Analýza a Strategický Plán Rozvoje pro sparesparrow/mcp-prompts

Shrnutí

Tato zpráva předkládá hloubkovou analýzu serveru sparesparrow/mcp-prompts a navrhuje strategický plán pro jeho další vývoj. Primárním cílem je posílit a dokončit klíčové funkcionality související s ukládáním promptů do souborů JSON a se systémem šablon. Analýza objasňuje, že explicitní úkoly ve formě souborů TODO- v adresáři .cursor/rules neexistují; místo toho jsou požadavky na vývoj odvozeny z implicitní struktury a obsahu souborů v samostatném repozitáři sparesparrow/cursor-rules. Tento poznatek je využit k definování pokročilých funkcí pro server. Priorita vývoje adaptéru pro PostgreSQL je v souladu se zadáním odložena. Závěrem zprávy je sada konkrétních, plně specifikovaných GitHub issues, které tvoří jasný a okamžitě použitelný akční plán pro následující vývojový cyklus.

Sekce 1: Strategická analýza architektury serveru mcp-prompts

Tato sekce zasazuje server mcp-prompts do technického a strategického kontextu. Analyzuje jeho architekturu ve vztahu k širšímu standardu Model Context Protocol (MCP) a hodnotí současný stav jeho klíčových komponent, čímž připravuje půdu pro podrobná doporučení v následujících částech.

1.1 Umístění mcp-prompts v širším ekosystému MCP

- **Základní koncept:** Model Context Protocol (MCP) je otevřený standard, představený společností Anthropic, navržený jako univerzální rozhraní – přirovnávané k „USB-C pro AI“ – mezi aplikacemi využívajícími LLM (klienty) a externími zdroji dat nebo nástroji (servery). Standardizuje tuto interakci pomocí zpráv ve formátu JSON-RPC 2.0 , čímž řeší problém integrace „N×M“, kde každá aplikace potřebuje vlastní konektor pro každý nástroj.
- **Role serveru:** MCP servery zpřístupňují tři hlavní typy schopností: **Nástroje (Tools)** (funkce, které může AI vykonávat), **Zdroje (Resources)** (data, která může AI nebo uživatel použít) a **Prompty (Prompts)** (šablonované zprávy a pracovní postupy).
- **Specializace projektu:** Server sparesparrow/mcp-prompts si v tomto ekosystému nachází specifickou a klíčovou pozici. Zaměřuje se výhradně na schopnost „Prompts“ s cílem stát se špičkovým řešením pro správu, ukládání, verzování a dynamické poskytování šablon promptů. Tím přímo řeší běžný problém v komplexních LLM pracovních postupech: správu roztroušených, natvrdo zakódovaných a obtížně aktualizovatelných promptů.
- **Kompatibilita:** Uváděná kompatibilita s klienty jako LM Studio, LibreChat a Claude Desktop dokazuje dodržování standardu MCP, což z něj činí cennou komponentu v širším ekosystému MCP nástrojů.

Zaměření projektu na úzkou specializaci je jeho strategickou výhodou, nikoli omezením. Zatímco mnoho MCP serverů se snaží být univerzálním nástrojem pro různé integrace , mcp-prompts se specializuje na jednu ze tří klíčových vlastností MCP. To mu umožňuje řešit jeden problém – správu životního cyklu promptů – výjimečně dobře. V komplexních agentních systémech je správa a inženýrství promptů samostatnou disciplínou. Díky svému zaměření se mcp-prompts může stát definitivním serverem pro každého vývojáře, který potřebuje robustní řešení pro správu promptů, a etablovat se jako klíčový, kompozitní stavební blok ve větší AI architektuře.

1.2 Hloubková analýza architektury: Adapter Factory a oddělené úložiště

- **Základní návrhový vzor:** Soubor README.md explicitně zmiňuje **Adapter Factory** a **Storage Adapters** (File, Postgres, MDC). Jedná se o klíčové architektonické rozhodnutí.
- **Mechanismus:** Server používá proměnnou prostředí STORAGE_TYPE k určení, který adaptér úložiště se má při spuštění inicializovat. To je v souladu s principem inverze závislostí (Dependency Inversion Principle), kde logika vyšší úrovně (server) není svázána s implementací nižší úrovně (úložiště).
- **Výhody:** Tento návrh poskytuje obrovskou flexibilitu a rozšiřitelnost. Umožňuje uživatelům měnit backend (např. přechod z jednoduchého souborového systému pro vývoj na robustní databázi PostgreSQL pro produkci) pouhou změnou konfigurace. Zároveň činí přidávání nových backendů (např. Redis, S3 nebo vektorová databáze) modulárním úkolem bez nutnosti měnit jádro serverové logiky.

Vzor Adapter Factory je klíčem k budoucí udržitelnosti a škálovatelnosti projektu. Umožňuje, aby požadavek na prioritizaci souborového adaptéru a odložení práce na PostgreSQL byl taktickou volbou, nikoli trvalým architektonickým omezením. Kód pro PostgreSQL adaptér může zůstat neaktivní, aniž by ovlivnil vývoj souborového adaptéru. Jak bude projekt zrát, objeví se nové potřeby v oblasti ukládání dat (např. README.md zmiňuje vektorové vyhledávání pro sémantické hledání promptů jako plánovanou funkci). Díky tomuto vzoru lze vytvořit nový VectorDBAdapter a integrovat jej bez nutnosti refaktorovat celou aplikaci. Tato prozíravost v původním návrhu znamená, že se projekt může snadno přizpůsobit rychle se vyvíjejícím potřebám AI vývoje.

1.3 Analýza souborového adaptéru úložiště: Současný stav a odvozené schopnosti

- **Současná implementace:** Na základě README.md používá souborový adaptér pro ukládání adresář specifikovaný proměnnou PROMPTS_DIR. Implicitně se předpokládá, že prompty jsou ukládány v jednotlivých souborech JSON. Dokumentace uvádí API/nástroje pro operace list, get, add, update a delete, které musí souborový adaptér implementovat.
- **Odvozená omezení:** Současný popis naznačuje jednoduchou, přímou implementaci souborových I/O operací. Ačkoli je to funkční pro jednoho uživatele nebo scénáře s nízkou mírou souběžnosti, tento přístup pravděpodobně postrádá robustnost pro produkční nasazení. Mezi klíčové neřešené výzvy patří:
    - **Integrita dat:** Není zmíněna validace schématu. Špatně formátovaný soubor JSON by mohl při čtení způsobit pád serverového procesu.

    - **Souběžnost:** Bez zamykání souborů by souběžné operace zápisu (např. dva uživatelé aktualizující stejný prompt přes API) mohly vést k tzv. race conditions a poškození dat.
    - **Škálovatelnost:** Výpis všech promptů pravděpodobně zahrnuje čtení každého souboru v adresáři, což se stane pomalým, jakmile počet promptů vzroste na tisíce. Indexační mechanismy nejsou zmíněny.
    - **Atomické operace:** Operace s více soubory (např. přejmenování promptu, které mění název souboru) není atomická. Pokud proces selže v polovině, systém zůstane v nekonzistentním stavu.

1.4 Analýza enginu pro šablonování promptů: Funkčnost a omezení

- **Současná implementace:** Server disponuje „výkonným šablonovacím“ enginem používajícím syntaxi {{variable}}. API zahrnuje koncový bod pro aplikaci šablon, což ukazuje, že substituce proměnných je klíčovou funkcí.
- **Odvozená omezení:** Syntaxe {{variable}} naznačuje jednoduchý mechanismus nahrazování řetězců, podobný Mustache nebo základnímu Handlebars. Ačkoli je to užitečné, jedná se o základní úroveň šablonování. Pokročilé agentní pracovní postupy často vyžadují sofistikovanější logiku přímo v promptech. Pravděpodobně chybějící funkce zahrnují:
    - **Podmíněná logika:** Není zmínka o blocích if/else pro zahrnutí/vyloučení částí promptu na základě vstupních proměnných.
    - **Cykly/Iterace:** Není zmínka o iteraci přes seznamy pro dynamické generování sekcí promptu (např. vytvoření číslovaného seznamu příkladů z pole).
    - **Pomocné funkce:** Není zmínka o vestavěných funkcích (např. pro formátování dat, manipulaci s řetězci nebo provádění jednoduchých výpočtů) v rámci šablon.
    - **Přístup k vnořeným/komplexním datům:** Není zmínka o přístupu k vnořeným vlastnostem objektů (např. {{user.profile.name}}).

1.5 Stav odprioritizovaných komponent: Adaptér pro PostgreSQL

- **Pokyn uživatele:** Požadavkem je odprioritizovat práci na adaptéru pro PostgreSQL.
- **Současný stav:** README.md potvrzuje jeho existenci jako konfigurovatelného STORAGE_TYPE s dedikovanými proměnnými prostředí pro připojení. Je uveden jako „rozšiřitelná možnost úložiště“.
- **Akce:** V souladu s požadavkem uživatele tato zpráva nenavrhne žádný nový vývoj pro tento adaptér. Jeho existence však potvrzuje sílu architektury Adapter Factory a slouží jako důkaz konceptu pro integraci alternativních, škálovatelnějších backendů.

Sekce 2: Prozkoumání vývojových direktiv v .cursor/rules

Tato sekce se přímo zabývá dotazem uživatele ohledně souborů TODO-. Objasňuje klíčové nedorozumění ohledně struktury projektu a přechází k produktivnější analýze implicitních požadavků.

2.1 Objasnění struktury repozitářů: Vztah mezi mcp-prompts a

sparesparrow/cursor-rules

- **Původní předpoklad dotazu:** Dotaz uživatele předpokládal, že adresář .cursor/rules a jeho soubory TODO- se nacházejí v repozitáři sparesparrow/mcp-prompts.
- **Zjištění z průzkumu:** Průzkum tento předpoklad vyvrací. Repozitář mcp-prompts sice uvádí adresář .cursor/rules, ale ten je určen pro *použití* adaptérem MDC, typicky připojeným přes Docker volume. Skutečný zdroj a definice těchto pravidel se nachází v samostatném repozitáři: sparesparrow/cursor-rules.
- **„MDC adaptér“:** Server mcp-prompts má adaptér úložiště typu mdc. Tento adaptér je navržen tak, aby četl soubory pravidel .mdc (formát podobný Markdownu) z repozitáře cursor-rules a zacházel s nimi jako se zdrojem promptů. Tím se vytváří jasný vztah spotřebitel-producent mezi oběma projekty.

2.2 Zjištění: Absence explicitních souborů s úkoly TODO-

- **Vyčerpávající prohledávání:** Důkladná kontrola seznamů souborů v obou repozitářích, mcp-prompts i cursor-rules , stejně jako cílené vyhledávání na GitHubu , potvrzuje, že **v žádném z repozitářů neexistují žádné soubory s prefixem TODO-**.
- **Závěr:** Předpoklad dotazu uživatele ohledně explicitních souborů TODO- je nesprávný. Žádné vývojové úkoly nebyly formálně zdokumentovány tímto způsobem.

2.3 Alternativní analýza: Odvození implicitních úkolů ze souborů pravidel .mdc

Soubory .mdc v repozitáři sparesparrow/cursor-rules nejsou jen data; jsou to specifikace, které implikují funkční požadavky na server mcp-prompts. Místo pouhého konstatování, že soubory TODO- neexistují, je produktivnější analyzovat, co existuje: samotné soubory .mdc. Tyto soubory mají definovanou strukturu s metadaty (např. description, globs, priority, dependencies) a obsahovými sekcemi (např. Core Principles, Code Standards). Aby byl „MDC adaptér“ v mcp-prompts užitečný, musí být schopen tato strukturovaná data parsovat, nikoli jen zacházet se souborem jako s monolitickým blokem textu. Například agent LLM by mohl potřebovat vyžádat si pouze sekci Validation Rules z pravidla solid-analyzer.mdc. To implikuje požadavek, aby server podporoval načítání dílčích sekcí nebo strukturovaného obsahu z těchto souborů. Struktura souborů .mdc tedy implicitně definuje sadu funkcí pro MDC adaptér v mcp-prompts.

- **Implicitní požadavky na MDC adaptér:**
    - **Parsování metadat:** Adaptér musí být schopen parsovat a využívat frontmatter ve formátu podobném YAML v každém souboru .mdc (např. priority, dependencies). To by mohlo být použito pro filtrování nebo řazení promptů.
    - **Přístup ke strukturovanému obsahu:** Server by v ideálním případě měl být schopen poskytovat specifické sekce souboru s pravidly (např. ## Code Standards) jako nezávislé bloky obsahu, nejen celý soubor.
    - **Zpracování závislostí:** Pole dependencies v souborech .mdc implikuje potřebu, aby server zvládal vztahy mezi prompty. Požadavek na mcp-typescript.mdc by mohl vyžadovat automatické zahrnutí obsahu z jeho závislosti, typescript.mdc. To představuje významnou funkci pro šablonování a kompozici.

Sekce 3: Plán rozvoje pro klíčové funkcionality

Tato sekce převádí analýzu do konkrétního, prioritizovaného plánu vývoje. Zaměřuje se na zdokonalení dvou funkcí identifikovaných v dotazu uživatele: souborového adaptéru úložiště JSON a enginu pro šablonování promptů.

3.1 Plánovaná položka 1: Posílení souborového adaptéru úložiště JSON

Cílem je vyvinout souborový adaptér z jednoduchého důkazu konceptu na robustní, spolehlivý a produkčně připravený mechanismus úložiště.

- **Fáze 1: Základní robustnost**
  - **Validace schématu:** Implementovat striktní validaci schématu pro všechna příchozí data promptů (při add/update) a pro soubory čtené z disku. Tím se zabrání poškození dat a zajistí předvídatelné chování. Vhodná by byla knihovna jako Zod nebo Ajv.
  - **Řízení souběžnosti:** Zavést mechanismus zamykání souborů (např. pomocí knihovny jako proper-lockfile), aby se předešlo race conditions během operací zápisu. To je klíčové pro jakékoli prostředí s více uživateli nebo řízené přes API.
- **Fáze 2: Výkon a škálovatelnost**
  - **Indexace:** Pro operaci list vytvořit a udržovat jediný indexový soubor (např. _index.json), který ukládá metadata pro všechny prompty. Tím se zabrání čtení každého jednotlivého souboru z adresáře při každém požadavku na výpis, což dramaticky zlepší výkon u velkých sad promptů. Index by byl aktualizován při každé operaci add/update/delete.
  - **Asynchronní I/O:** Zajistit, aby všechny souborové operace byly plně asynchronní, aby se neblokovala smyčka událostí Node.js a udržela se odezva serveru pod zátěží.
- **Fáze 3: Pokročilé funkce**
  - **Transakční zápisy:** Implementovat vzor „zápis do dočasného souboru a následné přejmenování“ pro aktualizace. Tím se zajistí, že operace aktualizace jsou atomické, což zabrání poškození souborů, pokud server selže uprostřed zápisu.
  - **Historie změn/Verzování (Volitelné rozšíření):** Zvážit jednoduchý systém verzování, kde se staré verze promptu ukládají (např. prompt-name.1.json, prompt-name.2.json) místo toho, aby byly přepsány. To by mohla být silná funkce pro auditování a návrat k předchozím verzím.

3.2 Plánovaná položka 2: Vývoj enginu pro šablonování promptů

Cílem je přeměnit šablonovací engine ze základního nástroje pro nahrazování proměnných na výkonný nástroj pro dynamickou konstrukci promptů, což umožní komplexnější a inteligentnější agentní pracovní postupy.

- **Fáze 1: Pokročilé řízení toku**
  - **Podmíněná logika:** Implementovat konstrukce ve stylu if/else. Syntaxe by mohla být inspirována Handlebars: {{#if user.isAdmin}}...{{else}}...{{/if}}. To umožní promptům přizpůsobit se různým kontextům.

    - **Iterace/Cykly:** Implementovat cyklické konstrukce pro iteraci přes pole dat. Syntaxe: {{#each items}}...{{this.name}}...{{/each}}. To je nezbytné pro dynamické generování seznamů, příkladů nebo few-shot promptů.
- **Fáze 2: Zpracování dat a použitelnost**
    - **Přístup k vnořeným proměnným:** Zajistit, aby engine mohl přistupovat k vnořeným vlastnostem v objektech JSON ({{user.profile.name}}) a k prvkům pole podle indexu ({{items}}).
    - **Pomocné funkce:** Zavést knihovnu vestavěných pomocných funkcí. Příklady: {{formatDate date "YYYY-MM-DD"}}, {{toUpperCase "string"}}, {{jsonStringify object}}. Tím se sníží potřeba předzpracovávat data před jejich odesláním do šablony.
- **Fáze 3: Kompozice a znovupoužitelnost**
    - **Částečné šablony/Vkládání (Partials/Includes):** Implementovat mechanismus pro vkládání jedné šablony promptu do druhé ({{> my_partial}}). To podporuje principy DRY (Don't Repeat Yourself) a umožňuje vytvoření znovupoužitelné knihovny komponent z úryvků promptů. Tím se přímo řeší implicitní požadavek na zpracování závislostí, který byl odvozen z analýzy souborů .mdc.

Sekce 4: Akční plán ve formě GitHub Issues

Tato sekce poskytuje sadu plně specifikovaných GitHub issues, připravených k zařazení do backlogu projektu. Tyto issues přímo odpovídají položkám plánu rozvoje definovaným v Sekci 3.

4.1 Souhrnná tabulka navrhovaných Issues

Tato tabulka slouží jako projektový plán na vysoké úrovni a shrnutí vývojového úsilí. Umožňuje vedoucímu projektu rychle pochopit rozsah práce, přiřadit priority a sledovat pokrok v klíčových oblastech (Úložiště a Šablonování).

Priorita	Oblast	Název Issue	Štítky
Kritická	Správa projektu	**Epic: Finalize Core JSON Storage & Templating**	epic, roadmap
Vysoká	Úložiště: Soubor	**Feat: Implement Schema Validation for Prompt JSON Files**	enhancement, storage:file, robustness
Vysoká	Úložiště: Soubor	**Feat: Add File Locking to Prevent Write Race Conditions**	enhancement, storage:file, robustness
Vysoká	Šablonování	**Feat: Advanced Templating - Conditional Logic (if/else)**	enhancement, templating, feature
Střední	Úložiště: Soubor	**Perf: Create Metadata Index for Fast Prompt Listing**	enhancement, storage:file, performance
Střední	Šablonování	**Feat: Advanced**	enhancement,

Priorita	Oblast	Název Issue	Štítky
		Templating - Iteration/Loops (each)	templating, feature
Střední	Šablonování	**Feat: Advanced Templating - Partials/Includes**	enhancement, templating, feature
Nízká	Úložiště: Soubor	**Feat: Implement Atomic Writes via Temp File Strategy**	enhancement, storage:file, robustness
Nízká	Šablonování	**Feat: Create Library of Built-in Template Helper Functions**	enhancement, templating, feature

4.2 Podrobný rozpis Issues

4.2.1 Epic: Finalize Core JSON Storage & Templating

- **Název:** Epic: Finalize Core JSON Storage & Templating
- **Popis:** Tento epic sleduje zastřešující cíl zdokonalit výchozí souborový adaptér úložiště a engine pro šablonování promptů do produkčně připraveného stavu. Zahrnuje všechny podřízené issues související s robustností, výkonem a rozšiřováním funkcí těchto klíčových komponent.
- **Kritéria přijetí:**
  - Všechny podřízené issues spojené s tímto epicem jsou uzavřeny.
  - Souborový adaptér úložiště je odolný vůči poškození dat a souběžnému přístupu.
  - Šablonovací engine podporuje řízení toku, iteraci a kompozici.
  - Pro všechny nové funkcionality jsou zavedeny end-to-end testy.
- **Štítky:** epic, roadmap

4.2.2 Feat: Implement Schema Validation for Prompt JSON Files

- **Název:** Feat: Implement Schema Validation for Prompt JSON Files
- **Popis:**
  - **Jako** vývojář používající server MCP-Prompts,
  - **chci,** aby server validoval strukturu všech souborů JSON s prompty proti striktnímu schématu při operacích čtení a zápisu,
  - **aby** byla špatně formátovaná nebo poškozená data včas odmítnuta, což zabrání běhovým chybám a zajistí integritu dat.
- **Kritéria přijetí:**
  1. Je definováno formální schéma JSON (nebo ekvivalent pomocí knihovny jako Zod) pro strukturu promptu.
  2. Koncové body API add a update validují příchozí data proti tomuto schématu a v případě neúspěšné validace vrátí chybu 400 Bad Request.
  3. Při spuštění server validuje existující soubory s prompty v PROMPTS_DIR. Měl by zaznamenat varování pro všechny neplatné soubory, ale neměl by havarovat.
  4. Koncový bod get pro známý neplatný soubor by měl vrátit chybu nebo surový obsah s varováním.
- **Štítky:** enhancement, storage:file, robustness, priority:high

4.2.3 Feat: Add File Locking to Prevent Write Race Conditions

- **Název:** Feat: Add File Locking to Prevent Write Race Conditions
- **Popis:**
  - **Jako** administrátor instance MCP-Prompts s více uživateli,
  - **chci,** aby souborový adaptér úložiště používal mechanismus zamykání souborů během všech operací zápisu (add, update, delete),
  - **aby** souběžné požadavky nepoškozovaly soubory s prompty a byla zajištěna konzistence dat.
- **Kritéria přijetí:**
  1. Do souborového adaptéru je integrována spolehlivá knihovna pro zamykání souborů (např. proper-lockfile).
  2. Před zápisem do souboru nebo jeho smazáním je získán zámek.
  3. Zámek je uvolněn po dokončení operace, i když dojde k chybě.
  4. Pokud nelze zámek získat v rozumném časovém limitu, požadavek API selže s příslušným stavovým kódem (např. 409 Conflict nebo 423 Locked).
  5. Jsou vytvořeny unit testy, které simulují souběžné pokusy o zápis a ověřují, že nedochází k poškození dat.
- **Štítky:** enhancement, storage:file, robustness, priority:high

4.2.4 Feat: Advanced Templating - Conditional Logic (if/else)

- **Název:** Feat: Advanced Templating - Conditional Logic (if/else)
- **Popis:**
  - **Jako** tvůrce promptů,
  - **chci** používat podmíněnou logiku (např. bloky if/else) v mých šablonách promptů,
  - **aby** se obsah promptu mohl dynamicky měnit na základě vstupních proměnných a kontextu.
- **Kritéria přijetí:**
  1. Šablonovací engine parsuje a správně interpretuje konstrukce podmíněné logiky (např. {{#if condition}}...{{else}}...{{/if}}).
  2. Podmínky mohou vyhodnocovat booleovské hodnoty, existenci proměnných a jednoduchá porovnání.
  3. Jsou přidány unit testy pokrývající různé scénáře podmíněné logiky, včetně vnořených podmínek.
- **Štítky:** enhancement, templating, feature, priority:high

4.2.5 Perf: Create Metadata Index for Fast Prompt Listing

- **Název:** Perf: Create Metadata Index for Fast Prompt Listing
- **Popis:**
  - **Jako** uživatel s velkým počtem promptů,
  - **chci,** aby operace výpisu (list) byla rychlá a efektivní bez ohledu na počet uložených promptů,
  - **aby** se zlepšila odezva a škálovatelnost serveru.
- **Kritéria přijetí:**
  1. Při spuštění a při každé operaci zápisu (add, update, delete) server udržuje jediný

indexový soubor (např. _index.json).
    2. Tento indexový soubor obsahuje klíčová metadata (např. ID, název, tagy, časová razítka) pro všechny prompty.
    3. Operace list čte pouze tento indexový soubor místo skenování celého adresáře.
    4. Je zaveden mechanismus pro obnovu indexu v případě, že je poškozen nebo nesynchronizovaný.
- **Štítky:** enhancement, storage:file, performance, priority:medium

4.2.6 Feat: Advanced Templating - Iteration/Loops (each)

- **Název:** Feat: Advanced Templating - Iteration/Loops (each)
- **Popis:**
    - **Jako** tvůrce promptů,
    - **chci** iterovat přes pole dat (seznamy) v mých šablonách,
    - **aby** bylo možné dynamicky generovat opakující se struktury, jako jsou seznamy příkladů, položky nebo few-shot prompty.
- **Kritéria přijetí:**
    1. Šablonovací engine podporuje cyklickou konstrukci (např. {{#each items}}...{{/each}}).
    2. V rámci cyklu je možné přistupovat k prvkům aktuální iterace (např. {{this}} nebo {{this.property}}).
    3. Jsou přidány testy pro iteraci přes prázdná pole, pole objektů a pole primitivních typů.
- **Štítky:** enhancement, templating, feature, priority:medium

4.2.7 Feat: Advanced Templating - Partials/Includes

- **Název:** Feat: Advanced Templating - Partials/Includes
- **Popis:**
    - **Jako** tvůrce promptů spravující komplexní sadu promptů,
    - **chci** možnost vkládat jednu šablonu promptu do druhé (partials),
    - **aby** bylo možné znovupoužívat běžné úryvky, dodržovat princip DRY a vytvářet kompozitní prompty.
- **Kritéria přijetí:**
    1. Je implementována syntaxe pro vkládání (např. {{> partial_name}}).
    2. Šablonovací engine dokáže načíst a vykreslit obsah zadané částečné šablony v místě vložení.
    3. Jsou podporovány vnořené částečné šablony.
    4. Je zaveden mechanismus pro detekci a prevenci rekurzivních vkládání.
- **Štítky:** enhancement, templating, feature, priority:medium

4.2.8 Feat: Implement Atomic Writes via Temp File Strategy

- **Název:** Feat: Implement Atomic Writes via Temp File Strategy
- **Popis:**
    - **Jako** administrátor systému,
    - **chci,** aby operace zápisu souborů byly atomické,
    - **aby** se zabránilo poškození souborů v případě selhání serveru nebo přerušení

procesu uprostřed zápisu.

- **Kritéria přijetí:**
  1. Při aktualizaci nebo přidávání promptu se obsah nejprve zapíše do dočasného souboru.
  2. Po úspěšném zápisu do dočasného souboru je tento soubor atomicky přejmenován na cílový název souboru.
  3. Tento mechanismus je použit pro všechny operace, které modifikují soubory s prompty.
- **Štítky:** enhancement, storage:file, robustness, priority:low

4.2.9 Feat: Create Library of Built-in Template Helper Functions

- **Název:** Feat: Create Library of Built-in Template Helper Functions
- **Popis:**
  - **Jako** tvůrce promptů,
  - **chci** mít k dispozici knihovnu vestavěných pomocných funkcí v šablonovacím enginu,
  - **aby** bylo možné provádět běžné transformace dat, jako je formátování řetězců, datumů nebo serializace JSON, přímo v šabloně.
- **Kritéria přijetí:**
  1. Je vytvořen rozšiřitelný mechanismus pro registraci pomocných funkcí.
  2. Je implementována základní sada funkcí (např. toUpperCase, toLowerCase, jsonStringify, formatDate).
  3. Dokumentace je aktualizována o seznam dostupných pomocných funkcí a příklady jejich použití.
- **Štítky:** enhancement, templating, feature, priority:low

Works cited

1. Model Context Protocol (MCP) - Anthropic API, https://docs.anthropic.com/en/docs/mcp 2. en.wikipedia.org, https://en.wikipedia.org/wiki/Model_Context_Protocol 3. Model Context Protocol (MCP): A comprehensive introduction for developers - Stytch, https://stytch.com/blog/model-context-protocol-introduction/ 4. Specification - Model Context Protocol, https://modelcontextprotocol.io/specification/2025-06-18 5. Model Context Protocol (MCP) an overview - Philschmid, https://www.philschmid.de/mcp-introduction 6. What is Model Context Protocol (MCP)? - IBM, https://www.ibm.com/think/topics/model-context-protocol 7. 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 8. Model Context Protocol - GitHub, https://github.com/modelcontextprotocol 9. wyattowalsh/languages.md at main - GitHub, https://github.com/wyattowalsh/wyattowalsh/blob/main/languages.md 10. What Is the Model Context Protocol (MCP) and How It Works - Descope, https://www.descope.com/learn/post/mcp 11. sparesparrow/cursor-rules: A library of rules for the Cursor ... - GitHub, https://github.com/sparesparrow/cursor-rules 12. awesome-mcp-servers/README.md at main - GitHub, https://github.com/TensorBlock/awesome-mcp-servers/blob/main/README.md