I'll create a UML class diagram describing the domain model for the prompt editor codebase. After reviewing the provided files, I'll produce a comprehensive diagram that captures both the actual classes from the codebase and the inferred relationships based on the API documentation.

Prompt Editor Domain Model Analysis

After analyzing the provided codebase and API documentation, I've created a UML class diagram that represents the domain model for the prompt editor application. The diagram captures both the actual implemented classes from the codebase and the inferred relationships based on the API documentation.

Key Components

Core Data Models

1. **Prompt**: The central entity representing a prompt template with its metadata and lineage. Key properties include:
   - A unique identifier
   - Template text with parameters
   - Metadata for author, version, and other attributes
   - Parent/lineage tracking for derived prompts
   - Ability to render with parameters
2. **PromptTemplate, PromptParameter, PromptConfig**: These classes support the core Prompt entity with specific responsibilities:
   - **PromptTemplate**: Holds the text template and parameters
   - **PromptParameter**: Defines typed parameters that can be injected into templates
   - **PromptConfig**: Contains LLM configuration settings (temperature, tokens, etc.)
3. **PromptVersion**: Represents historical versions of prompts with commit information

Backend Services

1. **PromptRepository**: Manages git-based storage and versioning for prompts
2. **LineageTracker**: Tracks prompt creation, updates, tests, and usage
3. **LLMService**: Provides an abstraction for interacting with LLM providers (Anthropic, OpenAI)

API Layer

The API layer includes request/response models and routes that connect the frontend to the backend services.

Frontend Components

I've inferred several frontend components based on the codebase: - **PromptLibrary**: For browsing and selecting prompts - **PromptEditor**: For editing prompt content and metadata - **TestRunner**: For testing prompts with parameters - **VersionControl**: For managing prompt versions

Integration with Agent Architecture

The diagram also shows how the prompt editor could integrate with the agent architecture described in the documentation (`IntentService`, `TeamIntentService`, `Orchestrator`).

Design Observations

1. The system follows the agent design principles mentioned in the documentation:
   - Single responsibility for each component
   - Minimal processing
   - Clear boundaries between components
   - Forward-only flow of data
2. The configuration design principles are also evident:
   - Hierarchical configuration
   - Smart merge behavior
   - Separation of responsibilities
3. The architecture facilitates:
   - Version control for prompts
   - Testing with different parameters
   - Tracking lineage and history

   - ## Integration with multiple LLM providers

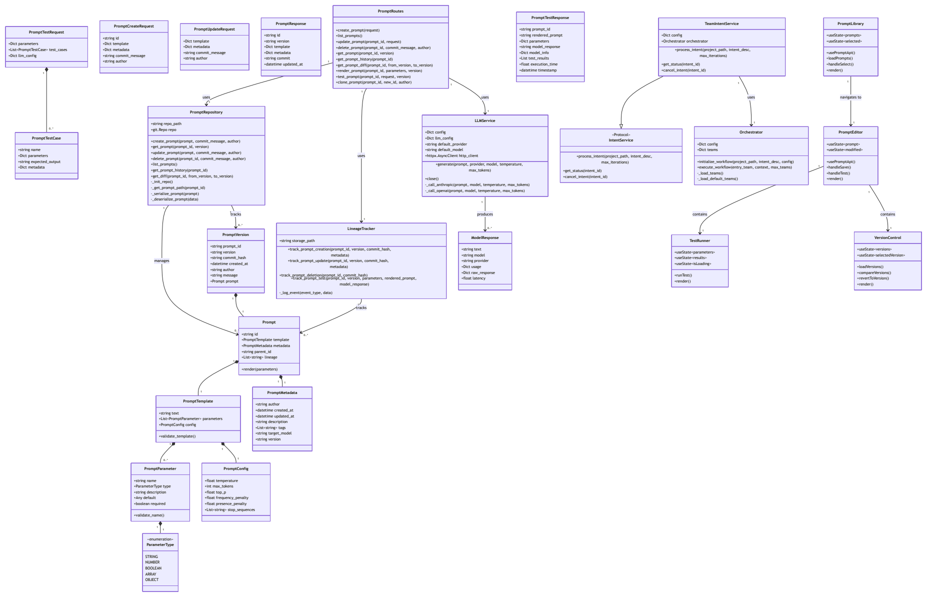

Diagram 0