

uxjonathan.com

Blueprint de documentación para construir con IA

8 documentos completos en Markdown, con las skills, prompts y checklists para pasar del vibe coding a Spec Driven Development, con la mirada de un Design Engineer.



Jonathan Pacheco
Design Engineer

```
docs/  
└─ PRD.md  
└─ UX_FLOWS.md  
└─ DESIGN_SYSTEM.md  
└─ COMPONENTS.md  
└─ ARCHITECTURE.md  
└─ SECURITY.md  
└─ TESTING.md  
└─ AGENTS.md
```

En **lavanda**, lo que agrega la mirada de Design Engineer.

Por qué documentar

Con IA es fácil pasar de una idea a software que funciona. El problema llega cuando la app crece: **tu agente de código empieza a adivinar** qué hace cada funcionalidad, cómo se ve, cómo se guardan los datos, cómo funciona la autenticación y dónde va cada archivo. Con el tiempo, cada parte de la app sigue supuestos distintos.

Dale suficiente contexto para que adivine menos e implemente más.

Con 8 documentos en `docs/` cubres todo. Cada uno responde una pregunta, trae su plantilla completa y las skills que conviene instalar para trabajarlo con tu agente. Las partes marcadas `Design Engineer` son lo que casi ninguna guía cubre: convertir los artefactos de diseño en specs que el agente puede cumplir y las pruebas pueden verificar.

Documento	Pregunta	Absorbe
PRD.md	¿Qué construimos y cómo sabemos que está listo?	Historias, criterios de aceptación, roadmap
UX_FLOWS.md	¿Qué hace el usuario y qué ve en cada estado?	Matriz de estados, casos borde
DESIGN_SYSTEM.md	¿Cómo se ve y suena el producto?	Tokens, accesibilidad, contenido y UX writing
COMPONENTS.md	¿Qué hace cada componente, en cada estado?	Contratos, Storybook, Figma
ARCHITECTURE.md	¿Cómo funciona el sistema y dónde viven los datos?	Base de datos, API, entorno, errores, despliegue, decisiones
SECURITY.md	¿Qué hay que proteger y cómo lo revisamos?	Secretos, auditorías con IA, infraestructura
TESTING.md	¿Cómo probamos que funciona?	Playwright, QA visual, accesibilidad, seguridad, observabilidad
AGENTS.md	¿Cómo debe trabajar el agente?	Reglas, flujo de trabajo, prompts, skills instaladas

```
mi-app/
├─ src/
├─ tests/
├─ docs/
│   ├─ PRD.md
│   ├─ UX_FLOWS.md
│   ├─ DESIGN_SYSTEM.md
│   ├─ COMPONENTS.md
│   ├─ ARCHITECTURE.md
│   ├─ SECURITY.md
│   ├─ TESTING.md
│   └─ AGENTS.md          ← en Claude Code, CLAUDE.md
├─ tokens.json
├─ .env.example
└─ package.json
```

Las skills y los comandos de esta guía cambian rápido. Antes de instalar, abre el repositorio de cada una y confirma el comando vigente.

01•PRD.md

Qué construyes, por qué, para quién, qué entra y qué no, y qué significa terminado. Tu agente debería poder leerlo y entender el producto sin que se lo expliques 30 minutos.

Créalo antes de implementar. **Actualízalo** cuando cambien requisitos, funcionalidades, flujos o alcance.

● PRD.md

```
# Documento de requisitos de producto
```

1. Resumen

```
Nombre: [Nombre]
```

```
Descripción en una línea: [Explica el producto en una frase]
```

```
Tipo: [SaaS / App móvil / Marketplace / Herramienta interna / Sitio web]
```

```
Plataforma principal: [Web / iOS / Android / Escritorio]
```

2. Problema

```
¿Qué problema resolvemos? [...]
```

```
¿Por qué importa? [...]
```

```
¿Cómo lo resuelven hoy los usuarios? [Alternativas actuales]
```

```
## 3. Objetivo
Principal: [Qué debe lograr el producto]
Secundarios:
- [Objetivo]

## 4. Usuarios
### [Persona 1]
Quién es: [...]
Necesidades: [...]
Puntos de dolor: [...]

## 5. Historias de usuario
- HU-01 Como [usuario], quiero [acción] para [resultado].
- HU-02 Como [usuario], quiero [acción] para [resultado].

## 6. Funcionalidades
### F-01 [Nombre]
Propósito: [Qué hace]
Flujo: [Ver UX_FLOWS.md, Flujo 1]
Entradas: [...] Salidas: [...]
Criterios de aceptación (Dado / Cuando / Entonces):
  Escenario: [caso feliz]
    Dado que [contexto]
    Cuando [acción]
    Entonces [resultado observable]
  Escenario: [caso de error]
    Dado que [...]
    Cuando [...]
    Entonces [...]
Casos borde:
- [Caso]

## 7. Fuera de alcance
Lo siguiente NO es parte de esta versión, a propósito:
- [Funcionalidad]

## 8. Alcance y roadmap
Ahora (MVP): [...]
Siguiendo: [...]
Después: [...]
No planeado: [...]

## 9. Métricas de éxito
Principales: [Métrica + meta]
```

Secundarias: [Métrica + meta]

10. Restricciones

Técnicas: [...] De negocio: [...] De diseño: [...] De tiempo: [...]

11. Supuestos

- [Supuesto]

12. Preguntas abiertas

- [Pregunta]

13. Definición de terminado

- [] Requisitos implementados
- [] Criterios de aceptación cubiertos por pruebas
- [] Flujos críticos funcionando
- [] Pruebas, lint, tipado y build pasan
- [] Revisión de seguridad sin hallazgos críticos
- [] Documentación actualizada

Design Engineer Los criterios en Dado / Cuando / Entonces se vuelven pruebas de Playwright casi literalmente. Así, lo que diseñaste es lo que se prueba.

Herramientas para este documento

Herramienta	Para qué
Plan mode de Claude Code (Shift + Tab)	Que el agente lea el PRD y proponga un plan sin tocar código
Prompt de arranque (ver AGENTS.md)	Que te haga solo las preguntas necesarias para completar el PRD

02•UX_FLOWS.md

Los recorridos reales del usuario y, para cada pantalla, qué se ve en cada estado. La mayoría de bugs visuales de una app hecha con IA vienen de estados que nadie diseñó: si no los defines, el agente los improvisa en cada pantalla.

● UX_FLOWS.md

Flujos de UX

Mapa

Inicio → Registro → Onboarding → Dashboard → Crear proyecto → Detalle → Invitar equipo

Flujo 1 – Registro

Disparador: [Desde dónde llega el usuario]

1. Abre la página de registro
2. Escribe su email
3. Crea una contraseña
4. Corre la validación
5. Se crea la cuenta
6. Se redirige al onboarding

Éxito: [Qué ve]

Falla: [Qué ve y cómo se recupera]

Criterios: [HU-01 en PRD.md]

Flujo 2 – Crear proyecto

1. Clic en "Crear proyecto"
2. Se abre un modal
3. Escribe el nombre
4. Envía
5. La API valida
6. Se crea el proyecto
7. Vuelve a la lista con aviso "Proyecto creado"

Matriz de estados

Pantalla	Cargando	Vacío	Error	
Parcial		Éxito		
Proyectos	Skeleton x3	"Crea tu primer proyecto"	Qué pasó + botón Reintentar	
"Algunos datos no		Lista por fecha		
		+ botón Crear		
cargaron"				
Detalle	Skeleton	No aplica	"No existe o no tienes acceso"	
No aplica		Vista completa		

Casos borde

- Sin conexión o red lenta
- Sesión expirada en medio del flujo
- Doble clic en enviar
- Datos muy largos (nombres, listas de 500 elementos)
- JavaScript deshabilitado, si aplica



Reglas

- Ninguna celda queda vacía; si no aplica, se escribe "No aplica"
- Todo error dice qué pasó y cómo resolverlo
- Todo estado vacío invita a una acción
- Ningún flujo se especifica sin un prototipo validado

Design Engineer

Valida cada flujo con un prototipo antes de escribir la spec. Corregir un prototipo es más barato que corregir una spec, y mucho más que corregir código.

Herramientas para este documento

Herramienta	Para qué
Figma MCP	Que el agente lea los frames del flujo y proponga la matriz de estados
Stitch o un prototipo HTML	Validar el recorrido antes de especificarlo
Playwright MCP	Que el agente recorra el flujo en un navegador real y reporte estados faltantes

03•DESIGN_SYSTEM.md

El vocabulario visual y verbal del producto. Sin él, la IA genera una pantalla con tarjetas redondeadas y otra con esquinas rectas, otra fuente, otro espaciado y botones distintos. El documento explica el porqué; `tokens.json` guarda el qué.

● DESIGN_SYSTEM.md

Sistema de diseño

1. Dirección

Estilo: [Minimalista / Editorial / Premium / Lúdico / Técnico]

Atmósfera: [Qué debe sentir el usuario]

Principios: [Principio] · [Principio] · [Principio]

2. Marca

Nombre: [...] Logo: [Reglas de uso] Voz: [Descripción]

3. Tokens (fuente de verdad: `tokens.json`)

Regla: ningún color, radio, sombra o espaciado se escribe a mano en un componente.

Si falta un token, se propone uno nuevo y se documenta aquí.

4. Color

Primario / Secundario / Fondo / Superficie: #HEX

Texto / Texto atenuado / Borde: #HEX

Éxito / Advertencia / Error: #HEX

5. Tipografía

Títulos: [Fuente] Cuerpo: [Fuente] Código: [Fuente]

H1 / H2 / H3 / Cuerpo / Pequeño: [Tamaño / Peso / Interlineado]

6. Espaciado

Unidad base: 4 px

Escala: 4 · 8 · 12 · 16 · 24 · 32 · 48 · 64 · 96

7. Radios y sombras

Radios: sm / md / lg / píldora = [Valor]

Sombras: sm / md / lg = [Regla]

8. Íconos

Librería: [Lucide / Phosphor / ...] Tamaño: [...] Estilo: [Outline / Filled]

9. Layout y responsive

Breakpoints: móvil 375 · tablet 768 · escritorio 1280

Contenedor máximo: [...] Gutter: [...]

10. Movimiento

Duraciones: 150 / 250 ms Curva: [...]

Se respeta prefers-reduced-motion.

11. Accesibilidad (WCAG 2.2 AA)

- Contraste 4.5:1 en texto normal; 3:1 en texto grande e íconos
- Todo usable solo con teclado; orden de foco = orden visual
- Foco visible en todo elemento interactivo
- Área táctil mínima de 24 × 24 px
- Cada campo tiene <label>; errores asociados con aria-describedby
- Imágenes informativas con alt; decorativas con alt=""
- No dependas solo del color para comunicar algo

12. Voz y UX writing

- Mayúscula solo en la primera palabra: "Crear proyecto"
- Los botones son verbos que dicen qué pasa: "Guardar cambios", no "Enviar"
- Una acción conserva su nombre en todo el flujo: "Publicar" → "Publicado"
- Los errores no piden disculpas: dicen qué pasó y cómo arreglarlo
- Nombra las cosas como las entiende el usuario, no como está construido el sistema

13. Glosario

Usamos	No usamos	
Proyecto	Workspace, Espacio	
Invitar	Agregar miembro	

14. Mensajes estándar

Error de red: "No pudimos guardar. Revisa tu conexión e inténtalo de nuevo."

Sin permisos: "No tienes acceso a este proyecto. Pídele acceso a su dueño."

15. Idiomas

Por defecto: [es-CO] Otros: [en]

Ningún texto visible vive dentro del componente; está en los archivos de idioma.

● TOKENS.json

```
{
  "color": {
    "surface": { "$type": "color", "$value": "#F8FAFC" },
    "on-surface": { "$type": "color", "$value": "#0B0F19" },
    "primary": { "$type": "color", "$value": "#4F46E5" },
    "border": { "$type": "color", "$value": "#E2E8F0" }
  },
  "radius": {
    "sm": { "$type": "dimension", "$value": "6px" },
    "md": { "$type": "dimension", "$value": "12px" }
  },
  "space": {
    "1": { "$type": "dimension", "$value": "4px" },
    "2": { "$type": "dimension", "$value": "8px" },
    "4": { "$type": "dimension", "$value": "16px" }
  }
}
```

Design Engineer

Usa el formato del W3C Design Tokens Community Group y genera de ahí tus variables CSS o tu tema de Tailwind. Si diseñas en Figma o Stitch, exporta los tokens en lugar de copiarlos a mano.



Skills y herramientas para este documento

Skill o herramienta	Para qué
frontend-design (Anthropic)	Que el agente tome decisiones visuales intencionales y no plantillas genéricas
web-design-guidelines (Vercel, agent-skills)	Revisar la interfaz contra buenas prácticas de UI y accesibilidad
Figma MCP	Leer variables y estilos directamente del archivo de diseño
DESIGN.md de Stitch	Exportar el sistema completo como punto de partida

● Terminal

```
# Skills oficiales de Anthropic (incluye frontend-design y webapp-testing)
/plugin marketplace add anthropics/skills

# Skills de Vercel para React y diseño web
npx skills add vercel-labs/agent-skills

# Figma MCP (servidor remoto oficial)
claude mcp add --transport http figma https://mcp.figma.com/mcp
```

04•COMPONENTS.md

Lo que agrego como Design Engineer

El puente entre Figma y el código: un contrato por componente. Una captura no dice qué pasa en hover, al fallar la red o al navegar con teclado; el contrato sí. Si usas Storybook, cada contrato tiene una historia por estado.

● COMPONENTS.md

```
# Contratos de componente

## Plantilla
### [NombreDelComponente]
Propósito: [Qué hace y cuándo NO usarlo]
Anatomía: [Partes visibles]
Props:
| Prop | Tipo | Default | Regla |
```

Variantes: [...]

Estados: default · hover · focus-visible · active · loading · disabled · error

Tokens: [color.*, radius.*, space.*]

Teclado: [Teclas y comportamiento]

Accesibilidad: [Rol, aria-*, texto alternativo]

Contenido: [Textos y su longitud máxima]

Responsive: [Cómo cambia en móvil]

Prohibido: [Usos incorrectos]

Storybook: [Historias, una por estado]

Figma: [Enlace al componente]

Ejemplo: Button

Propósito: dispara una acción. Si navega a otra página, es un Link.

Anatomía: [ícono opcional] + etiqueta + [spinner en loading]

Props:

Prop	Tipo	Default
variant	"primary" "secondary" "danger"	"primary"
size	"sm" "md" "lg"	"md"
loading	boolean	false
disabled	boolean	false

Estados: default · hover · focus-visible · active · loading · disabled

Tokens: color.primary, radius.sm, space.2, space.4

Teclado: Enter y Espacio activan. El foco siempre es visible.

Reglas:

- Un solo botón primario por vista
- En loading conserva su ancho (sin saltos de layout)
- La etiqueta es un verbo que dice qué pasa

Storybook: Primary, Secondary, Danger, Loading, Disabled, WithIcon

Reglas para todos los componentes

- Antes de crear uno nuevo, revisa si uno existente ya cumple
- Ningún componente nuevo sin contrato aprobado
- Ningún valor visual fuera de los tokens
- Los textos viven fuera del componente



Skills y herramientas para este documento

Skill o herramienta	Para qué
Figma MCP + Code Connect	Que el agente sepa qué componente de código corresponde a cada componente de Figma
Storybook	Spec viva: una historia por estado, base de las pruebas visuales
frontend-design (Anthropic)	Que los componentes nuevos respeten la dirección visual

05•ARCHITECTURE.md

Cómo está estructurado el sistema, dónde va cada cosa, cómo se comunica todo y por qué. Aquí viven también la base de datos, la API, las variables de entorno, el manejo de errores, el despliegue y las decisiones, para no tener siete archivos más.

```
● ARCHITECTURE.md

# Arquitectura

## 1. Visión general
[El sistema en un párrafo]
Usuario → Frontend → API → Servicios → Base de datos

## 2. Patrón y stack
Patrón: [Monolito modular / Serverless / ...] porque [...]
Frontend / Backend / Base de datos / Auth / Storage /
Hosting / Pagos / Email / Analítica: [Tecnología]

## 3. Estructura del proyecto
src/
├─ components/
├─ pages/
├─ services/
├─ hooks/
├─ lib/
├─ types/
└─ config/

Reglas de dependencia:
- La UI no accede directamente a la base de datos
- La lógica de negocio vive en services/
```

```
## 4. Flujo de datos
1. El usuario envía un formulario
2. El frontend valida
3. La API autentica
4. El servicio valida reglas de negocio
5. Se actualiza la base de datos
6. Se devuelve la respuesta y la UI se actualiza

## 5. Base de datos
Motor: [PostgreSQL / ...]   Hosting: [...]
Convenciones: tablas en plural y snake_case, PK id (uuid),
FK <tabla>_id, created_at y updated_at en todas las tablas
### users - id, email, name, created_at, updated_at
### projects - id, user_id → users, name, status, created_at, updated_at
Relaciones: un usuario tiene muchos proyectos.
Restricciones: email único; name obligatorio y único por usuario.
Índices: projects(user_id) para listar por usuario.
Borrado lógico: [Sí / No]   Auditoría: [created_by, updated_by]
Migraciones: todo cambio de esquema tiene migración documentada y probada,
y no rompe datos existentes en producción.
Datos de prueba: [Descripción del seed]

## 6. API
GET /projects      auth requerida → { "projects": [] }
POST /projects     auth requerida, body { "name": "Mi proyecto" }
Validación: name obligatorio, único por usuario
Error estándar: { "error": { "code": "PROJECT_NOT_FOUND", "message": "..." } }
Reglas: valida entradas, autoriza cada recurso, no expongas detalles internos

## 7. Manejo de errores
400 validación · 401 sin sesión · 403 sin permiso · 404 no existe ·
429 demasiadas peticiones · 500 error inesperado
Cada código tiene su mensaje en DESIGN_SYSTEM.md > Mensajes estándar.
Reintentos: [Qué operaciones se reintentan y cuántas veces]

## 8. Variables de entorno
Públicas: NEXT_PUBLIC_APP_URL
Privadas: DATABASE_URL, AUTH_SECRET
Todas documentadas en .env.example.

## 9. Despliegue
Entornos: desarrollo / staging / producción = [URL]
Proceso: pruebas → lint → build → deploy → pruebas de humo → monitoreo
Rollback: [Cómo se revierte]
```

```
## 10. Rendimiento y escalabilidad
- [Requisito, por ejemplo LCP < 2 s]

## 11. Escenarios de falla
¿Qué pasa si la base de datos no responde, la API tarda,
falla un servicio externo o falla la autenticación? [Comportamiento]

## 12. Decisiones (ADR)
### D-001 [Título]
Fecha · Estado: Aceptada
Contexto: [...] Decisión: [...] Alternativas: [...]
Consecuencias: [Beneficio] / [Costo]
```

Skills para este documento

Skill	Para qué
Skills de Supabase (supabase/agent-skills)	Buenas prácticas de Postgres, RLS y Auth si tu backend es Supabase
Skills de Vercel (vercel-labs/agent-skills)	Patrones de React y Next.js y despliegue en Vercel
Cloudflare (cloudflare/skills)	Workers, Pages, KV, D1 y R2 si despliegas en Cloudflare

06•SECURITY.md

La seguridad es un requisito desde el inicio, no un retoque final. Este documento define qué se protege y, sobre todo, cómo se revisa: con IA sobre el código y con reglas sobre la infraestructura.

● SECURITY.md

Seguridad

1. Autenticación

Proveedor: [...] Métodos: [email + contraseña / OAuth / magic link]
Sesiones: [duración, renovación, cierre]

2. Autorización

Roles: admin · usuario · [rol]
Permisos por rol: [...]

La autorización se aplica SIEMPRE en el servidor (o con RLS en la base de datos).

3. Secretos

- Nunca credenciales en el código ni secretos en el repositorio
- Nunca secretos privados en el cliente
- Variables de entorno; .env en .gitignore; .env.example sin valores
- Rotar de inmediato cualquier clave expuesta

4. Seguridad de la API

Validación de entradas en el servidor · rate limiting · CORS restringido ·
autenticación y autorización por endpoint · errores sin detalles internos

5. Infraestructura

WAF y protección DDoS · rate limiting en endpoints públicos ·
captcha invisible (Turnstile) en formularios · encabezados HTTP:
CSP, HSTS, X-Content-Type-Options, Referrer-Policy

6. Datos

Sensibles: [lista de campos]
Cifrado: [en tránsito y en reposo]
Retención y eliminación: [reglas]
Legal: [por ejemplo, Ley 1581 de 2012 en Colombia]

7. Logs

Nunca registres: contraseñas, API keys, tokens, datos personales sensibles.
Sí registra: eventos de autenticación, acciones importantes, errores, eventos de seguridad.

8. Subida de archivos

Tipos permitidos · tamaño máximo · validación en servidor · bucket privado

9. Dependencias

Revisión automática (Dependabot o npm audit); nada sin mantenimiento activo.

10. Revisión con IA

Antes de cada commit importante: /security-review
En cada pull request: GitHub Action de Claude Code Security Review
Antes de lanzar: auditoría completa con security-audit (Cloudflare)

11. Checklist

- [] Autenticación y autorización implementadas en servidor
- [] Secretos fuera del repo y del cliente
- [] Validación de entradas en servidor
- [] Rate limiting y WAF activos
- [] Encabezados HTTP verificados

- [] Subidas validadas
- [] Errores sin detalles sensibles
- [] Dependencias revisadas
- [] /security-review sin hallazgos críticos

Skills y herramientas de seguridad

Skill o comando	Capa	Qué hace
<code>/security-review</code> (Claude Code)	Código	Viene incluido en Claude Code. Revisa los cambios pendientes de tu rama y busca vulnerabilidades como inyección, XSS, fallas de autenticación y dependencias riesgosas.
<code>claude-code-security-review</code> (GitHub Action)	Código	El mismo análisis, automático en cada pull request, con comentarios sobre las líneas afectadas.
<code>security-audit</code> (Cloudflare)	Código	Auditoría en seis fases: un agente encuentra el problema y otro distinto intenta refutarlo antes de reportarlo.
<code>cloudflare-security</code> (comunidad, en español)	Infraestructura	Audita tu cuenta de Cloudflare y activa, con tu confirmación, WAF, rate limiting, Turnstile y secretos. Pensada para quien no es experto en seguridad.
<code>cloudflare</code> (cloudflare/skills)	Infraestructura	Skill oficial de la plataforma, incluye seguridad (WAF y DDoS) e infraestructura como código.

● Terminal

```
# 1. Código: incluido en Claude Code (actualiza a la última versión)
/security-review

# 2. Código en cada PR: .github/workflows/security.yml
#   - uses: anthropics/claude-code-security-review@main
#   with:
#     claude-api-key: ${ secrets.CLAUDE_API_KEY }

# 3. Auditoría completa antes de lanzar (Cloudflare, oficial)
npx skills add https://github.com/cloudflare/security-audit-skill --skill security-audit

# 4. Infraestructura en Cloudflare (comunidad, en español)
git clone https://github.com/santmun/cloudflare-security-skill.git \
  ~/.claude/skills/cloudflare-security
#   luego en Claude Code: /cloudflare-security

# 5. Plataforma Cloudflare (oficial)
/plugin marketplace add cloudflare/skills
/plugin install cloudflare@cloudflare
```

La regla práctica: **código** con `/security-review` en cada cambio y la GitHub Action en cada PR; **infraestructura** con la skill de Cloudflare una vez al configurar y cada vez que agregues un endpoint público.

07•TESTING.md

Que el build pase no significa que funcione. Este documento define qué es éxito, qué tipos de prueba usa el proyecto y con qué herramientas. La herramienta central es **Playwright**: abre un navegador real, recorre la app como un usuario y además le sirve a tu agente para ver lo que construyó.

● TESTING.md

Pruebas

1. Objetivo

Funcionalidad crítica: [lista]

Cada criterio de aceptación del PRD tiene al menos una prueba.

2. Tipos de prueba

Tipo	Qué prueba	Herramienta	
------	------------	-------------	--



Unitaria	Funciones y lógica de negocio aisladas	Vitest / Jest	
Componente	Un componente en todos sus estados	Storybook / Playwright CT	
Integración	API + base de datos + auth juntas	Vitest + base de prueba	
End-to-end	Recorridos completos en navegador real	Playwright	
Visual	Que nada se movió ni cambió de estilo	Playwright screenshots	
Accesibilidad	Contraste, roles, foco, etiquetas	@axe-core/playwright	
Rendimiento	LCP, CLS, peso de la página	Lighthouse CI	
Seguridad	Vulnerabilidades en código y dependencias	/security-review, audit	
Humo	Que producción responde tras desplegar	Playwright contra prod	

3. Estructura

tests/

└─ unit/

└─ integration/

└─ e2e/ ← Playwright

└─ visual/ ← capturas de referencia

└─ seed.spec.ts ← punto de partida para los agentes de Playwright

4. Flujos críticos

Flujo	Pasos	Resultado esperado	Criterio	
crear-proyecto	Dashboard → Crear → Guardar	Aparece en la lista + aviso	HU-03	

5. Estados

Cada fila de la matriz de UX_FLOWS.md tiene una prueba visual.

6. Umbrales

Lighthouse ≥ 95 · LCP < 2 s · CLS < 0.05 · 0 errores críticos de axe

Cobertura: prioriza lógica de negocio, auth, pagos y mutaciones de datos.

7. Comandos

```
npm test                # unitarias
npx playwright test     # end-to-end
npx playwright test --ui # modo visual para depurar
npx playwright show-report # reporte HTML
npm run lint · npm run typecheck · npm run build
```

8. Definición de terminado

- [] Pruebas, lint, tipado y build pasan
- [] Criterios del PRD cubiertos
- [] Cada estado de UX_FLOWS se ve como se definió
- [] Umbrales cumplidos
- [] /security-review sin hallazgos críticos

Instala Playwright desde el día uno

No lo dejes para el final. Con Playwright instalado, tu agente puede abrir la app, hacer clic, tomar capturas y comprobar que lo que construyó realmente funciona, en lugar de decirte "el código se ve bien".

● Terminal

```
# 1. Instala Playwright y sus navegadores
npm init playwright@latest
npx playwright install

# 2. Dale a Claude Code un navegador real (Playwright MCP)
claude mcp add playwright npx @playwright/mcp@latest

# 3. Agentes de prueba de Playwright (planner, generator, healer)
npx playwright init-agents --loop=claude

# 4. Accesibilidad dentro de tus pruebas
npm i -D @axe-core/playwright

# 5. Corre, depura y revisa
npx playwright test
npx playwright test --ui
npx playwright codegen http://localhost:3000 # graba un flujo y genera el código
```

Los tres agentes de prueba de Playwright (desde la versión 1.56) se instalan como subagentes de Claude Code y trabajan en cadena:

Agente	Qué hace
planner	Explora la app y escribe un plan de pruebas en Markdown
generator	Convierte ese plan en archivos de prueba de Playwright
healer	Corre las pruebas y repara las que fallan, por ejemplo si cambió el texto de un botón

● crear-proyecto.spec.ts

```
import { test, expect } from '@playwright/test';
import AxeBuilder from '@axe-core/playwright';

test('HU-03 crear proyecto con nombre válido', async ({ page }) => {
  await page.goto('/dashboard');
  await page.getByRole('button', { name: 'Crear proyecto' }).click();
  await page.getByLabel('Nombre').fill('Lanzamiento Q4');
  await page.getByRole('button', { name: 'Guardar' }).click();

  await expect(page.getByText('Proyecto creado')).toBeVisible();
  await expect(page).toHaveScreenshot('proyectos-exito.png'); // QA visual

  const a11y = await new AxeBuilder({ page }).analyze(); // accesibilidad
  expect(a11y.violations).toEqual([]);
});
```

Skills y herramientas de testing

Skill o herramienta	Para qué
Playwright MCP	Que el agente controle un navegador real mientras construye
Agentes de Playwright (init-agents)	Planear, generar y reparar pruebas end-to-end
webapp-testing (Anthropic)	Skill para probar apps web locales con Playwright: capturas, logs y depuración
playwright-cli (Microsoft)	Skill para manejar Playwright desde la terminal del agente
Lighthouse CI	Fallar el build si bajan rendimiento o accesibilidad

Design Engineer

Las pruebas funcionales no ven que un botón quedó 4 px corrido. `toHaveScreenshot()` sí: compara cada estado contra su captura de referencia en 375, 768 y 1280 px.

08 • AGENTS.md

Convierte los otros siete documentos en reglas operativas para tu agente. En Claude Code este archivo se llama CLAUDE.md, y `/init` te genera uno inicial a partir de tu código.

● AGENTS.md

```
# Instrucciones para el agente de código

## 1. Contexto
Proyecto: [Nombre]  Propósito: [Descripción]  Usuarios: [...]

## 2. Fuente de verdad
Antes de cualquier cambio importante, lee /docs:
PRD, UX_FLOWS, DESIGN_SYSTEM, COMPONENTS, ARCHITECTURE, SECURITY, TESTING.
No contradigas un documento sin señalar el conflicto.

## 3. Reglas
- Entiende el código existente antes de modificarlo
- Reutiliza componentes, tokens, utilidades y patrones
- No inventes requisitos, campos, APIs ni componentes ya definidos
- Ningún valor visual fuera de tokens.json
- Ningún componente nuevo sin contrato en COMPONENTS.md
- Nunca expongas secretos
- Sin dependencias nuevas sin preguntar
- Sin cambios en archivos no relacionados

## 4. Antes de implementar
1. Lee la documentación relevante
2. Revisa el código relevante
3. Identifica archivos, componentes y dependencias afectadas
4. Identifica cambios de datos, API y seguridad
5. Identifica las pruebas necesarias
6. Escribe un plan y espera aprobación

## 5. Durante
Cambios pequeños y enfocados. Preserva el comportamiento existente.

## 6. Después
1. Corre pruebas, lint, tipado y build
2. Corre Playwright y revisa las capturas
3. Corre /security-review
4. Revisa el diff y busca cambios no intencionales
5. Actualiza /docs si algo cambió
```



7. Conflictos

Si algo no está definido o los documentos chocan: no adivines, pregunta.

8. Skills disponibles

frontend-design · webapp-testing · Playwright MCP · agentes de Playwright · security-audit · [las de tu stack]

9. Terminado cuando

Requisitos cumplidos · pruebas y build pasan · revisión de seguridad sin hallazgos críticos · revisión de diseño sin hallazgos críticos · sin archivos de más · /docs actualizado

Flujo de trabajo con el agente

- 1 **Contexto.** "Lee /docs. Todavía no cambies código. Resume lo que entendiste y dime qué falta o qué se contradice."
- 2 **Plan.** "Haz un plan para [funcionalidad]: requisitos, archivos, componentes, datos, API, seguridad, pruebas, casos borde y riesgos."
- 3 **Revisa el plan.** Aquí mantienes el control.
- 4 **Implementa.** "Implementa el plan aprobado siguiendo /docs. Reutiliza lo existente. Nada de dependencias ni cambios de más."
- 5 **Valida.** "Corre pruebas, lint, tipado, build y Playwright. Luego /security-review. Dime qué cambió, qué validaste y qué queda pendiente."
- 6 **Revisa el diseño.** Design Engineer "Compara la pantalla contra tokens, contratos, estados, accesibilidad y textos en 375, 768 y 1280 px. Entrega una tabla de hallazgos con severidad."
- 7 **Actualiza la documentación.** "¿Esta implementación cambió algún requisito, la arquitectura, la API, los datos o la seguridad? Si es así, actualiza el documento."

Prompt maestro

Prompt

Estás trabajando dentro de un proyecto existente.

Trata /docs como fuente de verdad para producto, flujos, diseño, componentes, arquitectura, seguridad, pruebas y comportamiento del agente.

No inventes requisitos, campos, APIs, componentes ni patrones ya definidos.

Antes de un cambio no trivial: lee la documentación y el código relevantes, identifica archivos, dependencias, cambios de datos y API, seguridad y pruebas, y escribe un plan. Espera mi aprobación.

Con el plan aprobado: implementa el cambio correcto más pequeño, reutiliza patrones, sigue el sistema de diseño y la arquitectura, y preserva el comportamiento existente.

Después: corre pruebas, lint, tipado, build y Playwright; corre /security-review; revisa el diff; busca regresiones.

Si algo importante no está definido o los requisitos chocan, no adivines: pregunta.

Al final resume: qué cambió, archivos tocados, pruebas corridas, resultados, problemas pendientes y documentación por actualizar.

Prompt para arrancar un proyecto nuevo

● Prompt

Quiero construir una aplicación nueva. Todavía no escribas código.

Ayúdame a crear los 8 documentos de /docs en este orden:
PRD, UX_FLOWS, DESIGN_SYSTEM, COMPONENTS, ARCHITECTURE, SECURITY, TESTING, AGENTS.

Para cada uno: explica su propósito, identifica supuestos y preguntas abiertas, mantén los requisitos explícitos, no inventes decisiones de negocio y reutiliza las decisiones de forma consistente entre documentos.

Empieza por el PRD y hazme solo las preguntas necesarias.
No generes código hasta que yo apruebe la documentación.

Skills para este documento

Skill o comando	Para qué
<code>/init</code> (Claude Code)	Genera un CLAUDE.md inicial leyendo tu proyecto
<code>skill-creator</code> (Anthropic)	Convierte un proceso que repites, como tu revisión de diseño, en una skill propia

Antes de pedir código

- ☐ Problema, usuario, alcance y roadmap definidos en el PRD
- ☐ Criterios de aceptación en Dado / Cuando / Entonces
- ☐ Flujos mapeados, prototipo validado y matriz de estados sin celdas vacías
- ☐ Tokens exportados, accesibilidad y voz definidas
- ☐ Contrato para cada componente nuevo
- ☐ Stack, estructura, datos, API, entorno y despliegue documentados
- ☐ Reglas de seguridad escritas y /security-review configurado
- ☐ Playwright instalado y tipos de prueba definidos
- ☐ AGENTS.md (CLAUDE.md) listo con las skills que usa el proyecto

No empieces con "constrúyeme una app". Empieza con "aquí está todo lo que necesitas saber sobre la app".



Jonathan Pacheco

Design Engineer · Ing. de Sistemas. Convierto el diseño de producto en especificaciones que equipos y agentes de IA ejecutan sin adivinar.

[Ver la guía online](#)

✉ info@uxjonathan.com ☎ +57 318 330 5826

🌐 uxjonathan.com 📷 [@ux.jonathan](https://www.instagram.com/ux.jonathan)

