Plantilla Universal de Prompt para Xami-CX

Introducción y contexto

Descripción: Explica el propósito del manual y su ámbito de aplicación. Incluye objetivos clave, audiencia y beneficios de usar la plantilla.

Este manual describe en detalle la primera versión de la **Plantilla Universal de Prompt para Xami-CX**, diseñada para facilitar la creación y estandarizar agentes conversacionales en contextos simples y multi-agente.

Objetivos:

- Documentar cada bloque de la plantilla y su propósito.
- Establecer un proceso claro de uso, validación y mantenimiento.
- Asegurar consistencia, gobernanza y alineación con criterios de negocio y auditoría interna.

Audiencia:

- Desarrolladores e ingenieros de prompts.
- Equipos de QA y gobernanza de IA.
- Consultores y responsables de despliegue en Xami-CX.

Visión general de la plantilla

Descripción: Ofrece una visión de alto nivel de la estructura modular de la plantilla, resaltando sus componentes principales y su interacción.

La plantilla aporta un **esqueleto YAML/texto** que define:

- Metadatos esenciales del agente.
- Identidad, personalidad y tono de marca.
- Configuración de la Base de Conocimiento (BdC), ya sea interna o referenciada.
- Definición de flujos conversacionales, guard-rails de seguridad y manejo de PII.
- Mecanismos de ruteo y colaboración en arquitecturas multi-agente.
- Ajustes de observabilidad, memoria de contexto, escalamiento y checklist de despliegue.

Este enfoque modular permite:

- **Reutilizar** bloques comunes (BdC, guard-rails, PII) en múltiples agentes.
- **Herencia** y especialización rápida de sub-agentes colaboradores.
- **Auditoría** automatizada con el Agente Validador (modelo o3) para garantizar calidad.

La estructura completa puede consultarse en el “Xami-CX Agent Prompt Template”.

Estructura detallada de la plantilla

La plantilla aporta un **esqueleto YAML/texto** que define:

- Metadatos del agente.
- Identidad, personalidad y tono.
- Base de conocimiento (externa o embebida).
- Flujos conversacionales, guard-rails y manejo de PII.
- Mecanismos de ruteo y colaboración en arquitecturas multi-agente.
- Observabilidad, memoria, escalamiento y checklist de despliegue.

Este enfoque modular permite:

- **Reutilizar** bloques comunes (BdC, guard-rails, PII).
- **Herencia** y especialización rápida de agentes colaboradores.
- **Auditoría** automatizada con el Agente Validador (o3).

La estructura completa puede consultarse en el “Xami-CX Agent Prompt Template” .

Estructura detallada de la plantilla

A · Arquitectura de agentes

Descripción: Este bloque define la **topología** y las **relaciones** entre agentes dentro de Xami-CX. Permite especificar si tu solución es un agente único, un supervisor que rutea mensajes a sub-agentes o un colaborador que hereda configuraciones de un supervisor.

```
None
agent_type : "single"                # Valores: single |
supervisor | supervisor_router | collaborator
team_name  : "{{TODO: Equipo}}"      # Nombre del equipo o línea
de negocio
# Si agent_type == supervisor_router → añade:
sub_agents  : [...]                    # Lista de IDs de
sub-agentes gestionados
# Si agent_type == collaborator → añade:
supervisor_ref : "ID_Supervisor"    # Referencia al agente
supervisor
inherit_blocks : [...]                 # Bloques heredados del
supervisor
```

B · Rol & Alcance

Descripción: En este bloque defines la **responsabilidad** y el **nivel de autoridad** de cada agente dentro de la solución Xami-CX. Aquí se especifica si el agente actúa de forma autónoma, supervisa otros sub-agentes o ejecuta únicamente tareas asignadas. Además, se

enumeran las **acciones permitidas** para garantizar que el agente no realice operaciones fuera de su alcance.

```
None
role            : "Descripción clara del rol (ej. ‘Agente de atención al cliente’, ‘Supervisor de ruteo’, etc.)"
authority_level : "full|partial|none"  # full: decisiones completas; partial: permisos limitados; none: solo ejecución.
allowed_actions : [
  "leer_BdC",           # consultar base de conocimiento
  "invocar_subagent",    # redirigir a agentes colaboradores
  "hand-off_humano"      # iniciar proceso de escalamiento humano
]
```

C · Protocolo de Colaboración

Descripción: Este bloque define **cómo los agentes intercambian información** y **gestionan la continuidad de la sesión** en arquitecturas de múltiples agentes. Incluye el formato de intercambio de datos entre sesiones, las reglas de enrutamiento interno (si aplica), tiempos de espera y acciones por defecto en caso de que ningún agente responda en el plazo asignado.

```
None
handshake     : "sessionAttributes JSON"    # Formato del envelope de sesión compartido
routing_logic : "ruleset|manual|LLM-router" # Estrategia para decidir ruta (tabla, manual o LLM)
timeout_ms    : 4000                        # Tiempo máximo de espera antes de fallback
fallback      : "Derivar a asesor humano"   # Acción por defecto si no hay respuesta
```

D · Routing Rules

Descripción: Este bloque especifica **las reglas de enrutamiento** que el supervisor-router utiliza para **dirigir intenciones** a sub-agentes especializados. Cada regla indica:

- `intent`: Nombre o expresión regular que identifica la intención del usuario.

- `route_to`: ID exacto del sub-agente que debe manejar esa intención.
- `priority`: Orden de evaluación; valores más bajos tienen mayor prioridad.

Las reglas se procesan en orden ascendente de `priority`. Si ninguna coincide, el supervisor debe ejecutar el `fallback` definido en el **Protocolo de Colaboración**.

```
None
routing_rules:
  - intent   :
^(Consulta_Estatus|Ubicación_Actual|Generar_Reporte)$
    route_to : "SkyBot-Status"
    priority : 1
  - intent   : ^(Incidencia|Escalar_Humano)$
    route_to : "SkyBot-Incident"
    priority : 2
```

E · Herencia de bloques

Descripción: Este bloque indica **qué secciones de la plantilla** son heredadas automáticamente por los **sub-agentes** (collaborators) desde su supervisor, evitando duplicación y asegurando coherencia en comportamiento y políticas. Esto incluye configuraciones de Base de Conocimiento, guard-rails de seguridad y manejo de PII.

```
None
inherit_blocks:
  - "Base de Conocimiento"   # Estructura de datos y assets
compartidos
  - "Guard-rails"            # Directrices de seguridad y rechazo
de inyecciones
  - "Manejo de PII"         # Políticas de consentimiento y
validación de datos personales
```yaml
inherit_blocks: ["Base de Conocimiento","Guard-rails","Manejo de
PII"]
```

0–16. Bloques principales
```

Descripción: A continuación se presenta una tabla detallada con tres columnas que cubren el nombre del bloque, los campos clave y el propósito principal de cada sección.

Bloque	Campos clave	Propósito
0 Metadatos	`agent_name`, `version`, `language`, `author`, `last_updated`, `contact_owner`, `description`, `notes`	Centralizar información de identificación, control de versiones y responsables del agente.
1 Identidad & Personalidad	`tone`, `values`, `catch_phrases`	Definir la voz, estilo y elementos de marca para asegurar coherencia y consistencia en la comunicación.
2 Objetivo de Negocio & Público	`goal`, `audiences`, `key_kpis`	Especificar el propósito comercial, los segmentos de usuario y los indicadores clave de desempeño para guiar el diseño de la conversación.
3 Base de Conocimiento	`bdc_ref`, `bdc_version`, `bdc_note` (o assets embebidos)	Indicar la fuente de datos, versión y notas relevantes para consultas sobre información estructurada o documentos.
4 Guard-rails & Política	Lista de políticas de seguridad, rechazo, fallback	Definir restricciones, mecanismos de seguridad y respuestas en caso de solicitudes inadecuadas o intentos de inyección de prompt.
5 Variables & Placeholders	Sintaxis `{{variable}}`, valores por defecto o `TODO`	Declarar parámetros dinámicos para facilitar la personalización y evitar ambigüedades en valores críticos.
6 Flujos Conversacionales	`flows` → `id`, `trigger_keywords`, `steps`, `exit_condition`	Describir la lógica de diálogo, incluyendo disparadores, pasos detallados y condiciones de finalización de cada flujo.

7 Formato de Respuesta	`max_words`, `markdown`, `allowed_emojis`	Establecer límites de longitud, formato de salida y uso de elementos visuales (emojis) para garantizar claridad y uniformidad.
8 Manejo de PII	`consent_phrase`, `email_regex`, `phone_regex`, `pii_destination`	Configurar el consentimiento, validación y destino seguro de datos personales para cumplir con políticas de privacidad y normativas aplicables.
9 Herramientas / Power-Ups	`tools` → `name`, `description`, `inputs`, `outputs`, `auth`	Detallar integraciones con servicios externos, funciones Lambda o APIs, incluyendo definición de parámetros de entrada y salida.
10 Ejemplos de Interacción	Mensajes de usuario, respuestas de agente (caso feliz y adversarial)	Proporcionar escenarios de uso y pruebas adversariales para validar cobertura de flujos y defensa ante inyecciones.
11 Errores & Fallback	`default_error`, `no_intent`, `db_timeout`	Definir mensajes de error y rutas alternativas para asegurar que el agente maneje excepciones y situaciones imprevistas.
12 Checklist de Despliegue	Lista de ítems por verificar antes de publicar	Garantizar que todos los elementos (BdC, variables, flujos, validaciones) estén configurados y validados antes del lanzamiento.
13 Observabilidad & KPIs	`log_events`, `metrics_targets`	Especificar eventos a registrar y metas de métricas para monitorear la salud operativa y el desempeño del agente en producción.
14 Memoria & Contexto	`turn_window`, `store_session_vars`, `expiration_minutes`	Configurar la gestión de contexto conversacional y la retención de variables de sesión para optimizar la experiencia y uso de tokens.
15 Política de Actualización de BdC	`owner`, `refresh_cycle`, `process`	Documentar el responsable, frecuencia y procedimiento para mantener actualizada la base de conocimiento.

16 Criterios de Escalación Humana	`handoff_conditions`, `handoff_channel`, `handoff_payload`	Definir escenarios y canales de derivación a un operador humano para casos complejos o insatisfactorios.

Guía paso a paso de uso

Descripción: Esta sección muestra **cómo redactar un meta-prompt** para que la IA utilice la plantilla Xami-CX y genere automáticamente el prompt de sistema de tu agente, partiendo de un objetivo funcional.

1. **Define tu objetivo**: redacta en una frase clara la función principal del agente.
   Ejemplo: “Precalificar créditos personales según ingreso y edad del solicitante.”

2. **Prepara tu instrucción**: comienza tu meta-prompt con:

```
None
Usa la **Plantilla Universal Xami-CX** para crear el prompt de
sistema de un agente.
Objetivo funcional: "<tu objetivo>"
```

3. **Incluye la plantilla**: pega a continuación todo el contenido de la plantilla (bloques A–E y 0–16) para que la IA tenga el esqueleto completo.

4. **Especifica ajustes puntuales**: si necesitas valores concretos (p. ej., nombre del agente, versión, idioma), añádelos justo tras la instrucción.

```
None
agent_name: "Agente-Creditos"
version: "v1.0"
language: "es"
```

5. **Solicita la generación**: al final, pide explícitamente:

```
None
Genera el prompt de sistema completo en formato YAML, con todos
los bloques rellenos según el objetivo.
```

6. **Envía a la IA**: usa este meta-prompt como mensaje de usuario en un chat con GPT-4o-mini o 4o.

7. **Revisa la salida**: comprueba que todos los bloques estén completos y coherentes con tu objetivo.

8. **Valida con el Agente Validador**: copia el prompt generado y ejecútalo en un chat con modelo `o3` para detectar issues de lógica, seguridad e inyección.

9. **Aplica correcciones**: integra las recomendaciones del validador para pulir detalles o añadir guard-rails.

10. **Finaliza el prompt**: una vez validado, guarda el prompt como tu configuración de sistema en Xami-CX.

11. **Despliega y prueba**: publica el agente, verifica su comportamiento con ejemplos reales y ajusta métricas de KPIs (bloques 13–16).

Ejemplos de Meta-Prompt

A continuación dos ejemplos de meta-prompts que utilizan la **Plantilla Universal Xami-CX** para generar automáticamente el prompt de sistema:

1. **Agente simple** (CrediBot MX para precalificación de créditos):

```
None
Usa la Plantilla Universal Xami-CX para crear el prompt de
sistema de un agente.
Objetivo funcional: "Precalificar créditos personales según
ingreso y edad del solicitante."

agent_name: "CrediBot MX"
version: "v1.0"
```

```
language: "es"

Genera el prompt de sistema completo en formato YAML, con todos
los bloques A–E y 0–16 rellenados apropiadamente para este
agente.
```

2. **Agente multi-agente** (Supervisor con routing para SkyBot):

```
None
Usa la Plantilla Universal Xami-CX para crear el prompt de
sistema de un supervisor-router.
Objetivo funcional: "Orquestar rutas de consulta de estado e
incidencias para SkyBot-Suite."

agent_type: "supervisor_router"
team_name: "SkyBot-Suite"
sub_agents:
  - "SkyBot-Status"
  - "SkyBot-Incident"
version: "v1.0"
language: "es"

Genera el prompt de sistema completo en formato YAML, incluyendo
bloques A–E y 0–16 con valores adecuados para un supervisor con
routing.
```

Buenas prácticas y patrones

- **Nombres consistentes**: usar snake_case para identificadores.
- **Variables explícitas**: nunca dejar “X días” sin placeholder.
- **Bloques reutilizables**: compartir BdC, guard-rails y PII entre colaboradores.

- **Temperatura y determinismo**: prompts deterministas para auditoría (temp 0–0.2).
- **Control de versiones**: tag de Git y `version` en metadatos.
- **Testing adversarial**: incluir flujos de error y prompt-injection en ejemplos.