# Analýza a Strategický Plán pro MCP Prompts Server

## Část I: Strategická Analýza Serveru MCP Prompts

Tato část zprávy poskytuje hodnocení projektu na vysoké úrovni, definuje jeho účel, architektonickou robustnost a provozní zralost. Vytváří základ pro podrobný plán implementace v části II.

### 1.1 Strategický Imperativ: Řešení Fragmentace Promptů v Rámci Ekosystému MCP

**Jádro Problému** Projekt sparesparrow/mcp-prompts vznikl jako odpověď na kritický problém v moderním vývoji AI aplikací, známý jako "fragmentace promptů". Tento problém se projevuje v několika klíčových oblastech, které brzdí efektivitu a spolehlivost vývojových týmů. Bez centralizovaného systému pro správu promptů se týmy potýkají s absencí verzování, což znemožňuje sledování změn a návrat k předchozím funkčním verzím. Dále je obtížné provádět systematické A/B testování pro vyhodnocení výkonu různých variant promptů. Tento stav také přináší bezpečnostní rizika, protože chybí kontrola nad přístupem k citlivým a cenným promptům, které představují klíčové intelektuální vlastnictví. V neposlední řadě absence centrální platformy vede k neefektivní spolupráci, kde vývojáři a vedoucí týmů nemají jednotné místo pro sdílení, revize a schvalování promptů.
**Mise Projektu** Mise serveru MCP Prompts je stát se "jediným zdrojem pravdy" (single source of truth) pro správu, ukládání a poskytování promptů a jejich šablon pro interakce s velkými jazykovými modely (LLM). Cílem je centralizovat a standardizovat celý životní cyklus promptů, od jejich vytvoření a testování až po bezpečné sdílení v rámci organizace i s externími systémy.
**Kontext Protokolu MCP** Základním kamenem, na kterém je tento server postaven, je Model Context Protocol (MCP). Jedná se o otevřený standard, který byl představen společností Anthropic s cílem standardizovat způsob, jakým AI aplikace komunikují s externími nástroji a datovými zdroji. Protokol je často přirovnáván k "USB-C pro AI", což výstižně popisuje jeho ambici vytvořit univerzální rozhraní pro připojení různých komponent. Architektura MCP je založena na modelu Host-Client-Server, kde "Host" je aplikace, se kterou uživatel interaguje (např. IDE jako Cursor), "Client" je konektor v rámci hostitele a "Server" je externí služba poskytující data nebo funkce.
**Role Serveru v Ekosystému MCP** Projekt mcp-prompts plní v této architektuře roli "Serveru", který prostřednictvím standardizovaného API zpřístupňuje dvě klíčové schopnosti: "Prompts" (předdefinované šablony) a "Tools" (nástroje pro manipulaci s těmito šablonami). Nedávný refaktoring projektu s cílem využít nové registrační metody z MCP SDK verze 1.6.1 je jasným důkazem odhodlání držet krok s vyvíjejícím se standardem a zajistit maximální kompatibilitu. Hodnota tohoto projektu je tak umocněna jeho přísným dodržováním vznikajícího průmyslového standardu. Zatímco mnoho vývojových týmů řeší správu promptů pomocí interních, proprietárních nástrojů, které vedou k izolaci a nekompatibilitě , MCP nabízí standardizovanou cestu k interoperabilitě mezi různými AI aplikacemi a nástroji. Tím, že sparesparrow/mcp-prompts implementuje tento standard, překračuje rámec běžného "správce

promptů". Stává se z něj zaměnitelná, standardizovaná komponenta v širším a rostoucím ekosystému. Potenciál jeho přijetí není omezen pouze na jeho samostatné funkce, ale je přímo spojen s růstem a přijetím samotného protokolu MCP. Tým, který se rozhodne pro tento server, si nevybírá jen nástroj; investuje do filozofie interoperabilního a modulárního vývoje AI.

## 1.2 Architektonický Návrh: Flexibilní a Interoperabilní Jádro

**Technologický Základ** Projekt je koncipován jako moderní monorepo, což usnadňuje údržbu a škálování jednotlivých komponent. Jádro serveru je postaveno na osvědčeném technologickém stacku, který zahrnuje Node.js jako běhové prostředí, Express.js pro tvorbu API a TypeScript pro zajištění typové bezpečnosti a lepší čitelnosti kódu.
**Základní Funkcionalita** Server nabízí sadu základních, ale robustních funkcí, které tvoří jeho jádro:

- **Nástroje pro správu promptů:** Implementuje kompletní sadu nástrojů pro CRUD operace (Create, Read, Update, Delete) – add_prompt, get_prompt, update_prompt, delete_prompt, list_prompts – a také klíčový nástroj apply_template pro dynamické dosazování proměnných do šablon.
- **Šablonování a filtrování:** Poskytuje základní podporu pro šablony s proměnnými a umožňuje filtrování promptů podle tagů, což usnadňuje jejich organizaci a vyhledávání.
- **Stavové a monitorovací endpointy:** Zahrnuje /health endpoint, který poskytuje informace o stavu serveru, verzi a použitém úložišti, což je klíčové pro provozní monitoring.

**Rozhraní MutablePrompt: Motor Interoperability** Klíčovým a strategicky významným architektonickým prvkem je rozhraní MutablePrompt, které je v dokumentaci popsáno jako "výkonné". Jeho hlavním účelem je umožnit dynamickou konverzi promptů mezi různými formáty, což serveru dodává mimořádnou flexibilitu a odlišuje ho od jednodušších řešení. Podporované formáty a jejich strategický význam jsou:

- **JSON Formát:** Standardní interní reprezentace dat, kterou server používá pro svou vnitřní logiku.
- **MDC Formát:** Formát založený na Markdownu (Cursor Rules), který zajišťuje přímou integraci s moderními AI-first IDE, jako je Cursor. To umožňuje vývojářům spravovat prompty ve formátu, který je pro jejich nástroje nativní.
- **PGAI Formát:** Specializovaný formát navržený pro PostgreSQL, který explicitně zahrnuje podporu pro ukládání embeddings (vektorových reprezentací). Tento formát je klíčový pro budoucí implementaci vektorového vyhledávání.
- **Template Formát:** Dynamický formát s proměnnými, který slouží jako základ pro runtime aplikaci šablon.

**Abstrakční Vrstva Úložiště** Architektura zahrnuje flexibilní vrstvu pro ukládání dat s podporou tří různých adaptérů: souborový systém, databáze PostgreSQL a formát MDC. Tato modularita umožňuje nasazení serveru v různých prostředích, od jednoduchého lokálního vývoje (souborový systém) až po škálovatelné produkční nasazení vyžadující pokročilé funkce (PostgreSQL).
Tato architektura je proaktivně navržena pro sémantické schopnosti, nejen pro pouhé ukládání dat. Zatímco základní server pro správu promptů by mohl fungovat jako jednoduché key-value úložiště, tento projekt jde mnohem dál. Zahrnutí specifického formátu PGAI pro PostgreSQL, který je explicitně spojen s "podporou embeddings" a "vektorovým vyhledáváním" , odhaluje hlubší architektonickou vizi. Cílem není pouze *ukládat* prompty, ale také jim *sémanticky rozumět*. To řeší pokročilejší verzi problému "fragmentace promptů": otázka již nezní "Kde je

prompt review-code?", ale "Ukaž mi všechny prompty, které jsou sémanticky podobné tomuto vysoce výkonnému promptu." To projekt pozicuje jako inteligentní nástroj pro objevování a optimalizaci promptů, což je podstatně hodnotnější než prostá databáze. Rozhraní MutablePrompt tuto vizi dále posiluje tím, že umožňuje, aby prompty z jakéhokoli backendu byly potenciálně převedeny do vektorizovatelné podoby a prohledávány.

**Tabulka 1: Matice Schopností Úložných Adaptérů**

| Adaptér | Primární Případ Použití | Základní CRUD Podpora | Pokročilé Šablonování | Verzování | Sémantické/Vektorové Vyhledávání |
|---|---|---|---|---|---|
| **Souborový Systém** | Jednoduchý, lokální vývoj, rychlé prototypování | Implementováno | Implementováno | Plánováno | Není podporováno |
| **PostgreSQL** | Škálovatelná produkce, transakční integrita | Implementováno | Implementováno | Plánováno | Plánováno (přes PGAI formát) |
| **MDC Adaptér** | Přímá integrace s IDE (např. Cursor), správa v Markdownu | Implementováno | Implementováno | Plánováno | Není podporováno |

## 1.3 DevOps a Nasazení: Zralý a Uživatelsky Přívětivý Základ

**Kontejnerizace jako Základní Princip** Analýza projektu ukazuje, že Docker není pouhým doplňkem, ale primární a doporučenou metodou nasazení. Dokumentace README.md poskytuje podrobné a okamžitě použitelné příkazy docker run pro různé konfigurace, včetně perzistentního úložiště na souborovém systému nebo připojení k databázi PostgreSQL. To svědčí o zaměření na snadné a opakovatelné nasazení.

**Orchestrace pro Reálné Scénáře** Použití souboru docker-compose.yml pro orchestraci více služeb (např. spuštění serveru společně s databází PostgreSQL) je dalším znakem provozní zralosti. Projekt navíc obsahuje sadu manažerských skriptů (start, stop, test, build), které dále zjednodušují životní cyklus aplikace a snižují zátěž pro vývojáře a DevOps týmy.

**Kontinuální Integrace a Nasazení (CI/CD)** Přítomnost adresáře .github/workflows a konkrétního souboru docker-publish.yml potvrzuje existenci automatizovaného CI/CD pipeline. Lze předpokládat, že tento pipeline zajišťuje sestavení, testování a publikaci Docker obrazů do registru. Záznamy v CHANGELOG.md navíc zmiňují vylepšení, jako jsou "multi-stage builds" , což naznačuje aktivní snahu o optimalizaci a zabezpečení procesu sestavování.

Uživatelská zkušenost (UX) tohoto projektu se tak neomezuje pouze na jeho API, ale rozšiřuje se na celý životní cyklus nasazení a provozu. Zatímco typický open-source projekt může poskytnout pouze zdrojový kód a základní instrukce pro sestavení, mcp-prompts nabízí předpřipravené Docker obrazy, několik konfigurací docker-compose a manažerské skripty. To dramaticky snižuje vstupní bariéru pro potenciální uživatele. Vývojář může zprovoznit instanci podobnou produkčnímu prostředí jediným příkazem, aniž by musel ručně konfigurovat databázi, nastavovat proměnné prostředí a sestavovat kód od nuly. Tento důraz na provozní jednoduchost je sám o sobě klíčovou funkcí. Demonstruje hluboké porozumění celému pracovnímu postupu koncového uživatele (vývojáře nebo DevOps inženýra), což činí projekt mnohem atraktivnějším

pro rychlé přijetí a experimentování.

# Část II: Fázovaný Plán Implementace a Vylepšení

Tato část převádí strategickou analýzu do granulárního a akceschopného plánu. Je prezentována ve formátu požadovaném uživatelem a poskytuje jasnou cestu k dokončení robustního, funkčně kompletního a spolehlivého serveru. Každá fáze představuje logické seskupení úkolů, které na sebe navazují.

## TODOs

## Fáze 1: Dokončení základní architektury

- [ ] Implementovat rozhraní MutablePrompt pro všechny adaptéry úložišť (file, postgres, mdc) k zajištění konzistentní konverze formátů napříč všemi back-endy.
- [ ] Dokončit implementaci PostgreSQL adaptéru s plnou podporou pro PGAI formát, včetně ukládání a načítání vektorových embeddings.
- [ ] Implementovat MDC adaptér pro plnou obousměrnou podporu formátu Cursor Rules, nejen jako úložiště, ale i jako cíl konverze.
- [ ] Vytvořit jednotný a konfigurovatelný systém logování (např. s využitím Winston nebo Pino) napříč všemi komponentami serveru pro lepší ladění a monitorování.
- [ ] Implementovat robustní a standardizovaný mechanismus zpracování chyb, který poskytuje detailní a užitečné chybové zprávy klientům v souladu se specifikací JSON-RPC.

## Fáze 2: Rozšíření funkcionality šablon

- [ ] Implementovat pokročilý engine pro zpracování šablon, který podporuje podmíněnou logiku (if/else) a cykly pro tvorbu složitějších dynamických promptů.
- [ ] Přidat konfigurovatelnou podporu pro různé styly oddělovačů proměnných (např. {{handlebars}}, ${dollar-style}, <python-style>) pro zvýšení kompatibility s existujícími systémy.
- [ ] Vytvořit systém pro validaci proměnných v šablonách s podporou typové kontroly (string, number, boolean) a povinných polí, jak je definováno ve specifikaci MCP.
- [ ] Implementovat serverovou funkci pro automatickou extrakci všech unikátních proměnných z obsahu šablony, což usnadní tvorbu klientských UI.
- [ ] Přidat podporu pro vnořené šablony (partials), které umožní znovupoužití menších částí promptů v rámci větších a komplexnějších šablon.

## Fáze 3: Integrace s MCP ekosystémem

- [ ] Implementovat plnohodnotný Resource URI Parser schopný zpracovávat odkazy na externí zdroje (např. github://, file://) v souladu se specifikací MCP.
- [ ] Vytvořit Resource Router, který na základě URI dynamicky směruje požadavky na příslušné handlery zdrojů, což umožní serveru fungovat jako agregátor kontextu.
- [ ] Implementovat základní handlery pro klíčové typy zdrojů, minimálně pro @filesystem a @github, aby server mohl obohacovat prompty o obsah z externích systémů.

- [ ] Přidat podporu pro fallback strategie, kdy v případě nedostupnosti primárního zdroje (např. pád GitHub API) může server použít záložní nebo cachovaná data.
- [ ] Implementovat mechanismus pro obousměrnou synchronizaci promptů s dedikovaným GitHub repozitářem, což umožní Git-based workflow pro správu promptů.

## Fáze 4: Testování a optimalizace

- [ ] Vytvořit komplexní sadu unit testů s vysokým pokrytím kódu pro všechny klíčové komponenty, zejména pro logiku MutablePrompt a jednotlivé adaptéry.
- [ ] Implementovat sadu integračních testů s využitím Docker Compose, které ověří end-to-end funkčnost celého systému, včetně interakce s reálnou databází PostgreSQL.
- [ ] Vytvořit performance testy pro zátěžové testování API, specificky pro scénáře s velkým počtem promptů a častým voláním apply_template.
- [ ] Implementovat efektivní mechanismus pro stránkování (pagination) v list_prompts API endpointu, aby se zabránilo problémům s výkonem při načítání tisíců promptů.
- [ ] BUG: Opravit a přidat regresní test pro problém s file adaptérem na některých souborových systémech, jak je zmíněno v CHANGELOG.

## Fáze 5: Dokumentace a nasazení

- [ ] Vytvořit podrobnou a aktuální dokumentaci API s příklady použití pro každý endpoint. Zvážit automatické generování z OpenAPI specifikace, jejíž existence je cílem projektu.
- [ ] Aktualizovat README.md a vytvořit komplexní USER_GUIDE.md, který podrobně popisuje všechny funkce, konfigurace a pokročilé scénáře použití.
- [ ] Zavést a důsledně udržovat CHANGELOG.md pro transparentní sledování změn v každé verzi.
- [ ] Optimalizovat Dockerfile s využitím multi-stage builds pro minimalizaci velikosti finálního produkčního obrazu, jak již bylo započato.
- [ ] Vylepšit a zdokumentovat konfigurace v docker-compose.yml pro různé scénáře nasazení (vývoj, testování, produkce, plný ekosystém s ostatními MCP servery).
- [ ] Připravit a publikovat verzi 1.0.0 podle principů sémantického verzování, včetně aktualizace package.json a vytvoření Git tagu.

## Fáze 6: Pokročilé funkce a integrace

- [ ] Implementovat plnohodnotnou podporu pro vektorové vyhledávání s využitím PostgreSQL a jeho pg_vector nebo pg_embedding rozšíření, čímž se naplní vize PGAI formátu.
- [ ] Přidat podporu pro dávkové operace (batch processing) pro efektivní přidávání, úpravu a aplikaci šablon na velké množství promptů najednou.
- [ ] Vytvořit integraci s mcp-project-orchestrator serverem pro umožnění komplexních workflow, kde tento server spravuje šablony, které orchestrátor používá k vytváření projektů.
- [ ] Implementovat podporu pro Server-Sent Events (SSE) pro notifikace o změnách v promptech v reálném čase (např. prompts/list_changed), což je běžný pattern v MCP ekosystému.
- [ ] Prozkoumat a implementovat pokročilé bezpečnostní funkce, jako je Role-Based Access Control (RBAC) na úrovni jednotlivých promptů nebo kategorií, jak je naznačeno v

cílech projektu.

# Část III: Strategická Doporučení a Závěrečné Poznámky

Tato závěrečná část poskytuje strategické vodítko pro realizaci navrženého plánu a shrnuje celkovou hodnotu a potenciál projektu.

## 3.1 Prioritizace a Strategie Realizace

**Zaměření na Stabilní Jádro pro Verzi 1.0** Důrazně se doporučuje prioritizovat dokončení **Fáze 1 (Dokončení základní architektury)** a **Fáze 4 (Testování a optimalizace)**. Stabilní, spolehlivý a důkladně otestovaný server, i s omezenou sadou funkcí, je pro komunitu a potenciální adoptující mnohem cennější než široce funkční, ale nestabilní produkt. Oprava chyby zmíněná v CHANGELOG týkající se souborového adaptéru podtrhuje nutnost této stabilizace.
**Dodání Klíčové Hodnoty** Základní funkce z **Fáze 2 (Rozšíření funkcionality šablon)**, jako je podpora pro základní podmíněnou logiku a validaci proměnných, by měly být součástí verze 1.0, aby byl server okamžitě užitečný a demonstroval svou hlavní hodnotu.
**Fázované Zavádění Pokročilých Funkcí** Úkoly z **Fáze 3 (Integrace s MCP ekosystémem)**, **Fáze 5 (Dokumentace a nasazení)** a **Fáze 6 (Pokročilé funkce a integrace)** by měly být plánovány pro následné verze (např. 1.1, 1.2). Tento iterativní přístup umožňuje rychlejší dodání stabilního jádra produktu a zároveň sběr zpětné vazby od uživatelů, která může lépe nasměrovat vývoj složitějších funkcí, jako je plná ekosystémová integrace nebo vektorové vyhledávání.
Navržený plán přirozeně odráží typickou křivku zralosti infrastrukturního softwaru. Projekt začíná funkčním prototypem (současný stav). Navržený plán se nejprve zaměřuje na posílení a stabilizaci tohoto jádra (Fáze 1 a 4), aby byl spolehlivý. Toto je fáze "stabilizace". Následně rozšiřuje základní funkcionalitu, aby byla výkonnější (Fáze 2 a 5) a interoperabilní (Fáze 3). Toto je fáze "rozšiřování funkcí a integrace". Nakonec přidává špičkové, vysoce hodnotné funkce, které jej odlišují od konkurence (Fáze 6). Toto je fáze "inovace". Porozumění tomuto klasickému vývojovému vzoru umožňuje efektivně řídit očekávání zúčastněných stran a alokovat zdroje tak, aby byl nejprve vybudován pevný základ, než se přistoupí k ambicióznějším a potenciálně rizikovějším funkcím.

## 3.2 Závěrečná Analýza: Klíčová Komponenta pro Budoucnost Vývoje AI

**Shrnutí Silných Stránek** Na závěr je třeba znovu zdůraznit klíčové silné stránky projektu: silná architektonická vize soustředěná kolem rozhraní MutablePrompt , zralá DevOps a deployment strategie, která klade důraz na uživatelskou zkušenost , a strategické sladění s rostoucím a důležitým otevřeným standardem MCP.
**Potenciální Dopad** Server má vynikající pozici k tomu, aby se stal kritickou součástí infrastruktury pro jakýkoli tým, který staví aplikace na MCP stacku. Tím, že řeší hmatatelný problém fragmentace promptů standardizovaným způsobem, má potenciál urychlit vývoj, zlepšit spolupráci a zvýšit spolehlivost aplikací poháněných umělou inteligencí.
**Závěrečné Slovo** Projekt sparesparrow/mcp-prompts je více než jen repozitář s kódem; je to

promyšlené řešení naléhavého problému v oboru s jasnou cestou k tomu, aby se stal nejlepším ve své třídě a produkčně připraveným systémem. Úspěšná realizace navrženého plánu upevní jeho roli jako klíčové komponenty v budoucnosti modulárního a interoperabilního vývoje AI.

## Works cited

1. sparesparrow/mcp-prompts: Model Context Protocol server for managing, storing, and providing prompts and prompt templates for LLM interactions. - GitHub, https://github.com/sparesparrow/mcp-prompts 2. A beginners Guide on Model Context Protocol (MCP) - OpenCV, https://opencv.org/blog/model-context-protocol/ 3. Model Context Protocol (MCP) an overview - Philschmid, https://www.philschmid.de/mcp-introduction 4. Model context protocol (MCP) - OpenAI Agents SDK, https://openai.github.io/openai-agents-python/mcp/ 5. Model Context Protocol (MCP) - Anthropic API, https://docs.anthropic.com/en/docs/mcp 6. How does MCP work - Vellum AI, https://www.vellum.ai/blog/how-does-mcp-work 7. MCP Prompts Server - Glama, https://glama.ai/mcp/servers/@sparesparrow/mcp-prompts 8. Core architecture - Model Context Protocol, https://modelcontextprotocol.io/docs/concepts/architecture 9. MCP Prompts: Guide to Structured LLM Interactions - BytePlus, https://www.byteplus.com/en/topic/541190 10. Stop Losing Prompts: Build Your Own MCP Prompt Registry - DEV Community, https://dev.to/stevengonsalvez/stop-losing-prompts-build-your-own-mcp-prompt-registry-4fi1 11. Prompts - Model Context Protocol, https://modelcontextprotocol.io/docs/concepts/prompts 12. Specification - Model Context Protocol, https://modelcontextprotocol.io/specification/2025-03-26 13. MCP Prompt Testing: Tools, Tips & Best Practices - BytePlus, https://www.byteplus.com/en/topic/542175 14. The Ultimate Guide to MCP Servers: Best Options for Building AI-Ready Apps - Treblle, https://treblle.com/blog/mcp-servers-guide 15. Best Practices for Building MCP Servers | Anthropic Help Center, https://support.anthropic.com/en/articles/11596040-best-practices-for-building-mcp-servers 16. sparesparrow/mcp-project-orchestrator - GitHub, https://github.com/sparesparrow/mcp-project-orchestrator 17. sparesparrow/mcp-prompts-rs: Rust-based server for managing AI prompts using the Model Context Protocol (MCP) - GitHub, https://github.com/sparesparrow/mcp-prompts-rs 18. An under the hood look at how Pieces implemented an MCP server, https://pieces.app/blog/inside-mcp-server-implementation