Technical Architecture and Redesign Strategy for a Universal Model Context Protocol Registry and Generative Design Backend

The proliferation of Large Language Models (LLMs) has transitioned from standalone chat interfaces to integrated agentic ecosystems where the capacity for external tool invocation is the primary differentiator. The Model Context Protocol (MCP), originally introduced by Anthropic and later contributed to the Agentic AI Foundation, serves as an open standard to bridge the gap between AI models and disparate data sources.[1] For an organization such as mcpmessenger, the objective is to move beyond localized, single-user implementations toward an enterprise-grade, horizontally scalable registry and backend infrastructure. This report provides a comprehensive blueprint for the construction of this backend, specifically tailored to support a generative logo design frontend like the one hosted on Vercel.[3] By integrating microservices orchestration, event-driven communication via Apache Kafka, and strict adherence to the MCP v0.1 registry specification, the proposed architecture establishes a resilient foundation for the next generation of AI-powered workflows.[3]

Evolution and Significance of the Model Context Protocol

To understand the necessity of a dedicated registry backend, one must first examine the historical context of AI integrations. Prior to the formalization of MCP, developers were forced to engineer custom, ad-hoc wrappers for every unique tool-agent combination, creating an $N \times M$ integration complexity that was fundamentally unscalable.[7] The Model Context Protocol collapses this complexity into a unified interface, acting as a "USB-C for AI" that allows a single host application to discover and utilize any compliant server.[7] This standardization is not merely a technical convenience but a prerequisite for the industrialization of AI agents, enabling them to transition from reactive bots to proactive decision-makers with a form of working memory.[12]

The protocol is built upon JSON-RPC 2.0, providing a lightweight and stateless mechanism for message exchange.[1] It distinguishes between three primary participants: the host, which orchestrates the experience (e.g., Claude Desktop or a custom web dashboard); the client, which manages individual connections; and the server, which provides specific capabilities such as database access, file manipulation, or image generation.[1]

participant	Primary Responsibility	Context within Logo Design Platform
MCP Host	coordinating model interactions and UI rendering	The Next.js dashboard and design canvas.[3]
MCP Client	connection management and protocol negotiation	The middleware layer connecting the UI to backend services.[14]
MCP Server	capability exposure (tools, resources, prompts)	Services providing SVG generation, color theory analysis, and font lookup.[1]

The interaction lifecycle begins with a rigorous initialization phase where the client and server negotiate protocol versions and capabilities.[16] This ensures that the host application can dynamically adjust its interface based on the available tools, such as enabling or disabling a "Vision Analysis" button if the corresponding MCP server lacks the required capabilities.[14]

Deep Dive into the McpMessenger and SlashMCP Ecosystem

The existing infrastructure managed by mcpmessenger, specifically the SlashMCP project, demonstrates a sophisticated use of the protocol for document intelligence and multi-agent orchestration.[18] SlashMCP serves as an AI workspace that integrates OCR via AWS Textract and vision analysis via GPT-4o, providing a template for how a logo design platform can process multimodal inputs.[18] The backend for such a system requires more than simple request-response cycles; it demands a robust job-tracking mechanism where processing tasks are polled or streamed via WebSockets or Server-Sent Events (SSE).[18]

A logo design frontend built with Vercel's v0 leverages "vibe coding," a philosophy where natural language descriptions are transformed into production-ready React components using Tailwind CSS and shadcn/ui.[20] To support this on the backend, the system must be able to:

1. Manage dynamic MCP server registries that can be invoked directly from the chat interface.[18]
2. Orchestrate multiple agents to handle specialized design tasks, such as typography selection and vector rendering.[3]
3. Handle large asset uploads by generating S3 presigned URLs, ensuring that the heavy

lifting of file movement is decoupled from the application logic.[18]

The current SlashMCP implementation utilizes Supabase Edge Functions to bridge the gap between the frontend and the AI models, a pattern that offers low cold-start times and built-in authentication via Supabase Auth.[23] However, for a larger registry, a transition to a more traditional microservices architecture is necessary to ensure horizontal scalability and maintainability.[3]

Microservices Redesign: A Scalable Backend Blueprint

The architectural redesign of the MCP registry and management platform is centered on a microservices-based, cloud-native approach.[3] This transformation addresses the technical debt associated with monolithic designs by decomposing the system into independent services that communicate via high-performance protocols such as gRPC and asynchronous event buses like Apache Kafka.[3]

Core Microservices and Language Selection

The redesign mandates a polyglot stack to maximize efficiency and performance.[3] Go, specifically with the Go-Zero framework, is the preferred language for high-throughput infrastructure components due to its superior concurrency model and strong support for gRPC.[5] Python, leveraging FastAPI, is utilized for services that require deep integration with AI libraries and LLM SDKs.[12]

Service	Technology Stack	Primary Functionality
API Gateway	Go / Go-Zero	Central entry point handling auth, rate limiting, and WebSocket/SSE management.[3]
Auth Service	Go	Management of user identity, roles, and fine-grained authorization.[3]
Registry Service	Go	Authority for MCP server metadata, health tracking, and discovery.[3]
Routing Service	Python / FastAPI	Intent detection and intelligent orchestration of

		MCP agents.[3]
Multimodal Service	Python / FastAPI	Vision, OCR, and document ingestion pipelines (STT/TTS).[3]
Monitoring Service	Go	Persistent health checks and telemetry collection across the cluster.[3]

The use of a service mesh like Istio is recommended to manage the inter-service communication, providing built-in observability through Prometheus and Jaeger, and ensuring that failures in one service do not cascade throughout the system.[3]

Event-Driven Core with Apache Kafka

The "central nervous system" of the redesigned backend is Apache Kafka, which facilitates resilient, asynchronous communication between microservices.[3] Traditional synchronous API calls are ill-suited for generative AI workflows, where processing a vision request or generating a complex SVG logo can take several seconds.[32] By utilizing an event-driven architecture (EDA), the system can immediately acknowledge a user's request and update the UI as results become available.[19]

For example, when a user uploads a reference image for a logo design, the process follows an asynchronous event flow:

1. The API Gateway receives the upload and publishes a USER_QUERY_RECEIVED event to a Kafka topic.[3]
2. The Multimodal Service consumes this event, triggers the vision analysis worker, and publishes ANALYSIS_READY when the description of the image is complete.[3]
3. The Routing Service consumes the analysis and determines the next step, such as invoking a specific SVG-generation tool, and publishes a TOOL_INVOCATION_REQUESTED event.[3]
4. The generative MCP server executes the task and publishes the result, which is eventually pushed back to the frontend via WebSockets.[3]

This non-blocking behavior is essential for maintaining a responsive user experience in complex AI workflows.[19] The use of Kafka also ensures durability; if a service is temporarily offline, it can consume the missed events once it restarts, preventing data loss.[34]

Specification v0.1: Engineering the MCP Registry Service

The heart of the backend is the Registry Service, which must act as a canonical "app store" for MCP servers.[5] To be compatible with major hosts like VS Code and Claude Desktop, the registry must strictly follow the v0.1 MCP registry specification.[6] This specification defines standardized HTTPS endpoints that return server details and versioning information.[6]

Registry Architecture and Endpoints

A valid registry must support URL routing and provide JSON-structured metadata about each server.[6] The primary endpoints required for a v0.1 compliant registry are summarized below:

Endpoint	Method	Purpose
/v0/servers	GET	List all available servers with support for pagination and search.[6]
/v0/servers/{id}	GET	Fetch detailed metadata for a specific server by its unique identifier.[36]
/v0/publish	POST	Authenticated endpoint for maintainers to list new servers in the registry.[36]
/v0/health	GET	Status endpoint to verify the health and connectivity of the registry itself.[39]

The metadata stored in the registry is more than just a list of URLs. It must include the server's identity (a unique namespace like io.github.username/server-name), execution instructions (command-line arguments, environment variables), and metadata describing its capabilities (descriptions of tools and prompts).[7] This structured information allows AI models to understand not just where a tool is, but how to use it effectively.[41]

Database Schema and Data Modeling with Prisma

For the underlying data store, PostgreSQL is the standard choice for relational metadata, providing the necessary ACID compliance for registry operations.[40] The use of Prisma as an ORM enables declarative data modeling and type-safe query generation, which is crucial for maintaining a rapidly evolving schema.[42]

A robust registry schema must track:

- **Applications and Environments:** Different deployment stages (production, staging, development) for the same server.[40]
- **API Keys and Authentication:** Hashed credentials and OAuth tokens for managing access to private servers.[40]
- **Health Logs:** A history of automated health checks to ensure that the registry only serves endpoints that are currently operational.[3]
- **Tool and Resource Definitions:** Detailed JSON schemas for every tool exposed by a server, allowing the backend to perform pre-validation on requests.[14]

The registry also embraces federation, meaning the official MCP registry is not the only source of truth.[41] Organizations can build private sub-registries for internal use, while still conforming to the same API specification to ensure that the internal tooling remains compatible with external clients.[38]

Intelligent Routing and Intent Detection in Design Workflows

The Routing Service is arguably the most complex component of the backend, acting as the "brain" that connects natural language prompts to specific tool executions.[3] In the context of logo design, if a user asks to "Make the logo more aggressive," the Routing Service must determine whether this requires adjusting font weight, color saturation, or the geometry of the vector path.[12]

Intent Detection and Orchestration

Using Python's FastAPI and integration with LLMs, the Routing Service performs intent detection to map a user's prompt to the most appropriate MCP server in the registry.[3] This orchestration involves:

1. **Capability Aggregation:** Discovered tools from multiple backend servers are presented as a single interface to the host application.[48]
2. **Request Routing:** The service directs tool calls to the correct backend server based on the identified intent.[3]
3. **Context Management:** Maintaining a persistent session where the results of one tool call (e.g., "Analyze brand colors") can be used as input for the next call (e.g., "Generate logo variants").[12]

This "agentic" memory is facilitated by the protocol's sampling capability, where a server can ask the host to run LLM completions on its behalf.[1] This allows the Routing Service to stay model-independent, asking the primary LLM to refine its plan without needing to hardcode logic for every possible user request.[14]

Vector Retrieval and RAG Architecture

To provide fact-based, contextual answers—such as ensuring a new logo matches a client's historical branding—the backend incorporates a Retrieval-Augmented Generation (RAG) pipeline.[51] This involves vectorizing brand documentation and design assets and storing them in a specialized database like Qdrant or Pinecone.[51]

Feature	Qdrant	Pinecone
Deployment	Open-source, local, cloud, or hybrid.[51]	Fully managed, SaaS-only.[51]
Scaling	Resource-based pricing, performance tuning.[55]	Serverless, automatic scaling.[51]
Customization	Deep customization, binary quantization.[51]	Standardized configuration, minimal overhead.[54]
Compliance	Strong support for on-prem/regulated data.[51]	Enterprise certifications (SOC 2, GDPR).[51]

For a company like mcpmessenger, Qdrant is often the superior choice due to its open-source nature and flexible deployment options, which allow sensitive client design data to be kept within a secure execution perimeter.[35] The architecture ensures that raw data never leaves its origin; instead, embeddings are used to find relevant context, which is then provided to the LLM as part of the prompt.[35]

Multimodal Gateway: Processing Vision, Voice, and Documents

The Multimodal Service is the interface through which the system consumes non-textual data.[3] In a generative logo design application, the ability to process visual context is paramount. This service utilizes a worker-based architecture to handle the computational intensity of vision models.[18]

Vision Analysis and OCR Pipelines

When a user provides a design mockup or a wireframe, the Multimodal Service executes the following pipeline:

1. **Asset Ingestion:** The frontend calls a Supabase edge function to create a processing_job and receives a presigned URL to upload the asset directly to S3.[18]

2. **Vision Analysis Worker:** A worker fetches the asset and submits it to GPT-4o with a structured prompt designed to extract visual themes, color palettes, and stylistic choices.[18]
3. **OCR Worker:** For document-heavy uploads or hand-drawn sketches with annotations, AWS Textract is used to extract text and raw structural data.[18]
4. **Result Persistence:** The outputs (OCR text and vision summaries) are stored in an analysis_results table, uniquely indexed to allow for idempotent updates via the job_id.[18]

This structured approach prevents "code-hallucinations" by grounding the AI assistant in the actual contents of the design files.[22] Furthermore, the system includes future scaffolding for Whisper-based speech-to-text (STT) and Google Text-to-Speech (TTS) pipelines, enabling users to "talk through" their design revisions.[18]

Generative AI Tool Integration

The final output of the multimodal pipeline is the invocation of a generative tool. For logo design, this might involve the DALL-E MCP server or a local Stable Diffusion server.[21] These tools must be treated as untrusted and encapsulated behind a proxy that enforces strict authorization policies.[1] The generated images are processed, optimized, and then served back to the frontend, where the user can refine them using the dashboard's design mode.[4]

Security Architecture: Defending the Agentic Perimeter

Building a backend for AI agents introduces a new class of vulnerabilities. Unlike traditional software, where logic is fixed, AI agents decide what to do at runtime, making them susceptible to prompt injection and malicious tool calls.[60] A robust security model must be "Defense in Depth," incorporating network isolation, rigorous authentication, and input sanitization.[30]

Authentication, Authorization, and OAuth 2.1

The protocol mandates OAuth 2.1 as the non-negotiable baseline for any server exposed to the internet.[2] The backend must act as an OAuth proxy, managing the exchange of tokens between the host application and the MCP servers.[44]

Key security requirements for the registry and proxy layer:

- **Per-Client Consent:** The system must maintain a registry of approved client IDs per user and check this registry before initiating authorization flows.[44]
- **Scope Enforcement:** Users must be presented with the specific API scopes being requested (e.g., "read only" or "full write access") and must explicitly consent to each.[1]
- **State and Redirect Validation:** To prevent session hijacking and token exfiltration, the

system must generate cryptographically secure random state values and validate that the redirect_uri matches the registered value exactly.[44]
- **Token Passthrough Prohibition:** MCP servers must not accept tokens that were not explicitly issued for that server, preventing attackers from circumventing security controls like rate limiting or traffic monitoring.[44]

Hardening the Backend Infrastructure

Beyond the protocol level, the underlying infrastructure must be secured.[29] The service accounts running the microservices should be low-privilege users, and the system should utilize network segmentation (DMZs) to isolate sensitive components.[29] All communication must happen over encrypted channels (HTTPS/TLS), and credentials should be managed via environment variables or secret managers, never hardcoded in the source code.[29]

For data security, particularly when dealing with sensitive branding information, the "Golden Rule" is to treat all input as hostile.[29] This means:

- **Strict Schema Validation:** Every parameter coming from an LLM must be rigorously validated against its JSON schema before being passed to a tool.[29]
- **Sanitization:** Input must be sanitized to prevent SQL injection or other execution attacks, particularly when tools interact with databases or the filesystem.[29]
- **Output Redaction:** Logs and monitoring data should be scrubbed of personally identifiable information (PII) and secrets.[29]

Observability, Monitoring, and Health Checks

A production-grade MCP registry requires constant monitoring to maintain trust and reliability.[3] The Monitoring Service performs active health checks on all registered MCP servers, ensuring that the registry only routes requests to operational endpoints.[3]

Distributed Tracing and Correlation IDs

In a microservices architecture communicating via Kafka, understanding the full lifecycle of a request is impossible without distributed tracing.[19] The system must assign a unique correlation ID to every initial user query and propagate it through every event and service call.[19] Tools like Jaeger and Prometheus are used to visualize the "hops" a request takes—from the API Gateway to the Routing Service, through the Kafka event bus, to the MCP server, and back—capturing the latency and success rate of each step.[19]

Health Metrics and Alerting

The Monitoring Service should track a variety of metrics to ensure system health:

- **Availability and Latency:** The response time of individual MCP servers and the registry

itself.[39]

- **Authentication Failures:** Logging all failed attempts to connect or authorize, which can be an early indicator of a security breach.[29]
- **Queue Depth:** Monitoring Kafka topics to ensure that the asynchronous workers are keeping up with the volume of design requests.[34]
- **Resource Usage:** Tracking CPU and memory consumption of the multimodal and generation workers to guide auto-scaling decisions.[62]

Regular reviews of the registry are also necessary to prune unused servers, update documentation, and verify trust levels (e.g., re-verifying that a server is still "IT-verified").[57]

Implementation Best Practices and Technical Takeaways

For the engineering team at mcpmessenger, building out the backend is a phased process that begins with project environment setup and progresses through server initialization and tool implementation.[15]

Project Setup and Configuration

A consistent development environment is the first step toward a stable server. Using Docker or VS Code Dev Containers ensures that every developer is working with the same dependencies and runtime versions, preventing "version drift" and inconsistent validation.[63]

Development Phase	Recommended Practice
Environment Config	Use .env files and secret managers from day one.[60]
Core Message Loop	Harden the WebSocket/HTTP loop and JSON-RPC parsing before adding real tool logic.[63]
Tool Definition	Use standardized naming (snake_case) to ensure LLM tokenization is optimal.[7]
Logging	Write all logs to stderr or dedicated files to avoid corrupting stdio transports.[15]

When defining tools for the logo design platform, developers should follow the Single Responsibility Principle: each MCP server should have one clear, well-defined purpose.[30]

Instead of a monolithic "Mega-Server" for all design tasks, the system should feature focused services for typography, color, and vector generation.[30] This modularity simplifies testing, enhances reliability, and allows different teams to own different parts of the capability landscape.[30]

Testing with MCP Inspector

Verification is a continuous part of the development cycle. The MCP Inspector is a visual testing tool that allows developers to manually invoke tools, inspect resources, and view log output without needing to connect a full host application.[45] This facilitates "fail-fast" development, where schema errors or connection issues are identified long before they reach production.[32]

Performance Optimization

To ensure the backend responds quickly, especially during the interactive design phase, several optimization techniques should be employed:

- **Caching:** Frequently accessed data, such as brand guidelines or common font metadata, should be cached in Redis.[12]
- **Connection Pooling:** For services that interact with the PostgreSQL database or external APIs, connection pooling reduces the overhead of establishing new connections.[64]
- **Rate Limiting:** Implementing rate limits prevents individual users or malicious actors from overwhelming the generative workers.[30]

Conclusions and the Future of Universal Context

The backend for the mcpmessenger registry and logo design platform represents a shift from static software to a dynamic, agentic operating layer.[9] By building a microservices-based architecture that leverages gRPC for performance and Kafka for resilience, the organization can support high-throughput design workflows that are natively "agentic".[3]

The integration of the v0.1 MCP registry specification ensures that the platform remains compatible with the broader AI ecosystem, allowing it to function as a central hub for design-oriented context and tools.[5] As the protocol evolves to support richer interactions like long-running tasks and complex multi-step workflows, this redesigned backend will provide the necessary infrastructure to scale with the technology.[17]

In final analysis, the success of the platform depends on three pillars:

1. **Standardization:** Adhering to open protocols like MCP and OAuth 2.1 to ensure interoperability and security.[1]
2. **Scalability:** Using cloud-native microservices and event-driven patterns to handle the computational load of generative AI.[3]

3. **Governance:** Establishing a robust registry that serves as the single source of truth for approved, high-quality design tools.[6]

By synthesizing these elements into a cohesive technical strategy, the team can deliver a sophisticated backend that not only supports the existing frontend but also defines the state-of-the-art for agentic design platforms.

Works cited

1. Specification - Model Context Protocol, accessed December 18, 2025, https://modelcontextprotocol.io/specification/2025-06-18
2. 15 Best Practices for Building MCP Servers in Production - The New Stack, accessed December 18, 2025, https://thenewstack.io/15-best-practices-for-building-mcp-servers-in-production/
3. Architectural Redesign Report_ MCP Registry and Management Platform.md
4. Vercel v0 - AI-Powered UI Generator - Refine dev, accessed December 18, 2025, https://refine.dev/blog/vercel-v0/
5. A community driven registry service for Model Context Protocol (MCP) servers. - GitHub, accessed December 18, 2025, https://github.com/modelcontextprotocol/registry
6. Configure an MCP registry for your organization or enterprise - GitHub Docs, accessed December 18, 2025, https://docs.github.com/en/copilot/how-tos/administer-copilot/manage-mcp-usage/configure-mcp-registry
7. Use MCP servers in VS Code, accessed December 18, 2025, https://code.visualstudio.com/docs/copilot/customization/mcp-servers
8. Creating an MCP Server Using TypeScript - Cloudfronts, accessed December 18, 2025, https://www.cloudfronts.com/blog/creating-an-mcp-server-using-typescript/
9. MCP: 5 Things You Need to Know About the Model Context Protocol, accessed December 18, 2025, https://www.youtube.com/watch?v=I4c9I9CD3zM
10. Creating Your First MCP Server: A Hello World Guide | by Gianpiero Andrenacci | AI Bistrot | Dec, 2025, accessed December 18, 2025, https://medium.com/data-bistrot/creating-your-first-mcp-server-a-hello-world-guide-96ac93db363e
11. Unlocking AI Agents: A Deep Dive into the OpenAPI MCP Server by Ross Masters, accessed December 18, 2025, https://skywork.ai/skypage/en/unlocking-ai-agents-openapi-mcp-server/1977964570230050816
12. Integrating MCP Servers with FastAPI | by Dilip Bajaj - Medium, accessed December 18, 2025, https://medium.com/@bajajdilip48/integrating-mcp-servers-with-fastapi-00534a38b849
13. v0 by Vercel, accessed December 18, 2025, https://v0.app/

14. Architecture overview - Model Context Protocol, accessed December 18, 2025, https://modelcontextprotocol.io/docs/learn/architecture
15. Build an MCP server - Model Context Protocol, accessed December 18, 2025, https://modelcontextprotocol.io/docs/develop/build-server
16. model-context-protocol-resources/guides/mcp-server-development-guide.md at main - GitHub, accessed December 18, 2025, https://github.com/cyanheads/model-context-protocol-resources/blob/main/guides/mcp-server-development-guide.md
17. The official TypeScript SDK for Model Context Protocol servers and clients - GitHub, accessed December 18, 2025, https://github.com/modelcontextprotocol/typescript-sdk
18. mcpmessenger/slashmcp - GitHub, accessed December 18, 2025, https://github.com/mcpmessenger/slashmcp
19. MCP Event-Driven Architecture: Design & Implementation 2025 - BytePlus, accessed December 18, 2025, https://www.byteplus.com/en/topic/541257
20. Vercel v0: My Hands-On Guide to the AI Web Builder Changing the Game, accessed December 18, 2025, https://skywork.ai/skypage/en/Vercel-v0-My-Hands-On-Guide-to-the-AI-Web-Builder-Changing-the-Game/1974874791120400384
21. v0-mcp MCP Server Deep Dive: The Definitive Guide for AI Engineers - Skywork.ai, accessed December 18, 2025, https://skywork.ai/skypage/en/v0-mcp-MCP-Server-Deep-Dive-The-Definitive-Guide-for-AI-Engineers/1972503517736255488
22. Building Faster with V0 and Claude Code: Lessons Learned from Vibe Coding - Strapi, accessed December 18, 2025, https://strapi.io/blog/building-faster-with-v0-and-claude-code-lessons-learned-from-vibe-coding
23. Building ChatGPT Apps with Supabase Edge Functions and mcp-use, accessed December 18, 2025, https://supabase.com/blog/building-chatgpt-apps-with-supabase
24. Announcing the Supabase Remote MCP Server, accessed December 18, 2025, https://supabase.com/blog/remote-mcp-server
25. MCP | go-zero Documentation, accessed December 18, 2025, https://go-zero.dev/en/docs/tutorials/mcp/servers
26. From gRPC to MCP: Turning gRPC Services into AI-Native Tools - Dipjyoti Metia - Medium, accessed December 18, 2025, https://dipjyotimetia.medium.com/from-grpc-to-mcp-turning-grpc-services-into-ai-native-tools-9cca2d6a7f35
27. FastAPI with MCP: Build Enterprise AI Agents for API-Driven Apps | MintMCP Blog, accessed December 18, 2025, https://www.mintmcp.com/blog/build-enterprise-ai-agents
28. Building an MCP Server with FastAPI and FastMCP - Speakeasy, accessed December 18, 2025, https://www.speakeasy.com/mcp/framework-guides/building-fastapi-server
29. 4 Best Strategies to Secure Model Context Protocol - Knostic, accessed

December 18, 2025, https://www.knostic.ai/blog/strategies-secure-model-context-protocol
30. MCP Best Practices: Architecture & Implementation Guide, accessed December 18, 2025, https://modelcontextprotocol.info/docs/best-practices/
31. Kafka Event-Driven Architecture Done Right + Real Examples - Estuary, accessed December 18, 2025, https://estuary.dev/blog/kafka-event-driven-architecture/
32. Build Your Own Model Context Protocol Server | by C. L. Beard | BrainScriblr | Nov, 2025, accessed December 18, 2025, https://medium.com/brainscriblr/build-your-own-model-context-protocol-server-0207625472d0
33. Build an MCP client - Model Context Protocol, accessed December 18, 2025, https://modelcontextprotocol.io/docs/develop/build-client
34. Event Streaming with Kafka and FastAPI | AI-Driven Web, Mobile Apps for SMEs, Startups, accessed December 18, 2025, https://agilecyber.com/event-streaming-with-kafka-and-fastapi/
35. Secure Agentic RAG Architecture with MCP, Kafka, and Wayang Execution - Scalytics, accessed December 18, 2025, https://www.scalytics.io/blog/secure-scalable-mcp-kafka-architecture
36. MCP Registry - Model Context Protocol（MCP）, accessed December 18, 2025, https://modelcontextprotocol.info/tools/registry/
37. Getting Started With the Official MCP Registry API - Nordic APIs, accessed December 18, 2025, https://nordicapis.com/getting-started-with-the-official-mcp-registry-api/
38. How to find, install, and manage MCP servers with the GitHub MCP Registry, accessed December 18, 2025, https://github.blog/ai-and-ml/generative-ai/how-to-find-install-and-manage-mcp-servers-with-the-github-mcp-registry/
39. PostgreSQL Database Setup with Prisma - figma-mcp-server - GitHub, accessed December 18, 2025, https://github.com/haseebrj17/figma-mcp-server/blob/main/docs/DATABASE_SETUP.md
40. MCP Registry Server - Autogent MCP Ecosystem Docs, accessed December 18, 2025, https://autogentmcp.com/mcp-registry/
41. MCP Registry Architecture: A Technical Overview - WorkOS, accessed December 18, 2025, https://workos.com/blog/mcp-registry-architecture-technical-overview
42. prisma - MCP Server Registry - Augment Code, accessed December 18, 2025, https://www.augmentcode.com/mcp/prisma
43. Prisma MCP Server by prisma - Glama, accessed December 18, 2025, https://glama.ai/mcp/servers/@prisma/prisma
44. Security Best Practices - Model Context Protocol, accessed December 18, 2025, https://modelcontextprotocol.io/specification/draft/basic/security_best_practices
45. How to Build an MCP Server in Node.js to Provide Up-To-Date API Documentation | Snyk, accessed December 18, 2025, https://snyk.io/articles/how-to-build-an-mcp-server-in-node-js-to-provide-up-to-date-api/

46. Introducing the MCP Registry | Model Context Protocol Blog, accessed December 18, 2025, http://blog.modelcontextprotocol.io/posts/2025-09-08-mcp-registry-preview/
47. v0.dev: The Future of UI Development - CodeParrot AI, accessed December 18, 2025, https://codeparrot.ai/blogs/v0dev-the-future-of-ui-development
48. Enterprise MCP (Model Context Protocol) — Part Two - FactSet Insight, accessed December 18, 2025, https://insight.factset.com/enterprise-mcp-model-context-protocol-part-two
49. Kafka-MCP and Qdrant-MCP: Creating and Integrating MCP Servers with Claude for Desktop | by M K Pavan Kumar | Towards Dev - Medium, accessed December 18, 2025, https://medium.com/towardsdev/kafka-mcp-and-qdrant-mcp-creating-and-integrating-mcp-servers-with-claude-for-desktop-2da1ef7727bf
50. MCP Registry and AI Gateway - TrueFoundry, accessed December 18, 2025, https://www.truefoundry.com/blog/mcp-registry-and-ai-gateway
51. Qdrant Vs Pinecone - Which Vector Database Fits Your AI Needs? - Airbyte, accessed December 18, 2025, https://airbyte.com/data-engineering-resources/qdrant-vs-pinecone
52. Powering Your AI Agent: A Deep Dive into Aman Singh's Qdrant & OpenAI MCP Server, accessed December 18, 2025, https://skywork.ai/skypage/en/powering-ai-agent-qdrant-openai/1978298404016410624
53. Supabase vs Pinecone vs Weviate vs Qdrant: Choosing the Right Vector Database for your RAG Pipeline | by Zawanah | Medium, accessed December 18, 2025, https://medium.com/@zawanah/supabase-vs-pinecone-vs-weviate-vs-qdrant-choosing-the-right-vector-database-for-your-rag-pipeline-203f7f345bea
54. Qdrant vs Pinecone: Picking the Right Vector Database - Scout, accessed December 18, 2025, https://www.scoutos.com/blog/qdrant-vs-pinecone-picking-the-right-vector-database
55. Top Vector Database for RAG: Qdrant vs Weaviate vs Pinecone - Research AIMultiple, accessed December 18, 2025, https://research.aimultiple.com/vector-database-for-rag/
56. Model Context Protocol Servers - Augment Code, accessed December 18, 2025, https://www.augmentcode.com/mcp
57. Building an MCP Registry: Why It Matters and How to Get It Right - Obot AI, accessed December 18, 2025, https://obot.ai/building-an-mcp-registry-why-it-matters-and-how-to-get-it-right/
58. V0.Dev Review: Build your Website With No Code - Times Of AI, accessed December 18, 2025, https://www.timesofai.com/brand-insights/v0-dev-review/
59. How to Use v0 by Vercel for Beginners - Ultimate QA, accessed December 18, 2025, https://ultimateqa.com/how-to-use-v0-by-vercel-for-beginners/
60. Securing Model Context Protocol (MCP) Servers: Threats and Best Practices - Corgea, accessed December 18, 2025,

https://corgea.com/Learn/securing-model-context-protocol-(mcp)-servers-threats-and-best-practices
61. How to Secure Model Context Protocol (MCP) | by Tahir | Dec, 2025, accessed December 18, 2025, https://medium.com/@tahirbalarabe2/how-to-secure-model-context-protocol-mcp-01339d9e603c
62. How to build and deploy a Model Context Protocol (MCP) server | Blog - Northflank, accessed December 18, 2025, https://northflank.com/blog/how-to-build-and-deploy-a-model-context-protocol-mcp-server
63. How to Build Your Own MCP Server from Scratch [6 Steps] - Intuz, accessed December 18, 2025, https://www.intuz.com/blog/how-to-build-mcp-server
64. How to Build an MCP Server (Step-by-Step Guide) 2025 - Leanware, accessed December 18, 2025, https://www.leanware.co/insights/how-to-build-mcp-server
65. 5 Best Practices for Building MCP Servers - Snyk, accessed December 18, 2025, https://snyk.io/articles/5-best-practices-for-building-mcp-servers/
66. Model Context Protocol - GitHub, accessed December 18, 2025, https://github.com/modelcontextprotocol