Restrukturalizace Ekosystému MCP-Prompts: Strategický Návrh pro Přechod na Multi-Repoziřářovou Architekturu

Expertní profil

Titul: Systémový architekt & DevOps stratég **Popis:** Expert s více než 15 lety zkušeností s navrhováním a implementací škálovatelných a udržitelných softwarových architektur pro rozsáhlé podnikové systémy a open-source platformy. Specializuje se na Domain-Driven Design, mikroslužby, CI/CD automatizaci a optimalizaci Git workflow. Je známý pro svůj pragmatický přístup, který vyvažuje technickou čistotu s obchodními cíli a zajišťuje, že architektonické změny přinášejí hmatatelnou hodnotu v podobě rychlosti vývoje, spolehlivosti systému a budoucí udržitelnosti. Tento expert komunikuje komplexní technické strategie jasným, strukturovaným a akčním způsobem, často vytváří podrobné zprávy, které slouží jako plány pro inženýrské týmy.

Shrnutí

Tato zpráva předkládá komplexní strategii pro restrukturalizaci ekosystému mcp-prompts z existující monorepoziřářové struktury na federovanou multi-repoziřářovou architekturu. Cílem této transformace je podpořit strategickou vizi projektu stát se základním kamenem ekosystému Model Context Protocol (MCP) tím, že se zvýší jeho modularita, škálovatelnost a otevřenost pro komunitní příspěvky. Analýza současného stavu ukazuje, že ačkoli je projekt technicky vyspělý, jeho monolitická povaha brání nezávislému vývoji jeho klíčových komponent.
Navrhovaná architektura rozděluje projekt do sady specializovaných, nezávisle spravovatelných repozitářů, včetně repozitářů pro sdílené kontrakty, katalog promptů, jádrovou logiku, serverovou aplikaci, CLI a dedikovanou Android aplikaci. Tento model je doplněn o centrální meta-repozitář pro správu celkové dokumentace, sledování problémů a orchestraci CI/CD.
Zpráva dále poskytuje podrobný, fázovaný migrační plán, který klade důraz na zachování historie revizí pomocí nástroje git filter-repo. Tento plán je navržen tak, aby minimalizoval rizika a zajistil hladký přechod. Součástí návrhu je také federovaný CI/CD model, který využívá znovupoužitelné GitHub Actions workflows a mechanismus repository_dispatch pro orchestraci sestavení a vydávání napříč repozitáři.
Očekávanými výsledky této restrukturalizace jsou zvýšená autonomie vývojových týmů, efektivnější a rychlejší CI/CD cykly, zjednodušená správa závislostí a jasná cesta pro budoucí rozšíření ekosystému. Tato architektonická evoluce není pouhým technickým refaktoringem, ale strategickým krokem, který transformuje mcp-prompts z produktu na platformu, připravenou na budoucí růst a širší přijetí v rámci komunity.

I. Strategické zdůvodnění architektonické evoluce: Od

monolitu k federovanému ekosystému

1.1 Analýza současného stavu a strategické imperativy

Projekt mcp-prompts se nachází ve stavu značné technické zralosti. Jeho základ tvoří moderní technologický stack postavený na TypeScriptu, který zajišťuje typovou bezpečnost a lepší čitelnost kódu. Použití npm workspaces svědčí o dobře promyšlené správě interních balíčků v rámci monorepoziřáře. Projekt je podpořen komplexní sadou testů, která zahrnuje jak jednotkové testy (unit tests) pro jednotlivé komponenty, jako jsou služby a adaptéry , tak integrační testy (integration tests) ověřující spolupráci s externími systémy, jako je databáze PostgreSQL nebo HTTP server. Robustní Docker konfigurace, spravovaná pomocí docker-compose a sady skriptů, umožňuje snadné a opakovatelné nasazení v různých prostředích. Vše je zastřešeno sofistikovaným CI/CD pipeline definovaným v GitHub Actions, který automatizuje testování a sestavení.
Strategickým cílem projektu, jak je opakovaně zdůrazněno v analytických dokumentech, je stát se "jediným zdrojem pravdy" (single source of truth) pro správu promptů v rámci širšího ekosystému Model Context Protocol (MCP). Projekt není zamýšlen pouze jako nástroj, ale jako referenční implementace standardu MCP, navržená pro maximální interoperabilitu a kompozabilitu s ostatními komponentami ekosystému.
Jádrem problému, který brání plnému naplnění této vize, je však současná monorepoziřářová architektura. Ačkoli byla efektivní pro počáteční fázi vývoje, kdy umožňovala rychlé iterace a snadné sdílení kódu, nyní se stává brzdou. Pevné svázání komponent s odlišnými životními cykly a odpovědnostmi v jediném repozitáři omezuje jejich nezávislý vývoj a škálování.

1.2 Omezení monorepoziřáře v rostoucím ekosystému

Současná monolitická struktura s sebou přináší několik klíčových omezení, která se s rostoucí komplexitou projektu stávají stále citelnějšími:

- **Vázané verzování (Versioning Lockstep):** Nejzávažnějším problémem je, že změna v jedné, byť podružné, komponentě vynucuje vydání nové verze celého balíčku. Například přidání nového promptu do balíčku @spareSparrow/mcp-prompts-catalog by mělo vést pouze k vydání nové verze tohoto katalogu. V současném uspořádání však tato změna vyvolá vydání nové verze hlavního balíčku @spareSparrow/mcp-prompts , což je sémanticky nesprávné a pro uživatele matoucí.
- **Kognitivní a vlastnická zátěž:** Jediný velký repozitář ztěžuje vymezení jasných hranic vlastnictví a odpovědnosti. Tým, který se soustředí na vývoj Android aplikace , by neměl být zatěžován komplexitou databázového adaptéru pro PostgreSQL nebo detaily CI pipeline pro publikaci Docker obrazů. To zvyšuje kognitivní zátěž pro všechny vývojáře a zpomaluje proces onboardingu nových členů.
- **Neefektivita CI/CD:** Přestože je stávající CI pipeline robustní , jeho efektivita klesá s růstem projektu. Změna v izolovaném balíčku, jako je mcp-prompts-catalog, spouští testy pro celý projekt, včetně integračních testů serveru a databáze. To vede k plýtvání výpočetními zdroji a zbytečně prodlužuje dobu sestavení, což je v rozporu s principy rychlé a efektivní kontinuální integrace.

1.3 Výhody multi-repoziřářové architektury: Podpora federovaného

ekosystému

Přechod na multi-repoziřářovou architekturu nabízí řešení výše uvedených problémů a přináší strategické výhody, které jsou v souladu s vizí projektu:

- **Nezávislé životní cykly:** Rozdělení komponent do vlastních repozitářů umožňuje, aby každá z nich byla verzována, testována a vydávána nezávisle. To je v souladu s moderními postupy správy balíčků a principy mikroslužeb, kde každá komponenta má svůj vlastní, jasně definovaný životní cyklus.
- **Jasné vlastnictví a specializace:** Každý repozitář může mít svého dedikovaného vlastníka nebo tým, což podporuje hlubokou specializaci a odpovědnost. Tým zodpovědný za mcp-prompts-android může spravovat svůj kompletní vývojový cyklus bez nutnosti koordinace s týmem pracujícím na jádru serveru.
- **Usnadnění externích příspěvků:** Menší, zaměřené repozitáře s jasně definovaným účelem a rozhraním jsou pro nové a externí přispěvatele výrazně snazší na pochopení a přispívání. To je klíčové pro naplnění cíle projektu stát se centrální součástí otevřeného ekosystému MCP a přilákat komunitu vývojářů.

Tato restrukturalizace není pouze technickým cvičením, ale představuje strategický posun od vnímání mcp-prompts jako jednoho *produktu* k jeho etablování jako *platformy*. Strategické dokumenty opakovaně zdůrazňují roli projektu jako *referenční implementace* standardu MCP. Architektura již nyní obsahuje prvky navržené pro rozšiřitelnost, jako jsou zásuvné úložné adaptéry nebo rozhraní MutablePrompt. Plány na budoucí komponenty, jako je adaptér pro Elasticsearch nebo dedikovaná Android knihovna , dále potvrzují tuto ambici. Zatímco monorepoziřář je ideální pro vývoj soudržného produktu, pro podporu ekosystému, kde mohou třetí strany vyvíjet vlastní adaptéry nebo klientské knihovny, je federovaná, multi-repoziřářová struktura nezbytná. Umožňuje ostatním záviset na stabilních, verzovaných balíčcích (jako jsou contracts a core), aniž by museli klonovat a sestavovat celý monolit. Přechod na multi-repoziřářovou architekturu je tedy nezbytným technickým krokem k transformaci projektu na základní platformu, na níž může vyrůst širší ekosystém.

II. Navrhovaná multi-repoziřářová architektura

2.1 Hlavní principy: Domain-Driven Design a jasné hranice

Navrhovaná architektura je založena na principech Domain-Driven Design (DDD), kde jsou jednotlivé komponenty seskupeny podle jejich obchodní domény a životního cyklu. Každý nový repozitář představuje samostatný, ohraničený kontext (bounded context) s jasně definovaným API kontraktem. Tento přístup zjednodušuje a konsoliduje standardy napříč ekosystémem, například tím, že definuje společný jazyk prostřednictvím sdílených typů v repozitáři contracts, což je klíčové pro udržení konzistence.

2.2 Dekompozice komponent: Nová struktura repozitářů

Současný monorepoziřář bude rozdělen do následujících, logicky oddělených repozitářů:

- **mcp-prompts-contracts**: Základní stavební kámen celého ekosystému. Bude obsahovat výhradně TypeScript rozhraní (původně v src/interfaces.ts) a Zod validační schémata (původně v src/schemas.ts). Tento balíček nebude mít žádné běhové závislosti a bude sloužit jako sdílený kontrakt, na kterém budou záviset všechny ostatní repozitáře.

- **mcp-prompts-catalog**: Distribuovatelný npm balíček obsahující výchozí a ukázkové prompty. Tento repozitář převezme obsah z adresáře packages/mcp-prompts-catalog a kořenového adresáře prompts/. Jeho nezávislé verzování umožní aktualizovat katalog promptů bez nutnosti vydávat novou verzi serveru.
- **mcp-prompts-core**: Srdce obchodní logiky. Zde bude umístěna implementace služeb jako PromptService , WorkflowService , SequenceService a implementace všech úložných adaptérů (FileAdapter, PostgresAdapter atd.) z src/adapters.ts. Tento balíček bude záviset na mcp-prompts-contracts.
- **mcp-prompts-server**: Hlavní vstupní bod aplikace. Tento repozitář bude obsahovat kód pro spuštění serveru, včetně Express serveru (src/http-server.ts), integrace s MCP SDK (src/index.ts), logiky pro Server-Sent Events (SSE) (src/sse.ts) a veškeré Docker konfigurace (docker/). Bude záviset na mcp-prompts-core a bude zodpovědný za orchestraci jednotlivých služeb.
- **mcp-prompts-cli**: Dedikovaný repozitář pro nástroje příkazového řádku, jako je workflow-cli.ts. Oddělení CLI umožní jeho nezávislé verzování a distribuci, což usnadní jeho použití v různých automatizačních skriptech.
- **mcp-prompts-android**: Samostatný domov pro Android aplikaci. Bude obsahovat nativní Rust službu a kód Kotlin aplikace , což je v souladu s definovanou mobilní strategií.

2.3 Meta-repozitář: mcp-prompts-meta

Pro efektivní správu tohoto nového federovaného ekosystému bude vytvořen centrální meta-repozitář. Tento repozitář nebude obsahovat produkční kód, ale bude sloužit jako zastřešující hub pro celý projekt, v souladu s osvědčenými postupy pro řízení komplexních, distribuovaných projektů. Jeho obsah bude následující:

- **Globální dokumentace:** Architektonické diagramy na vysoké úrovni, projektová roadmapa (přesunuto z docs/08-roadmap.md) a průvodci pro přispěvatele (CONTRIBUTING.md).
- **Centrální sledování problémů (Issue Tracking):** Místo pro sledování problémů a návrhů, které se týkají více repozitářů najednou a vyžadují koordinovanou změnu.
- **Orchestrace CI/CD:** Bude obsahovat znovupoužitelné GitHub Actions workflows, které budou volány z jednotlivých repozitářů, čímž se zajistí konzistence a centralizovaná správa CI/CD logiky.
- **Koordinace vydávání (Release Coordination):** Skripty a nástroje pro orchestraci koordinovaného vydání nových verzí napříč více repozitáři.

Mapování z monorepoziřáře do multi-repoziřářové struktury

Následující tabulka poskytuje jednoznačný a akční plán pro migraci souborů a adresářů. Slouží jako klíčový kontrolní seznam pro inženýrský tým, aby se předešlo chybám a jasně definovaly hranice a odpovědnosti každého nového repozitáře.

Zdrojová cesta (v sparesparrow/mcp-prompts)	Cílový repozitář	Publikovaný balíček	Primární vlastník/tým	Klíčové odpovědnosti
src/interfaces.ts, src/schemas.ts	mcp-prompts-contracts	@sparesparrow/mcp-prompts-contracts	Jádro platformy	Definuje datové kontrakty a validační pravidla

Zdrojová cesta (v sparesparrow/mcp-prompts)	Cílový repozitář	Publikovaný balíček	Primární vlastník/tým	Klíčové odpovědnosti
				pro celý ekosystém.
packages/mcp-prompts-catalog/, prompts/	mcp-prompts-catalog	@sparesparrow/mcp-prompts-catalog	Komunita/Obsah	Spravuje kurátorovanou sbírku výchozích a ukázkových promptů.
src/prompt-service.ts, src/workflow-service.ts, src/adapters.ts, src/errors.ts	mcp-prompts-core	@sparesparrow/mcp-prompts-core	Jádro platformy	Implementuje stavově nezávislou obchodní logiku systému.
src/index.ts, src/http-server.ts, src/sse.ts, docker/	mcp-prompts-server	@sparesparrow/mcp-prompts	Jádro platformy	Poskytuje běhové prostředí serveru, API endpointy a Docker balíčky.
src/scripts/workflow-cli.ts	mcp-prompts-cli	@sparesparrow/mcp-prompts-cli	Nástroje	Vyvíjí a udržuje rozhraní příkazového řádku pro uživatele.
android_app/	mcp-prompts-android	(AAR/APK)	Mobilní tým	Vyvíjí, sestavuje a vydává Android aplikaci a nativní knihovnu.
.github/, docs/, README.md	mcp-prompts-meta	N/A	Správa projektu	Spravuje celkovou dokumentaci projektu, sledování problémů a CI/CD orchestraci.

III. Fázovaný migrační a exekuční plán

3.1 Předpoklady: Nástroje a nastavení prostředí

Před zahájením migrace je nutné zajistit, aby všichni členové týmu měli nainstalovaný nástroj git-filter-repo. Jedná se o moderní a doporučený nástroj pro přepisování historie Gitu, který je výrazně bezpečnější a výkonnější než zastaralý git filter-branch. Dále bude pro nahrání nově vytvořených repozitářů na GitHub vyžadován osobní přístupový token (Personal Access Token, PAT) s oprávněním repo.

3.2 Fáze 1: Extrakce základních balíčků (contracts a catalog)

Migrace bude zahájena extrakcí základních balíčků, které mají minimum závislostí a tvoří základ

pro ostatní komponenty. Tento postup umožní týmu osvojit si proces na jednodušších případech a rychle dosáhnout hmatatelných výsledků.

Postup pro mcp-prompts-contracts:

1. Na GitHubu bude vytvořen nový, prázdný repozitář s názvem mcp-prompts-contracts.
2. Bude proveden čerstvý klon původního monorepoziřáře sparesparrow/mcp-prompts do dočasného adresáře.
3. Pomocí příkazu git filter-repo bude izolována historie souborů src/interfaces.ts a src/schemas.ts. Tento proces zahrnuje filtrování podle cesty a potenciální přejmenování souborů do nové struktury v cílovém repozitáři.
   ```
   git filter-repo --path src/interfaces.ts --path src/schemas.ts
   ```
4. Bude vytvořen nový soubor package.json pro balíček @sparesparrow/mcp-prompts-contracts, který bude definovat jeho název, verzi a závislosti v souladu s osvědčenými postupy pro publikaci npm balíčků.
5. Nový GitHub repozitář bude přidán jako vzdálený (remote) a přepsaná historie bude nahrána pomocí git push --force.

Tento proces bude zopakován pro repozitář mcp-prompts-catalog, kde budou extrahovány adresáře packages/mcp-prompts-catalog/ a prompts/.

3.3 Fáze 2: Oddělení jádrové logiky a aplikací

Po úspěšném vytvoření a publikaci základních balíčků bude možné přistoupit k extrakci komponent, které na nich závisí.

Postup pro mcp-prompts-core:

1. Bude vytvořen nový repozitář mcp-prompts-core.
2. Opět bude naklonován původní monorepoziřář a pomocí git filter-repo budou izolovány relevantní soubory ze src/ (např. prompt-service.ts, workflow-service.ts, adapters.ts).
3. V novém package.json bude přidána závislost na balíčku @sparesparrow/mcp-prompts-contracts.
4. Přepsaná historie bude nahrána do nového vzdáleného repozitáře.

Tento postup bude analogicky aplikován pro repozitáře mcp-prompts-server, mcp-prompts-cli a mcp-prompts-android. Repozitář mcp-prompts-server bude po migraci záviset na @sparesparrow/mcp-prompts-core.

3.4 Fáze 3: Finalizace monorepoziřáře a založení meta-repozitáře

Po úspěšném přesunutí veškerého kódu do nových repozitářů bude původní monorepoziřář sparesparrow/mcp-prompts archivován. Tím se zabrání dalším změnám a bude jasně signalizováno, že vývoj pokračuje v nové, distribuované struktuře. Současně bude vytvořen repozitář mcp-prompts-meta, který se stane novým centrálním bodem pro dokumentaci a orchestraci. Soubor README.md v archivovaném monorepoziřáři bude aktualizován tak, aby odkazoval na mcp-prompts-meta jako na nový výchozí bod pro všechny přispěvatele a uživatele.

Samotný proces migrace, který se opírá o sérii příkazů git filter-repo a manipulaci se soubory, je náchylný k lidským chybám, zejména při aplikaci na více než šest repozitářů. Vzhledem k existující kultuře skriptování v projektu je strategicky výhodné tento proces automatizovat. Měl by být vytvořen hlavní migrační skript, který bude: a) definovat mapování zdrojových a cílových repozitářů, b) automatizovat klonování monorepoziřáře, c) spouštět správné příkazy git

filter-repo pro každý nový repozitář a d) kopírovat šablony konfiguračních souborů (package.json, CI workflow) do každého nového repozitáře. Tento skript by měl podporovat režim "dry-run", který umožní týmu zkontrolovat výsledné adresářové struktury před provedením jakýchkoli nevratných operací git push. Tím se vysoce rizikový manuální proces transformuje na nízkorizikový, automatizovaný a ověřitelný postup.

IV. CI/CD orchestrace pro multi-repoziřářový ekosystém

4.1 Federovaný CI/CD model

V multi-repoziřářovém prostředí hrozí fragmentace a duplikace CI/CD logiky. Aby se předešlo tomuto problému a zajistila se konzistentní a udržitelná strategie, bude implementován federovaný model CI/CD. Klíčovým prvkem tohoto modelu je využití **znovupoužitelných workflows** (reusable workflows) uložených v centrálním repozitáři mcp-prompts-meta v adresáři .github/workflows. Jednotlivé repozitáře (mcp-prompts-core, mcp-prompts-server atd.) budou obsahovat pouze velmi jednoduché soubory workflow, které budou pomocí spouštěče workflow_call volat tyto sdílené workflows. Tento přístup centralizuje logiku pro linting, testování, sestavování a publikaci, což usnadňuje její aktualizaci a propagaci napříč celým ekosystémem a zabraňuje duplikaci komplexní logiky v každém repozitáři.

4.2 Spouštění mezi repozitáři a sestavování závislostí

Pro správu závislostí mezi repozitáři bude využit mechanismus repository_dispatch. Tato událost umožňuje, aby workflow v jednom repozitáři spustilo workflow v jiném, což je klíčové pro automatizovanou validaci a integraci.
Příklad workflow pro aktualizaci contracts:

1. Pull request je sloučen do větve main v repozitáři mcp-prompts-contracts.
2. Spustí se workflow publish.yml, které provede testy, sestavení a publikaci nové verze balíčku @sparesparrow/mcp-prompts-contracts do npm registru.
3. Po úspěšné publikaci poslední úloha ve workflow použije akci peter-evans/repository-dispatch k odeslání události contracts-updated do repozitářů mcp-prompts-core a mcp-prompts-server.
4. V těchto navazujících repozitářích jsou spuštěny workflows, které naslouchají události on: repository_dispatch: types: [contracts-updated]. Tyto workflows automaticky spustí příkaz npm update @sparesparrow/mcp-prompts-contracts a provedou své testovací sady, aby ověřily kompatibilitu s novou verzí kontraktů.

4.3 Automatizovaný pipeline pro publikaci

Pro správu verzí a generování changelogů bude nasazen nástroj jako **Changesets**. Vývojáři budou s každým pull requestem vytvářet soubory changeset, kde popíší provedené změny.
Proces vydání (Release Workflow):

1. Manuální spuštění (workflow_dispatch) v meta-repozitáři zahájí koordinovaný proces vydání.
2. Workflow agreguje všechny čekající changesety.

3. Automaticky navýší verze v souborech package.json dotčených balíčků.
4. Vytvoří pull request zpět do větve main každého dotčeného repozitáře s navýšenými verzemi a aktualizovanými soubory CHANGELOG.md.
5. Po schválení a sloučení tohoto "Version PR" je vytvořen tag, který spustí finální workflows pro publikaci do příslušných registrů.

Matice workflow napříč repozitáři

Následující matice vizualizuje komplexní síť spouštěčů a akcí v novém ekosystému. Poskytuje jasný přehled o kauzálních řetězcích událostí, což je klíčové pro ladění a údržbu systému.

Spouštěcí událost	Zdrojový repozitář	Akce	Cílový repozitář	Spuštěné workflow
PR do main	mcp-prompts-contracts	Spustit linting & testy	mcp-prompts-contracts	ci.yml (volání meta-repozitáře)
Push tagu v*	mcp-prompts-contracts	Publikovat do npm	N/A	publish.yml
Úspěšná publikace	mcp-prompts-contracts	repository_dispatch	mcp-prompts-core	validate-dependency-update.yml
Úspěšná publikace	mcp-prompts-contracts	repository_dispatch	mcp-prompts-server	validate-dependency-update.yml
Push tagu v*	mcp-prompts-server	Sestavit & nahrát Docker image	Docker Hub	publish-docker.yml
Push tagu v*	mcp-prompts-android	Sestavit AAR & publikovat	GitHub Packages	publish-android.yml

V. Škálovatelný rámec pro budoucí rozšíření

5.1 Integrace strategie pro Android

Repozitář mcp-prompts-android bude plnohodnotnou součástí nové architektury, jak je definováno ve strategické analýze. Jeho konfigurační soubor build.gradle.kts bude konzumovat balíček @sparesparrow/mcp-prompts-contracts, čímž bude zajištěna konzistence typů a rozhraní. CI/CD pipeline pro tento repozitář bude mít v meta-repozitáři dedikované, znovupoužitelné workflow pro sestavení Android App Bundle (AAB) nebo Android Archive (AAR), spouštění nativních Rust testů a publikaci artefaktů do vhodného registru, jako je GitHub Packages.

5.2 Přidávání nových služeb: Scénář (Playbook)

Multi-repoziřářová architektura poskytuje jasný a opakovatelný scénář pro přidávání nových služeb, jako je například plánovaný adaptér pro Elasticsearch.

Postup:

1. Vytvořit nový repozitář, např. mcp-server-elasticsearch.
2. Inicializovat jej se standardními soubory package.json a tsconfig.json.
3. Přidat závislost na balíčku @sparesparrow/mcp-prompts-contracts.
4. Vytvořit minimální soubor .github/workflows/ci.yml, který bude volat znovupoužitelné workflow pro testování a sestavení z meta-repozitáře mcp-prompts-meta.

Tento standardizovaný proces dramaticky snižuje tření a složitost spojenou s rozšiřováním

ekosystému.

5.3 Model správy a přispívání

Repozitář mcp-prompts-meta se stává centrem správy (governance). Zde budou umístěny soubory CONTRIBUTING.md a CODE_OF_CONDUCT.md. Architektonická rozhodnutí, návrhy na nové repozitáře nebo významné změny v kontraktech budou spravovány prostřednictvím issues a pull requestů v tomto meta-repozitáři, což zajistí holistický pohled na vývoj celého systému.

VI. Doporučení a strategický výhled

6.1 Shrnutí klíčových doporučení

- **Přijmout navrhovanou strukturu** šesti repozitářů a jednoho meta-repozitáře pro maximalizaci modularity a autonomie týmů.
- **Provést migraci ve fázích**, jak je popsáno, s prioritou extrakce základních balíčků (contracts, catalog).
- **Implementovat federovaný CI/CD model** s využitím znovupoužitelných workflows v meta-repozitáři pro zajištění konzistence a snadné údržby.
- **Nasadit nástroj Changesets** pro robustní, automatizovaný proces verzování a vydávání napříč všemi balíčky.

6.2 Analýza rizik a jejich zmírnění

- **Zvýšená komplexita lokálního vývoje:** Nastavení lokálního prostředí se šesti a více repozitáři může být složitější než práce s jedním monorepoziřářem.
    - **Zmírnění:** V meta-repozitáři bude vyvinut hlavní docker-compose.yml soubor (navazující na docker-compose.integration.yml), který bude orchestrovat celý lokální vývojový stack. Bude poskytnuta jasná dokumentace a skripty (docker-manage.sh) pro snadné spuštění prostředí.
- **Peklo závislostí (Dependency Hell):** Správa verzí napříč vzájemně závislými balíčky může být náročná.
    - **Zmírnění:** Striktní používání nástroje Changesets a automatizovaných workflows pro validaci závislostí pomůže včas odhalit integrační problémy. Nástroj Dependabot bude nakonfigurován pro všechny repozitáře, aby udržoval závislosti aktuální.
- **Rizika spojená s přepisováním historie:** git filter-repo je destruktivní operace.
    - **Zmírnění:** Migrace musí být prováděna na čerstvých klonech, nikoli na primárním repozitáři. Automatizovaný migrační skript s režimem "dry-run" je nezbytný. Před zahájením musí být vytvořena kompletní záloha původního monorepoziřáře.

6.3 Strategický výhled

Tato architektonická transformace je klíčovou investicí, která přímo umožňuje naplnění strategické vize projektu. Posouvá mcp-prompts z pozice jediného nástroje na skutečnou, rozšiřitelnou platformu. Vytvořením stabilních, nezávisle verzovaných balíčků projekt snižuje

bariéru pro adopci a integraci, čímž se pozicuje jako de facto standard pro správu promptů v rámci rostoucího ekosystému MCP. Tento krok zajišťuje dlouhodobou životaschopnost, škálovatelnost a relevanci projektu.

Works cited

1. Essential Database Patterns for Microservices, https://gleecus.com/blogs/mastering-data-architecture-microservices/ 2. Database Per Service Pattern for Microservices - GeeksforGeeks, https://www.geeksforgeeks.org/system-design/database-per-service-pattern-for-microservices/ 3. Seven Principles for Creating a Great Enterprise Architecture ..., https://help.ardoq.com/en/articles/43971-seven-principles-for-creating-a-great-enterprise-architecture-metamodel 4. Monorepo vs. multi-repo: Different strategies for organizing repositories - Thoughtworks, https://www.thoughtworks.com/en-us/insights/blog/agile-engineering-practices/monorepo-vs-multirepo 5. 5 Hurdles of Multi-Repo Management (and How to Solve Them) - GitKraken, https://www.gitkraken.com/blog/multi-repo-management-hurdles-and-solutions 6. Git Filter-Repo: The Best Way to Rewrite Git History | Learn Version Control with Git, https://www.git-tower.com/learn/git/faq/git-filter-repo 7. Using the git filter-repo tool - Graphite, https://graphite.dev/guides/git-filter-repo 8. Migrating code from one Git repository to another while keeping the ..., https://brytecode.be/articles/git/migrating-code-between-git-repositories-retaining-history/ 9. Maintain History while moving Repo from polyrepo to Monorepo : r/git - Reddit, https://www.reddit.com/r/git/comments/1g1ehlq/maintain_history_while_moving_repo_from_polyrepo/ 10. How to Create an NPM Package: A Step-by-Step Guide - Reddit, https://www.reddit.com/r/npm/comments/1fb89ud/how_to_create_an_npm_package_a_stepbystep_guide/ 11. Creating and publishing scoped public packages | npm Docs, https://docs.npmjs.com/creating-and-publishing-scoped-public-packages/ 12. Sharing actions and workflows with your organization - GitHub Docs, https://docs.github.com/en/actions/sharing-automations/sharing-actions-and-workflows-with-your-organization 13. Triggering a workflow - GitHub Enterprise Cloud Docs, https://docs.github.com/enterprise-cloud@latest/actions/using-workflows/triggering-a-workflow 14. peter-evans/repository-dispatch: A GitHub action to create a ... - GitHub, https://github.com/peter-evans/repository-dispatch 15. Using GitHub Actions to trigger Actions across repos - amaysim.technology, https://www.amaysim.technology/blog/using-github-actions-to-trigger-actions-across-repos 16. How To Create An NPM Package | Total TypeScript, https://www.totaltypescript.com/how-to-create-an-npm-package 17. Publish in CI/CD - Nx, https://nx.dev/recipes/nx-release/publish-in-ci-cd 18. rust-ci-cd-template - crates.io: Rust Package Registry, https://crates.io/crates/rust-ci-cd-template 19. publish crates · Actions · GitHub Marketplace, https://github.com/marketplace/actions/publish-crates 20. Example of compiling a Rust library into an Android app - GitHub, https://github.com/gendx/android-rust-library 21. automated-build-android-app-with-github-action - GitHub Marketplace, https://github.com/marketplace/actions/automated-build-android-app-with-github-action 22. enefce/AndroidLibraryForGitHubPackagesDemo: Sample project showcasing the steps to publish and consume Android Libraries on the GitHub Packages Registry, https://github.com/enefce/AndroidLibraryForGitHubPackagesDemo