To autonomously develop the HERMES app for your hackathon sprint, you can provide the SPARC Orchestrator with a detailed prompt that leverages its specialized modes while explicitly addressing your constraint regarding mock and Jest data. The Orchestrator's role is to break down large objectives into delegated subtasks, ensuring secure, modular, testable, and maintainable delivery [1, 2].

Here's a prompt designed for the SPARC Orchestrator, incorporating the HERMES project details and your specific requirements:

**"SPARC Orchestrator, initiate the autonomous development of the HERMES AI voice agent system, targeting a production-ready Minimum Viable Product (MVP) within this weekend's hackathon sprint. Prioritize direct integration with specified third-party services over extensive mock or Jest-specific test data generation. Adhere strictly to the SPARC methodology, ensuring modularity, security, and maintainability (files under 500 lines, no hard-coded environment variables or secrets) at every step.**

**The core objective is to deliver an AI-powered voice agent for law firms capable of handling 24/7 inbound client communications, blending real-time voice processing with legal-safe AI responses and multi-CRM integration.**

**Delegate the following comprehensive plan across the SPARC modes:**

1.  **Specification & Pseudocode (`spec-pseudocode`):**
    *   **Capture the complete functional requirements for HERMES**: A 24/7 AI-powered voice agent handling client communications, incorporating real-time voice processing, legally compliant AI responses, and seamless CRM integration [3, 4].
    *   **Develop a modular pseudocode blueprint** for the **Voice Processing Pipeline** (utilizing OpenAI Whisper for STT, Kokoro FastAPI for TTS, WebSocket-based real-time streaming, Voice Activity Detection with <100ms target latency) [3, 5, 6], the **AI Response Engine** (integrating OpenRouter API with 8B LLMs, Tree of Thought (ToT) reasoning, Monte Carlo Tree Search (MCTS) validation, automatic profit margin tracking (20% minimum), and legal safety validators with disclaimer injection) [7-10], and the **CRM Integration Layer** (using an abstract adapter pattern, with primary focus on Clio via OAuth 2.0 and webhook support, and outlining unified data models for contacts and matters) [7-9, 11, 12].
    *   **Define strict legal safety constraints**: Prohibited phrases (e.g., "legal advice", "guaranteed outcome", "case evaluation"), required disclaimers in every response, citation verification, and a confidence threshold of 0.85 for response acceptance, with a fallback to human transfer on low confidence [13-16].
    *   **Crucially, design data models (`LegalContact`, `LegalMatter`) directly for CRM interaction, and avoid generating boilerplate mock data structures or Jest-specific test data within the specification itself, unless explicitly required for illustrating API contract flows [9, 12, 14].**

2.  **Architecture (`architect`):**

    *   **Design a scalable, secure, and modular microservices architecture** based on Python 3.11, FastAPI, Docker, and GCP Cloud Run for production [17, 18].
    *   **Detail the infrastructure stack**: PostgreSQL 15 for multi-tenant data, Redis 7 for caching and session management, and Celery with Redis broker for asynchronous tasks [17, 19, 20].
    *   **Define clear API boundaries, data flows, and service segmentation** for each core component, including the WebSocket voice endpoint, REST APIs for conversations and matters, and webhook endpoints [21-25].

3.  **Code (`code`), Testing (`tdd`), Debugging (`debug`), Security Review (`security-review`), and Optimization (`refinement-optimization-mode`):**
    *   **Implement the HERMES system iteratively**, focusing on the outlined components and their functionality.
    *   **For the TDD phase, prioritize functional and integration tests** that interact with the actual (or sandbox) external services (OpenAI Whisper, Kokoro, OpenRouter, Clio) to validate the system's behavior. **Explicitly avoid generating extensive mock data or Jest-specific test configurations, instead focusing on minimal test cases directly against the established interfaces and real data models.**
    *   **Ensure the implementation adheres to security requirements**: JWT tokens with tenant isolation, TLS 1.3 encryption, 90-day data retention for transcripts, 7-year retention for matters, HIPAA compliance capabilities, and comprehensive audit logging [26].
    *   **Implement performance targets**: <100ms voice processing, <500ms total voice latency, <2 seconds initial greeting, <200ms API response time (95th percentile), and 99.9% uptime SLA [17, 21].
    *   **Integrate profit tracking middleware** for LLM usage, calculating costs and revenue with a 50% markup to ensure a 20% minimum profit margin [10, 27-30].
    *   **Implement robust error recovery patterns**, including fallback to human operators on low confidence [28].
    *   **Ensure all code is modular, adheres to the <500 lines per file rule, and externalizes all configurations and secrets** to environment variables or managed secret services [2, 31-33].

4.  **Integration (`integration`):**
    *   **Assemble all developed components** (Voice Processing Pipeline, AI Response Engine, CRM Integration Layer) into a cohesive, functional system.
    *   **Verify the WebSocket voice conversation endpoint (`/voice/{tenant_id}`) and CRM webhook receivers (`/webhooks/{crm_type}`)** are fully functional and secure [22, 23].

5.  **Documentation (`docs-writer`) & Monitoring (`post-deployment-monitoring-mode`):**
    *   **Generate clear, concise Markdown documentation** covering HERMES's usage, API endpoints, setup, and configuration [24, 25].
    *   **Implement basic monitoring and analytics**, including Prometheus metrics for legal AI performance and conversation success rates, and structured logging for all legal interactions [16, 24].

**Ensure that every delegated subtask initiated with `new_task` is concluded with a concise `attempt_completion` summary, clearly outlining the outcomes and readiness for the next phase. Focus on achieving a demonstrable and functional HERMES MVP by the end of the sprint, prioritizing a working system over exhaustive, mock-data-dependent testing which is not required for this hackathon context."** [2, 34]