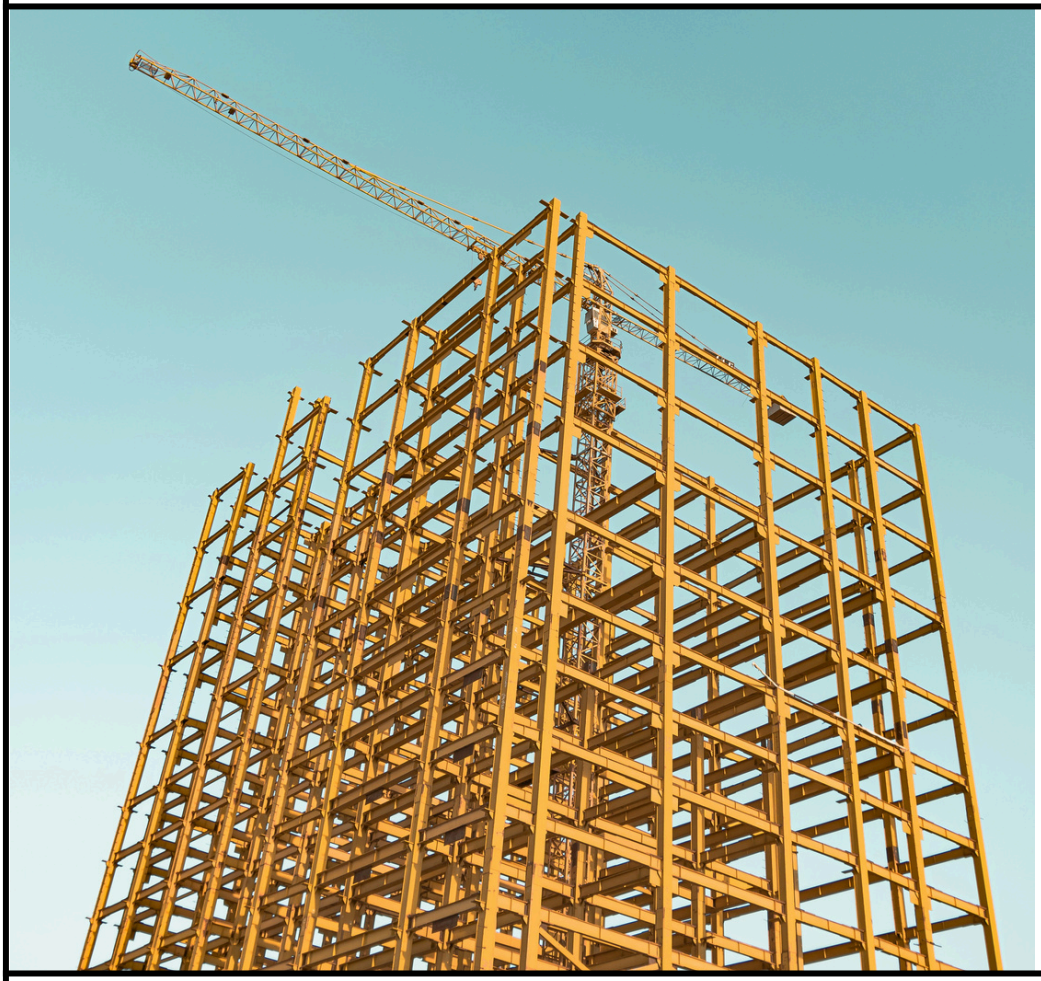
Incubator

WHITEPAPER PARA EMPRENDEDORES



Hallazgos clave:

- El 80% de la frustración con IA viene de expectativas mal comunicadas
- Definir el contexto correcto puede reducir hasta 70% el tiempo de ajustes
- No todos los proyectos necesitan el mismo estándar de código

EL FRAMEWORK INVISIBLE:

CÓMO DAR CONTEXTO A LA IA PARA CONSTRUIR TU PRODUCTO DIGITAL

RESUMEN EJECUTIVO

La inteligencia artificial está democratizando el desarrollo de software, pero muchos emprendedores están obteniendo resultados mediocres porque no entienden un principio fundamental: la IA no es mágica, es un albañil experto que necesita planos claros.
Este whitepaper presenta un framework práctico para que emprendedores sin experiencia técnica profunda puedan guiar efectivamente a herramientas como Lovable, V0, Cursor o Bolt para construir productos digitales de calidad.

1. EL PROBLEMA: LA ANALOGÍA DEL ARQUITECTO Y EL ALBAÑIL

1.1 Por qué la IA "falla"

Imagina contratar al mejor albañil del mundo y decirle: "Construye una casa". Sin planos, sin especificaciones, sin saber si es para una familia o un hotel.

¿El resultado? Probablemente algo funcional, pero no lo que necesitabas.

Esto es exactamente lo que pasa cuando le pides a la IA:

- "Crea una landing page para mi startup"
- "Hazme un dashboard"
- "Construye un sistema de reservas"

1.2 Lo que realmente necesita la IA

La IA necesita tres tipos de información:

1. Contexto de negocio: ¿Para qué es esto? ¿Quién lo usará?
2. Reglas técnicas: ¿Cómo debe estar construido?
3. Criterios de éxito: ¿Qué significa "bien hecho"?

2. EL FRAMEWORK DE TRES CAPAS

Capa 1: Contexto de Negocio (EL QUÉ)

Antes de escribir una línea de código, define:

a) Propósito del producto

```
Ejemplo malo: "Una app de delivery"
Ejemplo bueno: "Una app de delivery para
restaurantes locales en
barrios residenciales, donde los clientes son +40
años y prefieren
interfaz simple sobre funcionalidades complejas"
```

b) Usuario principal

- ¿Quién es? (edad, contexto, habilidades tecnológicas)
- ¿Qué problema tiene?
- ¿En qué momento del día lo usará?

c) Escala esperada

- ¿Es un MVP para validar o un producto para escalar?
- ¿Cuántos usuarios simultáneos esperas?
- ¿Necesita estar disponible 24/7?

Template práctico:

```
Este es un [tipo de producto] para [usuario
específico] que necesita
[resolver qué problema]. Se usará principalmente
[cuándo/dónde] y el
éxito se mide por [métrica clara]. Es un
[MVP/Producto escalable] que
debe estar listo en [timeframe].
```

2. EL FRAMEWORK DE TRES CAPAS

Capa 2: Reglas Técnicas (EL CÓMO)

Incluso en herramientas como Lovable donde no puedes crear archivos de reglas formales, puedes establecer estándares en cada prompt:

a) Stack tecnológico

```
"Usa React con TypeScript, Tailwind CSS para
estilos, y Supabase
para base de datos. Prefiere componentes
funcionales con hooks."
```

b) Estructura de código

```
"Organiza el código en carpetas: /components,
/pages, /utils, /hooks.
Cada componente debe tener máximo 200 líneas. Si
es más grande,
divídelo en subcomponentes."
```

c) Convenciones de nombres

```
"Componentes en PascalCase, funciones en
camelCase, constantes en
UPPER_SNAKE_CASE. Los archivos deben tener nombres
descriptivos que
expliquen su función."
```

d) Manejo de errores

```
"Todo fetch debe tener try-catch. Muestra mensajes
de error amigables
al usuario, nunca errores técnicos. Los loading
states son obligatorios."
```

e) Accesibilidad básica

```
"Todos los botones deben tener labels claros. Los
colores deben tener
contraste suficiente. Las imágenes necesitan alt
text descriptivo."
```

Capa 3: Criterios de Calidad (EL CUÁNTO)

Define tu "trade-off" según el contexto:

Para MVPs rápidos (2-4 semanas):

- ✓ Funciona y resuelve el problema core
- ✓ UX intuitiva
- ✓ Responsive básico
- (!) Código puede no ser perfecto
- (!) Optimización mínima

Para productos escalables (2-6 meses):

- ✓ Todo lo anterior +
- ✓ Código limpio y mantenible
- ✓ Testing básico
- ✓ Performance optimizado
- ✓ Seguridad robusta

Template de criterios:

```
Este proyecto es [MVP rápido/Producto escalable].
Prioridad 1: [ej: Velocidad de desarrollo]
Prioridad 2: [ej: UX simple]
Prioridad 3: [ej: Código limpio]

Podemos sacrificar: [ej: Optimización perfecta]
No podemos sacrificar: [ej: Seguridad de datos de
usuario]
```

3. APLICACIÓN PRÁCTICA POR HERRAMIENTA

3.1 Lovable / V0 (Herramientas sin archivos de reglas)

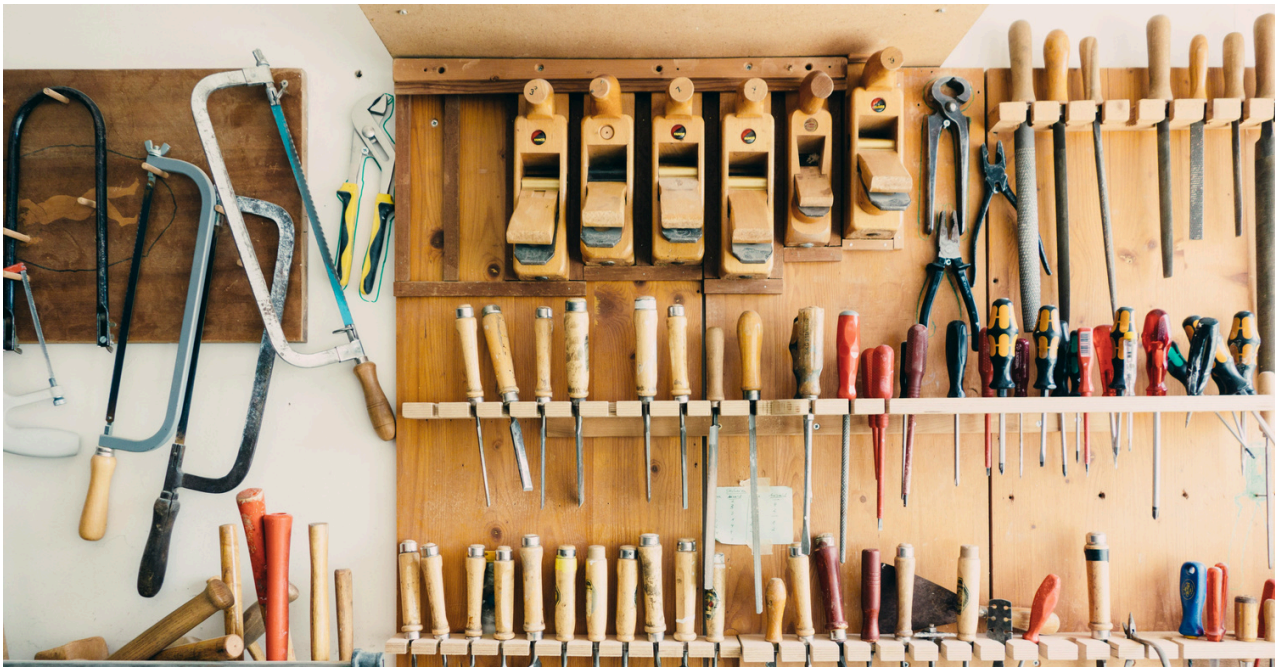
Estrategia: Contexto en cada conversación
Crea un "prompt master" que uses al inicio de cada sesión:

```
# CONTEXTO DEL PROYECTO
[Pega tu contexto de negocio de la Capa 1]

# REGLAS TÉCNICAS
Siempre que escribas código para este proyecto:
1. [Lista tus reglas de la Capa 2]
2. [...]
3. [...]

# CALIDAD ESPERADA
Este es un [MVP/Producto escalable].
Prioridades: [...]
```

Tip: Guarda este prompt en un documento y pégalo al inicio de cada nueva feature



3. APLICACIÓN PRÁCTICA POR HERRAMIENTA

3.2 Cursor (Herramientas con archivos de reglas)

Estrategia: Archivo .cursorrules
Crea un archivo `.cursorrules` en la raíz de tu proyecto:

```
# Contexto del Proyecto
[Tu Capa 1]

# Stack Tecnológico
- Framework: React 18 con TypeScript
- Estilos: Tailwind CSS
- Estado: Zustand
- Backend: Supabase

# Convenciones de Código
## Componentes
- Usar componentes funcionales con hooks
- Máximo 200 líneas por componente
- Props tipar con interfaces TypeScript
[...]

# Prácticas Obligatorias
- Todo fetch debe tener manejo de errores
- Loading states visibles
- Responsive mobile-first
[...]
```

3.3 Iteración y Refinamiento

El framework no es estático:

1. **Primera semana:** Define tu framework básico
2. **Semanas 2-4:** Observa qué errores se repiten
3. **Ajusta las reglas:** Agrega especificaciones para esos casos

Documenta excepciones: Si algo debe hacerse diferente, especifícalo

3. APLICACIÓN PRÁCTICA POR HERRAMIENTA

Ejemplo de evolución:

```
Regla inicial: "Usa Tailwind para estilos"

Después de ver problemas:
"Usa Tailwind para estilos. Para botones, usa
siempre estas clases:
- Primario: bg-blue-600 hover:bg-blue-700 text-
white px-4 py-2 rounded
- Secundario: bg-gray-200 hover:bg-gray-300
text-gray-800 px-4 py-2 rounded
Nunca uses inline styles ni estilos custom en
componentes individuales."
```

4. CASOS DE USO REALES

Caso A: Marketplace de Servicios Locales (MVP en 3 semanas)

Contexto definido:

- Plataforma para conectar prestadores de servicios del hogar con clientes
- Usuario principal: 35-55 años, habilidad tecnológica media
- MVP para validar demanda en un barrio específico
- Éxito: 50 servicios publicados en primer mes

Framework aplicado:

- Stack simple: Next.js + Supabase
- UX inspirada en Mercado Libre (familiar para usuarios)
- Código "suficientemente bueno" - prioridad a velocidad
- Sin optimizaciones complejas

Resultado:

- Lanzamiento en 19 días
- 15% del código necesitó ajustes manuales
- Producto validado, iterando en v2

Caso B: Sistema de Gestión para Clínica Médica (6 meses)

Contexto definido:

- Sistema para gestionar turnos, historias clínicas, facturación
- Usuarios: personal médico-administrativo
- Debe manejar datos sensibles (HIPAA compliance)
- Disponibilidad crítica 8am-8pm

Framework aplicado:

- Stack robusto: Next.js + PostgreSQL + Redis
- Testing obligatorio (unit + integration)
- Logging extensivo
- Code review manual de todo componente

Resultado:

- 40% más tiempo de desarrollo que un MVP
- Código production-ready desde el inicio
- Mantenimiento mínimo post-lanzamiento

5. ERRORES COMUNES Y CÓMO EVITARLOS

Error 1: Prompt único perfecto
☒ Escribir todo el proyecto en un solo prompt gigante
✓ Construir iterativamente, feature por feature, con contexto consistente

Error 2: Purismo prematuro
☒ El código debe ser perfecto desde el día 1
✓ Define tu trade-off según la etapa del proyecto

Error 3: Dejar que la IA decida todo
☒ Usa las mejores prácticas (muy vago)
✓ Usa React Query para fetching con stale time de 5 minutos (específico)

Error 4: No documentar aprendizajes
☒ Resolver el mismo problema 5 veces
✓ Agregar la solución a tus reglas para que no se repita

Error 5: Cambiar de herramienta por frustración
☒ Lovable no funciona, voy a probar Cursor
✓ Voy a refinar mi framework en Lovable antes de cambiar

6. CHECKLIST PARA EMPEZAR HOY

Antes de escribir código:

- [ ] He definido claramente QUÉ estoy construyendo y PARA QUIÉN
- [ ] Tengo claro mi trade-off: ¿MVP rápido o producto escalable?
- [ ] He listado mis 5-10 reglas técnicas no negociables
- [ ] Tengo ejemplos de "bien hecho" vs "mal hecho" para mi proyecto
- [ ] He creado mi "prompt master" o archivo de reglas

Durante el desarrollo:

- [ ] Inicio cada sesión pegando mi contexto
- [ ] Cuando la IA se desvía, no la culpo - refino mis instrucciones
- [ ] Documento patrones que funcionan en mis reglas
- [ ] Reviso que cada feature cumpla mis criterios antes de continuar

Después de cada sprint:

- [ ] ¿Qué tipo de errores se repitieron?
- [ ] ¿Qué regla debo agregar para evitarlos?
- [ ] ¿Alguna regla actual es demasiado restrictiva?
- [ ] ¿Mi framework sigue siendo relevante o debo ajustarlo?

7. LA EVOLUCIÓN DE LAS HERRAMIENTAS

Hoy (2025):

- Herramientas como Lovable lanzan features constantemente
- Lo que hoy no se puede hacer, en 2-3 meses estará disponible
- La capacidad de "moldear" la IA mejora exponencialmente

Perspectiva:

- Si hoy ajustas 20% del código → En 6 meses será 10% → En 1 año será 2%
- Las reglas que defines hoy te servirán en cualquier herramienta futura
- Aprender a "hablar" con la IA es la skill, no la herramienta específica

Recomendación:
No esperes la "herramienta perfecta". Empieza con lo que tienes, construye tu framework, y evoluciona con las herramientas.

8. CONCLUSIÓN: EL NUEVO LENGUAJE DEL EMPRENDEDOR TÉCNICO

Hace 10 años, los emprendedores necesitaban aprender a "hablar con developers". Hoy, necesitan aprender a "hablar con IA".
La buena noticia: Es más fácil iterar con IA que con humanos.
La mejor noticia: El framework que construyes te pertenece. Puedes:

- Aplicarlo en cualquier herramienta
- Pasárselo a un developer si escalas el equipo
- Usarlo como documentación técnica viva
- Mejorarlo constantemente sin "reentrenar" a nadie

El cambio de mindset:
De: "La IA tiene que adivinar qué quiero"
A: "Yo soy el arquitecto, la IA es mi mejor albañil"

AUTORES

Sobre los autores:

Nicolas, CEO de Incubator

Marcos, CTO de Incubator

¿Preguntas? Contacta a ana.massacane@incubator.com.ar

Incubator