# Technical Specification: Intelligent Prompt Generation System

## 1. Executive Summary

The Intelligent Prompt Generation System is a middleware application designed to intercept, enrich, and restructure user inputs before they are sent to an LLM. This document outlines the technical architecture required to support the "Layered Prompt Construction" and "Reverse Prompting" workflows.

## 2. System Architecture

### 2.1 High-Level Data Flow

1. **Client:** Captures user raw input and configuration (Tone, Output Type, Constraints).
2. **Service Layer (AI Integration):**
   * **Orchestrator:** Manages the pipeline within the client application.
   * **Gemini API Integration (Chain-of-Thought):**
     + **Pass 1 (Analysis):** Extracts key nouns and detects the domain immediately.
     + **Pass 2 (Refinement):** Checks for sufficiency. If the input is vague, it triggers the **Reverse Prompting** logic (rewriting the core task) internally.
     + **Pass 3 (Enrichment):** Selects high-value attributes.
     + **Pass 4 (Final Synthesis):** The AI Agent synthesizes all inputs (Tone, Format, Refined Task, Attributes) into a **single, fluid, natural language instruction set** (the "Expanded Prompt").
3. **Data Access:** Retrieves configuration templates (lookups) and stores usage history in Firebase Firestore.

## 3. Database Design (Schema & Lookups)

### 3.1 Entity Relationship Diagram (ERD)

erDiagram  
 Users ||--o{ PromptHistory : creates  
 PromptHistory }o--|| Lookup\_Tone : uses  
 PromptHistory }o--|| Lookup\_OutputType : uses  
 PromptHistory }o--|| Lookup\_Format : uses  
 PromptHistory }o--|| Lookup\_Domain : detected\_as  
  
 Lookup\_Domain ||--|{ DomainAttributes : contains  
  
 Users {  
 string user\_id PK  
 string email  
 timestamp created\_at  
 }  
  
 PromptHistory {  
 uuid id PK  
 string user\_id FK  
 text raw\_input  
 text final\_enriched\_prompt  
 boolean reverse\_prompting\_triggered  
 json configuration\_snapshot  
 timestamp created\_at  
 }  
  
 Lookup\_Tone {  
 string code PK "e.g., CREATIVE"  
 string display\_name  
 text instruction\_template "Persona injection..."  
 }  
  
 Lookup\_OutputType {  
 string code PK "e.g., DECK, CODE"  
 string display\_name  
 text scaffolding\_template  
 }  
  
 Lookup\_Format {  
 string code PK "e.g., JSON, SECTIONS"  
 string display\_name  
 text structure\_rules  
 boolean auto\_strip\_meta "Flag for Safe JSON Rule"  
 }  
  
 Lookup\_Domain {  
 string code PK "e.g., TECHNICAL, CULTURAL"  
 text classification\_keywords  
 }  
  
 DomainAttributes {  
 int id PK  
 string domain\_code FK  
 string attribute\_name "e.g., History, KPIs"  
 text injection\_prompt  
 }

## 4. Class Structure & Object Model

### 4.1 Core Classes

classDiagram  
 class PromptRequest {  
 +String rawText  
 +ToneType tone  
 +OutputType outputType  
 +FormatType format  
 +LengthType length  
 +Boolean stripMeta  
 +Boolean aestheticMode  
 }  
  
 class PromptBuilderService {  
 -EnrichmentEngine enrichmentEngine  
 -ReversePromptingAgent reverseAgent  
 +generate(PromptRequest req) PromptResult  
 }  
  
 class GeminiAPI {  
 +analyzeAndSynthesize(PromptRequest req) AIResponse  
 }  
  
 PromptBuilderService --> PromptRequest  
 PromptBuilderService --> GeminiAPI

## 5. Service Layer Logic

### 5.1 The Construction Algorithm (AI-Driven)

The construction logic has moved from client-side string concatenation to a **generative synthesis model**. The PromptBuilderService.generate() method now delegates the final assembly to the AI to ensure cohesion and narrative flow.

**Pseudo-Code:**

function generatePrompt(request) {  
 // 1. Prepare Configuration  
 const config = {  
 tone: LoadLookup(request.tone).instruction,  
 format: LoadLookup(request.format).rules,  
 type: LoadLookup(request.outputType).context,  
 length: LoadLookup(request.length).rules  
 };  
  
 // 2. Call Gemini API (Chain-of-Thought)  
 // The System Prompt instructs the AI to:  
 // A. Detect Domain  
 // B. Check Sufficiency (Reverse Prompt if needed)  
 // C. Select Attributes  
 // D. WRITE FINAL PROMPT (Cohesive Text)  
   
 const aiResponse = GeminiAPI.generateComplexPrompt(request.rawText, config);  
   
 // 3. Extract Result  
 // The result is no longer a concatenated string of parts,   
 // but a fully written paragraph generated by the AI.  
 let finalPrompt = aiResponse.final\_output.expanded\_prompt\_text;  
  
 return finalPrompt;  
}

## 6. API Endpoints

### POST /api/v1/prompts/expand

* **Input:** JSON body matching PromptRequest class.
* **Process:** Executes the Service Layer logic (Gemini Call).
* **Output:** JSON containing original\_text, expanded\_prompt (the cohesive text), detected\_domain, and processing\_metadata.