STŘEDOŠKOLSKÁ ODBORNÁ ČINNOST

Obor: 18. Informatika

Implementace mikroservisní architektury a dynamického pluginového ekosystému pro robustní sociální platformu s využitím orchestrace Kubernetes

Patrik Stohanzl

Pardubice 2025

STŘEDOŠKOLSKÁ ODBORNÁ ČINNOST

IMPLEMENTACE MIKROSERVISNÍ ARCHITEKTURY A DYNAMICKÉHO PLUGINOVÉHO EKOSYSTÉMU PRO ROBUSTNÍ SOCIÁLNÍ PLATFORMU S VYUŽITÍM ORCHESTRACE KUBERNETES

IMPLEMENTING MICROSERVICE ARCHITECTURE AND DYNAMIC PLUGIN ECOSYSTEM FOR A ROBUST SOCIAL PLATFORM USING KUBERNETES ORCHESTRATION

AUTOR	Patrik Stohanzl
ŠKOLA	DELTA - Střední škola informatiky a ekonomie
KRAJ	Pardubický
ŠKOLITEL	RNDr. Jan Koupil, PhD.
OBOR	18. Informatika

Pardubice 2025

Prohlášení

Prohlašuji, že svou práci na téma *Implementace mikroservisní architektury a dynamického pluginového ekosystému pro robustní sociální platformu s využitím orchestrace Kubernetes* jsem vypracoval/a samostatně pod vedením RNDr. Jana Koupila, PhD. a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Dále prohlašuji, že tištěná i elektronická verze práce SOČ jsou shodné a nemám závažný důvod proti zpřístupňování této práce v souladu se zákonem č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a změně některých zákonů (autorský zákon) v platném znění.

V Pardubicích dne: _____ _____

Patrik Stohanzl

Poděkování

Děkuji svému vedoucímu RNDr. Janu Koupilovi, PhD. za obětavou pomoc, podnětné připomínky a nekonečnou trpělivost, kterou mi během práce poskytoval.

Abstrakt

Tato práce se zabývá návrhem a implementací moderní sociální platformy Socigy, která řeší identifikované nedostatky současných sociálních sítí prostřednictvím inovativního přístupu k architektuře, bezpečnosti a rozšiřitelnosti. Práce analyzuje současný stav sociálních platforem a identifikuje jejich hlavní nedostatky, zejména v oblasti uživatelské kontroly, transparentnosti algoritmů a bezpečnosti.

Na základě této analýzy je navržena robustní mikroservisní architektura využívající Kubernetes pro orchestraci, HashiCorp Consul pro Service Mesh a PostgreSQL s Patroni pro zajištění vysoké dostupnosti dat. Klíčovým inovativním prvkem je implementace dynamického pluginového systému založeného na WebAssembly, který umožňuje bezpečné spouštění kódu třetích stran v sandboxovaném prostředí.

Práce detailně popisuje implementaci jednotlivých komponent systému, včetně autentizačních mechanismů využívajících standard FIDO2 (Passkeys), mikroservisního ekosystému a klientských aplikací pro mobilní a webové platformy. Zvláštní pozornost je věnována vývoji vlastního Rust frameworku pro tvorbu pluginů s podporou JSX-podobné syntaxe, který výrazně zjednodušuje vývoj rozšíření.

Výsledky benchmarkingu ukazují vynikající výkon Rust implementace pluginového systému, která dosahuje až 200 000 fps při zpracování virtuálního DOM. Práce také identifikuje omezení současné implementace a navrhuje směry budoucího vývoje, včetně plné integrace pluginů do hlavní aplikace, implementace multi-cluster řešení a pokročilých algoritmů pro personalizaci

obsahu.

Obsah

Kapitola 1

Úvod

1.1 Analýza potřeby nové sociální platformy

Sociální média se stala nedílnou součástí každodenního života. Při pohledu na čísla - přes 5,24 miliardy aktivních uživatelů po celém světě tráví v průměru více než dvě hodiny denně na těchto platformách. Je zřejmé, že jde o významný fenomén s rozsáhlým dopadem na společnost. Tato kapitola analyzuje potřebu vývoje nové sociální platformy v kontextu současných trendů, technologických možností a nedostatků existujících řešení.

1.1.1 Současný stav sociálních platforem

Ekosystém sociálních médií prochází v roce 2025 významnou transformací. Podle nejnovějších dat (1) průměrný uživatel internetu využívá téměř 7 různých sociálních platforem měsíčně. Tato fragmentace ukazuje na rostoucí specializaci platforem a diverzifikaci uživatelských potřeb v digitálním prostoru.

Při analýze současných trendů lze identifikovat několik klíčových změn. Algoritmy pro distribuci obsahu procházejí rekalibrací - platformy již nehodnotí úspěch pouze podle počtu zobrazení, ale zaměřují se více na kvalitu interakcí a udržení uživatelů. Tento posun odráží rostoucí konkurenci o pozornost uživatelů(2).

Zároveň dochází k saturaci informačního prostoru obsahem generovaným umělou inteligencí, což vytváří nové výzvy pro udržení autenticity. Uživatelé stále více preferují autentický obsah a transparentní komunikaci. Úspěšní tvůrci již nestaví svou strategii primárně na objemu obsahu, ale na analyticky podloženém přístupu k tvorbě relevantního obsahu pro konkrétní cílové skupiny.

Tyto změny vytvářejí prostor pro nové přístupy k návrhu platforem, které by lépe reflektovaly měnící se preference uživatelů a řešily nedostatky současných platforem v oblastech autenticity, personalizace a kontroly nad obsahem.

1.1.2 Problematika obchodně orientovaného přístupu

Při analýze současných sociálních sítí lze identifikovat, alespoň z uživatelského hlediska, důležitý aspekt. Dominantní platformy jsou primárně orientovány na obchodní zájmy, nikoliv na potřeby uživatelů. Toto se projevuje v několika klíčových aspektech.

Uzavřené algoritmy fungují jako netransparentní mechanizmy, které neposkytují dostatečnou kontrolu nad konzumovaným obsahem. Jsou navrženy tak, aby maximalizovaly zapojení uživatelů a dobu strávenou na platformě, což vede k vytváření „filtračních bublin“ (3) a podporuje informační války (IW) (4).

Dalším problémem je absence možností personalizace. Uživatelé mají minimální kontrolu nad vzhledem a funkcionalitou rozhraní, což omezuje schopnost přizpůsobit platformu vlastním potřebám. Vzhledem k dynamičnosti sociálních platforem je to významný problém. Při implementaci nových funkcí jsou uživatelé nuceni přijmout všechny změny bez možnosti zachovat preferované aspekty předchozích verzí. Tato absence kontroly představuje dlouhodobý problém pro vztah mezi platformou a jejími uživateli.

Ekonomický model současných platforem je často nevýhodný pro tvůrce obsahu. Platformy si běžně účtují až 45% z příjmů tvůrců a neposkytují dynamické modely podporující jejich růst. Například YouTube si ponechává 45%

ze všech přímých příjmů tvůrců, což limituje potenciál kreativní ekonomiky a může vést k odlivu talentů na alternativní platformy(5).

1.1.3 Technologické nedostatky současných řešení

Z technologického hlediska vykazují současné sociální platformy několik významných nedostatků. Chybí jim kvalitní multiplatformní podpora, což brání konzistentní uživatelské zkušenosti na různých zařízeních. Webová rozhraní jsou často neoptimalizovaná, primárně přizpůsobená mobilním zařízením bez adekvátního využití možností desktopových prohlížečů.

V oblasti bezpečnosti a autentizace zůstávají dominantní platformy pozadu. Zastaralé autentizační mechanizmy založené primárně na heslech představují bezpečnostní riziko a zhoršují uživatelskou zkušenost. Absence podpory modernějších přístupů, jako jsou Passkeys, které nabízejí vyšší úroveň zabezpečení při současném zjednodušení procesu přihlašování, ukazuje na technologickou stagnaci v této oblasti.

Ochrana zranitelných skupin, zejména dětí, je řešena nedostatečně. Implementované mechanizmy se často omezují pouze na základní omezení času bez sofistikovanějších přístupů k ochraně před nevhodným obsahem. Tento přístup neodpovídá rostoucímu důrazu na digitální wellbeing a bezpečnost online prostředí.

V případech, kdy platformy implementují základní filtrační mechanizmy pro ochranu mladších uživatelů, jako v případě YouTube Kids, dochází k fragmentaci uživatelského prostředí formou oddělených aplikací bez adekvátní integrace s hlavní platformou. Specializované verze mají významná omezení v personalizaci obsahu a chybí jim mechanizmy pro zachování preferencí při přechodu mezi prostředími. Absence graduálního přechodového modelu mezi platformami pro různé věkové kategorie představuje významný problém zejména pro adolescentní uživatele, kteří přerůstají dětské platformy, ale zároveň nejsou dostatečně chráněni před potenciálně nevhodným obsahem v prostředí pro dospělé.

1.1.4 Souhrn provedené analýzy

Na základě provedené analýzy lze dospět k závěru, že existuje významný prostor pro vývoj nové sociální platformy, která by řešila identifikované nedostatky současných řešení. Kombinace uživatelsky orientovaného přístupu, pokročilých architektonických principů, moderních autentizačních mechanismů a personifikovatelné platformy představuje slibný směr pro implementaci takové platformy.

1.2 Cíle práce

Hlavním cílem této práce je navrhnout a implementovat robustní sociální platformu s integrovaným ekosystémem pluginů, která řeší identifikované nedostatky současných řešení. Práce se zaměřuje na vytvoření komplexní architektury, která bude reflektovat aktuální technologické trendy a zároveň poskytne uživatelsky orientovaný přístup k sociálním interakcím v digitálním prostoru.

1.2.1 Specifikace cílů

Primárním cílem je vyvinout modulární architekturu sociální platformy, která umožní flexibilní rozšiřitelnost prostřednictvím pluginového systému založeného na technologii WebAssembly. Tato architektura je navrhována s ohledem na škálovatelnost, bezpečnost a udržitelnost kódu.

- Sekundárním cílem je implementace pokročilých autentizačních mechanismů s podporou standardu FIDO2 (6) a technologie Passkeys.
- Třetím cílem je vytvoření multiplatformního řešení s optimalizovaným uživatelským rozhraním pro různá zařízení, které poskytne konzistentní uživatelskou zkušenost napříč desktopovými i mobilními platformami.
- Čtvrtým cílem je návrh a implementace transparentních algoritmů pro distribuci obsahu, které uživatelům poskytnou větší kontrolu nad kon-

zumovaným obsahem a umožní personalizaci informačního toku podle individuálních preferencí.

- Pátým cílem je vytvoření graduálního přechodového modelu pro různé věkové kategorie uživatelů, který zajistí adekvátní ochranu zranitelných skupin při zachování možností personalizace a uživatelské svobody.
- Šestým cílem je implementace ekonomického modelu podporujícího tvůrce obsahu prostřednictvím spravedlivějšího rozdělení příjmů a dynamických nástrojů pro podporu růstu komunity.

Posledním cílem je evaluace vytvořeného řešení z hlediska uživatelské zkušenosti, technické efektivity a potenciálu pro dlouhodobou udržitelnost v dynamicky se měnícím prostředí sociálních médií.

Dosažení těchto cílů by mělo vést k vytvoření sociální platformy, která nejen řeší současné nedostatky existujících řešení, ale také poskytuje flexibilní základ pro budoucí inovace v oblasti sociálních interakcí v digitálním prostoru.

1.3 Struktura práce

Tato práce je strukturována do osmi hlavních kapitol, které systematicky pokrývají celý proces návrhu a implementace robustní sociální platformy s ekosystémem pluginů.

Úvodní kapitola představuje problematiku sociálních platforem v současném digitálním prostředí. Analyzuje potřebu nové sociální platformy v kontextu identifikovaných nedostatků existujících řešení a stanovuje cíle práce, které reflektují ambici vytvořit uživatelsky orientovanou sociální platformu s důrazem na modularitu a rozšiřitelnost.

Druhá kapitola se věnuje detailní analýze trhu a existujících řešení, přičemž definuje technologické předpoklady a rámec projektu. Zvláštní pozornost je věnována technologickému stacku zahrnujícímu Kubernetes a na ně na-

vazující další technologie. Kapitola rovněž specifikuje funkční a nefunkční požadavky, které formují základ pro následný návrh architektury.

Třetí kapitola představuje komplexní návrh architektury systému. Začíná celkovým přehledem architektury s důrazem na klientskou a serverovou část, pokračuje popisem mikroservisní architektury implementované na platformě Kubernetes a věnuje se komunikačním vrstvám, bezpečnostním aspektům a databázovému řešení. Diskutuje také limitace zvoleného prostředí.

Čtvrtá kapitola se zaměřuje na detailní implementaci klíčových komponent systému. Popisuje správu klíčů pomocí HashiCorp Vault, implementaci autentizačních mechanismů a bezpečnostních prvků, strukturu mikroservisního ekosystému a implementaci frontendové aplikace v React Native Expo a Next.js.

Pátá kapitola je věnována ekosystému uživatelských pluginů, který představuje jeden z hlavních inovativních prvků navrhovaného řešení. Popisuje aplikační vrstvu včetně verzovací API a mechanismů pro registraci a zpracování událostí, pluginovou vrstvu se zaměřením na sandboxing u nativních zařízení a implementaci uživatelského rozhraní s důrazem na dynamické renderování komponentů.

Šestá kapitola se zabývá monitoringem a benchmarkingem implementovaného systému, přičemž popisuje použité nástroje a metodiky pro monitorování výkonu a stability platformy.

Sedmá kapitola přináší diskuzi a vyhodnocení dosažených výsledků. Hodnotí míru naplnění stanovených cílů, porovnává implementované řešení s existujícími alternativami a identifikuje omezení současné implementace spolu s návrhy na budoucí rozvoj.

Závěrečná kapitola shrnuje klíčové poznatky získané během realizace projektu a formuluje doporučení pro budoucí výzkum a praxi v oblasti vývoje sociálních platforem.

Práce je doplněna přílohami obsahujícími diagramy architektury, ukázky kódu, konfigurační soubory a další relevantní dokumentaci, které poskytují detailnější vhled do technických aspektů implementovaného řešení.

Kapitola 2

Analýza a předpoklady

2.1 Analýza trhu a existujících řešení

Na rozdíl od obecné analýzy v úvodní kapitole se tato sekce zaměřuje na konkrétní technické implementace existujících sociálních platforem a jejich architektonické přístupy. Cílem je identifikovat specifické technické aspekty, které lze vylepšit v navrhovaném řešení.

2.1.1 Technické implementace dominantních platforem

Současné sociální platformy využívají různé architektonické přístupy k řešení problémů škálovatelnosti, bezpečnosti a uživatelské zkušenosti. Meta (Facebook, Instagram) implementovala rozsáhlou mikroservisní architekturu s proprietárními řešeními pro škálování a distribuci zátěže. Společnost vyvinula vlastní systém pro správu kontejnerů podobný Kubernetes a specializované nástroje pro monitorování výkonu. Z hlediska autentizace však platforma spoléhá primárně na tradiční hesla doplněná dvoufaktorovou autentizací, bez plné podpory moderních standardů jako FIDO2. (7)

X (dříve Twitter) prošel významnou architektonickou transformací, kdy původní monolitickou aplikaci nahradil mikroservisní architekturou. Platforma využívá kombinaci proprietárních a open-source technologií pro správu infrastruktury. Významným aspektem architektury X je důraz na real-time

zpracování dat, což vytváří specifické výzvy pro škálování a konzistenci. (8)

TikTok představuje moderní přístup s důrazem na efektivní distribuci video obsahu a pokročilé algoritmy pro personalizaci. Platforma využívá cloud-native přístup a proprietární řešení pro zpracování multimediálního obsahu. Z bezpečnostního hlediska však TikTok čelí kritice kvůli nedostatečné transparentnosti zpracování uživatelských dat. (9)

2.2 Technologické předpoklady a rámec projektu

V návaznosti na identifikované technologické mezery existujících platforem tato kapitola představuje technologický stack využitý při implementaci navrhovaného řešení. Zvolené technologie byly pečlivě vybrány s ohledem na jejich schopnost adresovat zjištěné nedostatky a poskytnout robustní základ pro moderní sociální platformu s dynamickým pluginovým systémem.

Zatímco dominantní platformy jako Meta a Twitter vyvinuly proprietární řešení pro správu kontejnerů a mikroslužeb, navrhované řešení staví na standardizovaných open-source technologiích, které umožňují větší flexibilitu a transparentnost. Namísto vytváření uzavřených ekosystémů se zaměřuje na implementaci otevřené architektury, která podporuje rozšiřitelnost a interoperabilitu.

2.2.1 Použité technologie

Kubernetes slouží jako platforma pro orchestraci kontejnerizovaných aplikací. Umožňuje efektivní škálování, automatizovanou správu a nasazení jednotlivých mikroslužeb. Mezi hlavní přínosy patří robustnost a flexibilita správy clusteru. V rámci implementace jsou využívány pokročilé funkce jako Ingress pro směrování provozu, pravidla síťové komunikace (Network Policies), tajemství (Secret) a konfigurační mapy (ConfigMaps) pro správu konfigurace a manažer certifikátů pro automatickou správu SSL certifikátů. Volba Kuber-

netes oproti alternativám jako Docker Swarm byla učiněna především kvůli potřebě vyšší škálovatelnosti a robustnějšího ekosystému pro správu mikroslužeb, které jsou klíčové pro architekturu moderní sociální sítě.

Consul je nástroj určený pro správu, registraci a konfiguraci mikroservis. Díky implementaci Service Mesh lze dosáhnout bezpečné komunikace mezi jednotlivými službami prostřednictvím mTLS. V rámci projektu je Consul využíván ve třech klíčových rolích: jako API Gateway pro centralizovanou správu přístupu k API, jako Service Mesh pro zabezpečenou komunikaci mezi službami s využitím Proxy Defaults a jako Terminating Gateway pro bezpečnou komunikaci s externími službami. Volba Consulu oproti alternativám jako Istio byla provedena na základě jeho nižších nároků na hardwarové zdroje při zachování klíčové funkcionality.

Jako relační databázový systém je využíván PostgreSQL s nadstavbou Spillo a Patroni pro zajištění vysoké dostupnosti. Tato kombinace umožňuje automatickou replikaci dat a failover v případě výpadku primárního uzlu. Výhody spočívají v robustnosti, podpoře pokročilých dotazovacích mechanismů a možnosti využití postgresql-operatoru pro zajištění vysoké dostupnosti. Spillo byl zvolen pro jeho jednoduchost při konfiguraci Patroni s PostgreSQL a nastavení databázových připojení.

HashiCorp Vault slouží k bezpečné správě tajemství a šifrovacích klíčů. Umožňuje dynamickou distribuci a bezpečné ukládání citlivých dat, čímž minimalizuje riziko jejich úniku. V rámci projektu je Vault plánován jako klíčový komponent pro správu šifrovacích klíčů a zabezpečení citlivých dat. Jedná se o jediné self-hosted řešení pro správu klíčů (KMS), které je dobře integrováno s Consulem a poskytuje komplexní možnosti pro zabezpečení dat v mikroservisním prostředí.

Pro vývoj mobilních aplikací byl zvolen framework React Native Expo, který umožňuje rychlý vývoj multiplatformních aplikací. Expo podporuje snadnou integraci s nativními funkcemi zařízení a zajišťuje konzistentní uživatelský zážitek napříč různými operačními systémy. Pro webovou část aplikace je využíván Next.js jako moderní React framework poskytující optimální vývojové

prostředí a výkonnostní optimalizace. Volba těchto technologií byla učiněna na základě jejich multiplatformního charakteru a synergie s existujícími znalostmi React ekosystému. V rámci implementace byly vyvinuty vlastní nativní moduly v Kotlinu pro podporu WebAssembly a Passkey autentizace, které nejsou standardně součástí React Native Expo.

WebAssembly je technologie umožňující spouštění vysoce výkonného kódu v sandboxovaném prostředí. V projektu je využívána pro implementaci dynamického pluginového systému, který zajišťuje modularitu a bezpečné rozšíření funkčnosti. WASM byl zvolen především pro jeho multiplatformní povahu a robustní bezpečnostní model, který zajišťuje, že spouštění pluginů třetích stran neohrozí stabilitu a bezpečnost aplikace.

Pro efektivní distribuci statického obsahu je využívána síť Cloudflare R2 CDN, která poskytuje globální síť pro rychlé doručování mediálního obsahu. Toto řešení významně snižuje latenci při načítání obrázků a videí a zároveň redukuje zátěž na backend infrastrukturu. Cloudflare R2 nabízí také ochranu proti DDoS útokům a efektivní caching mechanizmy, které optimalizují náklady na provoz.

Pro zajištění komplexního monitorování a logování je implementován stack Grafana, Loki a Prometheus běžící v Kubernetes clusteru mimo Consul Service Mesh. Tento stack poskytuje robustní nástroje pro sledování výkonu, detekci anomálií a analýzu logů, což je klíčové pro udržení stability a výkonu komplexního mikroservisního prostředí.

2.2.2 High Availability (HA)

Vysoká dostupnost představuje klíčový aspekt navrhované architektury, zajišťující nepřetržitý provoz systému i v případě selhání jednotlivých komponent. V kontextu implementovaného řešení zahrnuje HA několik úrovní redundance. Na úrovni mikroslužeb je vysoká dostupnost zajištěna prostřednictvím Consul Service Mesh, který umožňuje implementaci Mesh Gateway pro komunikaci mezi více clustery. Tento přístup zajišťuje efektivní škálování mikroslužeb uvnitř Service Mesh. Na úrovni databáze je implementována vy-

soká dostupnost pomocí Spillo s Patroni a PostgreSQL, což zajišťuje automatickou replikaci dat a mechanismus hlasování pro určení master uzlu v případě výpadku. Na úrovni infrastruktury je využíván Kubernetes pro automatické zotavení po výpadku a redistribuci zátěže mezi dostupnými uzly. Tento vícevrstvý přístup k vysoké dostupnosti je nezbytný pro zajištění spolehlivého provozu globálně dostupné sociální sítě s předpokládaným vysokým počtem současně připojených uživatelů.

Zvolený technologický stack představuje vyvážený kompromis mezi robustností, flexibilitou a bezpečností. Každá z uvedených technologií byla pečlivě vybrána s ohledem na přínos k dosažení cílů projektu, tj. vytvoření moderní, škálovatelné a bezpečné sociální sítě. Přestože implementace takto komplexního technologického stacku představuje značnou výzvu z hlediska učební křivky a integrace jednotlivých komponent, výsledné řešení poskytuje solidní základ pro realizaci inovativní sociální platformy s důrazem na bezpečnost, škálovatelnost a rozšiřitelnost.

2.3 Funkční a nefunkční požadavky

Na základě analýzy trhu a identifikovaných technologických mezer byly stanoveny následující funkční a nefunkční požadavky pro navrhovanou sociální platformu.

2.3.1 Funkční požadavky

Implementace robustního pluginového systému představuje klíčový funkční požadavek navrhované platformy. Systém musí umožnit rozšíření funkcionality prostřednictvím WebAssembly, zajistit bezpečné spouštění kódu v sandboxovaném prostředí a poskytnout standardizované API pro interakci s platformou. Pluginový systém by měl podporovat dynamické uživatelské rozhraní, registraci a zpracování událostí a verzování API pro zajištění dlouhodobé kompatibility.

Autentizační mechanizmy musí zahrnovat podporu moderních standardů,

zejména Passkeys využívajících FIDO2 protokol, což eliminuje potřebu tradičních hesel. Implementace QR kódu pro přihlášení představuje další požadavek, který zjednoduší proces autentizace napříč zařízeními. Tyto mechanizmy musí být implementovány s důrazem na bezpečnost a uživatelskou přívětivost.

Správa obsahu vyžaduje implementaci systému pro sdílení a ukládání multimediálního obsahu, včetně obrázků a textových příspěvků. Systém musí podporovat různé formáty obsahu a zajistit efektivní distribuci prostřednictvím CDN. Transparentní algoritmy pro personalizaci zobrazovaného obsahu s možností uživatelské kontroly představují další klíčový požadavek.

Sociální interakce musí být implementovány prostřednictvím systému uživatelských kruhů (circles), který umožní vytváření sociálních vazeb mezi uživateli. Tento systém musí podporovat různé typy vztahů, včetně přátelství, sledování a skupinových interakcí, a zajistit granulární kontrolu soukromí pro různé typy sociálních vazeb. Uživatelé musí mít možnost přidávat jiné uživatele do specifických kruhů a přizpůsobovat viditelnost obsahu pro jednotlivé kruhy.

2.3.2 Nefunkční požadavky

Škálovatelnost systému představuje zásadní nefunkční požadavek. Architektura musí být schopna horizontálního škálování pro podporu rostoucího počtu uživatelů a objemu dat. Implementace na platformě Kubernetes musí umožnit efektivní distribuci zátěže a optimalizaci výkonu při špičkách.

Bezpečnost vyžaduje implementaci komplexních opatření, včetně mTLS pro zabezpečenou komunikaci mezi mikroslužbami, šifrování dat v klidu i při přenosu a ochrany proti běžným typům útoků. Autentizační mechanizmy musí splňovat nejnovější bezpečnostní standardy a poskytovat robustní ochranu uživatelských účtů.

Observabilita systému musí být zajištěna prostřednictvím implementace monitorovacích a logovacích nástrojů. Tyto nástroje musí poskytovat komplexní přehled o výkonu a stavu systému, umožnit detekci anomálií a efektivní diagnostiku problémů.

Uživatelská zkušenost musí být konzistentní a optimalizovaná napříč různými

zařízeními a platformami. Implementace responzivního designu a optimalizace pro různé velikosti obrazovek a typy interakcí představují klíčové požadavky v této oblasti. Systém musí poskytovat intuitivní rozhraní pro správu uživatelských kruhů a personalizaci obsahu.

Dostupnost systému musí dosahovat vysoké úrovně (99,9% a vyšší) prostřednictvím redundance, automatického zotavení po výpadku a geografické distribuce služeb. Implementace strategie pro minimalizaci dopadu plánovaných údržbových prací představuje další požadavek v této oblasti.

Kapitola 3

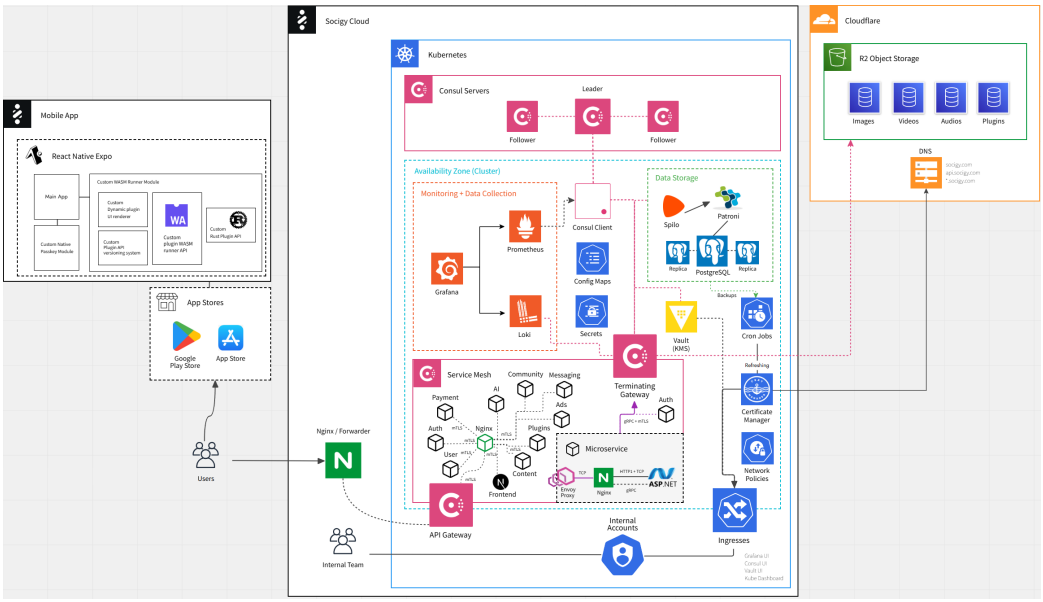
Návrh architektury systému

Tato kapitola představuje komplexní návrh architektury systému sociální platformy, která implementuje identifikované požadavky a řeší nedostatky současných řešení. Architektura je navržena s důrazem na modularitu, škálovatelnost a bezpečnost, přičemž využívá moderní technologické přístupy popsané v předchozí kapitole. Platformu byla pojmenována Socigy.

3.1 Celkový přehled architektury

Navržená architektura sociální platformy Socigy představuje komplexní ekosystém vzájemně propojených komponent, které společně tvoří robustní základ pro provoz moderní sociální sítě, jak je znázorněno na obrázku 3.1. Architektura je koncipována jako distribuovaný systém skládající se ze tří hlavních částí: klientské aplikace, cloudové infrastruktury a externích služeb pro ukládání a distribuci obsahu.

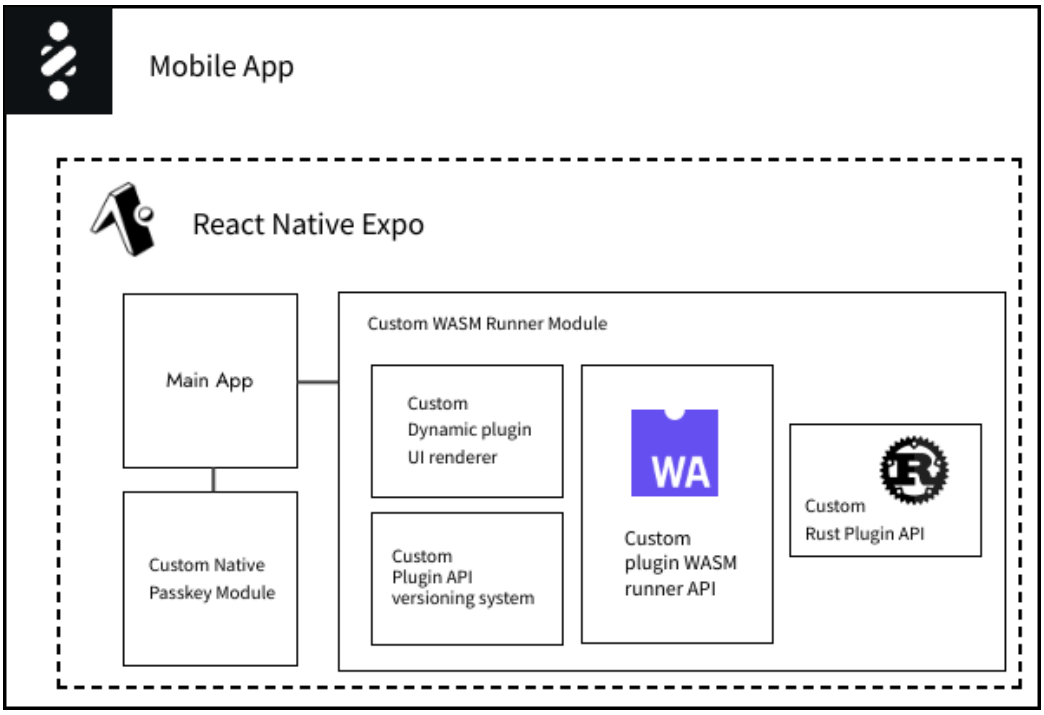
Z hlediska topologie je systém navržen jako vícevrstvá architektura, kde jednotlivé vrstvy mají jasně definované odpovědnosti a rozhraní. Tento přístup umožňuje nezávislý vývoj, testování a nasazení jednotlivých komponent, což zvyšuje agilitu vývojového procesu a usnadňuje údržbu systému.



Obrázek 3.1: Přehled celkové architektury

3.1.1 Přehled klientské strany

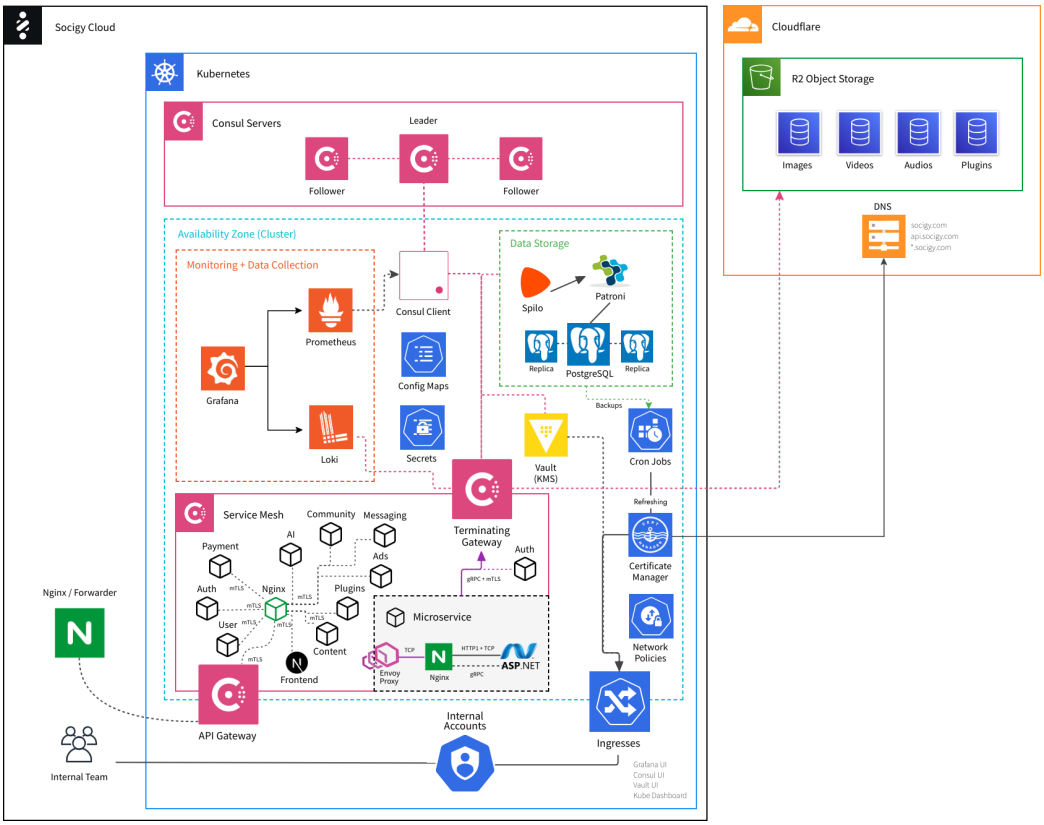
Klientská část architektury, zobrazená na obrázku 3.2, je reprezentována dvěma hlavními komponentami: mobilní aplikací vyvinutou s využitím frameworku React Native Expo a webovou aplikací implementovanou pomocí Next.js.



Obrázek 3.2: Přehled klientské architektury

3.1.2 Přehled serverové části

Serverová část architektury, označená jako "Socigy Cloud"a znázorněná na obrázku 3.3, představuje jádro celého systému a je implementována na platformě Kubernetes.



Obrázek 3.3: Přehled serverové architektury

3.2 Mikroservisní architektura na platformě Kubernetes

Mikroservisní architektura rozděluje komplexní systém do menších, nezávisle nasaditelných služeb komunikujících přes jasně definovaná rozhraní. Zvolený přístup umožňuje nezávislý vývoj, testování a nasazení jednotlivých komponent, což výrazně zvyšuje agilitu vývojového procesu a usnadňuje údržbu systému. V rámci implementace bylo identifikováno několik klíčových domén reprezentovaných samostatnými mikroslužbami:

- Autentizace
- Správa uživatelů

- Správa obsahu
- Pluginy
- Umělá inteligence
- Messaging
- Komunity
- Reklamy
- Platby

Kubernetes poskytuje komplexní sadu nástrojů pro orchestraci mikroslužeb. Požadovaný stav systému je definován pomocí deklarativních manifestů specifikujících počet replik, požadavky na výpočetní zdroje, síťové politiky a další konfigurační parametry. Zvolená metoda umožňuje verzování infrastruktury jako kódu a zajišťuje transparentní mechanismus pro správu změn.

Pro zajištění izolace a správu výpočetních zdrojů jsou využívány Kubernetes namespaces poskytující logické oddělení jednotlivých částí systému. Implementace umožňuje efektivní správu přístupových oprávnění a nastavení síťových politik.

Konfigurace mikroslužeb je realizována prostřednictvím ConfigMaps a Secrets, které zajišťují správu konfiguračních parametrů a citlivých informací. ConfigMaps slouží pro běžné konfigurační parametry, zatímco Secrets ukládají citlivé informace jako přístupové údaje k databázi nebo API klíče. Implementovaný přístup odděluje konfiguraci od kódu a umožňuje dynamickou aktualizaci bez nutnosti rekompilace a nasazení nových verzí aplikací.

Vysoká dostupnost a odolnost vůči výpadkům je zajištěna strategií rozložení zátěže napříč více uzly v rámci Availability Zone. Kubernetes automaticky distribuuje instance mikroslužeb mezi dostupné uzly a při výpadku jednoho uzlu přesouvá postižené instance na funkční uzly, čímž minimalizuje dopad na dostupnost služby.

Správa nasazení nových verzí mikroslužeb je realizována strategií rolling update, která postupně nahrazuje běžící instance novými verzemi bez výpadku služby. Implementace minimalizuje riziko spojené s nasazením nových verzí a umožňuje rychlé zotavení při problémech.

Pro monitorování stavu a výkonu mikroslužeb jsou implementovány readiness a liveness probes, které Kubernetes využívá k detekci nefunkčních instancí a jejich automatickému restartování. Readiness probes určují připravenost nově nasazené instance přijímat provoz, zatímco liveness probes detekují instance v nekonzistentním stavu vyžadující restart.

3.3 Komunikační vrstvy

Efektivní komunikace mezi komponentami systému představuje klíčový aspekt navržené architektury. Jak ukazuje obrázek 3.1, systém využívá specializované komunikační vrstvy zajišťující bezpečnou, spolehlivou a výkonnou výměnu dat.

Pro komunikaci mezi mikroslužbami v Service Mesh je primárně využíván protokol gRPC, který poskytuje vysoký výkon, nízkou latenci a podporu streamování dat. Tento protokol, založený na HTTP/2, umožňuje efektivní serializaci strukturovaných dat pomocí Protocol Buffers a podporuje bidirekcionální streamování, což je zásadní pro implementaci real-time funkcí sociální platformy. Výhodou gRPC je také generování klientských a serverových rozhraní z definičních souborů, což výrazně zjednodušuje vývoj a údržbu API.

Consul Service Mesh je implementován prostřednictvím Envoy proxy nasazených jako sidecary vedle každé instance mikroslužby. Tento přístup, označovaný jako sidecar pattern, umožňuje transparentní implementaci síťových funkcí bez nutnosti modifikace kódu mikroslužeb. Komunikace mezi mikroslužbami prochází přes Envoy proxy, která zajišťuje směrování, load balancing, circuit breaking a další pokročilé síťové funkce.

API Gateway, implementovaná pomocí Consul API Gateway, slouží jako

vstupní bod pro externí požadavky a směruje je na příslušné mikroslužby. Tato komponenta podporuje různé protokoly včetně HTTP/1.1, HTTP/2 a gRPC, což umožňuje efektivní komunikaci s různými typy klientů.

Pro zabezpečenou komunikaci s externími službami mimo Service Mesh je implementována Terminating Gateway. Ta zajišťuje terminaci mTLS a překlad mezi zabezpečenou komunikací uvnitř Service Mesh a potenciálně nezabezpečenou komunikací s externími systémy. Toto řešení umožňuje mikroslužbám bezpečně komunikovat s externími API, databázemi nebo legacy systémy.

Pro podporu real-time komunikace mezi klienty a serverem je využívána technologie SignalR, která poskytuje vysokoúrovňové abstrakce pro WebSockets, Server-Sent Events a Long Polling. Tato technologie umožňuje efektivní implementaci notifikací, chatů a dalších real-time funkcí s minimálními nároky na vývoj.

Směrování síťového provozu uvnitř Kubernetes clusteru mimo Service Mesh je realizováno prostřednictvím Kubernetes DNS poskytujícího service discovery mechanismus. Každá služba registrovaná v Kubernetes je dostupná přes DNS záznam ve formátu `<service-name>.<namespace>.svc.cluster.local`, což umožňuje transparentní komunikaci mezi službami bez znalosti fyzické lokace.

3.4 Bezpečnost nasazeného ekosystému

Bezpečnost představuje kritický aspekt navržené architektury, zejména u sociální platformy zpracovávající citlivá uživatelská data. Implementovaný bezpečnostní model zahrnuje několik vrstev ochrany.

Zabezpečení komunikace je realizováno prostřednictvím mutual TLS (mTLS), kdy komunikující strany vzájemně ověřují identitu pomocí certifikátů. Consul Service Mesh automaticky zajišťuje vydávání, distribuci a rotaci certifikátů pro služby. Tento mechanismus efektivně brání odposlouchávání komunikace a man-in-the-middle útokům, což je zásadní pro ochranu citlivých

uživatelských dat.

Access Control Lists (ACL) jsou implementovány na úrovni Consulu pro řízení přístupu ke službám a jejich API. Tento mechanismus umožňuje granulární definici oprávnění a zajišťuje, že každá služba má přístup pouze k nezbytným zdrojům.

Intentions v Service Mesh definují povolené komunikační cesty mezi službami. Je zde implementován princip nejnižších oprávnění (princip least privilege), kdy je komunikace mezi službami povolena pouze při explicitní definici.

Pro správu tajemství a šifrovacích klíčů je využíván HashiCorp Vault (KMS), který poskytuje centralizované úložiště pro citlivé informace s mechanizmy pro řízení přístupu a audit. Tato komponenta je klíčová pro implementaci šifrování dat v klidu i při přenosu a bezpečnou správu autentizačních údajů.

Certificate Manager zajišťuje automatickou správu SSL certifikátů v Kubernetes clusteru. Tento nástroj eliminuje rizika spojená s manuální správou certifikátů a zajišťuje šifrování veškeré externí komunikace pomocí platných certifikátů.

Network Policies definují povolené komunikační cesty na úrovni síťové vrstvy, čímž poskytují další úroveň izolace a ochrany proti laterálnímu pohybu při kompromitaci některé komponenty systému.

3.5 Limitace prostředí

Navržená architektura sociální platformy Socigy vykazuje přes svou robustnost několik limitací, které bude nutné adresovat v kontextu dlouhodobého vývoje a škálování systému.

Hlavní technologickou výzvou současné implementace je chybějící nativní podpora multi-port konfigurace ve stabilních verzích Consul Service Mesh. Tato limitace se projevuje zejména při implementaci komplexních komunikačních vzorů, kdy mikroslužby potřebují současně podporovat různé komunikační protokoly (gRPC, HTTP/1.1, SignalR) bez přímé implementace

HTTPS na úrovni aplikačního kontejneru.

K překonání této bariéry bylo nutné integrovat Nginx proxy server jako doplňkovou komponentu v každém kontejneru mikroslužby. Nginx vystavuje jeden port viditelný pro Envoy proxy a interně zajišťuje směrování požadavků mezi různými porty určenými pro specifické komunikační protokoly. Toto řešení sice efektivně obchází základní limitaci Service Mesh, ale zároveň zvyšuje komplexitu nasazení a vytváří dodatečnou výpočetní režii.

Další významnou limitací současné implementace je absence Mesh Gateway pro usnadnění komunikace mezi více Kubernetes clustery. Aktuální architektonická konfigurace omezuje nasazení systému na jeden cluster (Availability Zone), což výrazně limituje možnosti geografické distribuce a představuje riziko z hlediska vysoké dostupnosti při katastrofickém selhání datového centra.

Implementace Consul Service Mesh přirozeně generuje určitou výpočetní režii vyplývající z nutnosti provozovat Envoy proxy paralelně s každou instancí mikroslužby. Tato režie může být významná zejména v prostředích s omezenými výpočetními zdroji nebo při nasazení velkého počtu mikroslužeb s minimálními požadavky na výpočetní výkon.

Integrace s externími systémy prostřednictvím Terminating Gateway představuje potenciální výkonnostní úzké místo při zvýšeném objemu komunikace s externími službami. Aktuální implementace neumožňuje automatické škálování Terminating Gateway na základě dynamického zatížení, což vyžaduje kontinuální monitoring a ruční úpravy konfigurace.

Architektura byla primárně optimalizována pro nasazení v cloudovém prostředí. Migrace do on-premise infrastruktury nebo hybridního řešení by mohla vyžadovat významné úpravy zejména v oblastech automatického škálování, rozložení zátěže a service discovery.

3.6 Databázové řešení

Perzistenci dat v sociální platformě Socigy zajišťuje robustní databázové řešení založené na PostgreSQL s využitím Spillo a Patroni pro implementaci vysoké dostupnosti. Jak je znázorněno na obrázku 3.3, databázová architektura je navržena s důrazem na spolehlivost, výkon a odolnost vůči výpadkům.

PostgreSQL byl zvolen jako primární databázový systém díky pokročilým funkcím, včetně podpory komplexních datových typů, indexování pomocí GIN a GiST indexů, které jsou zásadní pro efektivní vyhledávání v sociálních datech, a robustnímu transakčnímu modelu zajišťujícímu integritu dat. Schéma databáze bylo navrženo s ohledem na specifické požadavky sociální platformy, zahrnující entity jako uživatelé, kruhy (circles), vztahy mezi uživateli a zprávy, což umožňuje efektivní reprezentaci sociálních vazeb a interakcí.

Pro zajištění vysoké dostupnosti je implementována architektura Master-Slave s využitím streamovací replikace, která zajišťuje kontinuální replikaci dat z primárního uzlu na replikační uzly. Konfigurace zahrnuje jeden primární uzel (Master) a dva replikační uzly (Replica), což poskytuje redundanci dat a možnost automatického failoveru při výpadku primárního uzlu.

Klíčovou komponentou pro správu architektury je Patroni, specializovaný nástroj pro orchestraci PostgreSQL clusterů. Patroni implementuje mechanizmy pro monitoring stavu databázových uzlů, detekci výpadků a automatickou volbu nového primárního uzlu z dostupných replik. Proces failoveru minimalizuje dobu výpadku při selhání primárního uzlu a zajišťuje kontinuitu služeb.

Spillo, PostgreSQL operátor pro Kubernetes, poskytuje integrační vrstvu mezi databázovým clusterem a Kubernetes ekosystémem. Komponenta zajišťuje automatizovanou správu databázových clusterů, včetně provisioningu, zálohování, obnovy a škálování.

Pro zajištění konzistence dat v distribuovaném prostředí byly implementovány transakční mechanizmy a strategie pro řešení konfliktů. Databázové schéma využívá pokročilé funkce PostgreSQL, jako jsou rozšíření uuid-ossp pro generování unikátních identifikátorů a pgtrgm pro efektivní fulltextové

vyhledávání.

Pravidelné zálohování a údržba databáze jsou zajištěny prostřednictvím Kubernetes Cron Jobs. Konfigurace databáze je spravována pomocí Kubernetes Config Maps, což umožňuje centralizovanou správu konfiguračních parametrů a jejich dynamickou aktualizaci bez nutnosti restartu databázových instancí.

Kapitola 4

Detail implementace klíčových komponent

4.1 Autentizace a bezpečnost

Implementace autentizačních mechanismů a bezpečnostních opatření je klíčovou součástí návrhu sociální platformy Socigy. Systém využívá moderní autentizační metody s důrazem na zajištění bezpečnosti uživatelských dat při současném zachování intuitivního uživatelského rozhraní.

4.1.1 Implementace Passkeys

Primárním autentizačním mechanismem jsou Passkeys založené na standardu FIDO2, které poskytují vysokou úroveň zabezpečení při současném zjednodušení procesu přihlašování. Na webové platformě je funkcionalita realizována prostřednictvím WebAuthn API, což umožňuje přímou integraci s biometrickými senzory zařízení a bezpečnostními klíči. (10)

Pro mobilní aplikaci vyvinutou v React Native Expo bylo nezbytné implementovat vlastní nativní modul pro platformu Android. Serverová část autentizačního systému využívá specializovanou knihovnu vyvinutou FIDO Alliance pro validaci autentizačních dat a správu registrovaných zařízení.

4.1.2 QR Code Sign-in

Alternativním autentizačním mechanismem je QR Code sign-in, který se v současné době nachází ve fázi implementace. Tento přístup umožňuje uživatelům přihlášení do webové aplikace prostřednictvím naskenování QR kódu mobilním zařízením, na kterém jsou již autentizováni.

4.1.3 Vícefaktorová autentizace

Významným bezpečnostním prvkem implementovaného systému je povinná vícefaktorová autentizace (MFA), která je aktivována automaticky při registraci nového uživatele. Primárním faktorem je email MFA, kdy systém vyžaduje verifikaci emailové adresy před dokončením registračního procesu. Dále je implementována podpora pro Time-based One-Time Password (TOTP) (11) jako alternativní MFA metoda.

4.1.4 Zařízením orientovaná autentizace

Autentizační systém je orientován na zařízení (device-oriented authentication), což přináší několik zásadních výhod. Uživatelé mohou spravovat svá autentizovaná zařízení prostřednictvím dedikovaného rozhraní v nastavení účtu, včetně možnosti okamžitého odebrání přístupových práv konkrétnímu zařízení. Tento přístup umožňuje implementaci pokročilých bezpečnostních politik na úrovni jednotlivých zařízení a využití autentizovaných zařízení pro sekundární účely, jako je autorizace citlivých operací nebo implementace přihlášení pomocí QR kódu.

4.1.5 Bezpečnostní mechanizmy na úrovni API

Na úrovni API a přenosu dat jsou implementována komplexní bezpečnostní opatření. Autentizační tokeny jsou ukládány v Secure, HttpOnly cookies, což eliminuje riziko jejich odcizení prostřednictvím JavaScript kódu. Pro ochranu proti Cross-Site Request Forgery (CSRF) útokům jsou implementovány Anti-

Forgery tokeny, které zajišťují, že požadavky na API pocházejí z legitimních zdrojů.

Významným bezpečnostním prvkem je implementace Cross-Origin Resource Sharing (CORS), která je realizována individuálně na úrovni každé mikroslužby. Tento decentralizovaný přístup byl zvolen z důvodu rozdílných požadavků na přístupnost dat v rámci jednotlivých služeb. CORS politiky jsou nastaveny tak, aby povolovaly přístup pouze z relevantních a ověřených domén, čímž se minimalizuje riziko neoprávněného přístupu k API. Individuální implementace CORS také umožňuje flexibilní přístup k budoucím rozšířením funkcionality, jako je například možnost embedování příspěvků na externích webových stránkách, kdy specifické endpointy mohou mít méně restriktivní CORS nastavení.

Komunikace mezi klientem a serverem je zabezpečena prostřednictvím TLS/SSL s využitím moderních kryptografických algoritmů. Pro zvýšení bezpečnosti webové aplikace jsou implementovány Content Security Policy (CSP) a další bezpečnostní hlavičky, které poskytují ochranu proti různým typům útoků, včetně Cross-Site Scripting (XSS), clickjacking a data injection. CSP definuje povolené zdroje pro načítání skriptů, stylů, obrázků a dalších typů obsahu, čímž minimalizuje riziko spuštění škodlivého kódu.

4.2 Mikroservisní ekosystém

Implementace mikroservisního ekosystému představuje jádro navržené sociální platformy. Mikroslužby jsou vyvinuty s využitím ASP.NET AOT 8, který poskytuje výkonnostní optimalizace prostřednictvím kompilace Ahead-of-Time. Architektura mikroslužeb je založena na principu Dependency Injection, což umožňuje flexibilní správu závislostí a usnadňuje testování jednotlivých komponent.

Komunikační infrastruktura mikroslužeb je realizována prostřednictvím kombinace několika protokolů - gRPC pro vysokovýkonnou komunikaci mezi službami, SignalR pro real-time komunikaci a standardní HTTP/1.1 pro

REST API. Jádrem komunikační vrstvy je Kestrel server, který poskytuje vysoký výkon a nízkou latenci. Pro řešení omezení Consul Service Mesh v oblasti multi-port konfigurace byl implementován vlastní Dockerfile s integrovaným Nginx, který funguje jako reverzní proxy. HTTP/1.1 a SignalR komunikace je mapována na port 5000, HTTP/2 gRPC na port 5001, zatímco Nginx poslouchá na portu 8080, který je následně využíván Envoy proxy v rámci Service Mesh.

4.2.1 Databázový ORM mapper

Významným aspektem implementace mikroservisního ekosystému je vývoj vlastního PostgreSQL ORM mapperu, který byl nezbytný z důvodu kompatibility s AOT kompilací. Entity Framework, standardní ORM framework pro .NET aplikace, neposkytuje plnou podporu pro AOT kompilaci, což vedlo k nutnosti implementace vlastního řešení pro objektově-relační mapování.

Vyvinutý mapper implementuje základní CRUD operace (Create, Read, Update, Delete) a poskytuje typově bezpečné rozhraní pro práci s databázovými entitami. Architektura mapperu je založena na generických třídách, které umožňují definovat mapování mezi databázovými tabulkami a doménovými objekty. Mapper využívá Npgsql jako nízkoúrovňového poskytovatele pro komunikaci s PostgreSQL databází a implementuje vlastní mechanizmy pro správu připojení, transakce a mapování výsledků dotazů.

Klíčovou funkcionalitou mapperu je podpora pro kompilaci dotazů v době sestavení aplikace, což je v souladu s principy AOT kompilace. Tato vlastnost eliminuje režii spojenou s dynamickou kompilací dotazů za běhu a přispívá k celkovému výkonu aplikace. Mapper také implementuje optimalizace pro dávkové operace a podporuje asynchronní přístup k databázi, což je kritické pro škálovatelnost mikroslužeb.

Z hlediska bezpečnosti implementuje mapper parametrizované dotazy, které efektivně brání SQL injection útokům. Mapper také poskytuje mechanizmy pro logování a monitorování databázových operací, což usnadňuje diagnostiku a optimalizaci výkonu.

4.2.2 Autentizační middleware

V rámci mikroservisního ekosystému byl implementován specializovaný autentizační middleware, který zajišťuje ověření identity uživatelů a poskytuje autentizační kontext pro zpracování požadavků. Middleware funguje na principu interceptoru, který zachytává příchozí požadavky a provádí autentizaci před jejich předáním cílovým handlerům.

Na Auth mikroslužbě funguje middleware lokálně prostřednictvím volání vlastně vytvořeného interního rozhraní ITokenService, které poskytuje metody pro validaci a správu autentizačních tokenů. Toto řešení minimalizuje síťovou komunikaci a optimalizuje výkon autentizačního procesu.

Na ostatních mikroslužbách je autentizace realizována prostřednictvím gRPC komunikace přímo s Auth mikroslužbou. Při příchodu požadavku middleware extrahuje autentizační token, zasílá jej prostřednictvím gRPC volání do Auth mikroslužby k validaci a následně přikládá získaná uživatelská data do HttpContextu. Tento přístup zajišťuje centralizovanou správu autentizace a konzistentní aplikaci bezpečnostních politik napříč celým ekosystémem.

4.2.3 Middleware pro validaci interních požadavků

Pro zabezpečení komunikace mezi mikroslužbami byl implementován vlastní middleware, který zajišťuje autentizaci a autorizaci interních požadavků. Původní návrh počítal s využitím mTLS certifikátů poskytovaných Envoy Proxy, což by umožnilo ověření identity volající služby na základě certifikátu.

V důsledku nutnosti implementace multi-port konfigurace a podpory různých komunikačních protokolů však bylo nezbytné přepnout Envoy Proxy do TCP módu, který neposkytuje možnost předávání certifikátů v požadavcích. Z tohoto důvodu byl návrh middlewaru modifikován a implementován alternativní přístup založený na OAuth klientech.

Nové řešení spočívá ve vytvoření OAuth klienta v Auth databázi pro každou mikroslužbu. Při interní komunikaci middleware ověřuje validitu klienta a jeho tajného klíče prostřednictvím gRPC požadavku zaslaného do

Auth mikroslužby. Tento přístup zajišťuje bezpečnou komunikaci mezi mikroslužbami i v prostředí s omezeními danými TCP módem Envoy Proxy.

4.2.4 Middleware pro extrakci uživatelských dat

Pro zlepšení uživatelské zkušenosti a zvýšení bezpečnosti byl implementován middleware, který analyzuje informace o klientském zařízení a prohlížeči. Middleware extrahuje fingerprint zařízení a UserAgent řetězec z příchozích požadavků a na základě těchto informací poskytuje kontextová data do HttpContextu.

Získané informace jsou využívány pro různé účely, včetně detekce potenciálně podezřelých přihlášení, optimalizace uživatelského rozhraní pro konkrétní zařízení a sběru analytických dat o používaných zařízeních a prohlížečích.

4.2.5 Autentizační mikroservisa

Autentizační mikroservisa zajišťuje centralizovanou správu autentizace, autorizace a auditování v celém systému. Jádrem mechanismu je implementace standardu FIDO2, poskytující bezpečné ověřování uživatelů bez tradičních hesel. Pro správu přihlášení a udržování stavu autentizace jsou využívány JWT (JSON Web Tokens).

Perzistence dat je zajištěna prostřednictvím dedikované `auth_db` databáze, logicky oddělené od ostatních mikroservisních databází. Toto řešení podporuje princip oddělení zodpovědností a umožňuje nezávislou správu autentizačních dat.

Implementace Passkeys představovala významnou výzvu, vyžadující pečlivou integraci s FIDO2 standardem a optimalizaci uživatelského zážitku napříč různými zařízeními a prohlížeči. Ačkoliv OAuth funkcionalita není v současnosti plně realizována, databázové schéma již obsahuje přípravu pro budoucí integraci této technologie.

Pro detailní pohled na strukturu databáze a konkrétní implementační detaily je v kapitole 9 uvedeno kompletní databázové schéma Autentizační

mikroservisy.

4.2.6 Uživatelská mikroservisa

Uživatelská mikroservisa zajišťuje správu uživatelských profilů a vztahů mezi uživateli. Zahrnuje ukládání a aktualizaci základních informací o uživatelích, jako jsou jméno, příjmení, e-mail a datum narození. Mikroservisa implementuje sofistikovaný systém pro správu různých typů vazeb mezi uživateli, včetně přátelství, sledování nebo blokování.

Data jsou ukládána v dedikované `user_db` databázi, jejíž schéma je optimalizováno pro efektivní dotazování komplexních vztahů mezi uživateli. Klíčovou funkcionalitou je správa uživatelských kruhů (circles), umožňující organizaci kontaktů do logických skupin s různými úrovněmi oprávnění.

Systém podporuje import a správu osobních kontaktů včetně mechanismů pro párování importovaných kontaktů s existujícími uživateli platformy. Bezpečnost a soukromí jsou zajištěny granulárním systémem oprávnění, který umožňuje uživatelům přesně kontrolovat přístup k jejich osobním informacím. Doplňuje jej robustní systém pro detekci a prevenci nežádoucího chování, včetně blokování uživatelů a reportování nevhodného obsahu.

4.2.7 Obsahová mikroservisa

Obsahová mikroservisa zajišťuje komplexní správu obsahu a interakcí uživatelů na platformě. Implementuje robustní systém pro nahrávání, ukládání a distribuci multimediálního obsahu, včetně obrázků, videí a textových příspěvků. Ačkoliv funkcionalita pro streamování není v současné implementaci plně realizována, architektura je připravena na budoucí integraci této funkce.

Významným aspektem je systém kategorizace obsahu a správy uživatelských zájmů, umožňující efektivní organizaci a vyhledávání obsahu. Mikroservisa zahrnuje základní implementaci obsahových AI profilů, které upravují doporučované kategorie a zájmy, což poskytuje základ pro budoucí implementaci pokročilejších algoritmů personalizace.

Data jsou ukládána v dedikované `content_db` databázi. Systém pro správu interakcí uživatelů s obsahem zahrnuje funkce pro hodnocení, komentování a sdílení příspěvků, s důrazem na výkon při zpracování velkého množství současných interakcí.

Architektura je navržena s ohledem na budoucí rozšíření o pokročilé metody moderace obsahu využívající modely umělé inteligence pro automatickou detekci a filtraci potenciálně škodlivého obsahu. Tato plánovaná integrace významně zvýší efektivitu moderačního procesu a posílí bezpečnost uživatelského prostředí.

4.2.8 Pluginová mikroservisa

Pluginová mikroservisa zajišťuje správu pluginů, jejich životního cyklu a rozšiřitelnosti platformy. Jádrem je systém pro správu celého procesu od vytváření přes publikování až po distribuci pluginů. Architektura implementuje sofistikované verzování, které umožňuje vývojářům publikovat aktualizace při zachování kompatibility s existujícími instalacemi. Systém podporuje různé stavy publikace včetně beta verzí, přípravy, recenze a finálního publikovaného stavu.

Významnou součástí je lokalizační systém umožňující přizpůsobení pluginů pro různé jazyky a regiony. Implementace podporuje ukládání lokalizovaných textů ve formátu JSON, což poskytuje flexibilitu při definici komplexních lokalizačních struktur s validací regionálních kódů.

Data jsou ukládána v dedikované `plugin_db` s komplexní strukturou tabulek pro metadata pluginů, verzí, instalací, lokalizací, assetů a uživatelských dat. Databázové schéma využívá sofistikované indexování pro optimalizaci výkonu dotazů a zajištění referenční integrity.

Systém správy instalací sleduje pluginy na úrovni uživatelů i zařízení, což umožňuje instalaci na různá zařízení s mechanismy pro monitoring stavu, statistiky využití a správu aktualizací. Bezpečnostní aspekty zahrnují validaci a verifikaci pluginů včetně kontroly integrity a detekce potenciálně škodlivého kódu s různými úrovněmi verifikačního statusu.

Pro ukládání a distribuci binárních souborů a assetů je implementována integrace s Cloudflare R2 Object Storage, zajišťující efektivní správu WebAssembly modulů, ikon a dalších souvisejících souborů. Součástí je také systém hodnocení a recenzí, který umožňuje uživatelům poskytovat zpětnou vazbu vývojářům s mechanismy pro prevenci zneužití.

4.3 Mobilní aplikace

Mobilní aplikace sociální platformy Socigy je implementována s využitím React Native Expo, což umožňuje vývoj multiplatformní aplikace s nativním výkonem. Aplikace zahrnuje klíčové funkce sociální sítě, včetně zobrazení příspěvků, správy profilů, řízení uživatelských kruhů, správy vztahů a obchodu s pluginy.

Pro optimalizaci výkonu a uživatelského zážitku byly implementovány pokročilé techniky. Jednou z nich je využití komponenty FlashList, která představuje optimalizovanou verzi standardní FlatList komponenty. FlashList nabízí vyšší výkon a nižší paměťovou náročnost při renderování dlouhých seznamů, což je klíčové pro plynulé procházení příspěvků a dalších datových sad. Ačkoliv FlashList není využit v celé aplikaci, jeho nasazení v kritických částech významně přispívá k celkovému výkonu.

Další významnou optimalizací je implementace cachování obrázků pomocí ExpoImage. Tato komponenta využívá výkonné nativní knihovny SDWebImage pro iOS a Glide pro Android, které zajišťují efektivní načítání a ukládání obrázků do mezipaměti. Díky tomu se výrazně snižuje množství síťových požadavků a zrychluje se zobrazení již dříve načtených obrázků.

Pro správu stavu aplikace byl zvolen přístup založený na React Contexts. Toto řešení umožňuje efektivní sdílení dat mezi komponentami bez nutnosti explicitního předávání props skrze celou komponentovou hierarchii. Implementace zahrnuje několik klíčových kontextů, jako jsou AuthContext pro správu autentizace, ThemeContext pro řízení vzhledu aplikace, a další specializované kontexty pro správu uživatelských dat a nastavení.

Proces instalace pluginů v mobilní aplikaci je navržen s ohledem na jednoduchost a bezpečnost. Uživatel může procházet dostupné pluginy v integrovaném obchodě, vybrat požadovanou verzi a iniciovat instalaci. Samotná "instalace"spočívá v registraci uživatele a zařízení na serveru jako uživatele daného pluginu, bez nutnosti stahování a spouštění kódu přímo v aplikaci. Toto řešení umožňuje centrální správu pluginů a jejich oprávnění.

4.4 Webová aplikace

Webová verze aplikace Socigy je vyvinuta s využitím frameworku Next.js, který poskytuje výkonné nástroje pro server-side rendering a optimalizaci výkonu. Funkcionalita webové aplikace zahrnuje stejné klíčové prvky jako mobilní verze, včetně zobrazení příspěvků, správy profilů, řízení uživatelských kruhů a vztahů, a obchodu s pluginy.

Pro správu stavu v webové aplikaci byl zvolen stavový manažer Zustand. Na rozdíl od React Contexts použitých v mobilní aplikaci, Zustand nabízí jednodušší API a lepší výkonnost při práci s komplexními stavy. Zustand umožňuje vytváření izolovaných úložišť stavu, což usnadňuje testování a údržbu kódu. Tato volba reflektuje specifické potřeby webové aplikace a odlišný přístup k architektuře front-endu ve srovnání s mobilní verzí.

Webová aplikace také zahrnuje specializovanou sekci pro vývojáře pluginů. Tato část poskytuje nástroje pro správu pluginů, včetně editace metadat, správy verzí, sledování logů, správy lokalizací a assets. Vývojáři mají také přístup k analytickým datům o svých pluginech, včetně hodnocení a recenzí od uživatelů. Součástí je i rozhraní pro správu databáze pluginu s přehledem o využití limitů.

Pro zajištění responzivního designu a konzistentního vzhledu napříč různými zařízeními je využit framework TailwindCSS. Tento utility-first CSS framework umožňuje rychlý vývoj responzivních rozhraní a snadnou customizaci designu.

Ačkoliv současná implementace pluginů nezahrnuje jejich přímou inte-

graci do hlavní aplikace, je to plánováno pro budoucí iterace. Toto rozhodnutí bylo motivováno potřebou důkladně otestovat koncept a optimalizovat architekturu před plnou integrací. Starší verze implementace, která zahrnovala přímou integraci pluginů, je k dispozici v přílohách práce pro srovnání a analýzu. (9, `/client/native/app-old-iteration`)

Obě verze aplikace, mobilní i webová, sdílejí společnou logiku pro klíčové funkce jako je zobrazení příspěvků, správa profilů a uživatelských vztahů. Tento přístup zajišťuje konzistentní uživatelský zážitek napříč platformami a zjednodušuje údržbu a vývoj nových funkcí.

Kapitola 5

Ekosystém uživatelských pluginů

Tato kapitola popisuje návrh a implementaci ekosystému uživatelských pluginů, který představuje jeden z klíčových inovativních prvků navržené sociální platformy. Systém umožňuje rozšiřitelnost platformy prostřednictvím modulů třetích stran, přičemž zajišťuje bezpečnost, stabilitu a konzistentní uživatelskou zkušenost.

5.1 Aplikační vrstva

Aplikační vrstva pluginového systému poskytuje rozhraní mezi jádrem platformy a uživatelskými pluginy. Tato vrstva implementuje mechanismy pro správu životního cyklu pluginů, komunikaci mezi pluginy a platformou, a zajišťuje bezpečnou exekuci kódu třetích stran.

5.1.1 Verzování API

Systém verzování API představuje klíčový aspekt pluginového ekosystému, který zajišťuje dlouhodobou kompatibilitu a udržitelnost. Implementovaný přístup umožňuje současný běh pluginů kompatibilních s různými verzemi

systémového API, což eliminuje nutnost pravidelných aktualizací pluginů při změnách v základní platformě.

Verzování je implementováno prostřednictvím sémantického verzování, kde pluginy specifikují požadovanou verzi API ve svém developerském rozhraní v aplikaci

Systém automaticky rozpoznává kompatibilní verze API a poskytuje pluginu odpovídající rozhraní. Tento mechanismus je implementován v třídě PluginCacheManager, která analyzuje požadovanou verzi API a vybírá nejvhodnější implementaci:

```
1  private fun resolveBestVersion(requestedVersion: String,
   ↪  availableVersions: List<String>): String? {
2      val baseVersion = if (requestedVersion.startsWith("^")) {
3          requestedVersion.substring(1)
4      } else {
5          requestedVersion
6      }
7
8      val requestedSemver = Semver(baseVersion,
       ↪  Semver.SemverType.NPM)
9      val rangeStart = requestedSemver
10     val rangeEnd = Semver("${requestedSemver.major + 1}.0.0",
       ↪  Semver.SemverType.NPM)
11
12     return availableVersions
13         .map { Semver(it, Semver.SemverType.NPM) }
14         .filter { it.compareTo(rangeStart) >= 0 &&
           ↪  it.compareTo(rangeEnd) < 0 }
15         .maxByOrNull { it }
16         ?.toString()
17 }
```

Tento přístup zajišťuje zpětnou kompatibilitu a umožňuje postupnou evoluci API bez narušení funkčnosti existujících pluginů.

5.1.2 Registrace a pouštění eventů

Systém událostí (events) umožňuje pluginům reagovat na akce uživatelů a změny v aplikaci. Implementace zahrnuje mechanismy pro registraci posluchačů událostí, distribuci událostí a jejich zpracování v rámci pluginů.

Pro usnadnění práce s událostmi se specializuje část vyvinutého Rust frameworku, který poskytuje vysokoúrovňové abstrakce pro registraci a zpracování událostí:

```
1  thread_local! {
2      pub(crate) static REGISTERED_EVENTS: Rc<RefCell<HashMap<String,
       ↪  Box<dyn FnMut(&UIEvent)>>>> =
       ↪  Rc::new(RefCell::new(HashMap::new()));
3  }
4
5  #[wasm_bindgen]
6  pub fn invoke_ui_event(id: String, event: String) {
7      let json_res = match
       ↪  serde_json::from_str::<UIEvent>(event.as_str()) {
8          Ok(res) => res,
9          Err(e) => {
10             logging::adv_error(
11                 format!("Failed to deserialize UIEvent -> {}",
                   ↪  e).as_str(),
12                 None,
13                 false,
14             );
15             return;
16         }
17     };
18
19     REGISTERED_EVENTS.with(move |events| {
20         if let Some(listener) = events.borrow_mut().get_mut(&id) {
21             listener(&json_res);
```

```
        }
    });
}
```

Tento kód umožňuje registraci posluchačů událostí a jejich vyvolání při interakci uživatele s komponentami pluginu. Události jsou serializovány do formátu JSON a předávány mezi nativním kódem a WebAssembly modulem.

Na straně klienta je implementován systém pro distribuci událostí mezi pluginy a jádrem aplikace:

```
    const eventNamesArray: string[] = [
      "onPointerEnter",
      "onPointerEnterCapture",
     // Další typy událostí
  ];
  // ... renderVDOM()
  const { type, events, children, key } = element;
  if (events) {
    Object.keys(events).forEach((event) => {
      if (eventNamesArray.includes(event)) {
        const eventIds = events[event];

        // Internal event callback
        const callback = (e: any) => {
          const eventData = JSON.stringify({
            type: event,
            ...e.nativeEvent,
          });
          eventIds.forEach((eventId) => {
            uiRegistry.invokeUiEvent(pluginId, eventId, eventData);
          });
        };

```

```
24          // Props passed to the rendered component
25          if (props) {
26            props[event] = callback;
27          } else {
28            props = { [event]: callback };
29          }
30        }
31      });
32    }
```

Tento přístup umožňuje pluginům reagovat na události uživatelského rozhraní, jako jsou kliknutí, změny rozměrů nebo jiné interakce, což poskytuje bohaté možnosti pro implementaci interaktivních komponent.

5.1.3 Dynamické uživatelské rozhraní

Dynamické uživatelské rozhraní představuje klíčovou funkcionalitu pluginového systému, která umožňuje pluginům definovat a vykreslovat vlastní komponenty v rámci aplikace. Implementace je založena na konceptu virtuálního DOM (VDOM), který umožňuje efektivní aktualizace uživatelského rozhraní.

Vyvinutý Rust framework poskytuje JSX-podobnou syntaxi pro definici uživatelského rozhraní:

```
1  #[ui_component]
2  struct Page {
3      render_string: bool,
4      content: Option<PropStr>,
5      image_url: Option<PropStr>,
6  }
7
8  impl UIComponent for Page {
9      fn render(&mut self) -> Option<UIElement> {
10         ui! {
11             <View class_name="flex-1 flex b" on_layout={|e| {
```

```
12                  info!("OnLayout event was fired on Page. Event:
                    ↪ {:?}", e);
13              }}>
14                  <Text class_name="text-2xl font-inter-bold
                    ↪ text-foreground">Your watchlist</Text>
15                  <Counter render_string={self.render_string}
                    ↪ image_url={self.image_url.clone().unwrap()}
                    ↪ content={self.content.clone().unwrap()} />
16              </View>
17          }
18      }
19  }
```

Tento kód definuje komponentu s vlastnostmi a metodou pro vykreslení, která využívá vlastně vyvinuté makro `ui!` pro definici struktury uživatelského rozhraní. Framework automaticky převádí tuto definici na virtuální DOM reprezentaci, která je následně serializována a předána do nativní aplikace.

Na straně klienta je implementován systém pro vykreslování komponent definovaných pluginy:

```
1  export default function Dynamic({
2    id,
3    defaultElement,
4    props,
5    uiRegistry,
6  }: DynamicProps) {
7    const [vdom, dispatchChange] = useReducer(dynamicVdomReducer,
     ↪ undefined);
8
9    useEffect(() => {
10     const registeredPlugin = uiRegistry.getComponentPlugin(id);
11     if (!registeredPlugin) {
12       return;
13     }
```

```
14
15      const changeSubscription = SocigyWasm.addListener(
16        "onComponentChange",
17        (data) => {
18          if (!data.changes) return;
19          const changes: VDOMChange[] = JSON.parse(data.changes);
20          dispatchChange(changes);
21        }
22      );
23
24      if (
25        !SocigyWasm.renderComponent(registeredPlugin, id,
          ↪  JSON.stringify(props))
26      ) {
27        console.error(`Failed to render dynamic component ${id}`);
28      }
29      return () => {
30        changeSubscription.remove();
31      };
32    }, [id]);
33
34    if (!vdom) {
35      return defaultElement;
36    }
37    return renderVDOM(vdom);
38  }
```

Tento kód zajišťuje vykreslení komponenty definované pluginem a její aktualizaci při změnách. Systém využívá virtuální DOM pro optimalizaci výkonu a minimalizaci počtu aktualizací reálného DOM.

5.2 Pluginová vrstva

Pluginová vrstva zajišťuje bezpečné spouštění kódu třetích stran a poskytuje rozhraní pro interakci s jádrem aplikace. Tato vrstva je implementována s využitím WebAssembly, což umožňuje efektivní a bezpečné provádění kódu v sandboxovaném prostředí.

5.2.1 Sandoxing u nativních zařízení

Implementace sandboxingu na nativních zařízeních představuje významnou technickou výzvu, zejména na mobilních platformách. Pro řešení tohoto problému byl vyvinut specializovaný nativní modul pro Android, který umožňuje bezpečné spouštění WebAssembly kódu v izolovaném prostředí.

Modul SocigyWasm implementuje rozhraní mezi nativním kódem a WebAssembly moduly:

```
1  class SocigyWasmModule : Module() {
2    private final var PluginCacheManager: PluginCacheManager? = null;
3
4    override fun definition() = ModuleDefinition {
5      Name("SocigyWasm")
6
7      OnCreate() {
8        SocigyWasmExceptions.setSendEvent(::sendEventWrapper);
9        try {
10         PluginCacheManager = PluginCacheManager(getContext(),
           ↪  ::sendEventWrapper);
11       } catch (e: Exception) {
12         sendEvent("onFatal", bundleOf(
13           "message" to "FATAL_ERR - " + e.toString(),
14           "uiDelay" to 0
15         ));
16       }
17     }
```

```
18
19      // Definice API pro komunikaci s React Native
20      // ...
21    }
22  }
```

Tento modul poskytuje rozhraní pro načítání, inicializaci a spouštění WebAssembly modulů v rámci React Native aplikace. Implementace využívá WebView jako runtime pro WebAssembly, což zajišťuje kompatibilitu s většinou Android zařízení.

Klíčovým aspektem implementace je izolace kódu pluginů, která je zajištěna prostřednictvím WebAssembly sandboxu. WebAssembly poskytuje bezpečnostní model založený na lineární paměti a omezeném přístupu k hostitelskému prostředí, což efektivně brání neoprávněnému přístupu k systémovým zdrojům.

Přístup k funkcionalitám platformy je řízen prostřednictvím explicitně exportovaných funkcí, které jsou dostupné pluginům:

```
1  WebView!!.addJavascriptInterface(ISocigyLogging(sendEvent),
   ↪  "SocigyLogging");
2  WebView!!.addJavascriptInterface(ISocigyInternal(sendEvent),
   ↪  "SocigyInternal");
3  WebView!!.addJavascriptInterface(ISocigyUtils(sendEvent),
   ↪  "SocigyUtils");
4  WebView!!.addJavascriptInterface(ISocigyPermissions(sendEvent),
   ↪  "SocigyPermissions");
5  WebView!!.addJavascriptInterface(ISocigyUI(sendEvent), "SocigyUI");
```

Tento přístup umožňuje přesnou kontrolu nad tím, k jakým funkcionalitám mají pluginy přístup, a implementaci systému oprávnění, který umožňuje uživatelům kontrolovat, jaké akce mohou pluginy provádět.

5.3 UI

Uživatelské rozhraní pluginového systému poskytuje mechanismy pro definici, registraci a vykreslování komponent definovaných pluginy. Tato část systému zajišťuje integraci pluginů do uživatelského rozhraní aplikace a poskytuje konzistentní uživatelskou zkušenost.

5.3.1 Definice komponentů a jejich registrace

Systém pro definici a registraci komponent umožňuje pluginům vytvářet vlastní uživatelské rozhraní, které je integrováno do aplikace. Implementace zahrnuje mechanismy pro definici komponent, jejich registraci v rámci aplikace a správu jejich životního cyklu.

Definice komponent v rámci pluginů je realizována prostřednictvím vyvinutého Rust frameworku, který poskytuje vysokoúrovňové abstrakce pro tvorbu uživatelského rozhraní:

```
#[ui_component]
struct Counter {
    render_string: bool,
    content: PropStr,
    image_url: PropStr,
}

impl UIComponent for Counter {
    fn render(&mut self) -> Option<UIElement> {
        ui! {
            <View class_name="flex-1 flex-row items-center
            ↪ justify-center">
                <Text>{self.content.to_string()}</Text>
                <Image source={{ uri: self.image_url.to_string() }}
                ↪ style={{ width: 100, height: 100 }} />
            </View>
        }
```

```
    }
}
```

Tento kód definuje komponentu s vlastnostmi a metodou pro vykreslení, která využívá makro ui! pro definici struktury uživatelského rozhraní. Komponenta je následně registrována v rámci aplikace:

```
let component_id = "2368bb7a-1021-49d1-85f3-7049fb15abed";
register_component::<MyComponent>(&component_id);
```

Na straně klienta je implementován systém pro registraci a správu komponent:

```
private internal_registerComponent(data: ComponentBasicEventData) {
  this.components.set(data.componentId, data.pluginId);
  let registered = this.plugins.get(data.pluginId);
  if (!registered) this.plugins.set(data.pluginId,
  ↪  [data.componentId]);
  else {
    registered.push(data.componentId);
    this.plugins.set(data.pluginId, registered);
  }
}
```

Tento kód zajišťuje registraci komponenty v rámci aplikace a její asociaci s příslušným pluginem. Registrované komponenty jsou následně dostupné pro vykreslení v rámci aplikace.

5.3.2 Renderování komponentů

Systém pro vykreslování komponent zajišťuje efektivní vykreslení uživatelského rozhraní definovaného pluginy. Implementace je založena na konceptu virtuálního DOM, který umožňuje efektivní aktualizace uživatelského rozhraní.

Vykreslování komponent je realizováno prostřednictvím specializovaného rendereru, který převádí virtuální DOM reprezentaci na nativní komponenty:

```tsx
function renderVDOM(
  element: UIElement | undefined,
  uiRegistry: UIRegistry,
  pluginId: string
): React.JSX.Element | undefined {
  if (!element) return undefined;
  else if (typeof element === "string") return
  ↪  <Text>{element}</Text>;

  const { type, events, children, key } = element;
  let props = { ...element.props };

  const Component = getElementByType(type);
  const renderedChildren = children?.map((x) =>
    renderVDOM(x, uiRegistry, pluginId)
  );

  if (type == "Fragment") {
    return <Component children={renderedChildren} />;
  }

  if (events) {
    // Events...
  }

  if (type == "External") {
    return (
      <Component
        key={element.key}
        {...props}
        children={renderedChildren}
        uiRegistry={uiRegistry}
      />
```

```
    );
  }

  return (
    <Component key={key} {...props} children={renderedChildren} />
  );
}
```

Tento kód převádí virtuální DOM reprezentaci na nativní React komponenty, které jsou následně vykresleny v rámci aplikace. Systém podporuje hierarchické komponenty, vlastnosti a události, což umožňuje vytváření komplexního uživatelského rozhraní.

Optimalizace

Pro zajištění optimálního výkonu při vykreslování komponent definovaných pluginy byly implementovány různé optimalizační techniky. Klíčovou optimalizací je využití virtuálního DOM, který umožňuje minimalizovat počet aktualizací reálného DOM.

```
function dynamicVdomReducer(
  state: UIElement | undefined,
  actions: VDOMChange[]
): UIElement | undefined {
  return produce(state, (draft) => {
    if (typeof draft === "string") return;
    else if (!draft) return actions[0].element;

    actions.forEach((action) => {
      switch (action.type) {
        case "addElement": {
          // ...
        }
        case "replaceElement": {
          // ...
```

```
16          }
17          // Další typy změn
18        }
19      });
20
21      return draft;
22    });
23  }
24
```

Tento kód implementuje reducer pro aplikaci změn na virtuální DOM. Systém podporuje různé typy změn, jako je přidání, odstranění nebo aktualizace elementů, což umožňuje efektivní aktualizace uživatelského rozhraní bez nutnosti překreslení celého stromu.

Další optimalizací je využití WebAssembly pro výkonné zpracování dat a logiky na straně pluginů. WebAssembly poskytuje výkon blízký nativnímu kódu, což umožňuje efektivní implementaci komplexních algoritmů a zpracování dat v rámci pluginů.

Kapitola 6

Monitoring a benchmarking

6.1 Monitorování a logování

Efektivní monitorování a logování představuje klíčový aspekt správy komplexní mikroservisní architektury sociální platformy Socigy. Pro zajištění robustního dohledu nad systémem byla implementována specializovaná monitorovací infrastruktura založená na moderních open-source nástrojích.

Jádrem monitorovacího systému je stack složený z Grafany, Promethea a Loki, který běží v dedikovaném namespace v rámci Kubernetes clusteru, avšak mimo Consul Service Mesh. Toto architektonické rozhodnutí zajišťuje nezávislost monitorovacích komponent na monitorovaném prostředí, což eliminuje potenciální kaskádové selhání v případě problémů s Service Mesh.

Prometheus slouží jako primární nástroj pro sběr a ukládání metrických dat z různých komponent systému. V implementovaném řešení Prometheus dynamicky objevuje monitorované cíle prostřednictvím Kubernetes API, což umožňuje automatické přidávání nových instancí mikroslužeb do monitorovacího systému bez nutnosti manuální konfigurace. Pro každou mikroslužbu jsou exponovány standardizované metriky zahrnující využití CPU, paměti, latenci požadavků a počet zpracovaných transakcí.

Loki doplňuje monitorovací infrastrukturu jako škálovatelný systém pro agregaci a analýzu logů. Na rozdíl od tradičních řešení pro správu logů, Loki

neindexuje obsah logů, ale pouze metadata, což významně snižuje nároky na úložiště a výpočetní zdroje. Implementace zahrnuje standardizovaný formát logů napříč všemi mikroslužbami, což usnadňuje jejich analýzu a korelaci. Logy jsou strukturovány v JSON formátu a obsahují kontextové informace jako ID požadavku, ID uživatele a další relevantní metadata, což umožňuje efektivní trasování požadavků napříč distribuovaným systémem.

Grafana slouží jako centrální vizualizační platforma, která integruje data z Promethea a Loki do přehledných dashboardů. Tento přístup umožňuje komplexní pohled na výkon a zdraví celého mikroservisního ekosystému.

Pro monitorování Kubernetes clusteru jsou využívány specializované exportery, které poskytují detailní pohled na stav jednotlivých uzlů, podů a dalších Kubernetes objektů. Tyto metriky jsou integrovány do centrálního monitorovacího systému, což umožňuje korelaci problémů na úrovni aplikace s potenciálními problémy na úrovni infrastruktury.

Consul Service Mesh poskytuje dodatečnou vrstvu monitorovacích dat prostřednictvím integrovaných metrik o komunikaci mezi službami. Tyto metriky zahrnují latenci, propustnost a chybovost jednotlivých volání mezi mikroslužbami, což umožňuje identifikaci úzkých hrdel a problematických služeb v rámci distribuovaného systému.

6.2 Benchmarking

Benchmarking představuje kritickou součást vývojového procesu sociální platformy Socigy, zejména v kontextu pluginového systému, který musí efektivně zpracovávat dynamické uživatelské rozhraní a interakce. V rámci vývoje byly provedeny rozsáhlé výkonnostní testy zaměřené na optimalizaci klíčových komponent systému.

Významným aspektem benchmarkingu byla evaluace různých implementačních přístupů pro pluginový systém. Původní implementace založená na AssemblyScript dosahovala v zátěžových testech výkonu přibližně 16 000 snímků za sekundu (fps). Tato metrika reprezentuje rychlost, s jakou je systém schopen

generovat aktualizace virtuálního DOM a připravovat JSON reprezentace pro vykreslení uživatelského rozhraní, bez započítání času potřebného pro skutečné vykreslení na obrazovku.

Následná reimplementace pluginového systému v jazyce Rust přinesla dramatické zlepšení výkonu, dosahující až 200 000 fps ve stejných testovacích podmínkách. Toto více než dvanáctinásobné zvýšení výkonu lze přičíst několika faktorům:

- Efektivnější správa paměti v Rustu ve srovnání s AssemblyScriptem
- Optimalizovaná implementace virtuálního DOM s minimálními režijními náklady
- Efektivnější serializace a deserializace JSON struktur
- Vylepšený algoritmus pro detekci změn a minimalizaci aktualizací

Benchmarking zahrnoval také měření latence při zpracování uživatelských interakcí, jako jsou kliknutí, změny rozměrů nebo jiné události. Rust implementace dosáhla průměrné latence pod 5 ms, což je hluboko pod prahem 100 ms, který je považován za hranici pro vnímání okamžité reakce uživatelského rozhraní.

Kromě výkonnostních testů pluginového systému byly provedeny také zátěžové testy mikroservisní architektury jako celku. Tyto testy simulovaly vysoké zatížení s tisíci současných uživatelů a měřily schopnost systému škálovat a udržet konzistentní výkon. Výsledky ukázaly, že implementovaná architektura je schopna efektivně škálovat horizontálně a udržet stabilní latenci i při vysokém zatížení.

Benchmarking odhalil několik potenciálních úzkých hrdel v systému, především v oblasti databázových operací a mezislužbové komunikace. Na základě těchto zjištění byla implementována řada optimalizací. Klíčovým vylepšením bylo zavedení pokročilého connection poolingu pro databázové operace, což výrazně snížilo latenci a zvýšilo propustnost systému. Další plánovanou optimalizací je implementace distribuovaného cachování často přistupovaných dat pomocí

Redis. Toto řešení má potenciál významně redukovat zátěž na primární databázi a zrychlit odezvu systému. Ačkoliv z časových důvodů nebylo Redis cachování dosud plně implementováno, předběžné analýzy naznačují, že by mohlo přinést až 30% zlepšení v rychlosti odezvy pro nejčastěji požadovaná data.

Výsledky benchmarkingu potvrzují, že zvolený technologický stack a implementační přístupy poskytují solidní základ pro výkonnou a škálovatelnou sociální platformu. Zejména přechod na Rust pro implementaci kritických komponent pluginového systému se ukázal jako klíčové rozhodnutí pro dosažení vynikajícího výkonu při zpracování dynamického uživatelského rozhraní.

Kapitola 7

Diskuze a vyhodnocení výsledků

V této kapitole se zaměřím na zhodnocení dosažených výsledků, porovnání implementovaného řešení s existujícími alternativami a identifikaci omezení současné implementace spolu s návrhy na budoucí rozvoj.

7.1 Hodnocení dosažených výsledků

Implementace sociální platformy Socigy představuje komplexní technologické řešení, které úspěšně adresuje řadu identifikovaných nedostatků existujících sociálních sítí. Při hodnocení dosažených výsledků je třeba zohlednit několik aspektů.

Z hlediska frontend implementace bylo dosaženo funkčního uživatelského rozhraní, které demonstruje základní koncept platformy. Využití React Native Expo pro mobilní aplikaci a Next.js pro webovou aplikaci umožnilo vytvořit multiplatformní řešení s konzistentní uživatelskou zkušeností. Ačkoliv současná implementace UI/UX nedosahuje všech původně zamýšlených cílů, jako je například podpora dokovacích tabů a plnohodnotný multitasking na webové platformě, poskytuje solidní základ pro další rozvoj.

Mikroservisní architektura implementovaná na platformě Kubernetes prokázala

svou efektivitu při řešení problémů škálovatelnosti a modularity. Implementace zahrnuje čtyři klíčové mikroslužby (Auth, User, Content, Plugin), které společně poskytují robustní základ pro funkcionalitu sociální platformy. Současná implementace je omezena na jeden Kubernetes cluster, což limituje možnosti geografické distribuce, nicméně architektura je navržena s ohledem na budoucí rozšíření do multi-cluster prostředí.

Významným úspěchem je implementace pluginového systému založeného na WebAssembly, který umožňuje bezpečné spouštění kódu třetích stran v sandboxovaném prostředí. Ačkoliv integrace pluginů do hlavní aplikace nebyla z časových důvodů dokončena, byla vytvořena kompletní infrastruktura pro vývoj, distribuci a správu pluginů, včetně vlastního Rust frameworku pro vývoj pluginů s podporou JSX-podobné syntaxe.

Z bezpečnostního hlediska bylo dosaženo významného pokroku implementací moderních autentizačních mechanismů, včetně podpory Passkeys (FIDO2) a vícefaktorové autentizace. Implementace mTLS v rámci Service Mesh zajišťuje zabezpečenou komunikaci mezi mikroslužbami. Plánovaná integrace HashiCorp Vault pro správu tajemství a šifrovacích klíčů nebyla z časových důvodů plně realizována, ačkoliv infrastruktura pro jeho nasazení byla připravena.

Monitoring a observabilita systému byly zajištěny implementací stacku Grafana, Loki a Prometheus, který poskytuje komplexní nástroje pro sledování výkonu a detekci anomálií. Tato infrastruktura je klíčová pro zajištění spolehlivého provozu a proaktivní identifikaci potenciálních problémů.

7.2 Porovnání s existujícími řešeními

Při porovnání implementovaného řešení s existujícími sociálními platformami je patrných několik významných rozdílů a inovací.

Z technologického hlediska se Socigy odlišuje od existujících platforem implementací moderních autentizačních mechanismů. Zatímco platformy jako Instagram a Facebook stále spoléhají primárně na tradiční hesla doplněná

dvoufaktorovou autentizací, Socigy implementuje podporu Passkeys založených na standardu FIDO2, což poskytuje vyšší úroveň zabezpečení při současném zjednodušení procesu přihlašování.

Významnou inovací oproti existujícím řešením je implementace zařízením orientované autentizace, která poskytuje uživatelům větší kontrolu nad přístupem k jejich účtům. Tento přístup umožňuje uživatelům spravovat svá autentizovaná zařízení a v případě potřeby okamžitě odebrat přístupová práva konkrétnímu zařízení bez nutnosti měnit přihlašovací údaje pro všechna ostatní zařízení.

Z hlediska uživatelského rozhraní současná implementace Socigy zaostává za vysoce optimalizovanými rozhraními dominantních platforem, které investovaly značné prostředky do vývoje a testování UI/UX. Nicméně, koncept podpory dockovacích tabů a multitaskingu na webové platformě představuje potenciální výhodu oproti existujícím řešením, která tyto funkce typicky nepodporují.

7.3 Omezení implementace a návrhy na budoucí rozvoj

Současná implementace sociální platformy Socigy má několik omezení, která představují příležitosti pro budoucí rozvoj.

Jedním z hlavních omezení je absence plné integrace pluginů do hlavní aplikace. Ačkoliv byla vytvořena kompletní infrastruktura pro vývoj a distribuci pluginů, jejich integrace do uživatelského rozhraní nebyla z časových důvodů dokončena. Budoucí vývoj by se měl zaměřit na dokončení této integrace, což by umožnilo uživatelům plně využívat potenciál pluginového systému.

Z bezpečnostního hlediska představuje významné omezení neúplná implementace HashiCorp Vault pro správu tajemství a šifrovacích klíčů. Ačkoliv infrastruktura pro nasazení Vaultu byla připravena, jeho plná integrace s ostatními komponentami systému nebyla dokončena. Budoucí vývoj by se

měl zaměřit na dokončení této integrace, což by poskytlo robustní řešení pro správu citlivých informací.

Pro optimalizaci výkonu by bylo vhodné implementovat distribuované cachování s využitím Redis, což by mohlo významně snížit latenci a zvýšit propustnost systému. Tato optimalizace by byla zvláště přínosná při škálování platformy na větší počet uživatelů.

Dalším směrem budoucího rozvoje by mohla být implementace pokročilých algoritmů pro personalizaci obsahu, které by poskytovaly uživatelům relevantnější obsah při zachování transparentnosti a kontroly. Toto by mohlo zahrnovat implementaci systému pro moderaci obsahu s využitím modelů umělé inteligence, což by zvýšilo bezpečnost platformy a kvalitu uživatelského prostředí.

Z hlediska rozšiřitelnosti pluginového systému by bylo vhodné implementovat pokročilejší mechanismy pro správu oprávnění a monitorování výkonu pluginů. Toto by mohlo zahrnovat implementaci systému pro dynamické přidělování zdrojů pluginům na základě jejich využití a implementaci pokročilých metrik pro sledování výkonu a stability pluginů.

Implementace těchto vylepšení by významně zvýšila konkurenceschopnost platformy Socigy a poskytla by uživatelům inovativní a bezpečné prostředí pro sociální interakce v digitálním prostoru.

Kapitola 8

Závěr

8.1 Shrnutí klíčových poznatků

Tato práce představila návrh a implementaci moderní sociální platformy Socigy, která řeší identifikované nedostatky existujících řešení prostřednictvím inovativního přístupu k architektuře, bezpečnosti a rozšiřitelnosti. Dosažené výsledky demonstrují potenciál zvolených technologických řešení pro vytvoření robustní a uživatelsky orientované sociální sítě.

Implementace mikroservisní architektury na platformě Kubernetes se ukázala jako vhodná volba pro zajištění škálovatelnosti a modularity systému. Kubernetes poskytl robustní základ pro orchestraci kontejnerizovaných aplikací, což umožnilo efektivní správu, škálování a nasazení jednotlivých mikroslužeb. Využití Service Mesh pomocí HashiCorp Consul přineslo významné výhody v oblasti zabezpečené komunikace mezi službami a centralizované správy síťových politik.

V oblasti frontendu bylo dosaženo významného pokroku implementací multiplatformního řešení s využitím React Native Expo pro mobilní aplikace a Next.js pro webovou aplikaci. Tento přístup zajistil konzistentní uživatelskou zkušenost napříč různými zařízeními a platformami, přičemž umožnil efektivní sdílení kódu mezi jednotlivými implementacemi.

Nejvýznamnějším přínosem práce je implementace pluginového systému

založeného na WebAssembly, který umožňuje bezpečné spouštění kódu třetích stran v sandboxovaném prostředí. Vyvinutý Rust framework pro tvorbu pluginů s podporou JSX-podobné syntaxe výrazně zjednodušuje vývoj rozšíření a poskytuje vývojářům intuitivní nástroje pro tvorbu uživatelského rozhraní. Benchmarking prokázal vynikající výkon Rust implementace, která dosahuje až 200 000 fps při zpracování virtuálního DOM, což představuje více než dvanáctinásobné zlepšení oproti původní AssemblyScript implementaci.

Z bezpečnostního hlediska bylo dosaženo významného pokroku implementací moderních autentizačních mechanismů, včetně podpory Passkeys (FIDO2) a vícefaktorové autentizace. Tento přístup eliminuje rizika spojená s tradičními hesly a poskytuje uživatelům vyšší úroveň zabezpečení při současném zjednodušení procesu přihlašování.

8.2 Doporučení pro budoucí výzkum a praxi

Na základě zkušeností získaných během vývoje platformy Socigy lze formulovat několik doporučení pro budoucí výzkum a praxi v oblasti moderních sociálních sítí.

Prioritou pro další vývoj by měla být plná integrace pluginového systému do hlavní aplikace, což by umožnilo uživatelům plně využívat potenciál rozšiřitelnosti platformy. Současně by bylo vhodné implementovat pokročilejší mechanismy pro správu oprávnění a monitorování výkonu pluginů, včetně systému pro dynamické přidělování zdrojů na základě jejich využití.

Významnou oblastí pro budoucí výzkum je implementace multi-cluster řešení s využitím Mesh Gateway, které by umožnilo transparentní komunikaci mezi službami nasazenými v různých clusterech a regionech. Tento přístup by významně zvýšil dostupnost a odolnost systému vůči výpadkům jednotlivých datových center.

Z hlediska uživatelského rozhraní existuje prostor pro implementaci pokročilých funkcí jako jsou dockovací taby a multitasking na webové platformě, což by poskytlo unikátní uživatelskou zkušenost odlišující se od existujících

řešení.

Pro optimalizaci výkonu by bylo vhodné implementovat distribuované cachování s využitím Redis, což by mohlo významně snížit latenci a zvýšit propustnost systému. Tato optimalizace by byla zvláště přínosná při škálování platformy na větší počet uživatelů.

Další směr výzkumu by se měl zaměřit na inovativní přístupy k sociálním interakcím, zejména na koncept uživatelských kruhů (circles), který poskytuje flexibilnější a granulární kontrolu nad sdílením obsahu a soukromím. Tento přístup by mohl být dále rozvinut implementací algoritmů pro automatické doporučování relevantních kruhů na základě vzorců interakcí a zájmů uživatelů.

V oblasti bezpečnosti by bylo vhodné dokončit integraci HashiCorp Vault pro správu tajemství a šifrovacích klíčů, což by poskytlo robustní řešení pro správu citlivých informací. Současně by měl být vyvinut pokročilejší systém pro detekci a prevenci neoprávněného přístupu, včetně implementace behaviorální analýzy pro identifikaci potenciálně podezřelých aktivit.

Implementace těchto doporučení by významně zvýšila konkurenceschopnost platformy Socigy a poskytla by uživatelům inovativní a bezpečné prostředí pro sociální interakce v digitálním prostoru.

Kapitola 9

Přílohy

Veškeré přílohy a dodatečné materiály můžete najít v následujícím GitHub repozitáři ve složce `/schemas`

Slovník

Cluster Seskupení uzlů (nodes) v Kubernetes, které společně provozují kontejnerizované aplikace a zajišťují jejich škálování, správu a dostupnost. 17

ConfigMap Objekt v Kubernetes umožňující ukládání konfiguračních dat odděleně od kontejnerových aplikací. 17

Docker Swarm Alternativní orchestrátor kontejnerů od Dockeru, nabízející jednodušší, ale méně škálovatelnou správu oproti Kubernetes. 18

Ingress Objekt v Kubernetes, který umožňuje směrování HTTP a HTTPS provozu k službám v clusteru na základě pravidel. 17

IW Informational Warfare. 11

Kubernetes Platforma pro orchestraci kontejnerizovaných aplikací, umožňující efektivní škálování, automatizovanou správu a nasazení mikroservis. 17

Mikroservisy Architektonický styl, ve kterém je aplikace rozdělena na menší, nezávislé služby komunikující mezi sebou. 18

NetworkPolicy Mechanismus v Kubernetes pro definici pravidel síťové komunikace mezi jednotlivými komponentami clusteru. 17

Orchestrace Automatizovaná správa nasazování, škálování a provozu kontejnerizovaných aplikací. 17

Passkeys Jsou jednodušší a bezpečnější alternativou k heslům. Umožňují vám přihlásit se pouhým otiskem prstu, skenem obličeje nebo zámkem obrazovky. 12, 13

Secret Bezpečný způsob správy citlivých informací, jako jsou hesla, OAuth tokeny nebo SSH klíče, v Kubernetes. 17

YouTube Sociální platforma na sdílení online video obsahu, vlastněná společností Google. 11, 72

YouTube Kids Oddělená platforma od YouTube, určená pro děti s velice základní ochranou proti nevhodnému obsahu. 12

Literatura

1. KEPIOS. *Global Social Media Statistics* [https://datareportal.com/social-media-users]. DataReportal, 2025. Datum citování: 21. Února 2025.
2. POWELL, Nicole. *Social Media Algorithms: How to Crack the Code in 2025* [https://www.halconmarketing.com/post/cracking-social-media-algorithms-in-2025]. Halcon, 2024. Datum citování: 15. Ledna 2025.
3. NGUYEN, Tien T.; HUI, Pik-Mai; HARPER, F. Maxwell; TERVEEN, Loren; KONSTAN, Joseph A. *Exploring the Filter Bubble: The Effect of Using Recommender Systems on Content Diversity* [https://archives.iw3c2.org/www2014/proceedings/proceedings/p677.pdf]. ACM Press, 2014. Datum citování: 15. Ledna 2025.
4. TADDEO, Mariarosaria. *Information Warfare: A Philosophical Perspective* [https://www.researchgate.net/publication/234627039_Information_Warfare_A_Philosophical_Perspective]. University of Oxford, 2021. Datum citování: 15. Ledna 2025.
5. MARTIN, Maddie. *How Much Money Do You Get Per View on YouTube? (2025 Stats)* [https://www.thinkific.com/blog/youtube-money-per-view]. Thinkific, 2024. Datum citování: 15. Ledna 2025.
6. ALLIANCE, FIDO. *What is FIDO2?* [https://fidoalliance.org/fido2/]. FIDO Alliance, [b.r.]. Datum citování: 15. Ledna 2025.

7. JAIN, Ayushi. *Decoding Instagram System Design & Architecture (And How Reels Recommendation Works?)* [https://www.techaheadcorp.com/blog/decoding-instagram-system-design-architecture-and-how-reels-recommendation-works/]. Tech Ahead, 2024. Datum citování: 15. Ledna 2025.

8. BRYANT, Daniel. *The Infrastructure Behind Twitter: Scaling Networking, Storage and Provisioning* [https://www.techaheadcorp.com/blog/decoding-tiktok-system-design-architecture/]. Info Q, 2017. Datum citování: 15. Ledna 2025.

9. SINHA, Deepak. *How TikTok Works: Decoding System Design & Architecture with Recommendation System* [https://www.techaheadcorp.com/blog/decoding-tiktok-system-design-architecture/]. Tech Ahead, 2024. Datum citování: 15. Ledna 2025.

10. MDN. *Web Authentication API* [https://developer.mozilla.org/en-US/docs/Web/API/Web_Authentication_API]. MDN Web Docs, 2025. Datum citování: 15. Ledna 2025.

11. ALI, Peshawa Jammal Muhammad. *Two-Factor Authentication 2FA: An Overview of HOTP and TOTP* [https://www.researchgate.net/profile/Peshawa-Muhammad-Ali/publication/375867152_Two-Factor_Authentication_2FA_An_Overview_of_HOTP_and_TOTP/links/65604e2fce88b8703107f7ac/Two-Factor-Authentication-2FA-An-Overview-of-HOTP-and-TOTP.pdf]. Koya University, 2023. Datum citování: 15. Ledna 2025.

Zadání maturitního projektu z informatických předmětů

Jméno a příjmení: *Patrik Stohanzl*

Pro školní rok: *2024/2025*

Třída: *4. A*

Obor: *Informační technologie 18-20-M/01*

Téma práce: *Návrh a realizace moderní sociální sítě*

Vedoucí práce: *RNDr. Jan Koupil, Ph.D.*

Způsob zpracování, cíle práce, pokyny k obsahu a rozsahu práce:

Cílem tohoto projektu je navrhnout a vyvinout moderní a inovativní platformu pro sociální interakci, která se bude odlišovat od stávajících řešení integrací pluginů a otevřením možností pro komunitu. Platforma bude robustní, škálovatelná a zaměřená na potřeby moderních uživatelů, poskytující jedinečný a obohacující zážitek.

Specifikace projektu:

1. **Analýza a návrh:**
    - Bude provedena analýza existujících sociálních sítí a jejich funkcí.
    - Budou definovány požadavky na systém a vytvořen návrh uživatelského rozhraní.
    - Bude navržena databázová struktura pro ukládání uživatelských dat, příspěvků a pluginů.
2. **Základní funkce:**
    - **Přihlášení/Registrace:**
        - Bude implementována bezpečná a pohodlná metoda pro registraci a přihlášení uživatelů.
        - Bude zajištěna autentizace pomocí Passkeys pro bezpečné a bezklikové přihlášení.
        - Bude umožněno přihlášení pomocí QR kódu pro snadné přihlášení na jiných zařízeních.
    - **Nahrávání obsahu:**
        - Uživatelům bude umožněno nahrávat fotografie a texty a sdílet je s ostatními uživateli.
    - **Prohlížení sdíleného obsahu:**
        - Bude zajištěna možnost zobrazovat příspěvky a obsah ostatních uživatelů.
    - **Přidávání přátel:**
        - Uživatelům bude umožněno přidávat své přátele a blízké jako kamarády.
3. **Integrace komunitních pluginů (nepovinný bod):**
    - Bude implementováno sandboxové prostředí pro bezpečnou exekuci kódu pluginů na klientské straně.

    - Budou poskytnuty nástroje pro vývoj a integraci pluginů komunitou.
4. **Výzkum a návrh technologického stacku:**
    - Bude proveden výzkum dostupných technologií a nástrojů vhodných pro vývoj tohoto typu aplikace.
    - Na základě výzkumu bude navržen technologický stack.
    - Při implementaci bude technologický stack ověřen a optimalizován.
5. **Notifikace a zasílání zpráv:**
    - Bude implementován systém notifikací pro informování uživatelů o nových příspěvcích, aktivitách a pluginových aktualizacích.
6. **Testování a dokumentace:**
    - Bude provedeno důkladné testování aplikace, včetně funkčních testů, testů uživatelského rozhraní a výkonu.
    - Bude vytvořena uživatelská a vývojářská dokumentace.
7. **Prezentace a obhajoba:**
    - Bude připravena prezentace projektu, která bude obsahovat demonstrační video ukazující hlavní funkce aplikace.
    - Bude připravena obhajoba projektu, včetně technických detailů a získaných zkušeností.

Požadované výstupy:

- Funkční webová aplikace pro komunikaci a sdílení obsahu
- Bezpečné metody přihlášení a registrace uživatelů.
- Systém pro nahrávání, správu a prohlížení obsahu.
- Možnost přidávání přátel a správy uživatelských spojení.
- (Volitelně) Sandboxové prostředí pro vývoj a integraci pluginů.
- Systém notifikací a zasílání zpráv.
- Uživatelská a vývojářská dokumentace.
- Prezentace a demonstrační video.

Hodnocení:

Projekt bude hodnocen na základě následujících kritérií:

- Kvalita a funkčnost uživatelského rozhraní.
- Schopnost aplikace plnit všechny stanovené cíle.
- Inovativnost a originalita řešení.
- Flexibilita a přizpůsobitelnost systému.
- Úroveň dokumentace a prezentace projektu.
- Kreativita a efektivnost řešení.

Stručný časový harmonogram (s daty a konkretizovanými úkoly):

- **Září:** Analýza problému, tvorba UML usecase diagramů, návrh vzhledu aplikace
- **Říjen-listopad:** Vývoj backendové části (registrace, autentizace, správa souborů, komunikační procesy a API, problémy nasazení)
- **Prosinec:** Vývoj frontendové části, příprava pro pluginy
- **Leden:** Finalizace projektu
- **Únor - březen:** Práce na dokumentaci a oprava chyb