TechnicalDocsMcpController

September 4, 2026

TechnicalDocsMcpController

Kind: Controller

Source: `atloria-monorepo/apps/api/src/technical-docs/technical-docs-mcp.controller.ts`

Hosts each project's technical docs as a live MCP server over Streamable HTTP, so coding agents
(Claude, Cursor, …) can add `…/technical-docs/:projectId/mcp` and query the docs + entity graph.

Stateless: each POST is a self-contained JSON-RPC exchange (no server-held session), which fits a
read-only docs server and scales trivially. Access mirrors the public docs — a publicly-published
project is open; otherwise a bearer token whose user can access the project is required. (Private
projects with the full MCP OAuth 2.1 flow are a follow-up.)

`TechnicalDocsMcpController` exposes each project’s technical documentation as a live MCP server over Streamable HTTP, enabling coding agents (Claude, Cursor, etc.) to query docs and the entity graph via a JSON-RPC interface. Each request is stateless (a self-contained JSON-RPC exchange), making it scalable and suitable for a read-only docs surface. Access mirrors the public docs: published projects are open; otherwise a bearer token must grant the caller access to the project.

Diagram

```
graph LR
  A[Agent / Client<br/>Claude, Cursor, CLI] -->|POST JSON-RPC<br/>/technical-docs/:projectId/mc
  B --> C{Authorization}
  C -->|Project is published| D[Allow]
  C -->|Not published| E[Verify Bearer Token<br/>User has project access]
  E -->|OK| D[Allow]
  E -->|Denied| F[401/403]
```

```
  D --> G[Docs + Entity Graph Provider]
  G --> H[JSON-RPC Response<br/>(Streamable HTTP)]
  H --> A
```

Usage

```
// Example: calling the MCP endpoint from Node.js using JSON-RPC over HTTP.
// This is the typical way a coding agent or tool would interact with the controller.

const projectId = "proj_123";
const baseUrl = "https://api.example.com";
const url = `${baseUrl}/technical-docs/${projectId}/mcp`;

// Include Authorization only for non-public (unpublished) projects.
const token = process.env.API_TOKEN; // optional
const headers: Record<string, string> = {
  "content-type": "application/json",
  ...(token ? { authorization: `Bearer ${token}` } : {}),
};

const jsonRpcRequest = {
  jsonrpc: "2.0",
  id: "req-1",
  method: "docs.search", // method names depend on your MCP toolset exposed by the server
  params: {
    query: "How is billing handled in the API?",
    limit: 5,
  },
};

const res = await fetch(url, {
  method: "POST",
  headers,
  body: JSON.stringify(jsonRpcRequest),
});

if (!res.ok) {
  const text = await res.text();
  throw new Error(`MCP request failed: ${res.status} ${res.statusText}\n${text}`);
}

const json = await res.json();
console.log("JSON-RPC response:", json);
```

AI Coding Instructions

- Preserve the stateless contract: every `POST` must be a complete JSON-RPC exchange (no server-held sessions, no per-connection state).

- Keep auth rules aligned with public docs behavior: allow published projects anonymously; require `Authorization: Bearer ...` and validate project access for unpublished projects.
- Treat this as read-only infrastructure: avoid adding mutations or side effects (writes, background jobs) to MCP handlers exposed through this controller.
- Maintain Streamable HTTP compatibility (chunked/streaming response semantics where applicable) and avoid middleware that buffers the entire response body.
- When extending methods or integrating new docs/entity sources, keep response shapes stable for agents and ensure errors are proper JSON-RPC errors (not ad-hoc HTTP payloads).

Relationships

- MODULE_DECLARES → `ragIndex`
- MODULE_DECLARES → `enrich`
- MODULE_DECLARES → `handle`
- MODULE_DECLARES → `mcpGet`
- MODULE_DECLARES → `getLlmsTxt`
- MODULE_DECLARES → `getLlmsFullTxt`
- MODULE_DECLARES → `getPageMarkdown`
- MODULE_DECLARES → `downloadSdk`
- MODULE_DECLARES → `ask`
- MODULE_DECLARES → `assist`
- MODULE_DECLARES → `assistApply`
- MODULE_DECLARES → `playgroundProxy`
- DEPENDS_ON → `TechnicalDocsMaterializerService`
- DEPENDS_ON → `TechnicalDocsService`
- DEPENDS_ON → `TechDocsRagService`
- DEPENDS_ON → `TechnicalDocsQueue`
- DEPENDS_ON → `PrismaService`
- DEPENDS_ON → `AIUsageService`
- DEPENDS_ON → `DraftDocumentService`
- DEPENDS_ON → `PlaygroundProxyService`
- DEPENDS_ON → `DocsVisibilityService`

Referenced By

- `TechnicalDocsModule` (MODULE_DECLARES)