PROJECT APM: FINAL MVP PRD

Product Name: APM (Agent Package Manager)

Version: 1.0.0 ("The Universal Bridge")

Mission: To become the "npm" of the Agentic AI era.

Tagline: "Write Once, Run on Claude, Cursor, or Codex."

---

1. Executive Summary

APM is a Rust-based CLI tool that solves the "Configuration Fatigue" in AI-assisted development. Instead of manually configuring System Prompts, MCP Servers, and Documentation files for every new project, developers run `apm install <agent>`.

APM acts as a **Transpiler**: It reads a universal `agent.yaml` definition and compiles it into the native format of the user's environment (injecting into `~/.claude/` for Claude Code or `.cursor/` for Cursor).

---

2. The Universal Schema: `agent.yaml`

This is the core IP. All agents in the registry must follow this strict schema.

YAML

```yaml
# agent.yaml - The Source of Truth

name: "rust-architect"

version: "1.0.0"

description: "Senior Rust Systems Engineer optimized for Tokio & Zero-cost abstractions"

author: "agenza-labs"


# 1. Identity (The Brain)

# This becomes the System Prompt or .mdc rule

identity:

  model: "claude-3-5-sonnet-latest"

  icon: "🦀"

  system_prompt: |
```

```
    You are a specialized Rust subagent.
    - You prefer composition over inheritance.
    - You use `anyhow` for apps and `thiserror` for libs.
    - You strictly follow borrow checker patterns.

# 2. Skills (The Knowledge)
# These become Markdown files or RAG context
skills:
  - name: "tokio-patterns"
    content: |
      # Tokio Best Practices
      - Use `tokio::spawn` for async tasks.
      - Use `task::spawn_blocking` for CPU-heavy work.
      - Avoid blocking the thread.

# 3. Tools (The Hands)
# These become MCP Server Configs (JSON)
mcp:
  - name: "cargo-mcp"
    command: "cargo"
    args: ["mcp-server"]
    env:
      RUST_LOG: "info"
  - name: "postgres-mcp"
    command: "docker"
    args: ["run", "--rm", "-i", "mcp/postgres"]
```

---

3. Technical Architecture

Stack: Rust (Stable).

Distribution: Single binary (apm).

Registry: GitHub Raw Content (Serverless).

The "Adapter" Pattern

The CLI uses a trait-based system to handle different editors.

```
Rust

// Logical Architecture

trait AgentInstaller {

    fn install_identity(&self, agent: &Agent);

    fn install_skills(&self, agent: &Agent);

    fn install_tools(&self, agent: &Agent);

}



struct ClaudeInstaller; // Implementation for ~/.claude

struct CursorInstaller; // Implementation for .cursor/rules
```

---

4. Command Specification

`apm init`

- **Action:** Detects installed editors (Claude Code, Cursor, VS Code).
- **Output:** Creates `~/.apm/config.toml` storing preference (e.g., `default_target = "cursor"`).

`apm list`

- **Action:** Fetches `registry.json` from GitHub.
- **Output:** Displays a clean ASCII table of available agents.

`apm install <agent_name> [flags]`

- **Flags:** `--target claude` (default), `--target cursor`, `--global`.

- **Logic:**
    1. Fetches `agent.yaml`.
    2. **If Target = Claude:**
        - Writes `~/.claude/agents/{name}.json` (Identity).
        - Writes `~/.claude/skills/{name}/*.md` (Skills).
        - Patches `claude_desktop_config.json` (MCP Tools).
    3. **If Target = Cursor:**
        - Creates `.cursor/rules/{name}-identity.mdc` (Identity).
        - Creates `.cursor/rules/{skill}.mdc` (Skills).
        - Patches `.cursor/mcp.json` (MCP Tools).
    4. **Validation:** Checks if required tools (Docker/Cargo) are in PATH.

---

5. Development Roadmap (5-Day Sprint)

Day 1: The Core & Registry

- **Goal:** Read the Registry.
- **Task:**
    - Initialize `apm-cli` Rust project.
    - Create `github.com/agenza-labs/registry`.
    - Upload `rust-architect.yaml`.
    - Implement `apm list` to fetch and print it.

Day 2: The Claude Adapter

- **Goal:** Native Claude Support.
- **Task:**
    - Implement `ClaudeInstaller` struct.
    - Write logic to locate `~/.claude` (using `dirs` crate).
    - Write logic to safely JSON-merge `claude_desktop_config.json` (using `serde_json`).

Day 3: The Cursor Adapter (The "Moat")

- **Goal:** Cross-Platform Magic.
- **Task:**
    - Implement `CursorInstaller` struct.
    - Write logic to generate `.mdc` files with correct frontmatter.
    - Write logic to create/update `.cursor/mcp.json`.

Day 4: Polish & UX

- **Goal:** "Apple-style" Minimalism.
- **Task:**

- Add indicatif progress bars ("Installing Rust Architect...").
- Add colored output (Green checks for success).
- Add error hints ("Docker not found. Please install Docker to use this agent.").

Day 5: Launch Content ("Golden Agents")

- **Goal:** 3 Perfect Agents.
- **Task:**
  1. **rust-architect:** Full Tokio/Actix expertise.
  2. **fullstack-next:** Next.js 15 + FastAPI + ShadcnUI.
  3. **qa-testing-squad:** Playwright + Jest configuration.
- **Marketing:** Record the demo video showing the same agent being installed into **both** Claude and Cursor.

---

6. Prompt for AI Implementation

Copy/Paste this to Opus 4.5 to start Day 1:

> "Act as a Senior Rust Systems Engineer. We are building apm (Agent Package Manager), a CLI that installs AI Agent configurations into Claude Code and Cursor.
>
> **Task:** Initialize the project structure.
>
> **Requirements:**
>
> 1. Use clap for CLI commands (init, install, list).
> 2. Use serde for parsing YAML (registry) and JSON (config files).
> 3. Use reqwest for fetching from GitHub.
> 4. Create a trait Installer with methods install_identity, install_skills, and install_tools.
> 5. Create a struct AgentConfig that matches this YAML schema: [Insert Schema from Section 2].
>
> Please write the main.rs and the models.rs file definitions."

---

7. Success Metric (The "Fastlane" Check)

- **MVP Success:** You can run apm install rust-architect on a fresh machine, open Claude Code, and ask "Create a new Actix project," and it automatically has the context and tools to do it without manual prompting.

- **Monopoly Indicator:** Users start asking, "Does this work for Cursor too?" (And you say "Yes").