GUIA PRÁCTICA 2026

# 7 PROMPTS

## PARA PROGRAMAR MÁS RÁPIDO

PROMPTS LISTOS PARA COPIAR Y PEGAR. DISEÑADOS PARA CUBRIR TODO TU FLUJO DE DESARROLLO: DESDE LA IDEA HASTA LA DOCUMENTACIÓN. SIN TEORÍA. SIN HUMO.

| 7 | 100% | 0 |
|---|---|---|
| PROMPTS | PRÁCTICO | RELLENO |

BIG school

## MAPA DE LA GUÍA


01 **El Arquitecto**
Planificación y diseño

02 **El Constructor**
Generación de código

03 **El Detective**
Debugging

04 **El Critico**
Code review

05 **El Optimizador**
Refactoring

06 **El Escudo**
Testing

07 **El Narrador**
Documentación


## Tu flujo de desarrollo, cubierto

Cada prompt encaja en una frase del ciclo real de construcción de software.

1.Diseñar | 2.Construir | 3.Depurar | 4.Revisar | 5.Optimizar | 6.Testear | 7.Documentar

# 01 El Arquitecto

## Fase: Planificación y Diseño del sistema

Antes de escribir una sola linea de codigo, necesitas pensar la estructura. Este prompt convierte una idea vaga en un plan tecnico con stack, estructura de archivos, modelo de datos y decisiones justificadas. Usalo al inicio de cada proyecto o funcionalidad nueva.

ARQUITECTURA | STACK TÉCNICO | MODELO DE DATOS | ESTRUCTURA

>_PROMPT LISTO PARA USAR

Actua como un arquitecto de software senior con mas de 15 anos de experiencia disenando sistemas escalables.

Necesito que diseñes la arquitectura tecnica completa para el siguiente proyecto:

**## Proyecto**
[Describe tu proyecto en 2-3 frases. Ej: "Una app web donde los usuarios pueden crear y compartir listas de lectura personalizadas, con recomendaciones basadas en sus gustos."]

**## Requisitos clave**
- Usuarios esperados: [numero estimado]
- Tipo de aplicacion: [web / movil / API / CLI / etc.]
- Restricciones tecnicas: [lenguaje o framework obligatorio, si aplica, o "Ninguna, recomienda tu"]

**## Lo que necesito que entregues**

1.Stack tecnologico recomendado: frontend, backend, base de datos, infraestructura. Justifica cada eleccion en una linea.

2.Estructura de carpetas del proyecto: muestra el arbol de archivos inicial.

3.Modelo de datos: entidades principales, sus campos clave y relaciones.

4.Diagrama de flujo: describe paso a paso el flujo principal del usuario (de la forma: Paso 1 -> Paso 2 -> ...).

5.Decisiones de diseño: lista las 3-5 decisiones arquitectónicas más importantes que has tomado y por que.

6.Riesgos tecnicos: identifica 2-3 posibles problemas y como mitigarlos.

**Consejo**: Cuanto mas contexto des en la descripcion del proyecto, mejor sera la arquitectura. Incluye el tipo de usuario, el problema que resuelves, y cualquier integración externa que necesites.

02

# El Constructor

## Fase: Generación de Código funcional

Cuando ya sabes que construir, necesitas codigo que funcione a la primera. Este prompt genera codigo limpio, modular y con manejo de errores incluido. No pide un ejemplo: pide produccion.

CÓDIGO | IMPLEMENTACIÓN | PRODUCCIÓN | BUENAS PRÁCTICAS

>_PROMPT LISTO PARA USAR

Actua como un desarrollador senior especializado en [tu lenguaje/framework. Ej: "TypeScript con Next.js y Prisma"].

Necesito que implementes lo siguiente:

**## Funcionalidad**
[Describe exactamente que debe hacer el codigo. Ej: "Un endpoint POST /api/auth/register que reciba email y password, valide los campos, hashee la contraseña con bcrypt, guarde el usuario en PostgreSQL y devuelva un JWT."]

**## Contexto técnico**
- Stack: [lenguaje, framework, ORM, etc.]
- Base de datos: [tipo y si ya existe el schema]
- Convenciones del proyecto: [nombrado de archivos, estilo de codigo, o "Usa convenciones estandar"]

**## Requisitos del código**

1.Código listo para produccion, no ejemplos simplificados.

2.Incluir validacion de inputs y manejo de errores completo.

3.Aplicar el principio de responsabilidad única.

4.Anadir tipos/interfaces cuando el lenguaje lo permita.

5.Incluir comentarios SOLO donde la logica no sea obvia.

6.Si necesitas alguna dependencia externa, indícala al inicio con el comando de instalación.

**## Formato de entrega**
Entrega el código en bloques separados por archivo. Indica la ruta de cada archivo como encabezado. Al final, incluye una sección breve "Como probarlo" con los pasos exactos.

**Consejo**: La clave es la especificidad. "Haz un login" produce codigo genérico. "Endpoint POST que reciba email/password, valide formato, hashee con bcrypt..." produce código que puedes usar directamente.

# 03 El Detective

## Fase: Debugging y Resolución de errores

Los bugs son inevitables. Lo que no es inevitable es pasar 3 horas buscando la causa. Este prompt usa razonamiento paso a paso (Chain of Thought) para forzar un análisis metódico: hipótesis, análisis, causa raiz y solución.

DEBUGGING | ERRORES | CHAIN OF THOUGHT | DIAGNÓSTICO

>_PROMPT LISTO PARA USAR

Actua como un debugger experto. Necesito que analices un problema en mi código de forma metódica.

**## El problema**
- Qué deberia pasar: [Comportamiento esperado. Ej: "El formulario deberia enviar los datos y redirigir a /dashboard."]
- Qué pasa en realidad: [Comportamiento actual. Ej: "La página se recarga pero no redirige. No aparece error en consola."]
- Mensaje de error (si hay): [Pega el error exacto o "No hay error visible"]
- Cuándo ocurre: [Siempre / solo a veces / solo en produccion / etc.]

**## Código relevante**
```
[Pega aquí el código donde crees que esta el problema]
```

**## Contexto adicional**
- Lenguaje/framework: [ej: React 19, Node 22]
- Que ya intente: [Lista lo que probaste. Esto evita sugerencias que ya descartaste]

**## Cómo quiero que respondas**
Sigue este proceso exacto:

1.Hipótesis inicial: Lista 3 causas posibles ordenadas por probabilidad.

2.Analisis linea por linea: Revisa el codigo y senala exactamente donde podría estar el fallo.

3.Causa raiz: Identifica la causa más probable y explica POR QUÉ provoca el comportamiento observado.

4.Solución: Muestra el código corregido con los cambios resaltados.

5.Prevención: Sugiere una práctica o patron para evitar este tipo de error en el futuro.

**Consejo**: Incluir "que ya intente" es clave. Sin eso, la IA te sugerirá lo primero y más obvio (que probablemente ya probaste). Con eso, va directo a causas menos evidentes.

# 04 El Crítico

## Fase: Revisión de Código

Tener un segundo par de ojos siempre mejora el código. Este prompt simula un code review riguroso: seguridad, rendimiento, mantenibilidad y puntuación. Es como tener un compañero senior disponible 24/7.

CODE REVIEW | SEGURIDAD | MANTENIBILIDAD | CALIDAD

>_PROMPT LISTO PARA USAR

Actúa como un code reviewer senior exigente pero constructivo. Revisa el siguiente código como si fuera un *Pull Request* en un equipo profesional.

**## Código relevante**

```
[Pega aquí tu código]
```

**## Contexto**

- Lenguaje/framework: [ej: Python con FastAPI]
- Que hace este código: [breve descripción de la funcionalidad]
- Es código de: [API / frontend / servicio / script / etc.]

**## Analiza cada una de estas dimensiones**

1.Seguridad: Hay vulnerabilidades? (inyección SQL, XSS, exposición de datos sensibles, secrets hardcodeados, etc.)

2.Rendimiento: Hay cuellos de botella? (queries N+1, operaciones O(n2), cargas innecesarias...)

3.Código limpio: Cumple con el principio de responsabilidad única? Los nombres son descriptivos? Hay duplicación?

4.Patrones y estructura: Usa patrones adecuados? La estructura es coherente con el framework?

5.Manejo de errores: Gestiona los edge cases? Los errores se manejan o se tragan silenciosamente?

**## Formato de respuesta**

Para cada dimensión:
- Estado: Bien / Mejorable / Problema
- Si hay problema: explica que, donde y muestra el código corregido.

Al final, da una puntuación global de 1 a 10 con un resumen de una linea.
Cierra con los 3 cambios de mayor impacto que harías si solo pudieras cambiar 3 cosas.

**Consejo**: Úsalo también con código que crees que "ya esta bien". Los problemas de seguridad y rendimiento rara vez son obvios para quien escribio el código.

05

# El Optimizador

## Fase: Refactoring y Rendimiento

Código que funciona no es lo mismo que buen código. Este prompt toma código funcional y lo transforma en código que es mas rápido, mas legible y mas fácil de mantener, sin romper nada.

REFACTORING | RENDIMIENTO | CLEAN CODE | ESCALABILIDAD

>_PROMPT LISTO PARA USAR

Actua como un ingeniero de rendimiento y clean code especializado en [tu lenguaje/framework].

Necesito que refactorices el siguiente código. Funciona, pero necesita mejorar.

**## Código actual**
```
[Pega aquí el código que funciona pero quieres mejorar]
```

**## Qué hace este código**
[Explica brevemente la funcionalidad]

**## Qué me preocupa**
[Elige uno o varios: "Es lento" / "Es difícil de leer" / "Es difícil de extender" / "No estoy seguro de que escale" / "Hay mucha duplicación"]

**## Reglas de refactor**

1.No cambiar el comportamiento externo. La entrada y salida deben seguir siendo exactamente las mismas.

2.Explicar cada cambio. No quiero código nuevo sin entender que cambiaste y por qué.

3.Mostrar el antes y después de cada bloque modificado.

**## Entrega**
- Código refactorizado completo.
- Tabla de cambios con 3 columnas: Que cambie | Por que | Impacto esperado.
- Si hay mejora de rendimiento, estima la complejidad antes (Ej: O(n2)) y después (Ej: O(n log n)).
- Si algún cambio introduce una dependencia nueva o un patrón diferente, justifícalo.

**Consejo**: La regla de oro del refactoring con IA: no aceptes cambios que no entiendas. La tabla "Qué cambie / Por qué / Impacto" esta ahí para eso. Si un cambio no tiene una justificación clara, descártalo.

# 06 El Escudo

## Fase: Testing y Cobertura

Escribir tests es la tarea que todos saben que deben hacer y nadie quiere hacer. Este prompt genera tests completos: camino feliz, edge cases, errores y mocks, listos para ejecutar.

TESTING | UNIT TEST | EDGE CASES | COBERTURA

>_PROMPT LISTO PARA USAR

Actua como un ingeniero de QA senior especializado en testing automatizado con [framework de tests. Ej: "Jest y React Testing Library" / "Pytest" / "JUnit" / "Vitest"].

Necesito que escribas una suite de tests completa para el siguiente código:

**## Código a testear**

```
[Pega aquí la función, modulo o endpoint que quieres testear]
```

**## Qué hace este código**
[Descripción breve de la funcionalidad]

**## Dependencias externas**
[Llama a una API? Usa base de datos? Depende de algún servicio? Lista lo que necesita mocking]

**## Requisitos de los tests**

Cubre estas 4 categorias obligatoriamente:

1.Happy path: El flujo normal funciona como se espera (minimo 2 tests).

2.Edge cases: Inputs vacíos, nulos, valores extremos, tipos incorrectos, caracteres especiales (mínimo 3 tests).

3.Gestión de errores: El código falla de forma controlada cuando debe fallar (mínimo 2 tests).

4.Integraciones: Mock de las dependencias externas, verificando que se llaman con los parametros correctos (si aplica).

**## Formato de entrega**
- Cada test debe tener un nombre descriptivo que explique que verifica (ej: "deberia_retornar_error_si_email_es_vacio").
- Agrupa los tests por categoria usando describe/context blocks.
- Incluye los mocks/fixtures necesarios.
- Al final, anade una lista resumen de todos los escenarios cubiertos.

**Consejo**: Los edge cases son donde mas valor aporta la IA. Tu probablemente testearias el happy path y un par de errores. La IA te forzará a pensar en inputs que no esperabas: strings de 10.000 carácteres, arrays vacios, tipos undefined, etc.

07

# El Narrador

## Fase: Documentación técnica

Documentar es lo que separa un proyecto mantenible de una bomba de tiempo. Este prompt genera documentación técnica clara que tu yo del futuro (y tu equipo) agradeceran. README, docstrings, guía de uso, todo incluido.

DOCUMENTACIÓN | README | API DOCS | ONBOARDING

>_PROMPT LISTO PARA USAR

Actua como un technical writer senior con experiencia en documentación de proyectos open source.
Necesito que generes documentación técnica completa para el siguiente código/proyecto:

**## Código / Proyecto**
```
[Pega el código, o describe el proyecto y sus módulos principales si es demasiado largo]
```

**## Contexto**
- Nombre del proyecto: [nombre]
- Stack: [lenguajes, frameworks, herramientas]
- Quien va a leer esto: [Ej: "Desarrolladores del equipo" / "Colaboradores open source" / "Yo mismo en 6 meses"]

**## Genera los siguientes documentos**

**### 1. README.md**
- Descripcion del proyecto (que problema resuelve, en 2-3 frases).
- Requisitos previos y versiones.
- Instalacion paso a paso (que funcione copiando y pegando).
- Ejemplo de uso rapido.
- Estructura del proyecto (arbol de archivos con descripcion de cada carpeta clave).
- Variables de entorno necesarias (tabla con nombre, descripcion, ejemplo, si es obligatoria).
- Como ejecutar los tests.
- Como contribuir (si aplica).

**### 2. Documentacion inline**
- Docstrings/JSDoc para cada función pública: que hace, parametros (nombre, tipo, descripcion), retorno, excepciones posibles, y un ejemplo de uso.

**### 3. Guia de API (si aplica)**
- Para cada endpoint: método, ruta, descripción, parametros, body de ejemplo, respuesta exitosa de ejemplo, posibles errores.

Escribe en un tono directo y técnico. Nada de frases de relleno. Ve al grano.

**Consejo**: La documentación mas útil es la que responde a la pregunta "Como lo arranco?" en menos de 30 segundos. Asegúrate de que la sección de instalación del README funcione literalmente con copiar y pegar.

# ESTOS 7 PROMPTS
## CUBREN EL CICLO COMPLETO DE DESARROLO

**NO** son fórmulas mágicas.

Son **puntos de partida** que:

→ Eliminan bloqueos
→ Aceleran decisiones
→ Producen código

**ADÁPTALOS A TU STACK.**
**ITERA SOBRE LOS RESULTADOS.**
**Y CONSTRUYE.**

# APRENDE MUCHO MÁS EN EL CURSO DE DESARROLLO CON IA

**NOS VEMOS AQUÍ PARA LA 1ª CLASE**

BIG school