**Prompt Engineer – Documentation**

Prompt Engineer is a full-stack application that allows users to input a scenario and constraints and receive a structured AI-generated analysis. The app integrates a React (Material UI) frontend with a Spring Boot backend and uses the Google Gemini API (or optionally OpenAI) for AI responses.

## 🚀 **Live Demo**

Want to see the app in action?  
I’ve deployed the full application—frontend + backend—on Heroku.  
👉 [Click here to try the live demo](https://prompt-engineering-c02ed45f9b1e.herokuapp.com/)

No setup required. Just enter a scenario and constraints to see AI-driven insights instantly!

**Delivery Instructions**

* **Repository Setup**:  
  Code is checked into a GitHub repository. The repository includes both the **React frontend** and **Spring Boot backend** in a single monorepo structure.
* 🔗 [GitHub Repo Link](https://github.com/baranirajendran/prompt-engineering)

**Repository Structure**

prompt-analyzer/

├── src/

│ ├── main/

│ │ ├── frontend/ → React Frontend

│ │ └── java/com/example/demo/ → Spring Boot Backend

├── target/ → Ignored Maven build output

├── .gitignore

├── pom.xml → Maven config with frontend-maven-plugin

├── README.md

**Running the Project**

**Prerequisites**

* Java 17+
* Node.js (>= 22.2.0 recommended)
* Maven 3.3.1+
* Git

### **Clone the Repository**

git clone <https://github.com/YOUR_USERNAME/prompt-engineering.git>

choose main branch

cd prompt-engineering

Setup Instructions

1. Configure Gemini API Key

Create or edit the `src/main/resources/application.properties` file and replace:

```properties

gemini.api.key=REPLACE\_WITH\_YOUR\_GEMINI\_API\_KEY

**Backend – Spring Boot (With Auto Frontend Build)**The project is configured with the [frontend-maven-plugin](https://github.com/eirslett/frontend-maven-plugin), so **the React frontend will be automatically built and bundled with the Spring Boot backend** during the Maven build process. No need to build the frontend manually.

**To Build and Run:**

**mvn clean install**

**mvn spring-boot:run**

The Maven build will:

* Install Node and NPM (locally)
* Run npm install and npm run build inside src/main/frontend
* Copy the built React app into src/main/resources/static
* Package everything into a Spring Boot JAR

Once started, the application is accessible at:

[**http://localhost:8080**](http://localhost:8080)

The main backend endpoint remains:

**<http://localhost:8080/api/analyze-scenario>**

**Screenshots**

## If validation fails, appropriate error messages will be shown.

✅ Successful Prompt SubmissionDisplays the prompt entered, selected constraints, and structured AI-generated recommendations:

A screenshot of a computer

AI-generated content may be incorrect.

🔴 Validation: Scenario Too Short

Displays an error if the scenario is less than 9 words:

A screenshot of a computer

AI-generated content may be incorrect.

🔴 Validation: Scenario Too Long

If the scenario exceeds 1000 characters, an error is shown:

A screenshot of a computer

AI-generated content may be incorrect.

🔴 Validation: Less than 2 Constraints

User must provide at least 2 constraints:

A screenshot of a computer

AI-generated content may be incorrect.

🔄 Audience Level Dropdown

Supports “Beginner”, “Intermediate”, and “Expert” audience levels:

A screenshot of a computer

AI-generated content may be incorrect.

**Request & Response Capture:**

A screenshot from the browser's Network tab showing the full request payload sent to the backend and the structured AI-generated response, confirming successful end-to-end data flow.

A screenshot of a computer

AI-generated content may be incorrect.

**AI Integration (Gemini/OpenAI)**

**Default: Google Gemini API**

Set your API key in:

src/main/resources/application.properties

gemini.api.key=REPLACE\_WITH\_YOUR\_KEY

**To use OpenAI instead:**

1. In DemoApplication.java, change:

@EnableConfigurationProperties(GeminiConfig.class)

to:

@EnableConfigurationProperties(OpenAiConfig.class)

1. In AiService.java:
   * **Uncomment** the OpenAI implementation
   * **Comment out** the Gemini-related code from:

@Autowired

private GeminiConfig geminiConfig;

up to before:

private String buildPrompt(...)

1. Replace your OpenAI API key in application.properties:

openai.api.key=REPLACE\_WITH\_YOUR\_OPENAI\_KEY

**Test Cases**

* 10 JUnit tests are defined in DemoApplicationTests.java.
* Run them using:
  + mvn test

**Form Validation (Frontend)**

* **Minimum 2 constraints**
* **Scenario** must have **at least 9 words** and **< 1000 characters**
* **Audience Level** options:
  + Basic
  + Intermediate
  + Expert

**API Call Details**

The AiService.java class is responsible for interacting with an external AI API (currently Google Gemini, optionally OpenAI) to generate structured responses based on user prompts. The service builds a prompt, makes the API call, and parses the structured response into a Java object.

**1. Class Overview**

@Service

public class AiService {

...

}

This is a @Service class managed by Spring that contains all logic related to calling the AI API.

**2. API Integration Options**

This service is configurable to use either:

* ✅ **Google Gemini** *(default)*
* 🔁 \**OpenAI GPT (commented block ready)*

To switch between them:

* Change @EnableConfigurationProperties(GeminiConfig.class) to OpenAiConfig.class
* Comment the GeminiConfig bean in this file
* Uncomment the OpenAI-related code

**3. Prompt Construction**

**private String buildPrompt(ScenarioAnalysisRequest request)**

This method builds a rich, **structured prompt** from the input ScenarioAnalysisRequest. It includes:

* Executive Summary
* Risks & Pitfalls
* Strategic Recommendations (with steps, indicators, difficulty)
* Tools & Resources
* Limitations & Ethics
* Implementation Timeline
* ROI Projection
* Alternative Approaches

It also injects the **user’s scenario and constraints** and appends a tone:

“Adapt your response for AUDIENCE LEVEL: Intermediate with TONE: Practical guidance.”

To change this tone or sections, **modify this method**.

**4. Making the Gemini API Call**

**String url = "https://generativelanguage.googleapis.com/v1beta/models/{model}:generateContent?key={API\_KEY}";**

A POST request is sent with a structured JSON body:

**{ "contents": [**

**{ "parts": [**

**{ "text": ".... the full prompt text ...." }**

**] } ] }**

The API returns a JSON response, and the app extracts the relevant text output.

**5. Parsing the Response**

**ScenarioAnalysisResponse parseApiResponse(String content)**

The raw AI response is split into sections using Markdown-like patterns (e.g., \*\*1. Executive Summary:\*\*), and populated into this structured object:

ScenarioAnalysisResponse {

String scenarioSummary;

List<String> potentialPitfalls;

String proposedStrategies; // Full Markdown block

Map<String, List<String>> recommendedResourcesStructured;

String disclaimer;

}

The logic uses:

* extractBulletPoints() for lists
* extractStructuredResources() for key-value-like breakdowns (e.g., Tools, Knowledge)

**6. Example Output Format**

Example of what the AI would return for **"Strategic Recommendations"**:

\*\*Pitfall 1: Scope Creep\*\*

\* \*\*Strategy:\*\* Implement a rigorous change request workflow.

\* \*\*Immediate Steps:\*\* Define MVP and hold client alignment meeting.

\* \*\*Indicators:\*\* Number of change requests approved/rejected.

\* \*\*Difficulty:\*\* Medium

### **7. Modify Prompt Behavior**

To adjust how the AI behaves, edit:

private String buildPrompt(ScenarioAnalysisRequest request)

You can:

* Add/remove sections (e.g., remove ROI or add "User Personas")
* Change prompt tone (e.g., from “Intermediate/Practical” to “Expert/Technical”)
* Adjust how much context you give the AI

## **Conclusion**

This project demonstrates how AI can assist in structured scenario analysis using a clean, full-stack architecture. With configurable AI providers, built-in validation, and production-ready deployment, the app is both flexible and scalable.