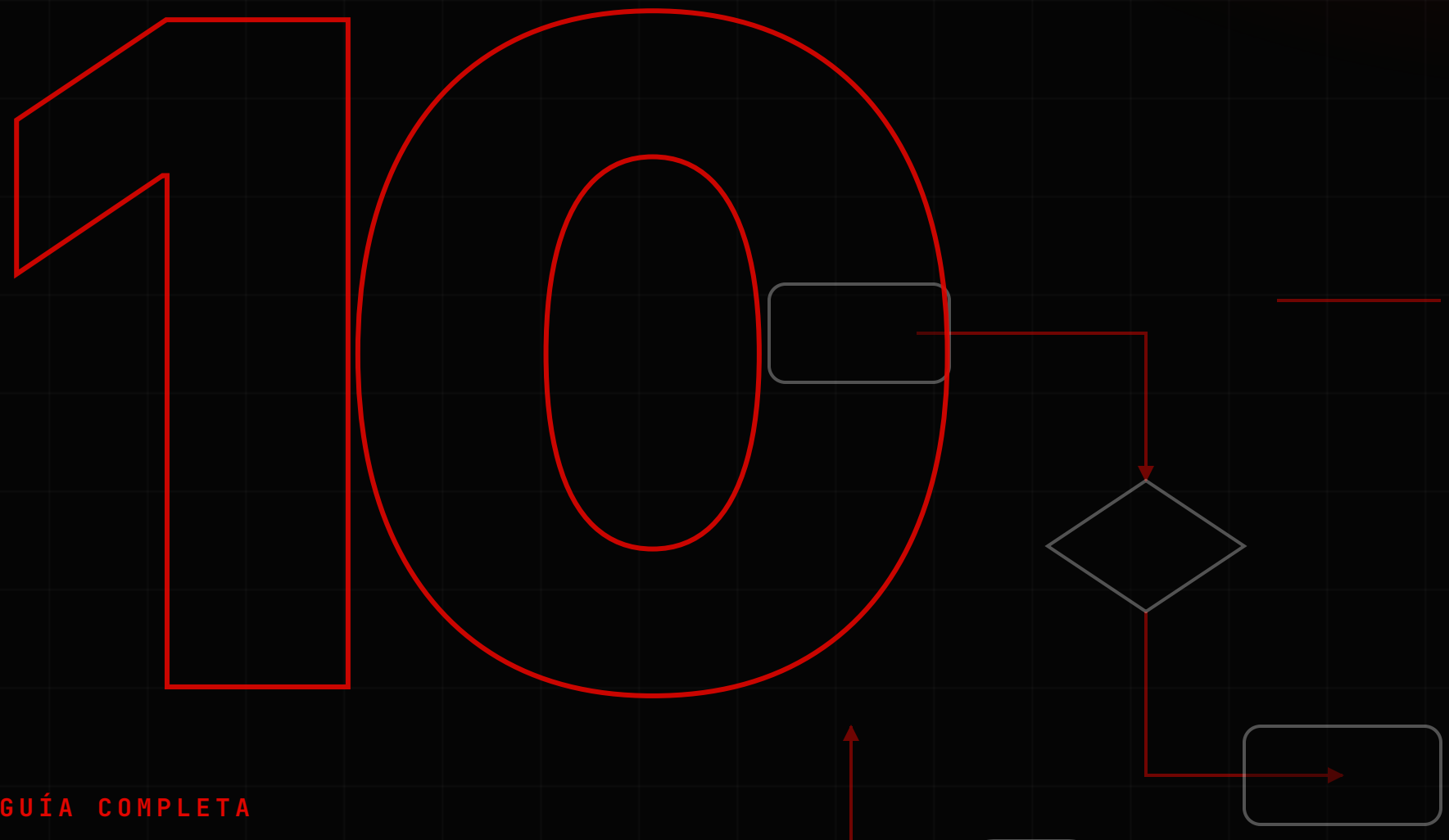
ANIMARE.dev

GUÍA · 2026



GUÍA COMPLETA

# 10 diagramas que todo programador debe conocer

Qué son, cómo se leen, cómo se crean y cómo mantenerlos vivos en proyectos reales.

| | | | | |
|---|---|---|---|---|
| 01 Flujo | 02 Estados | 03 ER | 04 Secuencia | 05 Arquitectura |
| 06 Clases | 07 Casos de uso | 08 Despliegue | 09 Gantt | 10 Red |

Oct 21, 2026

Guía de **animare.dev**

INTRODUCCIÓN

# Por qué los diagramas importan en proyectos reales

Un diagrama es una forma de **pensar y comunicar un sistema** antes de escribir código, mientras se escribe y después de entregarlo. Un buen diagrama cuesta minutos y evita semanas de malentendidos entre programadores, clientes y equipo.

## CUATRO FUNCIONES EN UN PROYECTO REAL

**01**

### Entender

el problema antes de construir.

**02**

### Acordar

con otras personas qué se va a hacer.

**03**

### Documentar

lo construido para quien llegue después.

**04**

### Detectar

errores cuando todavía son baratos de corregir.

EN EL DIAGRAMA

**una conversación**

EN PRODUCCIÓN

**dinero y clientes**

**Un error de diseño descubierto en un diagrama cuesta una conversación; descubierto en producción cuesta dinero y clientes.**

ANTES DE EMPEZAR

# Cómo usar esta guía

Cada uno de los 10 diagramas sigue la **misma estructura**, para que puedas convertirla en páginas visuales con un formato repetible.

CADA DIAGRAMA = 3 PÁGINAS

1. **Qué es y cuándo usarlo**
   La pregunta que responde el diagrama.
2. **Elementos y símbolos**
   El vocabulario básico.
3. **Cómo se lee**
   Cómo interpretarlo sin ayuda.
4. **Cómo crearlo paso a paso**
   El proceso real de dibujarlo.
5. **Cómo mantenerlo**
   Qué hacer cuando el proyecto cambia.
6. **Ejemplo real**
   Un caso concreto de proyecto.
7. **Errores comunes**
   Lo que hace que un diagrama sea inútil.
8. **Idea para la versión visual**
   Sugerencia de cómo dibujarlo en tu guía.

TRES PÁGINAS POR DIAGRAMA

**Página 1:** qué es, cuándo usarlo y el ejemplo dibujado. **Página 2:** elementos, cómo se lee y cómo crearlo. **Página 3:** ejemplo real, mantenimiento, errores e idea visual.

ÍNDICE

# Los 10 diagramas y la pregunta que responde cada uno


| | | |
|---|---|---|
| 01 | **Flujo** | ¿Qué pasos sigue un proceso y qué decisiones hay en el camino? |
| 02 | **Estados** | ¿En qué situaciones puede estar algo y cómo pasa de una a otra? |
| 03 | **Entidad-relación** | ¿Qué datos guardamos y cómo se relacionan? |
| 04 | **Secuencia** | ¿Quién habla con quién y en qué orden? |
| 05 | **Arquitectura** | ¿Cuáles son las piezas del sistema y cómo se conectan? |
| 06 | **Clases** | ¿Cómo se organiza el código orientado a objetos? |
| 07 | **Casos de uso** | ¿Qué puede hacer cada tipo de usuario con el sistema? |
| 08 | **Despliegue** | ¿Dónde corre cada pieza en servidores y dispositivos? |
| 09 | **Gantt** | ¿Qué se hace, cuándo y en qué orden de dependencia? |
| 10 | **Red** | ¿Cómo se conectan los equipos y por dónde viaja la información? |

FUNDAMENTOS COMUNES

# Crear, versionar y mantener cualquier diagrama

Antes de entrar a cada diagrama, hay principios que valen para todos. Si los aplicas, tus diagramas serán **útiles en lugar de decorativos**.

## 1 Un diagrama responde una sola pregunta

Si un diagrama intenta explicar el flujo, los datos y los servidores al mismo tiempo, no explica nada. Antes de dibujar, escribe la pregunta en una frase. Todo lo que no ayude a responderla se queda fuera.

EJEMPLO DE PREGUNTA

**"¿Cómo se paga una orden en el sistema?"**

## 2 Define el público

Dibujar el mismo diagrama para los dos públicos es la causa más común de confusión.

PARA EL CLIENTE

Casos de uso, flujo de negocio

Palabras del negocio y cero tecnicismos.

PARA EL EQUIPO

Secuencia, clases, despliegue

Puede ser técnico.

## 3 Nivel de detalle consistente

Todos los elementos de un diagrama deben estar en el mismo nivel de abstracción. Si necesitas más detalle, haz un segundo diagrama que haga zoom en una parte.

`Servidor` `función calcularTotal()` no mezclar en la misma figura

FUNDAMENTOS COMUNES

# Hazlo legible

Dos principios más que separan un diagrama útil de uno decorativo.

## 4 Nombres claros y estables

Usa los mismos nombres que usa el código y el equipo. Si en la base de datos la tabla se llama `orders`, el diagrama dice “orders” o “Orden” de forma consistente, no “Pedido” en un lado y “Venta” en otro.

`orders` `Orden` `Pedido` `Venta`

## 5 Legibilidad antes que perfección

**7–12**
**elementos principales**
Por diagrama; más que eso, divide.

**L›R**
**flujo de lectura**
De izquierda a derecha o de arriba abajo.

**0**
**cruces sin razón**
Evita cruzar líneas; si cruzas, que sea por una razón.

**1**
**color por tipo**
Usa color con intención y explica la leyenda.

**T·F·V**
**título, fecha y versión**
Siempre inclúyelos en el diagrama.

REGLA DE ORO

Un diagrama desactualizado es **peor que no tener diagrama**, porque engaña.

FUNDAMENTOS COMUNES

# Cómo mantener los diagramas vivos

1

## Guárdalos junto al código

Si usas texto como código (Mermaid, PlantUML, D2), el diagrama vive en el repositorio y se revisa en los mismos pull requests que el código.

2

## Asigna un responsable por diagrama

Alguien debe ser quien lo actualiza cuando cambia el sistema.

3

## Regla de cambio

Si un pull request modifica la base de datos, el flujo de pago o la infraestructura, el diagrama correspondiente se actualiza en el mismo pull request.

4

## Revisión periódica

Cada trimestre o cada hito, revisa los diagramas clave contra la realidad. Marca la fecha de última verificación.

5

## Versiona

Pon fecha y versión en la esquina. Guarda las versiones viejas importantes (por ejemplo, la arquitectura antes de una migración).

6

## Borra lo obsoleto

Un diagrama que ya no representa nada debe archivarse, no dejarse flotando.

FUNDAMENTOS COMUNES

# Herramientas habituales

| TIPO | HERRAMIENTAS | CUÁNDO CONVIENE |
|---|---|---|
| **Texto como código** | Mermaid, PlantUML, D2 | Equipos técnicos; el diagrama se versiona con Git |
| **Dibujo manual** | draw.io (diagrams.net), Excalidraw, Figma, Miro | Talleres, presentaciones y diagramas informales |
| **Especializadas** | dbdiagram.io (ER), Lucidchart, Visual Paradigm | Diagramas formales o colaboración con muchos roles |
| **Cronogramas** | GanttProject, Notion, Jira, Excel | Planeación y seguimiento de tiempos |

## ERRORES QUE APLICAN A TODOS LOS DIAGRAMAS

- ✕ Dibujar una vez al inicio y nunca más actualizarlo.
- ✕ Mezclar niveles de detalle.
- ✕ Usar símbolos sin leyenda.
- ✕ Hacerlo tan grande que nadie lo lee.
- ✕ Dibujarlo solo para cumplir con un requisito y no para resolver un problema real.

DIAGRAMA 01 DE 10 · FLUJO

# Diagrama de flujo

01

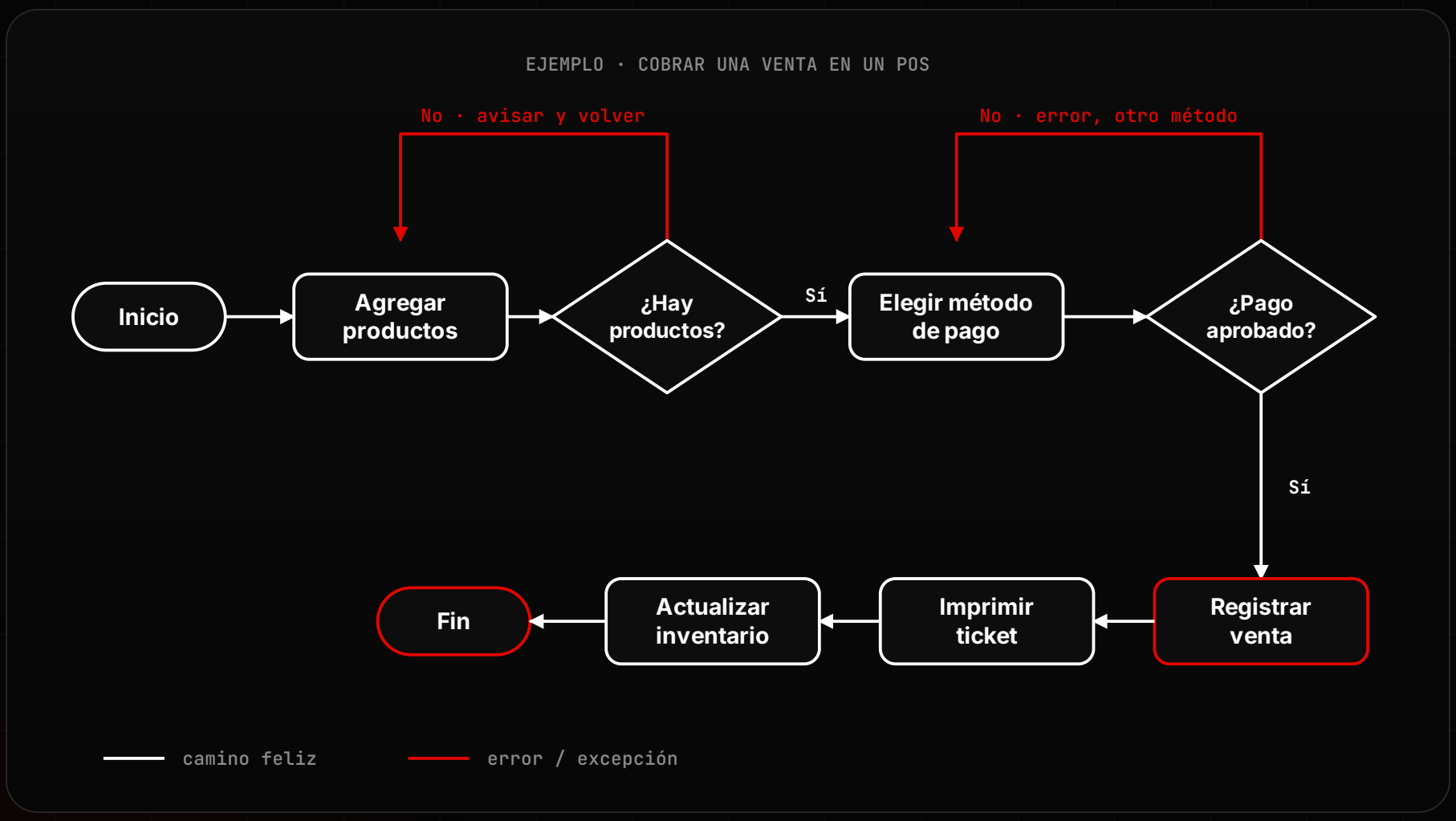
## ¿Qué pasos sigue un proceso y qué decisiones hay en el camino?

### QUÉ ES Y CUÁNDO USARLO

El diagrama de flujo muestra los pasos de un proceso y las decisiones que cambian su camino. Responde la pregunta: **¿qué pasa, paso a paso, desde que empieza hasta que termina?** Es el diagrama más universal: lo entiende un programador, un cliente y un gerente por igual.

› Úsalo para: explicar un algoritmo, documentar un proceso de negocio (cómo se procesa un pedido), planear la lógica antes de programarla, o encontrar cuellos de botella y pasos redundantes.

### EJEMPLO DIBUJADO



01 · FLUJO

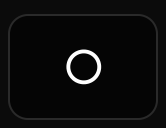
# Elementos y símbolos

**Inicio / Fin**
Óvalo o rectángulo redondeado
Donde empieza y termina el proceso

**Proceso**
Rectángulo
Una acción (“calcular total”)

**Decisión**
Rombo
Pregunta con salidas Sí/No o varias opciones

**Entrada / Salida**
Paralelogramo
Datos que entran o salen (“leer cantidad”)



**Conector**
Círculo pequeño
Une partes de un diagrama grande

**Flecha**
Línea con punta
Dirección del flujo

## CÓMO SE LEE

Se empieza en el óvalo de inicio y se sigue la flecha. En cada rombo se responde la pregunta y se toma la salida correspondiente. Cada camino debe terminar en un óvalo de fin; si un camino queda “colgado”, hay un error de lógica.

## CÓMO CREARLO PASO A PASO

1. **Define dónde empieza y dónde termina** el proceso. Sin límites claros el diagrama no termina nunca.
2. **Lista los pasos en orden** en una hoja, sin dibujar todavía. Verbos en infinitivo o presente: “validar pago”, “enviar recibo”.
3. **Identifica las decisiones.** Toda frase “si… entonces…” es un rombo.
4. **Dibuja el camino feliz primero:** lo que pasa cuando todo sale bien, en línea recta.
5. **Agrega las ramas de error y excepción:** pago rechazado, producto agotado, usuario sin permiso.
6. **Revisa** que cada rombo tenga todas sus salidas y que cada camino llegue a un fin.
7. **Simplifica.** Si un bloque tiene más de unos 15 pasos, sepáralo en subprocesos.

01 · FLUJO

# Ejemplo real

COBRAR UNA VENTA EN UN SISTEMA DE PUNTO DE VENTA (POS)

Inicio › el cajero agrega productos › ¿hay productos en el carrito? (No: avisar y volver; Sí: continuar) › elegir método de pago › ¿es efectivo? (Sí: capturar monto recibido y calcular cambio; No: enviar a la terminal) › ¿la terminal aprobó? (No: mostrar error y volver a elegir método; Sí: continuar) › registrar la venta › imprimir ticket › actualizar inventario › Fin.

Este diagrama sirve para **capacitar cajeros**, para que el programador no olvide el caso “pago rechazado” y para detectar que **falta decidir qué pasa si la impresora falla**.

## CÓMO MANTENERLO

- ☑ Actualízalo cada vez que cambie una regla de negocio (por ejemplo, un nuevo método de pago agrega una rama).
- ☑ Compáralo con el código de la función principal en cada revisión.
- ☑ Si el diagrama crece demasiado, divídelo: un diagrama general y diagramas de detalle enlazados.

## ERRORES COMUNES

- ✕ Rombos con una sola salida o sin etiquetas en las salidas.
- ✕ Mezclar decisiones técnicas con pasos de negocio.
- ✕ Dibujar solo el camino feliz y olvidar errores.
- ✕ Flechas que se cruzan en todas direcciones.

IDEA PARA LA VERSIÓN VISUAL

Dibuja el mismo ejemplo del POS con colores: **blanco** para el camino feliz, **rojo** para errores y gris para pasos opcionales. Así el lector ve de un vistazo cuánto del proceso son excepciones.

# Diagrama de estados

## ¿En qué situaciones puede estar algo y cómo pasa de una a otra?

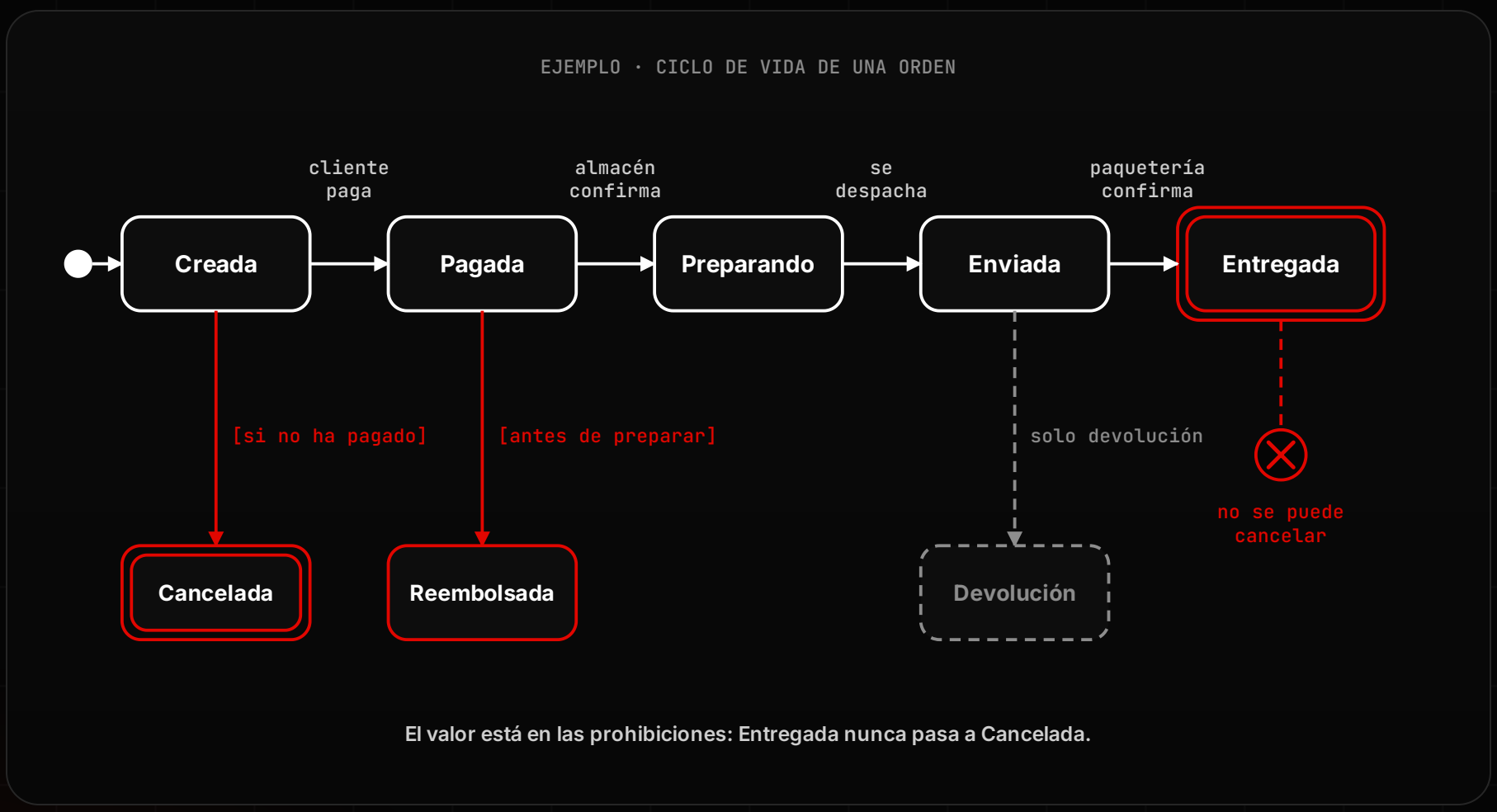
### QUÉ ES Y CUÁNDO USARLO

El diagrama de estados describe las situaciones en las que puede estar un objeto o sistema (sus **estados**) y los eventos que lo llevan de una situación a otra (las **transiciones**). Responde: **¿en qué estado puede estar esto y qué lo hace cambiar?**

El diagrama de flujo describe un proceso con principio y fin; el de estados describe algo que vive mucho tiempo y cambia: un pedido, un usuario, una suscripción, un ticket de soporte, una conexión.

› Úsalo cuando algo tiene un ciclo de vida, cuando ciertas acciones solo son válidas en ciertos estados (no puedes cancelar un pedido ya entregado), o cuando tienes muchos `if status == ...` dispersos en el código.

### EJEMPLO DIBUJADO




El valor está en las prohibiciones: Entregada nunca pasa a Cancelada.

# Elementos y símbolos

**Estado inicial**
`Círculo negro`
Dónde empieza el objeto

**Estado**
`Rectángulo redondeado`
Una situación estable (“Pagado”)

**Transición**
`Flecha`
Cambio de un estado a otro

**Evento**
`Texto sobre la flecha`
Lo que provoca el cambio (“cliente paga”)

**Condición (guarda)**
`Texto entre corchetes`
Requisito para que ocurra: [stock disponible]

**Estado final**
`Círculo con borde doble`
El ciclo de vida terminó

## CÓMO SE LEE

Se parte del círculo inicial. En cada estado se miran las flechas que salen: son las únicas cosas que pueden pasar ahí. **Si una acción no tiene flecha desde un estado, esa acción está prohibida en ese estado.**

## CÓMO CREARLO PASO A PASO

1. **Elige un solo objeto** (por ejemplo, “Orden”). Un diagrama por objeto.
2. **Lista todos los estados** que puede tener, como adjetivos o participios: Creada, Pagada, Preparando, Enviada, Entregada, Cancelada.
3. **Marca el estado inicial** y los finales.
4. **Dibuja las transiciones** y escribe el evento que las provoca.
5. **Agrega condiciones** donde haga falta: cancelar solo si aún no se envió.
6. **Revisa los estados huérfanos:** ¿hay alguno al que no se pueda llegar o del que no se pueda salir sin ser final?
7. **Revisa las transiciones imposibles:** ¿qué pasa si llega un evento que ese estado no espera? Decide si se ignora o es un error.

# Ejemplo real

**ESTADOS DE UNA ORDEN EN UNA TIENDA EN LÍNEA**

Creada › (cliente paga) › Pagada › (almacén confirma) › Preparando › (se despacha) › Enviada › (paquetería confirma) › Entregada (final).

Desde Creada se puede cancelar [si no ha pagado] › Cancelada (final). Desde Pagada se puede cancelar [antes de preparar] › Reembolsada. Desde Enviada ya no se puede cancelar: solo se puede iniciar una devolución.

El valor está en las **prohibiciones**: el diagrama deja claro que un pedido Entregada no puede pasar a Cancelada, y eso evita bugs de reembolsos erróneos.

## CÓMO MANTENERLO

- ☑ Cada vez que se agrega un estado nuevo (por ejemplo, “En revisión por fraude”), se actualiza el diagrama y el código del enum o máquina de estados.
- ☑ Lo ideal es que el código de la máquina de estados y el diagrama salgan de la misma fuente (por ejemplo, con XState o un diagrama Mermaid en el repo).
- ☑ Revisa que la base de datos no permita estados que el diagrama no contempla.

## ERRORES COMUNES

- ✕ Confundir estados con acciones (“Pagando” en lugar de “Pagada”).
- ✕ Estados que se solapan o son ambiguos.
- ✕ Olvidar las transiciones de error y cancelación.
- ✕ Hacer un mega diagrama con cientos de estados en lugar de usar estados anidados o dividir el objeto.

**IDEA PARA LA VERSIÓN VISUAL**

Dibuja el ejemplo como una “línea de vida” con cajas de colores y marca en **rojo** las flechas de cancelación. Añade una versión tipo **semáforo** que resalte qué acciones están permitidas en cada estado.

DIAGRAMA 03 DE 10 · ENTIDAD-RELACIÓN

# Diagrama entidad-relación (ER)

03

## ¿Qué datos guardamos y cómo se relacionan?

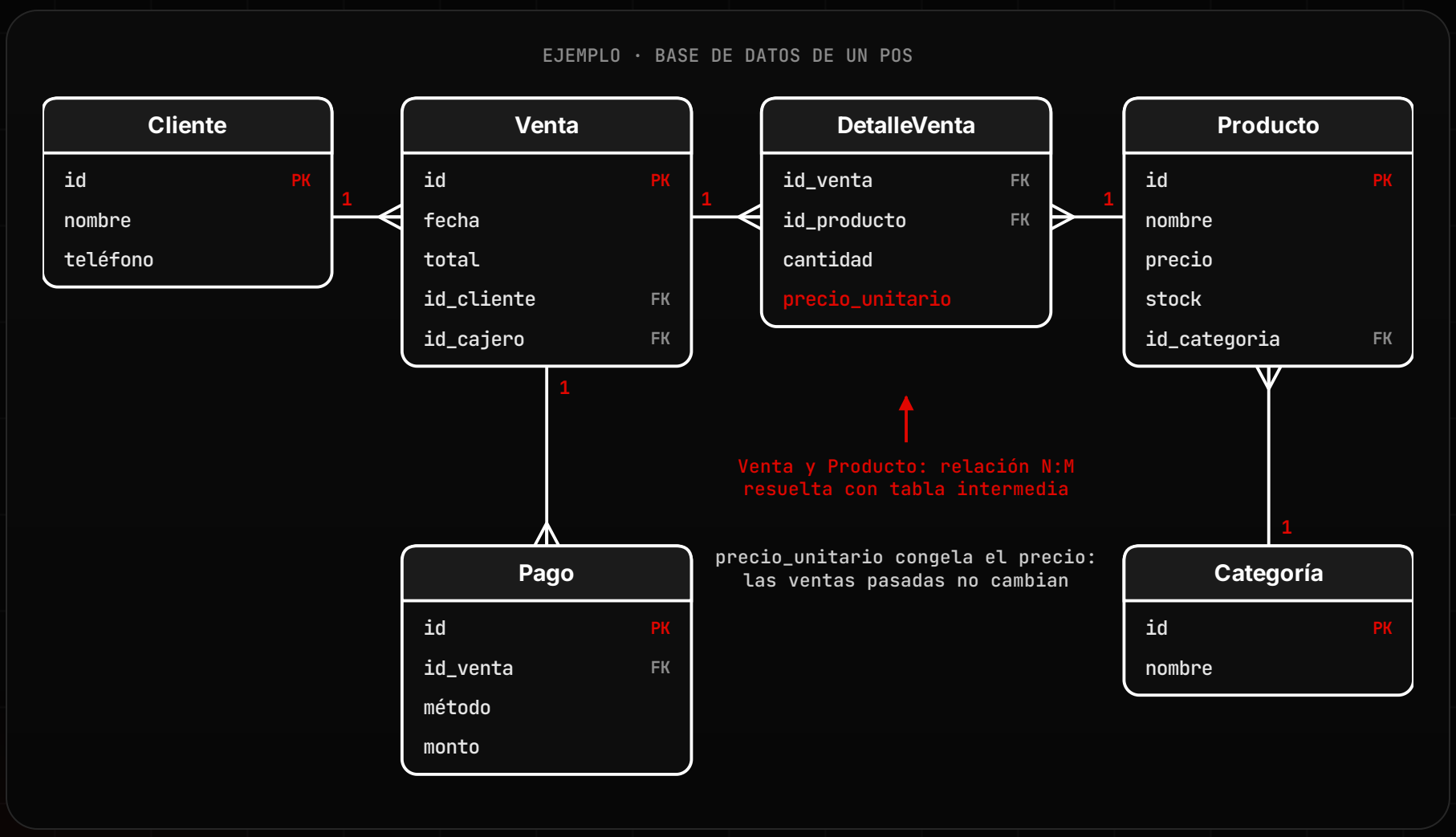
### QUÉ ES Y CUÁNDO USARLO

El diagrama ER muestra qué datos guarda un sistema (las **entidades**, que normalmente se vuelven tablas), qué información tiene cada uno (los **atributos**, que se vuelven columnas) y cómo se conectan entre sí (las **relaciones**). Responde: **¿qué información guardamos y cómo se relaciona?**

Es el plano de la base de datos.

› Úsalo antes de crear tablas, al integrar un sistema con otro, al explicar un modelo de datos a alguien nuevo o al detectar datos duplicados y mal organizados.

### EJEMPLO DIBUJADO



# Elementos y símbolos

### Entidad
Rectángulo / tabla

Algo de lo que guardamos datos (Cliente, Producto)

### Atributo
Fila dentro de la tabla

Dato de la entidad (nombre, precio)

### Clave primaria (PK)
Atributo marcado

Identifica de forma única cada registro

### Clave foránea (FK)
Atributo marcado

Apunta a la clave primaria de otra tabla

### Relación
Línea entre entidades

Cómo se vinculan

### Cardinalidad
Símbolos en los extremos (pata de gallo)

Cuántos de un lado se relacionan con cuántos del otro

Las cardinalidades básicas son **uno a uno (1:1)**, **uno a muchos (1:N)** y **muchos a muchos (N:M)**.

## CÓMO SE LEE

Se toma una relación y se lee en ambas direcciones con una frase. “Un cliente puede hacer muchas órdenes” y “una orden pertenece a exactamente un cliente”. **Si las dos frases tienen sentido de negocio, la relación está bien modelada.**

## CÓMO CREARLO PASO A PASO

1. **Identifica las entidades:** los sustantivos importantes del negocio (Cliente, Producto, Orden, Pago).
2. **Define los atributos** de cada una y elige una clave primaria.
3. **Define las relaciones con verbos:** el Cliente realiza Órdenes; la Orden contiene Productos.
4. **Define la cardinalidad** de cada relación en ambos sentidos y si es obligatoria u opcional.
5. **Resuelve los muchos a muchos con una tabla intermedia:** `order_items` con claves foráneas a las dos tablas, más cantidad y precio.
6. **Normaliza:** evita guardar el mismo dato en dos lugares. El nombre del cliente va solo en Cliente.
7. **Revisa con casos reales:** ¿puedo representar una venta con descuento? ¿un producto sin stock? ¿un cliente sin correo?

# Ejemplo real

**BASE DE DATOS DE UN POS**

Entidades principales: Cliente (id, nombre, teléfono), Producto (id, nombre, precio, stock, id_categoria), Categoría (id, nombre), Venta (id, fecha, total, id_cliente, id_cajero), DetalleVenta (id_venta, id_producto, cantidad, precio_unitario), Pago (id, id_venta, método, monto) y Usuario/Cajero.

Relaciones: un Cliente tiene muchas Ventas (1:N); una Venta tiene muchos DetalleVenta (1:N); un Producto aparece en muchos DetalleVenta (1:N), lo que forma la relación muchos a muchos Venta–Producto; una Categoría agrupa muchos Productos (1:N); una Venta puede tener varios Pagos (1:N), útil para pagos mixtos.

Guardar el `precio_unitario` en DetalleVenta es una **decisión de diseño importante**: si el precio del producto cambia mañana, las ventas pasadas no cambian.

## CÓMO MANTENERLO

- ☑ Cada migración de base de datos debe reflejarse en el diagrama. Lo ideal es generarlo automáticamente desde la base de datos real (por ejemplo, leyendo el esquema de Supabase/PostgreSQL), así nunca se desactualiza.
- ☑ Documenta los cambios importantes con fecha y motivo.
- ☑ Cuando agregues una tabla, revisa si la normalización sigue bien y si necesitas índices.
- ☑ Mantén una versión simple (las 8 a 10 tablas centrales) para explicar el sistema y otra completa para el equipo técnico.

## ERRORES COMUNES

- ✕ Dejar relaciones muchos a muchos sin tabla intermedia.
- ✕ Guardar datos repetidos (desnormalización sin razón).
- ✕ Nombres inconsistentes entre diagrama y base de datos real.
- ✕ Olvidar las claves foráneas.
- ✕ Meter en un solo diagrama las 80 tablas del sistema.

**IDEA PARA LA VERSIÓN VISUAL**

Usa **tarjetas tipo tabla** con la clave primaria resaltada, y líneas de relación con la notación de pata de gallo. Incluye una mini explicación de lectura: “1 cliente › N ventas” junto a la línea.

# Diagrama de secuencia

## ¿Quién habla con quién y en qué orden?

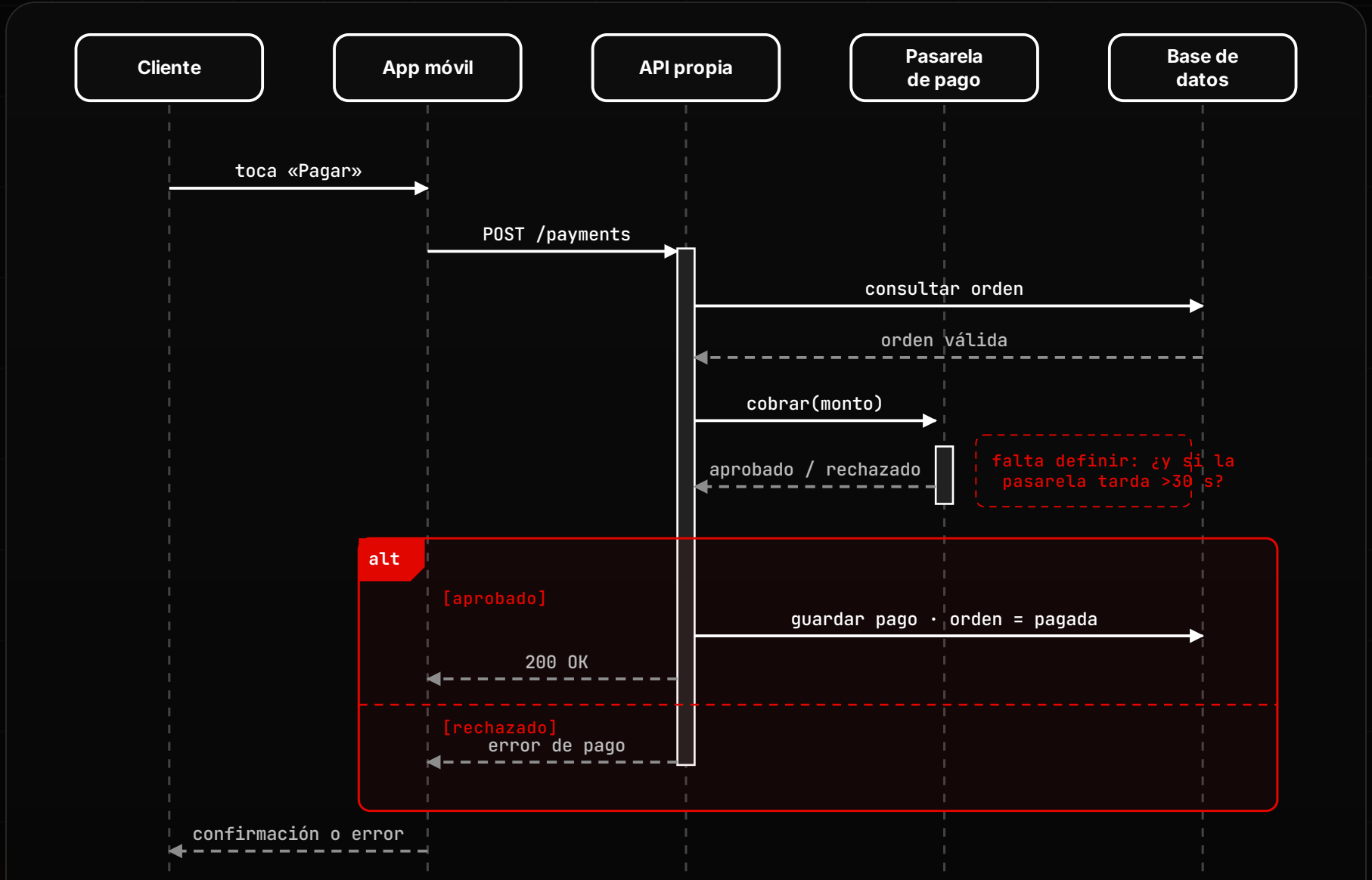
### QUÉ ES Y CUÁNDO USARLO

El diagrama de secuencia muestra cómo se comunican varias partes de un sistema **a lo largo del tiempo**: quién envía un mensaje, a quién y en qué orden. Responde: **¿quién le pide qué a quién, y en qué orden pasa?**

El flujo describe un proceso en general; la secuencia pone el foco en la conversación entre actores y componentes.

› Úsalo para documentar una llamada a una API, un inicio de sesión, un pago con un tercero (pasarela de pago), la comunicación entre microservicios, o para depurar por qué algo falla “en medio” de una cadena de llamadas.

### EJEMPLO DIBUJADO



04 · SECUENCIA

# Elementos y símbolos

### Participante

`Caja arriba con nombre`

Actor o componente (Usuario, App, API, Base de datos)

### Línea de vida

`Línea vertical punteada`

La existencia del participante en el tiempo

### Mensaje

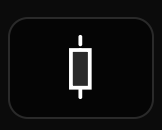
`Flecha horizontal`

Una petición o llamada de un participante a otro

### Respuesta

`Flecha punteada de regreso`

El resultado de un mensaje



### Barra de activación

`Rectángulo delgado sobre la línea de vida`

Cuando el participante está procesando

### Fragmento (alt, opt, loop)

`Recuadro con etiqueta`

Alternativas, partes opcionales o repeticiones

## CÓMO SE LEE

**El tiempo corre de arriba hacia abajo.** Cada flecha es un mensaje y se lee en orden. Las flechas sólidas son peticiones y las punteadas, respuestas. Un recuadro `alt` significa “o pasa esto o pasa lo otro”.

## CÓMO CREARLO PASO A PASO

1. **Elige un escenario concreto:** “iniciar sesión con correo y contraseña”, no “todo el sistema de usuarios”.
2. **Lista los participantes,** de izquierda a derecha en el orden en que aparecen: quien inicia a la izquierda y los sistemas externos a la derecha.
3. **Escribe los mensajes en orden** como llamadas con nombre claro (`POST /login`, `buscarUsuario(email)`).
4. **Agrega las respuestas** y qué datos devuelven.
5. **Agrega los caminos alternativos:** credenciales inválidas, tiempo de espera agotado, servicio caído.
6. **Marca lo asíncrono** (flechas abiertas) si algo no espera respuesta, como enviar un correo de bienvenida.
7. **Revísalo con alguien del equipo** que conozca los componentes: es la mejor forma de encontrar mensajes olvidados.

04 · SECUENCIA

# Ejemplo real

PAGO CON TARJETA EN UNA APP

Participantes: Cliente, App móvil, API propia, Pasarela de pago, Base de datos.

1. El Cliente toca “Pagar” y la App móvil envía `POST /payments` a la API.
2. La API valida la orden y consulta la Base de datos.
3. La API pide el cobro a la Pasarela de pago.
4. La Pasarela responde “aprobado” o “rechazado”.
5. Si aprobó (fragmento `alt`), la API guarda el pago, marca la orden como pagada y responde a la App; si rechazó, responde con el error.
6. La App muestra la confirmación o el mensaje de error.

Este diagrama permite ver, por ejemplo, que **falta definir qué pasa si la pasarela tarda más de 30 segundos** en responder.

## CÓMO MANTENERLO

- ☑ Cuando cambie un contrato de API (nuevos campos, nuevo endpoint, nuevo paso de autenticación), actualiza el diagrama.
- ☑ Cuando agregues un servicio intermedio (una cola de mensajes o un caché), aparece como participante nuevo.
- ☑ Mantén solo los escenarios importantes: login, pago, registro, sincronización. No hagas uno por cada endpoint.
- ☑ Si usas Mermaid o PlantUML, guarda el texto en el repositorio junto a la documentación de la API.

## ERRORES COMUNES

- ✕ Dibujar un escenario demasiado grande con 12 participantes.
- ✕ Mezclar mensajes de distintos casos de uso en un solo diagrama.
- ✕ Olvidar las respuestas y los errores.
- ✕ Nombrar mensajes de forma vaga (“enviar datos”).

IDEA PARA LA VERSIÓN VISUAL

Dibuja las líneas de vida verticales con **iconos** para cada participante, flechas **numeradas** y un recuadro de color para la alternativa aprobado/rechazado. Una animación o numeración paso a paso funciona muy bien aquí.

# Diagrama de arquitectura

05

## ¿Cuáles son las piezas del sistema y cómo se conectan?

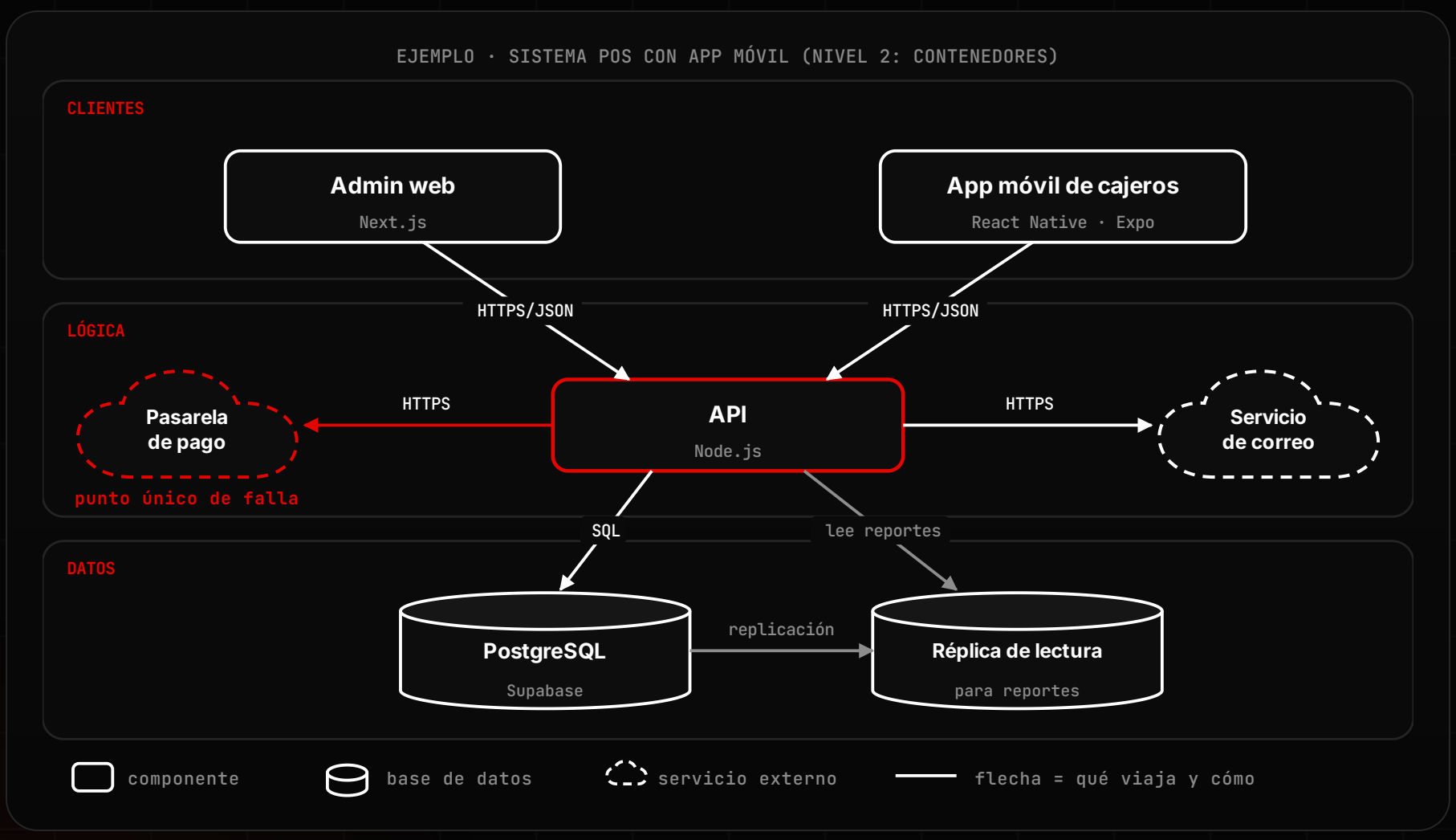
### QUÉ ES Y CUÁNDO USARLO

El diagrama de arquitectura muestra las piezas principales de un sistema (aplicaciones, servicios, bases de datos, servicios externos) y cómo se conectan. Responde: **¿de qué está hecho el sistema y cómo hablan sus partes entre sí?**

Es el mapa general.

› Úsalo para presentar un proyecto a alguien nuevo, para decidir dónde poner cada responsabilidad, para evaluar el impacto de un cambio (“si cambio esto, ¿qué más se afecta?”) o para explicar el sistema a un cliente o inversionista.

### EJEMPLO DIBUJADO



# Elementos y símbolos

### 1. Contexto
Para: todos, incluso no técnicos
El sistema como una caja y las personas y sistemas externos que interactúan con él

### 2. Contenedores
Para: equipo técnico
Aplicaciones, APIs, bases de datos, colas que componen el sistema

### 3. Componentes
Para: desarrolladores del módulo
Módulos internos de un contenedor

### 4. Código
Para: rara vez se dibuja a mano
Clases y funciones (se usa diagrama de clases)

No existe un estándar único; lo importante es ser consistente. Un enfoque muy usado es el **modelo C4**, que dibuja el sistema en niveles de zoom. Los elementos típicos son cajas (componentes), cilindros (bases de datos), nubes (servicios externos), flechas con etiqueta (qué datos o protocolo viajan, como HTTPS o SQL) y agrupaciones (zonas o capas).

## CÓMO SE LEE

Se empieza por quien usa el sistema (usuarios, navegadores, apps) y se siguen las flechas hacia adentro: interfaz, lógica de negocio, datos, servicios externos. **Cada flecha dice qué viaja y cómo.** Las agrupaciones muestran límites: lo que está dentro de una caja grande pertenece a una misma zona.

## CÓMO CREARLO PASO A PASO

1. **Empieza por el nivel 1 (contexto):** ¿quién usa el sistema y con qué sistemas externos se conecta?
2. **Baja al nivel 2:** lista las aplicaciones, APIs, bases de datos y servicios que forman el sistema.
3. **Dibuja las conexiones** con una etiqueta que diga qué viaja y con qué tecnología (“REST/JSON”, “SQL”, “WebSocket”).
4. **Agrupa por responsabilidad:** capa de presentación, capa de lógica, capa de datos.
5. **Marca lo externo y lo crítico:** servicios de terceros (pagos, correo) y las piezas que, si fallan, detienen todo.
6. **Escribe una leyenda** con el significado de formas y colores.
7. **Haz la prueba del recién llegado:** pídele a alguien que no conozca el proyecto que te explique el diagrama; lo que no entienda está mal dibujado.

# Ejemplo real

**ARQUITECTURA DE UN SISTEMA POS CON APP MÓVIL**

Nivel 2 (contenedores): una app web para administración (**Next.js**), una app móvil para cajeros (**React Native con Expo**), una API (**Node.js**) y una base de datos (**PostgreSQL** gestionada en Supabase). Servicios externos: una pasarela de pago y un servicio de correo para enviar recibos.

Conexiones: las dos apps hablan con la API por HTTPS/JSON; la API lee y escribe en la base de datos; la API llama a la pasarela de pago y al servicio de correo. Un módulo de reportes lee de una réplica de la base de datos para no ralentizar las ventas.

El diagrama permite ver que **la pasarela de pago es un punto único de falla** y que conviene definir qué hace el sistema si la pasarela cae (por ejemplo, permitir ventas en efectivo y reintentar después).

## CÓMO MANTENERLO

- ☑ Es el diagrama que más rápido se desactualiza: cada servicio nuevo, base de datos nueva o integración nueva debe reflejarse.
- ☑ Vincúlalo a las decisiones de arquitectura (ADR): cuando se toma una decisión importante, se actualiza el diagrama y se anota por qué.
- ☑ Revísalo al final de cada hito o trimestre y anota la fecha de verificación.
- ☑ Mantén los niveles separados: cambios pequeños tocan solo el nivel 3; cambios grandes tocan el 1 o el 2.

## ERRORES COMUNES

- ✕ Flechas sin etiqueta: nadie sabe qué viaja ni cómo.
- ✕ Mezclar niveles (una función junto a un servidor completo).
- ✕ Diagramas “bonitos” que no coinciden con lo que está en producción.
- ✕ Dibujar la arquitectura que se quiere tener, no la que existe, sin indicar cuál es.

**IDEA PARA LA VERSIÓN VISUAL**

Usa **capas horizontales** con colores (cliente, lógica, datos, externos) y flechas con iconos de protocolo. Muestra la arquitectura en **dos zooms** (contexto y contenedores) como si fuera un mapa que se acerca.

DIAGRAMA 06 DE 10 · CLASES

# Diagrama de clases

## ¿Cómo se organiza el código orientado a objetos?

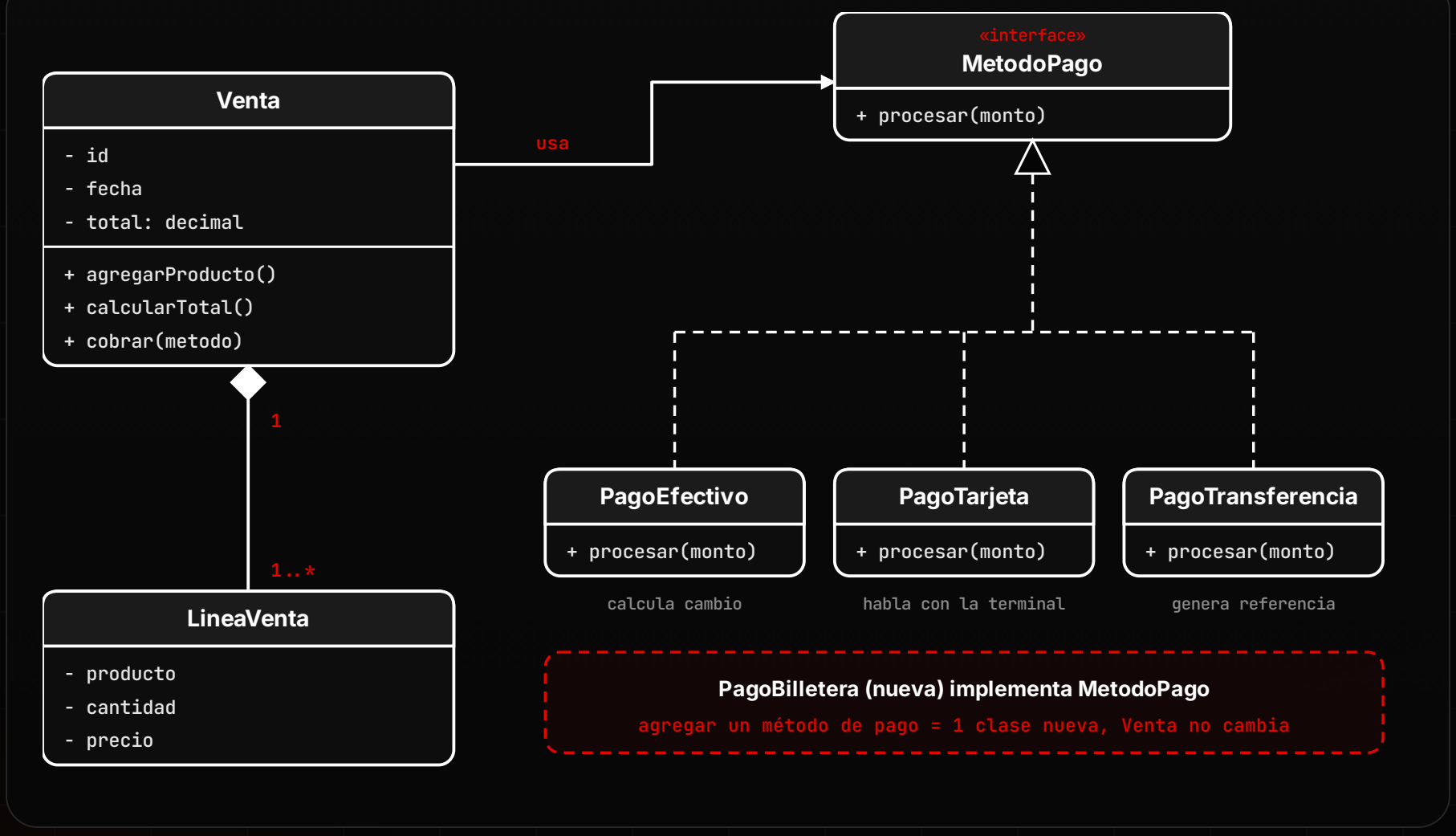
### QUÉ ES Y CUÁNDO USARLO

El diagrama de clases muestra la estructura del código orientado a objetos: las **clases**, sus datos (atributos), sus comportamientos (métodos) y cómo se relacionan entre sí. Responde: **¿cómo está organizado el código y qué responsabilidad tiene cada pieza?**

Se parece al diagrama ER, pero el ER describe datos guardados y el de clases describe el código en ejecución.

› Úsalo al diseñar un módulo antes de programarlo, al explicar una librería o dominio complejo, para revisar si las responsabilidades están bien repartidas o al refactorizar código difícil de entender.

### EJEMPLO DIBUJADO



06 · CLASES

# Elementos y símbolos

**+** **Público**
`+ agregarProducto()`
Se puede usar desde fuera

**-** **Privado**
`- total: decimal`
Solo dentro de la clase

**#** **Protegido**
`# validar()`
La clase y sus hijas

**Asociación**
`Línea simple · Cliente — Orden`
Una clase conoce a otra

**Herencia**
`Triángulo vacío · Cajero › Usuario`
“Es un”

**Agregación**
`Rombo vacío · Equipo — Jugador`
“Tiene un”; las partes pueden existir solas

**Composición**
`Rombo lleno · Orden — LineaOrden`
“Está hecho de”; las partes mueren con el todo

**Implementación**
`Punteada con triángulo · Efectivo › MetodoPago`
Implementa una interfaz

**1..*** **Multiplicidad**
`(1, 0..*, 1..*) · Orden 1 — 1..* LineaOrden`
Cuántos objetos se relacionan

Cada clase es una caja con **tres secciones: nombre, atributos y métodos.**

## CÓMO SE LEE

Se lee cada relación como una frase: “Un Cajero **es un** Usuario” (herencia), “una Orden **está compuesta de** líneas de orden” (composición), “una Orden **usa** un MetodoPago” (asociación). Los signos `+` y `-` indican qué parte de la clase se puede tocar desde fuera.

## CÓMO CREARLO PASO A PASO

1. **Define el alcance:** un módulo o un caso de uso, no todo el sistema.
2. **Identifica las clases** a partir de los sustantivos del problema (Orden, Producto, Pago, Cliente).
3. **Asigna responsabilidades:** cada clase debe hacer una cosa bien. Si una clase tiene 25 métodos, probablemente son varias clases.
4. **Agrega atributos y métodos principales.** No pongas todos los getters y setters; solo lo que explique el diseño.
5. **Dibuja las relaciones** y elige la correcta: herencia solo si realmente “es un”; composición si las partes dependen del todo.
6. **Busca interfaces** donde varias clases comparten comportamiento (MetodoPago con Efectivo, Tarjeta, Transferencia).
7. **Revisa el acoplamiento:** si todas las clases se conocen entre sí, el diseño es frágil.

# Ejemplo real

MÓDULO DE PAGOS DE UN POS

Interfaz **MetodoPago** con el método `procesar(monto)`. Implementaciones: **PagoEfectivo** (calcula cambio), **PagoTarjeta** (se comunica con la terminal) y **PagoTransferencia** (genera una referencia).

La clase **Venta** (atributos: id, fecha, total; métodos: `agregarProducto()`, `calcularTotal()`, `cobrar(metodo)`) está compuesta de varias **LineaVenta** (producto, cantidad, precio) y usa un MetodoPago.

La ventaja: agregar un nuevo método de pago (por ejemplo, **billetera digital**) solo requiere una clase nueva que implemente la interfaz, **sin modificar Venta**.

## CÓMO MANTENERLO

- ☑ Los diagramas de clases se desactualizan muy rápido si se dibujan a mano. Lo ideal es generarlos desde el código con herramientas, o limitarse a diagramas del diseño central.
- ☑ Actualiza el diagrama cuando cambien las responsabilidades o las relaciones, no por cada método nuevo.
- ☑ Úsalo como apoyo en revisiones de código y refactorizaciones: si el cambio rompe el diseño del diagrama, hay que discutirlo.
- ☑ En TypeScript y lenguajes modernos, a veces un diagrama de tipos o módulos es más útil que uno de clases estrictas.

## ERRORES COMUNES

- ✕ Usar herencia para todo en lugar de composición e interfaces.
- ✕ Clases “Dios” que lo saben y lo hacen todo.
- ✕ Incluir todos los atributos y métodos y dejar el diagrama ilegible.
- ✕ Dibujarlo una vez y no volver a tocarlo mientras el código cambia.

IDEA PARA LA VERSIÓN VISUAL

Dibuja las cajas de clase con **colores por responsabilidad** (dominio, servicios, interfaces) y una pequeña leyenda de los símbolos de relación. Un **antes y un después** de un refactor es un excelente contenido visual.

# Diagrama de casos de uso

07

## ¿Qué puede hacer cada tipo de usuario con el sistema?

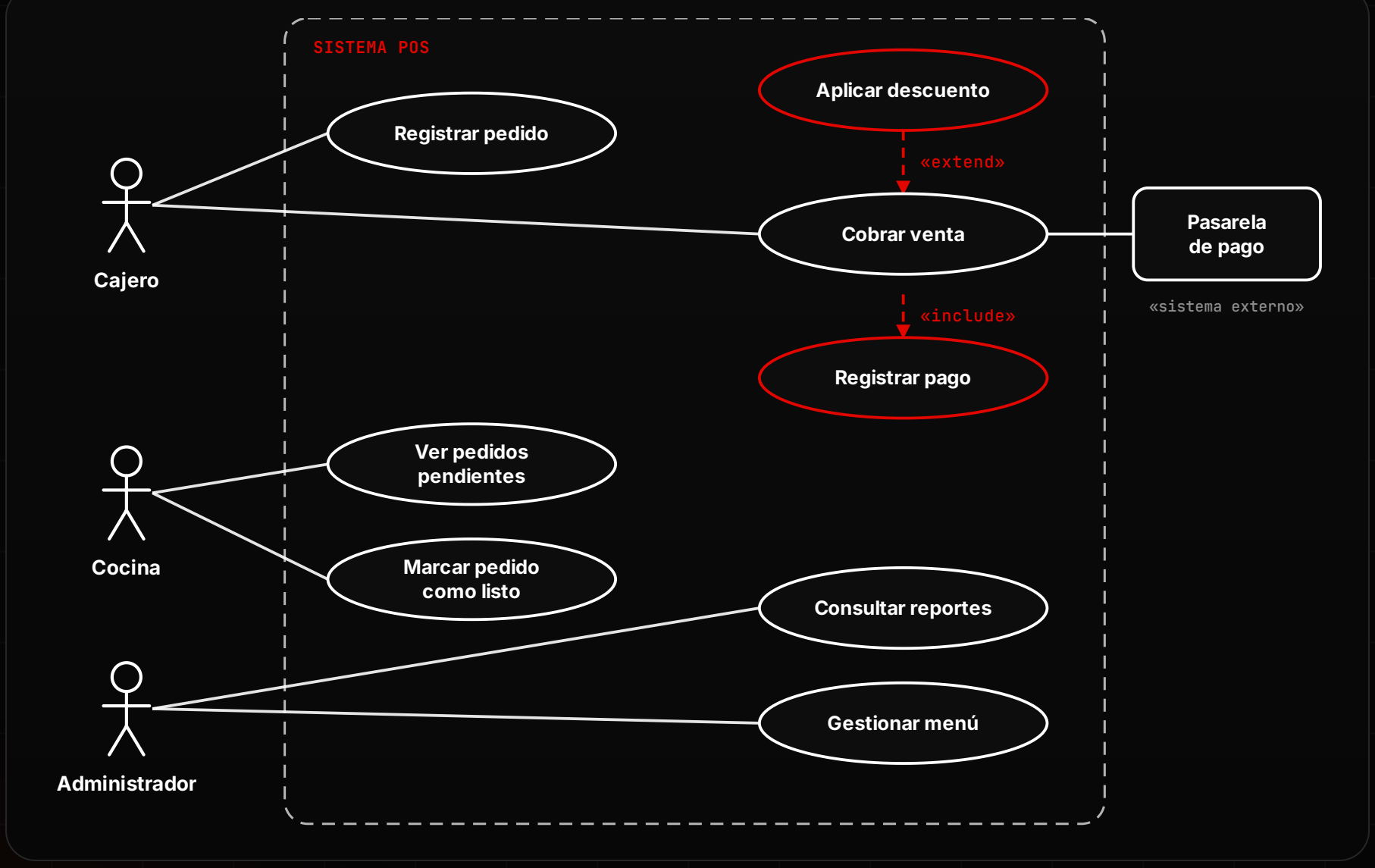
### QUÉ ES Y CUÁNDO USARLO

El diagrama de casos de uso muestra **qué puede hacer cada tipo de usuario** (los actores) con el sistema (los casos de uso), sin explicar cómo lo hace por dentro. Responde: **¿quién usa el sistema y para qué?**

Es el diagrama más cercano al cliente.

› Úsalo al inicio del proyecto para definir el alcance, para acordar con el cliente qué incluye el sistema y qué no, para repartir funcionalidades entre versiones o para armar los permisos por rol.

### EJEMPLO DIBUJADO



# Elementos y símbolos

### Actor
Muñeco de palo

Persona o sistema externo que interactúa (Cajero, Administrador, Pasarela de pago)

### Caso de uso
Óvalo

Una meta que el actor quiere lograr (“Cobrar venta”)

### Límite del sistema
Rectángulo grande

Todo lo que está dentro es parte del sistema

### Asociación
Línea simple

El actor participa en ese caso de uso

### Include
Flecha punteada «include»

Un caso siempre incluye a otro (Cobrar venta incluye Registrar pago)

### Extend
Flecha punteada «extend»

Un caso opcional que amplía a otro (Aplicar descuento extiende a Cobrar venta)

### Generalización
Flecha con triángulo

Un actor o caso es una versión especializada de otro

## CÓMO SE LEE

Se busca un actor, se siguen sus líneas y se leen los óvalos con los que se conecta: **esa es la lista de lo que esa persona puede hacer.** Lo que está fuera del rectángulo no es parte del sistema (por ejemplo, el banco).

## CÓMO CREARLO PASO A PASO

1. **Identifica los actores:** ¿qué tipos de usuarios hay y qué sistemas externos interactúan? Cuenta roles, no personas.
2. **Para cada actor, pregunta qué quiere lograr.** Escribe cada meta como verbo + objeto: “Registrar producto”, “Consultar reporte”.
3. **Dibuja el límite del sistema** y coloca los casos de uso dentro.
4. **Conecta actores y casos de uso.**
5. **Revisa el nivel de detalle:** un caso de uso es una meta completa con valor para el actor, no un clic (“Cobrar venta”, no “Presionar el botón de pagar”).
6. **Usa include y extend con moderación:** solo cuando un paso se repite en muchos casos o es claramente opcional.
7. **Valida con el cliente:** si dice “falta que yo pueda...”, acabas de descubrir un requisito antes de programarlo.

# Ejemplo real

**SISTEMA POS PARA UN LOCAL DE COMIDA**

Actores: Cajero, Cocina, Administrador y el sistema externo Pasarela de pago.

- **Cajero:** Registrar pedido, Cobrar venta, Imprimir ticket, Cancelar pedido.
- **Cocina:** Ver pedidos pendientes, Marcar pedido como listo.
- **Administrador:** Gestionar menú y precios, Consultar reportes de ventas, Gestionar usuarios, Abrir y cerrar caja.
- **Pasarela de pago:** participa en Cobrar venta con tarjeta.

Aplicar descuento **extiende** a Cobrar venta (es opcional) y Registrar pago está **incluido** en Cobrar venta (siempre ocurre). Con este diagrama el dueño del local puede confirmar qué incluye la primera versión **antes de que se escriba una línea de código**.

## CÓMO MANTENERLO

- ☑ Cada vez que se agregue una funcionalidad o un rol, se actualiza el diagrama; es la base de la lista de requisitos.
- ☑ Úsalo para el control de alcance: si un cliente pide algo que no está en el diagrama, es un cambio de alcance y se negocia.
- ☑ Relaciónalo con las historias de usuario o tickets: cada caso de uso debe poder trazarse a trabajo concreto.
- ☑ Marca qué casos de uso ya están implementados y cuáles son de versiones futuras.

## ERRORES COMUNES

- ✕ Casos de uso demasiado pequeños (un clic) o demasiado grandes (“Administrar todo”).
- ✕ Confundir actores con personas concretas en lugar de roles.
- ✕ Usar include y extend en todas partes hasta que nadie entienda el diagrama.
- ✕ Describir cómo funciona por dentro en lugar de qué logra el actor.

**IDEA PARA LA VERSIÓN VISUAL**

Usa el **muñeco de palo de animare.dev** como actor, con un color distinto para cada rol, y óvalos agrupados dentro de un marco que represente el sistema. Funciona bien una versión “antes y después” que muestre cómo cambian los permisos entre roles.

DIAGRAMA 08 DE 10 · DESPLIEGUE

# Diagrama de despliegue

08

## ¿Dónde corre cada pieza en servidores y dispositivos?

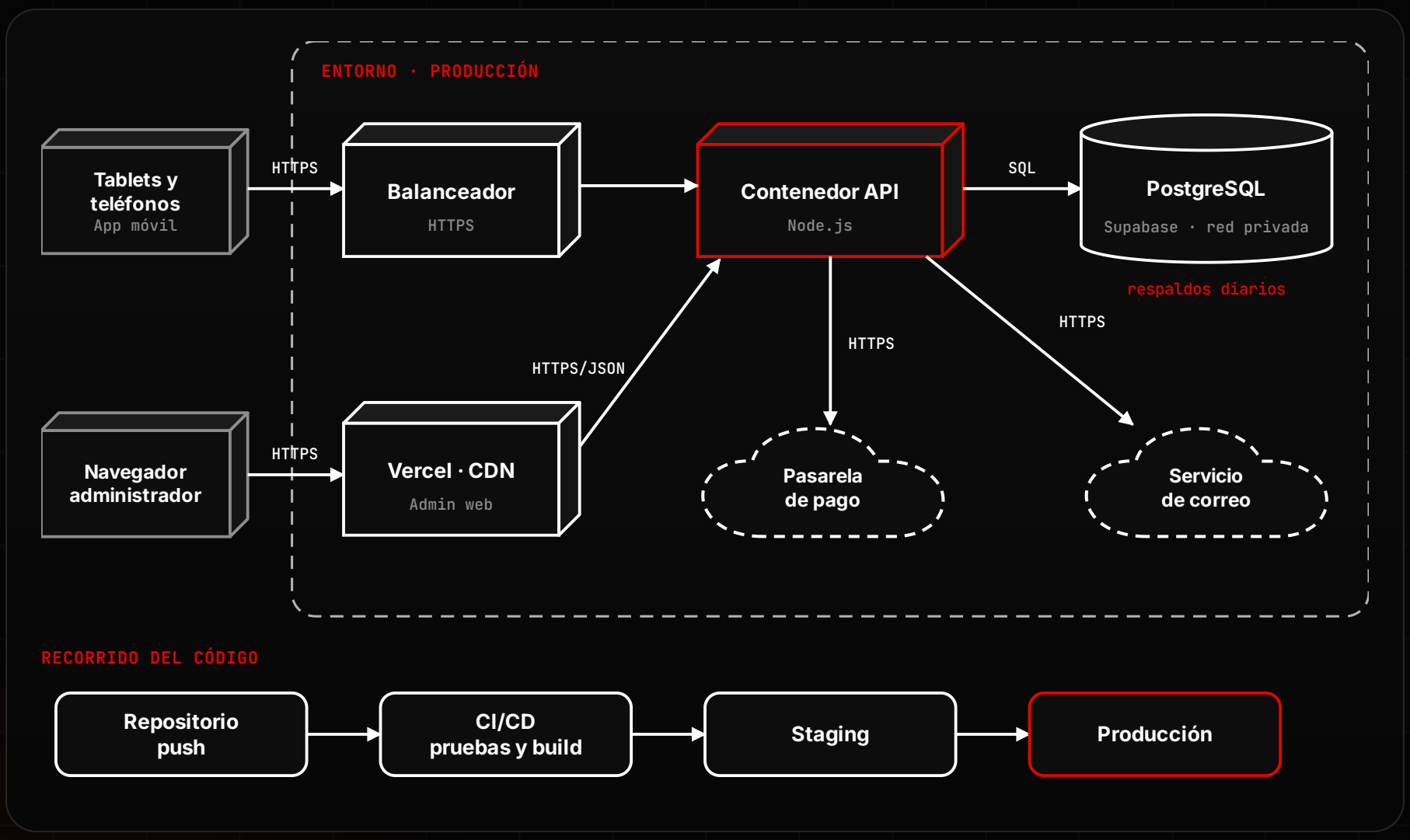
### QUÉ ES Y CUÁNDO USARLO

El diagrama de despliegue muestra **dónde corre físicamente cada parte del software:** en qué servidores, contenedores, dispositivos o servicios en la nube, y cómo se conectan. Responde: **¿dónde vive cada pieza cuando el sistema está funcionando?**

La arquitectura dice de qué está hecho el sistema; el despliegue dice en qué máquinas y entornos corre.

› Úsalo para planear la infraestructura, explicar cómo se publica una versión, decidir qué se escala y qué no, documentar entornos (desarrollo, pruebas, producción) o preparar un plan de recuperación ante fallos.

### EJEMPLO DIBUJADO



08 · DESPLIEGUE

# Elementos y símbolos

### Nodo
Cubo o caja en 3D

Un servidor, máquina virtual, contenedor o dispositivo

### Artefacto
Rectángulo con icono de documento

Lo que se despliega: una app, un binario, una imagen de Docker

### Entorno
Caja grande que agrupa nodos

Desarrollo, pruebas, producción

### Conexión
Línea etiquetada

Protocolo o canal entre nodos (HTTPS, SQL, SSH)

### Servicio gestionado
Nube o icono del proveedor

Base de datos, almacenamiento o colas en un servicio como Supabase, Vercel o AWS

## CÓMO SE LEE

Se leen **de afuera hacia adentro:** primero el usuario y su dispositivo, luego la entrada a la infraestructura (DNS, balanceador, CDN), después los servidores de aplicación y por último los datos. Cada caja dice dónde corre algo y cada línea, cómo se llega a él.

## CÓMO CREARLO PASO A PASO

1. **Parte del diagrama de arquitectura** y pregunta de cada pieza: ¿dónde corre?
2. **Define los entornos:** desarrollo, pruebas (staging) y producción. Dibuja al menos producción.
3. **Dibuja los nodos** y coloca dentro los artefactos que corren en cada uno.
4. **Agrega las conexiones** con protocolo y puerto relevante.
5. **Agrega la red y la seguridad:** firewall, red privada, qué es público y qué no. La base de datos casi nunca debe ser accesible desde internet.
6. **Marca la redundancia y la escala:** ¿más de una instancia? ¿réplica de la base de datos? ¿respaldos y dónde se guardan?
7. **Añade el proceso de publicación:** de dónde sale el código (repositorio), qué lo construye (CI/CD) y a dónde llega.

# Ejemplo real

**DESPLIEGUE DE UN SISTEMA POS EN LA NUBE · PRODUCCIÓN**

- **Dispositivos de cajeros:** tablets y teléfonos con la app móvil instalada, conectados por HTTPS.
- **Frontend de administración:** aplicación web desplegada en un servicio como Vercel, servida por CDN.
- **API:** contenedor Node.js en un servidor o servicio de aplicaciones, detrás de un balanceador con HTTPS.
- **Base de datos:** PostgreSQL gestionado (por ejemplo, Supabase), en red privada, con respaldos diarios automáticos.
- **Servicios externos:** pasarela de pago y servicio de correo, accedidos por HTTPS desde la API.

Cada despliegue sale del repositorio: se hace un push, el CI/CD ejecuta pruebas, construye y publica primero en staging y después en producción. Para un POS en un local físico: **¿qué pasa si se cae internet?** El diagrama puede mostrar un modo sin conexión en la tablet que sincroniza al reconectarse.

## CÓMO MANTENERLO

- ☑ Actualízalo con cada cambio de infraestructura: nuevo servidor, cambio de proveedor, nuevo entorno o servicio.
- ☑ Si usas infraestructura como código (Terraform, Docker Compose, archivos de configuración), el diagrama debe corresponder con esos archivos y revisarse al cambiarlos.
- ☑ Ten un diagrama por entorno cuando sean muy distintos (producción tiene réplicas y desarrollo no).
- ☑ Revisa que incluya lo que se necesita en una emergencia: dónde están los respaldos, cómo se restaura y quién tiene acceso.

## ERRORES COMUNES

- ✕ Mostrar los servidores pero no qué corre en cada uno.
- ✕ Olvidar la parte de seguridad y los respaldos.
- ✕ No diferenciar los entornos, de modo que nadie sabe qué está en producción.
- ✕ No actualizarlo tras una migración de proveedor.

**IDEA PARA LA VERSIÓN VISUAL**

Dibuja el sistema como un **mapa de capas** con iconos de dispositivos, nube y servidores. Añade una animación del recorrido del código desde el repositorio hasta producción, y un segundo escenario con la “caída de internet” en el local.

DIAGRAMA 09 DE 10 · GANTT

# Diagrama de Gantt

## ¿Qué se hace, cuándo y en qué orden de dependencia?

### QUÉ ES Y CUÁNDO USARLO

El diagrama de Gantt muestra las tareas de un proyecto como **barras sobre una línea de tiempo:** cuándo empieza cada una, cuánto dura y de cuáles depende. Responde: **¿qué se hace, cuándo, y qué tiene que terminar antes de qué?**

Es el diagrama de la planeación.

› Úsalo para estimar cuánto tarda un proyecto, repartir trabajo entre personas, comunicar fechas al cliente, detectar riesgos de retraso o seguir el avance real contra lo planeado.

### EJEMPLO DIBUJADO

09 · GANTT

# Elementos y símbolos

### Tarea
Barra horizontal

Trabajo con inicio y fin; su largo es la duración

### Hito
Rombo

Un momento clave sin duración (“Entrega de la v1”)

### Dependencia
Flecha entre barras

Una tarea no puede empezar hasta que otra termine

### Línea de hoy
Línea vertical

La fecha actual, para comparar con lo planeado

### Avance
Parte rellena de la barra

Porcentaje completado

### Ruta crítica
Barras resaltadas

La cadena de tareas que define la fecha final; si una se retrasa, todo se retrasa

### Responsable
Etiqueta o color

Quién hace la tarea

## CÓMO SE LEE

El eje horizontal es el tiempo. Cada fila es una tarea. Se miran las flechas para saber qué depende de qué y se busca la cadena más larga: **es la ruta crítica.** La línea de hoy muestra si el proyecto va adelantado o atrasado.

## CÓMO CREARLO PASO A PASO

1. **Define el objetivo** y la fecha de entrega deseada.
2. **Divide el proyecto en tareas** de tamaño manejable (ni de 10 minutos ni de 3 meses; idealmente de 1 a 10 días).
3. **Estima la duración** de cada tarea. Pídele la estimación a quien la hará y agrega margen para imprevistos.
4. **Define las dependencias:** ¿qué debe estar listo antes de empezar cada tarea?
5. **Coloca las barras** respetando las dependencias y la disponibilidad de las personas (nadie puede hacer 3 tareas a la vez).
6. **Marca los hitos:** entregas, revisiones con el cliente, publicaciones.
7. **Identifica la ruta crítica** y revisa si la fecha final es realista. Si no, recorta alcance, agrega personas a tareas paralelizables o mueve la fecha.

# Ejemplo real

**PRIMER PROYECTO DE POS A MEDIDA PARA UN CLIENTE · 8 SEMANAS**

Proyecto de 8 semanas con entrega y capacitación al final. La **ruta crítica** pasa por requisitos, diseño, ventas, integración con la terminal, pruebas y entrega.

El inventario puede avanzar en paralelo, así que un retraso pequeño ahí no mueve la fecha. En cambio, **un retraso en la integración con la terminal retrasa toda la entrega.**

## CÓMO MANTENERLO

- ☑ Un Gantt se actualiza con frecuencia: al menos una vez por semana, marcando el avance real.
- ☑ Cuando algo se retrasa, replantea las fechas posteriores en lugar de fingir que todo sigue igual.
- ☑ Guarda la línea base (el plan original) para comparar y aprender de tus estimaciones.
- ☑ Evita el nivel de detalle excesivo: es mejor un Gantt de 20 a 40 tareas bien mantenido que uno de 300 que nadie actualiza.
- ☑ Para equipos ágiles, usa el Gantt solo para los hitos grandes y deja el detalle a tableros por sprint.

## ERRORES COMUNES

- ✕ Estimaciones optimistas sin margen para errores.
- ✕ No marcar dependencias.
- ✕ Hacerlo una vez y nunca actualizarlo.
- ✕ Asignar a una persona varias tareas simultáneas al 100 %.
- ✕ Usarlo como herramienta de presión en lugar de herramienta de comunicación.

**IDEA PARA LA VERSIÓN VISUAL**

Dibuja las barras en **colores por responsable**, resalta la ruta crítica en **rojo** y coloca la línea de hoy. Un buen contenido sería mostrar “el plan” y “lo que realmente pasó” para explicar por qué se retrasan los proyectos.

# Diagrama de red

## ¿Cómo se conectan los equipos y por dónde viaja la información?

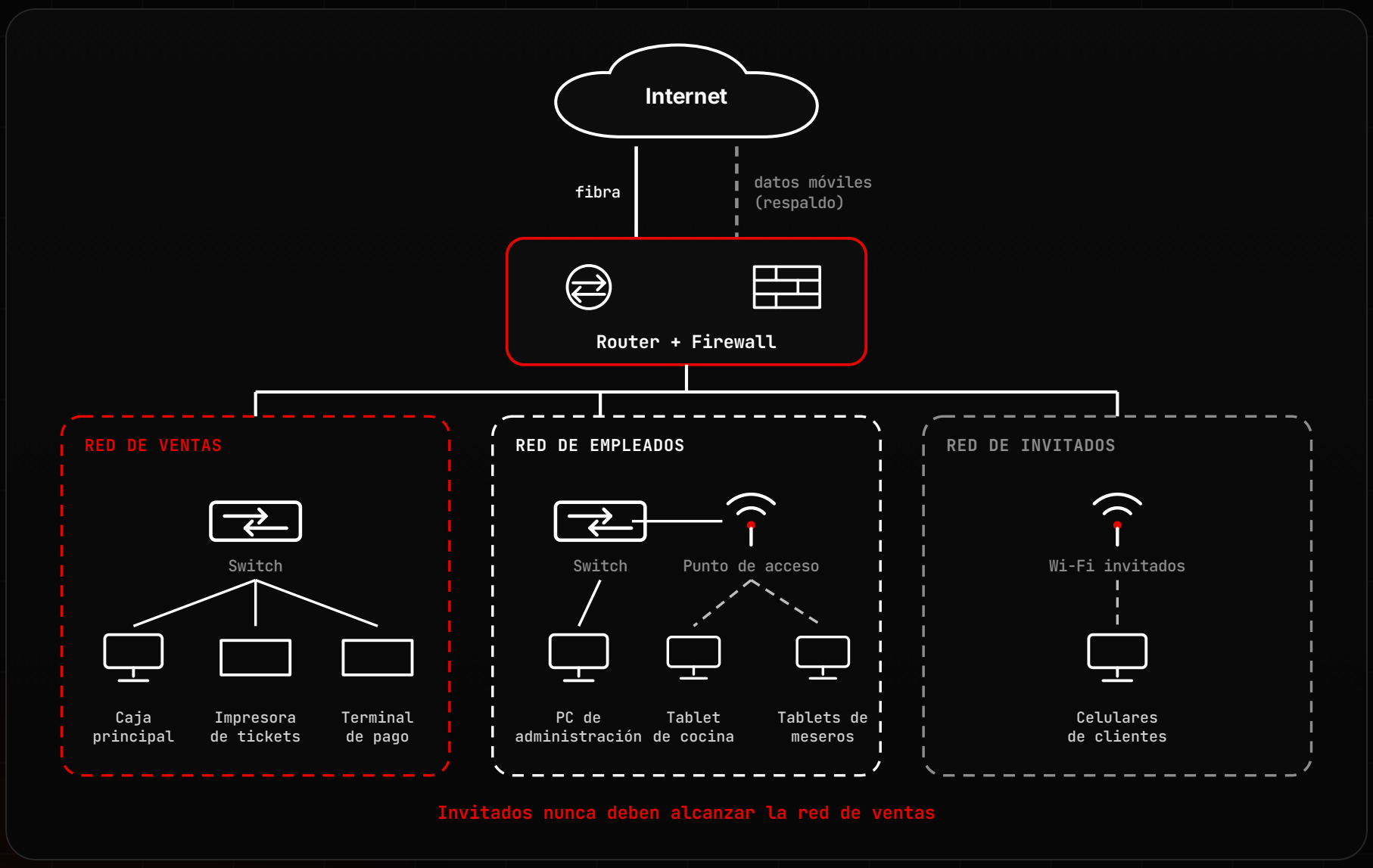
### QUÉ ES Y CUÁNDO USARLO

El diagrama de red muestra los **equipos físicos y lógicos** de una red (computadoras, servidores, routers, switches, impresoras, puntos de acceso, firewalls) y cómo se conectan entre sí y con internet. Responde: **¿cómo están conectados los equipos y por dónde viaja la información?**

El despliegue habla del software y dónde corre; el de red habla del cableado y los dispositivos que lo comunican.

› Úsalo para instalar o diagnosticar una red en un local u oficina, planear la seguridad (qué equipo puede hablar con cuál), documentar direcciones IP y segmentos, o entender por qué algo no se conecta.

### EJEMPLO DIBUJADO



# Elementos y símbolos

### Router
Círculo con flechas
Conecta la red local con internet y entre redes

### Switch
Caja con flechas cruzadas
Conecta varios equipos cableados dentro de una red

### Punto de acceso (AP)
Icono de antena
Da conexión inalámbrica (Wi-Fi)

### Firewall
Muro de ladrillos
Filtra el tráfico permitido y bloqueado

### Servidor
Torre o rack
Ofrece servicios (archivos, base de datos, impresión)

### Cliente
Computadora, tablet, teléfono
Equipos de los usuarios

### Internet / nube
Nube
Redes externas

### Enlace
Línea continua = cable · punteada = Wi-Fi
Conexión; se etiqueta con velocidad o tipo de cable

## CÓMO SE LEE

Se empieza en internet (arriba o a la izquierda), se pasa por el router y el firewall y se baja hacia los switches y puntos de acceso, hasta llegar a los equipos finales. **Cada línea representa una conexión real.** Las etiquetas indican direcciones IP, rangos y velocidades.

## CÓMO CREARLO PASO A PASO

1. **Haz un inventario** de todos los equipos que se conectan y dónde están físicamente.
2. **Define la entrada a internet:** proveedor, módem, router principal. Anota el tipo de conexión y la velocidad contratada.
3. **Dibuja el núcleo:** router, firewall y switch principal.
4. **Agrega los segmentos:** separa las redes por función (red de cajas y POS, de empleados y de invitados). Cada una con su rango de IP.
5. **Conecta los equipos finales** a su switch o punto de acceso, indicando si es por cable o Wi-Fi.
6. **Agrega etiquetas:** nombre del equipo, IP, tipo de enlace, puertos importantes.
7. **Marca la seguridad y la redundancia:** ¿hay firewall? ¿qué pasa si cae el enlace principal? ¿hay un segundo proveedor o datos móviles de respaldo?
8. **Verifica en el sitio:** recorre el lugar físicamente y confirma que el diagrama coincide con lo instalado.

# Ejemplo real

**RED DE UN LOCAL DE COMIDA CON POS**

- **Internet:** un enlace de fibra del proveedor, con datos móviles como respaldo.
- **Router y firewall** en la entrada, que separan tres redes: Red de ventas (POS, terminal de pago, impresora de tickets), Red de empleados (tablet de cocina, computadora de administración) y Red de invitados (Wi-Fi para clientes, sin acceso a las otras).
- **Switch** que conecta por cable la caja principal, la impresora de tickets y la terminal de pago.
- **Punto de acceso** que da Wi-Fi a la tablet de cocina y a las tablets de meseros.

Este diagrama muestra que **el Wi-Fi de invitados nunca debe poder alcanzar la red del POS.** También sirve para diagnosticar: si la impresora no imprime, indica que hay que revisar el cable al switch y su dirección IP antes que el software.

## CÓMO MANTENERLO

- ☑ Actualízalo cada vez que se agregue, mueva o retire un equipo, o cambie un proveedor de internet.
- ☑ Guarda la tabla de direcciones IP y su dueño (qué equipo tiene cada dirección) junto al diagrama.
- ☑ Revísalo después de cualquier cambio de seguridad (nuevas reglas de firewall, nuevas redes).
- ☑ Entrega una copia al cliente o responsable del local; en una emergencia es el primer documento que necesitará quien dé soporte.
- ☑ Anota fecha y versión, y quién hizo el último cambio.

## ERRORES COMUNES

- ✕ No incluir direcciones IP ni etiquetas de los enlaces.
- ✕ Mezclar el diagrama físico (cables y equipos) con el lógico (redes y rangos) sin distinguirlos.
- ✕ Olvidar documentar los equipos “escondidos” (repetidores, módems de respaldo).
- ✕ No actualizarlo tras cambios que hizo el proveedor de internet o un técnico externo.

**IDEA PARA LA VERSIÓN VISUAL**

Dibuja el local en **vista superior** con la red superpuesta: cada red de un color (ventas, empleados, invitados) y los equipos con iconos reconocibles. Muestra un segundo diagrama con el escenario “se cae el internet” y el enlace de respaldo activándose.

CASO INTEGRADOR

# Un proyecto real usando los 10 diagramas

Los diagramas no se usan todos a la vez ni en cualquier orden: cada fase necesita unos diferentes. Un sistema de punto de venta para un local de comida.

## FASE 1 Entender qué se va a construir

**Casos de uso** — Se acuerda con el dueño qué puede hacer cada rol (cajero, cocina, administrador). Resultado: el alcance de la versión 1.

**Flujo** — Se dibuja cómo funciona hoy el proceso de tomar pedidos y cobrar. Aparecen pasos que se pueden eliminar y errores frecuentes (pedidos que se pierden entre mesa y cocina).

## FASE 2 Diseñar la solución

**Arquitectura** — Se decide qué piezas tendrá el sistema (app móvil, panel web, API, base de datos) y qué servicios externos usa.

**Entidad-relación** — Se diseña la base de datos (productos, ventas, detalle de ventas, pagos, usuarios).

**Estados** — Se define el ciclo de vida de un pedido (Creado, En cocina, Listo, Entregado, Pagado, Cancelado) y qué acciones se permiten en cada estado.

**Clases** — Se diseña el módulo más complejo, como pagos y descuentos, para que agregar un método de pago nuevo sea fácil.

CASO INTEGRADOR

# Fases 3 y 4 · y cómo se conectan

## FASE 3 Planear y construir

**Gantt** Se reparten las tareas por semanas, se identifica la ruta crítica y se fijan las fechas de demostración y entrega.

**Secuencia** Para las partes delicadas (cobro con tarjeta, sincronización sin conexión, impresión de ticket) se dibuja la conversación entre app, API y servicios externos antes de programarla, y se descubren los errores a tiempo.

## FASE 4 Poner en marcha y dar soporte

**Despliegue** Se documenta dónde corre cada pieza, cómo se publica una versión nueva y dónde están los respaldos.

**Red** Se documenta la red del local (equipos, redes separadas, direcciones IP) para instalar y para diagnosticar problemas.

## CÓMO SE CONECTAN ENTRE SÍ

| DIAGRAMA ORIGEN | ALIMENTA A | CÓMO |
|---|---|---|
| **Casos de uso** | Flujo, secuencia | Cada caso de uso importante tiene un flujo o una secuencia que lo detalla |
| **Flujo** | Estados | Las decisiones del proceso revelan los estados de un objeto |
| **Arquitectura** | Despliegue, secuencia | Cada contenedor se ubica en un servidor y participa en las conversaciones |
| **Entidad-relación** | Clases | Las entidades principales suelen convertirse en clases del dominio |
| **Gantt** | Todos | El tiempo disponible decide qué diagramas vale la pena hacer en detalle |
| **Despliegue** | Red | Los nodos físicos del despliegue se conectan por la red del local |

LECCIÓN PRINCIPAL

# No se trata de dibujar los 10 por obligación

## Se trata de elegir el diagrama que responde la pregunta que tienes en ese momento.

| | |
|---|---|
| ¿Qué construimos? | Casos de uso |
| ¿Cómo funciona? | Flujo · Secuencia · Estados |
| ¿Cómo se guarda? | ER |
| ¿Cómo se organiza? | Arquitectura · Clases |
| ¿Cuándo lo hacemos? | Gantt |
| ¿Dónde corre? | Despliegue · Red |

CIERRE

# Qué diagrama elegir según la situación

| SI NECESITAS… | USA | FASE TÍPICA |
|---|---|---|
| Acordar con el cliente qué hará el sistema | Casos de uso | Inicio |
| Explicar un proceso o algoritmo paso a paso | Flujo | Inicio y diseño |
| Controlar el ciclo de vida de un objeto | Estados | Diseño |
| Diseñar o documentar la base de datos | Entidad-relación | Diseño |
| Ver el mapa general del sistema | Arquitectura | Diseño |
| Organizar el código orientado a objetos | Clases | Diseño y refactor |
| Entender una conversación entre componentes | Secuencia | Diseño y depuración |
| Planear fechas, tareas y dependencias | Gantt | Planeación y seguimiento |
| Documentar dónde corre cada pieza | Despliegue | Entrega y operación |
| Documentar equipos y conexiones | Red | Instalación y soporte |

CIERRE

# Checklist para revisar cualquier diagrama

- [x] Responde una sola pregunta clara, escrita en el título o la descripción.
- [x] Tiene título, fecha, versión y autor.
- [x] Todos los elementos están en el mismo nivel de detalle.
- [x] Los nombres coinciden con los del código, la base de datos y el equipo.
- [x] Tiene leyenda si usa símbolos o colores.
- [x] Una persona que no conoce el proyecto puede explicarlo sin ayuda.
- [x] Incluye los casos de error y excepción, no solo el camino feliz.
- [x] Tiene un responsable asignado para mantenerlo.
- [x] Está guardado junto al código o en un lugar que el equipo consulta.
- [x] Se revisó contra la realidad en el último hito o trimestre.

## RUTINA DE MANTENIMIENTO SUGERIDA

| FRECUENCIA | QUÉ HACER |
|---|---|
| **En cada cambio relevante** | Actualizar el diagrama afectado dentro del mismo pull request o tarea |
| **Cada semana** | Actualizar el Gantt con el avance real |
| **Cada hito o trimestre** | Revisar arquitectura, despliegue, ER y red contra la realidad; anotar la fecha |
| **Al cerrar un proyecto** | Archivar la versión final y los diagramas que ya no apliquen |

CIERRE

# Glosario

**ACTOR**
Persona o sistema externo que interactúa con el sistema.

**API**
Forma en que un programa pide cosas a otro.

**ARTEFACTO**
Lo que se despliega, como una app o una imagen de contenedor.

**CARDINALIDAD**
Cuántos elementos de una entidad se relacionan con cuántos de otra (1:1, 1:N, N:M).

**CI/CD**
Proceso automático que prueba y publica el código.

**CLAVE PRIMARIA (PK)**
Dato que identifica de forma única cada registro.

**CLAVE FORÁNEA (FK)**
Dato que apunta a la clave primaria de otra tabla.

**COMPOSICIÓN**
Relación en la que las partes dejan de existir cuando desaparece el todo.

**ENTIDAD**
Algo sobre lo que se guardan datos; suele ser una tabla.

**ESTADO**
Situación estable en la que puede estar un objeto.

**HITO**
Momento clave de un proyecto sin duración propia.

**INTERFAZ**
Contrato que dice qué métodos debe tener una clase, sin decir cómo los hace.

**LÍNEA DE VIDA**
Representación de un participante a lo largo del tiempo en un diagrama de secuencia.

**NORMALIZACIÓN**
Organizar los datos para no repetirlos.

**RUTA CRÍTICA**
Cadena de tareas que determina la fecha final de un proyecto.

**TRANSICIÓN**
Cambio de un estado a otro provocado por un evento.

PARA SEGUIR APRENDIENDO

# Dibuja los 10 de un mismo proyecto pequeño.

Con esta base puedes profundizar cada diagrama con ejemplos de tus propios proyectos. La comparación hace evidente **para qué sirve cada uno**.

GUÍA DE

ANIMARE.dev