Execute Task: 6. Build LLM integration with Ollama 6. Build LLM integration with Ollama

**Kiro**

Including Steering Documents

GenRules.md

I'll implement task 6 to build LLM integration with Ollama. Let me start by updating the task status and then examining the current codebase structure.

Task: *6. Build LLM integration with Ollama* from tasks.md

Status: In Progress

Now let me examine the current project structure to understand the existing architecture:

Read file(s)

config.py

Let me check the directory structure to understand the existing services:

Searched workspace

Examining the services directory to understand existing service implementations

Searched workspace

Examining the logic directory to understand existing logic implementations

Now let me examine some existing service implementations to understand the patterns:

Read file(s)

pald\_service.py

consent\_service.py

\_\_init\_\_.py

Now let me check the existing tests to understand the testing patterns:

Read file(s)

test\_pald\_service.py

Now I'll implement the LLM integration with Ollama. Based on the existing patterns, I need to create:

1. Abstract LLMProvider interface
2. OllamaProvider implementation
3. LLM service layer
4. Comprehensive unit tests

Let me start by creating the abstract LLM provider interface:

Created

llm\_provider.py

Now let me create the LLM service layer that uses the provider:

Created

llm\_service.py

Now let me create the logic layer for LLM operations:

Created

llm.py

Now let me create comprehensive unit tests for the LLM integration:

Canceled creation

test\_llm\_provider.py

Restart creating test\_llm\_provider. Please use Ollama, the installed modells are llama3.2, Llava, Mistral and a few more.

**Kiro**

Including Steering Documents

GenRules.md

I'll help you recreate the test\_llm\_provider.py file using Ollama with the available models (llama3.2, Llava, Mistral). Let me first examine the current test file and the LLM provider implementation to understand what needs to be tested.

Read file(s)

test\_llm\_provider.py

llm\_provider.py

llm\_service.py

Now I'll recreate the test\_llm\_provider.py file with comprehensive tests for Ollama using the available models (llama3.2, Llava, Mistral):

Accepted edits to

test\_llm\_provider.py

Now I'll create a comprehensive test file focused on Ollama with the available models:

Accepted edits to

test\_llm\_provider.py

Perfect! I've recreated the test\_llm\_provider.py file with comprehensive tests specifically designed for Ollama with the available models (llama3.2, Llava, Mistral). Here's what the new test file includes:

**Key Features:**

1. **Model-Specific Tests**: Tests for all three available models (llama3.2, Llava, Mistral)
2. **Comprehensive Coverage**: Tests for both regular and streaming responses
3. **Vision Model Support**: Special tests for Llava's image processing capabilities
4. **Parameter Testing**: Tests various Ollama parameters like temperature, top\_p, top\_k, etc.
5. **Error Handling**: Tests for timeouts, connection errors, and HTTP errors
6. **Integration Tests**: Optional integration tests that can run against a real Ollama instance
7. **Mock Tests**: Extensive mocking for unit testing without requiring Ollama to be running

**Test Categories:**

* **Data Classes**: Tests for LLMRequest, LLMResponse, and LLMStreamResponse
* **Mock Provider**: Tests for the MockLLMProvider used in testing
* **Ollama Provider**: Comprehensive tests for the OllamaProvider
* **Model-Specific Behavior**: Tests tailored to each model's characteristics
* **Integration Tests**: Optional tests that require a running Ollama instance

**Special Features:**

* **Vision Model Testing**: Specific tests for Llava's image processing capabilities
* **Streaming Response Testing**: Tests for real-time streaming responses
* **Token Count Extraction**: Tests for different token counting formats from various models
* **Model Information**: Tests for retrieving detailed model information
* **Health Checks**: Tests for service availability checking

The tests are designed to work with your existing codebase and follow the development rules you've established. They use proper mocking to avoid requiring a running Ollama instance for most tests, while also providing integration test options for when you want to test against the real service.

if task 6 is finished continue with task 7