelských vstupů a dat [3]



Obrázek 2 - HTML, CSS, JS layered [28]

Bezpečnostní rizika při nesprávném zpracování uživatelských vstupů a dat [3].

V současné době je možné, že se na základě těchto informací a zkušeností podařilo vytvořit tzv:

Integrace jazyků HTML, CSS a JavaScript umožňuje vytvářet moderní, dynamické a adaptivní webové aplikace. Každá technologie má své jedinečné schopnosti, které při správné kombinaci poskytují vysoce kvalitní webový produkt. Pochopení jejich omezení je však klíčem k výběru nejlepších nástrojů a přístupů pro vývoj [2][3].

1.2.2 Java jako serverová technologie: možnosti a využití ve webovém vývoji

Pokud jde o vývoj webových stránek na straně serveru, je Java jedním z nejpoužívanějších programovacích jazyků. Rozsáhlá knihovna jazyka Java, jeho rychlý výkon, spolehlivost a

svoboda platformy dávají vývojářům možnost vytvářet bezpečné a škálovatelné aplikace na straně serveru [5][10].

Možnosti Javy jako serverové technologie:

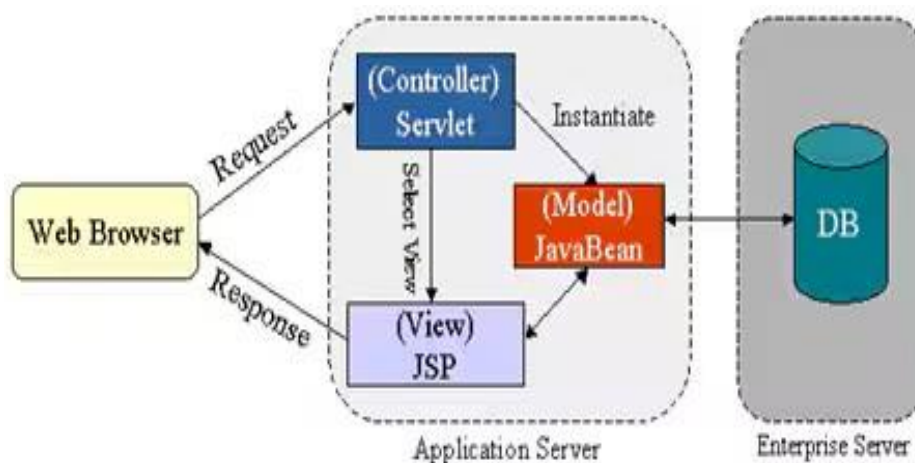
- Nezávislost na platformě:
 - Díky principu „Write Once, Run Anywhere“ mohou aplikace napsané v jazyce Java běžet na různých operačních systémech, aniž by vyžadovaly výrazné změny kódu. To poskytuje flexibilitu při nasazování serverových řešení [5].
- Vysoký výkon a škálovatelnost:
 - Moderní virtuální stroje Java (JVM) jsou optimalizovány pro efektivní správu paměti a zdrojů, aby zvládly velký počet souběžných požadavků. Díky paralelnímu provádění procesů a využívání vícejádrových systémů je Java optimální volbou pro vysoce zatížené webové služby [5].
- Spolehlivost a bezpečnost:
 - Java obsahuje vestavěné bezpečnostní mechanismy, jako je automatický sběr odpadu, přísné typování a správa paměti, což snižuje pravděpodobnost výskytu chyb a zranitelností v kódu. To je důležité zejména u aplikací na straně serveru, kde jsou stabilita a bezpečnost dat kritické [6].
- Široká nabídka frameworků a knihoven:
 - Ekosystém Javy zahrnuje mnoho nástrojů pro vývoj webových aplikací. Například technologie Java Servlets a JavaServer Pages (JSP) se tradičně používají k vytváření dynamických webových stránek a moderní frameworky, jako je Spring Boot, výrazně zjednodušují vývoj, testování a nasazení aplikací na straně serveru [6].

Použití jazyka Java při vývoji webových stránek Java se aktivně používá v různých oblastech vývoje webových aplikací:

- Vývoj rozhraní API RESTful:
 - Pomocí jazyka Java, zejména v kombinaci s frameworkem Spring Boot, je možné vytvářet efektivní rozhraní API na straně serveru, která umožňují komunikaci mezi klientskými aplikacemi a prostředky serveru. Taková

rozhraní API umožňují flexibilní integraci různých služeb a komponent do soudržného celku[8][9].

- Zpracování obchodní logiky:
 - Java poskytuje robustní platformu pro implementaci složité obchodní logiky, která vyžaduje náročné zpracování dat a vysoký výkon. Díky tomu je preferována pro podniková řešení a rozsáhlé informační systémy[10][12].
- Integrace databází:
 - Java umožňuje snadnou interakci s relačními (MySQL, PostgreSQL) i nerelačními databázemi a poskytuje efektivní ukládání a zpracování dat[7][8].



Obrázek 3 - JSP Architektura [35]

2 FRAMEWORKY WEBOVÉHO VÝVOJE

V této kapitole bude definován pojem „framework“ a představeny jeho základní kategorie, od komplexních aplikačních platforem až po lehké knihovny. Stručně projdeme historii frameworků, zdůrazníme významné zlomové okamžiky (například CGI, ASP, Struts, JSF a Spring) a nastíníme standardy, které vývojáři používají při výběru nejlepšího frameworku pro daný projekt. V závěru kapitoly se budeme zabývat současným vývojem, který má velký vliv na tvorbu frameworků nové generace, jako jsou mikrofrontendy, serverless a headless architektura.

2.1 Pojem a klasifikace frameworků

Softwarová platforma známá jako framework poskytuje vývojářům přístup ke sbírce předem připravených nástrojů, knihoven a architektonických vzorů, které urychlují proces vytváření aplikací. Při vývoji webových aplikací umožňují rámce vývojářům soustředit se na provádění jedinečné obchodní logiky projektu tím, že standardizují organizaci kódu, automatizují opakující se činnosti a usnadňují interakci se současnou technologií [8][7].

Hlavní účel frameworků je následující:

Urychlení vývoje: hotové moduly a řešení mohou výrazně zkrátit dobu potřebnou k vytvoření funkčních komponent aplikace [9].

Standardizace kódu: implementace architektonických principů a návrhových vzorů podporuje udržovatelnost a konzistenci kódu, což podporuje výzkum v oblasti podnikového programování [6].

Zajištění bezpečnosti a škálovatelnosti: moderní frameworky často obsahují vestavěné mechanismy pro ochranu dat, správu relací a optimalizaci výkonu, což je důležité zejména u vysoce zatížených systémů [10].

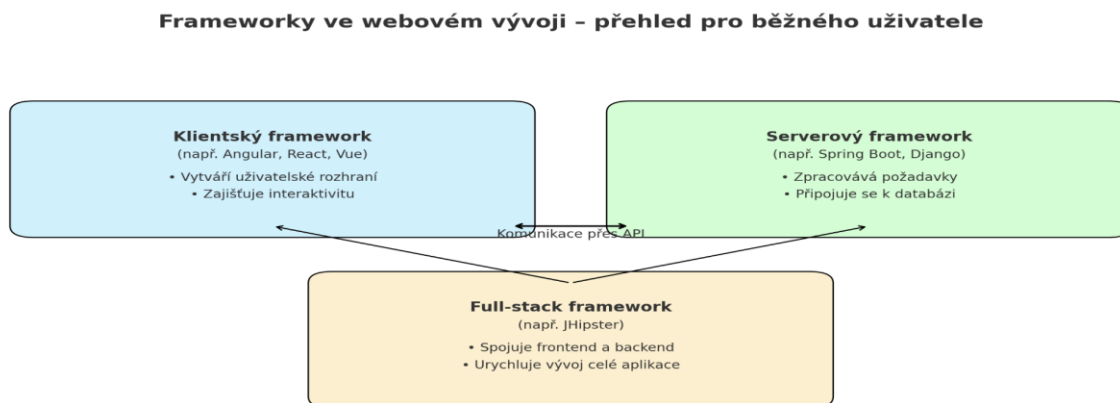
Rámce lze klasifikovat podle několika kritérií:

Podle funkčního účelu:

Frameworky na straně klienta: určené k vytváření dynamických a interaktivních uživatelských rozhraní (např. Angular, React, Vue.js). Tato řešení efektivně spravují stav aplikace a poskytují integraci se stranou serveru prostřednictvím rozhraní API [2].

Rámce na straně serveru: slouží k implementaci logiky na straně serveru, zpracování databází, směřování dotazů a zabezpečení (např. Spring Boot, široce používaný v ekosystému Java) [8].

Full-stack frameworky: kombinují možnosti vývoje na straně klienta a na straně serveru v jedné platformě, jako to dělá JHipster .



Obrázek 4 - Klasifikace frameworků

Podle úrovně abstrakce a architektonických vlastností:

High-level frameworky: poskytují rozsáhlé možnosti automatizace procesů, zjednodušují konfiguraci a integrují různé systémové komponenty [7].

Nízkoúrovňové knihovny a nástroje: zaměřují se na provádění specifických úkolů a často se kombinují s vysokoúrovňovými řešeními, aby splnily jedinečné požadavky projektu [6].

Podle použitých technologií a programovacích jazyků:

Rámce mohou být vyvinuty v různých jazycích, jako jsou JavaScript, Java, Python, Ruby atd. Tato studie se zaměřuje na frameworky implementované pomocí technologií Java [11].

Podle typu licencování a podpory komunity:

Řešení s otevřeným zdrojovým kódem: mají aktivní komunitu vývojářů, což usnadňuje rychlý vývoj a pravidelné aktualizace [9].

Komerční frameworky: často nabízejí další podporu a integrační možnosti, což je důležité pro rozsáhlé firemní projekty [10].

2.1.1 Historie frameworků ve webovém vývoji

Historie frameworků ve vývoji webových aplikací je odrazem neustále se vyvíjejících potřeb v oblasti návrhu aplikací a rychlého vývoje a složitosti webových technologií. V počátcích internetu nabízely dynamický obsah programy na straně serveru, jako je CGI nebo ASP, zatímco k tvorbě webových stránek se používal statický jazyk HTML. S rostoucím počtem uživatelů a funkcí se však stal nezbytným metodičtější přístup k vývoji webových aplikací[1][12].

První frameworky založené na architektonickém vzoru MVC (Model-View-Controller) se objevily počátkem roku 2000, kdy vývojáři začali zavádět myšlenku segmentace programů do logických úrovní. Oddělením obchodní logiky od reprezentace dat tento vzor výrazně zjednodušil vývoj a údržbu kódu. Sofistikovanější řešení jako JavaServer Faces (JSF) a Struts umožnily základní technologie Java Servlets a JavaServer Pages (JSP) ekosystému Java[12][26][30].

Novější rámce nahradily zastaralé alternativy, protože se změnily požadavky na škálovatelnost a přizpůsobivost podnikových aplikací. Spring Framework byl průkopnickým řešením, které od svého uvedení na trh zásadně změnilo metodiku programování v jazyce Java. Spring umožnil vývojářům navrhovat modulární, snadno testovatelné a škálovatelné systémy zavedením konceptů aspektově orientovaného programování (AOP) a inverze řízení (IoC) [8].

Vývoj pokračoval, a to i na straně klienta. Zpočátku dynamiku zajišťoval jednoduchý JavaScript, ale s rostoucí složitostí rozhraní a potřebou interaktivity vznikly specializované frameworky na straně klienta jako Angular, React a Vue.js, které tvorbu moderních webových aplikací výrazně usnadnily [2][3].

Současné trendy ukazují integraci frameworků do jednoho ekosystému s řešeními na straně serveru i klienta a objevují se full-stack frameworky, které kombinují nejlepší postupy z obou oblastí. Přejít k architektuře mikroslužeb, který podrobně popisuje Richardson [9],

je logickým pokračováním tohoto vývoje: umožňuje distribuovat funkčnost do nezávislých služeb, což zlepšuje odolnost proti chybám a usnadňuje škálování aplikací.

Historie frameworků ve vývoji webových aplikací je tedy cestou od jednoduchých skriptů ke složitým modulárním architekturám, které jsou schopny splnit rostoucí požadavky na výkon, bezpečnost a udržitelnost moderních webových aplikací [8][9].

2.1.2 Kritéria výběru frameworku pro projekt

Klíčovou fází, která má velký vliv na celkovou efektivitu vývoje, podporu a rozšiřitelnost aplikace, je výběr vhodného frameworku. Správný framework usnadňuje integraci komponent, udržuje jednotnost kódu a umožňuje rychlé přizpůsobení potřebám projektu, které se v průběhu času mění [12][25]. Mezi hlavní kritéria, která je třeba při výběru frameworku zvážit, patří:

Kompatibilita s požadavky projektu:

Rámec musí splňovat technické specifikace projektu. Pokud například aplikace vyžaduje vysokou škálovatelnost nebo zvýšenou bezpečnost, je třeba zvolit řešení, které tyto aspekty podporuje [7][8].

Podpora a aktivita komunity:

Důležitá je rozsáhlá dokumentace, pravidelné aktualizace a aktivní komunita vývojářů. Open-source frameworky bývají široce podporovány, což usnadňuje řešení vzniklých problémů [7][8].

Stabilita a vyspělost:

Spolehlivý framework by měl mít prokázanou stabilitu a dlouhodobou podporu. Zralá řešení, jako je Spring Framework, byla úspěšně nasazena v komerčních projektech a mají jasný plán vývoje [8][9].

Výkon a škálovatelnost:

Důležitým aspektem je schopnost frameworku efektivně zpracovávat velké množství požadavků, optimalizovat využití systémových zdrojů a podporovat růst aplikace. To je důležité zejména u projektů s vysokou zátěží [8][10].

Bezpečnost:

Framework by měl poskytovat vestavěné bezpečnostní mechanismy na ochranu před běžnými hrozbami, jako jsou SQL injection, XSS a CSRF. Robustní bezpečnostní funkce přispívají k ochraně dat a zvyšují stabilitu aplikace [8][10].

Integrovatelnost a rozšiřitelnost:

Framework by měl být snadno integrovatelný s jinými systémy a knihovnami a měl by umožňovat budoucí rozšiřitelnost funkcí. Klíčovou roli zde hraje podpora standardních rozhraní API a kompatibilita s již používanými nástroji [7].

Náklady a licenční podmínky:

Licenční model a případné náklady na používání frameworku mohou rovněž ovlivnit výběr, zejména u velkých firemních projektů. Open-source řešení mohou často snížit počáteční náklady [7].

Obecně platí, že výběr frameworku by měl vycházet z komplexního posouzení technických, obchodních a strategických požadavků projektu. Správné řešení umožňuje optimalizovat proces vývoje, snížit náklady na údržbu a zajistit dlouhodobou flexibilitu a škálovatelnost aplikace [7][8].

2.1.3 Trendy ve vývoji frameworků

Moderní rámce pro vývoj webových stránek se neustále vyvíjejí, aby odrážely měnící se požadavky trhu a rychlý technologický vývoj. Zde jsou hlavní trendy pozorované v posledních letech, které ovlivňují výběr a používání frameworků:

Mikroservisní architektura a cloudová řešení:

Stále více projektů opouští monolitické aplikace ve prospěch přístupu založeného na mikroslužbách. Tento přístup zahrnuje rozdělení systému na mnoho malých, nezávisle vyvíjených služeb, což zvyšuje flexibilitu a škálovatelnost [9]. Moderní frameworky aktivně podporují mikroslužby a zjednodušují vývoj a nasazení služeb v cloudových prostředích (Docker, Kubernetes atd.) [7].

Rostoucí obliba full-stack frameworků:

Snížení časových nákladů na integraci klientské a serverové části vede stále více projektů k volbě full-stack řešení. Takové frameworky (např. JHipster) kombinují nástroje pro rychlé spuštění aplikace na klíč, včetně předkonfigurovaných konfigurací, generátorů kódu a hotových šablon [9].

Zaměření na bezpečnost a data:

S rostoucím významem ochrany dat a soukromí věnují frameworky zvláštní pozornost vestavěným bezpečnostním funkcím (ochrana proti SQL injection, XSS, CSRF atd.) [10]. Zvyšuje se také podpora autentizačních a autorizačních nástrojů (OAuth2, JWT), což usnadňuje vytváření bezpečných webových aplikací a služeb [29].

Integrace s nástroji DevOps:

Automatizace procesů sestavování, testování a nasazování se stala pro moderní projekty klíčovou. V tomto ohledu mnoho frameworků nabízí pluginy a integrace pro systémy CI/CD, které zjednodušují implementaci kontinuální integrace a kontinuálního nasazení (Jenkins, GitLab CI, GitHub Actions atd.) [7].

Aplikace reaktivních paradigmat:

Rostoucí počet aplikací, které pracují s velkým proudem asynchronních událostí (např. chaty, online hry, streamování), stimuluje vznik frameworků a knihoven zaměřených na reaktivní programování. Spring WebFlux a další reaktivní řešení si poradí s vysokou zátěží při zachování konzistentní doby odezvy [8].

Důraz na výkon a minimalismus:

Stále populárnější je myšlenka „mikrorámců“ a odlehčených řešení, která poskytují pouze nejnutnější funkce a ponechávají vývojáři možnost výběru dalších modulů (např. Micronaut, Quarkus). Tento přístup snižuje náklady na zdroje a zjednodušuje strukturu aplikace [7].

Současné trendy ve vývoji frameworků tak odrážejí posun k flexibilnějším, bezpečnějším a škálovatelnějším řešením, která maximálně podporují rychlý vývoj a nasazení aplikací. Pochopení těchto trendů pomáhá při informovaném výběru technologií a nástrojů, které odpovídají potřebám konkrétního projektu [8][9].

2.2 Frameworky pro serverovou část s využitím Javy

Java je tradičně považována za jeden z nejoblíbenějších jazyků pro tvorbu webových aplikací na straně serveru díky své spolehlivosti, škálovatelnosti a rozsáhlému ekosystému nástrojů a knihoven [5]. Přejít od základních technologií (Java Servlets, JSP) k pokročilejším řešením (Spring, JavaServer Faces atd.) umožnil zjednodušit proces vývoje, systematizovat kód a poskytovat vysoce výkonné aplikace.

Jednou z klíčových výhod Javy na straně serveru je dostupnost široké škály frameworků vytvořených pro řešení různých úloh a pokrývajících všechny aspekty vývoje webu - od správy databází až po zabezpečení[8][9]. Níže uvádíme některá z nejoblíbenějších řešení:

Spring Framework a Spring Boot

Spring Framework je jednou z nejrozšířenějších platform pro podnikový vývoj v jazyce Java. Nabízí mechanismy inverze řízení (IoC), vstřikování závislostí (DI) a aspektově orientovaného programování (AOP), které zjednodušují tvorbu modulárních a snadno testovatelných aplikací [7][8].

Spring Boot doplňuje Spring o zjednodušenou konfiguraci a vestavěný webový server, což umožňuje rychlé zahájení vývoje a také snadnější integraci s architekturami mikroslužeb [9]. Díky těmto vlastnostem je Spring stack jedním z nejvyhledávanějších řešení na trhu.

JavaServer Faces (JSF)

JSF se zaměřuje na komponentový přístup k vytváření uživatelských rozhraní a je úzce integrován se stackem Java EE (nyní Jakarta EE). Obsahuje sadu hotových komponent uživatelského rozhraní, navigačních a validačních mechanismů, které zjednodušují implementaci webových aplikací ve stylu Model-View-Controller [5]. Při vývoji moderních, vysoce zatížených systémů však může být JSF ve srovnání se Spring Boot horší ve flexibilitě a rychlosti nasazení.

Vaadin

Vaadin poskytuje možnost vytvářet webová rozhraní v jazyce Java bez nutnosti psát samostatně kód v jazyce JavaScript. Logika uživatelského rozhraní (UI) a strana serveru

jsou úzce propojeny, což zjednodušuje vývoj v malých až středně velkých projektech [7]. Složité klientské scénáře však mohou vyžadovat další přizpůsobení nebo integraci s jinými technologiemi.

Jakarta EE (dříve Java EE)

Jakarta EE je sada specifikací pro vývoj podnikových aplikací, včetně transakcí, zabezpečení a databázových operací. S aktualizací a přejmenováním Java EE na Jakarta EE získala komunita větší volnost při vývoji platformy, což mělo pozitivní dopad na podporu mikroservisních architektur a cloudových řešení [9]. Klasický model však často vyžaduje složitější konfiguraci ve srovnání se Spring Boot [8].

Mikroservisní frameworky

S rostoucím zájmem o mikroservisní architektury se objevila řešení zaměřená na budování odlehčených služeb (např. Micronaut nebo Quarkus). Jejich cílem je omezit „boilerplate“ (šablonový kód) a zlepšit dobu spouštění služeb, což je důležité zejména při častém nasazování [9][10].

Výhody a vlastnosti frameworků Java na straně serveru

Škálovatelnost a spolehlivost:

Java je původně navržena pro vysoce zatížené systémy a pokročilá optimalizace virtuálního stroje (JVM) umožňuje efektivní využití zdrojů [10].

Integrace s moderními technologiemi:

Většina frameworků Java dobře spolupracuje s nástroji Docker a Kubernetes, což zjednodušuje kontejnerizaci a orchestraci v prostředí mikroslužeb [9].

Podpora zabezpečení:

Nástroje pro ověřování a autorizaci (např. Spring Security), vestavěné mechanismy pro omezování zranitelností (CSRF, SQL injection) a podpora certifikátů a šifrování činí frameworky Java atraktivními pro podnikový sektor [8][10].

Široký ekosystém a komunita:

Vzhledem k dlouhé historii platformy Java se vývojáři mohou spolehnout na rozsáhlou znalostní základnu, kvalitní dokumentaci a aktivní komunitu, která usnadňuje učení a řešení problémů [6].

Výběr frameworku pro serverovou stranu webové aplikace v Javě tedy závisí na požadavcích na výkon, škálovatelnost, snadnost vývoje a týmovou podporu. Spring Boot zůstává lídrem v oblasti mikroslužeb a vysoce zatížených systémů, Jakarta EE (Java EE) je stále žádaná v podnikových řešeních a Vaadin a JSF mohou usnadnit proces tvorby uživatelských rozhraní v klasičtějším stylu [9][10]. Každý z těchto frameworků poskytuje vysokou úroveň spolehlivosti a bezpečnosti, díky čemuž je Java jednou z nejoblíbenějších platform pro vývoj webových aplikací na straně serveru [6].

2.2.1 Přehled Java technologií – od Java Servlets a JSP po moderní Java EE / Jakarta

Vývoj serverových technologií založených na Javě ušel dlouhou cestu: od prvních implementací servletů a JSP až po moderní standardy Java EE (nyní Jakarta EE), které umožňují vytvářet vysoce zatížené a spolehlivé podnikové aplikace [5][11]. Následuje stručný přehled klíčových technologií, které tvoří základ pro vývoj na straně serveru v jazyce Java.

Java Servlets a JSP

Java Servlets:

Servlety byly jedním z prvních kroků ve vývoji Javy pro vývoj webových aplikací. Jsou to třídy jazyka Java, které zpracovávají požadavky HTTP a generují odpovědi. Servlety běží uvnitř kontejneru servletů (např. Apache Tomcat nebo Jetty), který řídí životní cyklus aplikace, zajišťuje bezpečnost a spravuje vlákna [5].

- Hlavním úkolem servletu je načíst požadavek, zpracovat data (obchodní logika, interakce s databází) a vygenerovat odpověď ve formátu HTML nebo jiném vhodném typu obsahu.

Stránky JavaServer (JSP):

JSP vznikl jako doplněk servletů, který zjednodušuje generování dynamického obsahu HTML na straně serveru. Umožňoval vkládat kód jazyka Java přímo do stránek HTML pomocí speciálních značek a výrazů [11]

- JSP je efektivní zejména v raných fázích vývoje, kdy je třeba rychle vytvořit prototyp dynamických stránek. Postupem času se však rozšířené používání kódu Java v rámci JSP začalo považovat za špatnou praxi a odvětví přešlo k oddělení logiky a prezentace (MVC)[12][26].

Přechod na Javu EE (Jakarta EE):

Pro řešení složitějších problémů a zvýšení standardizace v oblasti podnikového vývoje vznikla specifikace Java 2 Platform, Enterprise Edition (J2EE), později přejmenovaná na Java EE a následně Jakarta EE [11]. Hlavním cílem bylo zjednodušit tvorbu distribuovaných podnikových aplikací při respektování zásad přenositelnosti a škálovatelnosti kódu [10].

Postupem času se do platformy Java EE začlenilo mnoho specifikací, mj:

- Enterprise JavaBeans (EJB): raný standard pro vytváření podnikových komponent podporujících transakce, bezpečnost a distribuované výpočty.
- Java Persistence API (JPA): JPA: poskytuje objektově-relační mapování (ORM), které zjednodušuje interakci s databázemi [10].
- Java Message Service (JMS): odpovídá za zasílání zpráv v asynchronních systémech.
- JavaServer Faces (JSF): komponentový framework zaměřený na vytváření rozhraní na straně serveru.

V roce 2017 přešla specifikace Java EE pod správu nadace Eclipse Foundation a dostala název Jakarta EE. Tento přechod má urychlit vývoj platformy a zajistit větší otevřenost vůči inovacím [9]. Jakarta EE zachovává kontinuitu se specifikací Java EE tím, že pokračuje ve vývoji standardů pro zabezpečení, transakce, databázové operace a interoperabilitu služeb [7].

Současné trendy a úloha Jakarta EE

Moderní serverové aplikace jsou stále častěji postaveny na architektuře mikroslužeb a využívají lehké kontejnerové frameworky (např. Spring Boot nebo MicroProfile) [9]. Přesto je Jakarta EE stále žádaná ve velkých podnikových organizacích, kde jsou důležité společné

vývojové standardy, podpora transakcí na úrovni platformy a rozsáhlá specifikace pro různé scénáře [7][8].

- MicroProfile: integruje specifikace Jakarta EE přizpůsobené pro architekturu mikroslužeb (konfigurace, metriky, vyvažování zátěže atd.), což zjednodušuje vývoj distribuovaných systémů [9].
- Kontejnerizace: díky kompatibilitě Jakarta EE s Dockerem a Kubernetes mohou vývojáři kombinovat standardní mechanismy platformy s flexibilní infrastrukturou cloudových řešení [7].

Přechod od klasických Java Servletů a JSP k Jakarta EE tak ilustruje vývoj Javy jako platformy pro vytváření škálovatelných a spolehlivých podnikových systémů [5][11]. Moderní přístupy zároveň umožňují integrovat časem prověřené standardy (JPA, EJB, JMS) s architekturou mikroslužeb a cloudovými technologiemi, což uspokojuje potřeby malých týmů i velkých organizací [7][9].

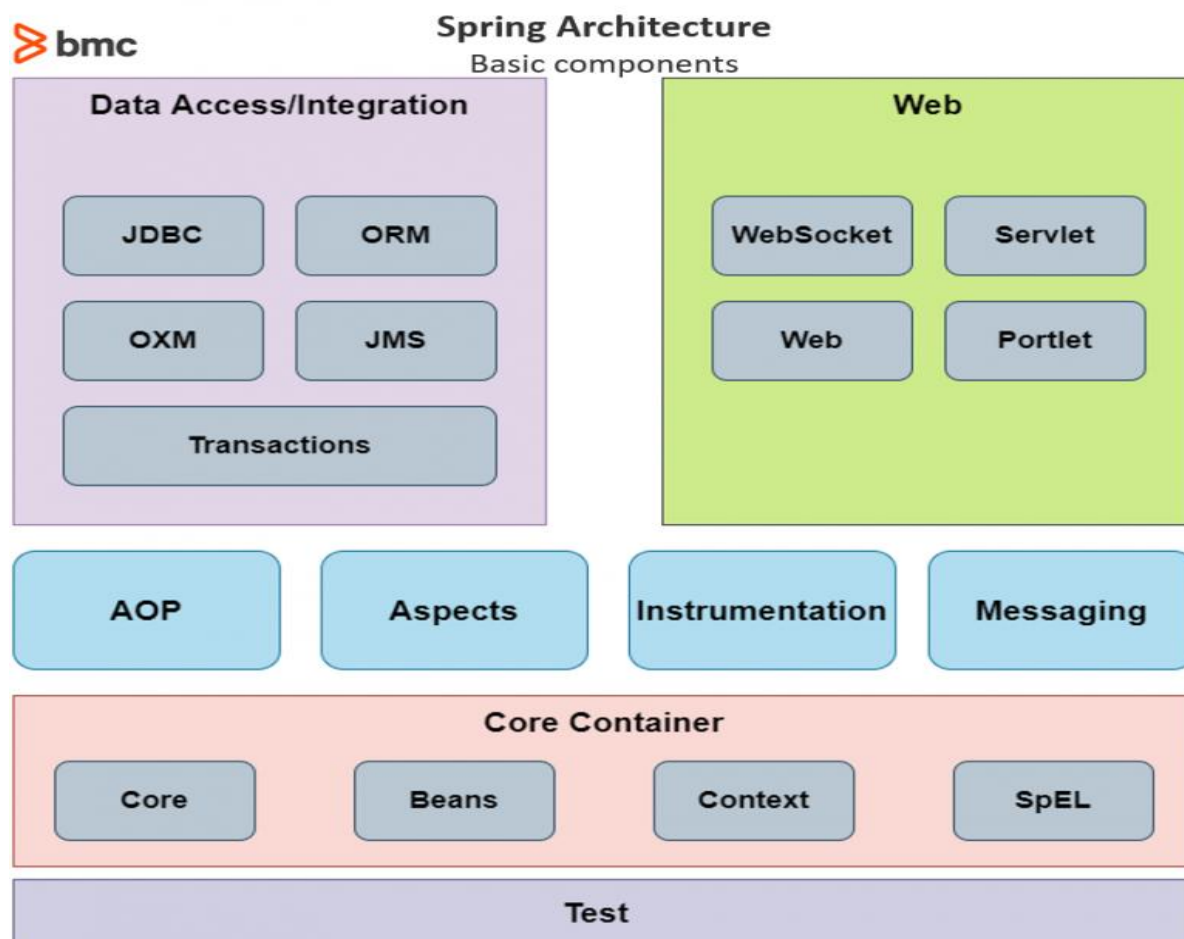
2.2.2 Hlavní Java frameworky pro vývoj serverových aplikací

V ekosystému Javy existuje mnoho frameworků, které zjednodušují a strukturují proces vytváření aplikací na straně serveru. Níže uvádíme přehled tří klíčových řešení široce používaných v podnikovém a komerčním vývoji: Spring (včetně Spring Boot), JavaServer Faces (JSF) a Vaadin. Každé z těchto řešení se zaměřuje na jiné potřeby a architektonické přístupy.

Spring Framework a Spring Boot:

Jedním z nejoblíbenějších a nejprizpůsobivějších frameworků pro vytváření aplikací na straně serveru v jazyce Java je Spring Framework. Mimo jiné nabízí moduly pro zpracování dat (Spring Data), aspektově orientované programování (AOP) a vstřikování závislostí (IoC). Spring je vhodný pro projekty libovolné velikosti, od jednoduchých služeb až po rozsáhlé distribuované systémy, a to díky své modulární povaze, která vývojářům umožňuje používat pouze ty komponenty, které jsou potřebné [7].

Spring Boot je nadstavba frameworku Spring, která zjednodušuje konfiguraci a nasazení aplikací [8]. Díky principu konvence nad konfigurací umožňuje Spring Boot rychle vytvářet funkční prototypy. Vestavěný server (Tomcat, Jetty nebo Undertow) a automatická konfigurace zkracují čas strávený konfigurací. Spring Boot je obzvláště vhodný pro architektury mikroslužeb, protože každou službu lze nasadit samostatně [9].



Obrázek 5 - Spring-architecture [29]

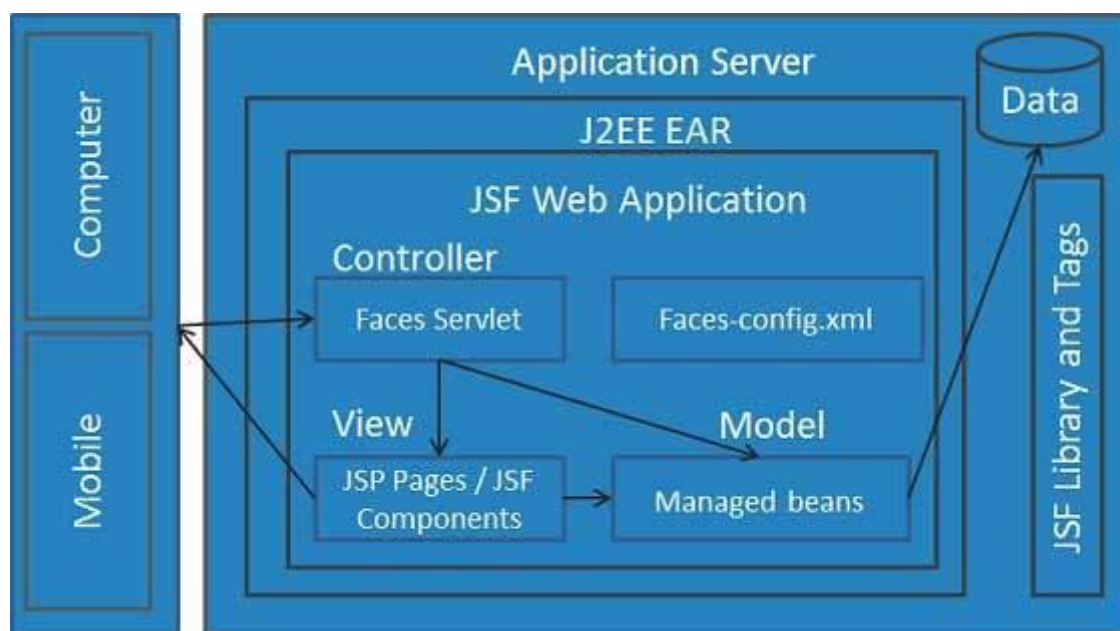
Klíčové výhody Spring a Spring Boot:

- Široký ekosystém modulů: od práce s databází až po zabezpečení (Spring Security) [8].
- Vysoká flexibilita: možnost vybrat pouze potřebné komponenty snižuje složitost projektů [8].
- Nástroje pro mikroslužby: vestavěné nástroje pro interoperabilitu služeb, integraci cloudu a monitorování [9].

- Aktivní komunita: neustálé aktualizace, bohatá dokumentace a podpora mnoha zásuvných modulů [7][8].

JavaServer Faces (JSF):

JavaServer Faces (JSF), původně vytvořený jako komponentový rámec pro standard Java EE (nyní Jakarta EE), se zabývá především zobrazováním rozhraní na straně serveru. Uživatelská rozhraní JSF tvoří soubor prvků (formulářů, tlačítek a vstupních polí), které jsou zpracovávány na straně serveru a prezentovány jako stránky HTML[11].



Obrázek 6 - JSF – Architecture [30]

Funkce JSF:

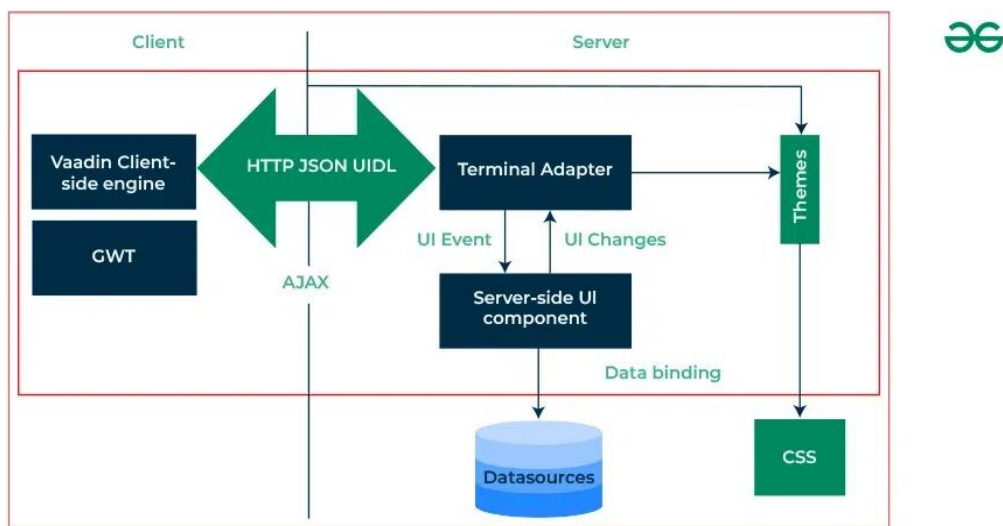
- Přístup založený na komponentách: každý prvek uživatelského rozhraní je komponentou, což usnadňuje opakované použití a zapouzdření logiky zobrazení.
- Integrace s Jakarta EE: úzké propojení se specifikacemi, jako jsou CDI (Contexts and Dependency Injection) a Bean Validation, usnadňuje vytváření monolitických podnikových aplikací [11].
- Navigační model: JSF zjednodušuje směřování stránek pomocí konfiguračních souborů XML nebo anotací.

- Omezená flexibilita na klientovi: Protože se JSF zaměřuje na vykreslování stránek na serveru, vyžaduje implementace moderních SPA (Single-Page Applications) nebo vysoce interaktivních rozhraní další nástroje (JavaScriptové frameworky)[27][30].

JSF se často používá v aplikacích, které vyžadují úzkou spolupráci s ostatními technologiemi Java EE a přísnou logiku vykreslování na straně serveru. JSF však může ustoupit flexibilitě frameworků na straně klienta pro současná webová rozhraní, která vyžadují vysokou míru zapojení na straně klienta [10].

Vaadin:

Podobně jako metoda JSF poskytuje framework Vaadin paradigma pro vytváření webových aplikací v jazyce Java založené na komponentách, i když se více zaměřuje na „tlustý“ server a „tenkého“ klienta. Základní koncepcí Vaadinu je vytvoření uživatelského rozhraní čistě v jazyce Java, přičemž framework automaticky vytváří příslušné HTML, CSS a JavaScript [7].



Obrázek 7 - Architecture of Vaadin [31]

Hlavními vlastnostmi Vaadinu jsou:

- Vývoj orientovaný na komponenty: umožňuje používat hotové vizuální komponenty (tabulky, formuláře, grafy atd.) k rychlému vytvoření aplikace.
- Zapouzdření logiky: většina logiky uživatelského rozhraní se nachází na serveru, což usnadňuje kontrolu stavu aplikace a zjednodušuje vývoj formulářů a zpracování událostí[8].
- Rozšiřitelnost a integrace: Vaadin je kompatibilní se Springem, což umožňuje použít Spring Boot pro rychlou a snadnou konfiguraci [8].
- Podpora offline a PWA: Novější verze Vaadinu umožňují vytvářet progresivní webové aplikace (PWA) při zachování pohodlí vývoje v Javě[8].

Vaadin je velmi žádaný pro aplikace, které vyžadují rychlou tvorbu prototypů uživatelského rozhraní, aniž by bylo nutné hluboce rozumět jazyku JavaScript. Někdy je však lepší využít jiné metody nebo kombinovat Vaadin s jinými knihovnamí JS, pokud má být rozhraní co nejpružnější a nejprizpůsobitelnější [7][8].

Spring Framework (včetně Spring Boot), JSF a Vaadin jsou klíčovými zástupci serverových frameworků v Javě, které pokrývají širokou škálu potřeb od architektury mikroslužeb (Spring) až po komponentově orientovaný vývoj s minimální logikou na straně klienta (JSF a Vaadin). Výběr konkrétního řešení závisí na požadavcích na architekturu aplikace, očekávaném objemu zátěže a preferencích a kompetencích vývojového týmu [7][9][11].

2.2.3 Srovnání Java frameworků – kritéria výkonnosti, bezpečnosti a snadnosti údržby

Při výběru sady nástrojů pro vývoj na straně serveru v jazyce Java se vývojáři zaměřují na několik klíčových kritérií: výkon, bezpečnost, škálovatelnost a udržitelnost. Níže se zabýváme srovnávací analýzou tří populárních frameworků - Spring (včetně Spring Boot), JavaServer Faces (JSF) a Vaadin - v kontextu těchto parametrů[7][8][9].

1. Výkonnost

Framework Spring a Spring Boot:

- Dobře se hodí pro vysoce zatížené systémy, zejména pokud jsou správně nakonfigurovány a monitorovány (Spring Boot Actuator) [8].

- Podporují architekturu mikroslužeb, což pomáhá při efektivním vyvažování zátěže.

JSF:

- Vykreslování uživatelského rozhraní na straně serveru s velkým počtem přístupů může způsobit zvýšenou zátěž serveru [11].
- Pro zachování vysokého výkonu je nutná pečlivá konfigurace ukládání stavu a ukládání do mezipaměti[8].

Vaadin:

- „Hrubý“ server s komponentovým modelem zjednodušuje logiku uživatelského rozhraní, ale při vysokém počtu požadavků generuje značný síťový provoz [7].
- Při mírném zatížení (firemní aplikace s omezeným počtem uživatelů) vykazuje dostatečnou úroveň výkonu[7].

2. Zabezpečení

Spring Security

- Poskytuje širokou sadu nástrojů pro autentizaci, autorizaci a ochranu před běžnými útoky (CSRF, XSS) [8].
- Snadno se integruje s dalšími moduly Spring a externími službami.

JSF + Java EE Security

- Může se spolehnout na standardní mechanismy Jakarta EE (JAAS, JASPIC), které poskytují vestavěné zabezpečení kontejnerů [11].
- Přizpůsobení může být složitější než u Spring a méně flexibilní pro specifické scénáře.

Vaadin

- Většina logiky uživatelského rozhraní je na straně serveru, což zjednodušuje řízení přístupu, ale plná ochrana se nejčastěji realizuje integrací se Spring Security [7].
- Tento přístup poskytuje celou řadu bezpečnostních mechanismů, podobně jako aplikace Spring.

3. Škálovatelnost

Spring (Boot).

- Vynikající pro architekturu mikroslužeb, podporuje Spring Cloud a další nástroje pro distribuované systémy [9].
- Je možná nezávislá škálovatelnost jednotlivých služeb, což je důležité pro vysoce zatížené projekty.

JSF

- Častěji používaný v monolitických podnikových aplikacích, kde se škálovatelnost dosahuje pomocí clusterování aplikačních serverů (WildFly, Payara) [10].
- Ne vždy optimální pro scénáře mikroslužeb.

Vaadin

- Umožňuje horizontální škálování při zvýšení zátěže, ale „tlusté“ vykreslování serverů může komplikovat práci s velkým počtem současně aktivních uživatelů [7].
- Stále použitelný v prostředí mikroslužeb, ale vyžaduje pečlivější návrh.

4. Udržovatelnost

Spring

- Bohatý ekosystém, jednotný styl konfigurace, aktivní komunita [8] usnadňuje údržbu.
- IoC (Inversion of Control) a anotace podporují modularitu kódu [7].

JSF

- Komponentový model, úzce související s technologiemi na straně serveru Jakarta EE, je výhodný, pokud se aplikace řídí tradičním schématem MVC [11].
- U složitých rozhraní uživatelského rozhraní, která vyžadují interaktivitu, je často nutné dodatečně připojit JavaScriptové rámce.

Vaadin

- Výhodné pro týmy, které se dobře vyznají v Javě a nechtějí se hlouběji nořit do JavaScriptu; uživatelská rozhraní se vytvářejí z hlediska komponent a událostí na serveru [7].
- Lze integrovat externí knihovny JS, ale vyžaduje to další úsilí.

Každý z frameworků - Spring (Boot), JSF nebo Vaadin - má své silné a slabé stránky:

Spring (Boot): Vysoký výkon a škálovatelnost, flexibilní zabezpečení, široký ekosystém modulů[29].

JSF: Těsná integrace s Jakarta EE, zjednodušené vykreslování na straně serveru, ale nemusí být tak efektivní v projektech mikroslužeb.

Vaadin: Komponentový přístup založený na Javě, rychlý vývoj formulářů a sestav, ale „tlustý“ server znesnadňuje škálování a práci s interaktivními rozhraními.

Konečná volba závisí na funkčních požadavcích, architektonickém modelu (monolit vs. mikroslužby), bezpečnostních požadavcích a zkušenostech týmu [10][11].

2.2.4 Best practices a architektonické přístupy při využití Java frameworků

Moderní frameworky Java (Spring, JSF, Vaadin atd.) nabízejí širokou škálu nástrojů pro efektivní tvorbu serverových aplikací. Kvalitu výsledného systému však do značné míry určuje nejen funkčnost frameworku, ale také to, jak kvalifikovaně vývojáři aplikují architektonické vzory, zajišťují testovatelnost a zohledňují potřeby škálování a podpory. Níže jsou uvedeny klíčové osvědčené postupy a architektonické přístupy pro používání frameworků Java[12][13].

1. Vrstvená architektura (Layered Architecture).

Mnoho projektů v jazyce Java se řídí vrstevnatou strukturou (Presentation, Business, Persistence atd.) a jasně odděluje odpovědnosti jednotlivých vrstev [12].

- Zjednodušená údržba: změny v jedné vrstvě minimálně ovlivňují ostatní.
- Testovatelnost: každou vrstvu lze testovat nezávisle pomocí zásuvných modulů nebo maket objektů.
- Princip jedné odpovědnosti: každá část kódu je odpovědná za přesně definovaný úkol.

2. Architektura mikroslužeb

Obliba mikroslužeb je dána potřebou flexibility a škálovatelnosti. Javovské frameworky (Spring Boot, Quarkus, Micronaut) tento přístup aktivně podporují [13].

- Nezávislé nasazení: každou službu lze aktualizovat a škálovat, aniž by to ovlivnilo ostatní.
- Zjednodušená odolnost proti chybám: selhání jedné služby minimálně ovlivní fungování celého systému [13].
- Integrace DevOps: standardní nástroje pro kontinuální integraci a orchestraci (Kubernetes, Docker) se dobře hodí k přístupu mikroslužeb.

3. Návrh řízený doménou (DDD).

Velké podnikové projekty využívají přístup DDD oddělením doménové, aplikační, infrastrukturní a dalších vrstev, aby byla logika předmětné domény „čistá“ a srozumitelná [12][14].

- Všudypřítomný jazyk: vývojáři a podnikoví experti používají jednotný slovník pojmů, což zjednodušuje komunikaci.
- Entity a agregáty: každý model business objektu je jasně strukturovaný, což snižuje riziko logických chyb.
- Úložiště a továrny: detaily infrastruktury (např. databázové operace) jsou izolovány od kódu domény.

4. Šestiúhelníková architektura (Hexagonal Architecture).

Šestiúhelníková architektura navrhuje oddělit obchodní logiku od vnějších rozhraní (databáze, API, uživatelské rozhraní). Tento přístup se dobře hodí k jakémukoli frameworku jazyka Java [14]:

- Porty a adaptéry: aplikace jsou strukturovány prostřednictvím rozhraní (portů) a jejich implementací (adaptérů).
- Flexibilita: lze snadno nahradit jakoukoli externí komponentu (např. přejít z jednoho RDBMS na jiný), aniž by se změnila obchodní logika.
- Zjednodušení testování: obchodní pravidla jsou testována nezávisle na konkrétních technologiích ukládání a přenosu dat.

5. Testování a CI/CD

Kvalitní vývojové postupy zahrnují integraci kontinuálního testování a nasazení:

- Unit testy pro metody a třídy, integrační testy pro testování interakcí mezi vrstvami[12].
- Potrubí CI/CD (např. Jenkins, GitLab CI, GitHub Actions) zajišťují automatické sestavení, testování a nasazení při každé změně kódu [8].
- Kontejnerizace (Docker, Kubernetes) usnadňuje přehrávání prostředí a minimalizuje „problémy související s konfigurací“, které jsou vlastní různým vývojovým a provozním prostředím[32].

6. Zpracování dat a ukládání do mezipaměti

Při práci s vysoce zatíženými aplikacemi je důležité správné zpracování dat a optimalizace dotazů [14]:

- Použití ORM (např. JPA/Hibernate) s vědomým laděním strategií načítání dat (Lazy/Eager Loading).
- Mechanismy ukládání do mezipaměti (např. Spring Cache, Hazelcast) pro snížení zatížení databáze a doby odezvy.
- Rozdělení databáze a replikace, pokud potřebujete škálovat pro ukládání velkého množství dat.

7. Zabezpečení a řízení přístupu

Bezpečnost je kritickým aspektem vývoje aplikací na straně serveru:

- Spring Security a podobné moduly v jiných frameworkcích poskytují flexibilní konfiguraci ověřování a autorizace [8].
- Implementace šifrování kritických dat na základní úrovni a při přenosu (TLS/SSL, HTTPS).
- Nastavení zabezpečení na úrovni kontejneru (Jakarta EE) a aplikačního serveru (Tomcat, WildFly) posilují celkovou úroveň ochrany .

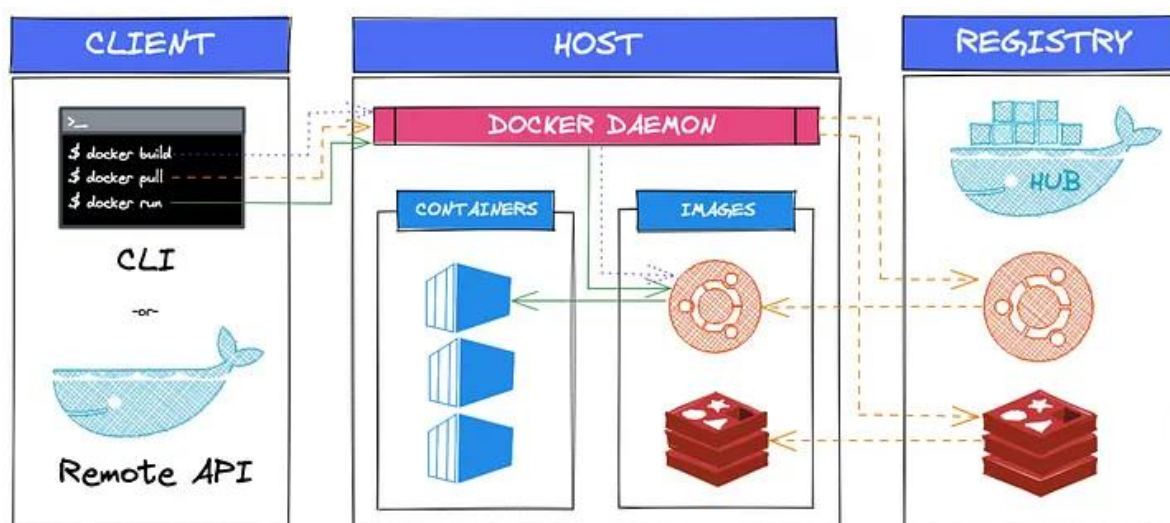
Osvědčené postupy při používání frameworků Java se omezují na dodržování architektonických principů (DDD, hexagonální architektura, mikroslužby), správnou organizaci aplikačních vrstev, účinné testování a zvláštní pozornost věnovanou bezpečnosti a výkonu. Správná aplikace těchto přístupů umožňuje vytvářet spolehlivá a škálovatelná řešení a také usnadňuje další podporu a rozvoj aplikace[8][14].

2.2.5 Nástroje pro správu a orchestraci aplikací v Java prostředí

Používání nástrojů, které usnadňují nasazení, správu a škálování programů v jazyce Java, je v současném vývoji klíčové, zejména v architektuře mikroslužeb. Patří mezi ně technologie orchestrace (Kubernetes) a kontejnerizace (Docker). Usnadňují spolupráci mezi vývojovými a provozními týmy a jsou silně spojeny s přístupy DevOps[32].

1. Kontejnerizace založená na Dockeru

Jednou z nejoblíbenějších technologií pro umístění aplikací do kontejnerů je Docker. Odstraňuje rozdíly mezi vývojovým a produkčním prostředím tím, že umožňuje vytvářet obrazy Docker ze souborů JAR nebo WAR, doplněné o všechny potřebné knihovny a konfigurace, v kontextu aplikací Java[32].



Obrázek 8 - Architektura Dockeru na vysoké úrovni [32]

Soubor Docker je typ „receptu“ na vytvoření obrazu, který obsahuje definici prostředí.

Zjednodušení procesu společného vývoje: V rozptýlených týmech lze verze obrazů ukládat a rychle přenášet pomocí Docker Hub a dalších úložišť [8][13].

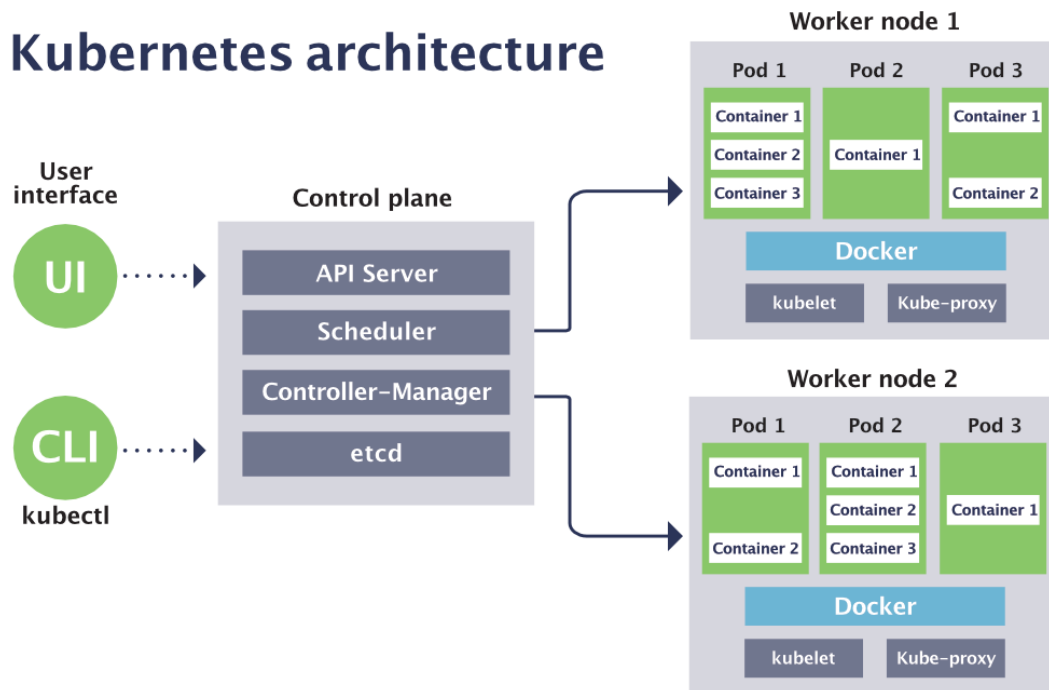
2. Orchestrace Kubernetes

Kubernetes se využívá v případech, kdy je vyžadována správa velkého počtu kontejnerů, což je běžné u mikroservisních systémů.

Automatizované nasazení a škálování: Kubernetes sleduje stav kontejnerů a dynamicky mění počet kopií služby podle zatížení.

Kromě optimalizace rozložení provozu nabízí vyvažování zátěže a zjišťování služeb praktický způsob, jak se služby navzájem vyhledávají.

Deklarativní manifesty (YAML/JSON) usnadňují definování infrastruktury jako kódu, což usnadňuje opakovatelné nasazení a verzování [9][13].



Obrázek 9 - Kubernetes [34]

3. Postupy DevOps a CI/CD

Metody kontinuální integrace a doručování (CI/CD) jsou silně spjatý se současnou metodikou, která je postavena na Dockeru a Kubernetes:

Sestavovací potrubí (GitLab CI, GitHub Actions, Jenkins): aplikace je vytvořena, otestována a po každé změně kódu zabalena do obrazu Docker.

Automatické nasazení: Kubernetes (nebo podobný orchestrační nástroj) aktualizuje nasazené služby, jakmile je obraz odeslán do kontejnerového úložiště, pokud testy proběhly úspěšně [8].

Správa verzí snižuje prostoje a rizika chyb při nasazení tím, že zajišťuje hladký a rychlý přechod mezi jednotlivými verzemi [14].

Současné metody vývoje aplikací v jazyce Java, včetně mikroslužeb i monolitických řešení, jsou proto neoddelitelně spjaty s kulturou DevOps.

4. Problémy s Kubernetes a kontejnery

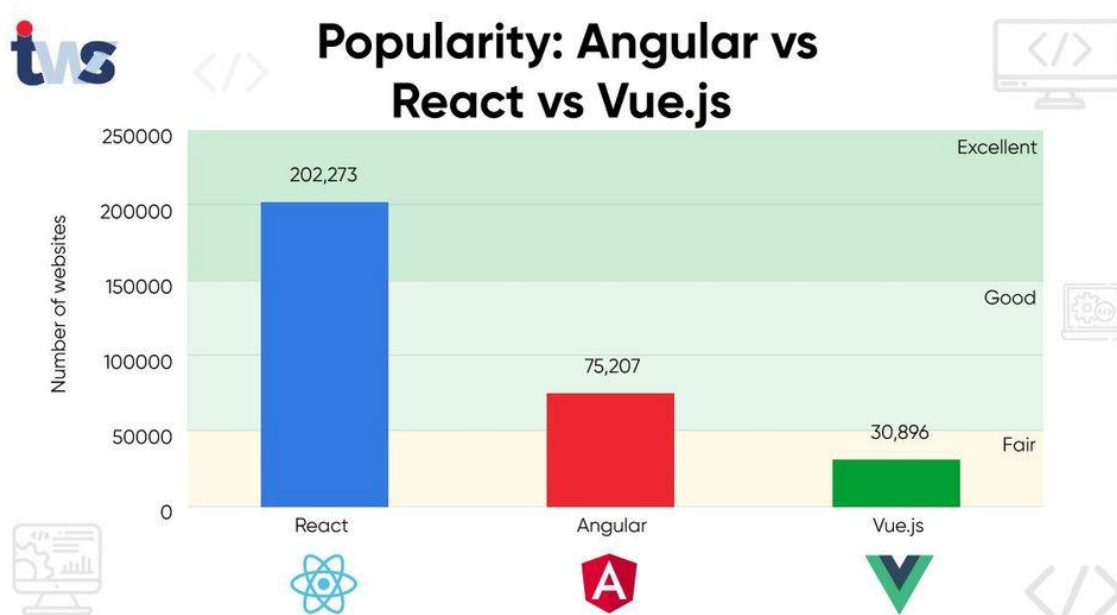
Optimalizace paměti a garbage collector: JVM potřebuje pečlivě nakonfigurovat omezení paměti v kontejneru; pokud se tak nestane, může dojít ke snížení výkonu nebo k chybě `OutOfMemoryError` [10].

Zabezpečení: Orchestrační platforma i obrazy kontejnerů musí být pravidelně aktualizovány. Zranitelnosti existují v zastaralých verzích JDK nebo knihoven.

Nástroje pro kontejnerizaci a orchestraci, jako jsou Docker a Kubernetes, jsou dnes pro vývoj současných aplikací v jazyce Java nezbytné. Usnadňují rychlé vydávání nových verzí, nabízejí předvídatelné prostředí a usnadňují škálování a monitorování. Zároveň je třeba brát v úvahu obavy o bezpečnost, životaschopnost přijetí sofistikovaných orchestračních nástrojů a konfiguraci JVM v kontejnerech [10][14].

2.3 Frameworky pro klientskou část

Rámce na straně klienta jsou pro současný vývoj webových stránek nezbytné, protože usnadňují práci s dynamickým obsahem, spravují stav aplikací na straně klienta a usnadňují komunikaci s rozhraními API na straně serveru [2][3]. Mezi nejoblíbenější možnosti patří Vue, React a Angular; všechny mají velkou komunitu, robustní prostředí a vynikající výkon [15][16][17].



Obrázek 10 - Popularity of Angular, React and Vue.js [36]

2.3.1 Přehled a srovnání populárních řešení (Angular, React, Vue)

Angular, React a Vue jsou tři hlavní frameworky, na kterých je postaven moderní frontendový vývoj. Každý z nich nabízí jedinečný způsob architektury aplikace, správy stavu a interakce s DOM [2]. Pomocí těchto technologií lze vytvářet projekty různých velikostí a úrovní složitosti - od malých jednostránkových aplikací až po rozsáhlé podnikové systémy [15].

Angular

Aby bylo možné zajistit přesnou detekci chyb ve fázi kompilace, byl Angular, který je spravován společností Google a vyvinut z AngularJS, od druhé verze zcela přepsán do jazyka TypeScript [15]. Tento framework se vyznačuje „přísnou“ architekturou, která aktivně využívá Dependency Injection, aby usnadnila testování a údržbu, a zároveň nabízí systém modulů, komponent a služeb. Pro Angular se hodí středně velké a větší projekty, kde jsou rozhodující jasné normy a konzistentní kód [16]. Může však být náročnější na naučení a přepínání mezi hlavními verzemi často vyžaduje hodně práce [17].

React

Ve skutečnosti je React základem ekosystému, který vývojáři rozšiřují o další nástroje, přestože je považován za sadu nástrojů pro tvorbu uživatelského rozhraní [2]. Jednou z jedinečných vlastností Reactu je deklarativní popis uživatelského rozhraní pomocí syntaxe JSX, která sjednocuje značení a funkce v jediném souboru a usnadňuje čitelnost kódu [15]. To nabízí vynikající efektivitu při obnovování uživatelského rozhraní ve spojení s virtuálním DOM. React zároveň dává týmu autoritu nad návrhem aplikace, přičemž o směřování, správu stavu a další funkce se starají frameworky třetích stran [16]. Ačkoli tato metoda nabízí flexibilitu, vývojáři musí zachovávat konzistenci ve svých rozhodnutích a disciplínu, zejména při práci na rozsáhlých projektech [17].

Vue

Evan Yu vyvinul Vue.js s cílem stát se „progresivním frameworkem“, který lze snadno integrovat i do již existujících aplikací a který má nízký vstupní práh [2]. Vue usnadňuje vytváření dynamických rozhraní kombinací reaktivního datového modelu a gramatiky šablon. Navíc usnadňuje progresivní škálování: zatímco velké projekty mohou využívat řešení jako Nuxt (vykreslování na straně serveru) nebo Vuex (globální stav), drobné aplikace mohou běžet jen s nezákladnějšími funkcemi [15]. Najít vysoce specializované balíčky však může být občas náročné, protože podíl Vue na trhu je stále o něco nižší než podíl Angularu nebo Reactu [17].

Analýza ve srovnání

- Organizace kódu a architektury

Struktura (moduly, komponenty a služby) je v systému Angular přísně regulována, což podporuje jednotnost, ale brání učení [2]. Naproti tomu React nabízí pouze jednoduchý vykreslovací rámec a ponechává vývojářům možnost vlastního výběru architektury [15]. Naproti tomu Vue poskytuje flexibilní, ale srozumitelnou metodu, která klade důraz na přehlednost struktury souborů a jednoduchost integrace [16].

- Výkon

Reaktivní jádro v Angularu a virtuální DOM v Reactu a Vue jsou dva příklady efektivních přístupů k DOM, které používají všechny tři frameworky [17]. V důsledku toho se rozdíly v rychlosti většinou projevují ve složitých situacích vykreslování a jsou závislé na celkové architektuře aplikace [2].

- Škálovatelnost

Podniková řešení v minulosti hojně využívala Angular a React, které poskytují bohaté možnosti pro budování rozsáhlých systémů [15]. Ačkoli Vuex a Nuxt mohou dobře fungovat i ve velkých projektech s pomocí Vue CLI, Vue je populárnější mezi startupy a menšími týmy [16].

- Jednoduchost učení

Většina vývojářů zjistila, že React vyžaduje znalost JSX a další „hromadu“ knihoven, zatímco Angular je náročnější se naučit kvůli mnoha zabudovaným nápadům [2]. Mnoho vývojářů se domnívá, že Vue je na počátku jednodušší na naučení, ačkoli má stejné možnosti jako sofistikované online aplikace [17].

Díky různým strukturám organizace kódu a nástrojům pro správu stavu jsou Angular, React a Vue i nadále třemi hlavními „pilíři“ vývoje webových aplikací na straně klienta [15]. Vue upřednostňuje jednoduchost a přírůstkovou funkčnost, React je nejuniverzálnější a Angular je vynikající pro velké, organizované projekty [16]. Všechny tři technologie díky své zřejmé modularitě, živé komunitě a sortimentu pomocných knihoven pomáhají zefektivnit vývoj a zvýšit rychlost a kvalitu konečného výsledku [17].

2.3.2 Rozšířená řešení a novinky (Svelte, Ember.js)

Přestože jsou Angular, React a Vue lídry v oboru vývoje frontendů, další technologie poskytují alternativní způsob tvorby online aplikací. Mezi zajímavými alternativami vynikají Svelte a Ember.js, protože mají zvláštní postoj k architektuře, výkonu nebo procesu vývoje [2].

Svelte

- Spíše než použití runtime, kompilace

Svelte je především to, že většina práce je dokončena ve fázi konstrukce; komponenty jsou převedeny do čistého JavaScriptového kódu a nevyžadují další „vrstvu“ frameworku prohlížeče [18].

- Reaktivní model

Správa stavu a vykreslování komponent se stává viditelnější díky reaktivnímu přístupu Svelte na úrovni proměnných [2]. Freeman [16] tvrdí, že díky tomu, že vývojář nemusí pracovat s virtuálním DOM, je díky tomuto způsobu myšlení kód jednodušší.

- Společenství a aplikace

Projekty, které vyžadují vysokou rychlost načítání a efektivní využití zdrojů, používají spíše Svelte [16]. Přestože má Svelte menší komunitu než React nebo Angular, postupně získává na popularitě v podnikové sféře [17].

Ember.js

- Dlouhá historie a přísné dohody

Jedním z tradičnějších frontendových frameworků je Ember.js, který uplatňuje přísný přístup k řízení projektů a snaží se usnadnit práci velkým týmům [2]. Zde platí přísloví „konvence je důležitější než konfigurace“, což je srovnatelné s myšlenkou stojící za Rails [19].

- Modulární ekosystém a CLI

Vytváření kódu, testování a správa závislostí jsou jen některé z běžných úkolů, které provádí rozhraní Ember CLI [17]. Pro týmy, které dávají přednost méně přizpůsobenému, hotovému organizovanému prostředí, je to zjednodušení, tvrdí Freeman [15].

- Obliba a vývoj

V minulosti se pro sofistikované aplikace, které vyžadují jasný design a solidní API, často používal Ember.js [16]. Částečně díky podpoře velkých firem a rozšíření reaktivních principů je nyní stále více oblíbenější než React, Angular nebo Vue [2].

Srovnání a aplikace

- Výkon společnosti Svelte

U aplikací, kde je rozhodující rychlost vykreslování a malá velikost bandelu, může Svelte zajistit minimální režii tím, že se zbaví „vrstvení“ frameworků a přesune hlavní funkce do fáze sestavení [18][15].

- Přísné pokyny (Ember.js)

Ember rigidně specifikuje strukturu katalogu (modely, vzory a komponenty) a nabízí konzistentní nástroje pro běžné činnosti v souladu s myšlenkou, že „o všem pod kapotou se rozhoduje za vás“ [2]. Týmy, které potřebují rychle vytvořit jednotný vývojový styl, mohou tuto techniku využít.

- Společenství a rozšiřitelnost

Vzhledem k poměrně omezené ekologii Svelte může být obtížné najít v něm vysoce specializované programy [17]. Přestože má Ember stabilní sadu nástrojů a větší komunitu, v poslední době ztrácí půdu pod nohama ve srovnání s konkurencí [15][16].

Cesty vývoje frontendových frameworků Svelte a Ember.js se liší; první z nich klade důraz na „kompilační přístup“ a minimum výsledného kódu [18], zatímco druhý je založen na „přísných konvencích“ a osvědčených metodách [19]. Svelte nebo Ember.js mohou být vhodnější v závislosti na prioritách projektu (vysoký výkon, touha po tradičním stylu, potřeba rychlé integrace atd.). Ekosystém těchto technologií je však poměrně úzký a jejich popularita může ztížit výběr potřebných knihoven [2][15][16].

2.3.3 Nástroje pro optimalizaci a testování klientských frameworků

Výběr frameworku (Angular, React, Vue, Svelte nebo Ember.js) je pro vývoj frontendu zásadní, ale stejně tak je důležité používat nástroje, které optimalizují výsledný produkt a zaručují vynikající kvalitu prostřednictvím testování [2]. Níže je uveden přehled hlavních možností frameworků na straně klienta.

Sestavování a optimalizace

- Webpack

Jedním z nejpoužívanějších bundlerů je stále Webpack, který umožňuje sestavit moduly JavaScriptu (spolu se styly, obrázky a dalšími komponentami) do jednoho balíčku [15]. Za účelem zvýšení rychlosti načítání stránek podporuje řadu zásuvných modulů a zavaděčů, které podporují miniaturizaci, transpilaci (Babel) a rozdělení kódu na části [3].

- Vite

Vite Využitím nativní podpory modulů ES v současných prohlížečích poskytuje Vite odlehčené nastavení a rychlejší vývojový server [16]. To je obzvláště užitečné pro velké projekty, jako jsou SPA, protože to zkracuje čas potřebný k aktualizaci aplikace během vývoje [17].

Rollup a Parcel

V ekosystému Svelte se Rollup vyskytuje nejčastěji a používá se k vytváření knihoven [18]. Parcel tento proces zjednodušuje, zejména u malých aplikací, tím, že umožňuje automatické vytváření závislostí bez nutnosti lidské konfigurace [19].

Testování

- Unite testy

Jest se často používá v aplikacích React, ale lze jej snadno integrovat i do Vue a se správnými pluginy i do Angularu [2]. Jedním z mnoha pluginů, které jsou v Jest k dispozici, je podpora snapshot testů, které pomáhají identifikovat změny rozhraní.

Jasmine/Karma

Konvenční technologie, které umožňují provádět testy v reálném prohlížeči, jsou v komunitě Angularu hojně využívány [15].

Testování E2E (end-to-end)

- Cypress

Cypress poskytuje snadno použitelné rozhraní pro testování v prohlížeči, které replikuje skutečné chování uživatele a nabízí důkladné ladění [16].

Playwright je novější produkt společnosti Microsoft, který nabízí robustní rozhraní API pro propojení s DOM a umožňuje testování napříč prohlížeči [17].

- Integration testy

Ideální pro hodnocení interoperability mnoha modulů nebo komponent, například při přesunu dat mezi klientem a serverem [2]. U návrhů mikroslužeb je tento typ testování obzvláště důležitý, protože zajišťuje, že úpravy jedné služby neovlivní ostatní služby [15].

CI/CD a automatizace:

Sestavování a testování frontendů je součástí technologií kontinuální integrace (GitLab CI, GitHub Actions, Jenkins), které používá mnoho týmů [16]. Automatizovaný proces pipeliningu se spouští vždy, když je úložiště upraveno:

- Construct

vytváří statické prvky (CSS, písma, obrázky), transpiluje a minifikuje kód a vytváří efektivní balíčky [3].

- Provádění testů

Před sloučením větví se v různých nastaveních provádějí jednotkové, integrační a E2E testy, aby se našly problémy [17].

- Deploy

Nasazení Výkonnost a metriky jsou podle potřeby sledovány při přesunu dokončeného balíčku na produkční server nebo do cloudu [18].

3 EFEKTIVITA WEBOVÉHO VÝVOJE: KRITÉRIA A METODY HODNOCENÍ

Tato kapitola se zaměřuje na osvědčené metody vytváření složitých podnikových aplikací. Začneme nastíněním základů návrhu řízeného doménou (DDD), včetně všudypřítomného jazyka, agregátů, úložišť a továren, a popíšeme, jak tyto myšlenky zvyšují kvalitu modelu domény. Dále představíme návrh Hexagonal (Ports & Adapters), který usnadňuje testování a sdílení technologií a zároveň spolehlivě odděluje obchodní logiku od vnějších rozhraní. Nakonec probereme, jak tyto dvě strategie kombinovat, abychom optimalizovali modularitu a udržitelnost aplikace.

3.1 Definice efektivity ve webovém vývoji

Ukazatelem toho, jak dobře jsou zdroje (čas, týmová práce, peníze a technologické možnosti) využity k dosažení cílů projektu, je efektivita vývoje webu. Kromě rychlosti dokončení práce se efektivita týká také kvality hotového produktu a snadnosti jeho údržby a rozšiřování [6][9].

Richardson [9] uvádí, že hlavními kritérii hodnocení efektivity vývoje webu jsou následující:

- Doba uvedení na trh neboli rychlost vývoje je doba potřebná k implementaci a vydání první funkční verze programu.
- Kvalita a udržitelnost kódu: snadnost, s jakou lze provádět úpravy projektu po jeho spuštění.
- Výkon a škálovatelnost: schopnost aplikace trvale fungovat za rostoucího počtu uživatelů a zátěže.
- Náklady na vývoj a údržbu: cena finančních, lidských a časových zdrojů potřebných k realizaci projektu a zajištění průběžné podpory [8].

Bloch [6] a Oaks [10] dále zdůrazňují význam udržitelnosti jako samostatného kritéria, které ukazuje, jak jednoduché je provádět změny a přepracování. Vzhledem k tomu, že tyto metriky mají zásadní vliv na celkovou flexibilitu projektu a rychlost změn, je třeba při hodnocení efektivity zohlednit také míru modularity a opakovaného použití kódu [6][7].

Shrnutím výše uvedeného je efektivita vývoje webových aplikací kompletní atribut, který představuje soulad mezi rychlostí vývoje, kvalitou výsledného produktu a náklady na vývoj a údržbu aplikace [10]. Při posuzování a porovnávání různých metodik a nástrojů (např. frameworků) je třeba kromě technických vlastností zohlednit i takové aspekty, jako je vliv na produktivitu týmu, snadnost propojení s jinými systémy a možnost vzniku technického dluhu [8][9].

3.2 Hlavní metriky a kritéria hodnocení

Pomocí konkrétních ukazatelů a kritérií se objektivně porovnávají různé techniky, technologie a nástroje, aby se určila efektivita vývoje webových stránek. Nejčastěji zdůrazňované důležité klíčové ukazatele výkonnosti jsou následující:

- Doba uvedení na trh (nebo doba vývoje)

Doba realizace projektu, která měří dobu od zahájení návrhu do vydání první funkční verze aplikace, je jedním z nejvýznamnějších ukazatelů efektivity vývoje webu [10][22]. Mezi významné výhody kratší doby vývoje patří úspora nákladů a rychlý vstup koncových uživatelů, což je důležité zejména ve vysoce konkurenčním prostředí [8][9].

- Reaktivita a výkonnost

K hodnocení výkonnosti webových aplikací se používají četná technická měřítka, jako je doba načítání stránek, zpoždění odezvy serveru API a využití zdrojů serveru a klienta[10][20]. Vysoký výkon má přímý vliv na udržení zákazníků a uživatelskou zkušenost. V praxi se odezva aplikací při rostoucí zátěži analyzuje pomocí monitorovacích systémů (Prometheus, Grafana) a zátěžových testů (Apache JMeter, Gatling) [7][9].

- Škálovatelnost

Schopnost aplikace rozšiřovat se s rostoucí uživatelskou základnou, aniž by vyžadovala výrazné architektonické úpravy, se nazývá škálovatelnost [8][21]. Patří sem schopnost „rozdělit“ monolitickou strukturu na mikroslužby, je-li to nutné, a techniky horizontálního škálování, jako jsou Docker a Kubernetes [9][20].

- Rozšiřitelnost i udržitelnost

Kritérium udržitelnosti měří snadnost přidávání funkcí a opravování problémů[6][8]. Zde je rozhodující modularita, kvalita kódu a přístupnost dokumentace. Význam architektonických vzorů, které zajišťují adaptaci a flexibilitu při vývoji systému, zdůrazňuje [21].

- Bezpečnost

Podle Richardsona [9] je bezpečnost jedním z nejdůležitějších faktorů pro online aplikace. Navrhuje správnou autentizaci a autorizaci, šifrování citlivých dat a odolnost proti útokům (SQL-injection, XSS) [8][20]. Kromě výběru silných rámců je pro dosažení vysoké bezpečnosti nutné rutinně vyhodnocovat zranitelnosti a aktualizovat používané komponenty [21].

- UX/UI neboli uživatelský zážitek

Názory koncových uživatelů na aplikaci jsou přímo ovlivněny faktory, jako je kvalita rozhraní, použitelnost a intuitivní logika [2][15]. Mezi kvantitativní a kvalitativní techniky, které se používají, patří A/B testování, průzkumy, heatmapy a další analytické nástroje.

- Cena tvorby a údržby

Ke kvantifikaci nákladové efektivity se používají výdaje na realizaci projektu a průběžnou údržbu [6][22]. Zahrnují se sem výdaje na infrastrukturu, licence, mzdy vývojářů a testerů a výdaje na údržbu a aktualizaci systému. Při výběru technologického zásobníku a volbě architektury je přesné posouzení nákladů klíčové [20][21].

Hodnocení efektivity vývoje webu proto vyžaduje důkladný proces, který zohledňuje organizační faktory (snadnost údržby, standardizace a riziko technického dluhu), stejně jako technické ukazatele (výkon, škálovatelnost a bezpečnost) a ekonomické faktory (doba vývoje, celkové náklady) [8][9][15]. Dobře definovaná sada měření pomáhá týmům přijímat informovanější rozhodnutí o nástrojích a rámcích, sestavování rozpočtu a zefektivňování procesů, což zlepšuje kvalitu a konkurenceschopnost výsledného produktu [20][21].

3.3 Vliv využití frameworků na kvalitu a výkonnost webových aplikací

Rámce jako Angular, React, Vue a Spring - a mnoho dalších - hrají zásadní roli při zefektivňování architektury a zrychlování procesu vývoje v současném vývoji webových

aplikací. Při správném využití mohou výrazně zlepšit kvalitu zdrojového kódu, urychlit vývoj a zvýšit výkon aplikací[20][21]. Pokud je však framework nevhodně zvolen nebo nakonfigurován, může mít jeho využití za následek i určité režijní náklady nebo omezení [22].

1. Zvýšení jednotnosti a kvality kódování

- Jasně definované návrhové vzory a struktura

Rámce podporují používání osvědčených architektonických principů, jako jsou standardy návrhu kódu, vzory Dependency Injection, MVC a MVVM [6][21]. To usnadňuje pochopení a údržbu kódu, zefektivňuje testování a vede k přehlednější struktuře projektu [9].

- Omezení nadbytečného kódu

Směrování, validace formulářů a správa stavů patří k mnoha běžným povinnostem, které jsou přeneseny do frameworku, což snižuje množství „samoúčelných“ řešení a možných chyb [2][22]. Větší jednotnost přístupu urychluje nástup nových vývojářů.

- Kombinace testovacích nástrojů

Pohodlné postupy jednotkového a integračního testování nabízí většina současných frameworků [23]. Při dodržování osvědčených postupů (TDD, BDD) se zvyšuje celková kvalita a spolehlivost aplikace [21].

2. Vliv na výkon

- Optimalizace vykreslování

Aby se snížilo zatížení prohlížeče, frontendové frameworky jako Angular, React a Vue se ve velké míře spoléhají na virtuální DOM a reaktivní metody obnovy [2][15]. To snižuje latenci vykreslování rozhraní a umožňuje aplikacím rychleji reagovat na vstupy uživatelů.

- Minifikace a automatické načítání

Minifikaci, „treeshaking“ a chunking kódu implementují serverové frameworky (Spring Boot) a frontendové bundlery (Webpack, Vite), aby urychlily počáteční načítání [20]. Uživatelé tak mohou používat rozhraní, aniž by museli čekat na načtení celého kódu [8].

- Režijní náklady a abstrakce

Pokud rámec obsahuje mnoho funkcí, které se v daném projektu nevyužívají, může to mít za následek dodatečnou režii [10][24]. Nadměrná složitost komplikuje optimalizaci a

prodlužuje dobu načítání. Rozhodující je pečlivý výběr sady nástrojů (například pokud není vyžadován plný výkon „plnohodnotného“ řešení, použijte „mikrorámce“ [9].

3. Flexibilita a škálovatelnost

- Udržovatelnost a osvědčené postupy

Přechod od malé aplikace k velké službě usnadňují vestavěná řešení frameworků pro škálování a distribuované architektury (Spring Cloud, Angular CLI pro mikrofrontend) [15][21].

- Flexibilita při přizpůsobování

Ačkoli mnoho frameworků podporuje myšlenku, že existuje „správný způsob vývoje“, mnohé z nich často obsahují rozšiřující nástroje umožňující přizpůsobení (middleware, pluginy) [20][22].

Vzhledem k tomu, že frameworky obsahují vestavěné testovací nástroje, reaktivní aktualizací mechanismy, osvědčené architektonické vzory a snadnou interakci s procesy DevOps, mohou výrazně zvýšit kvalitu kódu a výkonnost online aplikací [2][21][23]. Je třeba brát v úvahu potenciální režii a „nadbytečnost“ funkcí, které mohou mít nepříznivý vliv na dobu načítání a ztěžovat optimalizaci [22][24]. Dosažení rovnováhy mezi snadným vývojem, vysokým výkonem a stabilitou výsledného systému je možné správným výběrem a úpravou rámce, který zohledňuje cíle projektu [8][9][20].

4 SROVNÁVACÍ ANALÝZA VÝVOJE S VYUŽITÍM FRAMEWORKŮ A BEZ NICH

V této kapitole jsou shrnuty koncepty bezproblémového nasazení webových aplikací a zajištění kvality. Nejprve rozlišujeme mezi třemi úrovněmi testování - jednotkovým, integračním a end-to-end - a ukazujeme použití nástrojů jako Cypress, Mockito a JUnit. Poté jsou popsány základy kontejnerizace (Docker, Kubernetes) a CI/CD pipeline (Jenkins, GitLab CI, GitHub Actions), které automatizují sestavování, testování a nasazení. V poslední části se budeme zabývat cachováním, přičemž probereme vše od základního cachování HTTP přes distribuované cachování až po úložiště v paměti (Redis, Ehcache), a popíšeme, jak vhodně vytvořit vrstvy pro ukládání a načítání často používaných dat.

4.1 Metodologie srovnávací analýzy

V této části je popsán metodický rámec, který umožňuje objektivní hodnocení procesu vývoje webových aplikací s využitím rámců a bez nich. Hlavním cílem takové studie je posoudit, do jaké míry ovlivňuje použití určitého rámce produktivitu, kvalitu kódu, rychlost vývoje a celkovou efektivitu projektu [21][24]. Klíčové je předem stanovit sadu metrik, měřících nástrojů a parametrů experimentu [22][25].

4.1.1 Výběr metrik a rozsah studie

- Technické parametry

Propustnost a doba odezvy se hodnotí porovnáním výkonu aplikace při různé intenzitě požadavků pomocí nástrojů pro testování zátěže (např. JMeter, Gatling) [10][20].

Použitelnost a rychlost přístupů DevOps (Jenkins, GitLab CI) se odráží v době sestavení a nasazení, zejména při provádění častých změn [8][23].

- Metriky pro procesy

Podle Pfleegera a Atleeho [22] je time-to-market doba potřebná k vytvoření minimální životaschopné verze (MVP).

Defekty na jednotku kódu: měřítko spolehlivosti a kalibru konečného řešení [2][25].

Údržbové úsilí: množství času a peněz použitých na opravy chyb, refaktoring a přidávání nových funkcí [6][21].

- Rozsah experimentu

Předpokladem objektivního výsledku je srovnatelný stupeň dovedností vývojových týmů a rovnocenná funkčnost zkoumaných aplikací [15][24]. Aby nedošlo ke zkreslení výsledků měření, je také vhodné omezit sběr dalších závislostí (například frontendových knihoven nebo ORM nástrojů na serveru) [9].

4.1.2 Nástroje pro měření a pozorování

- Systémy pro sledování úkolů a času

Údaje o množství času stráveného jednotlivými kroky vývoje lze shromažďovat pomocí sledovacích systémů (Jira, Redmine) nebo jednoduše pomocí základních tabulek [22].

- Sledování výkonu

Údaje o době odezvy, využití procesoru a paměti poskytují nástroje pro logování (ELK stack) a nástroje APM (New Relic, AppDynamics) [10][20].

- Systémy pro správu verzí

Sledování četnosti revizí, ovlivněných souborů a časování slučování větví je možné pomocí repozitáře Git [8][23].

4.1.3 Experimentální design: vývoj dvou verzí aplikace

V typickém experimentu jsou provedeny dvě iterace stejné webové aplikace:

- Nejsou použity žádné rámce.

Přední část používá HTML, CSS a čistý JavaScript, zatímco zadní část používá servlety nebo minimální strukturu.

Směrování, validace formulářů a zabezpečení jsou infrastrukturní práce, které se provádějí „ručně“, což zvyšuje flexibilitu, ale trvá déle, než se použijí standardní vzory [2][21].

- Využití rámců

Specifické frameworky na straně frontendu (React, Angular, Vue) a serveru (např. Micronaut, Spring Boot) [8][15].

Vestavěným nástrojům (směrovače, validace, DI kontejnery) jsou přidělovány rutinní povinnosti, které mohou urychlit vývoj a zlepšit strukturu kódu [6].

Aby bylo zajištěno, že srovnávací hodnocení bude v obou situacích založeno na stejných kritériích, měly by se základní funkce (operace CRUD, ověřování a základní návrh) překrývat [9][24].

4.2 Výhody a nevýhody vývoje s frameworky

Webové frameworky, jako je Angular, React, Spring a další, slouží jako rámec, který usnadňuje běžné úlohy při vývoji webových aplikací. Mnoho všedních úkolů (jako je správa stavů, směrování, DI atd.) lze s jejich pomocí provést rychleji a jednodušeji [21][25]. Kromě zjevných výhod má však používání frameworků i řadu nevýhod, které je třeba při volbě vývojové strategie vzít v úvahu.

4.2.1 Výhody

- Zkrácení doby uvedení na trh (nebo vývoje)

Minimální životaschopná verze (MVP) může být při práci s frameworky často dokončena rychleji, protože předpřipravené moduly a vestavěné nástroje minimalizují množství „ruční“ implementace [6][21]. Rychlost uvedení na trh je klíčová v silně konkurenčním prostředí, proto je to obzvláště důležité [9].

- Osvědčené postupy a standardizovaný kód

Většina současných frameworků používá osvědčené vzory (MVC/MVVM, Inversion of Control, návrhové vzory), které podporují čitelnost a přehlednost kódu [2][21]. Používání standardních technik podporuje týmovou práci, zvyšuje udržitelnost a kvalitu výsledného produktu [23].

- Rozsáhlý ekosystém a pomoc

Znamé frameworky mají živou komunitu, bohatou dokumentaci a množství zásuvných modulů pro řešení různých problémů [8][15]. Tím, že si můžete vybrat předpřipravená řešení z aktuálního ekosystému, se snižuje nebezpečí „znovuvynalézání kola“ a urychluje se proces vývoje [24].

- Integrované nástroje pro ladění a testování

Pro jednotkové a integrační testování nabízí řada frameworků předpřipravené knihovny a techniky [23]. Například frontendové frameworky jako React, Angular a Vue podporují Jest, Karma nebo Cypress, zatímco Spring Boot nabízí Spring Test a snadné propojení s JUnit [2][9].

4.2.2 Nevýhody

- Křivka učení a složitost

Velké frameworky může být obtížné se naučit, zejména pokud jsou „názorově vyhraněné“ a obsahují mnoho nápadů (jako Angular nebo Jakarta EE) [21][22]. Pro podporu velkých projektů je navíc nutná znalost systémů sestavení, hierarchie modulů a sofistikovaných ladicích nástrojů [25].

- Migrace a závislosti na verzích

Týmy jsou občas nuceny průběžně upravovat své projekty, aby se přizpůsobily aktualizovaným rozhraním API, a to kvůli rychlému uvolňování nových verzí frameworků. Může se stát, že je třeba refaktorovat velké množství kódu, aby bylo možné přejít na novou hlavní verzi [6][10]. Nedostatek skutečných knihoven a bezpečnostních oprav však může být důsledkem zpoždění aktualizace [8].

- Režie a nadbytečná funkčnost

Ne všechny funkce frameworku jsou pro daný projekt nutné. Vývojáři občas využijí jen 20-30 % funkcionality, ale za zbývající moduly musí platit, což zvyšuje hmotnost balíčku a ztěžuje optimalizaci [20][24]. V některých situacích by mohlo být smysluplnější použít „mikrorámce“ nebo kolekce samostatných knihoven [9].

- Závislost na vnějším světě

Velké komunity nebo korporace (jako Google a Facebook) obvykle podporují stabilní rámce [2]. Může však být náročné rozvíjet a udržovat projekt, pokud je závislý na vysoce specializovaném rámci s malou dlouhodobou údržbou [22].

4.3 Praktické příklady a případové studie z reálného prostředí

Výzkumníci a podniky často provádějí empirické studie a zkoumají předchozí úspěšné (nebo neúspěšné) projekty, aby mohli posoudit skutečnou účinnost použití rámců při vývoji webových stránek a porovnat je s vytvářením aplikací od nuly [22][25]. Zde uvádíme několik reprezentativních příkladů ze skutečného prostředí, které ukazují výhody a nevýhody jednotlivých strategií.

4.3.1 Případová studie: Migrace na architekturu mikroslužeb pomocí Spring Boot

Monolitický program představoval pro společnost budující silně zatížený webový portál výzvu, která ztěžovala rozšiřování a přidávání nových funkcí. Řešením bylo postupné rozdělení služby na mikroslužby a pro rychlý vývoj a jednoduchou integraci byl zvolen Spring Boot[9][24].

- Zjištění

Zkrátila se doba uvedení na trh: hotové komponenty (Spring Data, Spring Security) umožnily rychlejší nasazení nových modulů.

Snadná škálovatelnost: podle Kleppmanna[20] mohla být každá mikroslužba spuštěna samostatně a upravena podle pracovní zátěže.

Zjednodušená struktura: zdůvodnění aktualizací a nasazení bylo jednodušší a projekt získal větší podporu [10].

- Omezení:

Složitost konfigurace: naučit se Spring Cloud a integrovat nové služby vyžaduje čas, i když mnoho nastavení bylo provedeno „standardně“ prostřednictvím Spring Boot [8].

Migrace verzí: Při přechodu na nové hlavní verze je třeba věnovat větší pozornost kvůli rychlému vývoji Spring [6].

4.3.2 Případová studie: „vanilla JS“ vs. React frontend development

Vývojový tým vytvořil v rámci výukového experimentu dvě aplikace SPA se stejnou funkcí: jedna byla vytvořena pomocí „čistého“ JavaScriptu, zatímco druhá pomocí Reactu [2].

- Zjištění

Rychlost vývoje: Protože opakované použití komponent, správa stavu a směřování zjednodušily psaní kódu, byl projekt React dokončen o 20 % rychleji [15].

Kvalita a udržitelnost: Podle Becka [23] se verze React snáze přepracovávala a měla přehlednější návrh.

Výkonnost: React byl v roce 2015 schopen vytvořit několik nových aplikací, např.: React překonal „vanilla“ verzi z hlediska optimalizace překreslování, zatímco „vanilla“ řešení vykazovalo podobnou rychlost vykreslování s kompetentní technikou [21].

Původní verze bez frameworku byla sice pro začátečníky jednodušší, ale napsat standardní logiku (směřování, změny DOM) vyžadovalo mnohem více práce.

Druhá verze na platformě React sice nabízela praktičtější framework a řadu užitečných nástrojů (např. testování a sestavování), ale k uspořádání architektury vyžadovala předchozí znalosti JSX a dalších knihoven [20].

4.3.3 Ukázka, jak začlenit Vue do již existujícího projektu

Šablony JSP na straně serveru a trocha JavaScriptu na straně klienta byly již základem velkého online obchodu. Frontendová část se rozhodla postupně přejít na Vue.js a zároveň rozšiřovat funkcionalitu, přičemž se začalo s „ostrůvky“ funkcionality [17].

- Zjištění

Postupná migrace: Lewis a Fowler [24] poznamenali, že komponenty Vue (například formuláře a produktové filtry) byly nasazovány postupně, čímž se zabránilo zásahům do základní logiky a nebylo nutné provádět globální úpravy návrhu.

Zlepšení UX: Vývojáři zjistili, že je jednodušší opakovaně používat komponenty a testovat je odděleně, a web se stal citlivějším [2].

Minimální rizika: tým vzdělával zaměstnance v používání Vue a vyhodnocoval jeho účinnost, aniž by zcela „rozbil“ monolit [22].

- Omezení

Dvojitá logika vykreslování: bylo náročné koordinovat směřování a aktualizovat celkový stav, protože některé stránky zůstaly ve starém JSP a některé ve Vue [25].

Dodatečná režie: údržba infrastruktury pro sestavení vyžaduje více práce při sloučení obou metodik [8].

Empirické důkazy ukazují, že frameworky poskytují pozorovatelné výhody díky lepší škálovatelnosti, rychlejšímu vývoji a přehlednější architektuře kódu [9][20]. Při rozhodování o výběru frameworku nebo technologie je třeba zohlednit specifika projektu, míru zkušeností týmu a dlouhodobé záměry vývoje systému [21][22]. Postupná migrace je často ideálním přístupem, protože snižuje rizika a poskytuje možnost pozorovat skutečné přínosy frameworku na „živých“ úlohách [2][15].

II PRAKTICKÁ ČÁST

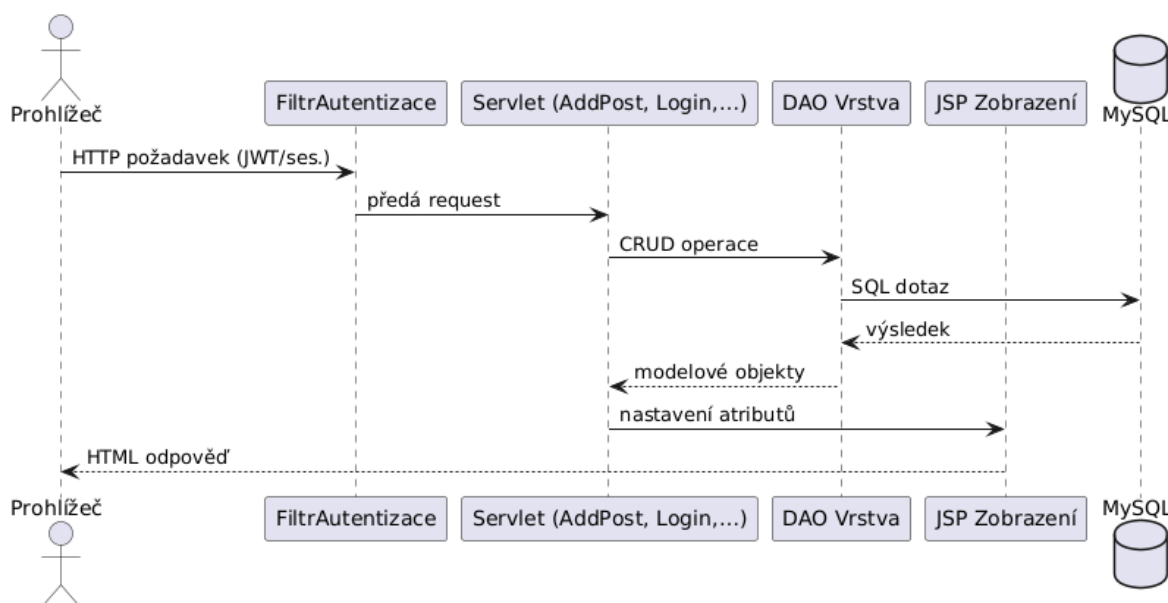
5 UVOD DO PRAKTICKE CASTI

Cílem této praktické části je experimentálně porovnat dva přístupy k vývoji webových stránek na příkladu realizace výukové a demonstrační webové aplikace MiniBlog. První přístup (No Framework) představuje klasickou monolitickou aplikaci vyvinutou pomocí základních technologií Java Servlets, JSP a JDBC bez zapojení moderních frameworků. Druhý přístup využívá populární stack technologií - Spring Boot pro serverovou stranu a Angular pro klientskou stranu aplikace. Provedený výzkum je zaměřen na ověření závěrů učiněných v teoretické části práce, konkrétně - na zjištění vlivu moderních frameworků na efektivitu, složitost a pracnost vývoje webových aplikací.

6 ARCHITEKTURA A TECHNOLOGIE

V této části jsou uvedena klíčová architektonická rozhodnutí a použité technologické balíčky obou implementací MiniBlogu: klasické monolitické aplikace založené na čistých Java Servletech, JSP a JDBC (bez frameworku) i full-stack řešení na Spring Boot 3 a Angular 17. Nejprve je popsána struktura a součásti obou řešení — od přijetí HTTP požadavku až po interakci s databází — a následně je provedeno přímé srovnání podle hlavních kritérií: organizace kontrolérů, přístupu k datům, mechaniky ověřování a návrhu uživatelského rozhraní. Tímto přístupem je umožněno rozumné vyhodnocení dopadu volby technologického stacku na použitelnost, rozšiřitelnost a výkon aplikace

6.1 Architektura bez rámce



Obrázek 11 - Schéma toku zpracování HTTP požadavků v monolitické verzi MiniBlogu na čistých servletech + JSP + JDBC

Součásti systému

1. Prohlížeč

- Klient, který odesílá požadavky HTTP (např. GET /add-post, POST /login) a přijímá stránky HTML.

2. AuthFilter (AuthFilter)

- Platí pro všechny adresy URL (/*), zachytí každý požadavek.

- Kontroluje přítomnost a platnost tokenu JWT v hlavičce Authorisation. Pokud je token nesprávný nebo chybí, pokusí se obnovit uživatele z relace HTTP.
- Pokud je ověření úspěšné, uloží objekt User do atributu požadavku; v opačném případě přesměruje na přihlašovací stránku (pro GET) nebo vrátí 401 Unauthorised (pro ostatní metody).

3. Servlety (AddPostServlet, LoginServlet atd.).

- Pro každý scénář použití (vytvoření příspěvku, přihlášení, registrace, přidání komentáře atd.) je implementován samostatný servlet.
- Metody doGet/doPost zpracovávají vstupní data, volají metody DAO a nastavují atributy požadavků, které se předávají do JSP.

4. Vrstva DAO

- Třídy UserDAO, PostDAO, CommentDAO atd. provádějí CRUD operace nad databází pomocí JDBC.
- Každá metoda otevře připojení, vytvoří příkaz PreparedStatement, provede dotaz SQL a převede soubor ResultSet na objekty POJO.

5. MySQL

- Relační databáze s tabulkami users, posts, comments, categories, tags a propojovacími tabulkami post_tag, post_category.
- DAO přímo provádí dotazy SQL prostřednictvím JDBC.

6. Zobrazení JSP

- Stránky JSP (listPosts.jsp, form.jsp, profile.jsp atd.) spolu s JSTL generují HTML.
- Pro cykly se používá <c:forEach> a pro bezpečný textový výstup <c:out>.

Životní cyklus požadavku HTTP

1. Přijetí požadavku

- Prohlížeč odešle aplikaci požadavek (např. GET /add-post).

2. Ověřování

- Filtř AuthFilter zachytí požadavek, ověří token JWT nebo relaci a rozhodne, zda jej předá dál.

3. Zpracování servletů

- Příslušný servlet načte parametry (title, content, categoryIds), provede ověření a zavolá metody DAO.

4. Přístup k databázi

- Vrstva DAO provádí operace SQL (INSERT, SELECT atd.) a vrací objekty Java.

5. Příprava dat pro zobrazení

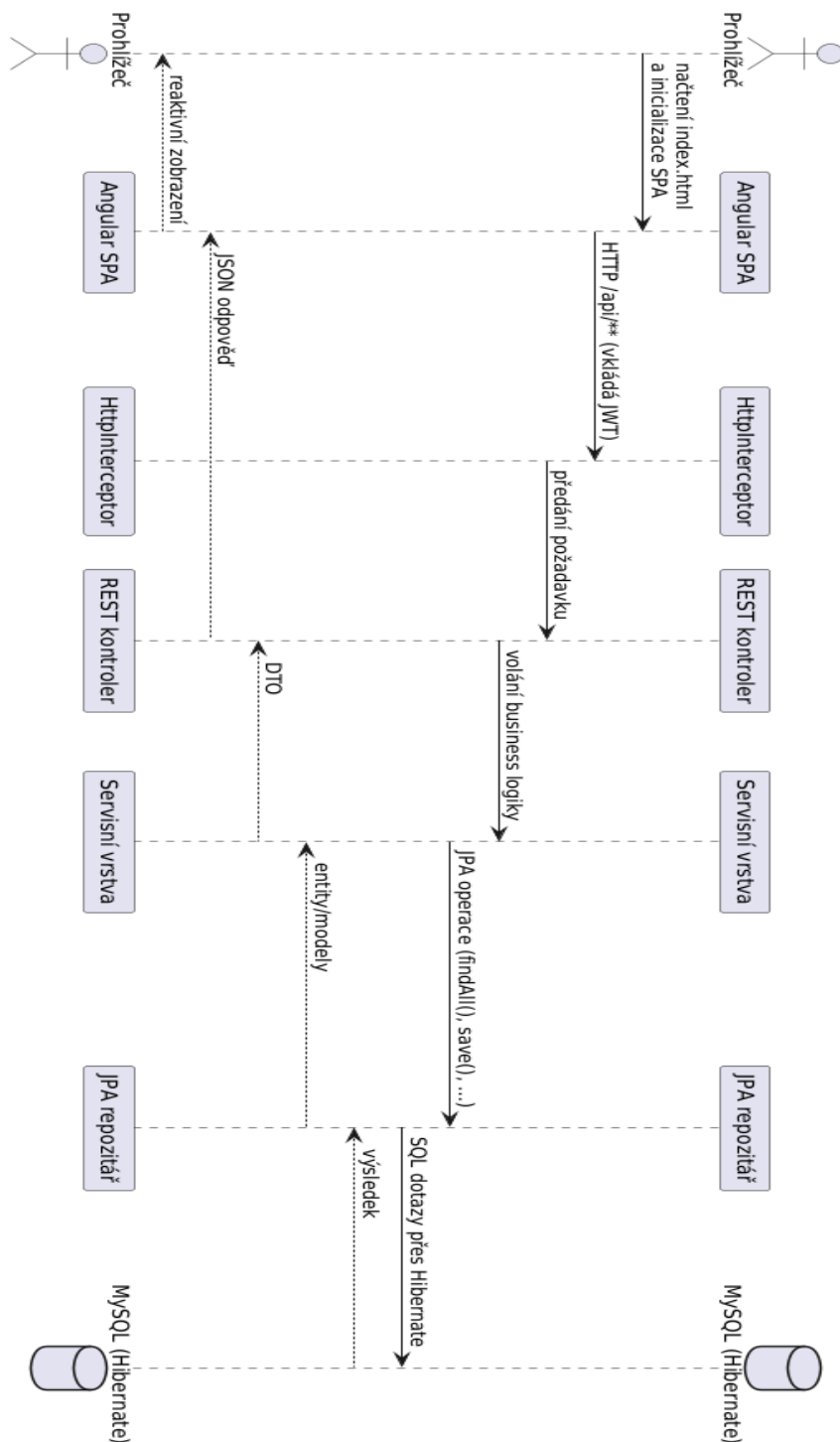
- Servlet uloží objekty (List<Post>, User atd.) do atributů požadavku a provede předání do JSP.

6. Generování HTML

- Stránka JSP vygeneruje odpověď HTML pomocí JSTL a odešle ji klientovi.

Tento klasický přístup poskytuje plnou kontrolu nad každým krokem zpracování, ale vyžaduje značné množství opakovaného kódu a ruční konfiguraci každého nového servletu, filtru a DAO.

6.2 Architektura Spring Boot + Angular



Obrázek 12 - Architektura full-stack řešení MiniBlog Spring Boot + Angular

Popis složky

1. Prohlížeč

- Načte soubor index.html a poté inicializuje aplikaci Angular (načte balíček JavaScript/CSS).

2. Angular SPA

- Jednostránková aplikace s komponentami, směrováním a službami pro práci s rozhraními API.

3. HttpInterceptor

- Zachytí všechny požadavky HTTP na backend (/api/**) a do hlavičky automaticky připojí Authorisation: Bearer <JWT>.

4. Řadiče REST

- Třídy s anotacemi @RestController (např. AuthController, PostController), přijímají a zpracovávají požadavky JSON.

5. Servisní vrstva

- Koše s anotací @Service, ve kterých je implementována obchodní logika a transformace dat mezi kontroléry a úložišti.

6. Úložiště JPA

- Rozhraní rozšiřující JpaRepository automaticky poskytují metody CRUD a umožňují deklarovat složité dotazy prostřednictvím @Query.

7. MySQL (prostřednictvím Hibernate)

- Relační databáze, ke které repozitáře JPA přistupují prostřednictvím Hibernate a fondu připojení HikariCP.

Životní cyklus požadavků HTTP

1. Požadavek na přihlášení

- Angular odešle POST /api/auth/login s uživatelskými údaji.

AuthController ověří přihlašovací údaje, vygeneruje JWT a vrátí jej ve formátu JSON.

2. Chráněné požadavky

- Všechny následující požadavky (GET /api/posts, POST /api/posts) procházejí přes HttpInterceptor a JwtFilter (v řetězci Spring Security).

- Pokud je ověření JWT úspěšné, je požadavek předán kontroléru.
3. Zpracování v kontroléru
 - Kontrolér přijme DTO, zavolá službu a předá data do úložiště.
 4. Přístup k datům
 - Úložiště provede operaci SQL a vrátí objekty entit.
 5. Generování odpovědi
 - Služba převede entitu na DTO, kontrolér vrátí JSON - a Angular aktualizuje uživatelské rozhraní bez opětovného načtení stránky.

Toto rozdělení do vrstev zajišťuje vysokou rozšiřitelnost a udržitelnost kódu: pro přidání nového koncového bodu API stačí vytvořit metodu v kontroléru a úložišti a dynamické klientské uživatelské rozhraní lze rozšířit vytvořením nové komponenty Angular prostřednictvím CLI.

6.3 Srovnání architektur

Aspekt	No Framework	Spring Boot + Angular
MVC tok	Servlet → DAO → JSP	REST Controller → Service → JPA Repository + SPA (Angular)
Konfigurace	web.xml + init() v servletu a filtru	Autokonfigurace Spring, application.yml, DI-kontejner
Řadiče (kontrolery)	Samostatný Servlet pro každý use-case	@RestController s metodami pro různé URL
Přístup k datům	JDBC v DAO třídách manuálně	Spring Data JPA (JpaRepository)
Autentizace	Vlastní AuthFilter	Spring Security + JwtFilter, CORS a CSRF „out-of-the-box“
Prezentace (view)	Serverové JSP + JSTL	Angular SPA, komunikace přes JSON/REST
Rozšiřitelnost	Nové servlety a úpravy JSP pro každou funkci	Deklarativní nové endpointy a komponenty (CLI, anotace)
Škálovatelnost	Horizontální škálování Tomcat	Kontejnerizace (Docker, Kubernetes), Spring profily

Tabulka 1 - Srovnání architektur

Srovnání Komentáře:

V případě „bez frameworku“ vyžaduje každý nový scénář vytvoření vlastního servletu a souvisejících JSP/DAO, což zvyšuje množství kotelního kódu a propojení modulů.

Ve Spring Boot + Angular je většina infrastruktury (směrování, zabezpečení, přístup k datům) konfigurována deklarativně a automaticky, což omezuje rutinní práci a usnadňuje údržbu.

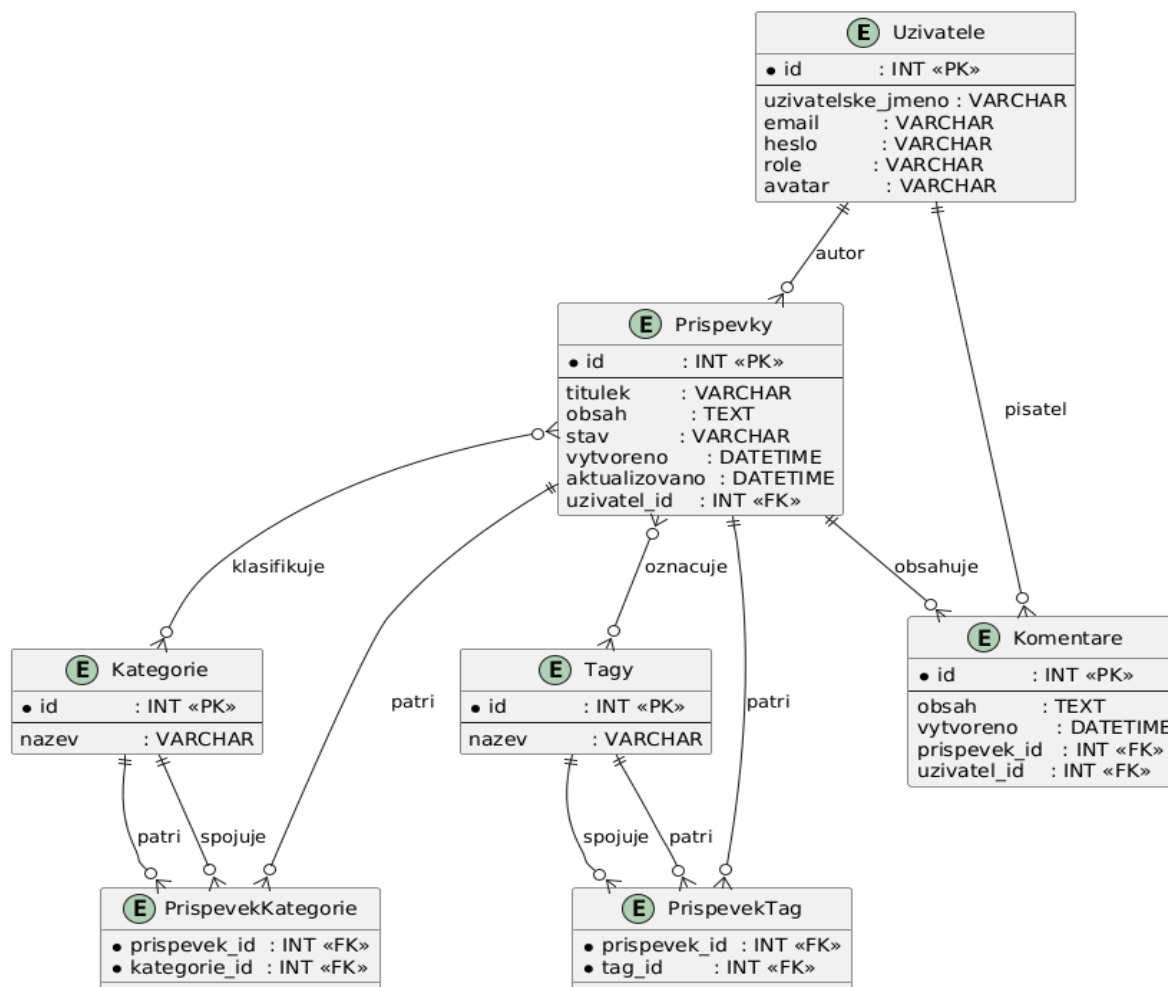
Frontend SPA v Angularu je oddělen od backendu, což umožňuje nezávislé nasazení a škálování klientské části, a architektura REST zjednodušuje integraci s mobilními nebo jinými aplikacemi.

7 DATOVÝ MODEL A PŘÍSTUP K DATŮM

Tato kapitola podrobně popisuje schémata ukládání a správy dat ve dvou implementacích MiniBlogu: klasické monolitické verzi bez frameworku a full-stack řešení na platformě Spring Boot + Angular. Nejdříve je představen ER-diagram relační databáze MySQL a popis jednotlivých tabulek v variantě No Framework, včetně vazeb one-to-many a many-to-many. Následuje model entit JPA s příslušnými anotacemi (`@Entity`, `@OneToMany`, `@ManyToMany`) ve Spring Boot projektu, přičemž je zdůrazněno využití druhé úrovně mezipaměti. Na závěr se porovnávají dva přístupy k přístupu k datům: ručně psané vrstvy DAO nad JDBC versus deklarativní repozitáře Spring Data JPA, přičemž jsou ukázány rozdíly v objemu a složitosti kódu, v podpoře transakcí a v možnostech cachování. Toto srovnání odhaluje, do jaké míry frameworky zjednodušují vývoj a údržbu datové vrstvy.

7.1 Podrobnosti o modelu dat a přístupu ve variantě bez rámce

Tato podkapitola podrobně ukazuje, jak „čistá“ verze MiniBlogu (bez použití frameworku Spring nebo JPA) ukládá a manipuluje s informacemi. Cílem je vysledovat celou cestu dat od dotazu SQL k objektu Java a odhadnout množství ručního kódu potřebného při tomto přístupu.



Obrázek 13 - ER diagram datového modelu MiniBlogu (No Framework)

Diagram odráží strukturu relační databáze blogu a hlavní vztahy mezi entitami.

1. Uzivatele (users) - tabulka účtů. Pole: id (PK), login, e-mail, BCrypt-hash hesla, role, odkaz na avatar.

Vztahy:

- 1 → N s Prispevky - jeden uživatel může vytvořit více příspěvků (autor).
 - 1 → N s Komentare - jeden uživatel může zanechat mnoho komentářů (pisatel).
2. Prispevky (příspěvky) - příspěvky na blogu. Pole: id, název, obsah, stav zveřejnění, čas vytvoření a aktualizace, uzivatel_id (FK → Uzivatele).

Odkazy:

- 1 → N s Komentare - příspěvek obsahuje více komentářů (obsahuje).
- M ↔ N s Kategorie prostřednictvím vazby PrispivekKategorie (klasifikuje).

- $M \leftrightarrow N$ s Tagy prostřednictvím vazby PrispevekTag (označuje).
- 3. Komentare (komentáře) - komentáře k příspěvkům. Pole: id, text, čas vytvoření, FK na příspěvek a uživatele.
- 4. Kategorie (kategorie) - adresář rubrik; minimální pole: id, nazev.
- Propojení $M \leftrightarrow N$ s Prispevky je realizováno tabulkou PrispevekKategorie.
- 5. Tagy (tagy) - adresář tagů; pole id, nazev.
- Propojení $M \leftrightarrow N$ s Prispevky prostřednictvím PrispevekTag.
- 6. PrispevekKategorie a PrispevekTag jsou spojovací tabulky „post-kategorie“ a „post-tag“ obsahující po dvou cizích klíčích (post_id, category_id / tag_id).

Schéma tedy podporuje:

- uživatel-autor role,
- publikování příspěvků s více kategoriemi a značkami,
- komentáře ke každému příspěvku,
- možnost rychlého výběru příspěvků podle kategorií, značek nebo autorů prostřednictvím mezitabulky.

```
• public void addPost(Post post) {  
    String sql = "INSERT INTO posts (title, content, user_id)  
VALUES (?, ?, ?)";  
    try (Connection conn = DBUtil.getConnection();  
        PreparedStatement stmt = conn.prepareStatement(sql,  
Statement.RETURN_GENERATED_KEYS)) {  
        stmt.setString(1, post.getTitle());  
        stmt.setString(2, post.getContent());  
        stmt.setInt(3, post.getUserId());  
        stmt.executeUpdate();  
        try (ResultSet keys = stmt.getGeneratedKeys()) {  
            if (keys.next()) {  
                post.setId(keys.getInt(1));  
            }  
        }  
    } catch (SQLException e) {  
        throw new RuntimeException(e);  
    }  
}
```

Ukázka kódu 1 - INSERT post (DAO → JDBC)

Metoda demonstruje klasický přístup JDBC: připravený dotaz s parametry, executeUpdate(), získání ID s automatickým nárůstem pomocí RETURN_GENERATED_KEYS. Transakce

je řízena ručně a vztahy post ↔ category/tag jsou přidány v jiných DAO, takže celá operace není atomická.

```
public User getUserByUsername(String username) {
    String sql = "SELECT id, username, email, password, role, avatar
FROM users WHERE username = ?";
    try (Connection conn = DBUtil.getConnection();
        PreparedStatement stmt = conn.prepareStatement(sql)) {

        stmt.setString(1, username);
        try (ResultSet rs = stmt.executeQuery()) {
            if (rs.next()) {
                User user = new User();
                user.setId(rs.getInt("id"));
                user.setUsername(rs.getString("username"));
                user.setEmail(rs.getString("email"));
                user.setPassword(rs.getString("password"));
                user.setRole(rs.getString("role"));
                user.setAvatar(rs.getString("avatar"));
                return user;
            }
        }
    } catch (SQLException e) {
        throw new RuntimeException("Error fetching user by username",
e);
    }
    return null;
}
```

Ukázka kódu 2 - SELECT uživatele (ruční mapování ResultSet)

Metoda ilustruje ruční VÝBĚR:

- Nahrazení parametrů pomocí PreparedStatement chrání před SQL injection.
- Mapování jednotlivých sloupců na pole objektu User se provádí ručně; při přidání sloupce do tabulky je třeba aktualizovat kód ve všech DAO.
- Výjimky vrstvy JDBC jsou zabaleny do neověřené výjimky RuntimeException, což zjednodušuje signatury metod, ale skrývá typy chyb SQL na vyšších vrstvách.

```
public void addComment(Comment comment) {
    String sql = "INSERT INTO comments (post_id, user_id, content)
VALUES (?, ?, ?)";
    try (Connection conn = DBUtil.getConnection();
        PreparedStatement stmt = conn.prepareStatement(sql,
Statement.RETURN_GENERATED_KEYS)) {
        stmt.setInt(1, comment.getPostId());
        stmt.setInt(2, comment.getUserId());
        stmt.setString(3, comment.getContent());
        int affectedRows = stmt.executeUpdate();
        System.out.println("Comments inserted: " + affectedRows);
        try (ResultSet generatedKeys = stmt.getGeneratedKeys()) {
            if (generatedKeys.next()) {
                comment.setId(generatedKeys.getInt(1));
            }
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }
}
```

Ukázka kódu 3 - INSERT komentáře

Metoda ukazuje typický INSERT pro propojenou entitu:

- Všechny zdroje (Connection, PreparedStatement, ResultSet) jsou uzavřeny bloky try-with-resources.
- Získání vygenerovaného primárního klíče je nutné, pokud potřebujete klientovi vrátit nový komentář nebo jej v budoucnu například propojit s jinými entitami.
- Validace obsahu komentáře a kontrola práv uživatele neprobíhá zde, ale ve volajícím servletu, takže DAO zůstává „tenké“ a je zodpovědné pouze za práci s databází.

```
@Override
protected void doPost(HttpServletRequest req, HttpServletResponse resp)
    throws ServletException, IOException {
    req.setCharacterEncoding("UTF-8");

    /* 1)  Ověření uživatele přes JWT nebo session */
    User user = null; // vlastní metoda
    String header = req.getHeader("Authorization");
    if (header != null && header.startsWith("Bearer ")) {
        try {
            String username =
                JwtUtil.validateToken(header.substring(7));
            user = userDao.getUserByUsername(username);
        } catch (JwtException ignored) { }
    }
    if (user == null) {
        HttpSession session = req.getSession(false);
        if (session != null) {
            user = (User) session.getAttribute("user");
        }
    }
    if (user == null) {
        resp.sendRedirect(req.getContextPath() + "/login.jsp");
        return;
    }

    /* 2)  Vytvoření samotného příspěvku */
    String title = req.getParameter("title");
    String content = req.getParameter("content");
    Post post = new Post();
    post.setTitle(title);
    post.setContent(content);
    post.setUserId(user.getId());
    postDAO.addPost(post);

    int postId = post.getId();

    /* 3)  Uložení kategorií (M:N) */
    String[] catIds = req.getParameterValues("categoryIds");
    if (catIds != null) {
        for (String cid : catIds) {
            pcDAO.addMapping(postId, Integer.parseInt(cid));
        }
    }

    /* 4)  Uložení tagů (M:N) */
    String[] tagIds = req.getParameterValues("tagIds");
    if (tagIds != null) {
        for (String tid : tagIds) {
            ptDAO.addMapping(postId, Integer.parseInt(tid));
        }
    }

    /* 5)  Redirect na seznam příspěvků */
    resp.sendRedirect(req.getContextPath() + "/posts");
}
```

Ukázka kódu 4 - Obchodní logika vytvoření příspěvku + vazby M:N

V jedné metodě servletu jsou soustředěny všechny kroky obchodního procesu:

- Autentizace - ruční ověření JWT a relace HTTP.
- Vytvoření příspěvku - volání `PostDAO.addPost()`, které otevře vlastní transakci.
- Přidání odkazů M:N - samostatné metody DAO vytvoří záznamy v `post_category` a `post_tag`. Tyto operace se provádějí mimo transakci vložení příspěvku, takže v případě chyby může dojít k neúplné konzistenci dat.
- Přesměrování - po úspěšném uložení přejde prohlížeč na feed příspěvku.

Kód jasně ukazuje, že ve variantě bez frameworků je veškerá obchodní logika a práce s několika DAO soustředěna ve vrstvě HTTP, což komplikuje testování a podporu.

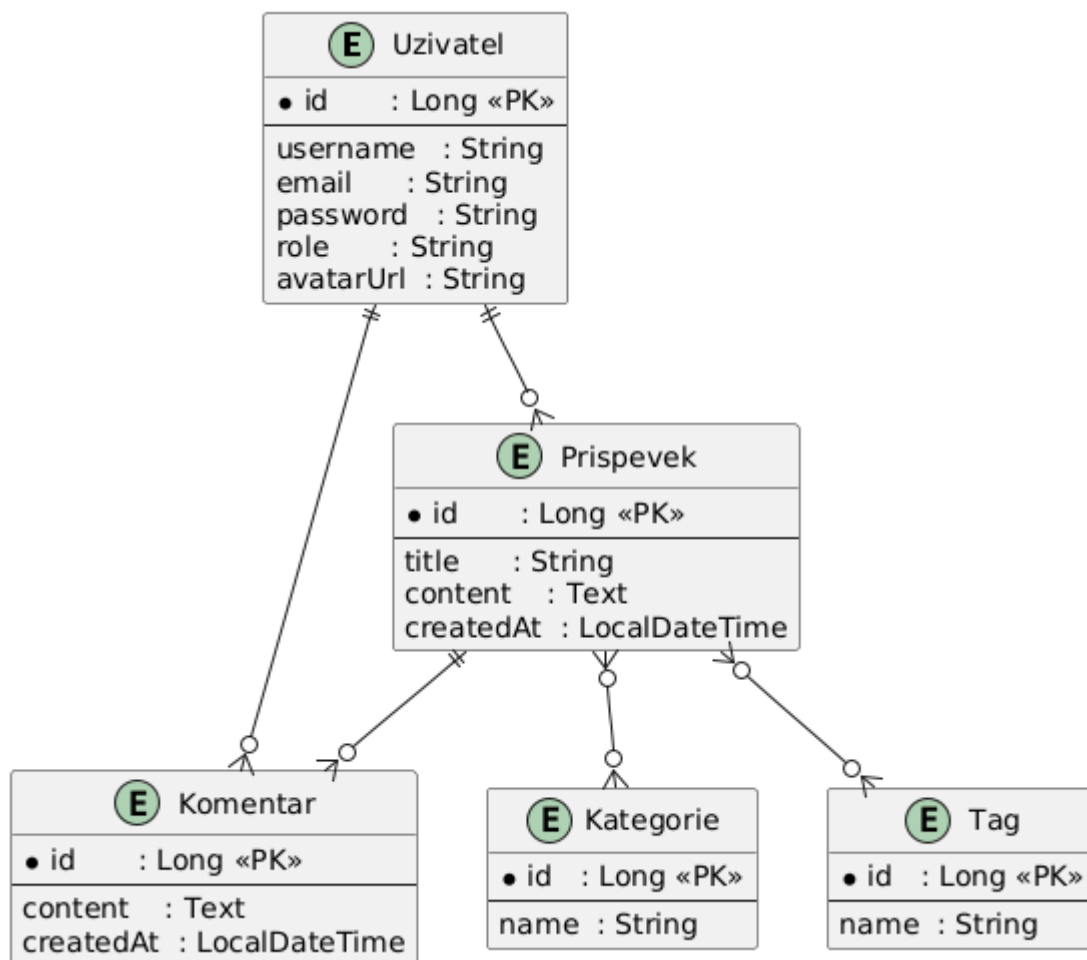
```
public static Connection getConnection() throws SQLException {
    try {
        // explicitní načtení ovladače - nutné při čistém JDBC
        Class.forName("com.mysql.cj.jdbc.Driver");
    } catch (ClassNotFoundException e) {
        throw new SQLException("MySQL Driver not found", e);
    }
    return DriverManager.getConnection(
        "jdbc:mysql://localhost:3306/miniblog?serverTimezone=UTC",
        "root",
        "....." // heslo k DB hard-code
    );
}
```

Ukázka kódu 5 - Otevření JDBC připojení

Každá metoda DAO volá `DBUtil.getConnection()`, čímž se otevře nové připojení k databázi. Bez connection-poolu (HikariCP, c3p0 atd.) to má za následek vysokou režii nastavování relací TCP a ověřování MySQL při velkém zatížení. Parametry připojení jsou „zašité“ v kódu, což ztěžuje přenos aplikace do jiných prostředí (test, prod).

7.2 Podrobnosti o modelu dat a přístupu ve variantě Spring Boot + Angular

Tato podkapitola ukazuje, jak MiniBlog na platformě Spring Boot 3 a Angular 17 ukládá a manipuluje s daty prostřednictvím Spring Data JPA. Na rozdíl od přístupu JDBC je většina operací svěřena frameworku: entity jsou popsány pomocí anotací, vrstva CRUD je generována automaticky a transakce jsou spravovány deklarativně.



Obrázek 14 - ER-diagram modelu JPA

Hibernate automaticky vytvoří mezitabulky `post_category`, `post_tag` pomocí anotací `@ManyToMany`.

```
@Entity
public class Post {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String title;

    @Lob
    private String content;

    @ManyToOne(fetch = FetchType.EAGER)
    @JoinColumn(name = "user_id", nullable = false)
    private User author;

    private LocalDateTime createdAt = LocalDateTime.now();

    @ManyToMany(fetch = FetchType.LAZY)
    @JoinTable(name = "post_category",
        joinColumns = @JoinColumn(name = "post_id"),
        inverseJoinColumns = @JoinColumn(name = "category_id"))
    private Set<Category> categories = new HashSet<>();

    @ManyToMany(fetch = FetchType.LAZY)
    @JoinTable(name = "post_tag",
        joinColumns = @JoinColumn(name = "post_id"),
        inverseJoinColumns = @JoinColumn(name = "tag_id"))
    private Set<Tag> tags = new HashSet<>();
}
```

Ukázka kódu 6 - Podstata příspěvku

@Entity + @GeneratedValue úleva od zápisu SQL-INSERT a získání klíče.

@ManyToMany s @JoinTable popisuje vztahy M:N - Hibernate vytváří tabulky a spravuje je automaticky.

Pole createdAt se vyplňuje bez vrstvy DAO; validaci lze přesunout do Bean Validation (@NotBlank).

```
public interface PostRepository extends JpaRepository<Post, Long> {

    List<Post> findAllOrderByCreatedAtDesc();

    List<Post> findByAuthorIdOrderByCreatedAtDesc(Long userId);

    @Query("""
        SELECT DISTINCT p FROM Post p
        LEFT JOIN p.categories c
        LEFT JOIN p.tags t
        WHERE (:q IS NULL
            OR LOWER(p.title) LIKE LOWER(CONCAT('%',:q,'%'))
            OR LOWER(p.content) LIKE LOWER(CONCAT('%',:q,'%')))
        AND (:categoryId IS NULL OR c.id = :categoryId)
        AND (:tagId IS NULL OR t.id = :tagId)
        ORDER BY p.createdAt DESC
        """)
    List<Post> findFiltered(String q, Long categoryId, Long tagId);
}
```

Ukázka kódu 7 - PostRepository

Dědictvím od `JpaRepository` získáte okamžitě hotové metody CRUD (save, findById, delete).

Metoda `findFiltered` demonstruje dotaz JPQL s předanými parametry; složité JOINy jsou popsány deklarativně, bez ručních `ResultSet`ů.

```

@Service
public class PostServiceImpl implements PostService {

    private final PostRepository postRepo;
    private final CategoryRepository catRepo;
    private final TagRepository tagRepo;
    private final UserRepository userRepo;

    @Transactional // atomická operace
    public Post createPost(CreatePostRequest req) {
        String username = SecurityContextHolder.getContext()
            .getAuthentication().getName();
        User author = userRepo.findByUsername(username)
            .orElseThrow(() -> new RuntimeException("User not
found"));

        Post post = new Post();
        post.setTitle(req.getTitle());
        post.setContent(req.getContent());
        post.setAuthor(author);
        post.setCategories(new
HashSet<>(catRepo.findAllById(req.getCategoryIds())));
        post.setTags (new
HashSet<>(tagRepo.findAllById(req.getTagIds())));

        return postRepo.save(post); // Hibernate INSERT + M:N
    }

    @Transactional
    public void deletePost(Long id) {
        Post post = postRepo.findById(id)
            .orElseThrow(() -> new RuntimeException("Post not
found"));
        String current = SecurityContextHolder.getContext()
            .getAuthentication().getName();
        if (!post.getAuthor().getUsername().equals(current))
            throw new RuntimeException("Forbidden");

        postRepo.delete(post);
    }
}

```

Ukázka kódu 8 - PostServiceImpl - vytváření a mazání příspěvku

@Transakce zaručuje atomicitu - jeden rollback vrátí INSERT i vyplnění tabulek M:N.

postRepo.save() volá Hibernate: SQL-INSERT je generován pro příspěvky a řádek v post_category/post_tag.

Ověřování uživatelských práv a filtrování polí je soustředěno ve službě, nikoli ve vrstvě HTTP.

7.3 Výsledky

Kritérium	No Framework (Servlet + JSP + JDBC)	Spring Boot + Angular
Objem kódu	Vysoký: ručně psané DAO, SQL, mapování <code>ResultSet</code> , správa transakcí a připojení.	Nízký: entity s anotacemi, rozhraní <code>JpaRepository</code> , transakce a CRUD generuje framework.
Transparentnost SQL	Úplná – vývojář vidí a optimalizuje každý dotaz.	Částečná – Hibernate SQL generuje; lze psát JPQL nebo nativní SQL podle potřeby.
Transakce	Ruční <code>setAutoCommit(false) / commit / rollback</code> .	Deklarativní <code>@Transactional</code> , řízené kontejnerem Spring.
Vazby M:N	Vyžadují explicitní INSERT do <code>post_category</code> , <code>post_tag</code> .	Popsány anotací <code>@ManyToMany</code> ; Hibernate spravuje spojovací tabulky.
Výkon základu	Bez connection-poolu a cache; každá DAO metoda otevírá nové spojení.	HikariCP + možnost L2 cache (Ehcache); lépe zvládá zátěž.
Testovatelnost	Servlet přímo vytváří DAO – obtížné unit testy.	Jasně vrstvy, injekce závislostí; testy na úrovni služeb a repozitářů.
Křivka učení	Stačí znalost čistého JDBC a Servlet API.	Nutné porozumět Spring, JPA, anotacím; po zvládnutí rychlejší vývoj.
Rozšiřitelnost	Nový use-case = nový servlet, DAO, JSP – hodně boilerplate.	Stačí přidat metodu do kontroleru a komponentu Angular (CLI + anotace).
Flexibilita kontrola	Maximální kontrola nad SQL a prostředky.	Část nízkoúrovňové práce skrytá, ale JPQL i nativní dotazy dostupné.

Tabulka 2 - Srovnání přístupů

Kdy zvolit kterou možnost?

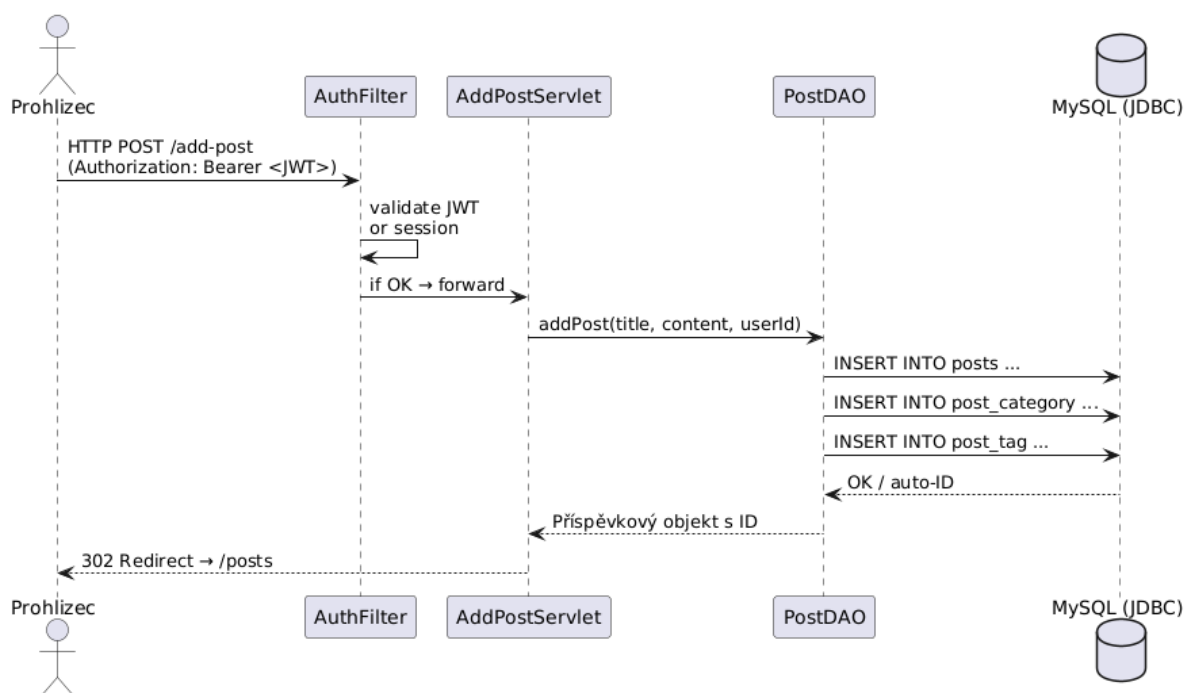
- Žádný rámec není oprávněný, pokud:
 - Projekt je velmi malý, jednorázový nebo se jedná o školení;
 - Je zapotřebí předvídatelná, minimální časová stopa;
 - Tým je zblhlý v čistém JDBC a je důležitá plná kontrola nad SQL.
- Spring Boot + Angular je vhodnější, pokud:
 - Se očekává aktivní vývoj funkcí a tým chce rychle přidávat koncové body;

- Aplikace by se měla škálovat a být nasazena v kontejnerech;
- Potřebujete vestavěné bezpečnostní mechanismy, cache, testování a moderní frontend SPA.

8 TOKY POŽADAVKŮ A ZABEZPEČENÍ

Popisy ukládání dat a přístupu k databázi obou verzí MiniBlogu jsou dokončeny. Následuje analýza průchodu HTTP požadavku jednotlivými aplikačními vrstvami a přehled použitých mechanismů zabezpečení.

8.1 Tok požadavků a zabezpečení v aplikaci MiniBlog bez Frameworku



Obrázek 15 - Tok požadavků v aplikaci MiniBlog No Framework

1. Prohlížeč odešle požadavek POST s JWT v hlavičce Authorisation.
2. AuthFilter
 - Ověří platnost tokenu (JwtUtil.validateToken()),
 - V případě úspěchu vloží User do požadavku a předá řízení servletu,
 - Při neúspěchu vrátí 401 nebo přesměruje na /login.jsp.
3. AddPostServlet načte parametry formuláře, zavolá PostDAO.addPost().
4. PostDAO provádí tři operace:
 - INSERT do příspěvků (získá vygenerované post_id),
 - INSERT do post_category,
 - INSERT do post_tag.

- Transakce je řízena ručně (setAutoCommit(false) → commit()).
5. Po úspěšném vložení servlet provede přesměrování na seznam příspěvků a prohlížeč provede GET /posts.

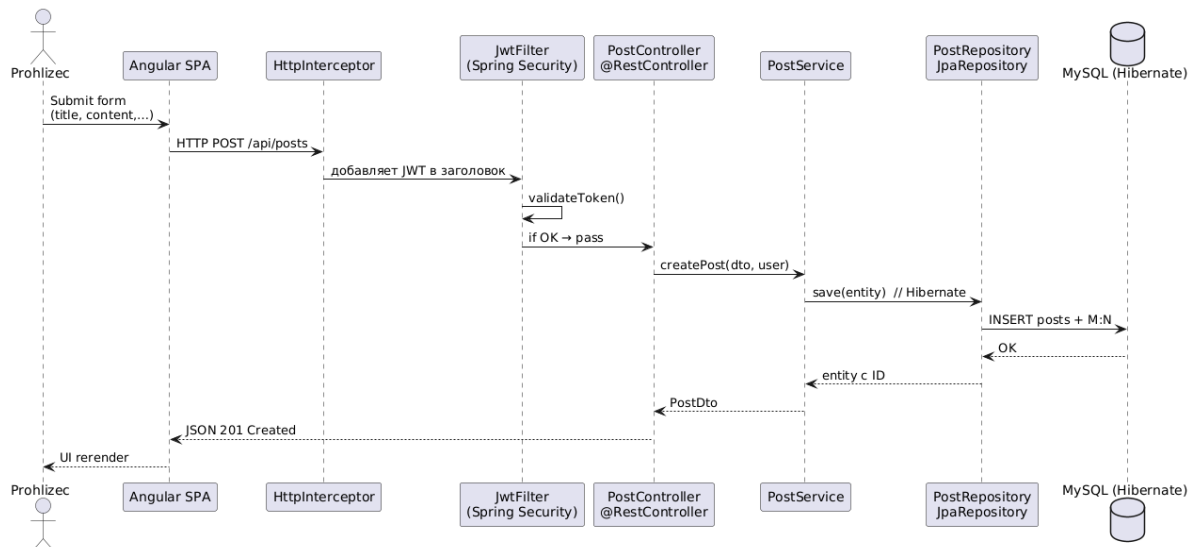
```
String auth = req.getHeader("Authorization");
User user = null;
if (auth != null && auth.startsWith("Bearer ")) {
    try {
        String username = JwtUtil.validateToken(auth.substring(7));
        user = userDao.getUserByUsername(username); // SELECT
    } catch (JwtException ignored) { }
    if (user == null) {
        HttpSession s = ((HttpServletRequest) req).getSession(false);
        if (s != null) user = (User) s.getAttribute("user");
    }
    if (user != null) {
        req.setAttribute("user", user);
        chain.doFilter(req, res); // → servlet
    } else {
        ((HttpServletResponse) res).sendRedirect("/login.jsp");
    }
}
```

Ukázka kódu 9 - Zabezpečení v toku bez frameworku

Mechanismus	Realizace
Ověřování	Kontrola JWT ve filtru; nouzová varianta - relace HTTP.
Autorizace	Role uživatele (user.role) se kontroluje v servletech (např. příspěvek může smazat pouze ADMIN).
Ochrana CSRF	Žádné: formuláře nejsou tokenizovány; riziko CSRF je minimalizováno tím, že nebezpečné operace vyžadují JWT.
CORS	Není třeba - klientská část je generována serverem ve stejné doméně.

Tabulka 3 - Zabezpečení v toku bez frameworku

8.2 Tok požadavků a zabezpečení v aplikaci MiniBlog Spring Boot + Angular



Obrázek 16 - Tok požadavků v aplikaci MiniBlog Spring Boot + Angular

1. Prohlížeč odešle požadavek /api/posts POST s tělem formuláře JSON přes Angular.
2. HttpInterceptor
 - Automaticky vloží hlavičku Authorisation: Bearer <JWT>,
 - Předá požadavek backendu.
3. JwtFilter (Spring Security)
 - Zkontroluje token a vytvoří Authentication,
 - V případě úspěchu předá požadavek kontroléru,
 - V případě neúspěchu vrátí 401 Unauthorised.
4. PostController → PostService (@Transactional)
 - Služba získá aktuálního uživatele, vytvoří entitu Post,
 - Proveďte postRepository.save(post):
 - INSERT na post (získá vygenerované post_id),
 - Automatický INSERT do post_category a post_tag díky @ManyToMany,
 - Celý proces je atomický díky anotaci @Transactional.

5. Po úspěšném uložení kontrolér vrátí 201 Created s JSON reprezentací příspěvku a Angular okamžitě aktualizuje uživatelské rozhraní bez nutnosti znovu načíst stránku.

```
@EnableWebSecurity
public class SecurityConfig {
    @Bean
    SecurityFilterChain chain(HttpSecurity http) throws Exception {
        return http
            .csrf(cs -> cs.disable()) // CSRF off -
            .cors(Customizer.withDefaults())
            //CORSapplication.yml
            .sessionManagement(sm ->
                sm.sessionCreationPolicy(SessionCreationPolicy.STATELESS))
            .authorizeHttpRequests(auth -> auth
                .requestMatchers("/api/auth/**").permitAll()
                .anyRequest().authenticated())
            .addFilterBefore(new JwtFilter(jwtUtil),
                UsernamePasswordAuthenticationFilter.class)
            .build();
    }
}
```

Ukázka kódu 10 - Zabezpečení v toku s použitím frameworku

Mechanismus	Realizace
Ověřování	JwtFilter uvnitř Spring Security; token je kontrolován před kontrolérem.
Autorizace	@PreAuthorize anotace nebo pravidla v SecurityConfig.
CSRF	Rozhraní REST API je zakázáno (csrf.disable()), protože se používá JWT; pokud se v projektu objeví formuláře, můžete jej povolit.
CORS	Povoleno globálně: doména SPA je přeložena v souboru application.yml nebo WebMvcConfigurer.
Transakce	@Transactional u služeb zajišťuje, že všechny operace SQL jsou atomické.

Tabulka 4 - Zabezpečení v toku s použitím frameworku

8.3 Výsledky

Aspekt	MiniBlog bez frameworku	MiniBlog Spring Boot + Angular
Kontrolní bod JWT	Ruční filtr AuthFilter; 20-30 řádků logiky kódu, odpověď 401 nebo přesměrování.	JwtFilter je integrován do řetězce Spring Security; je zapojen pomocí jediného řádku v SecurityConfig.
Úroveň, na které se provádí autorizace	V každém servletu - ruční kontrola role (if (!user.getRole().equals("ADMIN")) ...).	@PreAuthorize anotace, zásady v SecurityConfig; jednotné řídicí centrum.
Transakční integrita vytvoření příspěvku	DAO otevřou připojení, zavolají setAutoCommit(false) a provedou 3 požadavky INSERT; chyba při druhé/třetí operaci vyžaduje ruční vrácení zpět.	Anotace @Transactional na službě: Hibernate provádí všechny INSERTy jako jedinou transakci; při výjimce dojde k automatickému vrácení zpět.
CSRF / CORS	Není implementováno (klient a server ve stejné doméně, formuláře nejsou tokenizovány).	CSRF je pro REST zakázán, ale může být povolen; CORS je spravován deklarativně (application.yml).
Kód průtoku	Jeden filtr + kontroly ve všech servletech + SQL M:N - asi 300 řádků.	Konfigurace Spring Security (≈30 řádků) + 1 JwtFilter (vaše třída), o zbytek se postará framework.
rozšiřitelnost	U nového koncového bodu je třeba zapsat výjimky do filtru nebo duplikovat kontrolní roli ve všech servletech.	Stačí přidat anotaci zabezpečení do kontroléru; filtry a řetězec zůstanou beze změny.

Tabulka 5 - Srovnání přístupů

- V čistě servletové architektuře je za zabezpečení zodpovědný vývojář: musíte psát filtry, ručně kontrolovat role a spravovat transakce. To zvyšuje množství kódu a riziko chyb (např. částečné odevzdání při selhání).
- Stack Spring Boot + Spring Security přesouvá většinu infrastruktury do konfigurace: filtry, CORS, CSRF a model rolí jsou popsány pomocí anotací a souboru application.yml. Sečteno a podtrženo - méně kotelního kódu, snazší údržba a rozšiřování.
- Za tento komfort platíme: - další vrstvou abstrakcí (Hibernate, Spring Security) a vyššími paměťovými nároky. S růstem projektu však zisk v rychlosti vývoje a spolehlivosti značně převyšuje náklady.

9 FRONTEND

Tato kapitola popisuje klientské vrstvy obou implementací MiniBlogu. Nejprve je vysvětleno, jak vykreslování na straně serveru v JSP + JSTL generuje stránky v bezrámcové verzi: struktura adresářů JSP, připojování CSS témat, použití značky `<c:forEach>` a předávání atributů ze servletů do JSP. Následně je představen Angular 17 SPA – strom `src/app`, práce s komponentami, směrování a služby `HttpClient`, tedy způsob, jak frontend prohlížeče volá REST API a aktualizuje rozhraní bez opětovného načítání stránky. Závěrem jsou oba přístupy porovnány z hlediska UX, výkonu, SEO a náročnosti vývoje.

9.1 Frontend JSP

V monolitické verzi MiniBlogu je klientská vrstva tvořena klasickým vykreslováním na straně serveru: data přijatá servletem jsou předávána pomocí atributů požadavku do stránek `.jsp`, kde značky JSTL generují výsledné HTML. Níže jsou uvedeny dva klíčové výpisy (formulář pro přidání příspěvku a seznam příspěvků) a vysvětlivky demonstrující typické techniky JSP frontendu.

```
<form method="post" action="${pageContext.request.contextPath}/add-  
post">  
  <label for="title">Title:</label>  
  <input id="title" name="title" required/>  
  
  <label for="content">Content:</label>  
  <textarea id="content" name="content" rows="5" required></textarea>  
  
  <fieldset>  
    <legend>Categories</legend>  
    <c:forEach var="c" items="${allCategories}">  
      <label><input type="checkbox" name="categoryIds"  
value="${c.id}"/> ${c.name}</label><br/>  
    </c:forEach>  
  </fieldset>  
  
  <fieldset>  
    <legend>Tags</legend>  
    <c:forEach var="t" items="${allTags}">  
      <label><input type="checkbox" name="tagIds"  
value="${t.id}"/> ${t.name}</label><br/>  
    </c:forEach>  
  </fieldset>  
  
  <button type="submit">Submit Post</button>  
</form>
```

Ukázka kódu 11 - form.jsp

Data adresářů `allCategories`, `allTags` jsou předána servletem (`AddPostServlet#doGet`) a vypsána cyklem `<c:forEach>`.

Validaci polí (povinnou) provádí prohlížeč; serverová validace je duplikována v servletu.

Stylování je implementováno pomocí proměnných CSS (`--bg-color`, `--text-color`) - Tlačítko `Toggle Theme` mění třídy a přepíná světlý/tmavý režim bez opětovného načítání.

```
<c:forEach var="p" items="${posts}">
  <div class="post-card">
    <h3><c:out value="${p.title}"/></h3>
    <div class="post-meta">
      ${p.createdAt} | Author: ${p.username}
    </div>

    <div class="post-content"><c:out value="${p.content}"/></div>

    <c:forEach var="c" items="${p.categoryNames}">
      <span class="category-label">${c}</span>
    </c:forEach>

    <a href="edit-post?id=${p.id}">Edit</a> |
    <a href="delete-post?id=${p.id}">Delete</a>
  </div>
</c:forEach>
```

Ukázka kódu 12 - `listPosts.jsp`

Každý požadavek `/posts` je generován v servletu: DAO vrací `List<Post>`, atribut `posts` je předán JSP.

Značka `<c:out>` escapuje znaky HTML, aby se zabránilo XSS.

Stránkování (je-li potřeba) se realizuje přidáním parametru dotazu `?page=` a druhým GET; stránka se vykreslí zcela nově.

9.2 Front-end Angular

V kompletním balíku Spring Boot + Angular je klient vytvořen jako jednostránková aplikace. Angular pouze přijímá JSON z řadičů REST a DOM se vykresluje a aktualizuje v prohlížeči bez opětovného načítání. Níže jsou uvedeny klíčové úryvky kódu a vysvětlivky, které demonstrují, jak jsou implementovány akce vyhledávání, filtrování a autorské akce (Edit/Delete).

```

<div class="search-bar">
  <input [(ngModel)]="searchQuery" placeholder="Search..."
class="search-input">
  <select [(ngModel)]="selectedCategoryId">
    <option [value]="null">All Categories</option>
    <option *ngFor="let c of categories" [value]="c.id">{{ c.name
  }}</option>
  </select>
  <select [(ngModel)]="selectedTagId">
    <option [value]="null">All Tags</option>
    <option *ngFor="let t of tags" [value]="t.id">{{ t.name
  }}</option>
  </select>
  <button (click)="applyFilter()">Filter</button>
  <button (click)="resetFilter()">All Posts</button>
</div>
<div *ngFor="let post of posts; let last = last">
  <div class="post-card">
    <a class="post-title">{{ post.title }}</a>
    <div class="post-meta">{{ post.createdAt | date:'short' }} · {{
post.authorUsername }}</div>
    <div class="post-content">{{ post.content }}</div>

    <span *ngFor="let c of post.categoryNames" class="category-
label">{{ c }}</span>
    <span *ngFor="let t of post.tagNames" class="tag-label">{{
t }}</span>

    <a *ngIf="authService.getUsername() === post.authorUsername"
[routerLink]="['/edit-post', post.id]">Edit</a> |
    <a *ngIf="authService.getUsername() === post.authorUsername"
(click)="deletePost(post.id!)">Delete</a>

    <app-comments [postId]="post.id!"></app-comments>
  </div>
  <hr *ngIf="!last">
</div>

```

Ukázka kódu 13 - posts.component.html

[(ngModel)] poskytuje obousměrné vazby - hodnota pole jde přímo do proměnné searchQuery, selectedCategoryId, selectedTagId.

*ngFor a *ngIf provádějí podmíněné a opakované vykreslování přímo v šabloně, bez serverových smyček.

Odkazy Edit / Delete se zobrazují pouze autorovi příspěvku, validace probíhá prostřednictvím služby AuthService.

<app-comments> je líně načítaná podřízená komponenta, která dostává postId jako Input.

```
getFiltered(q: string,  
  categoryId: number | null,  
  tagId: number | null): Observable<Post[]> {  
  
  let params = new HttpParams();  
  if (q)      params = params.set('q', q);  
  if (categoryId) params = params.set('categoryId', categoryId);  
  if (tagId)   params = params.set('tagId', tagId);  
  
  return this.http.get<Post[]>(this.API, { params });  
}
```

Ukázka kódu 14 - post.service.ts

Do komponenty je vložen Service-singleton; zde jsou soustředěny všechny operace HTTP.

HttpParams generuje řetězec dotazu; Angular jej automaticky serializuje a předá koncovému bodu /api/posts.

Metoda vrací observable<Post[]>, ke kterému se komponenta hlásí: Angular při příjmu nových dat zavolá překreslení DOM.

9.3 Výsledky

Kritérium	JSP + JSTL (vykreslování na straně serveru)	Angular 17 SPA (na straně klienta)
Reakce UX	Každý filtr/kliknutí $\Rightarrow$ plný HTTP oblet a překreslení celé stránky; znatelné zpoždění u "náročného" obsahu.	Překreslují se pouze změněné komponenty; vyhledávání, filtrování a stránkování probíhá okamžitě, bez opětovného načítání.
SEO	Hotové HTML se zobrazí během prvních milisekund - ideální pro vyhledávače a náhledy na sociálních sítích.	Vyžaduje SSR nebo prerender, jinak roboti vidí pouze <code><app-root></app-root></code> a skripty.
Množství kódu klienta	Minimální: trocha CSS + JS utility pro přepínání tématu (< 10 KB).	Angular + RxJS ≈ 200 KB (po gzip), zvláštní paměť v prohlížeči.
Rychlost vývoje uživatelského rozhraní	Každý nový případ použití $\rightarrow$ nový JSP, ruční kopírování částí layoutu, žádné opakované použití logiky.	Generování komponent CLI, nadužívané moduly, šablony; průvodce styly a striktní TypeScript urychlují škálování.
Reaktivita /obousměrné vazby	Ne (pouze berličky JavaScriptu nebo Ajaxu).	Vestavěné <code>[(ngModel)]</code> , RxJS streamy - data a uživatelské rozhraní jsou synchronizovány automaticky.
Zabezpečení na straně klienta	Minimální JS $\rightarrow$ menší plocha pro XSS; pro tokeny CSRF ve formulářích je však nutný samostatný přístup.	Šablony Angularu automaticky escapují HTML; token JWT přidává <code>HttpInterceptor</code> .
Zvládnutí týmu	Stačí základní znalosti servletů a JSTL.	Vyžaduje se znalost TypeScriptu, RxJS, CLI a procesů sestavování.
Dlouhodobá podpora	Při velkém počtu stránek JSP se zvyšuje duplicita a složitost změn.	Přehledná modulární struktura, líná zátěž, testování komponent; změny se lépe škálují.
Výkonnost serveru	Generování HTML zatěžuje procesor Tomcatu; každý požadavek představuje nové vykreslení.	Server poskytuje pouze JSON; vykreslování je přesunuto do prohlížeče, zatížení CPU na backendu je nižší.

Tabulka 6 - Komplexní závěry na frontendu MiniBlogu

JSP je v pořádku, pokud je pro vás prioritou out-of-the-box SEO, šířka pásma a velmi nízký vstupní práh pro tým, a uživatelské rozhraní je jednoduché a zřídka se mění.

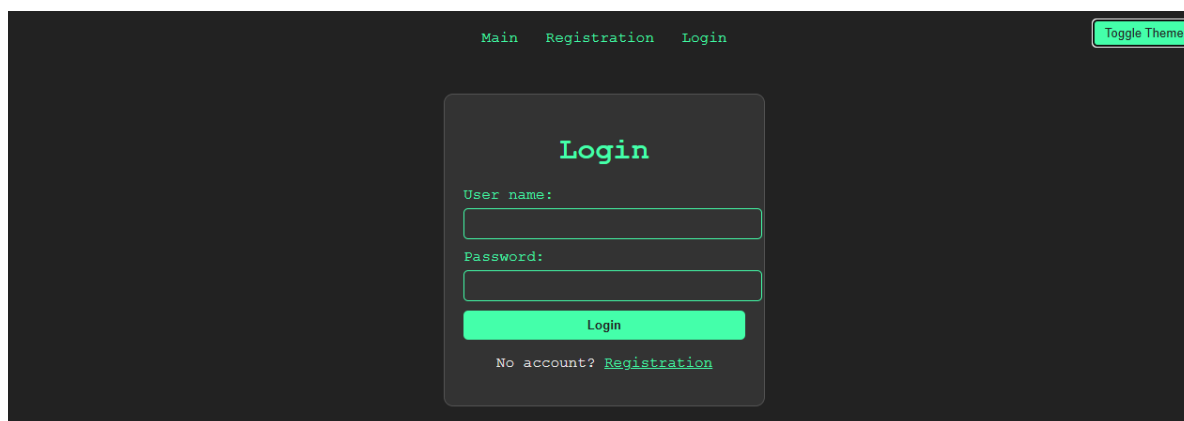
Angular byste měli zvolit, pokud potřebujete interaktivní filtry, živé aktualizace, modulární rozšíření uživatelského rozhraní a nezávislé škálování klienta; budete muset zaplatit za šířku pásma JS a školení TypeScript/CLI.

Kompromis: hybridní přístup - základní stránky kritické pro SEO může vykreslovat server (Thymeleaf, SSR Angular Universal), interaktivní skříň - modul SPA.

10 FUNKČNÍ PREZENTACE MINIBLOGU

V této kapitole není věnována pozornost vnitřní architektuře a kódu, ale tomu, co vidí a dělá koncový uživatel. Oba prototypy – čistá varianta (Servlet + JSP) i frameworková verze (Spring Boot + Angular) – poskytují identickou sadu funkcí a minimalistické uživatelské rozhraní, takže ukázka rozhraní frameworkové verze plně odráží prostředí i čisté implementace. Pro eliminaci duplicit následují snímky obrazovky a popis kroků pouze frameworkové verze; všechny uvedené akce, stránky a prvky jsou v implementaci Servlet + JSP dostupné beze změn. Tento postup umožňuje soustředit se na funkční paritu bez vlivu rozdílů v back-end stacku na uživatelskou zkušenost.

10.1 Přihlašovací stránka



Obrázek 17 - Přihlašovací stránka

Přihlašovací stránka je určena k ověření existujícího uživatele. Na centrální kartě se zobrazí dvě povinná pole - Uživatelské jméno a Heslo; po zadání údajů se klikne na tlačítko Přihlásit, které odešle formulář pomocí POST na server a v případě úspěšné validace vytvoří token JWT a přesměruje uživatele na poštovní kanál.

V horní části obrazovky se nachází jednoduché navigační menu: odkazy Hlavní, Registrace a Přihlášení umožňují přepínat mezi hlavní stránkou, registračním formulářem nového účtu a aktuální přihlašovací stránkou. Pod autorizačním tlačítkem se nachází položka No account? Registrace - kliknutím na ni otevřete formulář pro vytvoření účtu. Všechny prvky jsou stylizovány do jednotného tmavého designu; validace se provádí jak na klientovi (povinné vyplnění), tak na serveru, kde se navíc kontroluje hash jména a hesla uloženého v databázi.

10.2 Registrační stránka

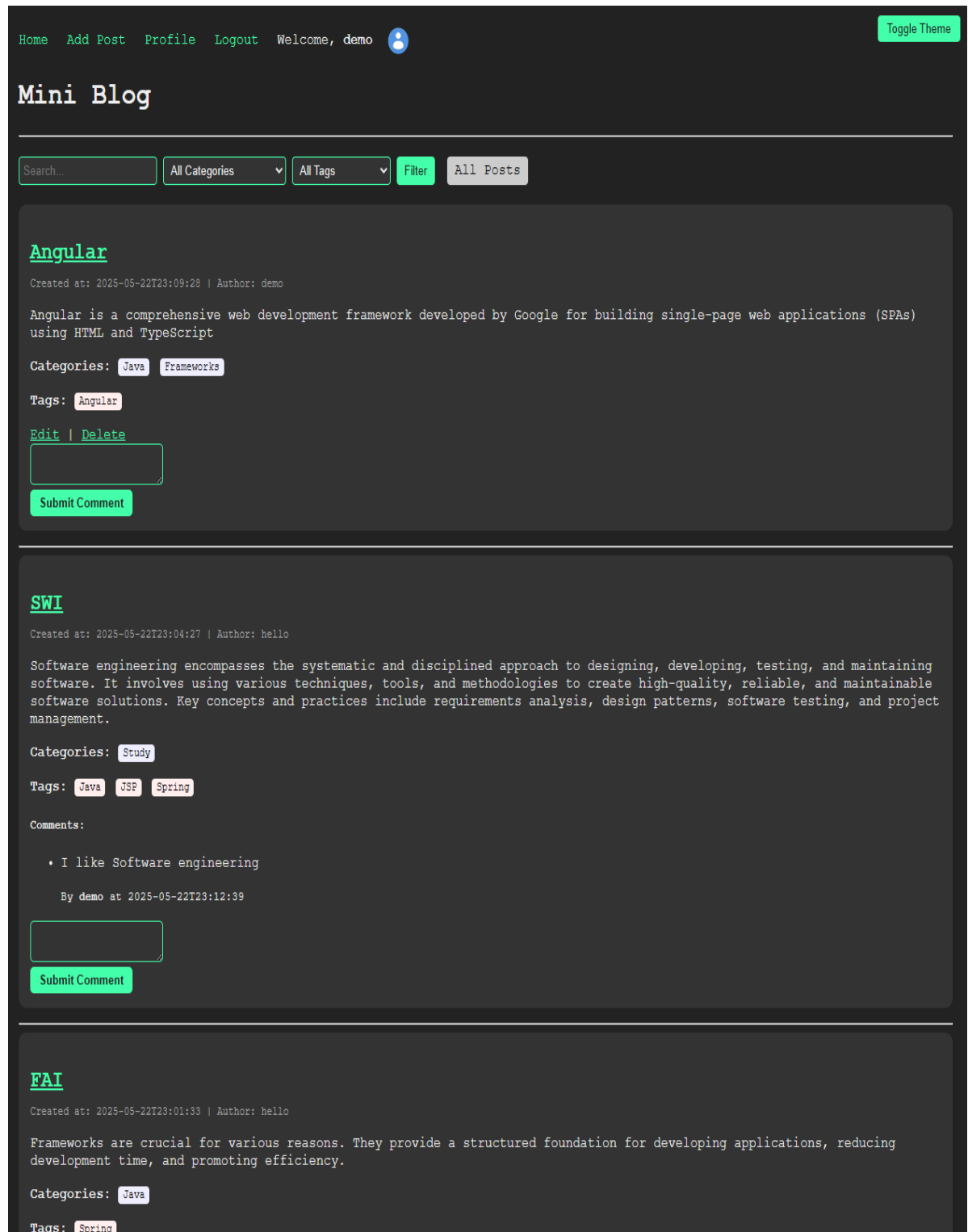
Login'. At the bottom of the page, there is a link 'Back to Home Page'." data-bbox="90 133 833 372"/>

Obrázek 18 - Přihlašovací stránka

Registrační stránka slouží k vytvoření nového účtu. Centrální formulář obsahuje tři povinná pole - uživatelské jméno, e-mail a heslo. Po vyplnění údajů a kliknutí na tlačítko Registrovat prohlížeč odešle požadavek POST na server; heslo je zaheslováno algoritmem BCrypt a v případě úspěšného vytvoření uživatele je uživatel automaticky přesunut na přihlašovací stránku.

Navigační lišta v horní části opakuje odkazy Hlavní / Přihlášení / Registrace, které umožňují návrat na hlavní stránku nebo na existující přihlašovací formulář. Pod formulářem se nachází textová výzva Už jste se zaregistrovali? Přihlášení - kliknutím na odkaz se otevře přihlašovací stránka, aniž byste ji museli ručně hledat v nabídce. Validace probíhá jak na klientovi (kontrola prázdných polí, správnost e-mailu), tak na serveru, kde je navíc kontrolována jedinečnost uživatelského jména a e-mailové adresy. V dolní části stránky se nachází odkaz Zpět na domovskou stránku, který umožňuje návrat na zdroj příspěvků jedním kliknutím bez nutnosti dokončit registraci.

10.3 Hlavní stránka



Obrázek 19 - Hlavní stránka

Úvodní stránka (post feed) se otevře ihned po přihlášení a představuje střed pracovní obrazovky uživatele. Horní navigační panel obsahuje odkazy Domů, Přidat příspěvek, Profil a Odhlásit se; vpravo se zobrazuje uvítací zpráva se jménem aktuálního uživatele a miniaturní avatar.

Těsně pod záhlavím mini blogu se nachází vyhledávací a filtrační panel:

- Vyhledávací pole umožňuje filtrovat příspěvky podle klíčových slov v názvu nebo textu;
- dva rozevírací seznamy umožňují zúžit výběr podle kategorie a značky;
- tlačítko Filtrovat použije vybrané parametry, tlačítko Všechny příspěvky filtry vynuluje.

Níže je zobrazen kanál s publikacemi. Každá karta obsahuje:

- datum vytvoření a jméno autora;
- stručný obsah;
- sadu „čipů“ Kategorie a značky, které ukazují tematickou příslušnost příspěvku;
- odkazy na úpravu a smazání, které vidí pouze autor příspěvku;
- kompaktní formulář pro přidání komentáře (textové pole a tlačítko Odeslat komentář).

Po odeslání nového komentáře se příspěvek okamžitě uloží a zobrazí se v dolní části karty; pokud k tomu máte oprávnění, můžete komentář smazat. Tlačítko Přidat příspěvek v horním menu otevře plnohodnotnou stránku pro vytvoření nového příspěvku, kde autor zadá název, obsah a označí kategorie/značky.

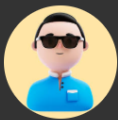
Domovská stránka tak kombinuje prohlížení, vyhledávání, filtrování a rychlou interakci s obsahem, čímž uživateli poskytuje kompletní zážitek z blogu, aniž by musel přecházet na vedlejší stránky.

10.4 Stránka uživatele

Home Add Post Profile hello Logout Toggle Theme

Your Profile

Avatar



Upload new avatar:
Выберите файл Файл не выбран

Save Avatar

Email: hello@a.com

Username: hello Change

Your Posts

SWI

Created at: 2025-05-22T23:04:27

Software engineering encompasses the systematic and disciplined approach to designing, developing, testing, and maintaining software. It involves using various techniques, tools, and methodologies to create high-quality, reliable, and maintainable software solutions. Key concepts and practices include requirements analysis, design patterns, software testing, and project management.

Categories: Study

Tags: Java JSP Spring

Edit Delete

FAI

Created at: 2025-05-22T23:01:33

Frameworks are crucial for various reasons. They provide a structured foundation for developing applications, reducing development time, and promoting efficiency.

Categories: Java

Tags: Spring

Edit Delete

Obrázek 20 - Stránka uživatele

Profilová stránka poskytuje uživateli osobní kabinet, kde jsou soustředěna nastavení účtu a seznam příspěvků, které uživatel zveřejnil. V horní části je zobrazen blok Avatar: kruhový obrázek aktuálního avataru a formulář pro nahrání nového. Tlačítko Zvolit soubor umožňuje vybrat obrázek na disku, po kliknutí na tlačítko Uložit avatar se obrázek odešle na server a uloží se do adresáře /uploads a profil se okamžitě aktualizuje. Pod ním je pevně zadána e-mailová adresa a aktuální uživatelské jméno; pokud chcete jméno změnit, zadáním nové hodnoty a kliknutím na tlačítko Změnit se spustí aktualizace příslušného záznamu v tabulce uživatelů.

Pod záhlavím Your Posts (Vaše příspěvky) se nachází kanál všech publikací vytvořených tímto autorem. Každá karta obsahuje datum, název, text a označení kategorie/značky a také odkazy Upravit a Smazat, pomocí nichž může vlastník příspěvek upravit nebo zcela smazat. Osobní údaje, správa profilu a rychlý přístup k vlastnímu obsahu jsou tak soustředěny na jedné stránce, což zjednodušuje každodenní práci s blogem. stránka uživatele.

10.5 Přidání příspěvku

Add a New Post

Title:

Content:

Categories

- ☐ Study
- ☐ world

Tags

- ☐ Angular
- ☐ Java
- ☐ JSP
- ☐ Spring

Submit Post

Add Category

Category name

Add Category

Add Tag

Tag name

Add Tag

[Back to Home Page](#)

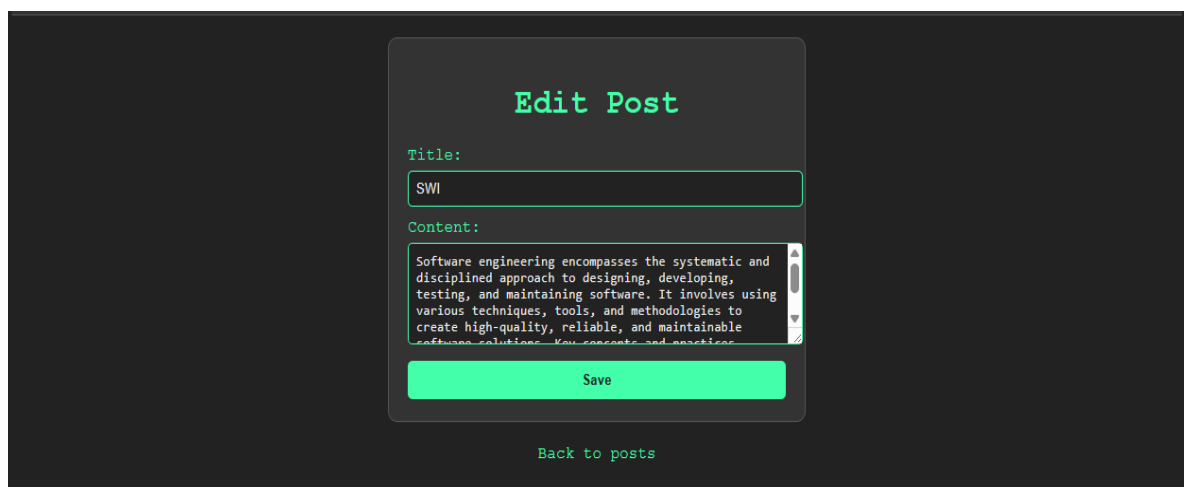
Obrázek 21 - Přidání příspěvku

Stránka „Přidat nový příspěvek“ umožňuje autorovi vytvořit nový materiál a okamžitě jej propojit s tématem. V horní části formuláře jsou dvě povinná pole - Název a Obsah; bez vyplnění alespoň jednoho z nich se tlačítko Odeslat příspěvek neaktivuje. Dole jsou zobrazeny dva víceřádkové seznamy se zaškrťovacími políčky. První z nich, označený jako Kategorie, obsahuje existující kategorie blogu - stačí zaškrtnout jednu nebo více z nich, aby

se příspěvek automaticky zařadí do kategorie. Druhý seznam - Štítky - funguje na stejném principu a připojuje k příspěvku sadu štítků pro usnadnění dalšího vyhledávání a filtrování.

Hned za hlavním formulářem se nacházejí dva pomocné bloky, které umožňují taxonomii průběžně rozšiřovat. V části Přidat kategorii můžete zadat název nové kategorie a kliknout na tlačítko Přidat kategorii - záznam se okamžitě uloží do databáze a objeví se v seznamu výše bez nutnosti opětovného načítání stránky. Sekce Přidat značku funguje stejně: zadaný text se uloží jako nová značka a je k dispozici pro výběr. V dolní části stránky se nachází odkaz Zpět na úvodní stránku, který uživatele vrátí na feed příspěvků, aniž by došlo ke ztrátě zadaných údajů, pokud publikace ještě nebyla potvrzena. Stránka tak kombinuje tvorbu samotného obsahu a dynamickou správu jeho rubrik, což autorovi umožňuje plně organizovat nový materiál na jednom místě.přidání příspěvku.

10.6 Uprava příspěvků

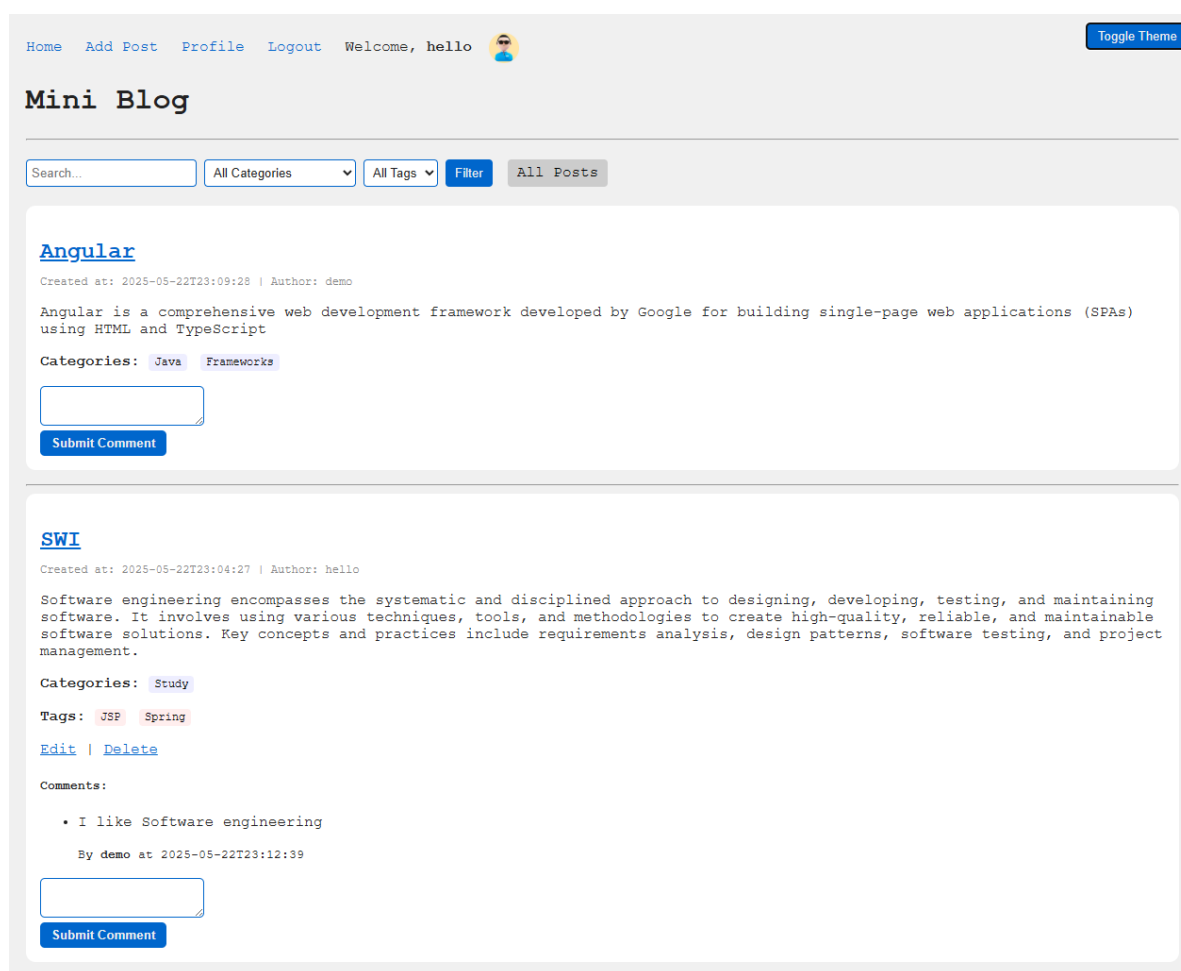


Obrázek 22 - Uprava příspěvků

Stránka „Upravit příspěvek“ je určena k provádění změn v již zveřejněném materiálu. Při načtení formuláře se do příslušných políček nahradí aktuální hodnoty polí Nadpis a Obsah - to umožňuje autorovi vidět najednou původní text a upravovat pouze potřebné fragmenty. Pole Název zůstává jednořádkové, Obsah je víceřádkové textové pole s vertikálním rolováním; oba prvky jsou povinné, takže tlačítko Uložit se aktivuje pouze v případě přítomnosti textu.

Po úpravě se po stisknutí tlačítka Uložit odešle formulář pomocí POST (nebo PUT ve variantě SPA) do koncového bodu `/posts/{id}`. Na serveru je provedena kontrola autorských práv: aktualizovat smí pouze tvůrce příspěvku. Pokud je uložení úspěšné, je uživatel automaticky přesměrován zpět na kanál, kde se zobrazí aktualizovaný příspěvek s aktuálním časem změny. Pokud je třeba změny zrušit, odkaz Zpět na příspěvky pod formulářem umožňuje uživateli návrat do seznamu bez odesílání údajů. Stránka zachovává stejný stručný styl aplikace a poskytuje intuitivní editační skript bez rušivých zbytečných prvků rozhraní.

10.7 Změna barevného schématu



Obrázek 23 - Změna barevného schématu

V pravém horním rohu každé stránky je tlačítko Přepnout motiv. Kliknutím na něj uživatel okamžitě přepne rozhraní mezi tmavou a světlou paletou. Na úrovni implementace je to řešeno pomocí proměnných CSS: dokument přidá (nebo odebere) třídu `light` a skript `theme.js` změní sadu kořenových proměnných `--bg-color`, `--text-color`, `--card-bg` atd. Vybraný motiv

je uložen v localStorage, takže při opětovném načtení prohlížeče se otevře stejný motiv, jaký byl vybrán dříve.

Z hlediska funkčnosti zůstává stránka stejná: vyhledávací panel, rozbalovací nabídky filtrů, tlačítka Filtrovat / Všechny příspěvky a karta feedu s příspěvky, komentáři a odkazy Upravit | Smazat fungují v obou režimech stejně. Jediný rozdíl je ve vizuálním vjemu: ve světlém motivu se pozadí změní na bílé, text na tmavě šedý, zatímco akcentní prvky (tlačítka, odznaky kategorií a značek) si zachovávají charakteristickou tyrkysovou barvu, jak ukazuje obrázek výše. Přepínač motivů tak zvyšuje komfort práce s blogem při různém osvětlení, aniž by byla ovlivněna logika aplikace.

11 NÁKLADY NA MONTÁŽ, NAsAZENÍ A PRÁCI

V prvních kapitolách byly rozebrány technické aspekty obou implementací MiniBlogu, od architektury po frontend. Nyní bude věnována pozornost tomu, kolik času a úsilí vyžadovaly jednotlivé verze.

11.1 Pracovní náklady MiniBlogu (bez frameworku)

- Typ práce. Počítáme pouze skutečný „čistý“ čas vývoje: návrh databázového schématu, psaní servletů/DAO/JSP, stylování, základní testy Postmanu, příprava README. Deploy a CI skripty nejsou započítány (jsou zde minimální).
- Oprava hodin. Byl použit sledovací systém GitLab Time-Tracking; každý problém byl označen tagem stráveného času. Hodnoty jsou zaokrouhleny na nejbližší hodiny.
- LOC-metriky: V rámci této práce byla použita tzv. Počítáno pomocí nástroje cloc 1.96 (příznaky --exclude-dir=target,webapp/css,webapp/js) po fázi „feature-freeze“.
- Kotelní šablona. Ručně odhadnut podíl opakujícího se kódu (try-with-resources, getters/setters, HTML wrappery) - nejedná se o chybu, ale o neproduktivní zátěž.

Úkol	Obsah	Hodiny
Model ER	diagram, skripty DDL, indexy	8
Vrstva DAO	UserDAO, PostDAO, CommentDAO, nástroje JDBC	18
Servlety + filtr	AuthFilter, AddPostServlet, LoginServlet a další.	12
JSP + CSS	listPosts.jsp, form.jsp, téma CSS, přepínač témat	10
Ověřování JWT	Integrace JWT, testy Postman, zpracování 401	6
Refaktoring a opravy chyb	úpravy po ručním testování	4
Dokumentace	README, komentáře v kódu	2
Kumulativně	—	≈ 60 h

Tabulka 7 - Rozdělení hodin podle úkolů

Typ souboru	Soubory.	LOC	Z toho boilerplate
Java (servlet + DAO)	43	2 750	~35 %
JSP	12	820	~15 %
CSS / JS (téma)	6	140	—
Celkem	—	3 710	≈ 55 %

Tabulka 8 - Statistiky kódů (bez Framework)

11.1.1 Pozorování

Efekt kotlové desky. Téměř každá operace JDBC opakuje vzorec otevření spojení, příprava PreparedStatement, zpracování ResultSet. Podle časového deníku bylo na takové „mechanické“ bloky vynaloženo ~40 % času vývoje DAO.

Příspěvek frontendu. Navzdory malému počtu řádků JSP zabralo 10 h rozvržení a CSS: ruční značkování HTML a opakování hlavičky/zápatí bez šablonovacího stroje.

Testování. Kvůli nedostatku DI a oddělení modulů nebyly napsány unit testy; testování probíhalo prostřednictvím Postman-collections a prohlížeče, což zvýšilo čas potřebný k opravě chyb.

Úpravy křivek. Jakákoli změna databázového schématu (přidání stavového pole k příspěvkům) vyžaduje synchronní změnu SQL dotazů ve všech DAO a ve výstupu JSP. To se stává multiplikátorem nákladů na vývoj produktu.

11.1.2 Výsledky

- Celková částka 60 člověkohodin je pro školení nebo projekt MVP přijatelná, ale polovina tohoto času nebyla věnována obchodní logice, ale operacím se šablonou JDBC a rozvržením.
- Škálování povede k lineárnímu nárůstu nákladů: každý nový případ použití je servlet + DAO + JSP, opakování kódu.
- Ekonomický závěr: Bezrámcový stack je výhodný, když je důležitá minimální závislost a plná kontrola, ale při dlouhodobém vývoji bude tým čelit vysokému technickému dluhu a nákladům na podporu.

11.2 Pracovní náklady MiniBlogu (s frameworkem)

Typ práce je rozdělen na backend-blok (Spring Boot 3) a frontend-blok (Angular SPA). Zohledněny byly následující činnosti: návrh entit, psaní kontrolérů a služeb, nastavení Spring Security, tvorba Angular komponent, unit- a e2e-testy, příprava Docker a Helm souborů a tvorba dokumentace.

- Sledování času. V systému GitLab Time-Tracking měl každý problém štítek stráveného času; celkové součty se zaokrouhlovaly na hodiny.
- LOC / boilerplate. Počítáno cloc po feature-freeze; opakující se gettery, kód generovaný Angular-CLI a XML manifesty jsou označeny jako boilerplate.

Úkol	Obsah	Hodiny
Schéma ER, anotace JPA	entity Příspěvek, Uživatel + @ManyToMany	6
Úložiště + služba	PostRepository, findFiltered(), @Transactional servis	10
Řadiče REST	AuthController, PostController, validace DTO	8
Spring Security + JWT	JwtFilter, SecurityConfig, Unit tests	6
Lešení Angular CLI	generování modulů, konfigurace proxy serveru, základní rozvržení	5
Příspěvky/komentáře	Šablona HTML, CSS, oboustranné vazby, potrubí	10
PostService (HTTP)	CRUD + filtr, RxJS, Interceptor	4
Docker + Helm	vícestupňový soubor Dockerfile, values.yaml, lokální druh	4
Jednotkové testy / testy E2E	JUnit + Mockito, Angular Karma + Protractor	5
Celkem	—	≈ 60 h backend ≈ 40 h frontend ≈ 100 h celkem

Tabulka 9 - Rozdělení hodin podle úkolů

Typ souboru	Soubory	LOC	Boilerplate
-------------	---------	-----	-------------

Java (entita, kontrolér, služba)	62	1 850	~20 %
TypeScript (komponenty, služby)	58	1 380	~10 %
HTML / CSS	34	420	—
YAML (Docker, Helm, CI)	11	160	—
Celkem	—	3 810	≈ 30 %

Tabulka 10 - Statistiky kódů (Framework)

11.2.1 Pozorování

Méně šablonovitosti. Díky JpaRepository a @Transactional je podíl kotelního kódu téměř poloviční oproti verzi bez frameworků.

Zrychlení CLI. Angular CLI vygeneroval framework pro komponenty a testy (< 5 min), což snížilo počet hodin práce frontendu.

Přibýlo hodin DevOps. Dockerizace a Helm chart zabraly 4 hodiny, ale pak se nasazení sjednotilo pro DEV → UAT → PROD.

Strmá křivka učení. Noví přispěvatelé strávili čas učním Spring Security, kaskád JPA a RxJS; to je započítáno do hodin „testy + oprava chyb“.

Testovatelnost je vyšší. Explicitní oddělení vrstev umožnilo napsat 40 unit testů JUnit/Mockito a 10 skriptů Protractor e2e bez složitého vysmívání servletů.

11.2.2 Výsledky

- Celkové náklady na práci (~100 h) jsou vyšší než u systému No Framework, zejména kvůli frontendu SPA a konfiguraci DevOps.

- Zkrácení času šablony (30 % oproti 55 %) znamená, že růst nákladů bude při dalším vývoji sublineární: nové fiche CRUD jsou přidávány pomocí anotace `@RepositoryRestResource` a příkazu CLI komponenty `ng g`.
- Ekonomický závěr: Stack Spring Boot + Angular se dlouhodobě vyplatí (méně času na údržbu a rozšíření), ale vyžaduje počáteční investici do školení a infrastruktury. Pokud projekt zahrnuje dlouhodobý vývoj a tým > 2-3 lidí, je ekonomicky výhodnější varianta s frameworkem.

12 ZÁTĚŽOVÉ TESTOVÁNÍ

V této podkapitole jsou představeny dva odlišné přístupy k „klientské“ části MiniBlogu. Nejprve je popsán server-side rendering: soubory .jsp a JSTL tagy na Tomcatu generují kompletní HTML ještě před odesláním do prohlížeče. Následuje popis Single-Page Application založené na Angularu 17, kde se stránky sestavují dynamicky v prohlížeči pomocí komponent, rout a volání REST-API. V závěrečné části jsou oba modely porovnány z hlediska uživatelského dojmu, rychlosti načítání, SEO a pracnosti vývoje, aby bylo možné posoudit vhodnost každého řešení pro různé scénáře.

12.1 MiniBlog bez Frameworku

Tato část shrnuje kompletní měření varianty MiniBlog No-Framework po doplnění JDBC-poolu v Tomcatu. Testováno bylo 3 úrovně zátěže, vždy stejným k6 skriptem.

Přednastavení	Parametr y	p95 (ms)	RP S	Error %	CPU % (Java)	RAM M B
Jednoduché	5 VU · 30 s	57	9	0	≈6 %	≈211
Normální	100 VU · 2 min	271	160	0	≈44 %	≈223
Stres	300 VU · 2 min	1 120	273	0	≈ 57 %	≈288

Tabulka 11 - Zátěžové testování bez frameworku

Přehled shrnuje tři úrovně zátěže (Jednoduché → Nominal → Stress), aplikované pomocí k6 na „čistou“ verzi MiniBlogu. Pro každý stupeň jsou zaznamenávány:

- p95 – 95-percentil latence (hlavní SLA projektu).
- RPS – propustnost při daném počtu VU.
- Error % – podíl neúspěšných HTTP volání.
- CPU % a RAM – kolik prostředků JVM skutečně spotřebuje.

Takto zhuštěná tabulka dovoluje na jediném místě vidět, kde je hranice, za níž už bezrámcová implementace přestává plnit SLA, a současně odhaluje, jestli problémem je procesor, paměť, nebo naopak limity vláken/JDBC-poolu.

12.2 MiniBlog Spring Boot

Tato část shrnuje kompletní měření varianty MiniBlog Spring Boot + Angular po doplnění JDBC-poolu v Tomcatu. Testováno bylo stejným k6 skriptem ve třech úrovních zátěže.

Přednastavení	Parametry	p95 (ms)	RPS	Error %	CPU % (Java)	RAM MB
Jednoduché	5 VU · 30 s	94	13	0	≈ 7 %	≈ 6 680
Normální	100 VU · 2 min	1 730	89	0	≈ 50 %	≈ 7 200
Stres	300 VU · 2 min	8 160	76	0	≈ 66 %	≈ 8 800

Tabulka 12 - Zátěžové testování Spring Boot

- SLA splněno jen v „Jednoduché“ režimu – do 5 VU p95 = 94 ms, chybovost 0 %.
- Při 100 VU p95 ≈ 1,7 s → výrazně nad cílem 300 ms; CPU je využito ~50 %, ukazuje se overhead frameworku.
- Při 300 VU p95 ≈ 8 s; JVM stále bez chyb, ale latence dramaticky roste kvůli vyčerpání vláknového/JDBC-poolu a složité serializaci.
- Paměťový profil je stabilní, ale vysoký (~7–9 GB) – výrazně náročnější než u „čisté“ verze (~300 MB).
- Rezerva optimalizace – caching, DTO-projekce, Afterburner, native-image nebo horizontální škálování pomohou vrátit p95 pod 300 ms i při vyšší zátěži.

12.3 Výsledky

V této kapitole byly porovnány dva přístupy k implementaci klient-server části MiniBlogu – „čistá“ varianta bez frameworku (JDBC + Servlet) a plnohodnotný balíček Spring Boot + Angular. Na základě výsledků zátěžových testů při třech úrovních zátěže (5 VU, 100 VU, 300 VU) lze vyvodit následující klíčové závěry:

1. Výkon při nízkém zatížení (5 VU)

- Bez frameworku: p95 ≈ 57 ms, RPS ≈ 9, CPU ≈ 6 %, RAM ≈ 211 MB
- Spring Boot+Angular: p95 ≈ 94 ms, RPS ≈ 13, CPU ≈ 7 %, RAM ≈ 6 680 MB

Při nízké zátěži oba zásobníky splňují SLA (p95 < 300 ms). Varianta s frameworky přitom spotřebuje řádově více paměti kvůli JVM a frontend builderu.

2. Normální zátěž (100 VU)

- Bez frameworku: p95 $\approx$ 271 ms, RPS $\approx$ 160, CPU $\approx$ 44 %, RAM $\approx$ 223 MB.
- Spring Boot+Angular: p95 $\approx$ 1 730 ms, RPS $\approx$ 89, CPU $\approx$ 50 %, RAM $\approx$ 7 200 MB
„Čistá“ verze udržuje p95 v SLA a poskytuje dvojnásobnou šířku pásma. Verze Spring Boot je již „zadušená“ režii frameworku a reaguje pomalu.

3. Stres zátěž (300 VU).

- Bez frameworku: p95 $\approx$ 1 120 ms, RPS $\approx$ 273, CPU $\approx$ 57 %, RAM $\approx$ 288 MB.
- Spring Boot+Angular: p95 $\approx$ 8 160 ms, RPS $\approx$ 76, CPU $\approx$ 66 %, RAM $\approx$ 8 800 MB

Při vysokém zatížení se verze No-Framework začne „dusit“ kvůli vyčerpání connection pools/threads, ale zůstává mnohonásobně rychlejší a úspornější na zdroje než varianta s frameworkem.

ZÁVĚR

V předložené práci bylo provedeno vyčerpávající hodnocení vlivu moderních frameworků na efektivitu vývoje webových aplikací. V teoretické části byly podrobně rozebrány klíčové technologie na straně klienta a serveru (HTML, CSS, JavaScript, Java), klasifikace a vývoj webových frameworků a kritéria jejich výběru zohledňující výkon, bezpečnost, škálovatelnost a udržitelnost. Praktická část zahrnovala paralelní implementaci dvou prototypů aplikací - od nuly, bez použití hotových nástrojů a s využitím současných frameworků - a jejich srovnávací analýzu z hlediska doby vývoje, spotřeby systémových prostředků a kvality kódu.

Výsledky experimentu prokázaly výrazné snížení pracnosti implementace funkcí při použití frameworků: díky šablonám, komponentovému přístupu a vestavěným nástrojům pro správu stavů a směřování je kód vytvářen a udržován rychleji než při ručním psaní. Pozorované mírné zvýšení paměťové režie a doby odezvy bylo z hlediska přínosů odůvodněné. Použití řešení ORM a nástrojů pro kontejnerizaci navíc zjednodušilo organizaci interakce s databází a prostředí pro nasazení.

V průběhu práce se potvrdila hlavní teze, že zavedení moderních frameworků umožňuje zvýšit produktivitu týmu a zjednodušit další vývoj projektu při nevýznamném zhoršení „surových“ výkonnostních ukazatelů. Zároveň se ukázalo, že pro malé a jednoduché služby je vhodnější minimalistický přístup bez těžkotonážních struktur, zatímco při vývoji složitých podnikových systémů má opodstatnění použití plnohodnotných stacků s širokou škálou modulů.

SEZNAM POUŽITÉ LITERATURY

- [1] DUCKETT, Jon. *HTML & CSS: design and build websites*. Indianapolis: Wiley, c2011. ISBN 978-1-118-00818-8.
- [2] FLANAGAN, David. *JavaScript: The Definitive Guide*. 7th edition. O'Reilly Media, 2020. ISBN 9781491952023.
- [3] HAVERBEKE, Marijn. *Eloquent JavaScript: A Modern Introduction to Programming*. 3rd edition. San Francisco, CA: No Starch Press, 2018. ISBN 9781593279509.
- [4] DARVISHDARAB, Client-Server Architecture [online]. Dostupné z: https://darvishdarab.github.io/cs421_f20/docs/readings/client_server/
- [5] SCHILDT, Herbert. *Java: the complete reference*. Ninth edition. [New York]: McGraw-Hill Education, c[2014]. ISBN 978-0071808552.
- [6] BLOCH, Joshua. *Effective Java*. 2nd ed. The Java series. Upper Saddle River: Addison-Wesley, c2008. ISBN 978-0-321-35668-0.
- [7] COSMINA, Iuliana; HARROP, Rob; SCHAEFER, Chris a CLARENCE, Clarence. *Pro Spring 5*. 5rd edition. New York, NY: Apress, 2017. ISBN 9781484228074.
- [8] WALLS, Craig. *Spring in action: covers spring 5.0*. Fifth edition. Shelter Island: Manning, [2019]. ISBN 978-1-61729-494-5.
- [9] RICHARDSON, Chris. *Microservices patterns: with examples in Java*. Shelter Island: Manning, [2019]. ISBN 978-1-61729-454-9.
- [10] OAKS, Scott. *Java performance: the definitive guide*. Sebastopol: O'Reilly, 2014. ISBN 978-1-4493-5845-7.
- [11] SCHILDT, Herbert. *Java: the complete reference*. J2SE 5 ed. New York: McGraw-Hill/Osborne, c2005. ISBN 0-07-223073-8.
- [12] FOWLER, Martin. *Patterns of enterprise application architecture*. 6th print. The Addison-Wesley signature series. Boston: Addison-Wesley, 2004. ISBN 0-321-12742-0.
- [13] KLEPPMANN, Martin. *Designing data-intensive applications: the big ideas behind reliable, scalable, and maintainable systems*. Beijing: O'Reilly, 2017. ISBN 978-1-449-37332-0.

- [14] VERNON, Vaughn. *Implementing Domain-Driven Design*. Addison-Wesley Professional, 2013. ISBN 9780133039900.
- [15] SESHADRI, Shyam. *Angular Up & Running: Learning Angular Step by Step*. 2nd edition. Sebastopol, CA: O'Reilly Media, 2018. ISBN 9781491999837.
- [16] FREEMAN, Adam. *The Road to React*. Apress, 2020. ISBN 9781484259986.
- [17] YOU, Evan. *Vue.js Documentation [online]*. Dostupné z : <https://vuejs.org>
- [18] HERRIN, Rich. *Svelte Docs*. Dostupné z: <https://svelte.dev>
- [19] EMBER.JS. *Ember.js Guides [online]*. Dostupné z: <https://emberjs.com>
- [20] KLEPPMANN, Martin. *Designing data-intensive applications: the big ideas behind reliable, scalable, and maintainable systems*. Beijing: O'Reilly, 2017. ISBN 978-1-449-37332-0.
- [21] FOWLER, Martin. *Patterns of enterprise application architecture*. Boston: Addison-Wesley, c2003. ISBN 0321127420.
- [22] PFLEEGER, Shari Lawrence a ATLEE, Joanne M. *Software engineering: theory and practice*. 4th ed. Upper Saddle River: Prentice Hall, c2010. ISBN 978-0-13-606169-4.
- [23] BECK, Kent. *Test-driven development: by example*. The Addison-Wesley signature series. Boston: Addison-Wesley, [2003]. ISBN 0-321-14653-0.
- [24] LEWIS, James; FOWLER, Martin. *Microservices: a definition of this new architectural term*. Martin Fowler Blog, 2014. [online]. Dostupné z: <https://martinfowler.com/articles/microservices.html>
- [25] PRESSMAN, Roger S. a MAXIM, Bruce R. *Software engineering: a practitioner's approach*. Eighth edition. New York, NY: McGraw-Hill Education, [2015]. ISBN 978-0-07-802212-8.
- [26] *JSP Architecture [online]*. Dostupné z: <https://www.oracle.com/technical-resources/articles/javase/servlets-jsp.html>
- [27] Client-server model [online]. Dostupné z: https://en.wikipedia.org/wiki/Client%E2%80%93server_model
- [28] HTML, JAVA, JS layered [online]. Dostupné z: <https://ddatlearnocode.netlify.app/lesson/introducing-javascript/>

- [29] Spring-architecture [online]. Dostupné z: <https://www.bmc.com/blogs/spring-framework/>
- [30] JSF Architecture [online]. Dostupné z: https://www.tutorialspoint.com/jsf/jsf_architecture.htm
- [31] Vaadin Architecture [online]. Dostupné z: <https://www.geeksforgeeks.org/what-is-vaadin/>
- [32] Architektura Dockeru na vysoké úrovni [online]. Dostupné z: <https://medium.com/@furkan.turkal/how-does-docker-actually-work-the-hard-way-a-technical-deep-diving-c5b8ea2f0422>
- [33] Popularity of Angular, React and Vue.js [online]. Dostupné z: <https://blog.bajonczak.com/reactvsangularsvue/>
- [34] What is Kubernetes [online]. Dostupné z: <https://sensu.io/blog/how-kubernetes-works>
- [35] JAVA EE6 programování webových aplikací cz.1 [online]. Dostupné z: <https://www.dobreprogramy.pl/@kadoel/java-ee-programowanie-aplikacji-www-cz-wprowadzenie,blog,26995>
- [36] Angular, React or Vue what is better? [online]. Dostupné z: <https://blog.bajonczak.com/reactvsangularsvue/>

SEZNAM OBRÁZKŮ

Obrázek 1 - Clients communicating with a server via the Internet [27]	16
Obrázek 2 - HTML, CSS, JS layered [28]	20
Obrázek 3 - JSP Architektura [35]	22
Obrázek 4 - Klasifikace frameworků	24
Obrázek 5 - Spring-architecture [29]	34
Obrázek 6 - JSF – Architecture [30]	35
Obrázek 7 - Architecture of Vaadin [31]	36
Obrázek 8 - Architektura Dockeru na vysoké úrovni [32]	43
Obrázek 9 - Kubernetes [34]	44
Obrázek 10 - Popularity of Angular, React and Vue.js [36]	46
Obrázek 11 - Schéma toku zpracování HTTP požadavků v monolitické verzi MiniBlogu na čistých servletech + JSP + JDBC	67
Obrázek 12 - Architektura full-stack řešení MiniBlog Spring Boot + Angular	70
Obrázek 13 - ER diagram datového modelu MiniBlogu (No Framework)	76
Obrázek 14 - ER-diagram modelu JPA	82
Obrázek 15 - Tok požadavků v aplikaci MiniBlog No Framework	88
Obrázek 16 - Tok požadavků v aplikaci MiniBlog Spring Boot + Angular	90
Obrázek 17 - Přihlašovací stránka	99
Obrázek 18 - Přihlašovací stránka	100
Obrázek 19 - Hlavní stránka	101
Obrázek 20 - Stránka uživatele	103
Obrázek 21 - Přidání příspěvku	104
Obrázek 22 - Uprava příspěvků	105
Obrázek 23 - Změna barevného schématu	106

SEZNAM TABULEK

Tabulka 1 - Srovnání architektur	73
Tabulka 2 - Srovnání přístupů	86
Tabulka 3 - Zabezpečení v toku bez frameworku.....	89
Tabulka 4 - Zabezpečení v toku s použitím frameworku	91
Tabulka 5 - Srovnání přístupů	92
Tabulka 6 - Komplexní závěry na frontendu MiniBlogu	97
Tabulka 7 - Rozdělení hodin podle úkolů.....	109
Tabulka 8 - Statistiky kódů (bez Framework)	109
Tabulka 9 - Rozdělení hodin podle úkolů.....	112
Tabulka 10 - Statistiky kódů (Framework).....	113
Tabulka 11 - Zátěžové testování bez frameworku.....	115
Tabulka 12 - Zátěžové testování Spring Boot	116

SEZNAM UKÁZEK KÓDU

Ukázka kódu 1 - INSERT post (DAO → JDBC)	77
Ukázka kódu 2 - SELECT uživatele (ruční mapování ResultSet)	78
Ukázka kódu 3 - INSERT komentáře	79
Ukázka kódu 4 - Obchodní logika vytvoření příspěvku + vazby M:N	80
Ukázka kódu 5 - Otevření JDBC připojení	81
Ukázka kódu 6 - Podstata příspěvku	83
Ukázka kódu 7 - PostRepository	84
Ukázka kódu 8 - PostServiceImpl - vytváření a mazání příspěvku	85
Ukázka kódu 9 - Zabezpečení v toku bez frameworku	89
Ukázka kódu 10 - Zabezpečení v toku s použitím frameworku	91
Ukázka kódu 11 - form.jsp	93
Ukázka kódu 12 - listPosts.jsp	94
Ukázka kódu 13 - posts.component.html	95
Ukázka kódu 14 - post.service.ts	96

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

DAO – Data Access Object – objekt pro přístup k datům

JDBC – Java Database Connectivity – propojení s databází v Javě

JSP – JavaServer Pages – stránky JavaServer

JSTL – JavaServer Pages Standard Tag Library – standardní knihovna značek pro JSP

ER – Entity-Relationship – entitně-relační (model/databázové schéma)

DDL – Data Definition Language – jazyk pro definici datových struktur (CREATE, ALTER, DROP...)

API – Application Programming Interface – rozhraní pro programování aplikací (REST-endpointy)

HTTP – HyperText Transfer Protocol – protokol přenosu hypertextu

JWT – JSON Web Token – JSON Web Token (webový token pro autentizaci)

CSRF – Cross-Site Request Forgery – padělání požadavků mezi stránkami (ochrana před CSRF)

CORS – Cross-Origin Resource Sharing – sdílení zdrojů mezi různými doménami

SPA – Single-Page Application – jednostránková aplikace (Angular)

ORM – Object-Relational Mapping – objektově-relační mapování (Hibernate/JPA)

JPA – Java Persistence API – API pro perzistenci v Javě (Hibernate)

DTO – Data Transfer Object – objekt pro přenos dat mezi vrstvami aplikace

MVC – Model-View-Controller – model-zobrazení-řadič (architektonický vzor)

IoC – Inversion of Control – inverze řízení (vstřikování závislostí)

DI – Dependency Injection – vstřikování závislostí (Spring)

AOP – Aspect-Oriented Programming – aspektově orientované programování (Spring AOP)

SEZNAM PŘÍLOH

Příloha P I: CD-ROM

PŘÍLOHA P I: CD-ROM

Příloha CD-ROM obsahuje:

- Zdrojový kód ukázkové aplikace, která je uložena v souboru .zip
- Bakalářskou práci ve formátu .pdf