MCP Sigil Resolver Server – Design Proposal

Background
Sigils are currently resolved inside the Django application through core.sigil_resolver, which inspects the SigilR

The product roadmap calls for exposing the same capabilities outside of Django via OpenAI's Model Context Pr

Goals
- Provide an MCP-compliant server that exposes the existing resolution logic without duplicating business rule
- Enable secure remote resolution for arbitrary text with nested sigils as well as utility discovery (e.g., which ro
- Support opt-in per-session context (current user, current entity instances) so downstream agents can resolve
- Keep the gway fallback so existing external integrations remain functional, but make it optional when MCP ac

Non-Goals
- Replacing the in-process Django resolver.
- Offering arbitrary database access beyond the existing sigil semantics.
- Shipping a production-grade authentication service; the first version will rely on API keys and reverse proxies

Protocol Surface
The server will expose the following MCP constructs:
- Tool – resolveSigils: Resolve one or more sigils inside a text payload using the current session context.
- Tool – resolveSingle: Resolve a single sigil token and return its value or unresolved form.
- Tool – describeSigilRoot: Provide metadata for a given sigil root leveraging Sigil Builder data.
- Tool – setContext: Update the thread-local sigil context to mimic request-derived state.
- Resource – sigilRoots: Publish changes to SigilRoot objects so sessions can react without polling.

Server Architecture
manage.py mcp_sigil_server
■■■ SigilResolverServer (mcp.server.sse.SseServer)
■■■ SigilSessionState (per-connection context)
■ ■■■ SigilContextAdapter ↔ core.sigil_context
■■■ SigilResolverService
■ ■■■ uses core.sigil_resolver.resolve_sigils
■■■ SigilRootCatalog (queries SigilRoot + caches metadata)

Entry point – New Django management command mcp_sigil_server imports django.setup(), instantiates an Sse
Session state – accept_session captures authentication headers, initializes SigilSessionState, and logs unresol
Resolver service – Wrap resolve_sigils, applying session context before each call and checking metadata for u
Sigil root catalog – Cache describeSigilRoot metadata by reusing core.sigil_builder queries.
Gway integration – Keep _resolve_with_gway untouched so the fallback path remains intact.

Configuration & Security
Dependencies – Add modelcontextprotocol to requirements and provide shared schemas in core/mcp.
Settings – Introduce settings.MCP_SIGIL_SERVER with host, port, and api_key plus a --public override.
Authentication – Require Authorization: Bearer tokens that match settings.MCP_SIGIL_API_KEYS.
Rate limiting – Enforce per-session throttles and expand to Redis-backed limits later if needed.

Implementation Plan
Foundations – Add core/mcp modules and implement SigilResolverService.
MCP server – Connect the service to the SseServer and expose tools/resources.

Admin & Ops – Document provisioning steps and extend deployment scripts.
Testing – Add unit and integration coverage, including context-aware fixtures.

Operational Considerations

Observability – Emit structured logs and metrics.
Error handling – Map validation errors to MCP ToolError responses.
Scalability – Run the MCP server alongside Django to avoid mismatched configuration.

Future Enhancements

- Add prompt registrations for sigil-building instructions.
- Support MCP jobs for long-running resolution tasks once available.
- Replace the gway fallback with a dedicated MCP proxy tool.