Here's a **detailed breakdown** of the features of a Model Context Protocol (**MCP**) server—including main types, sub-types, and other supporting features—so you have a comprehensive overview for designing or integrating such a server in your architecture.

---

1. Main Capability Types

An MCP server typically supports **three primary capability classes** (sometimes also referred to as "main types"). These define the kinds of things an MCP client (e.g., an AI agent or LLM-driven system) can do via the server. (Model Context Protocol)

1.1. Resources

Definition: Data or content exposed by the server that can be *read* (and sometimes written) by the client. These are akin to "files, records, responses, content blobs". (Model Context Protocol)
Examples / Sub-types:

- File-like data: document text, spreadsheets, logs
- API responses: database query results, search results
- Metadata or context snapshots for LLM use
  Key characteristics:
- URI-based access patterns. (Model Context Protocol)
- The server may allow listing of resources. (Hugging Face)
- The client can fetch and use these in generating responses.

1.2. Tools (Actions)

Definition: Functions (procedures, operations) that clients can *invoke* to cause an action or effect in the external system. Tools differ from mere read-only resources—they are procedural. (Qodo)
Examples / Sub-types:

- Create a ticket in a tracking system
- Run a query or execute a database change
- Trigger a deployment, run tests, send an email
  Key characteristics:
- Each tool has a name, input schema (JSON), output schema (JSON) and handler logic. (Qodo)
- Clients can discover the list of tools ("tools/list") at runtime. (Hugging Face)
- Tools may generate notifications or progress events.

1.3. Prompts (Templates / Workflows)

Definition: Pre-written prompt templates or workflow specifications provided by the server to the client as a guide or for reuse. (Model Context Protocol)
Examples / Sub-types:

- A prompt template for "summarize document and list action items"
- A workflow template for "onboarding new user: provision account, send welcome email, schedule call"

Key characteristics:

- Helps reuse common patterns, making it easier for LLMs/agents to perform standard tasks
- Exposed via "prompts/list" so the client can select applicable templates. (Hugging Face)

---

2. Sub-Types / Additional Features

Beyond the main three capability types, an MCP server can incorporate finer-grained or supporting features. These are often needed for robustness, performance, security and operational maturity.

2.1. Capability Discovery & Negotiation

- When a client connects, it should be able to ask: "What tools, resources, prompts are available?" (e.g., `tools/list`, `resources/list`, `prompts/list`). (Hugging Face)
- The server may negotiate capabilities (e.g., versioning, supported transports, schemas). (Model Context Protocol)
- Ensures clients adapt to server features dynamically.

2.2. Transport Layers & Protocols

- MCP typically uses JSON-RPC 2.0 for messaging. (Model Context Protocol)
- Transports supported can include: STDIO (local process), HTTP/HTTPS streaming, WebSocket or SSE for async notifications. (Home)
- The server must manage connection lifecycle, message framing, error handling. (WorkOS)

2.3. Concurrency & Session Management

- The server must handle multiple client connections concurrently (sessions). (Model Context Protocol)
- It may track progress of tool invocations, support cancellation, streaming outputs.
- It may provide structured logging, progress tracking, notifications for clients. (Model Context Protocol)

2.4. Security, Permissions & Governance

- The server must enforce authentication/authorization to restrict access to resources, tools. For example, in an enterprise the server might integrate with identity services (e.g., Microsoft Entra ID) for RBAC. (Microsoft Learn)
- Data masking or filtering may be applied to prevent exposure of sensitive data. (Medium)
- There are known research concerns about tool misuse and security risks in MCP workflows. (arXiv)

2.5. Real-Time Notifications & Change Streams

- Servers can push notifications to connected clients when state changes (e.g., new tool added, resource updated). (Model Context Protocol)
- Supports event-driven workflows and keeps client view synchronized.

2.6. Schema Management & Validation

- Tools and resources include input/output schemas (often JSON Schema) so clients/LLMs know exactly what to send/expect. (Model Context Protocol)
- Ensures structured and predictable interactions.

2.7. Logging, Monitoring, Telemetry

- Track invocation logs, errors, usage metrics, tool invocation performance. (Model Context Protocol)
- Helps with debugging, audit, SLA enforcement, operational transparency.

2.8. Context & Session State

- MCP servers may maintain session context (for particular clients or agents) and support stateful workflows (previous calls, history, etc.). (WorkOS)
- May integrate with memory stores, caches, or context stores to serve richer results.

2.9. Extensibility & Customization

- Ability to add new tools or resources dynamically, versioning, plug-in subsystems.
- A large ecosystem now exists (per GitHub list) of various MCP server implementations for different domains. (GitHub)

2.10. Performance, Caching & Optimization

- Caching of frequently-used resources, perhaps pre-fetching tools or schema information. (WorkOS)
- Ensure low latency for interactive LLM use.

---

3. Feature Map – Organized by Category

Here's a quick structured map of features you might include in your "complete features" list, grouped by functional category.

Category	**Feature**
Capability Types	Resources, Tools, Prompts
Discovery / Negotiation	tools/list, resources/list, prompts/list; versioning; capability negotiation
Transport & Protocol	JSON-RPC 2.0, STDIO, HTTP/SSE/WS, streaming responses
Session & Concurrency	Multi-client sessions, cancellation, progress tracking
Schema & Validation	Input/output JSON schemas, tool metadata
Real-Time & Eventing	Notifications of changes, streaming updates
Security & Governance	Authentication, authorization, RBAC, data filtering/masking
Logging & Monitoring	Structured logs, telemetry, usage metrics
Extensibility	Plugin architecture, dynamic tool/resource addition, version control
Performance & Optimization	Caching, offline resource indexing, response latency tuning

Category	Feature
Context Management	Session context, memory/state, history tracking

4. Practical Use Cases & Implementation Considerations

- If you're building one for your backend (you mentioned Rust microservices etc.), you'll want to decide **which parts to support** (Tools + Resources + Prompts) and **which transport** (maybe HTTP streaming).
- Ensure you build **schema definitions** for each tool so your LLM/agent knows how to call them.
- Security must be front and centre—tools can do things like deploy infra or query data; you need permission checks, auditing, possibly isolation.
- Design for concurrency and multi-session management; many clients/agents may connect simultaneously.
- Plan for discovery: clients need to know what's available.
- If it's enterprise-grade, add monitoring/telemetry/logging so you can trace usage and diagnose issues.

If you like, I can **generate a complete feature checklist** (e.g., in markdown or table form) tailored for your Rust microservice architecture (Auth, User Management, Gateway, Storage, etc.) so you can plug the MCP server as a service in your system. Would you like that?