General Guidelines for Building Custom MCP Servers for Klavis AI

This document outlines the foundational principles, development workflow, and quality standards we expect for all Model Context Protocol (MCP) server projects.

Our Core Development Philosophy

An MCP server is more than just an API wrapper; it's the vocabulary an AI agent uses to understand and interact with a system. Our philosophy is centered on creating a clear, reliable, and intuitive "language" for the AI.

- **User-Centric, AI-Driven:** The primary user of your tools is the Large Language Model (LLM). Your design choices for tool names and descriptions directly impact the AI's ability to succeed. Always ask: "If an AI only read this tool's name and description, would it know exactly when and how to use it?"
- **Atomicity is Key:** Tools must be atomic, meaning they should perform one specific job and do it well. Avoid creating a single, complex `manage_everything` tool. Instead, break down functionality into discrete actions like `list_users`, `get_user_details`, `create_user`, and `deactivate_user`. This granularity gives the AI precision and control.
- **Clarity Over Brevity:** Tool names should be descriptive and unambiguous (e.g., `create_pull_request`). The description is the most critical documentation; it must clearly explain the tool's purpose, its expected inputs, what it returns, and any important side effects.
- **Robust and Reliable:** Servers must be resilient. This includes graceful handling of API errors (e.g., invalid permissions, not found, rate limits) and returning clear, understandable error messages that the AI can potentially use to self-correct.

Standard Project Workflow

We follow a structured process for every new MCP server to ensure quality and alignment.

Step 1: Kick-off and API Familiarization
Every project begins with a specific brief that identifies the target service (e.g., a CRM, a project management tool, a cloud provider). Your first task is to familiarize yourself with its public API, paying close attention to:

- **Authentication:** Understand the required authentication mechanism (API Key, OAuth, etc.). Your server implementation must handle this securely, typically by reading credentials from environment variables.

- **Core Objects:** Identify the main resources in the API (e.g., `users`, `tickets`, `projects`).
- **Rate Limiting:** Be aware of any API usage limits.

Step 2: Tool Scoping and Definition
Following your API review, you are responsible for defining the server's complete toolset. Your task is to analyze the target service's most common user workflows and design a set of atomic, intuitive tools that map to those actions.

Step 3: Implementation
With the toolset defined, development can begin.
- **Language:** We prefer servers to be built using Python.
- **Code Quality:** The code should be clean, well-commented, and organized logically.

Step 4: Rigorous Testing and Documentation
This is the most critical phase and a non-negotiable part of our process. Testing validates that the tools not only function correctly but are also effectively triggered by natural language commands. For every tool you build, you must perform end-to-end testing as specified in our main **Contributor's Guide to Building MCP Servers** and general **Contributing Guide**.

Key Deliverables

For any MCP server, the final submission in the form of pull request to https://github.com/Klavis-AI/klavis must include:

1. **Complete Source Code:** The full, working source code for the MCP server, organized and documented.
2. **Comprehensive `README.md`:** A clear markdown file that explains:
    - The purpose of the server.
    - Step-by-step instructions for installation and setup.
    - Detailed guidance on how to acquire and configure the necessary API credentials (including environment variable names).
    - Instructions on how to run the server.
3. **Mandatory "Proof of Correctness":** You must provide short video recordings or a series of clear screenshots for **every single tool**. This evidence must show a natural language query in a client (like Claude Desktop or Cursor), the server logs confirming the correct tool was called, and the successful result. This serves as undeniable proof of functionality and living documentation.

Thank you!