DOI: https://doi.org/10.36910/6775-2524-0560-2025-61-26
УДК 004.8:004.4
Петренко Сергій Вікторович, к.п.н., доцент
https://orcid.org/0000-0002-5311-0743
Рівненський державний гуманітарний університет, м. Рівне, Україна

MODEL CONTEXT PROTOCOL: КОНЦЕПТУАЛЬНІ ЗАСАДИ ТА РЕАЛІЗАЦІЯ НА JAVA

Петренко С. В. Model Context Protocol: концептуальні засади та реалізація на Java. У статті розглянуто Model Context Protocol (MCP) як новітній відкритий стандарт інтеграції великих мовних моделей (LLM) із зовнішніми даними, сервісами та інструментами. Наголошується, що однією з головних проблем LLM є обмеженість знань у момент навчання та неможливість безпосередньої взаємодії з реальним світом. MCP, запропонований компанією Anthropic у 2024 році, створює безпечний і стандартизований механізм комунікації між моделями та зовнішнім середовищем, забезпечуючи інтероперабельність, розширюваність і гнучкість інтеграцій. У роботі систематизовано концептуальні засади протоколу, серед яких архітектура «хост-клієнт-сервер», роль транспортних механізмів, а також підтримка трьох базових примітивів: resources, prompts і tools. Показано, що MCP формує уніфікований підхід до створення агентних систем, які здатні отримувати контекст у режимі реального часу, виконувати дії у зовнішніх сервісах та значно знижувати вартість інтеграційних рішень. Окремо проаналізовано практичний аспект - реалізацію прототипу MCP-сервера на Java з використанням Spring AI та офіційного MCP Java SDK. Наведено приклади експонування ресурсів, підключення інструментів, створення промптів і тестування за допомогою MCP Inspector. Отримані результати підтверджують ефективність протоколу у завданнях інженерії програмного забезпечення, демонструючи продуктивність, зручність, масштабованість та надійність інтеграцій. Таким чином, стаття робить внесок у розвиток теорії й практики побудови інтелектуальних систем, що виходять за межі статичного знання та відкривають новий рівень автоматизації й співпраці між людиною та штучним інтелектом.
Ключові слова: Model Context Protocol, великі мовні моделі, Spring AI, Java, MCP сервер, MCP клієнт, ресурси, інструменти, промпти.

Petrenko S. Model Context Protocol: conceptual foundations and implementation in Java. The article discusses the Model Context Protocol (MCP) as the latest open standard for integrating large language models (LLMs) with external data, services, and tools. It emphasizes that one of the main problems with LLMs is their limited knowledge at the time of training and their inability to interact directly with the real world. MCP, proposed by Anthropic in 2024, creates a secure and standardized mechanism for communication between models and the external environment, ensuring interoperability, scalability, and flexibility of integrations. The paper systematizes the conceptual foundations of the protocol, including the host-client-server architecture, the role of transport mechanisms, and support for three basic primitives: resources, prompts, and tools. It is shown that MCP forms a unified approach to creating agent systems that are capable of obtaining context in real time, performing actions in external services, and significantly reducing the cost of integration solutions. A practical aspect is analyzed separately: the implementation of an MCP server prototype in Java using Spring AI and the official MCP Java SDK. Examples are given of exposing resources, connecting tools, creating prompts, and testing using MCP Inspector. The results confirm the effectiveness of the protocol in software engineering tasks, demonstrating the performance, convenience, scalability, and reliability of integrations. Thus, the article contributes to the development of the theory and practice of building intelligent systems that go beyond static knowledge and open up a new level of automation and collaboration between humans and artificial intelligence.
Keywords: Model Context Protocol, large language models, Spring AI, Java, MCP server, MCP client, resources, tools, prompts.

Постановка задачі. Великі мовні моделі (LLM) є потужними, але мають два основні обмеження: їхні знання заморожуються на момент навчання, і вони не можуть взаємодіяти із зовнішнім світом. Це означає, що вони не можуть отримати доступ до даних у реальному часі або виконувати такі дії, як бронювання зустрічі чи оновлення запису про клієнта.

Протокол контексту моделі (MCP) – це відкритий стандарт, розроблений для вирішення цієї проблеми. Введений компанією Anthropic у листопаді 2024 року, MCP забезпечує безпечну та стандартизовану «мову» для LLM для спілкування із зовнішніми даними, додатками та послугами. Він діє як міст, дозволяючи ШІ вийти за межі статичних знань і стати динамічним агентом, який може отримувати актуальну інформацію та вживати заходів, роблячи його більш точним, корисним та автоматизованим. [1]

У зв'язку з цим, метою статті є:

- з'ясувати концептуальні засади MCP: його архітектуру, ключові компоненти (MCP-сервер, MCP-клієнт, протокол транспорту, безпека, модель звернень і відповідей) та його відмінності від інших підходів – наприклад, від Retrieval-Augmented Generation (RAG).

© Петренко С. В.

- оцінити, як інтегрувати існуючі рішення з MCP: як приклади серверів, інструментів чи сервісів, що вже підтримують MCP, а також як підключати власні джерела даних чи бізнес-логіку до MCP-екосистеми [1].
- продемонструвати практичну реалізацію MCP-сервера на Java: вибір технологій, структура проєкту, приклад роботи (наприклад, підключення бази даних або стороннього API через MCP-сервер, обробка запитів, формат відповідей тощо).

Таким чином, завдання статті полягає у комплексному висвітленні концептуальних засад Model Context Protocol, зокрема в окресленні теоретичних основ протоколу та визначенні ключових функцій, які виконують MCP-сервер і MCP-клієнт. Важливо також проаналізувати переваги та обмеження використання MCP у порівнянні з іншими підходами інтеграції зовнішніх даних і сервісів. Окрему увагу приділено розробці та демонстрації прототипу MCP-сервера на Java, що надає змогу показати особливості реалізації ключових частин протоколу й оцінити практичність цього рішення з точки зору продуктивності, безпеки та зручності інтеграції.

Аналіз останніх досліджень і публікацій. Модель Context Protocol (MCP) стрімко формується як де-факто стандарт взаємодії LLM-додатків з зовнішніми джерелами даних і інструментами. Базові дефініції, архітектурні ролі «клієнт/сервер», а також концепти ресурсів, підказок (prompts) і тулів системно викладені в офіційній документації MCP, що позиціонує протокол як «USB-C для AI-застосунків», тобто уніфікований порт підключення до файлових систем, баз даних, сервісів тощо. Саме тут фіксуються ключові ідеї стандартизованої інтеграції та приклади застосування у робочих середовищах на кшталт IDE й чат-інтерфейсів [6]. Окремі дослідники (Sutter, 2025) [7] розглядають MCP як потенційно «відсутній стандарт» в інфраструктурі AI, підкреслюючи його роль у формуванні єдиної точки інтеграції.

Авторитетним джерелом для спільноти є специфікація протоколу (останнє видання від 18 червня 2025 р.), яка формалізує вимоги до сумісності, повідомлень і життєвих циклів взаємодії. Важливо, що MCP використовує датовані версійні ідентифікатори формату YYYY-MM-DD з принципом зворотної сумісності для інкрементальних оновлень – це забезпечує стабільність екосистеми й прогнозованість для розробників серверів/клієнтів [6].

Серед зарубіжних оглядів показовою є праця P. P. Ray (2025), у якій MCP описано як сесієорієнтований JSON-RPC каркас для інтероперабельної взаємодії LLM-клієнтів і зовнішніх ресурсів/інструментів; автор систематизує архітектурні патерни (resources, tools, prompts), окреслює стан галузі та фіксує ключові виклики – від безпеки й керування доступом до тестованості та сумісності реалізацій [9].

Дослідження на стику інженерії процесів і інтеграційних платформ пропонують розглядати MCP-сервери як нову парадигму автоматизації робочих потоків: у порівнянні з «legacy»-системами (на кшталт IFTTT/Zapier) підкреслюється перевага контекстуалізованого виклику інструментів і єдиного контракту взаємодії, що знижує вартість інтеграцій і спрощує розширення екосистеми [11]. Як підкреслює Pajo (2025), MCP-сервери можна розглядати як нову парадигму автоматизації робочих потоків, що перевищує за гнучкістю традиційні інтеграційні системи [10].

Водночас емпіричні роботи першої хвилі проблематизують сталість і безпеку MCP-екосистеми: за результатами великомасштабного аналізу 1 899 відкритих MCP-серверів (Hasan та ін., 2025) виявлено вісім класів вразливостей (лише три перетинаються з традиційними), 7,2 % проєктів містять загальні вразливості, а 5,5 % – випадки «tool poisoning»; при цьому 66 % мають «code smells», а 14,4 % – типові шаблони дефектів, що зумовлює потребу у спеціалізованих методах детекції загроз та інженерних практиках підтримуваності [8]. Подібні виклики відзначають і Hou та ін. (2025), які систематизували загрози безпеки MCP та окреслили напрями подальших досліджень [12].

Метою дослідження є обґрунтування концептуальних засад Model Context Protocol як сучасного стандарту інтеграції інтелектуальних систем та розробка підходів до його практичної реалізації у середовищі Java. У межах дослідження передбачено теоретичний аналіз архітектури MCP та функціональних ролей його основних компонентів (MCP-сервер і MCP-клієнт), розгляд функціоналу для розширення можливостей LLM, а також створення прототипу MCP-сервера на Java

© Петренко С. В.

з метою оцінки його практичної придатності для завдань інженерії програмного забезпечення, включаючи продуктивність та інтеграційну гнучкість.

Основна частина. Протокол Model Context Protocol (MCP) побудований за схемою «клієнт-хост-сервер», де один хост може запускати кілька інстансів клієнта. Такий підхід уможливлює вбудовування AI-функцій у різні застосунки, водночас зберігаючи чіткі межі безпеки та ізоляцію відповідальностей. Спираючись на JSON-RPC, MCP запроваджує сеансовий протокол із збереженням стану, зосереджений на обміні контекстом і координації операцій вибірки між клієнтами та серверами [6].

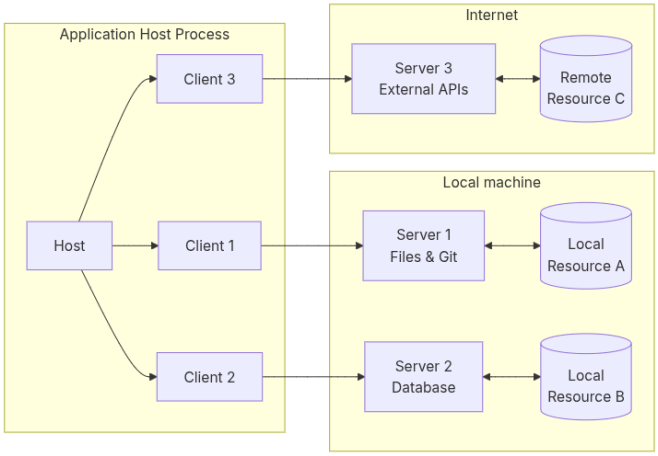

Рис. 1. Архітектура MCP

В специфікації MCP розробники виділяють три основні гравці: хост (host), клієнт (client) та сервер (server).

Процес «хоста» виступає контейнером і координатором. Він створює та керує кількома екземплярами клієнтів, регулює дозволи на підключення й життєвий цикл цих підключень, реалізує політики безпеки та механізми отримання згоди користувача, ухвалює рішення щодо авторизації. На рівні інтеграції з AI/LLM саме хост координує запити на семплювання (sampling) і забезпечує агрегування контексту між клієнтами, зберігаючи чітке розмежування відповідальностей.

Кожен клієнт створюється хостом і підтримує ізольоване з'єднання з конкретним сервером: встановлюється один сеанс із збереженням стану на один сервер (співвідношення 1:1). Клієнт узгоджує версії та можливості протоколу, маршрутизує повідомлення в обох напрямах, керує підписками й сповіщеннями, а також дотримується визначених меж безпеки між різними серверами. На прикладному рівні це означає, що одна хост-програма може керувати множиною клієнтів, кожен з яких працює із «своїм» сервером без взаємного впливу.

Сервери надають спеціалізований контекст і можливості: через примітиви MCP вони експонують ресурси, інструменти та промпти, виконуючи вузько сфокусовані завдання автономно. За потреби сервер ініціює запити на семплювання через інтерфейси клієнта, але зобов'язаний дотримуватися встановлених обмежень доступу та безпеки. Реалізації серверів можуть бути як локальними процесами, так і віддаленими сервісами. Така трикомпонентна організація забезпечує масштабованість, інтероперабельність і контроль доступу, необхідні для надійного включення AI-функцій у різні програмні середовища.

У Model Context Protocol вирізняють кілька взаємопов'язаних складових, що забезпечують цілісну роботу системи. Базою є протокол обміну повідомленнями, який формалізує типи JSON-RPC-повідомлень і їхню семантику. Поверх нього діє підсистема керування життєвим циклом: ініціалізація з'єднання, узгодження можливостей (capability negotiation) та контроль сеансу з підтриманням стану. Питання доступу розв'язуються в шарі авторизації, який охоплює аутентифікацію та авторизаційні механізми для HTTP-орієнтованих транспортів.

Функційність на стороні сервера представлена примітивами MCP – ресурсами (resources), підказками (prompts) та інструментами (tools), що експонують спеціалізовані можливості. На стороні клієнта зосереджені операції семплювання (sampling) і надання переліків кореневих каталогів (root directory lists), необхідних для контекстуалізації викликів. Поперечні сервісні можливості (utilities) – журналювання, автодоповнення аргументів, а також засоби діагностики й

© Петренко С. В.

трасування – забезпечують спостережуваність, відтворюваність і зручність розробки в реальних сценаріях інтеграції.

Наразі протокол підтримує два стандартні способи обміну JSON-RPC-повідомленнями: stdio (сервер як підпроцес) і Streamable HTTP, що замінив попередній HTTP+SSE. Streamable HTTP передбачає єдиний endpoint: POST для надсилання запитів (із можливістю потокових відповідей через SSE) та опційний GET-потік для ініціативних повідомлень сервера; обов'язкові вимоги включають перевірку Origin, локальне зв'язування та автентифікацію. Також визначено відновлення потоків (SSE id/Last-Event-ID), керування сеансами через заголовок Mcp-Session-Id, узгодження версії через MCP-Protocol-Version, а для зворотної сумісності - інструкції взаємодії зі старим HTTP+SSE. MCP лишається транспорт-агностичним і допускає кастомні транспорти за умови дотримання формату JSON-RPC і вимог життєвого циклу. [6]

Клієнт відіграє роль довіреного посередника між користувачем, LLM і MCP-серверами. Саме клієнт контролює межі доступу, веде сеанси, узгоджує можливості та надає серверу керовані «вікна» до контексту й моделей. Нижче подано стислу витримку основних клієнтських інтерфейсів, на яких тримається практична інтеграція.

Roots. Клієнт може експонувати «корені» файлової системи (workspace/project директори) і повідомляти сервер про їх зміни. Підтримка декларується через capability roots; сервер отримує перелік через roots/list, а у випадку змін клієнт надсилає notifications/roots/list_changed. Root має file://-URI й, за потреби, відображувану назву; зі сторони безпеки контролює межі доступу та здійснює валідацію шляхів.

Sampling. Сервер запитує у клієнта виконання LLM-семплювання (текст/зображення/аудіо) без необхідності ключів на стороні сервера - клієнт контролює вибір і доступ до моделей. Виклик здійснюється через sampling/createMessage, підтримуються modelPreferences (пріоритети cost/speed/intelligence та «hints» для підказки сімейств моделей). Рекомендовано «людину в циклі» для перегляду промптів і результатів.

Elicitation. Новий у ревізії 2025-06-18 механізм, що дає змогу серверу просити у користувача додаткові дані через клієнта. Запит надсилається як elicitation/create з обмеженою JSON-схемою (плоский об'єкт із примітивними полями), а відповідь має одна з дій: accept/decline/cancel. Заборонено вимагати чутливі дані; клієнт має чітко показувати, який сервер ініціює запит, і давати можливість відмови.

Оскільки клієнт у MCP забезпечує межі доступу та керує сеансами, сервер є стороною, котра «постачає» власне контекст і дієві можливості для LLM. У ревізії 2025-06-18 серверні примітиви чітко класифіковано: prompts (шаблони взаємодії з моделлю), resources (структуровані дані/вміст для контексту) та tools (викликані моделлю функції для дій і отримання інформації). Взаємодія клієнта, сервера і моделі організована так, щоб користувач контролював виклики промптів, застосунок - подачу ресурсів, а модель - звернення до інструментів, що разом формує керовані й відтворювані робочі сценарії.

Prompts - заздалегідь підготовлені шаблони повідомлень/інструкцій; користувач свідомо їх обирає (user-controlled). Для підтримки промптів сервер декларує capability prompts і, за потреби, надсилає сповіщення про зміну переліку (notifications/prompts/list_changed). Доступні виклики prompts/list і prompts/get.

Resources - ідентифіковані URI дані (файли, схеми БД, інша контекстна інформація), подача яких зазвичай керується застосунком (application-controlled). Підтримуються resources/list, resources/read, параметризовані шаблони (resources/templates/list) і, опційно, підписки/сповіщення про зміни; відповідні можливості вмикаються через resources.subscribe/resources.listChanged.

Tools - викликані моделлю (model-controlled) функції з чітким inputSchema та, за бажання, outputSchema для структурованих відповідей (structuredContent). Перелік інструментів і виклики здійснюються через tools/list та tools/call, зміни переліку сигналізуються notifications/tools/list_changed.

Промпти можуть містити мультимедійні повідомлення й вбудовані ресурси; інструменти можуть повертати посилання на ресурси або вбудовувати їх безпосередньо - це забезпечує прозоре «прошивання» контексту в діалог із моделлю. Для автодоповнення аргументів промптів і URI ресурсів сервер може увімкнути Utilities → Completion і відповідати на completion/complete, що дає IDE-подібні підказки в реальному часі. У всіх підсистемах специфікація вимагає валідації вхідних даних, контролю доступів, лімітів та уважної обробки помилок, аби мінімізувати ін'єкції, витоки і зловживання інструментами.

© Петренко С. В.

У реалізаційній частині ми зосереджуємося на Java та Spring AI, оскільки Spring AI уже має вбудовану підтримку MCP і спирається на офіційний MCP Java SDK, що істотно спрощує створення як сервера, так і клієнта. [java-sdk + spring] Варто підкреслити, що офіційні SDK існують для TypeScript і Python, а також підтримуються інші мови - зокрема Go, Kotlin, C#, Ruby, PHP. Це дає змогу обирати стек під конкретні вимоги проєкту, зберігаючи сумісність на рівні протоколу.

Для відтворюваних прикладів ми спиратимемося на довідники Spring AI і офіційні інструкції х «швидкого старту» MCP (сервер/клієнт), а також на репозиторій референсних серверів, що демонструють типові примітиви (resources, prompts, tools) і практики безпечного підключення [11].

У практичній частині ми наводимо реалізацію MCP на Java з використанням Spring AI, який інтегрує офіційний MCP Java SDK, фокусуючись лише на специфічних частинах імплементації протоколу. Обрана зв'язка надає усталені примітиви MCP (prompts, resources, tools), готові транспорти (STDIO), а також підтримку життєвого циклу. Ключова перевага підходу - Spring-конфігурація та анотації, які мінімізують шаблонний код і дають змогу задавати контракти MCP декларативно.

Увесь код проєкту розміщено у відкритому репозиторії github [2]. Для запуску локально потрібно встановити JDK верії 21, або вище та інструмент побудови Maven. Файлова структура відповідає типовому Spring-проєкту: src/main/java/... (серверні класи MCP, конфігурація бінів), src/main/resources/... (налаштування application.yml), pom.xml.

Пропоноване рішення наслідує три ролі MCP – host, client, server. У нашому сценарії Spring-застосунок підіймає сервер MCP, який експонує примітиви tools/resources/prompts, а клієнт (IDE/чат або MCP Inspector) ініціює сеанс, узгоджує можливості й викликає інструменти/зчитує ресурси. MCP працює у сеансовій моделі з підтриманням стану, що забезпечує повторюваність, контроль доступів і передбачуваність інтеграцій.

Серед базових стереотипів Spring, як то @Configuration, @Bean, @Component, @Service, варто відзначити @Tool, який є новим і дає змогу одним маркером оголосити інструмент без написання зайвого коду. Це нова анотація, яка з'явилась доволі недавно і значно спрощує кодову базу та швидкість розробки.

Конфігурація сервера знаходиться у файлі application.properties, а клієнта у application-client.properties, що уможливлює запуск проєкту в різних ролях через профілювання. Важливою умовою є вимкнення банера і консольного логування, щоб міг працювати транспорт STDIO.

У реалізації MCP клієнта та сервера зі Spring AI все максимально спрощено: достатньо додати у конфігураційний pom-файл лише одну залежність для сервера (spring-ai-starter-mcp-server) і одну для клієнта (spring-ai-starter-mcp-client). Це дає змогу швидко підняти інфраструктуру без складних налаштувань чи додаткових бібліотек.

Як зазначалось вище інструменти маркуються анотацією @Tool, проте, щоб всі вони були видимі при інтеграції нам необхідно оголосити бін, що повертає список ToolCallback для AI моделі. Клас ToolCallback має статичний метод from(Object... sources), який автоматично знаходить анотовані методи та робить їх доступними клієнту MCP. Таким чином, для інтеграції достатньо вказати лише один бін, що забезпечує повну видимість інструментів сервісу.

У конфігураційному класі PromptConfig визначено два ключові методи для створення prompts та sampling.

Перший - prompts(), який оголошений як Spring-бін і повертає список SyncPromptSpecification. Усередині цього методу створюється промпт із назвою "keyword" та описом пошукового запиту, що має обов'язковий аргумент - ключове слово. На основі введених користувачем даних формується повідомлення типу PromptMessage з роллю USER, у якому автоматично будується запит: "Find one book with '" + keyword + "' in the title". У результаті метод повертає колекцію з однією специфікацією промпту, що дає змогу клієнту динамічно передавати параметри та отримувати уніфікований пошуковий запит.

Другий метод - samplingTool(), який повертає ToolCallbackProvider. Усередині нього створюється інструмент sample_with_mcp, що реалізований через FunctionToolCallback. Цей інструмент приймає аргументи у вигляді запису SummarizeArgs (поля: текст і максимальна кількість токенів), формує об'єкт CreateMessageRequest з системним промптом "You are a concise assistant that summarizes text" та передає його до LLM через MCP-клієнт. Отриманий результат повертається у вигляді словника, де ключ "summary" містить коротке резюме тексту, а ключ "model" - інформацію про використану модель. У разі відсутності можливостей семплінгу на стороні клієнта метод повертає повідомлення про помилку.

© Петренко С. В.

Завдяки такому підходу реалізується єдина архітектура для створення промптів і інтеграції інструментів через Spring-біни. Це забезпечує узгодженість між визначенням специфікацій (prompts()) та підключенням інструментів (samplingTool()), а також підвищує зручність розширення системи при подальшій роботі з MCP.

У рамках реалізації визначено метод bookResources(BookService service), який оголошується як Spring-бін та повертає список специфікацій ресурсу McpServerFeatures.SyncResourceSpecification.

У методі формується ресурс allBooksResource з такими вхідними параметрами конструктора McpSchema.Resource:

- URI – "books://all": унікальний ідентифікатор ресурсу, за яким клієнт може звертатися до даних;
- Назва ресурсу – "All available books": зрозумілий опис сутності;
- Опис – "Complete list of famous service in CSV format": додаткова характеристика, що пояснює зміст ресурсу;
- MIME-тип – "text/csv": визначає формат представлення даних для клієнта;
- Іконка/зображення – null: параметр не використовується у даній реалізації.

Для створеного ресурсу формується специфікація allBooksSpec типу SyncResourceSpecification. Вона описує, як саме ресурс буде оброблятися при запиті. У даному випадку визначено обробник, який:

1. Отримує запит (request) і посилання на сесію обміну (exchange);
2. Викликає метод сервісу service.getAllBooksAsCsv(), який повертає рядок із повним переліком книг у форматі CSV;
3. Формує результат у вигляді об'єкта ReadResourceResult, що містить список об'єктів TextResourceContents. Кожен з них включає:

- URI ресурсу (ідентичний до вхідного request.uri())
- MIME-тип ("text/csv")
- Тіло ресурсу (CSV-рядок зі списком книг).

Метод повертає список специфікацій, що містить єдиний ресурс allBooksSpec. Таким чином, клієнт MCP, звертаючись за URI books://all, отримує повний список книг у форматі CSV. Це дає змогу абстрагувати джерело даних (сервіс BookService) від протоколу доступу (MCP) та забезпечує уніфіковану інтеграцію.

Далі необхідно виконати збірку проєкту за допомогою команди Maven, яка сформує виконуваний артефакт у вигляді jar-файлу. Варто зазначити, що збирання потрібно виконувати щоразу після внесення змін у код, аби усі зміни потрапили у новий артефакт і були доступні при роботі з додатком.

Важливим етапом є тестування та верифікація роботи додатку. Для цього в екосистемі Model Context Protocol передбачено спеціальний інструмент - MCP Inspector. Це універсальна утиліта з графічним інтерфейсом, яка дає змогу підключатися до MCP-сервера, переглядати доступні ресурси, інструменти й промпти та виконувати запити безпосередньо з робочого середовища.

MCP Inspector виступає своєрідним «діагностичним модулем»: він візуалізує описані на сервері можливості, відображає структуру оголошених інструментів (наприклад, get_book чи add_book) та ресурси (books://all), а також дає змогу вручну надсилати аргументи й отримувати результат у форматі, який повертає протокол. Це суттєво спрощує процес відлагодження, оскільки дослідник чи розробник одразу бачить, як саме інтерпретується запит клієнтом та яку відповідь формує сервер. Зокрема, можна протестувати, чи правильно формується список книг у CSV-форматі, чи доступний пошук книги за назвою, а також як обробляється додавання нового запису.

У подальшому використання MCP Inspector можна розглядати як інженерну практику, що забезпечує швидку ітераційну перевірку прототипів та зменшує ризики інтеграційних помилок на етапі розробки.

Для тестування реалізованого MCP-сервера необхідно попередньо встановити та сконфігурувати MCP Inspector. Для з'єднання з сервером потрібно налаштувати інспектор, а саме: обрати транспортний тип, додати команду запуску (у нашому випадку java), а також аргументи (шлях до побудованого артефакту) (рис. 2).

© Петренко С. В.

Рис. 2. Конфігурація MCP Inspector

Якщо всі параметри було внесено коректно, то має відбутися з'єднання з MCP сервером, який ми розробили. На сторінці інспектора присутнє меню для навігації та виконання запитів, секція логування з можливістю встановлення відповідного рівня системних повідомлень, історію викликів та нотифікації від сервера (рис. 3).

На рисунку (рис. 3) наведено інтерфейс MCP Inspector v0.15.0 під час підключення до реалізованого сервера. У лівій частині вікна розташована панель конфігурації, де можна обрати тип транспорту, задати команду запуску та аргументи (шлях до JAR-файлу сервера), а також визначити змінні середовища та рівень логування. Тут же відображається статус підключення (зелений індикатор Connected) та вивід повідомлень про помилки з боку сервера.

Центральна частина інтерфейсу демонструє доступні ресурси (наприклад, Actor by ID, All available books, All Famous Actors), а також шаблони ресурсів, які можуть бути використані для формування типових запитів. Кожен ресурс можна викликати окремо, що надає можливість перевірити його роботу безпосередньо з інспектора.

У правій частині вікна відображається результат запиту до ресурсу All available books. Дані представлені у форматі CSV із переліком книг та відповідними атрибутами (Title, Author, Year of pub). Це приклад інтеграції сервісу BookService, який формує дані у вигляді таблиці та повертає їх через протокол MCP.

У нижній частині вікна знаходиться секція History, де зберігається журнал виконаних викликів (наприклад, resources/read, tools/call). Це допомагає відслідковувати послідовність операцій і повторно запускати необхідні запити. Поряд розташований блок Server Notifications, у якому можуть з'являтися повідомлення від сервера.

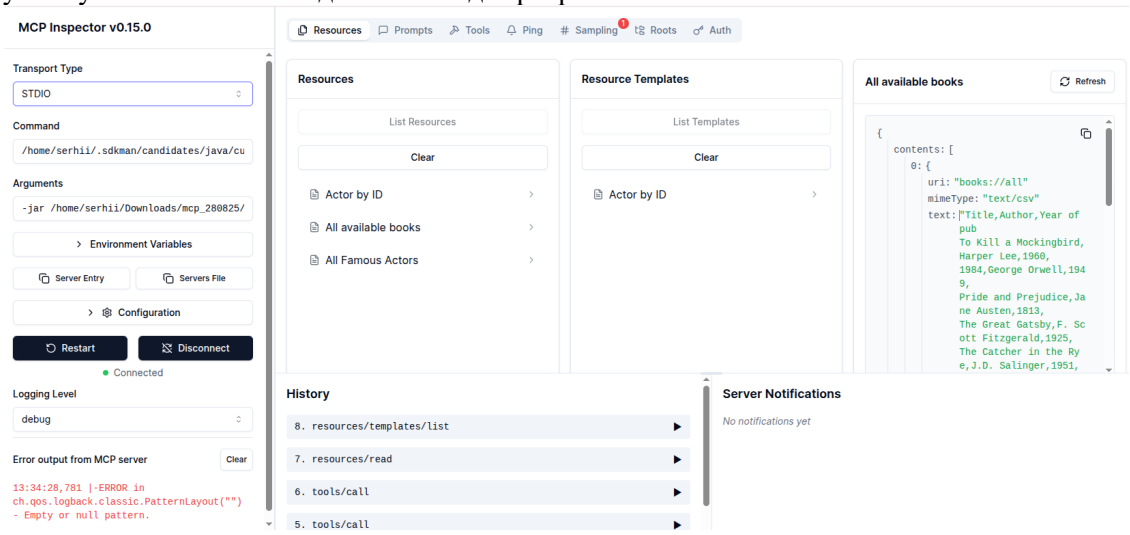

Рис. 3. Інтерфейс MCP Inspector v0.15.0

Висновки та перспективи подальшого дослідження. Проведене дослідження засвідчує, що Model Context Protocol є перспективним стандартом для інтеграції інтелектуальних систем із зовнішнім середовищем. Завдяки архітектурі «хост-клієнт-сервер» MCP забезпечує масштабованість, інтероперабельність та контроль доступу, необхідні для надійного впровадження AI-функцій у програмні продукти. Практична реалізація MCP-сервера на Java із застосуванням

© Петренко С. В.

Spring AI показала можливості швидкого розгортання, мінімізації шаблонного коду та забезпечення високої відтворюваності інтеграцій. Використання MCP Inspector підтвердило ефективність протоколу як діагностичного інструменту й засобу для швидкої перевірки прототипів.

Подальші дослідження доцільно спрямувати на:

- розробку методів підвищення безпеки MCP-сервісів, зокрема протидію «tool poisoning»;
- розширення підтримки мультимодальних інтерфейсів і транспортних протоколів;
- вивчення продуктивності MCP у масштабованих системах та хмарних середовищах;
- створення типових інженерних практик для тестування й підтримуваності серверів і клієнтів.

Таким чином, MCP формує нову парадигму автоматизованих робочих процесів і відкриває перспективи для розвитку гнучких та надійних інтелектуальних платформ.

Список бібліографічного опису

1. Anthropic. Introducing the Model Context Protocol. URL: https://www.anthropic.com/news/model-context-protocol (дата звернення: 16.10.2025)
2. GitHub репозиторій з вихідним кодом реалізації MCP. URL: https://github.com/serhii-petrenko/coffeejug_mcp (дата звернення: 16.10.2025)
3. GitHub репозиторій протоколу. URL: https://github.com/modelcontextprotocol (дата звернення: 16.10.2025)
4. MCP Inspector URL: https://modelcontextprotocol.io/legacy/tools/inspector (дата звернення: 16.10.2025)
5. MCP документація. URL: https://modelcontextprotocol.io/ (дата звернення: 16.10.2025)
6. MCP специфікація. URL: https://spec.modelcontextprotocol.io/ (дата звернення: 16.10.2025)
7. Michal Sutter Is Model Context Protocol MCP the Missing Standard in AI Infrastructure? URL: https://surl.lu/qpduyp (дата звернення: 16.10.2025)
8. Mohammed Mehedi Hasan, Hao Li, Emad Fallahzadeh, Gopi Krishnan Rajbahadur, Bram Adams, Ahmed E. Hassan Model Context Protocol (MCP) at First Glance: Studying the Security and Maintainability of MCP Servers URL: https://doi.org/10.48550/arXiv.2506.13538 (дата звернення: 16.10.2025)
9. Partha Pratim Ray. A Survey on Model Context Protocol: Architecture, State-of-the-art, Challenges and Future Directions. TechRxiv. April 18, 2025. URL: https://doi.org/10.36227/techrxiv.174495492.22752319/v1 (дата звернення: 16.10.2025)
10. Paul Pajo Model Context Protocol Servers: A Novel Paradigm for AI-Driven Workflow Automation and Comparative Analysis with Legacy Systems URL: http://dx.doi.org/10.13140/RG.2.2.15241.76640 (дата звернення: 16.10.2025)
11. Spring AI MCP overview. URL: https://docs.spring.io/spring-ai/reference/api/mcp/mcp-overview.html (дата звернення: 16.10.2025)
12. Xinyi Hou, Yanjie Zhao, Shenao Wang, Haoyu Wang Model Context Protocol (MCP): Landscape, Security Threats, and Future Research Directions. URL: https://arxiv.org/abs/2503.23278 (дата звернення: 16.10.2025)

References

1. Anthropic. Introducing the Model Context Protocol. URL: https://www.anthropic.com/news/model-context-protocol (date of access: 16.10.2025)
2. GitHub repository with the source code for the MCP implementation. URL: https://github.com/serhii-petrenko/coffeejug_mcp (date of access: 16.10.2025)
3. GitHub protocol repository. URL: https://github.com/modelcontextprotocol (date of access: 16.10.2025)
4. MCP Inspector URL: https://modelcontextprotocol.io/legacy/tools/inspector (date of access: 16.10.2025)
5. MCP documentation. URL: https://modelcontextprotocol.io/ (date of access: 16.10.2025)
6. MCP specification. URL: https://spec.modelcontextprotocol.io/ date of access: 16.10.2025)
7. Michal Sutter Is Model Context Protocol MCP the Missing Standard in AI Infrastructure? URL: https://surl.lu/qpduyp (date of access: 16.10.2025)
8. Mohammed Mehedi Hasan, Hao Li, Emad Fallahzadeh, Gopi Krishnan Rajbahadur, Bram Adams, Ahmed E. Hassan Model Context Protocol (MCP) at First Glance: Studying the Security and Maintainability of MCP Servers URL: https://doi.org/10.48550/arXiv.2506.13538 (date of access: 16.10.2025)
9. Partha Pratim Ray. A Survey on Model Context Protocol: Architecture, State-of-the-art, Challenges and Future Directions. TechRxiv. April 18, 2025. URL: https://doi.org/10.36227/techrxiv.174495492.22752319/v1 (date of access: 16.10.2025)
10. Paul Pajo Model Context Protocol Servers: A Novel Paradigm for AI-Driven Workflow Automation and Comparative Analysis with Legacy Systems URL: http://dx.doi.org/10.13140/RG.2.2.15241.76640 (date of access: 16.10.2025)
11. Spring AI MCP overview. URL: https://docs.spring.io/spring-ai/reference/api/mcp/mcp-overview.html (date of access: 16.10.2025)
12. Xinyi Hou, Yanjie Zhao, Shenao Wang, Haoyu Wang Model Context Protocol (MCP): Landscape, Security Threats, and Future Research Directions. URL: https://arxiv.org/abs/2503.23278 (date of access: 16.10.2025)

© Петренко С. В.