Assignment: Prompt Engineering Avanzado para Middleware Vertical

Técnicas Avanzadas de Prompt Engineering - Nova Solution Systems

---

📋 Información General

Vertical: Middleware & Backend - APIs & Microservices (Spring Boot, Quarkus, Kafka) **Nivel:** Intermedio
Tipo de actividad: Assignment (Tarea) de Moodle con entregables **Duración estimada:** 150-180 minutos
Formato de entrega: Documento PDF + Archivo ZIP con estructura organizada **Calificación:** 0-10 puntos
Puntaje mínimo para aprobar: 8/10

🎯 Objetivo del Assignment

Demostrar dominio de **técnicas avanzadas de Prompt Engineering** (Few-Shot, Chain-of-Thought, JSON Schema, Prompt Chaining, Long Context, Debiasing) aplicadas a escenarios complejos de desarrollo backend/middleware, incluyendo **evaluación con métricas objetivas** y **versionamiento de prompts**.

📖 Contexto del Proyecto

Cliente: Banco del Pacífico (BDP) **Proyecto:** Modernización de Core Bancario con arquitectura de microservicios **Tu rol en Nova:** Senior Backend Engineer en vertical Middleware

Stack tecnológico:

- **Backend:** Spring Boot 3.2 + Java 21 + Quarkus
- **APIs:** REST + GraphQL + gRPC
- **Messaging:** Apache Kafka + RabbitMQ
- **Data:** PostgreSQL + MongoDB + Redis Cache
- **Security:** OAuth2 + JWT + Spring Security
- **Testing:** JUnit 5 + Mockito + Testcontainers + REST Assured
- **Observability:** Micrometer + Prometheus + Grafana + OpenTelemetry
- **Deploy:** Kubernetes + Helm Charts + GitOps (ArgoCD)

Desafíos del proyecto:

- Migrar 120+ endpoints legacy de monolito COBOL/Java EE (12 años de antigüedad)
- Diseñar arquitectura de 25 microservicios con event-driven patterns
- Cumplir con regulaciones bancarias (CNBV, PCI-DSS, SOC2)
- Performance: P99 latency < 200ms para transacciones críticas
- Disponibilidad 99.99% (downtime máximo: 52 minutos/año)
- Procesamiento de 50,000 transacciones por segundo (TPS) en picos

---

📝 Parte 1: Aplicación de Técnicas Avanzadas (2.0 puntos)

Para cada escenario, identifica **2 técnicas avanzadas** que aplicarías y justifica su elección.

Escenario A: Generación de Microservicios con Few-Shot

Necesitas generar 12 microservicios nuevos para el core bancario. Tienes 3 microservicios de referencia ya implementados (AccountService, TransactionService, CustomerService) con estructura completa (controllers, services, repositories, DTOs, exception handling, tests). Necesitas que el LLM genere los microservicios restantes manteniendo consistencia total de arquitectura, patterns, seguridad y testing.

Técnicas a considerar: Few-Shot, JSON Schema, System Instructions

Escenario B: Análisis de Cumplimiento de OpenAPI en 15 Especificaciones

Debes revisar 15 archivos OpenAPI/Swagger de diferentes equipos para asegurar que cumplan con los estándares corporativos de diseño de APIs (naming conventions, versionamiento, error handling, security schemes, rate limiting). Cada especificación tiene ~800-1200 líneas de YAML.

Técnicas a considerar: Long Context, Chain-of-Thought, Debiasing

Formato de entrega Parte 1:

```
ESCENARIO A:
- Técnica 1: [nombre]
  Justificación: [3-4 líneas explicando por qué esta técnica específica es
ideal]
- Técnica 2: [nombre]
  Justificación: [3-4 líneas explicando cómo complementa la técnica 1]
- Riesgo sin estas técnicas: [2-3 líneas]
```

💻 Parte 2: Diseño de Prompts Avanzados con Entregables (5.0 puntos)

Caso 1: Generación de REST Endpoints con Few-Shot + JSON Schema (2.0 puntos)

Objetivo: Generar 3 endpoints REST nuevos usando Few-Shot Prompting con validación OpenAPI.

Endpoints de referencia (ya implementados):

```java
// Endpoint 1: POST /api/v1/accounts - Crear cuenta
@PostMapping("/api/v1/accounts")
@Operation(summary = "Create new account", security =
@SecurityRequirement(name = "bearer-jwt"))
@ApiResponses({
    @ApiResponse(responseCode = "201", description = "Account created"),
    @ApiResponse(responseCode = "400", description = "Invalid input"),
    @ApiResponse(responseCode = "401", description = "Unauthorized")
})
public ResponseEntity<AccountResponse> createAccount(
    @Valid @RequestBody CreateAccountRequest request
) {
```

```
    // Implementación
}

// Endpoint 2: GET /api/v1/transactions/{id} - Obtener transacción
@GetMapping("/api/v1/transactions/{id}")
@Operation(summary = "Get transaction by ID")
@CacheResult(cacheName = "transactions")
public ResponseEntity<TransactionResponse> getTransaction(
    @PathVariable @Pattern(regexp = "^TXN-[0-9]{10}$") String id
) {
    // Implementación
}

// Endpoint 3: PUT /api/v1/customers/{customerId}/status - Actualizar
estado
@PutMapping("/api/v1/customers/{customerId}/status")
@Operation(summary = "Update customer status")
@PreAuthorize("hasRole('ADMIN')")
public ResponseEntity<CustomerResponse> updateStatus(
    @PathVariable UUID customerId,
    @Valid @RequestBody UpdateStatusRequest request
) {
    // Implementación
}
```

Endpoints a generar:

1. **POST /api/v1/transfers** - Crear transferencia entre cuentas (con validación de fondos)
2. **GET /api/v1/accounts/{accountId}/balance** - Consultar saldo con cache
3. **PATCH /api/v1/loans/{loanId}/payment** - Registrar pago de préstamo

Requerimientos técnicos:

- Spring Boot 3.2 + Java 21 records
- OpenAPI 3.0 annotations completas
- Validación con Jakarta Validation (Bean Validation 3.0)
- Security: JWT authentication + role-based authorization
- Exception handling centralizado con @ControllerAdvice
- Unit tests con JUnit 5 + Mockito + REST Assured

Prompt a diseñar:

1. Escribe un prompt usando **Few-Shot** (incluye los 3 endpoints de referencia como ejemplos)
2. Define un **JSON Schema** para la estructura OpenAPI esperada
3. Especifica que el output debe seguir el patrón exacto de los ejemplos

ENTREGABLE 1 (archivo `prompts/caso1-endpoints.md`):

```
# Prompt Generación de Endpoints REST - v1.0

## Contexto
```

```
[Tu prompt completo aquí]

## JSON Schema de OpenAPI
[El schema que definiste para validar la especificación OpenAPI]

## Output del LLM
[Pega aquí los 3 endpoints que generó el LLM]

## Validación OpenAPI
- ✅ /❌ Endpoint Transfers cumple especificación OpenAPI 3.0
- ✅ /❌ Endpoint Balance cumple especificación OpenAPI 3.0
- ✅ /❌ Endpoint Loan Payment cumple especificación OpenAPI 3.0
- ✅ /❌ Annotations de seguridad presentes
- ✅ /❌ Validaciones Jakarta Validation correctas
- ✅ /❌ Exception handling implementado
- ✅ /❌ Unit tests con cobertura >80%

## Métrica: Conformidad OpenAPI
Endpoints válidos según spec OpenAPI: X/3 (X%)
Swagger UI renderiza sin errores: ✅ /❌
```

Caso 2: Análisis de Seguridad en APIs con Long Context + Chain-of-Thought (1.5 puntos)

Objetivo: Analizar 5 especificaciones OpenAPI para validar cumplimiento de seguridad.

Especificaciones a analizar: Se proporcionan 5 especificaciones OpenAPI en el directorio `/openapi-specs/` del curso:

- `payment-service-api.yaml` (Servicio de Pagos - ~150 líneas)
- `loan-service-api.yaml` (Servicio de Préstamos - ~150 líneas)
- `card-service-api.yaml` (Servicio de Tarjetas - ~140 líneas)
- `notification-service-api.yaml` (Servicio de Notificaciones - ~180 líneas)
- `audit-service-api.yaml` (Servicio de Auditoría - ~160 líneas)

Nota: Estas especificaciones tienen diferentes niveles de cumplimiento de seguridad intencionalmente para que practiques el análisis.

Criterios de seguridad (OWASP API Security Top 10 + CNBV):

1. Security schemes definidos (OAuth2/JWT)
2. Todos los endpoints sensibles tienen @SecurityRequirement
3. Rate limiting configurado (X-RateLimit headers)
4. Input validation en todos los request bodies
5. Error responses NO exponen stack traces ni información sensible
6. Versionamiento de API presente (/api/v1, /api/v2)
7. CORS policies restrictivas documentadas

Prompt a diseñar:

1. Usa **Long Context** para cargar las 5 especificaciones simultáneamente

2. Aplica **Chain-of-Thought** para que el LLM muestre su análisis paso a paso:
    - Paso 1: Identificar security schemes declarados
    - Paso 2: Verificar @SecurityRequirement en endpoints críticos
    - Paso 3: Validar presencia de rate limiting
    - Paso 4: Revisar validaciones de input (schemas, patterns)
    - Paso 5: Analizar error responses (no exponen info sensible)
    - Paso 6: Generar recomendaciones de hardening

ENTREGABLE 2 (archivo `prompts/caso2-api-security.md`):

```
# Prompt Análisis de Seguridad APIs - v1.0

## Contexto
[Tu prompt completo con las 5 especificaciones OpenAPI incluidas]

## Razonamiento del LLM (Chain-of-Thought)
[Pega aquí el análisis paso a paso que generó el LLM]

## Tabla de Resultados de Seguridad
| API Spec | Security Scheme | Auth en Endpoints | Rate Limit | Input
Validation | Error Handling | Cumple OWASP |
|----------|-----------------|-------------------|-----------|-------------
-----|-----------------|--------------|
| Payment Service | ✅/❌ | X/Y endpoints | ✅/❌ | X% | ✅/❌ | ✅/❌ |
| Loan Service | ✅/❌ | X/Y endpoints | ✅/❌ | X% | ✅/❌ | ✅/❌ |
| ... | ... | ... | ... | ... | ... | ... |

## Vulnerabilidades Detectadas
[Lista de vulnerabilidades por severidad: CRITICAL, HIGH, MEDIUM, LOW]

## Métricas
- APIs que cumplen 100% criterios: X/5 (X%)
- Endpoints sin autenticación detectados: XX
- Vulnerabilidades CRITICAL: XX
- Vulnerabilidades HIGH: XX
- Score promedio de seguridad: XX/100
```

Caso 3: Diseño de Microservicios con Prompt Chaining (1.5 puntos)

Objetivo: Generar estrategia de descomposición de monolito en microservicios.

Contexto del monolito legacy: Tienes un monolito de gestión de préstamos con 45,000 líneas de código que maneja:

1. Solicitud de préstamos (validaciones de cliente, scoring crediticio)
2. Aprobación de préstamos (workflow multi-nivel: analista → gerente → director)
3. Desembolso de fondos (integración con core bancario)
4. Generación de tabla de amortización
5. Procesamiento de pagos mensuales

6. Cálculo de intereses y penalizaciones
7. Reestructuración de deuda
8. Cancelación anticipada
9. Reportes regulatorios (CNBV)
10. Notificaciones a clientes (email, SMS, push)

Restricciones de negocio:

- Transacciones de desembolso DEBEN ser ACID (no eventual consistency)
- Scoring crediticio requiere consulta a buró externo (latencia ~500ms)
- Tabla de amortización se calcula una vez y se cachea
- Pagos pueden ser parciales (requiere lógica de aplicación compleja)
- Reporte CNBV debe generarse diariamente a las 00:00 con datos consistentes

Prompt a diseñar: Usa **Prompt Chaining** con 4 prompts secuenciales:

Prompt 1: Analizar el monolito y extraer bounded contexts (DDD) → JSON con dominios **Prompt 2:** Diseñar microservicios por dominio (responsabilidades, APIs, data stores) **Prompt 3:** Definir comunicación inter-servicios (REST sync vs Kafka async vs gRPC) **Prompt 4:** Generar estrategia de migración incremental (Strangler Fig Pattern)

ENTREGABLE 3 (archivo `prompts/caso3-microservices.md`):

```
# Prompt Chaining - Microservices Decomposition - v1.0

## Prompt 1: Extracción de Bounded Contexts (DDD)
[Tu prompt 1]

### Output Prompt 1
[JSON con bounded contexts identificados]

## Prompt 2: Diseño de Microservicios
[Tu prompt 2 que usa el output del prompt 1]

### Output Prompt 2
[Arquitectura de microservicios propuesta con responsabilidades]

## Prompt 3: Comunicación Inter-Servicios
[Tu prompt 3 que usa outputs anteriores]

### Output Prompt 3
[Diagrama de comunicación: REST/Kafka/gRPC por caso de uso]

## Prompt 4: Estrategia de Migración
[Tu prompt 4]

### Output Prompt 4
[Plan de migración incremental con fases]

## Validación de Arquitectura
[Evalúa la arquitectura propuesta contra estos criterios:]
```

```
- ✅ /❌ Cada microservicio tiene una sola responsabilidad
- ✅ /❌ No hay dependencias cíclicas entre servicios
- ✅ /❌ Transacciones ACID manejadas correctamente
- ✅ /❌ Eventual consistency manejada donde es aceptable
- ✅ /❌ Fallback strategies para dependencias externas (buró)
- ✅ /❌ Data ownership bien definido (no shared databases)
- ✅ /❌ Estrategia de migración es incremental (low risk)

## Métricas
- Microservicios propuestos: X
- Promedio de responsabilidades por servicio: X (objetivo: 1-3)
- Llamadas síncronas vs asíncronas: XX% / XX%
- Riesgo de migración: LOW/MEDIUM/HIGH
- Complejidad estimada (story points): XXX
```

🔄 Parte 3: Evaluación con Métricas + Versionamiento (3.0 puntos)

3.1 Iteración de Prompt con Métricas (2.0 puntos)

Escenario: Escribiste este prompt para generar un endpoint de transferencia:

```
Genera un endpoint REST para transferencias bancarias en Spring Boot.
Debe validar fondos y retornar la transacción creada.
```

Resultado v1.0 (output del LLM):

```java
@RestController
public class TransferController {

    @PostMapping("/transfer")
    public String transfer(@RequestBody Map<String, Object> data) {
        String from = (String) data.get("from");
        String to = (String) data.get("to");
        double amount = (double) data.get("amount");

        // TODO: validar fondos
        // TODO: crear transacción

        return "Transfer completed";
    }
}
```

Análisis de problemas:

Usa la tabla de métricas para evaluar el endpoint:

Métrica	v1.0 Score	Objetivo	Gap

Métrica	v1.0 Score	Objetivo	Gap
OpenAPI documentation (@Operation)	0%	100%	-100%
Input validation (Jakarta Validation)	0%	100%	-100%
Security (@PreAuthorize, JWT)	0%	100%	-100%
Exception handling (@ControllerAdvice)	0%	100%	-100%
Proper DTOs (no Map<String, Object>)	0%	100%	-100%
Business logic (validar fondos)	0%	100%	-100%
HTTP status codes correctos (201, 400, 409)	0%	100%	-100%
Unit tests (JUnit + Mockito)	0%	100%	-100%
Integration tests (REST Assured)	0%	100%	-100%
Logging/Observability (SLF4J, metrics)	0%	100%	-100%
TOTAL	**0%**	**100%**	**-100%**

Tu tarea:

1. **Refina el prompt iterativamente** hasta alcanzar >95% en todas las métricas
2. **Documenta cada versión** (v1.0, v2.0, v3.0, ...) con los cambios específicos
3. **Usa técnicas avanzadas:** Few-Shot, JSON Schema, System Instructions

ENTREGABLE 4 (archivo `prompts/caso4-iteration.md`):

```
# Iteración de Prompt: Transfer Endpoint

## v1.0 - Prompt Inicial
[Prompt inicial proporcionado arriba]

### Output v1.0
[Código del endpoint]

### Métricas v1.0
[Tabla de métricas con scores]

### Análisis de Gap
[Explica qué falta y por qué falló]

---

## v2.0 - Primera Iteración
[Tu prompt mejorado v2.0]

### Cambios aplicados:
- [Técnica 1 agregada: ej. Few-Shot con ejemplo de endpoint completo]
- [Técnica 2 agregada: ej. JSON Schema de estructura esperada]
- [Especificación agregada: ej. OpenAPI annotations, security, DTOs]
```

```
### Output v2.0
[Código del endpoint mejorado]

### Métricas v2.0
[Tabla de métricas con nuevos scores]

---

## v3.0 - Segunda Iteración (si es necesaria)
[Continúa iterando hasta alcanzar >95% en todas las métricas]

---

## Prompt Final (vX.Y)
[Tu prompt final que alcanza >95% en todas las métricas]

### Output Final
[Código del endpoint final completo]

### Métricas Finales
[Tabla de métricas con scores >95%]

### Validación con Tests

Comandos ejecutados:
- ./mvnw test
- ./mvnw jacoco:report

Coverage Report obtenido:
- Line Coverage: XX% (objetivo: >90%)
- Branch Coverage: XX% (objetivo: >85%)
- Method Coverage: XX% (objetivo: >90%)

## Resumen de Iteraciones

| Versión | Técnicas Aplicadas | Score Total | Iteraciones Necesarias |
|---------|--------------------|-------------|------------------------|
| v1.0 | Ninguna | 0% | - |
| v2.0 | [técnicas] | XX% | - |
| v3.0 | [técnicas] | XX% | - |
| vX.Y | [técnicas] | >95% ✅ | X iteraciones |
```

3.2 Justificación de Técnicas (1.0 punto)

Escribe una justificación técnica de:

1. **Por qué cada técnica avanzada fue necesaria** (Few-Shot, JSON Schema, etc.)
2. **Cómo impactó en las métricas** (antes/después de aplicar cada técnica)
3. **Lecciones aprendidas** sobre prompt engineering avanzado para backend/middleware

Formato: 200-300 palabras

📤 Formato de Entrega

Estructura del archivo ZIP:

```
Assignment_Middleware_Intermedio_[TuNombre]_[Fecha].zip
/
  prompt-engineering-middleware-intermedio/
    - README.md (resumen del assignment)

    - prompts/
      - caso1-endpoints.md
      - caso2-api-security.md
      - caso3-microservices.md
      - caso4-iteration.md (con versiones v1.0, v2.0, v3.0...)

    - outputs/ (outputs de los LLMs)
      - endpoints-generados/
      - analisis-seguridad/
      - arquitectura-microservicios/

    - openapi-specs/ (copiar las 5 especificaciones proporcionadas)
      - payment-service-api.yaml
      - loan-service-api.yaml
      - card-service-api.yaml
      - notification-service-api.yaml
      - audit-service-api.yaml

    - screenshots/ (evidencias de ejecución en Cell CLI)
      - caso1-execution.png
      - caso2-execution.png
      - caso3-execution.png
      - caso4-execution.png

    - documento-final.pdf
```

Nombre del archivo ZIP: `[practica a la que perteneces]_[nombre]_[apellido paterno]_[apellido materno]_[fecha].zip`

Ejemplo: `Middleware_Miguel_Angel_Coronel_Cruz_20250127.zip`

Contenido del PDF:

```
================================================
ASSIGNMENT: PROMPT ENGINEERING AVANZADO - MIDDLEWARE
Técnicas Avanzadas - Nivel Intermedio - Nova
================================================

NOMBRE: [Tu nombre]
VERTICAL: Middleware & Backend
```

```
FRAMEWORK: [Spring Boot / Quarkus / Micronaut]
FECHA: [DD/MM/YYYY]
ARCHIVO ZIP: Assignment_Middleware_Intermedio_[TuNombre]_[Fecha].zip

================================================
PARTE 1: APLICACIÓN DE TÉCNICAS AVANZADAS
================================================
[Tus 2 escenarios con justificaciones]

================================================
PARTE 2: DISEÑO DE PROMPTS AVANZADOS
================================================

CASO 1: ENDPOINTS REST CON FEW-SHOT + JSON SCHEMA
[Resumen + link a prompts/caso1-endpoints.md]
📸 Ver screenshot: screenshots/caso1-execution.png
✅ Métrica OpenAPI conformidad: X%

CASO 2: SEGURIDAD APIs CON LONG CONTEXT + COT
[Resumen + link a prompts/caso2-api-security.md]
📸 Ver screenshot: screenshots/caso2-execution.png
✅ Métrica de cumplimiento OWASP: X%

CASO 3: MICROSERVICIOS CON PROMPT CHAINING
[Resumen + link a prompts/caso3-microservices.md]
📸 Ver screenshot: screenshots/caso3-execution.png
✅ Métrica de calidad arquitectura: X%

================================================
PARTE 3: EVALUACIÓN CON MÉTRICAS
================================================

3.1 ITERACIÓN DE PROMPT
[Resumen + link a prompts/caso4-iteration.md]
📸 Ver screenshot: screenshots/caso4-execution.png
✅ Score final: X% (iteraciones: X)
✅ Coverage: XX% líneas, XX% branches

3.2 JUSTIFICACIÓN DE TÉCNICAS
[Tu justificación de 200-300 palabras]

================================================
```

📊 Criterios de Evaluación (Escala 0-10)

Parte 1: Aplicación de Técnicas Avanzadas (2.0 puntos)

- Escenario A: 1 pts (técnicas + justificación + riesgos)
- Escenario B: 1 pts

Parte 2: Diseño de Prompts Avanzados (5.0 puntos)

- Caso 1 - Endpoints Few-Shot + JSON Schema: 2.0 pts
  - Prompt usa Few-Shot correctamente (0.5)
  - JSON Schema OpenAPI bien definido (0.5)
  - Output cumple spec OpenAPI >80% (0.5)
  - Entregable completo con validación (0.5)
- Caso 2 - Seguridad Long Context + CoT: 1.5 pts
  - Prompt usa Long Context (5 specs) (0.5)
  - Chain-of-Thought visible en análisis (0.5)
  - Métricas de seguridad OWASP calculadas (0.5)
- Caso 3 - Microservicios Prompt Chaining: 1.5 pts
  - 4 prompts encadenados correctamente (0.5)
  - Output de cada paso usa el anterior (0.5)
  - Arquitectura validada con checklist (0.5)

Parte 3: Evaluación con Métricas (3.0 puntos)

- 3.1 Iteración de Prompt: 2.0 pts
  - Documenta ≥3 versiones (0.5)
  - Métricas calculadas por versión (0.5)
  - Score final >95% (0.5)
  - Aplica técnicas avanzadas (0.5)
- 3.2 Justificación: 1.0 pt

CALIFICACIÓN FINAL:

- **Total:** 10.0 puntos
- **Mínimo para aprobar:** 8.0/10

💡 Consejos Avanzados para Middleware Vertical

1. **Few-Shot es clave para microservicios** - 3 servicios de ejemplo generan los 12 restantes con consistencia
2. **JSON Schema garantiza OpenAPI compliance** - Valida spec antes de implementar
3. **Long Context para análisis masivo** - Procesa múltiples specs/contratos simultáneamente
4. **Chain-of-Thought para decisiones arquitectónicas** - Haz que el LLM explique trade-offs
5. **Prompt Chaining para diseño complejo** - Descompón arquitectura en pasos validables
6. **Métricas objetivas** - No confíes solo en "funciona", mide coverage, security, performance
7. **Versionamiento de prompts** - Trata prompts como código: v1.0, v2.0, changelog
8. **System Instructions** - Define el rol una vez: "Eres un experto en Spring Boot + Microservicios + Seguridad"
9. **Testcontainers para validación** - Valida que el código generado funcione con dependencias reales
10. **OpenAPI validation tools** - Usa spectral/openapi-generator para validar specs automáticamente

---

📚 Recursos Permitidos

✅ Material del curso (todos los módulos 01-10) ✅ Documentación oficial (Spring Boot, Quarkus, OpenAPI, Kafka, OWASP) ✅ Especificaciones OpenAPI del directorio `/openapi-specs/` ✅ Tus notas

personales

❌ Usar IA para generar respuestas del assignment ❌ Copiar de compañeros

---

🎯 Diferencias vs Nivel Básico

Aspecto	Básico	Intermedio
Técnicas	5 básicas	10 avanzadas
Entregables	Solo prompts + screenshots	Prompts + outputs validados + métricas
Complejidad	Casos simples	Casos multi-técnica complejos
Evaluación	Cualitativa	**Cuantitativa con métricas**
Versionamiento	No	**Sí (v1.0, v2.0, ...)**
Validación	Manual	**Automatizada (tests, OpenAPI validation)**
Duración	90-120 min	**150-180 min**

Assignment creado para Middleware Vertical - Nivel Intermedio - Nova Solution Systems 2025