TIPRE

Sistema de Premios

Documentación técnica, comercial y operativa

Producto	**Sistema de Premios — sorteo con voucher QR firmado**
Piezas	**Backend (Spring Boot) · Landing móvil (Preact) · Gestor web (React)**
Stack	**Java 17 · Spring Boot 4 · SQL Server · Flyway · Vite**
Cliente	**Supermercados Caracol — Sorteo Aniversario 50 años**
Proveedor	**Tipre**
Audiencia	**Instalación, operación y soporte del sistema**

chance de sorteo: el ticket imprime un **voucher con QR firmado**, el cliente lo escanea desde su celular, ingresa su DNI y participa de sorteos semanales. En paralelo, cada participación construye un **padrón propio de clientes con consentimiento** para futuras acciones comerciales.

Ver toda la documentación en PDF

De un vistazo

- **Cliente:** Supermercados Caracol · **Proveedor:** Tipre
- **Campaña de referencia:** "Sorteo 50 años" — 100 premios, vigencia 01/09 → 09/10/2026
- **Piezas construidas:** backend de participación, landing móvil, gestor web
- **Sin dependencia de nube:** corre en un servidor del cliente, con backup diario

Qué problema resuelve y para quién

Una promoción de sorteo tradicional obliga a sincronizar las cajas con un sistema central: cada ticket emitido tiene que "avisarle" a la web que existe. Eso es frágil (si se cae la red entre la caja y el server, se pierden participaciones) y lento de montar. El Sistema de Premios elimina esa dependencia con una idea central:

> **El QR es autocontenido.** Todo lo que la web necesita del ticket —sucursal, caja, fecha de emisión, número de comprobante— viaja **firmado dentro del token**. No hay sincronización cajas ↔ web. La caja imprime, y la web valida la firma sola.

A partir de ahí, el sistema le sirve a tres actores distintos:

- El **cliente de Caracol** escanea el QR de su ticket y participa en segundos, desde el celular, sin instalar nada.
- El **operador de Caracol** administra toda la campaña desde el **gestor web**: carga premios, arma los sorteos, los ejecuta, gestiona a los ganadores y sus entregas, y consulta reportes.
- El **equipo de soporte** puede rastrear cualquier voucher, cliente o chance para responder consultas y resolver incidencias.

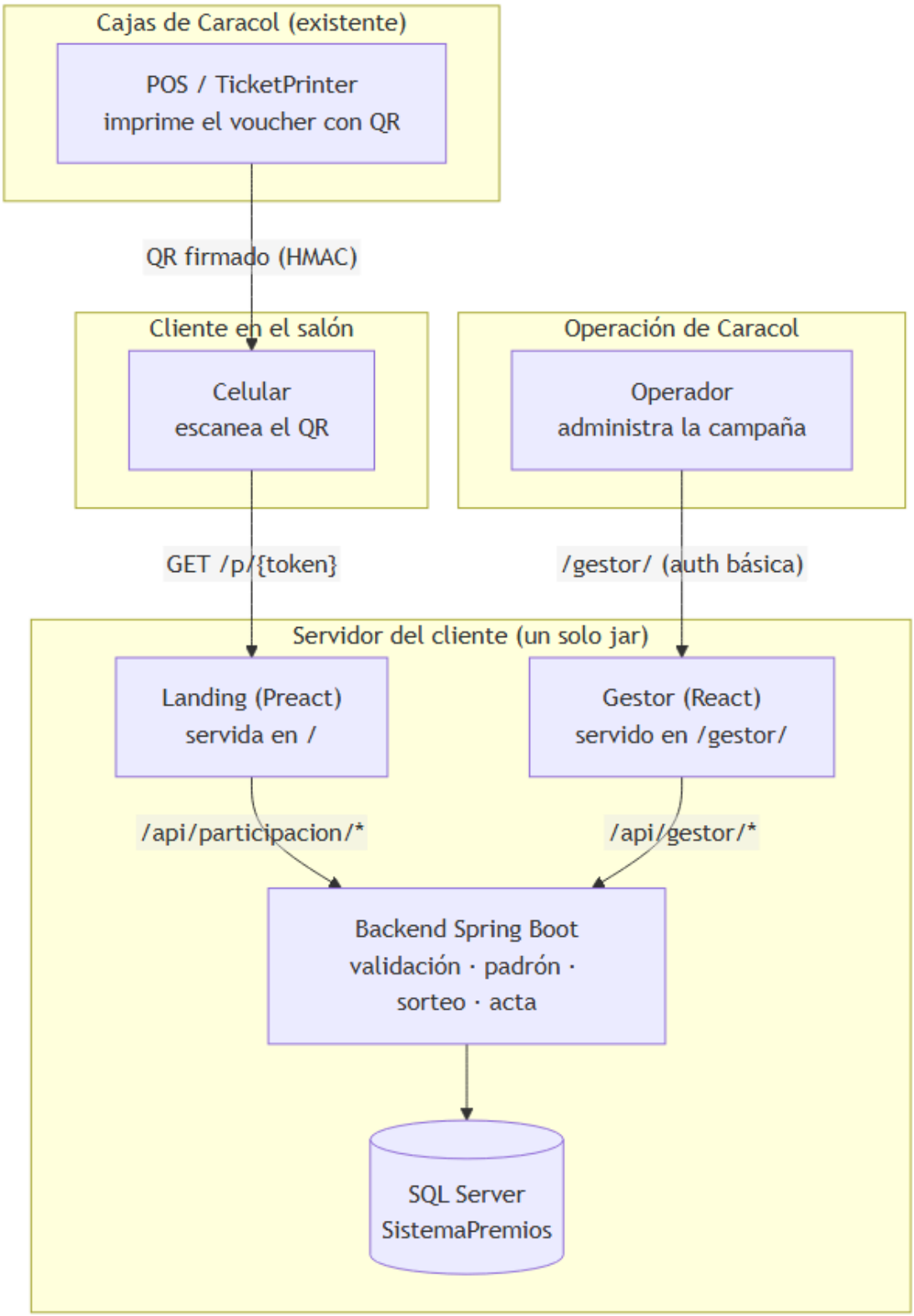
Las tres piezas del sistema

El Sistema de Premios no es una sola aplicación, sino un **backend** y **dos frontends** con propósitos —y presupuestos de peso— muy distintos. Todo vive en un mismo repositorio y se empaqueta en

un único jar que sirve las tres cosas.

Pieza	Stack	Qué hace
Backend	Java 17 · Spring Boot 4 · SQL Server · Flyway	Fuente de verdad: valida vouchers (casos A–E), administra padrón, chances, premios y su ciclo, motor de sorteo + acta reproducible, antifraude y la API del gestor.
Landing	Preact + Vite (build mínimo)	La web que abre el QR. Anónima, de un solo uso, carga instantánea en 3G de salón. Funciona en móvil y PC. Incluye escaneo del DNI por cámara.
Gestor	React 18 · Vite · Tailwind (estilo shadcn) · PWA	Panel de administración para Caracol: campañas, premios, sorteos, ganadores, clientes, reportes y modo prueba.

Diagrama de alto nivel


Cajas de Caracol (existente)
POS / TicketPrinter
imprime el voucher con QR
QR firmado (HMAC)
Cliente en el salón
Celular
escanea el QR
Operación de Caracol
Operador
administra la campaña
GET /p/{token}
/gestor/ (auth básica)
Servidor del cliente (un solo jar)
Landing (Preact)
servida en /
Gestor (React)
servido en /gestor/
/api/participacion/*
/api/gestor/*
Backend Spring Boot
validación · padrón ·
sorteo · acta
SQL Server
SistemaPremios

El contrato entre las cajas y el sistema es **únicamente el token** (ver El token del QR): el sistema no conoce el POS ni el backoffice de promociones de Caracol, sólo sabe validar la firma de lo que llega en el QR.

Cómo está organizada esta documentación

Seguimos el framework Diátaxis: cuatro tipos de página según lo que el lector necesita en ese momento.

Sección	Para qué sirve	Pregunta que responde
Entender el sistema	Comprender el porqué y el cómo	"¿Cómo funciona? ¿Por qué está diseñado así?"
Referencia técnica	Consultar datos exactos	"¿Qué endpoints, modelos y configuración hay?"
Instalación y puesta en marcha	Resolver una tarea concreta	"¿Cómo lo instalo / opero paso a paso?"
Manual de usuario	Operar y dar soporte	"Soy de Caracol, ¿cómo uso el sistema día a día?"
Propuesta comercial	Presentar el producto	"¿Qué es y qué valor tiene, en dos páginas?"

Por dónde empezar

- **Si sos técnico y vas a instalarlo:** Arquitectura → Instalar en el servidor.
- **Si sos de Caracol y vas a operarlo:** Manual de usuario.
- **Si querés la visión de producto:** Propuesta comercial.

Arquitectura del sistema

El Sistema de Premios no es una aplicación monolítica ni un enjambre de servicios: es **un backend que oficia de fuente de verdad y dos frontends que lo consumen**, todo empaquetado en un único artefacto que corre en un servidor del cliente. Esta página explica el modelo mental —dónde vive la verdad, quién le habla a quién y por qué está partido así—, no los endpoints (eso vive en la Referencia de API). Si te llevás una sola idea de acá, que sea esta: **el QR del ticket es autocontenido**, y de esa decisión cuelga casi toda la arquitectura.

La idea central: el QR autocontenido

Una promoción de sorteo tradicional obliga a **sincronizar las cajas con un sistema central**: cada ticket que se emite tiene que "avisarle" a la web que existe, para que después la web pueda validarlo. Ese acople es frágil y caro. Si se cae la red entre la caja y el servidor en el momento de la compra, la participación se pierde o el voucher nace inválido. Y montar esa sincronización con ~250 cajas de una cadena de supermercados es un proyecto en sí mismo.

El Sistema de Premios elimina esa dependencia de raíz. Todo lo que la web necesita saber del ticket —sucursal, caja, **fecha de emisión**, número de comprobante— viaja **firmado dentro del token** del QR. La caja imprime; la web valida la firma sola, sin preguntarle a nadie.

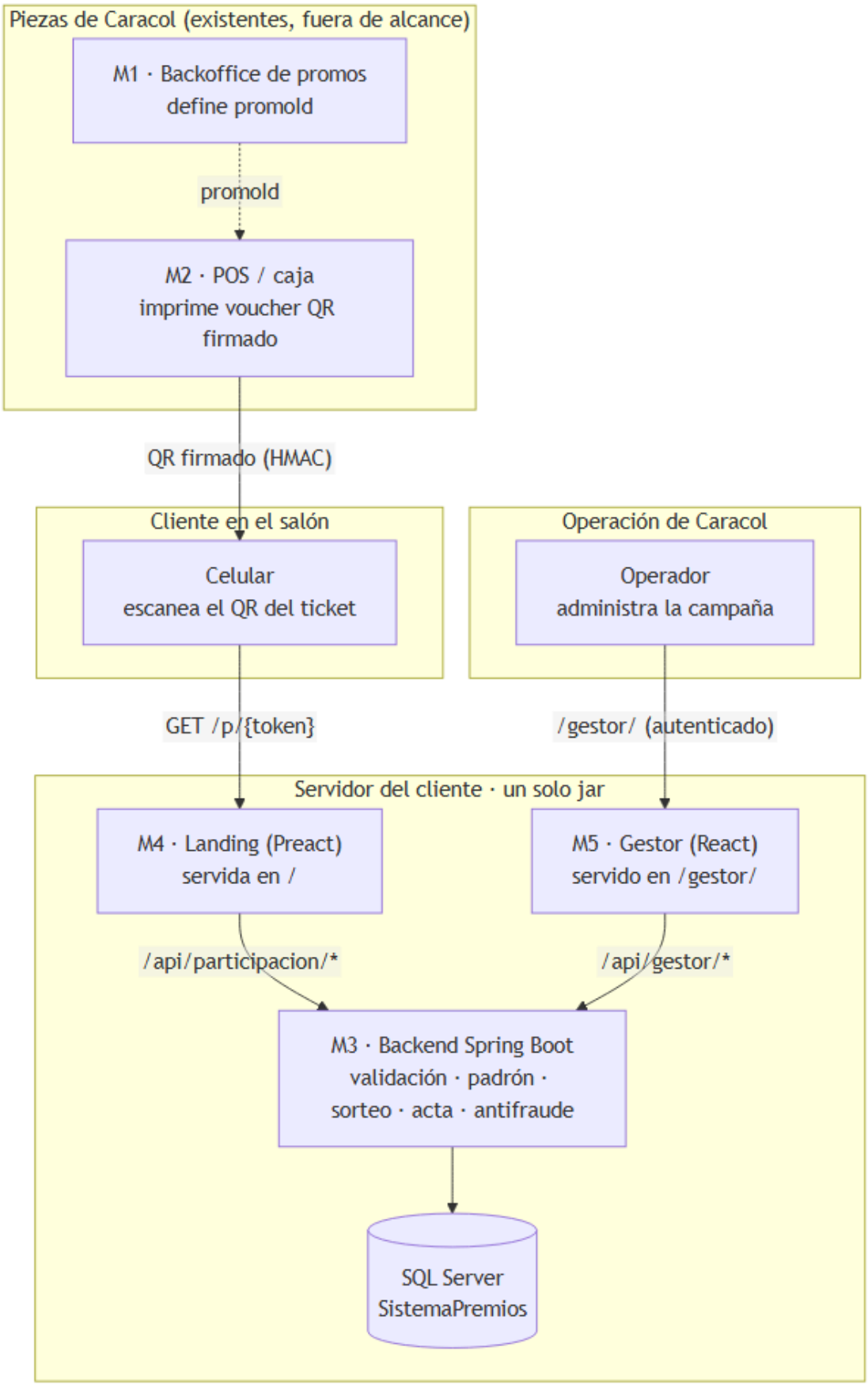
> **El contrato con las cajas es el token, y nada más**
>
> El sistema **no conoce** el POS de Caracol ni su backoffice de promociones. No hay una API entre las cajas y este sistema, no hay una tabla compartida, no hay un job de sincronización. La única superficie de contacto es el **layout del token** (Anexo A): 26 bytes que la caja firma con HMAC-SHA256 y el backend verifica. Ver El token del QR.

Esto tiene una consecuencia que conviene tener presente en toda la documentación: como el ticket puede guardarse y escanearse días después, **toda decisión temporal se resuelve por la fecha de emisión firmada en el token**, nunca por la fecha en que el cliente lo registró. Vigencia, día del tope diario y a qué sorteo pertenece una chance: todo sale del offset 8 del token. El porqué está en ADR-002, y es un invariante de la CONSTITUTION del proyecto.

Las tres piezas y cómo encajan

El sistema es un **backend** (la fuente de verdad) y **dos frontends** con propósitos —y presupuestos de peso— deliberadamente distintos. En el mapa de módulos del proyecto, estas tres piezas son **M3, M4 y M5**; conviven con piezas de Caracol que quedan **fuera de alcance** pero se integran.

Módulo	Pieza	Stack	Rol
M3	**Backend**	Java 17 · Spring Boot 4 · SQL Server · Flyway	Fuente de verdad: valida vouchers (casos A–E), administra padrón, chances, premios y su ciclo, motor de sorteo + acta, antifraude y la API del gestor.
M4	**Landing**	Preact + Vite (build mínimo)	La web que abre el QR. Anónima, de un solo uso, carga instantánea en 3G de salón. Incluye el escaneo del DNI por cámara.
M5	**Gestor**	React 18 · Vite · Tailwind (estilo shadcn) · PWA	Panel de administración para Caracol: campañas, premios, sorteos, ganadores, clientes, reportes y modo prueba.
M1	Backoffice de promos	*(existente, LaEmpresa)*	Configura las promociones; emite el `promoId` que viaja en el token. **Fuera de alcance.**
M2	POS / emisión	*(existente, LaEmpresa)*	Imprime el voucher con QR firmado en la caja. **Fuera de alcance.**
M6	Puesta en marcha	*(operación)*	Instalación, HTTPS, ambiente de prueba, salida en vivo. Ver Instalar en el servidor.


Piezas de Caracol (existentes, fuera de alcance)
M1 · Backoffice de promos
define promold
promold
M2 · POS / caja
imprime voucher QR
firmado
QR firmado (HMAC)
Cliente en el salón
Celular
escanea el QR del ticket
Operación de Caracol
Operador
administra la campaña
GET /p/{token}
/gestor/ (autenticado)
Servidor del cliente · un solo jar
M4 · Landing (Preact)
servida en /
M5 · Gestor (React)
servido en /gestor/
/api/participacion/*
/api/gestor/*
M3 · Backend Spring Boot
validación · padrón ·
sorteo · acta · antifraude
SQL Server
SistemaPremios

Fijate en la línea punteada entre M1 y M2: es la única relación con el mundo de Caracol, y ocurre **antes** de que el sistema entre en juego. Desde el QR impreso para adelante, todo es de este sistema.

Por qué la landing y el gestor son piezas separadas

No es cosmético. Los dos frontends resuelven problemas opuestos:

- La **landing** la abre el cliente en el salón, desde su celular, con la señal 3G que haya. El 99 % de la audiencia entra desde el teléfono, y —en palabras del operador— *"de eso depende que lo usen"*. Su presupuesto es el peso: tiene que pintar la primera pantalla en menos de 1,5 segundos en 3G lento. Por eso corre sobre **Preact** en lugar de React: mismo código escrito contra la API de React, pero un runtime de ~4 KB en vez de ~45 KB. El bundle inicial bajó de ~51 KB a ~14 KB gzip. El porqué completo está en ADR-019.
- El **gestor** lo usa un operador de Caracol desde una oficina, sin presión de red ni de audiencia masiva. Ahí sí conviene el stack completo (React + Tailwind + shadcn parcial + TanStack Query), porque el valor está en la riqueza de la interfaz, no en el peso. Se queda en React 18 sin cambios.

La topología de repos y frontends está decidida en ADR-015: un **monorepo** con tres módulos, un solo pipeline, versionado atómico de los contratos API ↔ frontends.

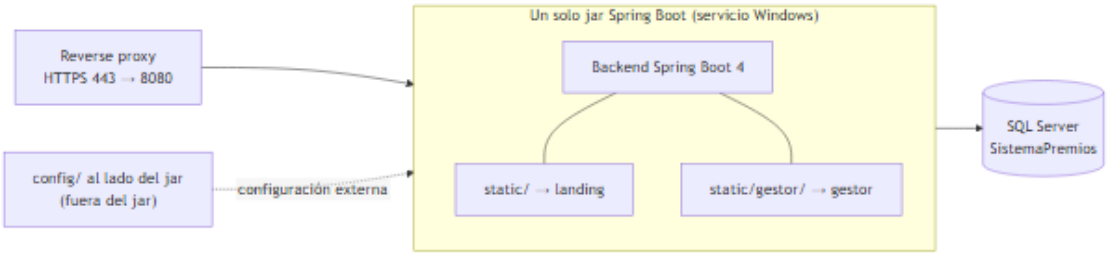
> **El backend manda sobre los contratos**
>
> Los frontends **no inventan endpoints ni formatos**: los consumen. El backend define el contrato JSON (`snake_case`) y los tipos de la landing y el gestor lo mapean. Si documento y realidad no coinciden, gana el backend. Este criterio es tan fuerte que el arranque del backend **valida que la property `snake_case` esté presente**: si falta, toda la API cambiaría de formato y la landing moriría, así que el sistema prefiere no arrancar antes que arrancar roto (ver ADR-029).

El jar único: tres piezas, un artefacto

Acá hay una decisión que sorprende viniendo de la separación anterior: aunque backend, landing y gestor son módulos distintos con builds independientes, **se despliegan como un solo jar de Spring Boot**.

En la puesta en marcha (M6), el build empaqueta `landing/dist` bajo `static/` y `gestor/dist` bajo `static/gestor/` dentro del mismo jar (perfil Maven `full`). El backend sirve los tres:

- `/` → la landing (lo que abre el QR).
- `/gestor/` → el gestor (autenticado).
- `/api/participacion/*` y `/api/gestor/*` → la API.

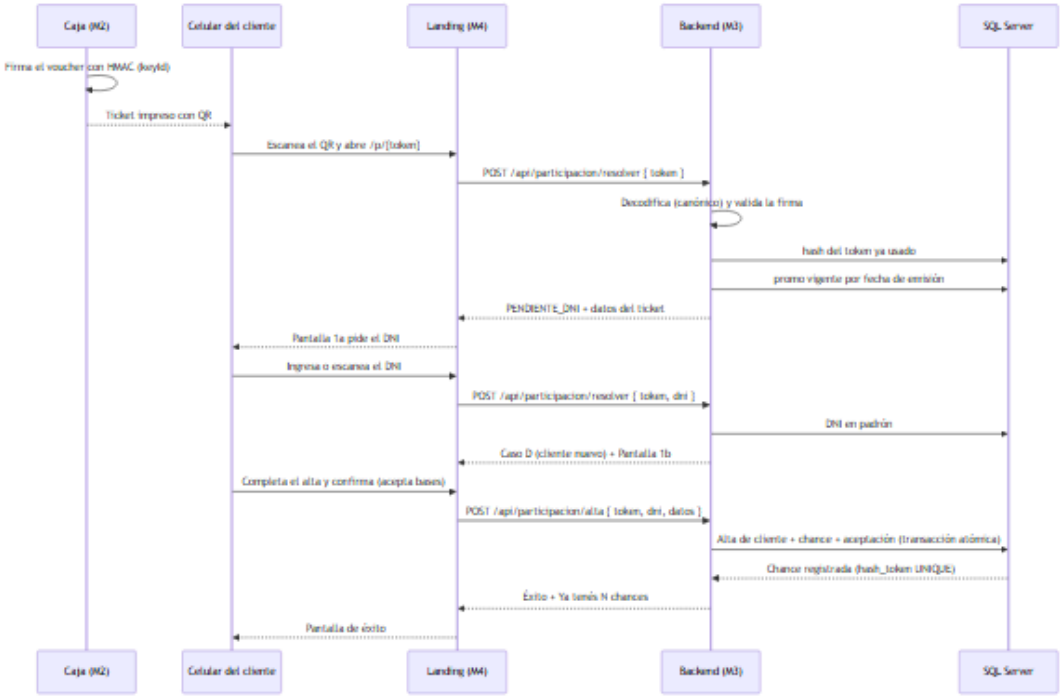


La razón es el contexto de despliegue: **un solo servidor del cliente, sin nube**, con backup diario. Un artefacto único es lo más simple de instalar, versionar y respaldar. No hay contenedores que orquestar ni servicios que coordinar; hay un jar corriendo como servicio de Windows detrás de un reverse proxy que termina HTTPS. La topología es deliberadamente chata porque las cargas son chicas y no justifican nada más.

> ⚠ **La configuración vive FUERA del jar**
>
> Un matiz importante del jar único: el artefacto **no embebe ningún archivo de configuración**. Puertos, TLS, rutas de bases y branding, flags por cliente, URL de la base: todo vive en una carpeta `config/` **al lado** del jar. Si esa carpeta falta, el sistema **no arranca en silencio con defaults invisibles**: falla rápido con un mensaje que dice qué archivo falta y dónde ponerlo. Los secretos (clave HMAC, password de la DB) van solo en variables de entorno, nunca en un archivo versionado. El porqué está en ADR-029.

Una participación de punta a punta

Veamos el camino feliz completo: un cliente que compra, escanea y participa por primera vez (caso D, cliente nuevo). El detalle de cada bifurcación —firma inválida, ya usado, fuera de vigencia, tope diario— vive en Los casos de participación; acá seguimos el hilo de una participación exitosa para ver cómo colaboran las piezas.



Los momentos que hacen que esto sea robusto:

- `resolver` **no consume nada.** Es una consulta: dice en qué caso cae el voucher sin registrar la participación. Recién `confirmar` (caso E) o `alta` (caso D) consumen la chance. Esto permite que la landing pida el DNI en el momento justo sin quemar el voucher si el cliente abandona.
- **El consumo es atómico a nivel base de datos.** La chance se inserta con el **hash del token** como constraint `UNIQUE`. Dos envíos simultáneos del mismo voucher —cosa común con 3G inestable y clientes que tocan dos veces— resuelven a **exactamente una** chance; el segundo choca contra el índice único y se traduce al caso B ("ya participó"), nunca a doble chance. Ver ADR-011 y ADR-024.
- **En la base nunca vive el token en claro, solo su hash.** Una filtración de la DB no permite reconstruir direcciones de participación válidas. Es un invariante de seguridad (ver Antifraude y seguridad).
- **El alta es una sola transacción.** Cliente + chance + aceptación de bases se insertan juntos o no se inserta nada. Un retry de un formulario cortado resuelve a caso B, no a un cliente a medio crear.

Dónde sigue

- Para el vocabulario exacto —qué es una Chance, un Acta, un PremioTipo— pasá por el Glosario del dominio.
- Para las bifurcaciones del flujo, Los casos de participación (A–E + tope).
- Para cómo se sortea de forma auditable, Motor de sorteo y acta.
- Para el detalle de entidades y columnas, el Modelo de datos.

Glosario del dominio

El Sistema de Premios tiene un vocabulario propio, en español, que aparece por todos lados: en el modelo de datos, en los endpoints, en las bases del sorteo y en las conversaciones con Caracol. Esta página define los términos reales del sistema, agrupados por categoría, para que cuando los leas en el código, en un acta o en una reunión sepas exactamente de qué se habla. Muchos términos se apoyan en otros: si uno te lleva a otro, seguilo, porque el dominio es una red, no una lista suelta.

> **Fuente de autoridad**
>
> El vocabulario canónico vive en el `CONTEXT.md` del proyecto; el porqué de cada forma vive en las Decisiones de arquitectura (ADRs), referenciadas como (Dn/ADR-nnn). Cuando necesites el detalle de columnas, índices y estados, la referencia es el Modelo de datos.

Campaña y sorteos

La estructura sobre la que se monta todo: una promoción que contiene varios eventos de sorteo.

Término	Definición
Campaña	La promoción completa (la "Sorteo 50 años"). Contiene N Sorteos y define los textos de la landing, la vigencia, la regla de chances, el tope diario, las bases (`version_bases`), la clave de firma vigente y el lote de premios. Solo puede existir **una** Campaña en estado vigente, garantizado a nivel base de datos (ADR-013).
Sorteo	Cada evento dentro de la Campaña: los semanales (`Sorteo Semana 1..N`) más el **Sorteo de cierre**. Tiene fecha y hora publicada —referencia en las bases, **no** un disparo automático—, premios asignados con prelación y, una vez ejecutado, su Acta.

promoId	Identificador de la promoción dentro del token (offset 1, 4 bytes), nativo del backoffice de Caracol (M1). Su política de asignación y reuso está **DIFERIDA** (ADR-014): fuera del alcance de esta entrega.

El voucher y su token

El cupón físico y los datos firmados que viajan dentro del QR. Acá está el corazón de la idea "autocontenida".

Término	Definición
Voucher	El cupón impreso en el ticket de compra con su QR. Es **autocontenido**: todo lo que la web necesita para validarlo viaja firmado dentro del token, sin sincronización cajas ↔ web.
Token	Los 35 caracteres de la URL del QR: `base64url(datos[16 bytes] + HMAC-SHA256 truncada a 10 bytes)`. El layout normativo está en el Anexo A. En la base se persiste su **hash**, nunca el token en claro.
Codificación canónica	La forma única a la que se normaliza el token antes de validar o registrar. Garantiza que dos codificaciones del mismo voucher resuelvan al mismo registro, para que nadie saque doble chance por una variante de encoding.
Fecha de emisión	La fecha y hora en que la caja imprimió el voucher (offsets 8/10 del token). Es **la fecha que manda**: toda decisión temporal —vigencia, día del tope, a qué sorteo pertenece la chance— se resuelve por emisión, nunca por la fecha de registración (ADR-002).

Participación: chances y clientes

Lo que ocurre cuando un cliente escanea un QR y participa.

Término	Definición
Chance	Una participación registrada: voucher consumido + DNI. La regla de esta campaña es **1 voucher = 1 chance**. Su fecha relevante es siempre la de

	emisión del ticket.
Cliente / Padrón	Una persona registrada por DNI (único). Hay **un solo** tipo `Cliente`, tanto para quien participa como para altas post-campaña (ADR-005); `fecha_nacimiento`, `sexo` y `email` son opcionales. El conjunto de clientes es el **padrón**, permanente y propio de Caracol.
Casos A–E	Los cinco resultados posibles de escanear un QR. Ver Los casos de participación para el flujo completo. En síntesis: **A** firma inválida (error genérico) · **B** token ya usado · **C** promoción no vigente por fecha de emisión (alta al padrón sin chance) · **D** vigente + DNI nuevo (alta + chance) · **E** vigente + DNI registrado (confirmación de un toque).
Tope diario	El máximo de chances por DNI por **día de emisión** (valor de esta campaña: 5). El voucher que lo supera se rechaza **sin consumirse** y se informa al cliente (ADR-006).
Consentimiento de privacidad	Permiso permanente, por Cliente: acepta la política de privacidad y las comunicaciones comerciales (ADR-007).
Aceptación de bases	Aceptación por Campaña: `version_bases` + fecha y hora. En las pantallas de participar, el botón de confirmar **implica** la aceptación (UX de un toque); el link "Bases y condiciones" está en el footer de todas las pantallas.

El sorteo y el acta

Los términos del motor que decide ganadores de forma auditable. Profundizados en Motor de sorteo y acta.

Término	Definición
Elegibles	Las chances que participan de un sorteo: las de la ventana `[fecha_sorteo_anterior, fecha_sorteo_actual)` por fecha de emisión (ADR-001), excluyendo los DNIs ya adjudicados en la Campaña (ADR-003).
Adjudicado	Un DNI que salió ganador en un acta, **retire o no** el premio. Queda excluido de los sorteos siguientes de la Campaña. Es distinto de "Otorgado" (que es el premio ya

	entregado): la exclusión es por haber sido sorteado, no por haber retirado.
Acta	El registro **inmutable y reproducible** de una ejecución de sorteo: lista ordenada de `chance_id` elegibles + hash + seed + `algo@version` + filtros + prelación + ganadores y suplentes. Cumple que `lista + seed + algo@version → mismo resultado` (ADR-004).
Prelación	El orden 1.º, 2.º, … de los premios asignados a un sorteo. Se corresponde 1 a 1 con el orden del acta: el primer sorteado se lleva el premio de prelación 1, y así.
Suplente	Una posición extra sorteada en la misma acta (10 por acta) que recibe el premio ante la incontactabilidad o el rechazo del titular, en orden. Un suplente promovido también pasa a ser adjudicado.

Premios y su ciclo

Cómo se modela lo que se reparte. Profundizado en Premios y su ciclo de vida.

Término	Definición
PremioTipo	La definición de un premio por tipo y cantidad ("Bicicleta playera × 10"). No es una unidad física: es la plantilla de la que el sistema genera las unidades.
PremioUnidad	La unidad física individual generada de un tipo. Recorre el ciclo `Disponible → Asignado → Sorteado → Otorgado`, más el terminal `NO_OTORGADO`. El `stock_disponible` es siempre el COUNT exacto de unidades en estado `Disponible`, nunca un contador paralelo que se pueda descuadrar.
NO_OTORGADO	El estado **terminal** de una unidad tras el cierre de la campaña: suplentes agotados o plazo de retiro vencido sin entrega, siempre con motivo y auditoría (ADR-008). Distinto de `Disponible`: cierra la unidad sin dejar stock fantasma.
Plazo de retiro	Los días que tiene el ganador para retirar el premio desde el primer contacto registrado (parámetro de esta campaña: 10 días corridos). Vencido, habilita la devolución al stock o el pase a `NO_OTORGADO`.

Claves y seguridad

Los términos de la firma del token y su rotación.

Término	Definición
ClaveFirma / keyId	El registro `clave_firma(key_id, secreto_cifrado, estado, vigencia_*)`. El servidor valida contra la clave que indica el `keyId` (offset 0) del token, lo que permite **rotar la clave sin corte de servicio** (ADR-010). Es HMAC simétrico: el "keystore" es una tabla cifrada, no un JKS de PKI. El secreto se guarda cifrado at-rest, nunca en claro en repo ni logs.
Reconocimiento del dispositivo	Un mecanismo opcional para acelerar la segunda participación desde el mismo teléfono sin re-pedir el DNI. El backend emite un **secreto opaco** que el navegador guarda; el DNI **nunca** toca el navegador (ADR-027). Se llama "reconocimiento", nunca "token", para no colisionar con el voucher.

Las piezas y los módulos

Quién es quién en la arquitectura. Ampliado en Arquitectura del sistema.

Término	Definición
Gestor	La web de administración para Caracol (M5): campañas, premios, sorteos, ganadores, clientes y reportes. Autenticada, sin presupuesto de peso.
Landing	La web de participación (M4): la página que abre el QR. Anónima, de un solo uso, carga instantánea en 3G. Corre sobre Preact para pesar poco.
M1–M6	Los módulos del proyecto: **M1** backoffice de promos (existente de Caracol), **M2** POS de emisión (existente), **M3** backend de participación, **M4** landing, **M5** gestor, **M6** puesta en marcha. Lo construido en este proyecto (*greenfield*) es M3, M4 y M5.

WorkDone	El proyecto de referencia visual y de stack para el gestor (React + Vite + shadcn parcial). La landing **no** hereda su peso.

Para ver cómo estos términos se conectan en flujos concretos, seguí por Los casos de participación o el Motor de sorteo y acta.

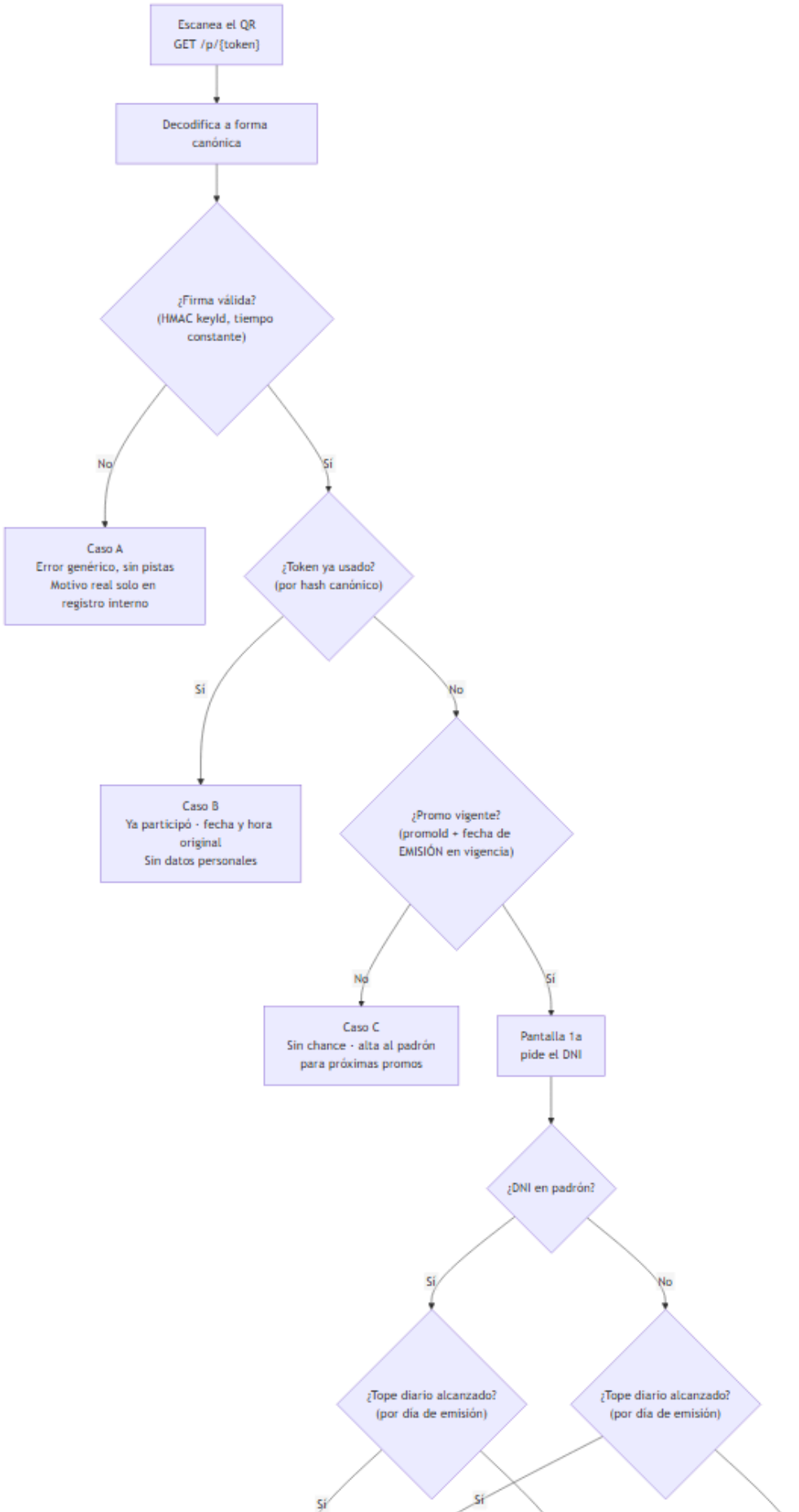
Los casos de participación (A–E + tope)

Cuando un cliente escanea el QR de su ticket, el sistema tiene que decidir **qué mostrarle** y **qué registrar**. Esa decisión no es un `if` suelto: es un árbol de preguntas encadenadas, en un orden que importa. Cada rama del árbol es uno de los **cinco casos A–E**, más una rama especial de **tope diario**. Esta página explica el árbol completo —por qué las preguntas van en ese orden y qué se consume en cada hoja— para que puedas leer una participación cualquiera y saber exactamente dónde cayó y por qué.

El detalle de las pantallas que ve el cliente vive en La experiencia del cliente; acá nos quedamos en la lógica de decisión.

El árbol de decisión

El backend resuelve la participación con una secuencia de preguntas. El orden está pensado para **no dar pistas a un atacante** y para **no quemar el voucher de un cliente legítimo** antes de tiempo.


Escanea el QR
GET /p/{token}
Decodifica a forma canónica
¿Firma válida?
(HMAC keyId, tiempo constante)
No
Sí
Caso A
Error genérico, sin pistas
Motivo real solo en registro interno
¿Token ya usado?
(por hash canónico)
Sí
No
Caso B
Ya participó · fecha y hora original
Sin datos personales
¿Promo vigente?
(promoId + fecha de EMISIÓN en vigencia)
No
Sí
Caso C
Sin chance · alta al padrón para próximas promos
Pantalla 1a pide el DNI
¿DNI en padrón?
Sí
No
¿Tope diario alcanzado?
(por día de emisión)
¿Tope diario alcanzado?
(por día de emisión)
Sí
Sí

Las preguntas, en orden y con su porqué

1. ¿La firma es válida? → Caso A

Lo primero es verificar el HMAC del token contra la clave que indica su `keyId`. Si no valida —firma adulterada, `keyId` inexistente o revocado—, el cliente recibe un **error genérico, sin ninguna pista** de qué falló. El motivo real ("firma inválida") queda **solo en el registro interno** (`VoucherRechazado`), que alimenta las señales antifraude.

> **Nunca darle un oráculo a quien intenta forzar**
>
> El error genérico no es pereza de UX: es seguridad. Si el sistema dijera "firma inválida" vs. "token ya usado" vs. "promo vencida", le estaría enseñando a un atacante a distinguir tokens bien formados de basura. Hacia afuera, todos los rechazos se parecen; la verdad vive puertas adentro. Es un invariante de la CONSTITUTION. La comparación de la firma, además, se hace en **tiempo constante** para no filtrar información por *timing* (ver Antifraude y seguridad).

2. ¿El token ya se usó? → Caso B

Si la firma es válida, se pregunta por el **hash canónico** del token contra las chances ya registradas. Si aparece, el voucher ya participó: se muestra la **fecha y hora de la participación original**, sin datos personales. Un cliente que reescanea su propio ticket ve "ya participaste el día tal"; un retry de un formulario que se cortó a mitad de camino resuelve también acá, y por eso **nunca produce doble chance** (ADR-011).

3. ¿La promo está vigente? → Caso C

Se valida el `promoId` y, sobre todo, que la **fecha de emisión** del ticket caiga dentro de la vigencia de la campaña. Si no —un ticket emitido antes del inicio o después del cierre—, no hay chance, pero **sí hay una oportunidad comercial**: se ofrece el alta al padrón para próximas promociones (pantalla 4a para un cliente nuevo, 4b para uno ya registrado). Caracol se lleva un cliente en el padrón aunque ese voucher no participe.

> **Por qué la fecha de emisión y no la de registración**
>
> Si la vigencia se midiera por cuándo el cliente registra el voucher, alguien podría **guardar tickets** y registrarlos en la semana "conveniente". Midiendo por la fecha de emisión firmada en el token, ese hueco se cierra: la fecha no es manipulable porque viaja firmada. Vale para la vigencia, para el día del tope y para el bucketing a sorteo. El porqué está en ADR-002, y es un invariante.

4. ¿El DNI está en el padrón? → Caso E o Caso D

Recién cuando el voucher es válido y vigente, la landing pide el DNI (tipeado con reingreso de confirmación, o escaneado del código de barras del DNI físico con la cámara). Con el DNI en mano:

- **Está en el padrón → Caso E.** El cliente ya existe. Se lo saluda ("Hola {nombre}"), confirma en un toque, se registra +1 chance y se muestra el total acumulado. Es el camino más rápido.
- **No está → Caso D.** Cliente nuevo. Se piden los datos de alta (nombre, apellido, celular obligatorio de 10 dígitos normalizado, fecha de nacimiento, sexo, email opcional) y, al confirmar, se hace el alta + la chance. El botón de confirmar implica la aceptación de bases.

La rama transversal: tope diario

Antes de consumir el voucher en los casos D y E, hay un último control: **¿este DNI ya tiene 5 chances con fecha de emisión de ese día?** Si es así, el voucher se **rechaza sin consumirse** y se informa al cliente. El tope es un valor de campaña (5 en esta), pensado como señal antifraude.

> **El rechazo por tope es definitivo para ese día, y no consume el voucher**
>
> Como el día se resuelve por **emisión**, no hay "reintentá mañana": mañana ese mismo voucher sigue perteneciendo al mismo día de emisión, que ya está colmado. Pero —y esto es lo justo— el voucher **no se consume**: no se le quema una chance al cliente por un límite temporal. El intento queda en el registro interno con motivo `tope_diario` para las señales antifraude. Ver ADR-006.

Dos reglas transversales que sostienen todo

Por debajo de los casos, hay dos invariantes que hacen que el flujo funcione con 3G malo y clientes que tocan dos veces:

- **El consumo es atómico a nivel base de datos.** La chance se inserta con el **hash del token** como constraint `UNIQUE`. Aunque lleguen envíos simultáneos del mismo voucher, se registra **exactamente una** chance; el segundo choca contra el índice y resuelve a caso B. En los casos D, el alta de cliente + la chance + la aceptación de bases son **una sola transacción**: se insertan juntas o no se inserta nada. Un formulario cortado no deja un cliente a medias (ADR-011, ADR-024).
- **Todo lo temporal usa la fecha de emisión.** No es solo la vigencia (caso C): el día del tope diario y —más adelante— a qué sorteo pertenece la chance también se resuelven por la fecha embebida y firmada en el token. Esta es la pieza que cierra el hueco de guardar vouchers.

Un detalle de API que vale conocer

La landing resuelve el caso en **dos pasos**, y por una buena razón. El endpoint `POST /api/participacion/resolver`:

- **Sin DNI**, con un token vigente, no puede saber todavía si el cliente es nuevo o conocido, así que devuelve un estado intermedio `PENDIENTE_DNI`: es lo que dispara la pantalla 1a que pide el DNI.
- **Con DNI**, ya resuelve a D, E o tope.

Ninguna llamada a `resolver` consume el voucher: es una consulta. El consumo ocurre recién en `confirmar` (caso E) o `alta` (caso D), ambos idempotentes. Este desdoblamiento fue una enmienda de diseño (SCR-001): la firma original solo con `{token}` no alcanzaba para producir D/E/tope, que dependen del DNI. El detalle de los contratos está en la Referencia de API.

Dónde sigue

- Para ver qué pasa con las chances una vez registradas, Motor de sorteo y acta.
- Para las señales antifraude que alimentan estos rechazos, Antifraude y seguridad.
- Para las pantallas concretas que ve el cliente, La experiencia del cliente.

Motor de sorteo y acta

El sorteo es el momento más delicado del sistema: reparte premios reales, ante clientes reales, con bases legales de por medio. Un reclamo —"¿por qué ganó ese y no yo?"— tiene que poder responderse con evidencia, no con "confiá en el sistema". Por eso el motor de sorteo está diseñado alrededor de una sola obsesión: **que el resultado sea reproducible y auditable**. Esta página explica cómo se eligen los elegibles, cómo se sortea, y por qué el acta es la pieza legal que protege a Caracol y al cliente por igual.

Qué chances entran: los elegibles

No todas las chances participan de todos los sorteos. Cada sorteo toma **su ventana** de chances, definida por la **fecha de emisión** del ticket.

- La ventana de un sorteo es `[fecha_sorteo_anterior, fecha_sorteo_actual)` : las chances emitidas desde el sorteo previo hasta este. El primer semanal toma desde el inicio de vigencia; lo emitido después del último semanal cae en el **sorteo de cierre** (ADR-001).
- De esa ventana se **excluyen los DNIs ya adjudicados** en actas previas de la campaña: quien ya ganó no vuelve a entrar (ADR-003).

> **Bucketing por emisión: por qué un voucher tardío participa de su semana**
>
> Como la ventana se mide por emisión, un voucher emitido el lunes 08/09 y registrado recién el 16/09 participa del sorteo cuya ventana contiene el 08/09, no del que estaba corriendo cuando se registró. La fecha de registración no cambia a qué sorteo pertenece la chance: manda la emisión firmada en el token. Esto hace que el query de elegibles sea **determinista y auditable**, y cierra el hueco de guardar vouchers para la semana conveniente.

La exclusión de adjudicados y el dedup por DNI

Hay dos reglas contra "ganar dos veces", y operan en momentos distintos:

- **Exclusión entre actas (por adjudicación).** Un DNI que salió ganador en un acta queda **adjudicado** —retire o no el premio— y se excluye de los elegibles de todos los sorteos

siguientes de la campaña. La exclusión es por haber sido sorteado, no por haber retirado: no depende del estado de entrega. Los suplentes promovidos también quedan adjudicados.

- **Dedup dentro del acta (por DNI).** Al recorrer la lista ordenada de una misma ejecución, un DNI con **varias chances** gana a lo sumo **un premio por acta**: sus chances extra se saltan. Alguien con 3 chances no ocupa 3 posiciones de ganador.

> **Ningún DNI gana dos premios, ninguna unidad va en dos actas**
>
> Estas dos reglas —dedup por DNI intra-acta y exclusión por adjudicación entre actas— son invariantes de la CONSTITUTION, no preferencias. Están respaldadas además por ADR-024: la ejecución de sorteos de una misma campaña se **serializa** con un lock de campaña, para que dos sorteos ejecutándose casi a la vez no le den el premio al mismo DNI por leer estado desactualizado. En la operación real los sorteos se ejecutan de a uno, pero la defensa está igual.

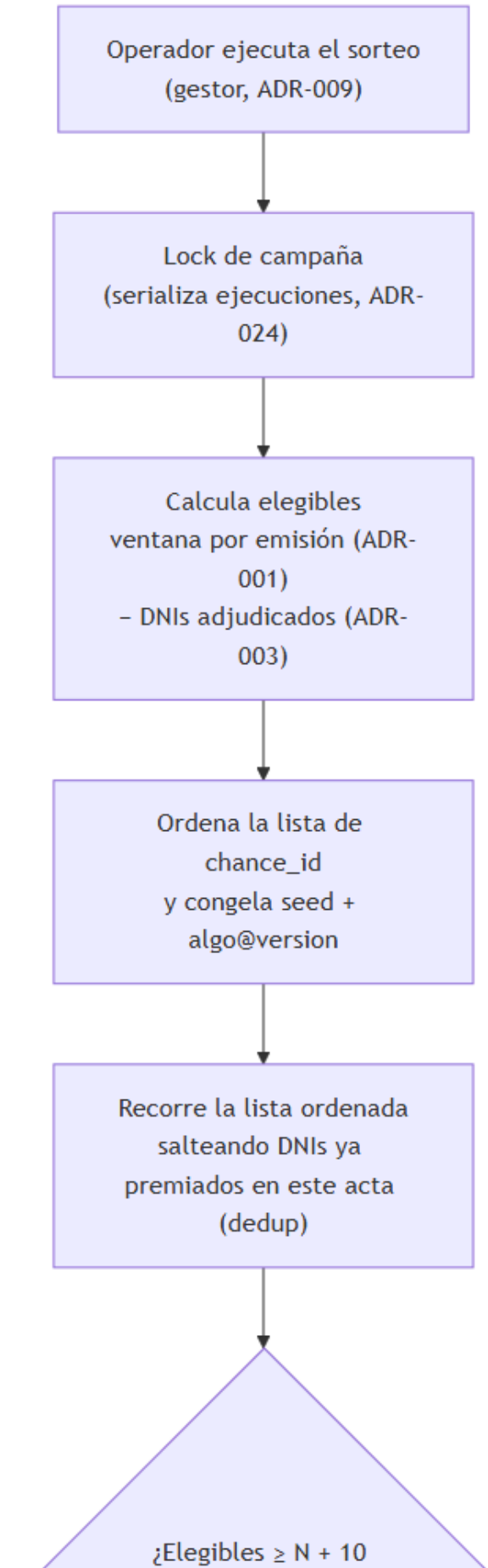
Cómo se ejecuta: siempre por decisión humana

El sorteo **nunca** se dispara solo. La hora publicada en las bases (los lunes 10:00, el cierre el sábado) es una **referencia**, no un *trigger*. La ejecución ocurre por la acción de un operador desde el gestor, con un botón "Ejecutar sorteo" que muestra un resumen de elegibles y pide confirmación.

> **Por qué no hay scheduler**
>
> Un scheduler ejecutaría el sorteo con datos que nadie revisó, y podría fallar en silencio un lunes 10:00 sin nadie delante. Con ejecución manual **siempre hay un humano responsable** frente al acta, no hay código de scheduling que auditar, y un feriado o una demora se maneja operativamente sin tocar nada. El porqué está en ADR-009, y es un invariante de la CONSTITUTION. El paso a paso operativo está en Ejecutar un sorteo.

El proceso, paso a paso

Cada ejecución sortea `N ganadores + 10 suplentes` en **una sola acta ordenada**. La `N` no es un parámetro suelto: se **deriva de las unidades de premio asignadas** a ese sorteo. La prelación es 1 a 1: el primer sorteado se lleva el premio de prelación 1, el segundo el de prelación 2, y así.


Operador ejecuta el sorteo (gestor, ADR-009)
Lock de campaña (serializa ejecuciones, ADR-024)
Calcula elegibles ventana por emisión (ADR-001) – DNIs adjudicados (ADR-003)
Ordena la lista de chance_id y congela seed + algo@version
Recorre la lista ordenada salteando DNIs ya premiados en este acta (dedup)
¿Elegibles ≥ N + 10

Cuando faltan elegibles: se sortea lo que hay

¿Qué pasa si una semana floja deja menos chances elegibles que posiciones a cubrir? El motor **no falla y no espera**: sortea sobre los elegibles que existan. Si alcanzan para K posiciones (K < N), el acta registra K resultados y las unidades sin ganador quedan en estado `Asignado`, a la espera de una decisión del operador (devolución al stock o `NO_OTORGADO` según el momento). Con cero elegibles, el acta se registra vacía y válida, sin error.

> **El acta refleja fielmente la escasez**
>
> Fallar la ejecución bloquearía un sorteo publicado; esperar más participaciones violaría la fecha de las bases. Sortear lo disponible es la única opción que respeta lo prometido. El acta muestra exactamente lo que hubo. Ver ADR-012.

El acta: inmutable y reproducible

Acá está el corazón de la auditabilidad. Cada ejecución persiste un **snapshot completo e inmutable**:

- la **lista ordenada** de `chance_id` elegibles,
- el **hash** de esa lista (para detectar cualquier alteración),
- el **seed** (semilla del generador pseudoaleatorio),
- el `algo@version` (el algoritmo y su versión, p. ej. `fisher-yates-sha256@1`),
- los **filtros** aplicados (la ventana de emisión y las exclusiones),
- la **prelación** de premios y los **ganadores y suplentes** en orden.

La propiedad que esto garantiza es simple y poderosa: **`lista + seed + algo@version → mismo resultado`, siempre**. Cualquiera que tenga el acta puede volver a correr el motor sobre ese snapshot y obtener exactamente los mismos ganadores y suplentes. Ante un reclamo, el resultado se reproduce; no se argumenta.

El acta ejecutada no se altera jamás

No existe un endpoint ni una operación que modifique un acta ejecutada: es inmutable por diseño, y hay un **golden test obligatorio** de reproducibilidad que la CONSTITUTION exige antes de dar por converger el backend. Cambiar el algoritmo obliga a una nueva `algo@version`; las actas viejas se siguen reproduciendo con la suya. Ver ADR-004. Una ejecución errónea **no se "re-ejecuta"**: se resuelve por decisión humana documentada fuera del sistema (nueva asignación + nuevo sorteo).

Lo sorteado vs. lo entregado: dos cosas distintas

Un matiz que evita confundir el artefacto legal con la operación: el acta congela el **premio sorteado originalmente** de cada posición (`premio_unidad_id_original`), y ese dato **no cambia nunca**. El estado de entrega —a quién se le entregó, si se devolvió, si pasó a un suplente— evoluciona con la operación y se exporta **por separado**, rotulado como "estado al momento del export".

Así, dos exports del mismo acta en momentos distintos tienen una sección de **acta idéntica** (el sorteo, inmutable) y, eventualmente, una sección de **entrega distinta** (lo operativo, que se mueve). El sorteo protege su valor legal sin perder la trazabilidad de las entregas. El porqué está en ADR-025.

Dónde sigue

- Para el ciclo de vida de los premios que este motor asigna y sortea, Premios y su ciclo de vida.
- Para las entidades `Acta` , `ResultadoSorteo` y `AsignacionPremio` , el Modelo de datos.
- Para operar un sorteo real, Ejecutar un sorteo.

Premios y su ciclo de vida

Un premio en este sistema no es un dato estático: nace como una definición, se materializa en unidades físicas, se asigna a un sorteo, se sortea, y termina entregado o cerrado. Cada uno de esos pasos es una transición con reglas, y toda transición queda auditada. Esta página explica ese ciclo de vida —por qué está modelado en dos niveles, cómo se mueve una unidad de premio de punta a punta, y cómo el sistema garantiza que el inventario de la campaña **cierre exacto** al final.

Dos niveles: PremioTipo y PremioUnidad

El premio se modela en dos capas, y la distinción es la clave de todo lo demás:

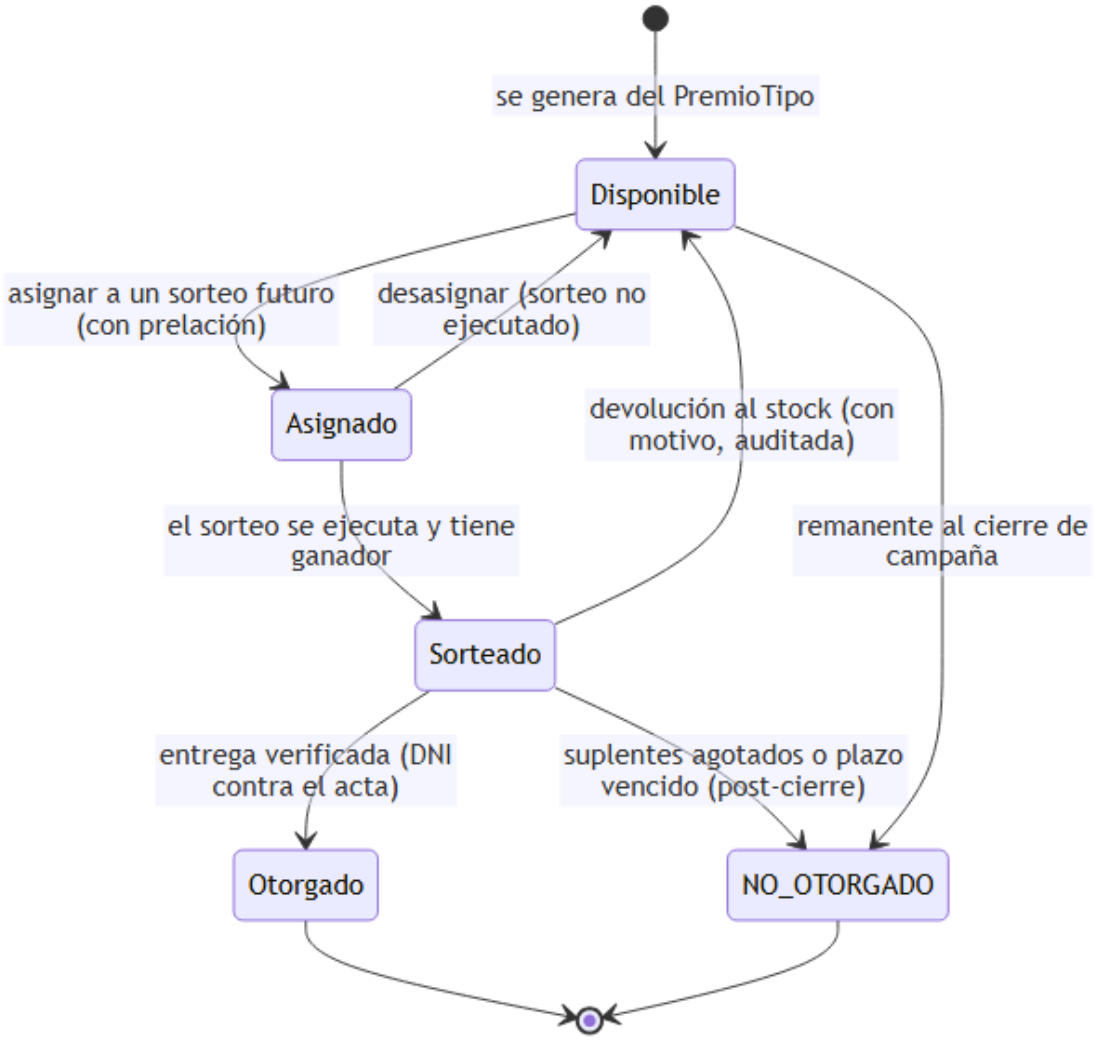
- **PremioTipo** es la **definición**: "Bicicleta playera × 10", con su valor unitario y su cantidad. Es la plantilla, no algo físico.
- **PremioUnidad** es cada **unidad individual** generada de ese tipo. Si el tipo dice "× 10", el sistema genera 10 unidades. Son las que se asignan, se sortean y se entregan.

El operador carga los premios por tipo y cantidad, y el sistema genera las unidades. Es un **lote único** para toda la campaña.

> **El stock nunca se descuadra**
>
> El `stock_disponible` **no es un contador paralelo** que alguien incrementa y decrementa: es el **COUNT exacto** de unidades en estado `Disponible`. Al ser una consulta sobre el estado real, no puede desincronizarse de la verdad. Es un invariante de la CONSTITUTION. La lista real de tipos, cantidades y valores la entrega Caracol; el modelo es genérico (`PremioTipo × cantidad → PremioUnidad`) y no bloquea el desarrollo.

El ciclo de estados de una unidad

Una `PremioUnidad` recorre estados con transiciones definidas. El camino feliz es `Disponible → Asignado → Sorteado → Otorgado`, pero hay retrocesos legítimos y un estado terminal para cerrar lo que no se entregó.



Estado	Qué significa
Disponible	La unidad está en el stock, sin asignar. Cuenta para `stock_disponible`.
Asignado	Reservada para un sorteo futuro, con su prelación. Todavía no se sorteó.
Sorteado	Tiene un ganador en un acta ejecutada. Aún no se entregó.
Otorgado	Entregada al ganador, con verificación de DNI. Estado **terminal**.

NO_OTORGADO	Cerrada sin entregar tras el cierre de campaña, con motivo. Estado **terminal**.

Asignar y "repartir automáticamente"

Antes de un sorteo, las unidades se **asignan** a él con su prelación (el orden en que se reparten). Para ahorrarle trabajo al operador, el gestor ofrece "**repartir automáticamente**": distribuye el stock en partes iguales entre las semanas como **punto de partida**. La asignación final y la prelación siguen siendo decisión del operador; el reparto automático es una sugerencia, no una imposición.

Los retrocesos: desasignar y devolver

No todo avanza en línea recta, y el modelo lo contempla:

- **Desasignar** (`Asignado → Disponible`): mientras el sorteo **no se ejecutó**, una unidad asignada puede volver al stock.
- **Devolución al stock** (`Sorteado → Disponible`): una unidad ya sorteada cuya entrega no prospera —y todavía hay sorteos futuros— vuelve al stock para reasignarse. Siempre **con motivo y auditada**.

> ⚠ **Toda reasignación o devolución exige motivo y queda auditada**
>
> Ningún retroceso es silencioso. Quién, cuándo y por qué quedan registrados en un `EventoPremio` . Es un invariante de la CONSTITUTION: el inventario de premios es dinero real y su historia tiene que ser reconstruible.

La entrega

Cuando un ganador retira su premio, la entrega registra **fecha + sucursal + responsable + verificación del DNI contra el acta**. Esa verificación de DNI es la que ata a la persona que retira con la que salió sorteada: no alcanza con presentarse, hay que ser quien el acta dice.

Una unidad pasa a `Otorgado` —estado terminal— recién con esa entrega verificada.

Cuando el titular no aparece: suplentes, plazo y cierre

El camino de entrega tiene desvíos previstos por las bases:

- **Incontactable o rechazo del titular** → el premio pasa al **siguiente suplente** del acta, en orden. Los suplentes están justamente para esto (10 por acta).
- **Suplentes agotados o plazo de retiro vencido** → según el momento:
  - Si **todavía hay sorteos futuros**, la unidad se **devuelve al stock** para reasignarse.
  - Si es **post-cierre** y ya no hay a dónde devolverla, pasa a `NO_OTORGADO`.

El **plazo de retiro** (10 días corridos desde el primer contacto registrado) **no se automatiza**: un scheduler que despoje a un ganador legítimo violaría la ejecución-por-humano y sería injusto. En cambio, se **visibiliza**: la vista de ganadores muestra si el plazo está vencido, y registrar una entrega sobre un plazo vencido exige una **confirmación explícita** del operador —advierte, no bloquea. La decisión sigue siendo humana. Ver ADR-025.

> **Por qué existe NO_OTORGADO**
>
> Tras el cierre de campaña ya no hay "próximos sorteos" a los que devolver una unidad no entregada. Sin un estado terminal, la máquina de estados quedaría abierta y el stock arrastraría unidades fantasma. `NO_OTORGADO` cierra la unidad de forma explícita, con motivo (suplentes agotados o plazo vencido) y auditoría. Así, **cada unidad termina en `Otorgado` o `NO_OTORGADO`**, y la rendición ante Caracol es un simple COUNT: nada queda en el limbo. Ver ADR-008.

La auditoría: EventoPremio

Cada cambio de estado de una unidad —asignado, desasignado, sorteado, otorgado, devuelto al stock, reasignado a suplente, no otorgado— deja un registro `EventoPremio` con **motivo, usuario y fecha/hora**. No es un log opcional: es la trazabilidad que permite reconstruir la historia completa de cualquier premio, desde que se generó del tipo hasta su estado terminal.

> **El inventario cierra exacto**
>
> La combinación de un `stock_disponible` que es COUNT real, un estado terminal explícito y un evento por cada transición da una propiedad valiosa al cierre: el inventario cuadra sin ambigüedad. Cada unidad tiene un estado final y una historia auditada de cómo llegó ahí.

Dónde sigue

- Para cómo se sortean los ganadores que después se entregan acá, Motor de sorteo y acta.

- Para las entidades `PremioTipo`, `PremioUnidad`, `EventoPremio` y `ResultadoSorteo`, el Modelo de datos.
- Para operar entregas y pases a suplente, Gestionar ganadores y entregas.

Antifraude y seguridad

Un sorteo con premios reales atrae a quien quiere ganarle al sistema, y un padrón de clientes es un dato sensible que hay que cuidar. El Sistema de Premios trata estas dos cosas —**evitar el fraude** y **proteger los datos**— como requisitos de primera clase, no como un agregado. Esta página explica las defensas: qué señales delatan un abuso, qué límites frenan a un atacante, y qué invariantes de seguridad no se rompen por ninguna conveniencia. La base normativa son los §8 y §9 del SPEC y la CONSTITUTION del proyecto, que define lo **nunca aceptable**.

Antifraude: las señales que delatan el abuso

El fraude en un sorteo por vouchers tiene patrones reconocibles. El panel del gestor expone **señales de concentración** que hacen visible lo que a simple vista se perdería:

Señal	Qué delata
Concentración por DNI	Muchos vouchers apuntando a un mismo DNI: alguien acumulando chances de forma anómala.
Concentración por conexión	Muchas participaciones desde una misma conexión: un bot o una granja operando desde un punto.
Concentración por caja	Vouchers concentrados en una caja puntual: posible emisión irregular o una caja comprometida.
Firma inválida dispersa	Muchos rechazos por firma inválida con origen disperso: alguien probando tokens fabricados a ver si alguno pega.

Estas señales se alimentan del registro interno de rechazos (`VoucherRechazado`), donde queda el **motivo real** de cada intento fallido. Ninguno de esos motivos viaja al cliente: hacia afuera, todo rechazo es genérico (ver más abajo).

Los límites que frenan el abuso en caliente

Las señales son para observar; los límites son para **frenar**. Hay tres, y operan en momentos distintos del flujo:

- **Rate limit por conexión.** El propio sistema aplica límites de frecuencia por conexión, para que un bot no pueda martillar el endpoint de participación. Se aplica sobre `/api/**`, así que cubre también los caminos de reconocimiento del dispositivo.
- **Verificación anti-robots invisible (honeypot).** El envío del formulario lleva una verificación anti-robots **invisible** para el cliente legítimo: no hay CAPTCHA que resolver, no hay fricción. Un bot que llena todos los campos cae en la trampa; una persona real ni se entera de que existe.
- **Tope diario por DNI.** El máximo de chances por DNI por **día de emisión** (5 en esta campaña). El voucher que lo supera se **rechaza sin consumirse** y se informa. Al resolverse por día de emisión, el rechazo es definitivo para ese día: no hay "reintentá mañana". Ver ADR-006 y Los casos de participación.

> **El tope no le quema la chance a nadie**
>
> El tope diario rechaza **sin consumir** el voucher: un cliente legítimo que compró mucho un día no pierde sus vouchers, simplemente no puede convertir más de 5 en chances de ese día de emisión. La defensa antifraude no castiga al cliente honesto.

Seguridad del token: la firma y su secreto

El token es la única superficie de contacto con las cajas, así que su integridad es crítica. Las defensas están pensadas para que un atacante no pueda **fabricar** vouchers ni **aprender** del sistema probándolo.

- **Comparación de la firma en tiempo constante.** El HMAC del token se compara en tiempo constante, nunca con un `==` byte a byte que termine antes al primer byte distinto. Un `==` ingenuo filtraría, por *timing*, cuánto de la firma acertó un atacante, dándole una vía para forjar una firma válida a fuerza de medir. Tiempo constante cierra ese canal (CWE-208).
- **El secreto HMAC, cifrado at-rest.** El secreto de firma vive **cifrado** (master key vía variable de entorno / Vaultwarden), nunca en claro en el repositorio ni en los logs. El "keystore" es una tabla `clave_firma` cifrada, no un archivo suelto. Ver ADR-010.
- **Rotación de clave sin corte.** Como el token lleva su `keyId`, se puede rotar la clave dando de alta una nueva y validando ambas durante la transición. Si una caja se compromete, se rota el `keyId`: es un **riesgo aceptado y documentado**, mitigado por la rotación más las señales de concentración por caja.

Errores genéricos hacia afuera: no darle un oráculo al atacante

Hacia el cliente, **todos los rechazos se parecen**: un error genérico, sin decir si fue firma inválida, token ya usado o promo vencida. El motivo real queda solo en el registro interno. Si el sistema distinguiera los errores, le estaría enseñando a un atacante a separar tokens bien formados de basura, o a mapear el estado de vouchers ajenos. Es un invariante de la CONSTITUTION.

Seguridad de los datos: qué se guarda y qué no

El principio rector es que **una filtración de la base no debe permitir reconstruir nada útil**. De ahí varias decisiones:

- **En la base vive el hash del token, nunca el token.** Quien acceda a la DB no puede reconstruir direcciones de participación válidas (`/p/{token}`) a partir de los hashes. Mismo criterio, misma protección (CWE-312).
- **El DNI nunca se persiste en el navegador.** El reconocimiento del dispositivo —el "Hola {nombre}" de la segunda participación— **no** guarda el DNI en `localStorage`, ni siquiera un hash del DNI (un DNI son ~90 millones de valores, un hash se revierte por fuerza bruta). Usa un **secreto opaco** emitido por el backend que no significa nada sin el servidor, y en la base se persiste **solo el hash** de ese secreto. Ver ADR-027.
- **El reconocimiento no filtra datos ni autoriza.** El endpoint de reconocimiento devuelve **solo** primer nombre + DNI enmascarado, nunca el DNI completo ni contacto. Y reconocer **no** habilita a editar datos: modificar un cliente exige haber presentado el DNI en la sesión. El teléfono puede ser compartido (la caja del súper, un familiar), así que reconocer nunca implica participar por otro: siempre hay confirmación explícita, y "No soy yo" borra el secreto.

Seguridad de la aplicación

Las defensas transversales que aplican a todo el backend:

- **SQL siempre parametrizado**, nunca concatenado (CWE-89): ninguna entrada de usuario se pega dentro de una query.
- **Toda entrada externa se valida antes de usarse** (CWE-20).
- **Sin secretos en logs ni en el repositorio** (CWE-532 / CWE-798): las credenciales de la DB y la master key van solo por variables de entorno.

- **La configuración operativa vive fuera del jar** y el arranque falla rápido si falta, en vez de arrancar con defaults invisibles. El archivo de configuración versionado **jamás** contiene un valor secreto, solo referencias `${SP_*}`. Ver ADR-029.

Auditoría: la evidencia que se captura, no se narra

La última capa de seguridad es la trazabilidad. El sistema no "cuenta" lo que hizo: lo **registra** para que sea verificable.

- **Actas reproducibles.** Cada sorteo deja un snapshot inmutable que reproduce el resultado exacto (`lista + seed + algo@version → mismo resultado`). Un reclamo se responde reproduciendo, no argumentando. Ver Motor de sorteo y acta y ADR-004.
- **Eventos de premio.** Cada cambio de estado de una unidad queda auditado con quién, cuándo y por qué (`EventoPremio`). Ver Premios y su ciclo de vida.
- **Cambios de datos del cliente auditados.** Toda corrección de datos de un cliente queda registrada (quién por DNI, qué campo, valor anterior → nuevo, cuándo) en el registro interno, nunca expuesta por la API pública.
- **Contactos con ganadores registrados.** Los intentos de contacto quedan asentados como prueba frente al plazo de retiro de las bases.

> **Evidencia por ejecución, nunca narrada**
>
> Es un principio de la CONSTITUTION: la evidencia se **captura por la ejecución** del sistema, no se relata. Un acta que se reproduce, un evento que registra una transición, un contacto asentado con fecha: todo es verificable de forma independiente. Esa es la diferencia entre "confiá en nosotros" y "mirá el registro".

Dónde sigue

- Para las bifurcaciones del flujo que producen estos rechazos, Los casos de participación.
- Para las claves de firma y su rotación, ADR-010 y el Token del QR.
- Para las defensas de concurrencia (lost updates, ejecución concurrente de sorteos), ADR-024.

Decisiones de arquitectura (ADRs)

Un **ADR** (Architecture Decision Record) registra una decisión de diseño importante: el contexto en el que se tomó, la opción elegida y sus consecuencias. Son la memoria del *por qué* del sistema — cuando alguien se pregunte "¿por qué está hecho así y no de esta otra forma?", la respuesta vive acá.

Estos ADRs se escribieron durante el desarrollo del Sistema de Premios y se conservan tal cual. El SPEC dice *qué* hace el sistema; estos documentos explican *por qué* tiene la forma que tiene.

Índice

ADR-011	Idempotencia por constraints de base de datos
ADR-012	Elegibles insuficientes: el motor sortea lo que hay
ADR-013	Dominio Campaña / Sorteo
ADR-014	Política de promoId: DIFERIDA
ADR-015	Topología: monorepo con dos frontends de presupuesto distinto
ADR-016	Stack backend: Java + Spring Boot + SQL Server
ADR-017	Suite E2E visual/funcional con Playwright contra el jar empaquetado
ADR-018	Upgrade a Spring Boot 4 (framework 7, security 7, hibernate 7, jackson 3) + consola boot-ui (solo dev)
ADR-019	Preact (via preact/compat) en la landing
ADR-020	Gate de análisis estático del backend (Checkstyle, PMD, SpotBugs, FindSecBugs)
ADR-021	DTOs de respuesta en los endpoints del gestor (cierre de RAPI-DTO-001 / ENTITY_LEAK)
ADR-022	Gestor responsive (sidebar colapsable) + PWA instalable
ADR-023	Agregar un sorteo suelto (ad-hoc) a una campaña

ADR-024	Concurrency hardening: bloqueo optimista + serialización de ejecución por campaña
ADR-025	Acta con premio original congelado + separación del estado de entrega; plazo visible; paginación de clientes
ADR-026	Escaneo de DNI con decodificador PDF417 propio (independiente del navegador)
ADR-027	Reconocimiento del dispositivo por secreto opaco (memoria del DNI sin guardar el DNI)
ADR-028	Completar y corregir datos del cliente desde la landing
ADR-029	Configuración operativa completa FUERA del jar (fail-fast en todo perfil)

ADR-001 — Bucketing de chances a sorteo por ventana de emisión

- **Status:** accepted (D1 del BRIEF, aceptada por el operador — no re-litigar)
- **Context:** Cada sorteo semanal debe saber qué chances le pertenecen. Los clientes pueden guardar vouchers y registrarlos tarde.
- **Alternatives:** por fecha de registración (abre el hueco de guardar vouchers para la semana "conveniente"); por ventana de emisión (elegida).
- **Decision:** Cada sorteo toma las chances con **fecha de emisión** en `[fecha_sorteo_anterior, fecha_sorteo_actual)`. El primer semanal toma desde el inicio de vigencia. Lo emitido tras el último semanal cae en el **sorteo de cierre**.
- **Consequences:** El query de elegibles del acta es determinista y auditable; un voucher registrado tarde participa del sorteo de su semana de emisión si este no se ejecutó aún, o del siguiente que lo capture por ventana.

Implementation Plan

- Query de elegibles en el motor de sorteo (`backend`): filtro por `chance.fecha_emision` contra la ventana del sorteo + exclusión ADR-003.
- La ventana se congela en el snapshot del acta (`filtros`).

Verification

- [ ] Test: chance emitida lunes 08/09 y registrada el 16/09 entra al sorteo del 21/09 (ventana `[14/09, 21/09)`), no al del 14/09.
- [ ] Test: chance emitida después del 05/10 cae en el sorteo de cierre.
- [ ] El acta persiste la ventana usada en `filtros`.

ADR-002 — Toda decisión temporal por fecha de emisión del ticket

- **Status:** accepted (D2 del BRIEF; invariante en CONSTITUTION)
- **Context:** El voucher viaja autocontenido; el cliente puede registrarlo días después de la compra. ¿Qué fecha manda para vigencia, tope diario y bucketing?
- **Alternatives:** fecha de registración (manipulable por el cliente reteniendo vouchers); fecha de emisión del token, offsets 8/10 (elegida).
- **Decision:** Validez de la promo (casos C/D), día del tope diario y bucketing de chances resuelven SIEMPRE por **fecha de emisión** embebida y firmada en el token.
- **Consequences:** Cierra el hueco de guardar vouchers. Un voucher emitido dentro de vigencia y registrado más tarde participa: su chance cae en la ventana que corresponde a su emisión (ADR-001), con el sorteo de cierre como última ventana. La fecha de registración queda solo como dato operativo/antifraude.

Implementation Plan

- Decodificador del token expone `fecha_emision` (días desde 01/01/2026 + minutos).
- Todos los predicados temporales del flujo A–E y del motor usan ese valor.

Verification

- [ ] Test: voucher emitido 09/10 23:50 y registrado 11/10 → participa (cierre).
- [ ] Test: voucher emitido 11/10 (post-vigencia) → caso C, sin chance.
- [ ] Test: tope diario cuenta por día de emisión, no de registro.

ADR-003 — Exclusión de ganadores por adjudicación + dedup por DNI intra-acta

- **Status:** accepted (D3 del BRIEF; invariante en CONSTITUTION)
- **Context:** Las bases definen que quien ya ganó no vuelve a ganar. ¿Excluir por "retiró el premio" o por "salió sorteado"? ¿Y dentro de un mismo acta?
- **Alternatives:** exclusión por retiro (deja ganar dos veces a quien aún no retiró); exclusión por adjudicación (elegida).
- **Decision:** Un DNI **adjudicado** (ganador en un acta, retire o no) queda excluido de los elegibles de todos los sorteos siguientes de la campaña. Intra-acta, al recorrer la lista ordenada se **deduplica por DNI**: un DNI con varias chances gana a lo sumo un premio por acta.
- **Consequences:** La exclusión es computable en el momento del sorteo sin depender del estado de entrega; los suplentes promovidos también son adjudicados.

Implementation Plan

- Query de elegibles: `NOT EXISTS (resultado_sorteo ganador/suplente_promovido con mismo cliente)` dentro de la campaña.
- Motor: al recorrer la lista ordenada, saltar chances de DNIs ya premiados en este acta.

Verification

- [ ] Test: ganador de semana 1 con nuevas chances no aparece elegible en semana 2.
- [ ] Test: DNI con 3 chances en el acta recibe un solo premio; sus otras chances se saltan.
- [ ] Test: suplente promovido queda excluido de sorteos siguientes.

ADR-004 — Acta reproducible con snapshot completo

- **Status:** accepted (D4 del BRIEF; invariante en CONSTITUTION)
- **Context:** El sorteo debe ser auditable ante clientes y organismos: cualquier reclamo debe poder reproducir el resultado exacto.
- **Alternatives:** loguear solo ganadores (no reproducible); snapshot completo (elegida).
- **Decision:** Cada ejecución persiste: lista ordenada de `chance_id` elegibles + hash de la lista + seed + `algo@version` + filtros aplicados + prelación de premios + ganadores y suplentes en orden. El acta ejecutada es **inmutable**.
- **Consequences:** `lista + seed + algo@version → mismo resultado`, siempre. Cambiar el algoritmo exige nueva `algo@version` (las actas viejas se reproducen con la suya).

Implementation Plan

- Motor de sorteo determinista: PRNG sembrado (seed) + shuffle/selección documentada, versionada como `algo@version` (ej. `fisher-yates-sha256@1`).
- Snapshot serializado en `acta` antes de publicar resultados; hash de la lista para detectar cualquier alteración.
- Golden test de reproducibilidad (obligatorio, CONSTITUTION).

Verification

- [ ] Golden: acta conocida (lista + seed + version) reproduce ganadores y suplentes exactos.
- [ ] Test: re-ejecutar el motor sobre el snapshot del acta da idéntico resultado.
- [ ] No existe endpoint ni operación que modifique un acta ejecutada.

ADR-005 — Padrón con un solo tipo Cliente

- **Status:** accepted (D5 del BRIEF)
- **Context:** El alta post-campaña (Pantalla 4a) registra menos campos que el alta en vigencia (Pantalla 1b). ¿Dos entidades (`Cliente` + `Lead`) o una?
- **Alternatives:** entidad `Lead` aparte (duplica identidad y migraciones al convertirse en cliente); un solo `Cliente` con campos nullable (elegida).
- **Decision:** Un único `Cliente` con `fecha_nacimiento` / `sexo` / `email` nullable y sus consentimientos asociados. El alta post-campaña es el mismo `Cliente` con menos campos.
- **Consequences:** El padrón es uniforme para acciones comerciales; los campos faltantes se completan en participaciones futuras sin migrar registros.

Implementation Plan

- Tabla `cliente` única, `dni` unique; `origen` (participacion | post_campania) como dato.
- El alta 4a crea `Cliente` + `ConsentimientoPrivacidad` , sin `AceptacionBases` ni chance.

Verification

- [ ] Test: alta 4a crea Cliente con nullables vacíos y consentimiento de privacidad.
- [ ] Test: ese mismo DNI participando en una campaña futura completa sus campos sin duplicar registro.

ADR-006 — Tope diario: rama propia, rechazo sin consumo

- **Status:** accepted (D6 del BRIEF; valor confirmado en discovery: 5)
- **Context:** Antifraude exige limitar chances por DNI por día. ¿Qué pasa con el voucher que supera el tope: se consume o no?
- **Alternatives:** consumir el voucher al rechazar (quema la chance del cliente por un límite temporal — injusto); rechazar sin consumir (elegida).
- **Decision:** El flujo tiene rama "**tope diario alcanzado**". Como el día se resuelve por emisión (ADR-002), el voucher que supera el tope se **rechaza sin consumir** y se informa al cliente. Tope configurable por campaña; valor de esta campaña: **5**.
- **Consequences:** El día de emisión de un voucher es fijo, por lo que el rechazo por tope es definitivo para ese DNI: con 5 chances registradas de un día de emisión, todo voucher adicional emitido ese día queda afuera. No se consume el token (el registro interno guarda el intento en `VoucherRechazado` para señales antifraude), pero no existe "reintentar mañana": mañana el voucher sigue perteneciendo al mismo día de emisión ya colmado.

Implementation Plan

- Chequeo transaccional previo al consumo: `COUNT(chance WHERE cliente + dia_emision) >= tope` → rechazo informativo sin insertar chance ni marcar el token como usado.
- Parámetro `tope_chances_dni_dia` en `campania`.

Verification

- [ ] Test: 5 chances del día X registradas; 6º voucher emitido el día X → rechazado, token NO consumido, mensaje de tope.
- [ ] Test: mismo DNI con voucher emitido el día X+1 → participa normal.
- [ ] Test: el rechazo queda en registro interno con motivo `tope_diario`.

ADR-007 — Consentimiento: dos objetos distintos

- **Status:** accepted (D7 del BRIEF)
- **Context:** La Ley 25.326 exige consentimiento para datos personales; las bases de una promoción son otro acto jurídico, versionado y por campaña.
- **Alternatives:** un booleano único (mezcla actos de distinta vigencia y alcance); dos objetos (elegida).
- **Decision:** `ConsentimientoPrivacidad` (permanente, por cliente) y `AceptacionBases` (por campaña, con `version_bases` + `fecha_hora`). En Pantalla 2/1b el botón de confirmar participación **implica** la aceptación de las bases de esta campaña (UX de un toque, con el link a las bases visible).
- **Consequences:** El padrón sobrevive a la campaña con su consentimiento propio; auditoría exacta de qué versión de bases aceptó cada participante y cuándo.

Implementation Plan

- Dos tablas (ver DOMAIN-MODEL). Alta 1b: crea ambos. Confirmación E: asegura `AceptacionBases` de esta campaña si no existe.
- Landing: footer con link a `/bases/{version}.pdf` en todas las pantallas.

Verification

- [ ] Test: primera participación crea consentimiento + aceptación con versión y timestamp.
- [ ] Test: segunda participación en la misma campaña no duplica la aceptación.
- [ ] Las bases son descargables desde la landing sin JS adicional.

ADR-008 — Estado terminal NO_OTORGADO

- **Status:** accepted (D8 del BRIEF)
- **Context:** Tras el cierre de campaña ya no hay "próximos sorteos" a los que devolver una unidad no entregada. La máquina de estados quedaba abierta.
- **Alternatives:** devolver a `Disponible` para siempre (stock fantasma post-campaña); estado terminal explícito (elegida).
- **Decision:** `NO_OTORGADO` como estado terminal, distinto de `Disponible`, con motivo (suplentes agotados | plazo de retiro vencido) + auditoría completa.
- **Consequences:** El inventario de la campaña cierra exacto: cada unidad termina en `Otorgado` o `NO_OTORGADO`; la rendición ante Caracol es un COUNT.

Implementation Plan

- Transición `Sorteado → NO_OTORGADO` (y `Disponible → NO_OTORGADO` al cierre para remanentes) con `EventoPremio` obligatorio (motivo, usuario, fecha).
- Pantalla de ganadores: acción explícita del operador, nunca automática.

Verification

- [ ] Test: unidad con suplentes agotados pasa a NO_OTORGADO con motivo y evento.
- [ ] Test: NO_OTORGADO no cuenta en `stock_disponible` ni es asignable.
- [ ] Test: transición sin motivo → rechazada.

ADR-009 — Ejecución del sorteo por decisión humana

- **Status:** accepted (D9 del BRIEF; invariante en CONSTITUTION)
- **Context:** Los sorteos tienen hora publicada en bases. ¿Scheduler automático o disparo manual?
- **Alternatives:** scheduler (ejecuta con datos que nadie revisó; falla silenciosa un lunes 10:00 sin operador delante); disparo humano (elegida).
- **Decision:** El sorteo se ejecuta **solo** por acción del operador desde el gestor. La hora programada es referencia publicada en bases, nunca un trigger.
- **Consequences:** Siempre hay un humano responsable delante del acta; no existe código de scheduling que auditar; un feriado o demora se maneja operativamente.

Implementation Plan

- Botón "Ejecutar sorteo" en el panel (con confirmación y resumen de elegibles).
- Ningún cron/scheduler en el backend para ejecución de sorteos.

Verification

- [ ] No existe código de scheduling de sorteos en el repo.
- [ ] Test: la ejecución exige usuario autenticado del gestor y queda auditada.

ADR-010 — Claves HMAC por keyId en tabla cifrada

- **Status:** accepted (D10 del BRIEF; secreto cifrado como invariante en CONSTITUTION)
- **Context:** La firma del token es HMAC-SHA256 (simétrica): el mismo secreto vive en ~250 cajas y en el servidor. Hay que poder rotar sin corte de servicio.
- **Alternatives:** JKS/keystore de PKI (concepto de firma asimétrica que acá no aplica); secreto único hardcodeado (sin rotación); tabla `clave_firma` por `key_id` (elegida).
- **Decision:** "Keystore" = tabla `clave_firma(key_id, secreto_cifrado, estado, vigencia_desde, vigencia_hasta)`. El server valida contra la clave que indica el `keyId` (offset 0) del token. Rotación: alta de nueva clave con nuevo `keyId`, distribución a cajas, ambas válidas durante la transición.
- **Consequences:** Compromiso de una caja se mitiga rotando `keyId` — **riesgo aceptado y documentado** (RISKS.md). El secreto se guarda cifrado at-rest (master key vía env/Vaultwarden), nunca en claro en repo ni logs.
- **Riesgo aceptado:** quien extraiga el secreto de una caja puede fabricar vouchers válidos hasta la rotación. Mitigación: rotación de keyId + señales antifraude (concentración por caja/DNI/conexión).

Implementation Plan

- Tabla `clave_firma`; cifrado del secreto con AES-GCM y master key por variable de entorno; caché en memoria del secreto descifrado.
- Validación: leer `keyId` del token → clave correspondiente → HMAC en tiempo constante.
- **Precisión (QA pass 1):** la aceptación de una clave en validación se resuelve por `estado` (activa y rotada validan; revocada no), NUNCA por reloj contra `vigencia_*`: un voucher emitido con una clave luego rotada debe seguir validando ("rotación sin corte"). Las fechas de vigencia son informativas para operación/rotación en cajas.
- Clave de prueba para el ambiente de test (M6) con `keyId` reservado.

Verification

- [ ] Test: token firmado con clave rotada (estado activa en su vigencia) valida OK.

- [ ] Test: `keyId` inexistente/revocado → caso A (error genérico).
- [ ] El secreto no aparece en claro en repo, logs ni dumps de configuración.

ADR-011 — Idempotencia por constraints de base de datos

- **Status:** accepted (D11 del BRIEF; consumo atómico como invariante en CONSTITUTION)
- **Context:** Conexiones 3G de salón: envíos duplicados, retries de formularios cortados y doble tap son la norma, no la excepción.
- **Alternatives:** locks de aplicación (frágiles ante múltiples workers); idempotencia a nivel DB (elegida).
- **Decision:** Alta de cliente idempotente por `dni` UNIQUE; consumo de voucher atómico por `hash_token` UNIQUE (inserción única de la chance). El retry de una registración cortada resuelve a **caso B** (ya participó), nunca a doble chance.
- **Consequences:** La corrección no depende del comportamiento del cliente ni del número de instancias del backend; el conflicto DB se traduce a la respuesta del caso B.

Implementation Plan

- Constraints UNIQUE (`cliente.dni` , `chance.hash_token`) + manejo del conflicto en la transacción de participación (INSERT ... ON CONFLICT / captura de duplicate key).
- La transacción alta-de-cliente + chance + aceptación de bases es una sola unidad.

Verification

- [ ] Test de concurrencia: N envíos simultáneos del mismo token → exactamente 1 chance.
- [ ] Test: retry post-corte del alta 1b → caso B con fecha de la participación original.
- [ ] Test: dos altas simultáneas del mismo DNI → un solo Cliente.

ADR-012 — Elegibles insuficientes: el motor sortea lo que hay

- **Status:** accepted (D12 del BRIEF)
- **Context:** Puede haber menos chances elegibles que `ganadores + suplentes` (semana floja, exclusiones acumuladas).
- **Alternatives:** fallar la ejecución (bloquea el sorteo publicado); esperar más participaciones (viola la fecha publicada); sortear lo disponible (elegida).
- **Decision:** El motor sortea sobre los elegibles existentes. Si alcanzan para K < N posiciones, el acta registra K resultados y las unidades sin ganador quedan para decisión del operador (devolución al stock / NO_OTORGADO según el momento).
- **Consequences:** El sorteo nunca se bloquea; el acta refleja fielmente la escasez.

Implementation Plan

- Motor: `min(elegibles_unicos_por_dni, posiciones)` posiciones cubiertas, en orden.
- Unidades no sorteadas permanecen `Asignado` hasta acción del operador (auditada).

Verification

- [ ] Test: 3 elegibles (post-dedup) con 5 premios + 10 suplentes → acta con 3 resultados, 2 unidades sin ganador siguen Asignado.
- [ ] Test: 0 elegibles → acta vacía válida, sin error.

ADR-013 — Dominio Campaña / Sorteo

- **Status:** accepted (D13 del BRIEF; "única vigente" como invariante en CONSTITUTION)
- **Context:** El PDF usa "sorteo" para la campaña entera y para cada evento semanal. Términos sobrecargados generan bugs de alcance (¿"único vigente" aplica a qué?).
- **Alternatives:** entidad única "Sorteo" con hijos implícitos; jerarquía explícita (elegida).
- **Decision:** `Campaña` contiene N `Sorteos` (`Sorteo Semana 1..N` + `Sorteo de cierre`). El invariante "único vigente" es sobre **Campaña**, garantizado a nivel base de datos. Glosario canónico en CONTEXT.md.
- **Consequences:** Textos de landing, vigencia, reglas y clave viven en Campaña; fechas, premios asignados y acta viven en cada Sorteo.

Implementation Plan

- Constraint parcial: única fila `campania` con `estado = 'vigente'` (unique index sobre `estado` filtrado, o columna de exclusión equivalente).
- La generación de sorteos por regla ("semanal, lunes 10:00") produce filas editables + el cierre.

Verification

- [ ] Test: segunda campaña a vigente con una ya vigente → rechazada por la DB.
- [ ] Test: la regla de programación genera 5 semanales + cierre editables.

ADR-014 — Política de promoId: DIFERIDA

- **Status:** deferred (D14 del BRIEF — no abrir en esta entrega)
- **Context:** El token lleva `promoId` (offset 1, 4 bytes) tal cual el Anexo A, nativo del backoffice de promociones.
- **Decision:** La política de asignación/reuso de `promoId` entre altas y campañas se define más adelante. **No es alcance de esta entrega y no se re-litiga en discovery ni en devloop.** El sistema lo trata como identificador opaco que debe coincidir con el `promo_id` configurado en la Campaña vigente.
- **Consequences:** El backend solo compara igualdad `token.promoId == campania.promo_id`. Cualquier semántica adicional (rangos, reuso, colisiones históricas) queda explícitamente fuera.

Verification

- [ ] Ningún código asume semántica de `promoId` más allá de la igualdad con la campaña.

ADR-015 — Topología: monorepo con dos frontends de presupuesto distinto

- **Status:** accepted (Opción B del BRIEF §3 ya decidida; estructura de repos cerrada por discovery)
- **Context:** El BRIEF fija la topología de frontends (Opción B: gestor con stack WorkDone completo; landing separada y mínima) y delega en discovery la estructura de repos (mono/multi).
- **Alternatives:**
- Tres repos (backend / landing / gestor): aísla builds pero triplica gestión de versiones, PRs y puesta en marcha para un equipo chico con deadline 01/09.
- Monorepo con tres módulos (elegida): un clone, un pipeline, versionado atómico de contratos API ↔ frontends.
- **Decision:** Monorepo `SistemaPremios` con:
- `backend/` — M3 (API pública + API gestor + motor + DB).
- `landing/` — M4: React 18.3 + Vite + TS **pelado** (sin shadcn/Radix), presupuesto de carga instantánea en 3G, selects nativos, JSON `snake_case` mapeado en `src/types/api.ts`.
- `gestor/` — M5: stack WorkDone completo (React 18.3 + Vite 5 + TS 5.6 + Tailwind 3.4 + shadcn/ui `new-york` parcial [alert/badge/button/card/input/label/table + toast] + Radix solo label/slot/toast + lucide-react + TanStack Query v5 + axios + react-router-dom v6 + zustand v5 + react-hook-form + zod + react-konva).
- Reglas duras de UI del BRIEF §3: sin otra librería de UI; ante faltantes, Tailwind puro antes que un primitive Radix nuevo; selects nativos con `selectCls`.
- **Consequences:** Dos builds independientes con presupuestos de peso distintos; la landing no hereda el bundle del gestor. `uscha.config.json` registra los tres módulos como repos lógicos.

Implementation Plan

- Estructura de directorios de arriba + `uscha.config.json` `repos[]` actualizado.

- Cada frontend con su `package.json` y build propio; backend sirve estáticos o se publican por separado detrás del mismo subdominio (decisión operativa M6).

> **Nota (M6, decisión operativa cerrada):** se optó por que el backend sirva los estáticos — un único jar Spring Boot empaqueta `landing/dist` bajo `static/` y `gestor/dist` bajo `static/gestor/` (perfil Maven `full`, `frontend-maven-plugin`), en vez de publicarlos por separado detrás de un reverse proxy. Ver SPEC.md §10 y `SpaForwardController` (backend).

Verification

- [ ] `landing/package.json` no contiene shadcn, Radix ni librerías de UI pesadas.
- [ ] Build de landing y gestor generan bundles separados.
- [ ] `gestor/` replica el stack WorkDone exacto (versiones mayores).

> **Nota (enmienda, ver ADR-019):** la landing (`landing/`) migró su runtime de React 18.3 a Preact vía `preact/compat` (docs/E2E-PLAN.md A9, presupuesto de 3G) — el código fuente sigue escrito contra la API de React, solo cambia qué lo ejecuta. El gestor no se ve afectado por esta enmienda: sigue en React 18.3 tal cual queda descripto arriba.

ADR-016 — Stack backend: Java + Spring Boot + SQL Server

- **Status:** accepted (stack Java derivado del pack; motor de base **decidido por el operador en devloop: SQL Server** — reemplaza la propuesta inicial PostgreSQL)
- **Context:** El módulo `qrtoken` existe validado en **C89 + espejo Java** y la CONSTITUTION obliga a reusarlo, no reimplementarlo. El servidor destino es una máquina del cliente (sin nube), con backup diario y servicio con reinicio automático. WorkDone (referencia de la casa) es una webapp Java (`src/main/webapp`).
- **Alternatives:**
- Node/TypeScript end-to-end: unifica lenguaje con los frontends, pero obliga a portar `qrtoken` (violación de REUSE-FIRST) o a un bridge frágil.
- Java + Spring Boot 3 (elegida): consume el espejo Java de `qrtoken` tal cual, stack conocido por LaEmpresa, despliegue single-jar como servicio.
- **Decision:** `backend/` en **Java 17 + Spring Boot 3** (single jar), **SQL Server 2022** como base (instancia de LaEmpresa; DB `SistemaPremios`), **Flyway** (módulo `flyway-sqlserver`) para migraciones versionadas con rollback documentado, Maven como build. El módulo `qrtoken` (espejo Java) se integra como fuente vendoreada con marcador de procedencia.
- **Tests contra el motor real:** la suite corre contra la misma instancia SQL Server local (DB `SistemaPremios_test`), NO contra H2/HSQLDB — los invariantes viven en constraints dialecto-específicos (unique filtrado de "única campaña vigente", consumo atómico) y un motor sustituto no los prueba.
- **Consequences:** Validación de token idéntica a la de caja (verificación cruzada C↔Java ya hecha en prototipo); constraints e idempotencia (ADR-011) sobre SQL Server (índices filtrados + unique constraints); un solo artefacto desplegable + dos bundles estáticos. Credenciales de DB SOLO por variables de entorno / config local no trackeada (CONSTITUTION: secrets nunca en repo).

Implementation Plan

- `backend/pom.xml` Spring Boot 3 (web, data-jpa/jdbc, validation, security para gestor), driver `mssql-jdbc` , `flyway-sqlserver` .

- Módulo `qrtoken` Java vendoreado (`QrToken.java` + package + autotest como JUnit).
- Perfiles: `dev` (clave de firma de prueba; DB local por env), `test` (`SistemaPremios_test`), `prod` (master key y credenciales por env).

Verification

- [ ] `qrtoken` no está reimplementado: la validación llama al espejo Java existente.
- [ ] Migraciones Flyway con par down/rollback documentado por cada una destructiva.
- [ ] `mvn -q test` corre verde en `backend/` contra `SistemaPremios_test` (SQL Server).
- [ ] Ninguna credencial de DB en archivos trackeados.

ADR-017 — Suite E2E visual/funcional con Playwright contra el jar empaquetado

- **Status:** accepted (operador aprobó la dependencia nueva el 2026-08-21)
- **Context:** El sistema reparte 100 premios con acta legal. Los 117 tests de backend y 71 de frontend prueban invariantes y componentes, pero nada prueba el flujo completo en un navegador real sobre la topología de producción (un jar que sirve API + landing + gestor). La landing se usa en un 99 % desde celulares baratos en el salón.
- **Alternatives:**
- Pruebas manuales con checklist (SMOKE-M6): necesarias para lo físico (cámara, ticket real) pero no repetibles ni escalables a 100+ casos.
- Agente con navegador (Claude en Chrome): útil para explorar; no es determinista ni corre en CI.
- Playwright (elegida): emulación de dispositivos reales (Chromium Android, **WebKit** como motor de Safari iOS), screenshots con baseline, throttling de red, paralelismo, reportes JUnit que el ledger ingiere.
- **Decision:** Carpeta `e2e/` propia (`@playwright/test` , Chromium + WebKit). El sistema bajo prueba es el **jar** arrancado por el runner con perfil `e2e` : DB `SistemaPremios_e2e` (clean+migrate por corrida), rate limit desactivado salvo un test aislado, clave demo por seed. Los tests fabrican sus tokens (helper TS espejo de `QrToken` , validado contra el token de referencia) y eligen la **fecha de emisión** que necesitan: el bucketing es por emisión, no hace falta manipular el reloj. Proyectos: `android-chico` (360×740, baselines principales), `iphone-webkit` (390×844), `android-grande` , `escritorio` (humo); gestor en escritorio + tablet. Plan y matriz completa: `docs/E2E-PLAN.md` .
- **Consequences:** ~130 tests, 8-12 min con 4 workers. Baselines visuales atados a la máquina que los generó (regenerar en CI Linux). Nunca corre contra `SistemaPremios` ni `_test` . El escaneo PDF417 queda fuera (cámara física → SMOKE-M6). Los reportes entran al ledger como repo `e2e` (dimensión integration).

Implementation Plan

- `e2e/` con `playwright.config.ts` (webServer = `java -jar` con env e2e), `helpers/token.ts` , `helpers/api.ts` (seed vía API del gestor), `fixtures/` , specs por grupo

A/B/C del plan.

- Backend: `application-e2e.properties` (DB e2e, flyway clean habilitado, rate limit off).
- `uscha.config.json`: repo `e2e` tipo node para el ledger.

Verification

- [ ] `npx playwright test` verde local con los 4 proyectos; JUnit en `e2e/reports/junit.xml`.
- [ ] Journey maestro: otorgados + no otorgados + stock = 100 al final de la campaña.
- [ ] Ninguna conexión a DBs que no sean `SistemaPremios_e2e`.

ADR-018 — Upgrade a Spring Boot 4 (framework 7, security 7, hibernate 7, jackson 3) + consola boot-ui (solo dev)

- **Status:** accepted (operador aprobó el upgrade y la consola de desarrollo, 2026-08-21)
- **Context:** Spring Boot 3.4 salió de soporte OSS en 2025-12; el lanzamiento a producción es 2026-09-01, así que seguir en 3.4 sin parches de seguridad hasta esa fecha no es aceptable. Además se pidió sumar `boot-ui` (consola de desarrollo de Julien Dubois: introspección de salud/métricas/JPA/seguridad en el navegador), que requiere Spring Boot 4.
- **Alternatives:**
- Quedarse en Spring Boot 3.5 (última minor de la línea 3.x, soportada más tiempo que 3.4): evita el trabajo de migración de Jackson 3 / Spring Framework 7, pero `boot-ui` no corre ahí — quedaría afuera del alcance pedido.
- Spring Boot 4.x (elegida): versión GA más nueva resoluble en Maven Central al momento del upgrade, **4.1.1** (verificado contra `maven-metadata.xml` de `spring-boot-starter-parent` — 4.1.1 es la última release, 4.2.0-M1 es milestone, no GA). Es la única línea que soporta `boot-ui`.
- **Decision:** `backend/pom.xml` parent → `spring-boot-starter-parent:4.1.1`. Se agrega `com.julien-dubois.bootui:bootui-spring-boot-starter:1.14.1` (última 1.x en Maven Central al momento del upgrade) con su comportamiento **AUTO** por defecto: activa solo con perfil `dev` / `local` (o con `spring-boot-devtools` presente), se auto-apaga en `prod` / `production`, y rechaza requests no-loopback aunque esté activa (`bootui.allow-non-localhost=false` por defecto). El perfil `e2e` (`application-e2e.properties`, nuevo — reservado por ADR-017 pero sin contenido hasta ahora) **no** fija `bootui.enabled` a mano: `AUTO` ya resuelve sola los tres casos que importan (ver Consequences).
- **Consequences:**
- **Jackson 3:** Spring Boot 4 trae Jackson 3, que renombra los paquetes de `jackson-core` y `jackson-databind` de `com.fasterxml.jackson.*` a `tools.jackson.*` (las *anotaciones*, `com.fasterxml.jackson.annotation.*`, se quedan donde estaban — no cambian). `ObjectMapper`, `TypeReference` y la excepción de (de)serialización se mudan; `JsonProcessingException` (checked, extendía `IOException`) desaparece a favor de `JacksonException` (unchecked). Afecta `SorteoService`, `SeedMinimo`, `GestorApiTest`,

`SeedMinimoTest`. `spring.jackson.property-naming-strategy: SNAKE_CASE` sigue siendo la property correcta y el contrato snake_case de toda la API sigue intacto — confirmado con la suite completa y a mano contra el jar real (`POST /api/gestor/campanias` con body snake_case).

- **Flyway modularizado:** Boot 4 separó la autoconfiguración de Flyway a un módulo propio (`spring-boot-flyway` / `spring-boot-starter-flyway` / `spring-boot-starter-flyway-test`). `FlywayMigrationStrategy` se muda de `org.springframework.boot.autoconfigure.flyway` a `org.springframework.boot.flyway.autoconfigure`. `backend/pom.xml` reemplaza la dependencia directa `flyway-core` por `spring-boot-starter-flyway` (que la trae transitiva, versión 12.4.0 gestionada por Boot); `flyway-sqlserver` se queda como estaba (Boot no la gestiona).
- **MockMvc modularizado:** Boot 4 separó el soporte de test de MockMvc a `spring-boot-starter-webmvc-test` / `spring-boot-webmvc-test`. `AutoConfigureMockMvc` se muda de `org.springframework.boot.test.autoconfigure.web.servlet` a `org.springframework.boot.webmvc.test.autoconfigure`. `MockMvc`, `MockMvcRequestBuilders`, `MockMvcResultMatchers` (paquete `org.springframework.test.web.servlet`, de Spring Framework, no de Boot) no cambiaron. No hizo falta `spring-boot-starter-data-jpa-test`: ningún test de este repo usa `@DataJpaTest`.
- **Regresión real de Spring Framework 7 en resolución de `@ControllerAdvice` (HIGH, detectada por el smoke manual, no por la suite):** sin `@Order` explícito, Spring elige entre `@ControllerAdvice` candidatos por orden de registro del bean, no por especificidad del `basePackages`. Bajo Spring Framework 7 ese orden cambió lo suficiente como para que el catch-all de `Exception` de `GlobalExceptionHandler` ganara la carrera antes que el `DataIntegrityViolationException` específico de `GestorExceptionHandler` — el alta duplicada de "campaña vigente" volvía **500** en vez de **409**. La suite completa (123 tests, incluido `EsquemaInvariantesTest` que prueba la traducción de excepción a nivel repositorio) pasaba igual, porque nadie ejercitaba el camino completo controller→advice con una inserción real duplicada — `GestorExceptionHandlerTest` sólo invoca el handler aislado. Se agregó `@Order(HIGHEST_PRECEDENCE)` a `GestorExceptionHandler` y `@Order(LOWEST_PRECEDENCE)` explícito a `GlobalExceptionHandler`, más un test de regresión end-to-end (`GestorApiTest.crearCampania_duplicadaVigente_409`) que pega contra el controller real, no contra el handler aislado.
- **`boot-ui` no ensancha la seguridad existente:** detecta Spring Security y se auto-registra permitAll SOLO en `/bootui`, `/bootui/**`, `/bootui/api`, `/bootui/api/**` (log propio de `BootUiSpringSecurityAutoConfiguration` lo confirma); `/api/gestor/**` sigue en 401 sin auth, `/api/salud` sigue público, verificado a mano.

- **Trae actuator + micrometer + opentelemetry + archunit transitivos** (`bootui-spring-boot-starter` depende de `spring-boot-starter-actuator`, `spring-boot-micrometer-tracing-opentelemetry`, `micrometer-tracing-bridge-otel`, `archunit` — todos en scope `compile`, siempre en el jar, no solo en un perfil dev). Es necesario para que la introspección de `boot-ui` funcione; los endpoints de actuator expuestos por web siguen limitados al default de Boot (`health`, `info`) — no se cambió `management.endpoints.web.exposure.include`.
- **Los assets estáticos de la consola (HTML/JS/CSS) quedan descargables incluso con `bootui.enabled` en AUTO-apagado o en `prod`** (confirmado a mano: `GET /bootui/index.html` → 200 con `prod` solo, sin `dev`). `bootui-ui` empaqueta su SPA bajo `classpath:/META-INF/resources/bootui/` (patrón webjar), que Spring sirve como recurso estático genérico apenas el jar está en el classpath — independiente del `enabled` de la librería. La API funcional (`/bootui/api/**`, la única que expone datos vivos de la app) y el filtro loopback-only SÍ están correctamente gateados y devuelven 404/403 fuera de `dev` / `local` — el shell estático es HTML/JS inerte sin backend que lo alimente, severidad baja (revela que la herramienta está en el classpath, no expone datos), pero se documenta en vez de ocultarlo.
- **`application-e2e.properties` no fija `bootui.enabled=OFF`:** se evaluó hacerlo (como pide un enfoque ingenuo de "doble capa de defensa") pero se descartó porque con `--spring.profiles.active=dev,e2e` el último perfil activo pisa al anterior — un OFF fijo en `e2e` ganaría sobre el `dev` explícito y dejaría la consola inalcanzable incluso cuando alguien la pide a propósito para depurar una corrida E2E, rompiendo ese caso de uso. `bootui.enabled=AUTO` (default de la librería, sin pisar nada) ya resuelve los tres casos reales sin este problema: `e2e` sola (CI) apagada, `prod,e2e` apagada, `dev,e2e` prendida — los tres confirmados a mano contra el jar empaquetado real.

Implementation Plan

- `backend/pom.xml`: parent `spring-boot-starter-parent:4.1.1`; `flyway-core` reemplazado por `spring-boot-starter-flyway`; `spring-boot-starter-webmvc-test` agregado (scope test); `com.julien-dubois.bootui:bootui-spring-boot-starter:1.14.1` agregado; `jacoco-maven-plugin` a 0.8.15 (soporte de class files más nuevos).
- Imports migrados a los paquetes nuevos de Jackson 3 (`tools.jackson.*`) y del Flyway/MockMvc modularizados de Boot 4, en `SorteoService`, `SeedMinimo` y los tests que los usan.
- `@Order` explícito en `GestorExceptionHandler` (`HIGHEST_PRECEDENCE`) y `GlobalExceptionHandler` (`LOWEST_PRECEDENCE`) — fix forzado por la regresión de Spring Framework 7 descripta arriba.

- `backend/src/main/resources/application-e2e.properties` (nuevo, sin contenido de `bootui.enabled` — ver Consequences).
- `docs/SMOKE-M6.md` §0: nota de cómo abrir la consola en dev y por qué prod queda apagada sola.
- Tests: `GestorApiTest.crearCampania_duplicadaVigente_409` (regresión del fix de `@Order`), `BootUiConsoleTest` (apagada fuera de dev/local vía `/bootui/api/health`; prendida + loopback-only con perfil dev vía `MockMvc` con `remoteAddr` no-loopback).

Verification

- [x] `mvn -q test` (`SP_TEST_DB_URL` → `SistemaPremios_test3`) verde: 127 tests, 0 failures, 0 errors, 22 clases de test.
- [x] `mvn -q package -DskipTests` empaqueta backend + landing + gestor en un solo jar (perfil `full`).
- [x] Smoke contra el jar en :8093 (perfil `e2e`, DB `SistemaPremios_test3`): `/api/salud` 200, `/gestor/` 200, `/p/x` 200, `/api/gestor/panel` sin auth 401, alta duplicada de campaña vigente 409 (no 500 — regresión encontrada y corregida).
- [x] Smoke `--spring.profiles.active=dev,e2e`: `GET /bootui` 200 desde localhost; rechaza (403) un caller no-loopback (`X-Forwarded-For` spoofeado, mismo mecanismo de confianza de proxy que ya documentaba JD-B1 para el rate limit); `/api/gestor/**` sigue protegido.
- [x] Smoke `--spring.profiles.active=prod,e2e`: `GET /bootui` 404 (capa API/loopback correctamente apagada; el shell estático queda servible como recurso inerte, documentado arriba).
- [ ] Compresión gzip en `/gestor/` — **no verificable**: `server.compression` nunca estuvo configurado en este repo (ni antes ni después del upgrade); hallazgo pre-existente, fuera de alcance de este ADR, no es una regresión del upgrade.

ADR-019 — Preact (via preact/compat) en la landing

- **Status:** accepted (operador 2026-08-22 — enmienda a ADR-015)
- **Context:** docs/E2E-PLAN.md A9 fija el objetivo de performance de la landing: la Pantalla 1a visible en menos de 1.5 s bajo el perfil "3G lento" del e2e (99% de la audiencia real entra desde el celular, operador: "de eso depende que lo usen"). Con el estado inicial embebido en el HTML y los assets precomprimidos (brotli/gzip) ya en producción, el bundle inicial medía ~51 KB gzip / ~45 KB brotli — de los cuales el runtime de React (`react` + `react-dom`) representaba ~45 KB, la mayor parte del peso. El código propio de la landing (componentes, hooks, `ErrorBoundary` de clase) pesa unos pocos KB; el costo real era el framework, no el producto.
- **Alternatives:**
- **Quedarse en React 18.3** — cero riesgo de compatibilidad, pero el runtime sigue pesando ~45 KB gzip sin margen para bajar del objetivo de 1.5 s en 3G real.
- **Preact vía `preact/compat` (elegida)** — Preact (~4 KB) implementa la misma API de React (hooks, componentes de clase, Context, refs) a través de la capa de compatibilidad `preact/compat`; el código fuente no cambia, solo el runtime que lo ejecuta. `@preact/preset-vite` alias `react` / `react-dom` / `react/jsx-runtime` a `preact/compat` / `preact/jsx-runtime` en build time.
- **Reescribir en vanilla JS u otro framework más chico** — descartado: reescritura completa de 8 pantallas + validaciones + escaneo de DNI para un ahorro similar al de la opción 2, con mucho más riesgo y sin reutilizar la suite de tests existente.
- **Decision:** Adoptar Preact vía `preact/compat` **solo en la landing** (`landing/`). El gestor (`gestor/`, SPA interna de uso operativo, sin presupuesto de peso — ver ADR-015) se queda en React 18.3 sin cambios: no hay presión de 3G ni de audiencia masiva ahí.
- `landing/package.json`: `preact` como dependencia; `@preact/preset-vite` reemplaza a `@vitejs/plugin-react` como devDependency (mismo soporte JSX/Babel + HMR, y ya trae el alias react→preact/compat resuelto en vez de declararlo a mano). Los paquetes runtime `react` / `react-dom` (18.3.1) se DESINSTALARON — con el alias, nada los ejecuta nunca; `@types/react` / `@types/react-dom` alcanzan solos para que `tsc` resuelva los tipos (confirmado: `tsc --noEmit` sigue en verde sin los paquetes runtime instalados).
- El código fuente de la landing (componentes, hooks, `ErrorBoundary`) sigue escrito contra la API de React tal cual — ningún import cambia salvo la infraestructura de test (ver

Consequences).

- **Consequences:**
- **Bundle inicial:** 51.33 KB → 14.29 KB gzip (12.89 KB brotli) — el runtime de React/ReactDOM (~45 KB) se reemplaza por Preact (~4 KB); el código propio de la landing (~9-10 KB) queda intacto.
- **API de React intacta:** hooks, `Component` de clase (`ErrorBoundary`), `createRoot` / `StrictMode` , refs y el mapeo `onChange` →evento de input de los formularios (DNI/celular/etc.) funcionan igual — cubierto por la suite existente sin reescribir ningún test de comportamiento.
- **Riesgo de incompatibilidades menores de `preact/compat` :** mitigado por (a) un relevamiento previo del código fuente que confirmó que la landing NO usa `React.lazy` / `Suspense` (el escaneo de DNI carga su chunk pesado con un `import()` dinámico manual, sin la API de lazy-loading de React), `useId` , `createPortal` , `forwardRef` ni `useImperativeHandle` — superficie de compat conocida como más frágil que no aplica acá; y (b) la suite de vitest (51 tests) + el e2e completo corriendo en verde tal cual, sin ajustar ningún assert de comportamiento.
- **Infraestructura de test, sí cambió:** `@testing-library/react` asume `react-dom` real — internamente hace `require("react-dom")` / `require("react-dom/client")` como CommonJS, y Vitest ejecuta esos requires de dependencias de `node_modules` con el `require` nativo de Node (no pasan por el resolver de Vite), así que ni `resolve.alias` ni `vi.mock` los interceptan de forma confiable — terminaba montando los elementos de Preact (ya vía JSX transformado por `@preact/preset-vite`) con el reconciler REAL de `react-dom` (dos runtimes distintos en el mismo árbol: "Objects are not valid as a React child", refs inválidos). Se reemplazó por `@testing-library/preact` (adaptador oficial, misma API `render` / `screen` / `cleanup` / `act` , implementado directamente sobre Preact) en los 6 archivos de test que renderizaban componentes — sin tocar ningún assert de comportamiento.
- **Gestor no cambia:** stack, bundle y tests del gestor quedan exactamente igual que antes de este ADR.
- **Implementation Plan:**
- `landing/package.json` : agregar `preact` ; reemplazar `@vitejs/plugin-react` por `@preact/preset-vite` ; agregar `@testing-library/preact` ; quitar `@testing-library/react` .
- `landing/vite.config.ts` : `preact()` en vez de `react()` en `plugins` .
- `landing/tests/setup.ts` y los 6 archivos de test que renderizan componentes: `@testing-library/react` → `@testing-library/preact` .
- Medir bundle (gzip + brotli) y el test "3G lento" (e2e, 3 corridas) antes/después; ajustar el umbral del test (`docs/adr/../E2E-PLAN.md` A9) al valor medido honesto.
- **Verification:**

- [x] `landing/dist` gzip inicial ≤ 60 KB (AC-24) — 14.29 KB, con amplio margen.
- [x] Suite vitest de landing (51 tests) en verde sin reescribir asserts de comportamiento.
- [x] `tsc --noEmit` y `eslint .` de landing sin errores.
- [x] E2E completo (Playwright, jar empaquetado) en verde.
- [x] Gestor sin cambios de stack, bundle ni tests.

ADR-020 — Gate de análisis estático del backend (Checkstyle, PMD, SpotBugs, FindSecBugs)

- **Status:** accepted (operador aprobó el gate, 2026-08-22)
- **Context:** el ledger uscha (`QA-LEDGER.json`) marcaba `backend.static_gate` como UNMEASURED — el placeholder `java-qa-gate` quedó pendiente desde el bootstrap del proyecto (`CLAUDE.md` §"Project adapter", línea "Static gate: `<e.g. mvn -q verify -Pqa → ...>` "). `landing` y `gestor` ya reportan static-gate real (eslint + tsc, `QA-LEDGER.json`), pero `backend` no tenía ningún linter/SAST corriendo. Este es un sistema con actas legales de sorteo (ver ADR-004) y datos personales de clientes (padrón, DNI, celular) — necesita un piso de linting + SAST, no solo tests unitarios.
- **Alternatives:**
- **Sin gate** — descartada: es exactamente el estado actual (UNMEASURED) que este ADR viene a cerrar; el ledger no puede medir lo que no se ejecuta.
- **SonarQube** — hay integración disponible en el toolkit (skill `sonarqube:sonar-*`), pero requiere levantar/mantener un server (SonarQube Community o Cloud) y autenticar el repo contra él; para un backend de un solo módulo Maven es una pieza de infraestructura adicional a operar y actualizar, desproporcionada frente al problema (medir el backend, no correr una plataforma).
- **Plugins Maven locales (elegida)** — `maven-checkstyle-plugin` + `maven-pmd-plugin`
  - `spotbugs-maven-plugin` + `findsecbugs-plugin` : corren dentro del build existente, sin servicio externo, producen XML que el ledger ya sabe parsear (`qa_ledger.py ingest-gate` busca exactamente `target/checkstyle-result.xml` , `target/pmd.xml` , `target/spotbugsXml.xml`), y son la misma familia de herramientas que ya usa el ecosistema Java/Maven sin agregar infraestructura.
- **Decision:** agregar un perfil Maven `qa` a `backend/pom.xml` , atado a la fase `verify` (no a `test` , para que `mvn test` — el comando de todos los días — siga rápido y sin este costo). El perfil corre:
- `maven-checkstyle-plugin` con un ruleset propio de alta señal (`backend/config/checkstyle.xml`), no `sun_checks` (demasiado ruidoso para este tamaño de proyecto): imports con wildcard, catch vacío, pares equals/hashCode, llaves obligatorias, imports sin usar, `System.out` / `System.err` prohibidos en prod. Magic numbers y javadoc quedan OFF (no aportan señal aquí).

- `maven-pmd-plugin` con ruleset propio (`backend/config/pmd-ruleset.xml`): `bestpractices` + `errorprone` núcleo, excluyendo reglas ruidosas para este codebase (`LawOfDemeter`, `OnlyOneReturn`, guard de log statements, reglas de comentarios).
- `spotbugs-maven-plugin` (con el plugin `findsecbugs-plugin` sumado a su classpath), `effort=Max`, `threshold=Medium`, salida XML.
- Ninguno de los tres falla el build (`failOnViolation=false` / `failOnError=false`): **el ledger es el gate, no Maven** — el perfil solo tiene que producir las tres XML; la severidad se evalúa en `qa_ledger.py ingest-gate` (FindSecBugs pisa a HIGH/SECURITY, SpotBugs prio1→HIGH, Checkstyle error→HIGH, PMD prio1→BLOCKER, prio2→HIGH), igual que ya ocurre con eslint/tsc en `landing` / `gestor`.
- **Consequences:**
- `mvn test` no cambia: sigue sin correr análisis estático, mismo tiempo que hoy.
- `mvn verify -Pqa` (o `-Pqa` explícito en CI/ledger) es el único comando nuevo, y es el que documenta `CLAUDE.md` §"Project adapter" → "Static gate".
- Los hallazgos nuevos entran al gate del ledger: los normalizados a BLOCKER/CRITICAL/HIGH se arreglan en el mismo cambio (ver Verification); el resto (MEDIUM/LOW) va a `ISSUES-DEFERRED.md` bajo "static-gate backend (ADR-020, 2026-08-22)", sin bloquear.
- Una falsa alarma verificada (no un hallazgo real) puede suprimirse con `@SuppressFBWarnings`/suppression file puntual, siempre con una razón de una línea junto a la supresión — nunca para ocultar un hallazgo real.
- `spotbugs-maven-plugin` corre sobre bytecode target Java 17 (`maven.compiler.release=17`) aunque el JDK de build sea 25; se fija la versión de `com.github.spotbugs:spotbugs` compatible con ambos (4.8.x) para evitar el mismo tipo de fricción de versión que ya se resolvió en ADR-018 con Jackson 3/Boot 4.

Implementation Plan

- `backend/pom.xml`: perfil `qa` (activation manual, `-Pqa`) atado a `verify`, con `maven-checkstyle-plugin`, `maven-pmd-plugin`, `spotbugs-maven-plugin` (+ `findsecbugs-plugin` como dependencia del plugin).
- `backend/config/checkstyle.xml` (ruleset de alta señal, nuevo).
- `backend/config/pmd-ruleset.xml` (ruleset enfocado bestpractices/errorprone, nuevo).
- `CLAUDE.md` §"Project adapter" → línea "Static gate" actualizada al comando real.
- Hallazgos ≥HIGH del primer run: arreglados en el mismo cambio (con test de regresión si aplica); MEDIUM/LOW documentados en `ISSUES-DEFERRED.md`.

Verification

- [x] `mvn -q test` verde (mismo set de tests, sin regresión de tiempo).
- [x] `mvn -q verify -Pqa` genera `target/checkstyle-result.xml`, `target/pmd.xml` y `target/spotbugsXml.xml`, los tres no vacíos y parseables.
- [x] `qa_ledger.py ingest-gate --repo backend --iteration 6` corre limpio a nivel gate (0 hallazgos HIGH/CRITICAL/BLOCKER sin resolver tras el pase de fixes).
- [x] `qa_ledger.py readiness --record` refleja `backend.static_gate` medido (deja de estar UNMEASURED).

ADR-021 — DTOs de respuesta en los endpoints del gestor (cierre de RAPI-DTO-001 / ENTITY_LEAK)

- **Status:** accepted (operador aprobó la escalación, 2026-08-22)
- **Context:** el gate estático (ADR-020) detectó 15 hallazgos `ENTITY_LEAK` (FindSecBugs/SECURITY, floor a HIGH por el ledger) en 7 controllers del gestor (`ActaGestorController`, `CampaniaGestorController`, `ChanceGestorController`, `ClienteGestorController`, `GanadorGestorController`, `PpremioGestorController`, `SorteoGestorController`): devolvían entidades JPA (`Campania`, `Cliente`, `Sorteo`, `Chance`, `PremioTipo`, `AsignacionPremio`, `ResultadoSorteo`, `ContactoGanador`) directo en la respuesta HTTP. El mismo hallazgo ya estaba documentado por los boot-ui advisors del 2026-08-21 como `RAPI-DTO-001` y quedó escalado en `ISSUES-DEFERRED.md` en vez de arreglarse dentro de ADR-020, porque (a) requería un patrón arquitectónico nuevo (qué campos exponer, cómo mapear, quién lo mantiene — CLAUDE.md regla 4), (b) cambiaba la forma de 15 respuestas que consume hoy un frontend real (`gestor/`) y (c) era deuda ya reconocida y depriorizada (CLAUDE.md regla 6, "legacy baseline"). Este ADR resuelve esa escalación con el operador ya habiendo aprobado el cambio dedicado. La API pública (`participacion` / `padron`, consumida por `landing/`) ya devolvía DTOs desde su diseño original — no tenía este hallazgo y no se toca en este ADR.
- **Alternatives:**
- **`@JsonIgnore` / `@JsonIgnoreProperties` puntual sobre la entidad** — descartada: no resuelve el acoplamiento de fondo (el contrato HTTP sigue atado 1:1 al esquema JPA — cualquier cambio de columna rompe la API), y SpotBugs sigue marcando `ENTITY_LEAK` porque el tipo devuelto sigue siendo `@Entity`.
- **Proyecciones de Spring Data (interfaces)** — descartada para este caso: la mayoría de los endpoints afectados ya tienen la entidad cargada en memoria (viene de un `save` / `findById` previo dentro del mismo método, no de una query nueva); envolverla en una interfaz de proyección hubiese exigido re-consultar o forzar el shape en tiempo de query sin necesidad real.
- **Un DTO genérico (`Map<String, Object>` armado a mano por endpoint)** — descartada: es lo que ya hacían `ActaGestorController#ver`, `ClienteGestorController#detalle` y `ChanceGestorController#voucher`, y es exactamente el patrón que

`ISSUES-DEFERRED.md` (RAPI-ERR-003 y esta misma entrada) señala como frágil: sin tipo, sin autocompletado, sin garantía de que el mapeo cubra todos los campos.

- **`record` de respuesta por entidad, con mapeo explícito campo-a-campo (elegida)** — un `record` por entidad en `api/gestor/dto` (p. ej. `CampaniaResponse`, `SorteoResponse`, `ChanceResponse`, ...), con un factory estático `de(entidad)` que copia cada getter. Compuestos (`ActaDetalleResponse`, `ClienteDetalleResponse`, `VoucherTrazabilidadResponse`) reemplazan los `Map<String,Object>` ad hoc que ya armaban a mano `ActaGestorController` / `ClienteGestorController` / `ChanceGestorController`.
- **Decision:**
- Un `record` de respuesta por entidad tocada, con los MISMOS nombres de componente que los getters del dominio (p. ej. `SorteoResponse.suplentesASortear`, no `suplentesToSortear` ni ningún otro nombre "mejorado"). Esto es deliberado: Jackson usa `SNAKE_CASE` global (`application.yml`) y colapsa mayúsculas consecutivas (`suplentesASortear` → `suplentes_asortear`, no `suplentes_a_sortear` — ya documentado como gotcha JD-G2 en `gestor/src/types/api.ts`); mantener el nombre idéntico garantiza el mismo JSON byte a byte sin tener que razonar de nuevo sobre la transformación de Jackson para cada campo.
- Mapeo vía un factory estático `De.de(entidad)` en el propio record (mismo patrón que ya usaba `SorteoResumenResponse` / `PremioUnidadResponse`, preexistentes), no un mapper externo (MapStruct u otra librería) — cero dependencias nuevas para un mapeo 1:1 sin lógica.
- Ningún campo se agrega ni se quita respecto de lo que la entidad ya serializaba: se verificó campo a campo contra `gestor/src/types/api.ts` (que ya reflejaba el JSON real de las entidades) antes de escribir cada DTO, precisamente para no romper nada consumido por `gestor/` ni por los specs de `e2e/`.
- Los tres endpoints que embebían entidades dentro de un `Map<String,Object>` (`acta` + `resultados`, `cliente` + `chances`, `chance` + `rechazos`) pasan a un DTO compuesto (`ActaDetalleResponse`, `ClienteDetalleResponse`, `VoucherTrazabilidadResponse`) con las mismas dos claves que ya esperaba el frontend (`ActaDetalle`, `ClienteDetalle`, `VoucherTrazabilidad` en `api.ts`) — no estaban en los 15 hallazgos de SpotBugs (el tipo de retorno genérico `Map<String,Object>` no deja que el analisis estatico vea la entidad adentro), pero es el mismo problema de fondo y se corrigen en el mismo cambio por consistencia (misma entidad, mismo controller, cero costo extra de diseño).
- `ClaveFirmaGestorController` no se toca: ya devolvía DTOs (`ClaveFirmaResponse`) desde su diseño original (AC-18) y no tenía hallazgos.
- **Consequences:**
- El contrato HTTP del gestor queda desacoplado del esquema de persistencia: agregar o renombrar una columna JPA ya no cambia el JSON que ve el frontend sin que alguien lo

decida explícitamente en el DTO.

- Los 15 hallazgos `ENTITY_LEAK` (FindSecBugs/SECURITY) desaparecen del gate estático; ninguno se suprimió con `@SuppressFBWarnings` (eran hallazgos reales, no falsos positivos).
- Las entidades de dominio (`Campania`, `Sorteo`, `Chance`, etc.) no tienen asociaciones JPA (`@OneToMany` / `@ManyToOne`) — todas modelan relaciones como IDs manuales — así que no había riesgo real de lazy-loading al serializar; el hallazgo era puramente de acoplamiento de contrato, no de N+1 ni de `LazyInitializationException`.
- Los tres records compuestos (`ActaDetalleResponse`, `ClienteDetalleResponse`, `VoucherTrazabilidadResponse`) hacen una copia defensiva (`List.copyOf`) de sus listas en el constructor compacto — sin esto, SpotBugs marca `EI_EXPOSE_REP` / `EI_EXPOSE_REP2` sobre el propio record nuevo (un `record` no genera copia defensiva sola); se corrigió en el mismo cambio para no dejar hallazgos MEDIUM nuevos.
- Nueva convención en `api/gestor/dto`: los DTOs de respuesta de esta ronda son `record` (no las clases con getters explícitos que ya existían — `SorteoResumenResponse`, `PremioUnidadResponse`, etc.). Se deja así (no se migran los DTOs viejos): ambos estilos serializan igual bajo Jackson/SNAKE_CASE, y migrar los preexistentes es un cambio cosmético sin valor de negocio, fuera del alcance de esta escalación.
- `PanelService` no tenía entidades embebidas en su respuesta (ya devolvía `Map<String, Object>` con primitivos/DTOs propios) — no requirió cambios en este ADR.

Implementation Plan

- `backend/src/main/java/.../api/gestor/dto/`: 13 records nuevos (`CampaniaResponse`, `SorteoResponse`, `ChanceResponse`, `ClienteResponse`, `VoucherRechazadoResponse`, `ResultadoSorteoResponse`, `ContactoGanadorResponse`, `PremioTipoResponse`, `AsignacionPremioResponse`, `ActaResponse`, `ActaDetalleResponse`, `ClienteDetalleResponse`, `VoucherTrazabilidadResponse`).
- 7 controllers actualizados para devolver el DTO en vez de la entidad: `ActaGestorController` (`ver`, `resultados`, `exportJson`), `CampaniaGestorController` (`vigente`, `obtener`, `crear`, `editar`, `subirBases`, `generarSorteos`), `ChanceGestorController` (`chances`, `voucher`), `ClienteGestorController` (`buscar`, `detalle`, `editar`), `GanadorGestorController` (`contacto`, `entrega`, `pasarSuplente`), `PremioGestorController` (`cargarTipo`), `SorteoGestorController` (`editar`, `asignarPremio`).
- `ISSUES-DEFERRED.md`: `RAPI-DTO-001` y la escalación `ENTITY_LEAK` marcadas RESUELTO.

Verification

- [x] `mvn -q test` verde (suite completa, mismos asserts de forma de respuesta — `GestorApiTest` cubre AC-18/19/23/26/27/28/29/30/31).
- [x] `mvn -q verify -Pqa`: 0 hallazgos `ENTITY_LEAK` (antes: 15); 0 hallazgos nuevos `EI_EXPOSE_REP` / `EI_EXPOSE_REP2` en los DTOs compuestos (corregido con `List.copyOf` en el constructor compacto).
- [x] `gestor/`: `npm test` (49 tests) y `npm run build` (`tsc --noEmit && vite build`) en verde sin tocar `gestor/src/types/api.ts` ni ningún componente.
- [x] E2E completo (`e2e/`, todos los proyectos): gestor (51 passed, 3 skipped por diseño), landing (95 passed, 6 skipped — WebKit sin CDP throttling), transversal, rate-limit, modo-prueba, cámara — todos en verde.
- [x] `qa_ledger.py resolve-escalation --repo backend` — escalación cerrada.

ADR-022 — Gestor responsive (sidebar colapsable) + PWA instalable

- **Status:** accepted (operador 2026-08-22; enmienda parcial a ADR-015 sobre el uso del gestor)
- **Context:** ADR-015 definió el gestor como herramienta interna de escritorio (stack WorkDone). En la práctica hay una tarea genuinamente móvil — **registrar la entrega del premio en la sucursal** (verificar DNI del ganador, sucursal, responsable) — que hoy es incómoda: el sidebar mide 240px fijos y nunca se colapsa, comiendo dos tercios de un teléfono de 375px. El operador pidió que el gestor "se vea bien en el celular" sin abrir un segundo proyecto (app nativa).
- **Alternatives:**
- App móvil nativa del gestor completo: otro código, otro stack (rompe ADR-015), fricción de app store, mantenimiento — sobre-ingeniería para tareas mayormente de escritorio.
- Responsive del gestor actual + PWA instalable (elegida): reusa todo el stack existente, un solo código, y cubre al personal de sucursal que administra desde el teléfono.
- **Decision:**
- **Responsive:** el sidebar se colapsa a un drawer (hamburguesa) por debajo de `md` (768px); el contenido pasa a ancho completo; las tablas siguen con scroll horizontal dentro de su contenedor (no scroll del body). Solo Tailwind + los componentes existentes, sin librería nueva (ADR-015).
- **PWA:** `manifest.webmanifest` (nombre "Sistema Premios", scope/start_url `/gestor/`, display standalone, theme = rojo de branding, íconos) + un service worker mínimo que cachea el shell de la app para instalabilidad — **NUNCA** cachea respuestas de `/api` (el gestor es una herramienta viva; datos siempre frescos). Servido por el jar bajo `/gestor/`. Compatible con el CSP estricto (SW y manifest son same-origin `self`).
- **Consequences:** El gestor entero queda usable en teléfono/tablet para la entrega en mostrador, sin segundo codebase. La PWA da ícono en el teléfono. El branding del cliente (color/logo, ADR-branding) alimenta el theme del manifest. No se toca la landing.

Implementation Plan

- `gestor/src/components/layout/AppShell.tsx`: sidebar → `hidden md:flex` + drawer overlay con botón hamburguesa en un header móvil; cierra al navegar; foco/aria correctos.
- `gestor/public/manifest.webmanifest` + íconos (192/512, maskable); `<link rel="manifest">` y meta theme-color en index.html; registro del SW en `main.tsx` (solo en prod build).
- `gestor/public/sw.js` (o vite-plugin-pwa si no agrega peso/stack — evaluar; preferir SW a mano si el plugin trae mucho): precache del shell, `NetworkOnly` para `/api` y `/bases`.
- Backend: servir `/gestor/manifest.webmanifest` y `/gestor/sw.js` con content-type correcto y `Service-Worker-Allowed: /gestor/`; el SW debe poder tener scope `/gestor/`.

Verification

- [ ] AC-36 — Sidebar colapsa a drawer < 768px; todos los ítems de navegación alcanzables por el menú hamburguesa; se cierra al elegir uno.
- [ ] AC-37 — Sin scroll horizontal del body en el gestor a 375px (las tablas scrollean dentro de su contenedor).
- [ ] AC-38 — PWA instalable: manifest válido (nombre, íconos 192/512, start_url `/gestor/`, display standalone) y service worker registrado con scope `/gestor/`.
- [ ] AC-39 — El service worker NUNCA sirve respuestas cacheadas de `/api/**` (siempre red).

ADR-023 — Agregar un sorteo suelto (ad-hoc) a una campaña

- **Status:** accepted (operador 2026-08-22)
- **Context:** SPEC §3 define un calendario fijo de 6 sorteos (5 semanales + cierre), generado por `SorteoGeneracionService.generar` y editable fecha a fecha vía `PUT /api/gestor/sorteos/{id}`. El operador pidió poder sumar un sorteo **extra** (por ejemplo, una acción puntual con un lote de premios adicional) sin tener que borrar y regenerar todo el calendario.
- **Restricción dura (no se negocia — ADR-001):** ADR-001 congela la ventana de un sorteo en el momento en que se ejecuta (`filtros` del acta es un snapshot inmutable). El bucketing de elegibles depende de que la secuencia `orden` de los sorteos de una campaña coincida con el orden cronológico real de `fecha_hora_publicada` (`SorteoService.calcularVentana` usa el sorteo con `orden` inmediato anterior para derivar el `desde` de la ventana). Insertar un sorteo nuevo **antes o entre** sorteos ya ejecutados correría el riesgo de invalidar a qué sorteo pertenece cada chance ya sorteada — inaceptable (CONSTITUTION: un acta ejecutada es inmutable).
- Por eso: la `fecha_hora_publicada` del sorteo nuevo debe ser **estrictamente posterior** a la del **último sorteo ya ejecutado** de la campaña (si hay alguno). No hay excepción.
- **Alternatives:**
- Exigir borrar y regenerar todo el calendario para agregar un sorteo — descartada: obliga a re-planificar semanas que ya podrían tener premios asignados, y `SorteoGeneracionService.generar` ya rechaza regenerar si hay algo ejecutado (JD-A4).
- Permitir insertar el sorteo en cualquier posición cronológica, incluso antes de un ejecutado, recalculando `filtros` de actas ya persistidas — descartada: viola directamente ADR-001/CONSTITUTION (un acta ejecutada nunca se re-escribe).
- **Elegida:** permitir insertar el sorteo en cualquier posición cronológica **posterior al último ejecutado**, y renumerar `orden` solo de los sorteos NO ejecutados para que la secuencia completa (ejecutados + no ejecutados) siga siendo estrictamente cronológica.
- **Decision:**
- Nuevo endpoint `POST /api/gestor/campanias/{id}/sorteos` (distinto de `.../sorteos/generar`, que crea la tanda completa). Body: `{nombre, fecha_hora_publicada,`

`suplentes_a_sortear}` (este último con default 10, SPEC §3). `tipo` siempre `semanal` — no se crea un tipo de sorteo nuevo; el único `cierre` sigue siendo el que genera la regla de programación.

- Validaciones (400/409 vía `GestorExceptionHandler`, mismo patrón que el resto del gestor):
  a. La campaña debe existir (400 si no) y no estar `cerrada` (409 — no tiene sentido agregar sorteos a una campaña que ya terminó).
  b. `fecha_hora_publicada` dentro de `[vigencia_desde, vigencia_hasta]` de la campaña (400 si no — simplifica sobre "hasta la fecha del cierre": la vigencia ya es el límite superior natural de la campaña).
  c. **ADR-001 (crítico):** si existe algún sorteo `ejecutado`, la fecha debe ser estrictamente posterior a la `fecha_hora_publicada` del último ejecutado (409 si no, mensaje explícito).
- Renumeración: los sorteos `ejecutado` **nunca** cambian de `orden` (su acta ya congeló esa ventana). Los sorteos `pendiente` (los existentes no ejecutados + el nuevo) se ordenan cronológicamente por `fecha_hora_publicada` y se renumeran a partir de `max(orden de los ejecutados) + 1` — como el nuevo sorteo es posterior a todos los ejecutados, naturalmente cae después de ellos en esa renumeración.
- **Consequences:** El operador puede sumar un sorteo puntual sin tocar el historial ya ejecutado. `SorteoService.calcularVentana` sigue funcionando sin cambios: al mantenerse el invariante "orden == orden cronológico", las ventanas de los sorteos no ejecutados quedan contiguas y sin solape automáticamente. La restricción dura empeora la flexibilidad (no se puede insertar "en el medio" del pasado), pero es exactamente lo que ADR-001 exige.

Implementation Plan

- `backend/.../api/gestor/dto/AgregarSorteoRequest.java`: DTO de request (`nombre`, `fechaHoraPublicada`, `suplentesASortear` con default 10).
- `backend/.../api/gestor/SorteoGeneracionService.agregarSorteo(...)`: valida campaña/vigencia/ ADR-001, inserta y renumera.
- `backend/.../api/gestor/CampaniaGestorController`: `POST /{id}/sorteos` → delega en el servicio, devuelve `SorteoResponse`.
- `gestor/src/pages/CampaniaPage.tsx`: botón "Agregar sorteo" + form (nombre, fecha/hora, suplentes) junto a la tabla de sorteos; invalida `sorteos-campania` + `panel` al guardar.
- `gestor/src/api/gestor.ts` + `gestor/src/types/api.ts`: `agregarSorteo` + tipos.

Verification

- [ ] Agregar sin sorteos ejecutados → 201, queda en su posición cronológica correcta.
- [ ] Agregar con fecha posterior al último ejecutado → 201, ejecutados conservan su `orden`.
- [ ] Agregar con fecha ≤ fecha del último ejecutado → 409 con mensaje claro (ADR-001).
- [ ] Agregar con fecha fuera de la vigencia de la campaña → 400.
- [ ] Agregar a una campaña `cerrada` → 409.
- [ ] Test de integración: el sorteo agregado, al ejecutarse, toma únicamente las chances de su propia ventana (`calcularVentana` sigue dando ventanas contiguas y sin solape tras la renumeración).
- [ ] Gestor: el form arma el body correcto; un 409 se muestra como mensaje claro (no un toast genérico ilegible).

ADR-024 — Concurrency hardening: bloqueo optimista + serialización de ejecución por campaña

- **Status:** accepted (2026-08-22; reabierto por judgment-day externo — PF-001/002/003)
- **Context:** Un análisis adversarial de 4 jueces ciegos encontró carreras de concurrencia que las pasadas de QA single-threaded no habían detectado. Confirmadas en código:
- **CRIT-1**: `PremioService.registrarEntrega/pasarASuplente/devolverAlStock/marcarNoOtorgado` leen la unidad con `findById` (sin lock) y ninguna entidad mutable tiene `@Version`. Dos operadores sobre la misma unidad → lost update (unidad "disponible" tras entregarla, o asociada a dos resultados). Los locks previos (ADR JD-A3) cubrieron `asignar` / `ejecutar` / `desasignar` pero NO la fase de entrega.
- **CRIT-3**: `SorteoService.ejecutar` toma lock pesimista del **sorteo** (`findByIdForUpdate(sorteoId)`), no de la **campaña**. Dos sorteos distintos de la misma campaña pueden ejecutarse a la vez; la consulta de adjudicados (`AdjudicacionCampaniaService`) lee estado commiteado, así que ninguno ve al ganador del otro → un DNI gana en ambos, violando la exclusión (ADR-003) y las bases legales.
- **Alternatives:**
- Solo pesimista puntual (lo que había): insuficiente — no cubre la fase de entrega ni serializa ejecuciones entre sorteos de una campaña.
- `@Version` (optimista) en las entidades mutables + serialización de la ejecución a nivel **campaña** (elegida): defensa en profundidad. El optimista atrapa cualquier lost update de la fase de entrega con `OptimisticLockException` ; la serialización por campaña garantiza la exclusión bajo ejecución concurrente.
- **Decision:**
- `@Version Long version` en `PremioUnidad` , `AsignacionPremio` , `ResultadoSorteo` (migración `V9` agrega la columna `version` con default 0). Un `OptimisticLockException` en la fase de entrega se traduce a **409** ("otra operación modificó el premio; reintentá") vía `GestorExceptionHandler` — nunca un lost update silencioso.
- `SorteoService.ejecutar` toma primero un **lock pesimista de la fila** `campania` (`campaniaRepository.findByIdForUpdate`) además del lock del sorteo. Así dos ejecuciones

de sorteos distintos de la MISMA campaña se serializan: la segunda espera a que la primera commitee sus ganadores, y la exclusión de adjudicados (ADR-003) los ve.

- Validación `vigenciaDesde < vigenciaHasta` en `CampaniaRequest` (400) y en DB (CHECK, `V9`).
- `resolver` sin campaña vigente → respuesta limpia (caso C / genérica), nunca 500 colgante.
- **Consequences:** Escrituras de la fase de entrega se vuelven seguras ante concurrencia sin bloquear a los lectores; las ejecuciones de una campaña se serializan (impacto nulo en la operación real — los sorteos se ejecutan manualmente uno por vez, ADR-009). El costo es una columna `version` por entidad y un lock de campaña por ejecución.

Verification

- [x] AC-40 — Entrega y devolución concurrentes sobre la MISMA unidad: exactamente una gana; la otra recibe 409 por conflicto optimista; el estado final es consistente (nunca Disponible tras Otorgado, ni dos resultados con la misma unidad).
- [x] AC-41 — Dos sorteos distintos de la misma campaña ejecutándose concurrentemente se serializan por el lock de campaña; ningún DNI adjudicado en el primero aparece como ganador en el segundo.
- [x] AC-42 — Campaña con `vigencia_desde >= vigencia_hasta` → 400 (validación de request) y la DB rechaza el insert directo (CHECK).
- [x] AC-43 — `resolver` / `padron alta` sin campaña vigente → respuesta definida (caso C o mensaje genérico), nunca 500 que deje la landing colgada.

ADR-025 — Acta con premio original congelado + separación del estado de entrega; plazo visible; paginación de clientes

- **Status:** accepted (2026-08-23; reabierto por judgment-day — CRIT-2, #6, #7)
- **Context (CRIT-2):** El acta garantiza reproducibilidad del sorteo (seed+lista+algo → mismos ganadores), pero su export incluye `premio_unidad_id` y `estado_entrega`, que MUTAN con la operación (entrega, devolución, pase a suplente). Dos exports del mismo acta muestran premios/estados distintos, confundiendo el artefacto legal (lo sorteado) con el estado operativo (lo entregado). El sorteo en sí es inmutable; la entrega evoluciona.
- **Decision (CRIT-2):**
- `resultado_sorteo` gana `premio_unidad_id_original` (BIGINT nullable), seteado UNA VEZ al ejecutar (`SorteoService.persistirResultados`) y NUNCA modificado por las operaciones de entrega. Migración `V10` : agrega la columna y la backfillea con el `premio_unidad_id` actual para actas ya existentes.
- El export del acta se parte en dos secciones EXPLÍCITAS:
  a. **Acta del sorteo (INMUTABLE)** — fecha_ejecucion, seed, algo_version, hash_lista, cantidad_participantes, y por cada posición: orden, tipo, chance/cliente, y el **premio sorteado originalmente** (`premio_unidad_id_original`). Reproducible: matchea `hash_lista` y `reproducir()` .
  b. **Estado de entrega (snapshot, con `exportado_en`)** — por cada resultado: premio actual, estado_entrega, datos de entrega. Claramente rotulado como estado al momento del export.
- Dos exports en momentos distintos: la sección 1 es idéntica; la 2 puede diferir.
- **Decision (#6 — plazo de retiro):** NO se automatiza (un scheduler violaría ADR-009 y podría despojar a un ganador legítimo). Se VISIBILIZA: la vista de ganadores expone `plazo_vencido` (días desde el primer contacto vs `campania.plazo_retiro_dias`); `registrarEntrega` sobre un resultado con plazo vencido exige un flag de confirmación explícito (no bloquea, advierte).
- **Decision (#7 — clientes):** la búsqueda del padrón exige un término mínimo (query vacío no materializa todo el padrón) y se pagina server-side (`Pageable` , tope por página). Evita cargar decenas de miles de filas al abrir la pantalla en una campaña con volumen.

- **Consequences:** El acta protege su valor legal (lo sorteado nunca cambia) sin perder la trazabilidad operativa (estado de entrega como snapshot). El plazo queda como decisión informada del operador. La pantalla de clientes escala con el padrón.

Verification

- [x] AC-44 — `premio_unidad_id_original` congelado al ejecutar; el export separa "acta (inmutable)" de "estado de entrega (snapshot)". Tras una devolución, dos exports: la sección de acta es idéntica (hash estable, reproduce() matchea), la de entrega difiere.
- [x] AC-45 — Ganadores expone `plazo_vencido` (días desde primer contacto vs plazo); `registrarEntrega` con plazo vencido sin confirmación → advertencia/409; con confirmación explícita → procede. Nunca auto-transiciona a NO_OTORGADO.
- [x] AC-46 — Búsqueda de clientes: query vacío no carga el padrón completo (400/paginado); respuesta paginada (page/size/total). Test con volumen confirma que no materializa todo.

ADR-026 — Escaneo de DNI con decodificador PDF417 propio (independiente del navegador)

- **Status:** accepted (2026-08-23; enmienda a la estrategia de AC-22, reabierto por prueba física en Samsung S24 / Chrome)
- **Context:** El escaneo del DNI (AC-22) se apoyaba en `BarcodeDetector` (Shape Detection API de Chromium). Prueba en dispositivo real (Samsung S24, Chrome de Android — el navegador más común del público objetivo): el botón "Escanear DNI" NO aparece porque `getSupportedFormats()` no incluye `pdf417`. Chrome de Android soporta QR pero no PDF417 sin el módulo de códigos de Google Play Services; Samsung Internet y Firefox no traen `BarcodeDetector`. Conclusión: la estrategia nativa deja el escaneo **indisponible para la mayoría del público de salón**. La carga manual (fallback de AC-22) funciona, pero la feature de conveniencia prácticamente no existe.
- **Alternatives:**
- Aceptar carga manual como principal (statu quo): cero costo, pero el "escaneá tu DNI" casi nunca aparece.
- Decodificador PDF417 en JS/WASM (elegido): funciona en TODOS los navegadores con cámara, independiente de `BarcodeDetector`. Se carga **lazy** (dynamic `import()` al tocar "Escanear"), así NO afecta el presupuesto de 3G de la primera pantalla (AC-24). Como el decodificado es NUESTRO, controlamos el charset → la Ñ/acentos se resuelven de raíz (cierra QA-L5, ya no es especulación).
- **Decision:**
- Integrar un decodificador PDF417 JS/WASM (evaluar `zxing-wasm` vs `@zxing/library`; elegir el que decodifique de forma confiable el PDF417 del DNI argentino y permita manejar el encoding Latin-1). Carga diferida en `lib/dniScanner.ts` (ya es un módulo dinámico).
- `BarcodeDetector` queda como camino rápido OPCIONAL solo donde soporte `pdf417`; el decodificador propio es el motor por defecto. El botón "Escanear DNI" se muestra en cualquier navegador con cámara (getUserMedia), no solo donde hay `BarcodeDetector`.
- Charset: el payload del DNI argentino se decodifica respetando su codificación real (Latin-1/ISO-8859-1) para que apellidos con Ñ/acentos lleguen correctos al padrón.
- Presupuesto: el decodificador NO entra al bundle inicial (lazy). El presupuesto de 60 KB gzip de la primera pantalla se mantiene (AC-24).

- **Consequences:** El escaneo funciona en todo el parque de teléfonos, no solo en un subconjunto de Chrome. Dependencia nueva de la landing, pero fuera del camino crítico (lazy). ADR-015 ("landing pelada") se enmienda: la restricción de peso aplica al bundle INICIAL; un módulo de escaneo diferido está permitido porque no lo penaliza.

Verification

- [ ] AC-48 — El botón "Escanear DNI" aparece en cualquier navegador con cámara (no depende de `BarcodeDetector`); el decodificador PDF417 propio lee un DNI de prueba y autocompleta.
- [ ] AC-49 — Charset: un payload PDF417 con apellido "PEÑA"/"MARÍA JOSÉ" decodifica con la Ñ y los acentos correctos (unit test sobre el payload conocido — cierra QA-L5).
- [ ] Bundle inicial de la landing sigue ≤ 60 KB gzip (el decodificador es lazy, AC-24).

ADR-027 — Reconocimiento del dispositivo por secreto opaco (memoria del DNI sin guardar el DNI)

- **Status:** accepted (2026-08-23; feature nueva pedida por el operador — acelerar la 2ª participación sin re-pedir el DNI)
- **Context:** En la 2ª participación desde el mismo teléfono, hoy el cliente vuelve a ingresar (o re-escanear) el DNI aunque ya esté en el padrón. Se quiere "recordar" a la persona para que el flujo sea un saludo + confirmación de un toque. El camino ingenuo — guardar el DNI en `localStorage` — **no es aceptable**: deja el DNI en claro en el navegador (CWE-312). Un hash del DNI en el cliente **tampoco** protege: un DNI son ~90 millones de valores, así que el hash se revierte por fuerza bruta con la propia app (falsa privacidad). El teléfono además puede ser **compartido** (la caja del súper, un familiar), así que reconocer no puede implicar participar.
- **Alternatives:**
- DNI en claro en `localStorage`: simple, pero expone el DNI en el navegador. **Rechazado** (viola CONSTITUTION — CWE-312).
- Hash del DNI en el cliente: falsa privacidad (espacio de ~90M → brute-forceable). **Rechazado.**
- **Secreto opaco emitido por el backend (elegido):** al participar, el backend genera un secreto aleatorio (≥128 bits, CSPRNG); el navegador guarda solo `{ secreto, nombre }`. El DNI **nunca** toca el navegador. El secreto no significa nada sin el servidor (es un identificador opaco, no deriva del DNI). En DB se persiste **solo el hash** del secreto (mismo patrón que `hash_token` del voucher, AC-03), asociado al cliente.
- **Decision:**
- **Naming (evita colisión):** en TODO el sistema "token" ya es el voucher QR (`hash_token`, `/p/{token}`). Esta feature se llama **"reconocimiento"**, nunca "token" a secas. Tabla `reconocimiento_dispositivo`, secreto en cliente = `secretoReconocimiento`, columna en DB = `hash_reconocimiento`, endpoint = `POST /api/participacion/reconocer`.
- **Emisión:** al consumir una chance (alta caso D o confirmar caso E), el backend emite un secreto opaco nuevo, guarda su hash en `reconocimiento_dispositivo (cliente_id, hash_reconocimiento UNIQUE, creado_en, ultimo_uso_en)`, y lo devuelve **una sola vez** en la respuesta. El navegador lo guarda en `localStorage` junto al primer nombre. **Excepción (no rota):** el path `confirmar-reconocido` NO emite un secreto nuevo — el dispositivo ya

presentó uno válido y lo reusa, garantizando "1 por dispositivo". Solo los caminos por DNI (alta caso D, confirmar caso E por DNI = primer contacto del dispositivo) emiten. El frontend confía en el secreto que emite el backend: lo guarda cuando viene, y en su ausencia (`confirmar-reconocido`) conserva el que ya tiene.

- **Cardinalidad: N por cliente, 1 por dispositivo** (tabla aparte, no columna en `cliente`). Un cliente puede ser reconocido desde el celu de casa Y el del trabajo sin pisar secretos.
- **Vida: permanente, sin expiración**, cruza campañas (decisión explícita del operador — ver Consequences). El padrón de clientes es permanente, así que reconocer entre campañas es coherente con el modelo de datos.
- **Lectura:** `POST /api/participacion/reconocer` recibe el secreto, lo hashea, busca la fila y devuelve **solo** `{ nombre, dni_mascarado }` (reusa `ClienteResumenDto`). Nunca DNI completo, nunca email/celular. Secreto no encontrado / cliente borrado → respuesta que instruye alta por DNI, **nunca** 500 ni error colgante (mismo criterio que AC-43).
- **Reconocer ≠ participar:** aun con secreto válido, participar exige **confirmación explícita del usuario + un voucher vigente**. El saludo "Hola {nombre} — confirmá tu DNI ***5555" nunca dispara la chance solo.*** ***"No soy yo"** borra el secreto del `localStorage` y cae al alta por DNI (protege el teléfono compartido).
- **Confirmar-por-reconocimiento (el one-tap real):** `POST /api/participacion/confirmar-reconocido` recibe `{ token (voucher), secreto }` — **sin DNI del cliente**. Hashea el secreto, lo mapea a cliente del lado servidor, y consume la chance para ese cliente por el MISMO camino que `confirmar` caso E (voucher vigente, consumo atómico, tope diario, exclusión). El DNI **nunca viaja** del cliente: se resuelve server-side desde el secreto — refuerza la privacidad, no la debilita. Secreto no encontrado → respuesta que instruye alta por DNI, nunca 500. La landing lo invoca **solo** con el tap explícito de "Confirmar y participar" (cumple AC-53: no participa solo). Sin este endpoint el saludo sería cosmético (reconoce pero igual pide el DNI) y la feature no entregaría su valor — por eso es parte de PR1, no diferible.
- **Seguridad del endpoint:** cae bajo `/api/**` → ya cubierto por `RateLimitFilter` (AC-19). El secreto es ≥128-bit CSPRNG (no adivinable ni enumerable).
- **Configurable por cliente (`sp.reconocimiento.habilitado`, default `true`):** toda la feature vive detrás de un flag en el YAML **externo** (ADR-029). En `false`: el backend **no** emite secreto (`secretoReconocimiento` queda null en alta/confirmar), y `/reconocer` + `/confirmar-reconocido` responden como "secreto no encontrado" (instruyen alta por DNI, nunca error); la landing no guarda secreto ni intenta reconocer — **pide siempre el DNI** (flujo clásico). Le da al cliente una perilla de privacidad. Asume **single-tenant** (una instancia por supermercado, igual que el branding); si fuera multi-tenant, el flag iría por cliente en DB, no en YAML. **Un solo booleano**, sin sub-opciones (YAGNI).
- **Consequences:**

- El DNI nunca queda en el navegador; el reconocimiento es privado por construcción.
- La 2ª participación es un saludo + un toque, sin re-tipear el DNI.
- Consistente con el patrón existente (se persiste el hash, no el secreto).
- El secreto permanente es un **credencial bearer que no caduca**: si el teléfono se pierde o se vende, quien lo tenga ve "{nombre} ***{4 dígitos}" y puede participar como esa persona de un toque.*** ***Blast radius acotado y aceptado:** solo expone primer nombre + DNI enmascarado (nunca el DNI completo ni contacto), siempre exige confirmación + voucher vigente, y "No soy yo" lo borra. La conveniencia de no re-pedir el DNI se prioriza sobre ese riesgo acotado.
- Schema nuevo: tabla `reconocimiento_dispositivo` (migración V11).

Implementation Plan

- **Affected paths:**
- `backend/.../db/migration/V11__reconocimiento_y_datos_cliente.sql` — tabla `reconocimiento_dispositivo` (FK `cliente_id`, `hash_reconocimiento` UNIQUE, `creado_en`, `ultimo_uso_en`).
- `backend/.../dominio/ReconocimientoDispositivo.java` (entity) + repositorio.
- `backend/.../participacion/ParticipacionService.java` — emitir secreto al consumir chance (alta/confirmar); método `reconocer(secreto, ip)`.
- `backend/.../api/ParticipacionController.java` — `POST /api/participacion/reconocer`.
- `backend/.../api/dto/ParticipacionResponse.java` — agregar `secretoReconocimiento?` (solo en respuesta de alta/confirmar, `NON_NULL`); reusar `ClienteResumenDto` para `reconocer`.
- `landing/src/api/client.ts` + `types/api.ts` — `reconocerDispositivo(secreto)`; persistencia en `localStorage` (nuevo `lib/reconocimiento.ts`: get/set/clear `{ secreto, nombre }`).
- `landing/src/App.tsx` — al abrir `/p/{token}`, si hay secreto guardado → pantalla de saludo (reconocido) antes de pedir DNI; "No soy yo" limpia y cae al flujo por DNI.
- **Patterns:** hash del secreto = mismo enfoque que `hash_token` (AC-03). Endpoint bajo `/api/**` para heredar rate-limit. Respuesta enmascarada = `DniUtil.mascarar` (ya existe).
- **Tests:** golden del enmascarado; test de que `reconocer` con secreto inexistente cae a alta; test de que el DNI no se escribe en `localStorage`; test de que reconocer no consume chance.

Verification

- [x] AC-50 — Al consumir chance (alta/confirmar), el backend emite un secreto opaco ≥128-bit CSPRNG; en DB se persiste solo `hash_reconocimiento` (nunca el secreto), asociado al cliente; el secreto vuelve una sola vez en la respuesta. Evidencia: `ReconocimientoEmisionTest` — Tests run: 1, Failures: 0, Errors: 0 (SQL Server real, V11 aplicada).
- [x] AC-51 — `POST /api/participacion/reconocer` con secreto válido devuelve solo `{ nombre, dni_mascarado }` (nunca DNI completo, email ni celular); secreto no matcheado → respuesta que instruye alta por DNI, nunca 500. Evidencia: `ReconocimientoApiTest` — Tests run: 3, Failures: 0, Errors: 0.
- [x] AC-52 — El DNI nunca se persiste en el navegador: `localStorage` guarda solo `{ secretoReconocimiento, nombre }` (test sobre el flujo de participación). Evidencia: `tests/reconocimiento.test.ts` (5 tests) + `tests/reconocimiento-flujo.test.tsx` — vitest run, 0 fallos.
- [x] AC-53 — Reconocer nunca participa solo: con secreto válido, participar exige confirmación explícita + voucher vigente; "No soy yo" borra el secreto del navegador y cae al alta por DNI. Evidencia: `ReconocimientoApiTest.ac53_reconocerNoConsumeChance` + `reconocimiento-flujo.test.tsx` (backend real + vitest, 0 fallos).
- [x] AC-61 — Confirmar-por-reconocimiento: `POST /api/participacion/confirmar-reconocido` con `{token, secreto}` válidos mapea el secreto→cliente server-side y consume la chance para ese cliente (caso E) SIN que el DNI viaje del cliente (el request no lleva DNI); exige voucher vigente; secreto no encontrado → respuesta que instruye alta por DNI, nunca 500; la landing lo invoca solo con el tap explícito de "Confirmar y participar". Evidencia: `ConfirmarReconocidoApiTest` — Tests run: 3, Failures: 0, Errors: 0 (SQL Server real) + `reconocimiento-flujo.test.tsx` AC-61 (vitest, 0 fallos). Suite completa del backend: Tests run: 217, Failures: 0, Errors: 0 — BUILD SUCCESS.
- [x] AC-62 — Flag `sp.reconocimiento.habilitado`: en `false`, alta/confirmar NO devuelven `secretoReconocimiento` (null), `/reconocer` y `/confirmar-reconocido` responden instruyendo alta por DNI (nunca 500), y la landing pide siempre el DNI (no guarda ni intenta reconocer); en `true`, la feature opera normal (AC-50..53, AC-61). Evidencia: `ReconocimientoFlagDeshabilitadoTest` (4 tests) + `CampaniaPublicaFlagDeshabilitadoTest` (1 test) — Tests run: 5, Failures: 0, Errors: 0 (SQL Server real); `reconocimiento-flujo.test.tsx` AC-62 (vitest, 0 fallos). Suite completa del backend: Tests run: 222, Failures: 0, Errors: 0 — BUILD SUCCESS. Suite completa de landing: 20 archivos, 96 tests, 0 fallos. **Hallazgo durante AC-62 (fijado en el mismo commit):** el guard de PERF del estado embebido (`SpaForwardController`) cortaba el chequeo de reconocer() ANTES de correrlo — como esa ruta se ejecuta en TODA visita real a `/p/{token}`, el reconocimiento nunca se disparaba en produccion. Cubierto con un test de regresion (AC-51, camino embebido) antes del fix.

ADR-028 — Completar y corregir datos del cliente desde la landing

- **Status:** accepted (2026-08-23; feature nueva pedida por el operador — el cliente que vuelve debe poder sumar/corregir sus datos)
- **Context:** Hoy el caso E (cliente ya registrado que vuelve) es una confirmación de un toque (`Pantalla2Confirmar.tsx`), sin editar nada. Dos huecos reales: (1) si el cliente no cargó email, no puede sumarlo nunca desde la landing; (2) si tipeó **mal el celular**, no puede corregirlo — y el celular es **el canal para contactar al ganador**: un celular mal cargado = premio no entregable. Corregir datos es sensible: el teléfono puede ser compartido, así que no cualquiera que llegue con un voucher debe ver/editar los datos de otro. **Sub-caso crítico (origen de los datos):** el alta puede ser por **escaneo** del DNI (los datos vienen del documento) o **manual** (la persona los tipea — la carga manual siempre está disponible, AC-22). Si el alta fue manual, `nombre/apellido/sexo/fecha` son tan falibles como el email: un "Gonzáles" por "González" tipeado a mano no puede quedar trabado para siempre. Hoy el backend **no sabe** si el alta fue por escaneo o a mano — recibe el mismo payload en ambos casos.
- **Alternatives:**
- No tocar nada (statu quo): el cliente nunca completa/corrige desde la landing; todo pasa por el operador en el gestor. Simple, pero un celular mal cargado queda mal para siempre salvo que el operador lo note (y no hay momento en que el cliente pida corregirlo).
- Mostrar y editar todo siempre: flexible, pero **expone** datos en un celular potencialmente compartido y permite pisar el contacto de otra persona. **Rechazado.**
- **Editar gated por presentación del DNI, y con la identidad gobernada por el origen de los datos (elegido):** el DNI es la identidad/auth del sistema. Quien presentó el DNI (escaneo o carga manual) puede ver/corregir **lo suyo**; el reconocimiento por dispositivo (ADR-027) **no** alcanza para editar. Los campos de identidad son de solo lectura **solo cuando vinieron del escaneo** (documento = verdad); si el alta fue manual, son corregibles.
- **Decision:**
- **Agregar opcional vacío:** el confirmar de caso E sigue siendo **un toque**. Debajo, un link discreto "**Agregar {opcional}**" aparece **solo si el opcional está vacío** (hoy el único opcional es `email`; el celular es obligatorio en el alta, AC-21). Campo inline, opcional, se puede confirmar sin llenarlo. El label dice "**Agregar**" (no "Modificar"): solo suma lo que falta, sin exponer lo existente.

- **Corregir dato existente:** exige **haber presentado el DNI en la sesión**.
  - *Path DNI* (caso E del flujo actual — llegó ingresando/escaneando el DNI): aparece "Revisá tus datos" con los campos editables, enmascarados hasta tocar "editar".
  - *Path reconocimiento* (ADR-027 — reconocido sin pedir DNI): **solo** confirmar de un toque; para corregir, "Revisar mis datos" **pide el DNI primero**. En un celular compartido, nadie edita los datos de otro sin tener su DNI en la mano.
- **Origen de los datos (nuevo `datos_origen` en `cliente`):** la landing YA sabe si escaneó (estado `datosEscaneados` en `Pantalla1bAlta`); manda un flag en el alta y el backend persiste `datos_origen ∈ { escaneo, manual }`. Gobierna la editabilidad de la **identidad**:

Campo	`datos_origen = escaneo`	`datos_origen = manual`
`nombre/apellido/sexo/fecha_nacimiento`	**solo lectura** (documento = verdad; para cambiar, re-escanear)	**editable** con DNI presentado
`email/celular`	editable con DNI presentado	editable con DNI presentado

- **Re-escaneo corrige y verifica:** un cliente `manual` que escanea el DNI corrige sus datos de identidad Y sube su `datos_origen` a `escaneo` (queda verificado por documento → identidad pasa a solo lectura). No se obliga (por eso existe la carga manual), pero es el mejor camino cuando hay cámara.
- **Transacción:** un solo submit. "**Confirmar y participar**" guarda los datos corregidos **y** suma la chance en una transacción. Una edición NO permitida (identidad escaneada) se rechaza **antes** de consumir la chance (no gasta el voucher). Abandonar antes de confirmar no persiste cambios. **Límite conocido (SCR-001):** el consumo de chance vive en su propia transacción `REQUIRES_NEW` (garantía de AC-05, "un voucher = una chance", intocable). Por eso, en la **rara colisión concurrente** sobre el mismo cliente (landing vs gestor, o dos pestañas), la participación **se registra** (chance sumada) pero la corrección se **rechaza con 409** y debe reintentarse — nunca hay doble chance ni corrupción silenciosa. Cerrar ese hueco exigiría tocar el aislamiento de `consumirChance` (riesgo a AC-05); se prioriza el invariante sobre el edge.
- **Auditoría:** todo cambio de dato del cliente desde la landing genera un **evento interno** en tabla nueva `evento_dato_cliente (cliente_id, campo, valor_anterior, valor_nuevo, cambiado_en, hash_token, ip)` — cubre contacto E identidad manual. Guarda los **valores reales** (no enmascarados): es un registro **interno** para investigación antifraude, nunca expuesto por la

API pública (mismo criterio que `voucher_rechazado` y AC-33). El display al usuario va enmascarado.

- **Concurrencia:** el path de edición escribe sobre `Cliente`, que **hoy no tiene** `@Version`. Se agrega la columna `version` a `cliente` y `@Version` a la entidad (reusa el patrón de ADR-024). Edición de landing vs gestor sobre el mismo cliente → una gana, la otra 409; la landing re-lee y re-muestra. Sin lost update sobre el celular del ganador.
- **Validación:** al corregir se reaplica la validación del alta — celular 10 dígitos normalizado, email con formato, sexo ∈ {M,F,X}, fecha DD/MM/AAAA (AC-21).
- **Consequences:**
- El ciclo se cierra: el cliente puede **agregar** lo que falta y **corregir** lo que tipeó mal (contacto siempre; identidad si el alta fue manual), siempre presentando el DNI.
- El principio "documento = verdad" se aplica **donde de verdad corresponde** (cuando escaneó), sin trabar al cliente que se registró a mano.
- El teléfono compartido queda protegido: editar exige el DNI, no basta el reconocimiento.
- Trazabilidad completa de cambios de datos (antifraude).
- Schema nuevo: `evento_dato_cliente` + `cliente.version` + `cliente.datos_origen` (migración V12).
- **Atomicidad acotada (SCR-001, aceptado):** datos+chance atómicos en el caso común; bajo colisión concurrente, la chance cuenta y la corrección se rechaza 409 (reintentable). Daño real mínimo y recuperable (el cliente conserva su chance; el operador o el próximo voucher corrigen); AC-05 (no doble chance) queda intacto. Decisión del operador: enmendar AC-57, no destabilizar el núcleo.
- La landing debe transmitir el flag de escaneo en el alta (dato que hoy se pierde).
- `Pantalla2Confirmar.tsx` gana estados de UI (agregar / revisar / editar) — acotados, sin salir del presupuesto de la landing (AC-24); la edición no agrega peso crítico.

Implementation Plan

- **Affected paths:**
- `backend/.../db/migration/V11__reconocimiento_y_datos_cliente.sql` — `ALTER TABLE cliente ADD version` (NOT NULL default 0) + `ALTER TABLE cliente ADD datos_origen` (CHECK IN ('escaneo','manual'), NOT NULL, default por back-fill según lo que se sepa del padrón existente — o 'manual' conservador) + tabla `evento_dato_cliente`.

- `backend/.../dominio/Cliente.java` — `@Version Long version` + `datos_origen` (enum STRING).
- `backend/.../dominio/EventoDatoCliente.java` (entity) + repositorio.
- `backend/.../participacion/ParticipacionService.java` — el alta persiste `datos_origen` desde el flag del request; método `confirmarConDatos(token, dni, cambios, ip)` que, en una transacción, valida el DNI presentado, aplica solo los cambios permitidos según `datos_origen`, escribe `evento_dato_cliente` y consume la chance (caso E). Rechaza cambios de identidad si el origen es `escaneo`. El re-escaneo con datos nuevos actualiza identidad y sube `datos_origen` a `escaneo`.
- `backend/.../api/ParticipacionController.java` — extender `POST /api/participacion/confirmar` y `.../alta` para transportar el flag de escaneo y los cambios editados opcionales.
- `backend/.../api/dto/*` — `AltaRequest` / `ConfirmarRequest`: flag `datosDeEscaneo` + `cambios` opcionales.
- `landing/src/screens/Pantalla1bAlta.tsx` — enviar `datosDeEscaneo` según `datosEscaneados`.
- `landing/src/screens/Pantalla2Confirmar.tsx` — "Agregar {email}" (solo si vacío) + "Revisá tus datos" (editable según DNI presentado + `datos_origen`) + enmascarado hasta editar + "escaneá para corregir" cuando origen manual.
- `landing/src/api/client.ts` + `types/api.ts` — extender los requests con el flag y los cambios.
- **Patterns:** `@Version` optimista = ADR-024. Registro interno no expuesto = `voucher_rechazado`. Validación de campos = la del alta (AC-21).
- **Tests:** editar celular por path DNI → guardado + `evento_dato_cliente` escrito; editar apellido con `datos_origen=manual` → permitido; editar apellido con `datos_origen=escaneo` → rechazado; re-escaneo de cliente manual → identidad actualizada + `datos_origen` pasa a `escaneo`; editar por path reconocimiento sin DNI → rechazado; edición concurrente landing/gestor → una 409; confirmar guarda datos y chance atómicamente.

Verification

- [ ] AC-54 — Caso E con opcional vacío: "Agregar {email}" aparece solo si `email` está vacío; se puede confirmar sin completarlo; completarlo lo guarda junto con la chance en la misma transacción.
- [ ] AC-55 — Corregir datos exige DNI presentado: por el path DNI los campos permitidos son editables (enmascarados hasta editar); por el path reconocimiento (sin DNI), "Revisar mis

datos" exige presentar el DNI antes de editar.

- [ ] AC-56 — Con `datos_origen = escaneo`, `nombre/apellido/sexo/fecha_nacimiento` son de solo lectura en la landing (un intento de modificarlos es rechazado por el backend); `email` y `celular` editables con DNI presentado.
- [ ] AC-57 — Confirmar en caso E guarda los datos corregidos y suma la chance en una sola transacción atómica; abandonar antes de confirmar no persiste cambios.
- [ ] AC-58 — Todo cambio de dato del cliente genera un `evento_dato_cliente` interno (cliente, campo, valor anterior→nuevo, timestamp, voucher/ip); nunca aparece en una respuesta de la API pública.
- [ ] AC-59 — Concurrencia: edición landing vs gestor sobre el mismo cliente → una gana, la otra 409 (bloqueo optimista `@Version` en `Cliente`), sin lost update; la corrección reaplica la validación de celular (10 dígitos) y email.
- [ ] AC-60 — Con `datos_origen = manual`, `nombre/apellido/sexo/fecha_nacimiento` son editables con DNI presentado (el cliente corrige su propio typo); escanear el DNI actualiza esos datos y sube `datos_origen` a `escaneo`, tras lo cual quedan de solo lectura (AC-56).

ADR-029 — Configuración operativa completa FUERA del jar (fail-fast en todo perfil)

- **Status:** accepted (2026-08-24; pedido explícito del operador: "siempre el yaml de configuración por completo tiene que estar fuera del .jar, nada de sorpresas con yaml adentro del big fat jar")
- **Context:** Hoy `backend/src/main/resources/application.yml` viaja DENTRO del fat jar (junto con `application-e2e.properties` y `application-load.properties`). Cambiar un valor operativo en un despliegue (puerto, TLS, rutas de bases/branding, flags por cliente como `sp.reconocimiento.habilitado` de ADR-027, logging, URL de DB) obliga a reempaquetar el jar — o peor: el sistema arranca en silencio con defaults embebidos que el operador no ve. Los SECRETOS ya viven afuera (env vars `SP_DB_PASSWORD`, `SP_MASTER_KEY`, `SP_GESTOR_PASSWORD`, …); lo que falta externalizar es el resto de la configuración. Se conecta con el despliegue single-tenant por supermercado (branding + flags por cliente) y con M6 (Windows Service / jpackage: jar + carpeta `config/` es el layout natural).
- **Alternatives:**
- Statu quo (yml embebido): cambiar config = reempaquetar; defaults invisibles. **Rechazado** (es exactamente la "sorpresa" que el operador quiere matar).
- Split contrato/operativa: el jar conserva un yml mínimo con el contrato del código (Jackson snake_case, JPA validate, Flyway) y solo lo operativo va afuera. Menos frágil ante un yaml externo incompleto, pero deja config adentro del jar. **Descartado por el operador** — eligió lo literal; la fragilidad se mitiga con el chequeo de contrato (abajo).
- Fail-fast solo en prod: dev/CI seguirían con config embebida. **Descartado por el operador** — eligió uniformidad total; la fricción se mitiga versionando la referencia.
- **Decision:**
- **El jar NO embebe ningún archivo de configuración.** `src/main/resources` queda sin `application*.yml` / `application*.properties` (los archivos de `src/test/resources` no viajan en el jar y no cambian). Los perfiles de harness (`e2e`, `load`) también se mudan a `backend/config/`.
- **La config vive en `config/` junto al jar** (`backend/config/application.yml` en el repo). Se usa la búsqueda estándar de Spring Boot (`./config/` relativo al working dir) — **cero código de carga custom**; solo el chequeo de arranque es nuestro.

- **Referencia versionada, sin secretos:** `backend/config/application.yml` se versiona en git como referencia documentada (resuelve dev y CI sin fricción: ambos corren desde `backend/` y Spring la encuentra sola). Los secretos siguen SOLO en env vars — el archivo versionado referencia `${SP_*}` pero jamás contiene un valor secreto (CONSTITUTION §Security). En prod, el deploy copia jar + `config/` y el operador edita SU copia.
- **Fail-fast en TODO perfil:** sin `config/application.yml` el arranque **falla rápido** con un mensaje claro que dice qué archivo falta y dónde ponerlo. Nunca arrancar en silencio con defaults. Mecanismo: la config externa lleva una propiedad centinela (`sp.config.externa: true`) que solo existe en el archivo; un chequeo temprano de arranque la exige.
- **Chequeo de contrato al arranque:** además del centinela, el arranque valida que las claves críticas del contrato del código estén presentes y con el valor esperado — `spring.jackson.property-naming-strategy: SNAKE_CASE` (si falta, TODA la API cambia de formato y la landing muere), `spring.jpa.hibernate.ddl-auto: validate`, config de Flyway. Si falta una → **no arranca y nombra la clave faltante**. Esto cierra el riesgo de la opción literal ("borrar una línea de más rompe la API en silencio").
- **Consequences:**
- Cambiar cualquier valor operativo = editar un archivo y reiniciar. Cero reempaquetado.
- Ningún default invisible: lo que el sistema usa es lo que el operador ve en `config/`.
- La config queda documentada y con historia git (la referencia versionada).
- Layout natural para M6 (Windows Service / jpackage: jar + `config/`).
- Un despliegue MAL copiado (sin `config/`) no arranca — a propósito: mejor no arrancar que arrancar distinto a lo esperado.
- El yaml externo carga también el contrato técnico; el chequeo de contrato es obligatorio para que un archivo incompleto no pueda producir un sistema "andando pero roto".
- CI y dev dependen del archivo versionado (si alguien lo borra del repo, nada arranca — el fail-fast lo hace evidente al instante).
- Un IDE test runner cuyo directorio de trabajo sea la RAÍZ DEL REPO (p.ej. el default del VS Code Java Test Runner) rompe todo `@SpringBootTest` con el mensaje centinela — Spring busca `./config/` relativo a ese working directory, no lo encuentra ahí. Se soluciona configurando el working directory del runner a `backend/` (el default de módulo de IntelliJ ya funciona sin tocar nada). Trade-off de developer experience aceptado y documentado acá para que no sea una sorpresa silenciosa.

Implementation Plan

- **Affected paths:**
- `backend/src/main/resources/application.yml` → **mover** a `backend/config/application.yml` (versionado; secretos siguen como `${SP_*}`).
- `backend/src/main/resources/application-e2e.properties`, `application-load.properties` → mover a `backend/config/`.
- NEW chequeo de arranque (centinela + claves de contrato): `EnvironmentPostProcessor` o `ApplicationListener<ApplicationEnvironmentPreparedEvent>` en `soporte/` — falla con mensaje claro (`IllegalStateException`) nombrando archivo/clave faltante. Registrado en `META-INF/spring.factories` o equivalente Boot 4.
- `.github/workflows/ci.yml`: sin cambios esperados (los jobs corren con `working-directory: backend` → Spring encuentra `backend/config/`); verificar en verde.
- Scripts/docs de arranque (dev-env, M6): documentar que el jar exige `config/` al lado.
- **Patterns:** búsqueda estándar `./config/` de Spring Boot (sin custom loaders); fail-fast con mensaje sin secretos (mismo criterio que `GestorSecurityConfig` F7).
- **Tests:** arranque sin config externa → falla con el mensaje esperado; config sin la clave centinela → falla; config sin snake_case → falla nombrándola; jar empaquetado no contiene `application*.yml|properties`; flag operativo cambiado en el archivo externo → comportamiento cambia tras reinicio sin reempaquetar.

Verification

- [x] AC-63 — El jar empaquetado NO contiene ningún `application*.yml` ni `application*.properties` (test que inspecciona el jar); `src/main/resources` queda sin archivos de configuración. Evidencia: `ConfigExternaEmpaquetadoTest` 2/2 (recursivo).
- [x] AC-64 — Fail-fast: arrancar (cualquier perfil) sin `config/application.yml` → el proceso NO arranca y el error nombra el archivo esperado, el working directory real y dónde ponerlo; nunca arranca con defaults embebidos. Evidencia: `ConfigExternaEnvironmentPostProcessorTest` 3/3.
- [x] AC-65 — Chequeo de contrato: config externa sin `spring.jackson.property-naming-strategy=SNAKE_CASE` (o sin `ddl-auto=validate`) → el arranque falla nombrando la clave faltante. Evidencia: `ConfigExternaEnvironmentPostProcessorTest` (ac65 x2).
- [x] AC-66 — Operable sin reempaquetar: cambiar un valor operativo (p.ej. `sp.reconocimiento.habilitado`) SOLO en `config/application.yml` y reiniciar cambia el

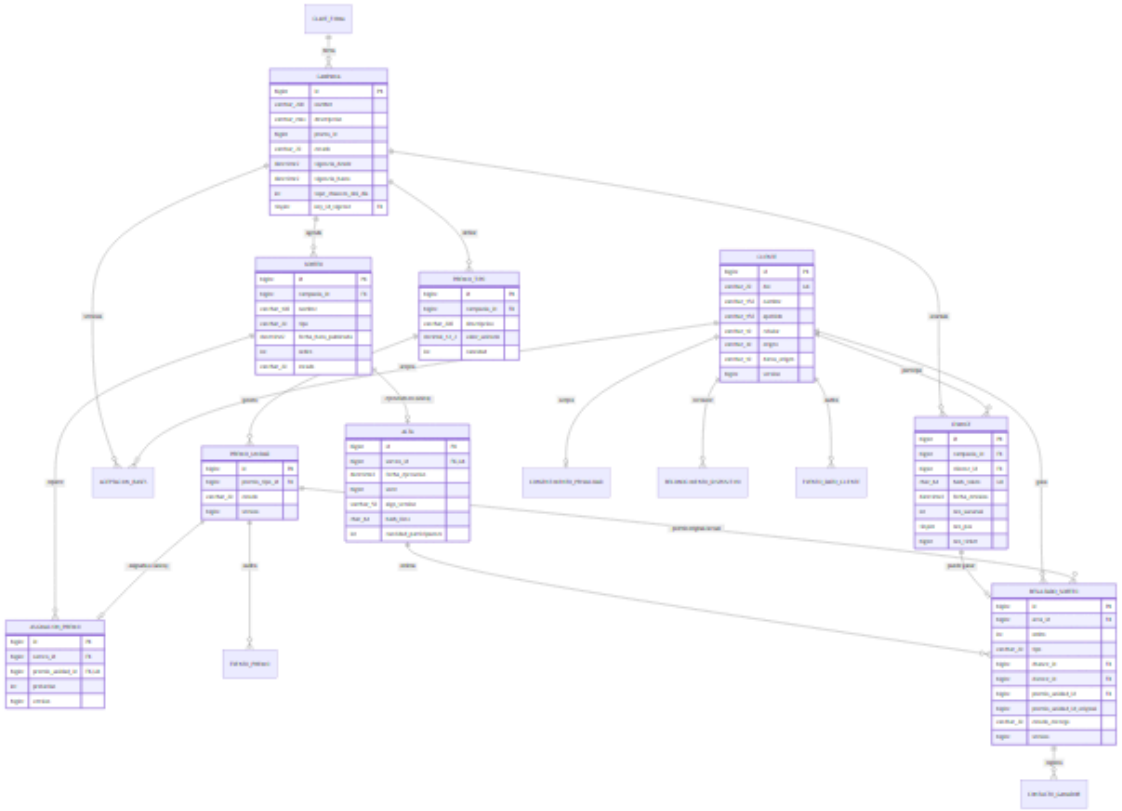
comportamiento observable; el archivo versionado de referencia no contiene ningún secreto (escaneo del archivo).

Modelo de datos

El Sistema de Premios corre sobre **SQL Server**, versionado con **Flyway** (`V1` a `V12` al momento de escribir esto). Esta página documenta el esquema real — columnas, tipos SQL Server e invariantes — tal como quedó tras aplicar todas las migraciones. Para el porqué de cada decisión de diseño (por qué una sola campaña vigente, por qué el acta es inmutable, por qué `Cliente` es un solo tipo) anda a Decisiones de arquitectura; acá vas a encontrar la forma, no el motivo.

> **Fuente de la verdad**
>
> Todo lo que sigue está tomado directo de las migraciones en `backend/src/main/resources/db/migration/` (V1 a V12). Si alguna vez este documento y el código no coinciden, gana el código — avisá para corregir la página.

Diagrama de entidades



El diagrama es un resumen

Muestra las columnas y relaciones más relevantes para entender el modelo. `clave_firma`, `voucher_rechazado`, `consentimiento_privacidad`, `aceptacion_bases`, `evento_premio`, `contacto_ganador`, `voucher_prueba`, `reconocimiento_dispositivo` y `evento_dato_cliente` se documentan en detalle más abajo, no todas en el ER.

Convenciones del esquema

- **Nombres de tabla y columna**: `snake_case`, en español (mismo vocabulario que CONTEXT.md del proyecto).
- **PKs**: `BIGINT IDENTITY(1,1)` en casi todas las tablas — la excepción es `clave_firma`, cuya PK es el propio `key_id TINYINT` (no hace falta un id sintético para una clave).
- **Timestamps**: `DATETIME2`, casi siempre `NOT NULL DEFAULT SYSUTCDATETIME()` para las marcas de auditoría (`creado_en`, `fecha_registro`, `fecha_hora`).

- **Enums como `VARCHAR` + `CHECK`**: el proyecto no usa tipos enumerados nativos de SQL Server; cada estado es un `VARCHAR` acotado por un `CONSTRAINT ck_..._... CHECK (columna IN (...))`. Se listan los valores válidos en cada tabla más abajo.
- **Bloqueo optimista (`version`)**: `premio_unidad`, `asignacion_premio`, `resultado_sorteo` y `cliente` tienen una columna `version BIGINT NOT NULL DEFAULT 0` (agregada en V9 y V12). Hibernate la incrementa en cada `UPDATE`; si otra transacción ya avanzó la fila, la operación falla con `OptimisticLockException`, traducido a **409** por el manejador de errores del gestor. Motivo: son las cuatro entidades que dos operadores del gestor pueden llegar a mutar en simultáneo (entrega, pase a suplente, devolución al stock, edición de cliente).
- **Hash, nunca el dato sensible en claro**: `chance.hash_token`, `voucher_prueba.hash_token` y `reconocimiento_dispositivo.hash_reconocimiento` persisten un `CHAR(64)` (SHA-256 hex) — nunca el token del QR ni el secreto de reconocimiento del dispositivo. Ver El token del QR para el detalle de cómo se calcula ese hash.

campania

La promoción completa. Solo puede existir **una** fila con `estado = 'vigente'` — invariante garantizada a nivel de base de datos, no solo en la capa de aplicación.

Columna	Tipo SQL Server	Notas
`id`	`BIGINT IDENTITY(1,1)`	PK
`nombre`	`VARCHAR(200) NOT NULL`	Badge visible en la landing
`descripcion`	`VARCHAR(MAX) NOT NULL`	Texto de campaña
`promo_id`	`BIGINT NOT NULL`	Id de promoción del backoffice (M1) — política de reuso diferida (ADR-014)
`estado`	`VARCHAR(20) NOT NULL`	`borrador` | `vigente` | `cerrada`

`vigencia_desde` / `vigencia_hasta`	`DATETIME2 NOT NULL`	Ventana de vigencia por fecha de **emisión** del voucher
`regla_chances`	`INT NOT NULL DEFAULT 1`	1 voucher = 1 chance
`tope_chances_dni_dia`	`INT NOT NULL DEFAULT 5`	Tope diario por DNI
`bases_url`	`VARCHAR(500) NOT NULL`	Ruta pública del PDF de bases vigente
`version_bases`	`VARCHAR(50) NOT NULL`	Versión inmutable de las bases aceptadas
`plazo_retiro_dias`	`INT NOT NULL DEFAULT 10`	Días desde el primer contacto para retirar el premio
`key_id_vigente`	`TINYINT NOT NULL`	FK a `clave_firma.key_id`

⚠ **Invariante: única campaña vigente**

```
CREATE UNIQUE INDEX ux_campania_vigente ON campania (estado) WHERE estado =
'vigente';
```

Es un índice único **filtrado**: no restringe cuántas filas `borrador` o `cerrada` puede haber, solo impide que dos campañas estén `vigente` al mismo tiempo. Un `INSERT` / `UPDATE` que viole esto falla en el `flush` inmediato (`saveAndFlush`), no en un commit posterior.

También hay un `CHECK (vigencia_desde < vigencia_hasta)` (`ck_campania_vigencia` , V9) como segunda línea de defensa por si algún camino de escritura se saltea la validación del DTO.

sorteo

Cada evento de sorteo dentro de una campaña: los semanales más el de cierre.

Columna	Tipo SQL Server	Notas

`id`	`BIGINT IDENTITY(1,1)`	PK
`campania_id`	`BIGINT NOT NULL`	FK → `campania`
`nombre`	`VARCHAR(100) NOT NULL`	"Semana 1".."Semana 5", "Cierre"
`tipo`	`VARCHAR(20) NOT NULL`	`semanal` | `cierre`
`fecha_hora_publicada`	`DATETIME2 NOT NULL`	Referencia en bases — **no** dispara la ejecución
`orden`	`INT NOT NULL`	Orden dentro de la campaña
`estado`	`VARCHAR(20) NOT NULL`	`pendiente` | `ejecutado`
`suplentes_a_sortear`	`INT NOT NULL DEFAULT 10`	Suplentes de esta acta

Invariante

La ejecución es siempre manual desde el gestor (ADR-009). `ejecutado` implica que existe un `acta` para ese sorteo (`uq_acta_sorteo`); nunca hay un sorteo `ejecutado` sin acta.

premio_tipo

Catálogo de premios por tipo y cantidad ("Bicicleta playera × 10").

Columna	Tipo SQL Server	Notas
`id`	`BIGINT IDENTITY(1,1)`	PK
`campania_id`	`BIGINT NOT NULL`	FK → `campania`
`descripcion`	`VARCHAR(200) NOT NULL`	Descripción comercial

`valor_unitario`	`DECIMAL(12,2) NOT NULL`	Valor de referencia
`cantidad`	`INT NOT NULL`	Cuántas unidades genera

premio_unidad

Cada unidad física individual generada a partir de un `premio_tipo`.

Columna	Tipo SQL Server	Notas
`id`	`BIGINT IDENTITY(1,1)`	PK
`premio_tipo_id`	`BIGINT NOT NULL`	FK → `premio_tipo`
`estado`	`VARCHAR(20) NOT NULL DEFAULT 'Disponible'`	`Disponible` → `Asignado` → `Sorteado` → `Otorgado` | `NO_OTORGADO`
`version`	`BIGINT NOT NULL DEFAULT 0`	Bloqueo optimista (V9)

> **Invariante: stock_disponible no es un contador paralelo**
>
> `stock_disponible` **siempre** se calcula como `COUNT(*) WHERE estado = 'Disponible'` sobre esta tabla en el momento de la consulta — nunca hay una columna separada que lo cachee y pueda desincronizarse.

asignacion_premio

Vincula una `premio_unidad` a un `sorteo`, con su orden de prelación.

Columna	Tipo SQL Server	Notas
`id`	`BIGINT IDENTITY(1,1)`	PK

`sorteo_id`	`BIGINT NOT NULL`	FK → `sorteo`
`premio_unidad_id`	`BIGINT NOT NULL UNIQUE`	FK → `premio_unidad` — una unidad, a lo sumo, una asignación viva
`prelacion`	`INT NOT NULL`	1º, 2º… orden de asignación de premios dentro del sorteo
`version`	`BIGINT NOT NULL DEFAULT 0`	Bloqueo optimista (V9)

Invariante: prelación única por sorteo

`CREATE UNIQUE INDEX ux_asignacion_sorteo_prelacion ON asignacion_premio (sorteo_id, prelacion)` (V2) — segunda línea de defensa a nivel DB; el servicio ya valida antes de insertar.

cliente

El padrón. Una persona identificada por DNI — un único tipo de entidad tanto para quien participa del sorteo como para altas post-campaña (D5 del dominio: sin un tipo `lead` aparte).

Columna	Tipo SQL Server	Notas
`id`	`BIGINT IDENTITY(1,1)`	PK
`dni`	`VARCHAR(20) NOT NULL UNIQUE`	Alta idempotente por DNI
`nombre` / `apellido`	`VARCHAR(150) NOT NULL`	
`celular`	`VARCHAR(10) NOT NULL`	Normalizado, solo dígitos
`fecha_nacimiento`	`DATE NULL`	`CHECK (fecha_nacimiento IS NULL OR <= hoy)` (V8)

`sexo`	`VARCHAR(1) NULL`	`M` | `F` | `X`
`email`	`VARCHAR(255) NULL`	
`creado_en`	`DATETIME2 NOT NULL DEFAULT SYSUTCDATETIME()`	
`origen`	`VARCHAR(30) NOT NULL`	`participacion` | `post_campania`
`version`	`BIGINT NOT NULL DEFAULT 0`	Bloqueo optimista (V12) — edición concurrente landing/gestor
`datos_origen`	`VARCHAR(10) NOT NULL DEFAULT 'manual'`	`escaneo` | `manual` (V12) — gobierna si la identidad es editable

datos_origen: por qué existe (V12 / ADR-028)

Si el cliente cargó sus datos escaneando el código de barras del DNI físico (`datos_origen = 'escaneo'`), la landing trata nombre/apellido/sexo/fecha de nacimiento como **solo lectura** — no se pueden "corregir" datos que vinieron de una fuente confiable sin pasar por soporte. Si vinieron de carga manual, sí son editables presentando el DNI. El celular y el email nunca cuentan como "cambio de identidad": siempre son editables.

chance

Una participación registrada: un voucher consumido + un DNI. Es la entidad de mayor volumen del sistema — tiene tres índices dedicados a los caminos calientes del motor de sorteo y el panel.

Columna	Tipo SQL Server	Notas
`id`	`BIGINT IDENTITY(1,1)`	PK
`campania_id`	`BIGINT NOT NULL`	FK → `campania`
`cliente_id`	`BIGINT NOT NULL`	FK → `cliente`

`hash_token`	`CHAR(64) NOT NULL UNIQUE`	SHA-256 hex de la codificación canónica del token
`fecha_emision`	`DATETIME2 NOT NULL`	Del token (offsets 8/10) — **la** fecha que gobierna toda decisión temporal
`nro_sucursal`	`INT NOT NULL`	Del token (offset 5)
`nro_pos`	`TINYINT NOT NULL`	Del token (offset 7)
`nro_ticket`	`BIGINT NOT NULL`	Del token (offset 12)
`key_id`	`TINYINT NOT NULL`	Del token (offset 0)
`fecha_registro`	`DATETIME2 NOT NULL DEFAULT SYSUTCDATETIME()`	Cuándo se registró en el server (distinta de `fecha_emision`)

Invariante: consumo atómico, nunca el token en claro

`uq_chance_hash_token` es la única defensa contra duplicar una chance ante un doble envío simultáneo del mismo voucher: el `INSERT` que pierde la carrera falla por violación de UNIQUE, y ese camino resuelve a Caso B (ya usado), nunca a una segunda chance. `hash_token` es lo único que se persiste — el token en sí **nunca** toca la base de datos. Ver El token del QR.

Índices de camino caliente: `(cliente_id, campania_id, fecha_emision)` y `(campania_id, fecha_emision)` para la ventana de elegibles del motor de sorteo y el tope diario (V2); `(campania_id, fecha_registro)` para las señales antifraude del panel (V7).

voucher_rechazado

Registro interno de todo voucher rechazado — nunca se le muestra al cliente el motivo real (mensaje siempre genérico), pero queda acá para antifraude y soporte.

Columna	Tipo SQL Server	Notas

`id`	`BIGINT IDENTITY(1,1)`	PK
`hash_token`	`CHAR(64) NULL`	Nulo si la firma era inválida (no hay payload confiable que hashear)
`motivo`	`VARCHAR(30) NOT NULL`	`firma_invalida` | `ya_usado` | `fuera_vigencia` | `tope_diario` | `bot` (V4)
`fecha`	`DATETIME2 NOT NULL DEFAULT SYSUTCDATETIME()`	
`ip`	`VARCHAR(45) NOT NULL`	
`nro_sucursal` / `nro_pos`	`INT` / `TINYINT NULL`	Solo si el token era decodificable

El motivo `bot` (V4) es el honeypot: un campo invisible del formulario (`contacto_web`) que solo un bot llena — si llega no vacío, se registra acá y la respuesta es idéntica a la de Caso A.

acta

Snapshot inmutable y reproducible de la ejecución de un sorteo.

Columna	Tipo SQL Server	Notas
`id`	`BIGINT IDENTITY(1,1)`	PK
`sorteo_id`	`BIGINT NOT NULL UNIQUE`	FK → `sorteo` — un sorteo, a lo sumo, un acta
`fecha_ejecucion`	`DATETIME2 NOT NULL`	
`seed`	`BIGINT NOT NULL`	Semilla del sorteo

`algo_version`	`VARCHAR(50) NOT NULL`	Versión del algoritmo de sorteo
`filtros`	`VARCHAR(MAX) NOT NULL`	Ventana + exclusiones, serializado
`lista_chance_ids`	`VARCHAR(MAX) NOT NULL`	Lista ordenada, snapshot completo de elegibles
`hash_lista`	`CHAR(64) NOT NULL`	Hash de la lista, para verificar integridad
`cantidad_participantes`	`INT NOT NULL`	

Invariante: inmutable y reproducible (D4)

`lista_chance_ids + seed + algo_version → siempre el mismo resultado`. No existe una operación de "re-ejecutar" un acta; una ejecución errónea se corrige por decisión humana documentada fuera del sistema (nueva asignación de premios + nuevo sorteo). Ver El motor de sorteo y el acta.

resultado_sorteo

Cada posición del acta: ganador o suplente, en orden.

Columna	Tipo SQL Server	Notas
`id`	`BIGINT IDENTITY(1,1)`	PK
`acta_id`	`BIGINT NOT NULL`	FK → `acta`
`orden`	`INT NOT NULL`	1..N+suplentes
`tipo`	`VARCHAR(20) NOT NULL`	`ganador` | `suplente`

`chance_id` / `cliente_id`	`BIGINT NOT NULL`	FK → `chance` / `cliente`
`premio_unidad_id`	`BIGINT NULL`	**Mutable**: cambia con pase a suplente / devolución al stock
`premio_unidad_id_original`	`BIGINT NULL`	**Inmutable** (V10): el premio efectivamente sorteado, nunca tocado por operaciones de entrega
`estado_entrega`	`VARCHAR(30) NOT NULL DEFAULT 'pend_contacto'`	`pend_contacto` | `contactado` | `entregado` | `pasado_a_suplente` | `devuelto` (V3)
`dni_verificado_en_entrega`	`VARCHAR(20) NULL`	DNI presentado al momento de entregar
`fecha_entrega` / `sucursal_entrega` / `responsable_entrega`		Datos de la entrega física
`version`	`BIGINT NOT NULL DEFAULT 0`	Bloqueo optimista (V9)

Por qué dos columnas de premio_unidad_id (V10 / ADR-025)

El acta tiene que seguir siendo reproducible (`seed + lista + algo → mismos ganadores`), pero `premio_unidad_id` **muta** con `registrarEntrega` / `pasarASuplente` / `devolverAlStock` . `premio_unidad_id_original` se fija una única vez al ejecutar el sorteo y nunca se vuelve a tocar — es lo que separa, en el export del acta, la sección "inmutable" (lo sorteado) de la sección "estado de entrega" (snapshot, lo entregado a la fecha del export).

contacto_ganador

Cada intento de contacto con un ganador o suplente — prueba frente al plazo de retiro.

Columna	Tipo SQL Server	Notas
`id`	`BIGINT IDENTITY(1,1)`	PK
`resultado_id`	`BIGINT NOT NULL`	FK → `resultado_sorteo`
`fecha`	`DATETIME2 NOT NULL`	
`canal`	`VARCHAR(20) NOT NULL`	`llamada` | `whatsapp` | `otro`
`resultado_contacto`	`VARCHAR(200) NOT NULL`	Texto libre

clave_firma

Claves HMAC de firma del token, por `key_id` — permiten rotar sin cortar la emisión.

Columna	Tipo SQL Server	Notas
`key_id`	`TINYINT`	PK (no hay id sintético)
`secreto_cifrado`	`VARBINARY(512) NOT NULL`	Cifrado at-rest, nunca en claro
`estado`	`VARCHAR(20) NOT NULL`	`activa` | `rotada` | `revocada`
`vigencia_desde`	`DATETIME2 NOT NULL`	
`vigencia_hasta`	`DATETIME2 NULL`	

consentimiento_privacidad

Consentimiento permanente del cliente a comunicaciones — una fila por cliente.

Columna	Tipo SQL Server	Notas
`id`	`BIGINT IDENTITY(1,1)`	PK

`cliente_id`	`BIGINT NOT NULL UNIQUE`	FK → `cliente`
`fecha_hora`	`DATETIME2 NOT NULL`	
`acepta_comunicaciones`	`BIT NOT NULL`	

aceptacion_bases

Aceptación de las bases y condiciones, por campaña y versión.

Columna	Tipo SQL Server	Notas
`id`	`BIGINT IDENTITY(1,1)`	PK
`cliente_id`	`BIGINT NOT NULL`	FK → `cliente`
`campania_id`	`BIGINT NOT NULL`	FK → `campania`
`version_bases`	`VARCHAR(50) NOT NULL`	Las versiones son inmutables (ADR-007): nunca se pisa un PDF ya aceptado
`fecha_hora`	`DATETIME2 NOT NULL`	

Invariante

`UNIQUE (cliente_id, campania_id, version_bases)` — el botón de confirmar participación implica esta aceptación (UX de un toque, sin checkbox aparte).

evento_premio

Auditoría de todo cambio de estado de una `premio_unidad`.

Columna	Tipo SQL Server	Notas
`id`	`BIGINT IDENTITY(1,1)`	PK
`premio_unidad_id`	`BIGINT NOT NULL`	FK → `premio_unidad`
`evento`	`VARCHAR(30) NOT NULL`	`asignado` | `desasignado` | `sorteado` | `otorgado` | `devuelto_a_stock` | `reasignado_suplente` | `no_otorgado`
`motivo`	`VARCHAR(300) NULL`	Obligatorio en la práctica para toda reasignación/devolución
`usuario`	`VARCHAR(150) NOT NULL`	Operador del gestor que hizo el cambio
`fecha_hora`	`DATETIME2 NOT NULL DEFAULT SYSUTCDATETIME()`	

voucher_prueba

Auditoría de cada voucher generado por el **modo prueba** del gestor (nunca habilitable en producción — el bean del controller ni se registra sin el flag `sp.modo-prueba.enabled`).

Columna	Tipo SQL Server	Notas
`id`	`BIGINT IDENTITY(1,1)`	PK
`campania_id`	`BIGINT NOT NULL`	FK → `campania`
`key_id`	`TINYINT NOT NULL`	

`fecha_emision`	`DATETIME2 NOT NULL`	
`hash_token`	`CHAR(64) NOT NULL UNIQUE`	Mismo criterio que `chance`: nunca se persiste el token en claro
`usuario`	`VARCHAR(150) NOT NULL`	Operador que lo generó
`creado_en`	`DATETIME2 NOT NULL DEFAULT SYSUTCDATETIME()`	

reconocimiento_dispositivo

Agregada en V11 (ADR-027). Permite que la landing "reconozca" un dispositivo ya usado para saltar el tipeo del DNI en visitas siguientes, sin persistir el DNI en el navegador.

Columna	Tipo SQL Server	Notas
`id`	`BIGINT IDENTITY(1,1)`	PK
`cliente_id`	`BIGINT NOT NULL`	FK → `cliente` — **N filas por cliente**, una por dispositivo
`hash_reconocimiento`	`CHAR(64) NOT NULL UNIQUE`	Hash del secreto de reconocimiento; nunca el secreto en claro
`creado_en`	`DATETIME2 NOT NULL DEFAULT SYSUTCDATETIME()`	
`ultimo_uso_en`	`DATETIME2 NULL`	

Vida permanente, cruza campañas

A diferencia de `chance` (scoped a una campaña), el reconocimiento de dispositivo no expira ni se acota a una campaña — decisión explícita del ADR-027.

evento_dato_cliente

Agregada en V12 (ADR-028). Auditoría de todo cambio de dato de un cliente — nunca expuesta por la API pública, mismo criterio que `voucher_rechazado`.

Columna	Tipo SQL Server	Notas
`id`	`BIGINT IDENTITY(1,1)`	PK
`cliente_id`	`BIGINT NOT NULL`	FK → `cliente`
`campo`	`VARCHAR(30) NOT NULL`	Qué campo cambió
`valor_anterior` / `valor_nuevo`	`NVARCHAR(255) NULL`	
`cambiado_en`	`DATETIME2 NOT NULL DEFAULT SYSUTCDATETIME()`	
`hash_token` / `ip`	`CHAR(64) NULL` / `VARCHAR(45) NULL`	Contexto del cambio, si vino de la landing

Índices de soporte de FK

SQL Server, a diferencia de MySQL/InnoDB, **no crea automáticamente** un índice para cada foreign key. La migración `V6__indices_fk.sql` agrega un índice no-clustered dedicado a cada una de las 10 FKs que no tenían ya un índice compuesto cuyas columnas líderes coincidieran — sin esto, los joins contra la tabla referenciada podían forzar un table scan completo de la tabla hija.

Ver también

- El token del QR (Anexo A) — cómo se calcula `chance.hash_token`.
- API HTTP — superficie que lee y escribe sobre este esquema.
- Casos de participación (A–E) — el flujo de negocio que crea filas en `chance`, `cliente` y `voucher_rechazado`.

- El motor de sorteo y el acta — cómo se llenan `acta` y `resultado_sorteo`.

El token del QR (Anexo A)

Todo el sistema de participación depende de un solo contrato: el **token** de 35 caracteres que viaja en la URL del QR impreso en cada ticket. Esta página es la referencia normativa de ese token — el layout de bytes, cómo se calcula, cómo se valida y qué se persiste. El **Anexo A** al que hace referencia el título es la sección normativa del PDF de especificación funcional (v2.2) que define este layout; el módulo `qrtoken` (C89 en el POS + espejo Java en el backend) es su única implementación.

> **Por qué importa que esto sea exacto**
>
> El POS corre en **MS-DOS sobre Borland C++ 3.1** (hardware real de las cajas de Caracol) y el backend corre en **Java 17 / Spring Boot**. Son dos mundos que nunca se sincronizan entre sí: lo único que comparten es la clave HMAC. Si el layout de bytes no coincide byte a byte entre ambas implementaciones, cada voucher impreso sería inválido. Por eso el módulo se trata como **código compartido validado**, no como algo que se reimplementa por conveniencia.

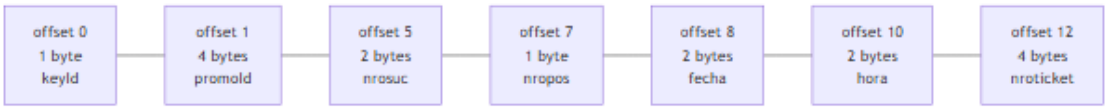
La idea: el QR es autocontenido

El ticket imprime una URL con este formato:

```
https://sorteo.supercaracol.com.ar/p/<token>
```

Todo lo que la landing necesita para resolver la participación —sucursal, caja, fecha y hora de emisión, número de comprobante, y qué campaña corresponde— viaja **firmado dentro del token**. No existe sincronización entre las cajas y el servidor web: la caja imprime offline, y el servidor valida la firma sin depender de ningún dato compartido salvo la clave.

Layout del payload (16 bytes, big-endian)



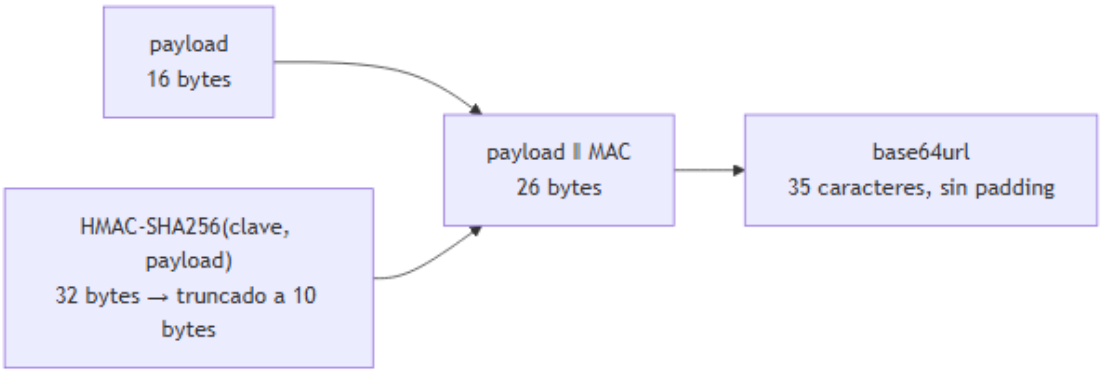
Offset	Largo	Campo	Descripción
0	1	`keyId`	Id de clave de firma, para rotación sin corte (0–255)
1	4	`promoId`	Promoción, `long` nativo del backoffice de POS (0–2^{32}-1). Política de asignación/reuso **diferida** (ADR-014)
5	2	`nrosuc`	Sucursal (0–65535)
7	1	`nropos`	Caja / POS (0–255)
8	2	`fecha`	Días transcurridos desde 2026-01-01
10	2	`hora`	Minutos del día (0–1439)
12	4	`nroticket`	Número de comprobante (0–4294967295)

16 bytes exactos, sin relleno explícito entre campos — el layout es una tira contigua.

> **La firma autentica, no encripta**
>
> Los campos viajan **legibles a propósito**: la landing muestra directamente "Ticket 45678 · Sucursal 4 · 19/08/2026 14:32" leyendo el payload decodificado, sin ninguna consulta adicional. Lo que el HMAC garantiza es que ese payload **no fue alterado ni forjado** — no que sea secreto. Con 80 bits de MAC (ver más abajo), forjar un token es computacionalmente inviable.

Cómo se calcula el token

```
token = base64url( payload[16 bytes] ‖ HMAC-SHA256(clave, payload)[0..9] )
      = base64url( 26 bytes )
      = 35 caracteres, sin padding
```



1. Se arma el `payload` de 16 bytes con el layout de la sección anterior.
2. Se calcula `HMAC-SHA256(clave, payload)` (32 bytes) y se **trunca a los primeros 10 bytes** (80 bits).
3. Se concatena `payload ‖ mac_truncado` → 26 bytes.
4. Se codifica en `base64url` sin padding → 35 caracteres. Ese string es el `<token>` de la URL.

> **Por qué 10 bytes de MAC y no los 32 completos**
>
> 80 bits es inviable de enumerar por fuerza bruta, y dejar el MAC truncado permite que la URL completa entre en un QR **versión 3** — que imprime nítido en la impresora térmica de 58/80 mm de las cajas. Con el MAC completo, la URL sería más larga y exigiría una versión de QR más densa, con peor tasa de lectura en térmica.

Codificación canónica (obligatoria en la validación)

`base64url` tiene una particularidad: los últimos bits de un string codificado pueden ser relleno, así que **más de un string** puede decodificar exactamente a los mismos 26 bytes. Si el servidor no normalizara esto, alguien podría "reescribir" un token ya usado con una codificación alternativa — misma firma válida, pero un `hash_token` distinto — y duplicar la chance sobre el mismo voucher físico.

Por eso, `validar()` exige la **codificación canónica**: decodifica el token, y si volver a codificar esos mismos bytes no reproduce el string original, lo rechaza. Dos codificaciones del mismo voucher siempre resuelven al mismo registro.

Qué se persiste: el hash, nunca el token

```
QrToken.Datos d = QrToken.validar(token, keyId -> claves.get(keyId));
String pk = QrToken.hashToken(token);   // esto se persiste, no el token
```

`chance.hash_token` (y también `voucher_prueba.hash_token`) guardan un `CHAR(64)` — el hash SHA-256 en hexadecimal de la codificación canónica del token. El token en sí **nunca** toca la base de datos. Motivo: si la base de datos se filtrara, nadie podría reconstruir URLs de QR válidas a partir de los hashes. Ver la tabla `chance` en el modelo de datos para el resto de las columnas que acompañan al hash.

Comparación en tiempo constante

La verificación del MAC usa `MessageDigest.isEqual` (equivalente a una comparación de tiempo constante) en vez de una comparación byte a byte con salida temprana — evita que un atacante pueda inferir el MAC correcto midiendo cuánto tarda cada intento fallido (timing attack).

Vector de referencia

Este es el vector de prueba que ambos autotests (el de C89 del POS y el de Java del backend) deben reproducir exactamente. **La clave es solo para test — nunca se usa en producción.**

```
Clave  : "CaracolDemoKey2026!"            (SOLO PARA TEST — no usar en prod)
Datos  : keyId=1 promoId=1 suc=4 pos=2  19/08/2026 14:32  ticket=45678
Token  : AQAAAAEABAIA5gNoAACybvOHZD3fe037qgw
```

```
# Autotest C (POS, BC3.1 o gcc para CI)
BCC -ms QRTEST.C QRTOKEN.C && QRTEST
# o: gcc -x c -std=c89 -pedantic -Wall -Wextra -O2 -o qrtest QRTEST.C QRTOKEN.C
&& ./qrtest

# Autotest Java (backend)
javac QrToken.java && java QrToken
```

Ambos autotests deben terminar en `TODO OK` y producir el token de arriba. Si alguna vez difieren, el layout de bytes o el HMAC dejaron de coincidir entre las dos implementaciones — bloqueante para cualquier despliegue.

Rotación de claves

- `keyId` (offset 0) permite rotar la clave de firma **sin cortar** la emisión: el backend mantiene un mapa `keyId → clave` y acepta cualquier clave vigente; el POS emite siempre con la clave activa del momento.
- El registro `clave_firma(key_id, secreto_cifrado, estado, vigencia_desde, vigencia_hasta)` vive en la base de datos — el secreto se cifra at-rest y nunca se loguea ni se devuelve en ninguna respuesta de la API del gestor. Ver clave_firma en el modelo de datos y el recurso `claves` en la API del gestor.
- En el binario DOS del POS, la clave queda embebida en el ejecutable: rotarla por campaña acota la ventana de exposición si alguien la extrae del binario.

Ver también

- Modelo de datos — dónde y cómo se persiste `hash_token`.
- API HTTP — los endpoints públicos que reciben el token (`resolver`, `confirmar`, `alta`) y el recurso de claves del gestor.
- Casos de participación (A–E) — cómo cada resultado de validar el token (firma inválida, ya usado, fuera de vigencia, vigente) deriva en una pantalla distinta de la landing.

API HTTP

El backend expone dos superficies HTTP bien separadas, sobre el mismo jar: la **API pública** que consume la landing (anónima, sin login) y la **API del gestor** (autenticada, para el operador de Caracol). Esta página documenta ambas — método, ruta, body y respuesta — tal como están implementadas hoy.

> **Convenciones generales**
>
> - **JSON en `snake_case`**, siempre — `spring.jackson.property-naming-strategy: SNAKE_CASE` está fijado globalmente (`application.yml`). Un campo Java `premioUnidadId` viaja como `premio_unidad_id` en el body y en la respuesta.
> - **Respuestas de error genéricas hacia el cliente público**: la API de participación nunca revela el motivo real de un rechazo (firma inválida, ya usado, etc.) — ese detalle queda solo en el registro interno (`voucher_rechazado`) y en logs. Ver Errores más abajo.
> - Todas las rutas bajo `/api/gestor/**` exigen **HTTP Basic** (usuario/password por variable de entorno). El resto de la API (`/api/participacion/**`, `/api/padron/**`, `/bases/**`, `/api/campania/vigente`, `/api/salud`, `/api/branding`) es pública (`permitAll`).

API pública (landing)

Cubre los cinco resultados de escanear un QR (Casos A–E) más el alta al padrón fuera de campaña. Para el detalle del flujo de negocio detrás de cada caso, ver Casos de participación (A–E).

GET /p/{token}

Sirve el HTML de la landing (Preact) con el **estado inicial embebido** en un `<script id="sp-estado" type="application/json">`: `{token, campania, resolucion}` — el mismo payload que hoy resolverían `GET /api/campania/vigente` + `POST /api/participacion/resolver` (sin `dni`), pero sin los dos viajes de red adicionales antes del primer render (importante en 3G de salón). Si algo falla al resolver cualquiera de los dos campos, cae a `null` — nunca rompe la página, y el frontend cae al camino de fetch de siempre.

POST /api/participacion/resolver

Decodifica el token (validación canónica del Anexo A) y resuelve a qué caso corresponde, **sin consumir el voucher**. Es de lectura — se puede llamar más de una vez sobre el mismo token sin efecto.

```
// request
{
  "token": "AQAAAAEABAIA5gNoAACybvOHZD3fe037qgw",
  "dni": "30123456"
}
```

`dni` es **opcional**. Sin él, un token con firma válida y vigente resuelve al estado intermedio `PENDIENTE_DNI` (la Pantalla 1a recién ahí pide el DNI). Con `dni` presente, la resolución avanza hasta D, E o TOPE.

```
// response — ejemplo Caso E (DNI ya registrado)
{
  "caso": "E",
  "mensaje": "Participación confirmada.",
  "ticket": {
    "nro_ticket": 45678,
    "nro_sucursal": 4,
    "fecha_emision": "2026-08-19T14:32:00"
  },
  "cliente": { "nombre": "Juan", "dni_mascarado": "***23456" },
  "chances_acumuladas": 3,
  "datos_origen": "manual",
  "email_mascarado": null,
  "celular_mascarado": "***1234"
}
```

`caso`	**Significa**	**Pantalla**
`A`	Firma inválida	Error genérico, sin pistas
`B`	Token ya usado	"Ya participó" + `fecha_registro_original`
`C`	Fuera de vigencia por fecha de emisión	Alta a padrón sin chance (Pantalla 4)

`PENDIENTE_DNI`	Vigente, sin `dni` en el request	Pide DNI (Pantalla 1a)
`D`	Vigente + DNI nuevo	Requiere `POST /alta`
`E`	Vigente + DNI ya en el padrón	Requiere `POST /confirmar`
`TOPE`	DNI ya alcanzó el tope diario de chances	Rechazo sin consumir el voucher

`ParticipacionResponse` es la forma **única** que comparten `resolver` / `confirmar` / `alta` / `padron/alta`: cada campo viaja solo si aplica al caso (`@JsonInclude(NON_NULL)`).

POST /api/participacion/confirmar

Caso E: el DNI ya está en el padrón. Consume la chance — es **idempotente** (un retry sobre el mismo token nunca duplica la chance; resuelve a Caso B).

```
// request — confirmación simple
{ "token": "...", "dni": "30123456" }
```

```
// request — con corrección/completado de datos (ADR-028)
{
  "token": "...",
  "dni": "30123456",
  "cambios": {
    "email": "juan@mail.com",
    "celular": "3511234567",
    "nombre": null,
    "apellido": null,
    "sexo": null,
    "fecha_nacimiento": null
  },
  "datos_de_escaneo": false
}
```

`cambios` es opcional y cada campo dentro de él también lo es — solo se aplica lo no nulo. El `celular` / `email` siempre son editables presentando el DNI; nombre/apellido/sexo/fecha de nacimiento solo si `cliente.datos_origen = 'manual'` (ver modelo de datos). Intentar editar identidad de un cliente `datos_origen = 'escaneo'` responde **400 genérico** (nunca expone la regla al cliente).

POST /api/participacion/alta

Caso D: DNI nuevo. Alta de cliente + consumo de la chance, transaccional.

```
// request
{
  "token": "...",
  "dni": "30123456",
  "nombre": "Juan",
  "apellido": "Pérez",
  "celular": "3511234567",
  "fecha_nacimiento": "1990-05-10",
  "sexo": "M",
  "email": "juan@mail.com",
  "datos_de_escaneo": false
}
```

`celular` se normaliza a 10 dígitos automáticamente (se descarta todo lo que no sea número: espacios, guiones, `+54`). El request incluye además un campo honeypot invisible (`contacto_web`), documentado más abajo en Antifraude.

POST /api/padron/alta

Caso C (Pantalla 4a): el voucher es válido pero está fuera de la vigencia de la campaña por fecha de emisión. No se registra chance — solo alta al padrón para próximas promociones.

```
// request
{
  "token": "...",
  "dni": "30123456",
  "nombre": "Juan",
  "apellido": "Pérez",
  "celular": "3511234567"
}
```

GET /api/campania/vigente

Datos públicos de la campaña vigente que la landing necesita para pintarse (nombre, descripción, bases, vigencia) más la identidad visual del cliente (`cliente_nombre`, `color_primario`, `logo_url` — los mismos valores que `GET /api/branding`) y el flag `reconocimiento_habilitado`. **404** si no hay ninguna campaña vigente (la base de datos garantiza a lo sumo una).

```
// response
{
  "nombre": "Sorteo 50 años",
  "descripcion": "...",
  "bases_url": "/bases/v1.pdf",
  "version_bases": "v1",
  "vigencia_desde": "2026-09-01T00:00:00",
  "vigencia_hasta": "2026-10-09T23:59:00",
  "cliente_nombre": "Supermercados Caracol",
  "color_primario": "#...",
  "logo_url": "/branding/logo.png",
  "reconocimiento_habilitado": true
}
```

Reconocimiento de dispositivo (extensión, ADR-027)

Tres endpoints adicionales permiten que la landing "recuerde" un dispositivo ya usado para saltar el tipeo del DNI en visitas siguientes — sin persistir el DNI en el navegador (el DNI nunca baja al cliente; solo un secreto opaco).

Método	Ruta	Para qué
POST	`/api/participacion/reconocer`	`{secreto}` → `{encontrado, cliente?}`. `encontrado=false` es una respuesta válida (nunca error) — instruye a caer al alta por DNI de siempre
POST	`/api/participacion/confirmar-reconocido`	`{token, secreto}` — variante de `confirmar` sin DNI: el cliente se resuelve server-side desde el secreto
—	—	Al confirmar/dar de alta con éxito (D/E), la respuesta incluye `secreto_reconocimiento` **una sola vez** — el frontend lo guarda para la próxima visita

Antifraude

- **Honeypot invisible**: `AltaRequest` y `PadronAltaRequest` aceptan un campo opcional `contacto_web`, que el frontend renderiza oculto (`display:none`, sin label). Un humano nunca lo completa; un bot que llena todos los campos sí. Si llega no vacío, la respuesta es idéntica a Caso A (sin pistas) y se registra en `voucher_rechazado` con motivo `bot` — nunca crea cliente ni chance.

- **Rate limit** por conexión sobre `/api/*` y `/p/*`.
- **Tope diario** de chances por DNI (parámetro de campaña, default 5) — rechazo sin consumir el voucher, caso `TOPE`.

Errores y códigos de estado

Situación	HTTP	Body
Validación de campos (`@Valid`)	`400`	`{"mensaje": "Datos invalidos.", "errores": {"campo": "detalle"}}`
Método no soportado	`405`	`{"mensaje": "Metodo no soportado."}`
Asset estático inexistente	`404`	`{"mensaje": "No encontrado."}`
Edición de identidad no permitida (ADR-028)	`400`	`{"mensaje": "No pudimos guardar esos datos."}`
Conflicto de concurrencia (`@Version`, edición simultánea)	`409`	`{"mensaje": "otra operacion modifico tus datos; volve a intentar"}`
Cualquier otro error interno	`500`	`{"mensaje": "Error procesando la solicitud"}` (detalle real solo en logs)

API del gestor (autenticada)

Todo bajo `/api/gestor/**`, protegido con **HTTP Basic** — usuario/password fijados por variable de entorno (`SP_GESTOR_USER`, `SP_GESTOR_PASSWORD`; el arranque falla rápido si falta la password, nunca genera una al vuelo). A diferencia de la API pública, los errores de negocio acá sí devuelven el motivo real (staff autenticado) — ver Errores del gestor.

Campañas

Método	Ruta	Para qué

GET	`/api/gestor/campanias/vigente`	Campaña vigente (404 si no hay ninguna)
GET	`/api/gestor/campanias/{id}`	Campaña por id
POST	`/api/gestor/campanias`	Alta de campaña
PUT	`/api/gestor/campanias/{id}`	Edición
POST	`/api/gestor/campanias/{id}/bases`	Sube el PDF de bases (multipart, `archivo` + `version` opcional) — versiona automáticamente, valida firma de bytes PDF y tamaño (≤10 MB), nunca sobrescribe una versión existente (inmutable, ADR-007)
GET	`/api/gestor/campanias/{id}/sorteos`	Lista los sorteos de la campaña
POST	`/api/gestor/campanias/{id}/sorteos/generar`	Genera la tanda completa de sorteos (semanales + cierre)
POST	`/api/gestor/campanias/{id}/sorteos`	Agrega un sorteo suelto a una campaña existente
POST	`/api/gestor/campanias/{id}/premios/repartir`	Reparte automáticamente el stock disponible en partes iguales entre sorteos, como punto de partida

```
// POST /api/gestor/campanias — request
{
  "nombre": "Sorteo 50 años",
  "descripcion": "...",
  "promo_id": 1,
  "estado": "vigente",
  "vigencia_desde": "2026-09-01T00:00:00",
  "vigencia_hasta": "2026-10-09T23:59:00",
  "regla_chances": 1,
  "tope_chances_dni_dia": 5,
  "plazo_retiro_dias": 10,
  "key_id_vigente": 1
}
```

```
// POST /api/gestor/campanias/{id}/sorteos/generar — request
{ "frecuencia": "semanal", "dia_hora": "MONDAY 10:00" }
```

Premios

Método	Ruta	Para qué
POST	`/api/gestor/premios/tipos`	Carga un `premio_tipo` (`{campania_id, descripcion, valor_unitario, cantidad}`)
GET	`/api/gestor/premios/tipos?campania_id=`	Lista tipos con `stock_disponible` calculado
GET	`/api/gestor/premios/unidades?campania_id=&estado=`	Lista unidades por estado (paginado, hasta 500)
POST	`/api/gestor/sorteos/{id}/premios`	Asigna una unidad a un sorteo, con prelación
GET	`/api/gestor/sorteos/{id}/asignaciones`	Lista las asignaciones de un sorteo
POST	`/api/gestor/asignaciones/{id}/desasignar`	Desasigna (`{motivo}`), body obligatorio — se cambió de `DELETE` a `POST` porque un `DELETE` con

body no tiene semántica garantizada en proxies/gateways

```
// POST /api/gestor/sorteos/{id}/premios — request
{ "premio_unidad_id": 501, "prelacion": 1 }
```

Sorteos

Método	Ruta	Para qué
PUT	`/api/gestor/sorteos/{id}`	Edita nombre / `fecha_hora_publicada` / `suplentes_a_sortear` de un sorteo `pendiente` (rechaza si ya está `ejecutado`; la fecha nueva no puede romper el orden monótono respecto al sorteo anterior/siguiente)
POST	`/api/gestor/sorteos/{id}/ejecutar`	**Dispara el sorteo** — siempre manual (ADR-009). Genera el acta

```
// POST /api/gestor/sorteos/{id}/ejecutar — response
{
  "acta_id": 12,
  "sorteo_id": 3,
  "fecha_ejecucion": "2026-09-07T10:00:05",
  "cantidad_participantes": 4231,
  "hash_lista": "e3b0c4..."
}
```

Actas

Método	Ruta	Para qué
GET	`/api/gestor/actas/{id}`	Detalle completo: sección inmutable (lo sorteado) + snapshot de estado de entrega
GET	`/api/gestor/actas/{id}/resultados`	Lista de resultados, cada uno con `plazo_vencido` / `dias_desde_contacto` calculados

GET	`/api/gestor/actas/{id}/export.json`	Igual a `ver`, pero con `Content-Disposition: attachment`
GET	`/api/gestor/actas/{id}/export.csv`	Export CSV: bloque de metadata del acta + tabla inmutable de resultados + tabla snapshot de estado de entrega

La respuesta separa siempre dos secciones: **acta** (inmutable — premio originalmente sorteado, `premio_unidad_id_original`) y **estado_entrega** (snapshot al momento de la consulta — `premio_unidad_id` actual, que muta con las operaciones de ganadores). Ver El motor de sorteo y el acta para el porqué de esta separación.

Ganadores y entregas

Método	Ruta	Para qué
POST	`/api/gestor/resultados/{id}/contactos`	Registra un intento de contacto (`{canal, resultado}`) — pasa el resultado de `pend_contacto` a `contactado` si era el primero
POST	`/api/gestor/resultados/{id}/entrega`	Registra la entrega física (`{sucursal, dni_verificado, confirmar_vencido}`) — verifica el DNI contra el acta; si el plazo de retiro venció, exige `confirmar_vencido: true` (409 sin él)
POST	`/api/gestor/resultados/{id}/pasar-suplente`	Pasa al siguiente suplente de la misma acta (`{motivo}`)
POST	`/api/gestor/unidades/{id}/devolver`	Devuelve la unidad al stock (`{motivo}`) — habilita reasignarla a un sorteo posterior
POST	`/api/gestor/unidades/{id}/no-otorgado`	Marca la unidad como `NO_OTORGADO`, estado terminal (`{motivo}`)

```
// POST /api/gestor/resultados/{id}/entrega — request
{ "sucursal": "Sucursal Centro", "dni_verificado": "30123456",
"confirmar_vencido": false }
```

Las cuatro operaciones que mutan `premio_unidad` (desasignar, pasar a suplente, devolver, no-otorgado) comparten el mismo contrato: `{"motivo": "..."}`, obligatorio — cada cambio queda auditado en `evento_premio` con usuario y motivo.

Clientes

Método	Ruta	Para qué
GET	`/api/gestor/clientes?q=&page=&size=`	Búsqueda por DNI/apellido/celular. `q` obligatorio, mínimo 2 caracteres (nunca devuelve el padrón completo sin filtro); paginado server-side, `size` tope 200
GET	`/api/gestor/clientes/{id}`	Detalle + historial de chances del cliente
PUT	`/api/gestor/clientes/{id}`	Corrección de nombre/apellido/celular/email (solo los campos presentes se aplican)

Chances y vouchers

Método	Ruta	Para qué
GET	`/api/gestor/chances?sucursal=&caja=&fecha=&top=`	Rastreo de chances con filtros opcionales; `top` acotado (default 200, máx. 1000)
GET	`/api/gestor/vouchers/{hash}`	Trazabilidad de un voucher por su `hash_token`: la chance que registró (si existe) + rechazos previos con ese mismo hash

Claves de firma

Método	Ruta	Para qué
GET	`/api/gestor/claves`	Lista metadatos de claves — **nunca** el secreto, ni siquiera cifrado
POST	`/api/gestor/claves`	Alta (`{key_id, secreto}`) — el secreto en claro solo viaja en este body, se cifra antes de persistir
POST	`/api/gestor/claves/{keyId}/revocar`	Revoca (`{motivo}`) — 409 si ya está revocada o si es la clave vigente de la campaña vigente (hay que rotar la campaña primero)

```
// POST /api/gestor/claves — request
{ "key_id": 2, "secreto": "..." }
```

Ver ClaveFirma en el modelo de datos y rotación de claves en el Anexo A.

Modo prueba

Solo existe si `sp.modo-prueba.enabled=true` (el bean ni se registra si el flag está apagado — la ruta responde **404**, nunca 403, para que la UI distinga "no habilitado" de "sin permiso"). **Nunca habilitable en producción.**

Método	Ruta	Para qué
GET	`/api/gestor/modo-prueba/estado`	`{"enabled": true}` — la UI lo usa para decidir si muestra el banner
POST	`/api/gestor/modo-prueba/vouchers`	Genera N vouchers de prueba firmados (`{campania_id, nro_sucursal, nro_pos, fecha_emision, cantidad}`)
GET	`/api/gestor/modo-prueba/vouchers?campania_id=`	Auditoría de vouchers de prueba generados

Panel

Método	Ruta	Para qué
GET	`/api/gestor/panel`	Métricas de un vistazo: participación, stock, señales antifraude

Export CSV

Método	Ruta	Para qué
GET	`/api/gestor/export/padron.csv`	Padrón completo: `dni,nombre,apellido,celular,email,origen,creado_en`
GET	`/api/gestor/export/pendientes-contacto.csv`	Ganadores/suplentes pendientes de contacto, con celular ya normalizado a 10 dígitos

Errores del gestor

A diferencia de la API pública, acá el operador **sí** recibe el motivo real de un error de negocio — es staff autenticado, no el cliente final.

Situación	HTTP	Body
Argumento inválido (`IllegalArgumentException`)	`400`	`{"mensaje": "<motivo real>"}`
Estado inválido (`IllegalStateException`) — ej. sorteo ya ejecutado, clave ya revocada	`409`	`{"mensaje": "<motivo real>"}`
Conflicto de concurrencia (`@Version` o deadlock detectado)	`409`	`{"mensaje": "otra operacion modifico el premio; reintenta"}`
Violación de integridad (UNIQUE/CHECK/FK)	`409`	`{"mensaje": "conflicto de integridad de datos"}` (sin

		nombrar una causa específica que puede no aplicar)
Archivo de bases demasiado grande	`413`	`{"mensaje": "el archivo supera el tamano maximo permitido (10 MB)"}`

Ver también

- Modelo de datos — las tablas detrás de cada recurso.
- El token del QR (Anexo A) — el `token` que reciben `resolver` / `confirmar` / `alta` / `padron/alta`.
- Casos de participación (A–E) — el flujo de negocio completo detrás de la API pública.
- El motor de sorteo y el acta — el flujo detrás de `ejecutar` y las actas.

Backend

El backend es un servicio **Spring Boot** que concentra todo el dominio del Sistema de Premios: valida vouchers, administra el padrón de clientes, corre el motor de sorteo, genera el acta reproducible y expone la API que consumen la landing y el gestor. Es, además, quien **empaqueta y sirve** esos dos frontends — no hay servidor web separado.

Esta página es la referencia del backend como componente: stack, estructura de paquetes, arranque, seguridad y build. Para el detalle de configuración operativa (variables de entorno, perfiles), ver Configuración.

Stack

Tecnología	Versión	Para qué
Java	17 (`maven.compiler.release`)	Lenguaje base
Spring Boot	4.1.1	Framework de aplicación (última GA al momento del upgrade, ver ADR-018)
Tomcat	11 (embebido, gestionado por Boot 4)	Servidor HTTP
Hibernate	7 (gestionado por Boot 4)	ORM, `ddl-auto: validate`
Jackson	3 (paquete `tools.jackson.*`)	Serialización JSON, `SNAKE_CASE`
SQL Server	2022	Base de datos, driver `mssql-jdbc`
Flyway	módulo `flyway-sqlserver`	Migraciones versionadas, corren solas al boot

Maven	—	Build, `mvn package`

> ⚠ **Jackson 3 no es `com.fasterxml.jackson.*`**
>
> Spring Boot 4 trae Jackson 3, que renombra `jackson-core` / `jackson-databind` al paquete `tools.jackson.*`. Las *anotaciones* (`com.fasterxml.jackson.annotation.*`) no cambiaron. `JsonProcessingException` (checked) desaparece a favor de `JacksonException` (unchecked). Si copiás un snippet viejo con `import com.fasterxml.jackson.databind.ObjectMapper`, no compila — el import correcto es `tools.jackson.databind.ObjectMapper`. Detalle completo en ADR-018.

Estructura de paquetes

Todo vive bajo `com.tipre.sistemapremios`:

```
backend/src/main/java/com/tipre/sistemapremios/
├── SistemaPremiosApplication.java   # entrypoint @SpringBootApplication
├── adjudicacion/                    # asignación de premios a ganadores/suplentes
├── api/                             # controllers públicos + filtros HTTP
│   ├── dto/                         # DTOs de la API pública (snake_case)
│   └── gestor/                      # controllers del gestor (autenticados) + sus
DTOs
├── clavefirma/                      # cifrado/descifrado del secreto HMAC
(ClaveFirma)
├── config/                          # configuración de Flyway por perfil (e2e,
load)
├── dominio/                         # entidades JPA (Campania, Cliente, Sorteo,
Chance, ...)
├── modoprueba/                      # generador de vouchers de prueba (solo
gestor, nunca prod)
├── participacion/                   # servicio central: resolver/confirmar/alta,
reconocimiento
├── premio/                          # PremioUnidad y su ciclo de estado
├── qrtoken/                         # espejo Java del token firmado (compartido
con el POS)
├── repositorio/                     # Spring Data JPA repositories
├── soporte/                         # arranque: config externa, seed, avisos de
env vars legacy
└── sorteo/                          # motor de sorteo + generación de acta
```

`api/` concentra toda la superficie HTTP. Lo que distingue a `api/gestor/` del resto no es la ubicación en disco sino el prefijo de ruta `/api/gestor/**`, que es lo que `GestorSecurityConfig` usa para exigir autenticación.

> **`qrtoken` es un módulo vendoreado, no reimplementado**
>
> El backend no reinventa la validación del token del QR: usa el mismo módulo que corre en el POS (espejo Java de un módulo C89). Ver El token del QR para el formato del payload y el flujo de firma/validación.

Cómo arranca

El entrypoint es mínimo — toda la lógica de arranque vive en `soporte/` y en la configuración de Spring Boot:

```
@SpringBootApplication
public class SistemaPremiosApplication {
    public static void main(String[] args) {
        SpringApplication.run(SistemaPremiosApplication.class, args);
    }
}
```

Secuencia real de arranque:

1. **Chequeo de config externa** (`ConfigExternaEnvironmentPostProcessor`, corre antes que cualquier bean) — exige que `./config/application.yml` exista y traiga el centinela `sp.config.externa: true`, más las claves de contrato (`SNAKE_CASE`, `ddl-auto: validate`). Sin eso, el proceso ni intenta levantar el contexto. Ver Configuración → config externa.
2. **Flyway migra la base automáticamente.** No hay paso manual: al conectar, Flyway aplica cualquier migración pendiente contra `SistemaPremios` antes de que el contexto termine de levantar.
3. **Hibernate valida el esquema** contra las entidades (`ddl-auto: validate`) — nunca lo altera. Si el esquema real no coincide con lo que las entidades esperan, el arranque falla ahí, no en producción silenciosamente.
4. **Beans de seguridad y guardas de arranque fail-fast** — `GestorSecurityConfig` exige `SP_GESTOR_PASSWORD`, `ClaveFirmaConfig` exige `SP_MASTER_KEY`, `ModoPruebaConfig` corta el arranque si `sp.modo-prueba.enabled=true` y el perfil activo contiene `prod`.
5. **`SeedMinimo` (opcional)** — solo si se pasa `--sp.seed.archivo=<ruta.json>`, carga campaña + sorteos + clave de firma desde un JSON, idempotente. Es el fallback operativo si el gestor no llega a tiempo para cargar la campaña a mano.

Seguridad

Spring Security cubre solo lo que necesita quedar detrás de credenciales; el resto de la API pública queda `permitAll` a propósito (participación, padrón, bases, campaña vigente, salud, branding — la landing es anónima por diseño).

```
.authorizeHttpRequests(auth -> auth
    .requestMatchers("/api/participacion/**", "/api/padron/**", "/bases/**",
            "/api/campania/vigente", "/api/salud", "/api/branding",
"/branding/**")
    .permitAll()
    .requestMatchers("/actuator/health").permitAll()
    .requestMatchers("/actuator/**").authenticated()
    .requestMatchers("/api/gestor/**").authenticated()
    .anyRequest().permitAll())
.httpBasic(Customizer.withDefaults())
```

- **`/api/gestor/**` exige HTTP Basic**, sin sesión (`SessionCreationPolicy.STATELESS`) — el gestor manda el header `Authorization` en cada request, nunca una cookie.
- **Usuario y password del gestor vienen de `SP_GESTOR_USER` / `SP_GESTOR_PASSWORD`**, nunca hardcodeados. Si falta `SP_GESTOR_PASSWORD`, el arranque falla con `IllegalStateException` — a propósito. La versión anterior generaba una password aleatoria y la logueaba en texto claro para que el operador la leyera; eso viola la regla de que un secreto nunca aparece en logs, así que se sacó: sin credencial, no arranca.
- **`/actuator/**` queda cerrado salvo `/actuator/health`**, y ese health no muestra detalle para un caller anónimo (`show-details: when-authorized`). `actuator` llega transitivo con la consola de desarrollo `bootui` (ver ADR-018) — sin esta restricción explícita, `/actuator/env`, `configprops`, `beans`, `loggers`, `mappings` y `conditions` quedaban alcanzables sin autenticación, volcando variables de entorno completas.
- **CSP estricta**, con dos excepciones angostas y documentadas: `'wasm-unsafe-eval'` en `script-src` (necesario para compilar el WebAssembly del escáner de DNI, no habilita `eval()` de JS arbitrario) y `camera=(self)` en Permissions-Policy (la landing usa la cámara para escanear).

Rate limiting

Filtro propio, sin dependencias nuevas — mitigación mínima contra abuso (fuerza bruta de tokens, scraping), no un sustituto de un WAF real.

Config	Default	Qué hace
`sp.rate-limit.enabled`	`true`	apagado en el perfil de test para no disparar 429 espurios contra bursts legítimos de la suite
`sp.rate-limit.limite`	`30`	requests permitidos por ventana
`sp.rate-limit.ventana-ms`	`60000`	tamaño de la ventana (60 s)
`sp.rate-limit.cap-entradas`	`10000`	tope de IPs trackeadas antes de purgar entradas vencidas

Cubre `/api/**` y también `/p/**` (el estado inicial de la landing se resuelve server-side ahí, así que un scan de tokens contra `/p/{token}` pega tan fuerte como uno directo a `POST /api/participacion/resolver`).

> ⚠ **El filtro se registra explícito, ANTES de Security**
>
> `RateLimitFilterConfig` lo registra con `Ordered.HIGHEST_PRECEDENCE`, a propósito. Si quedara con el orden por defecto de Spring Boot, correría **después** de la cadena de seguridad: un intento con credenciales inválidas contra `/api/gestor/**` nunca llegaría a contarse para el rate limit, y un atacante podría probar passwords sin freno.

> ⚡ **Detrás de un reverse proxy, `X-Forwarded-For` tiene que estar sanitizado**
>
> Sin proxy, `request.getRemoteAddr()` ya es la IP real. Con un proxy delante (Caddy en el runbook de despliegue), el proxy **debe** pisar el `X-Forwarded-For` entrante con la IP real del cliente — si no, cualquiera puede falsificar su IP en el header y esquivar el rate limit. `forward-headers-strategy: framework` ya está activado en `application.yml` para que Spring resuelva `getRemoteAddr()` correctamente detrás del proxy.

Modo prueba

Generador de vouchers de prueba (con su propio QR) invocado desde el gestor, para poder probar el flujo completo sin imprimir tickets reales. Vive detrás de dos guardas independientes:

1. **`sp.modo-prueba.enabled`** (default `false`): si la feature no está prendida, no existe (404), no solo está "escondida".
2. **Fail-fast contra prod**: si `sp.modo-prueba.enabled=true` **y** el perfil activo contiene `prod`, el contexto no levanta. No hay combinación posible de "modo prueba en producción".

Seed inicial (`SeedMinimo`)

Fallback para cargar la campaña sin depender de que el gestor esté listo. Se activa **solo** con `--sp.seed.archivo=<ruta.json>` (nunca por default) y es idempotente: correrlo dos veces sobre el mismo archivo no duplica campaña, sorteos ni clave de firma.

```
java -jar sistemapremios-backend.jar --sp.seed.archivo=seed-minimo.json
```

El archivo trae `campania`, `sorteos[]` y `clave_firma` (con el secreto en texto plano, **nunca** committeado — ver `docs/seed-minimo.ejemplo.json` en el repo del proyecto para el formato). Antes de escribir nada valida que `campania.key_id_vigente` coincida con `clave_firma.key_id`: un typo ahí dejaría la campaña firmando con una clave que no es la declarada, y se corta antes de tocar la base.

Cifrado de secretos at-rest

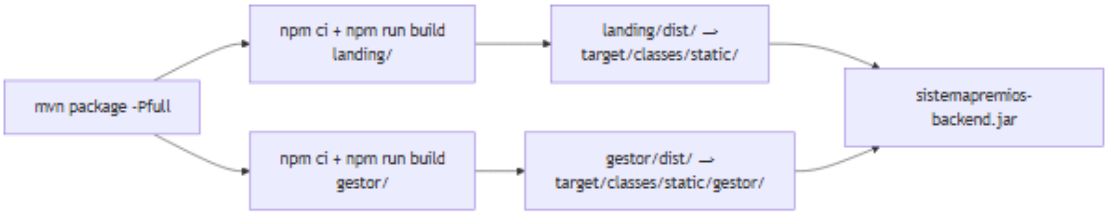
El secreto HMAC que firma los tokens del QR (`clave_firma.secreto_cifrado`) nunca se persiste en claro. `CifradorSecretos` cifra con **AES-256-GCM**, usando una master key de 32 bytes que llega por `SP_MASTER_KEY` (Base64) y nunca se guarda en el repo.

- Formato: `IV(12 bytes) || ciphertext+tag(GCM)` — IV aleatorio en cada cifrado, nunca reusado.
- Sin `SP_MASTER_KEY` configurada, el bean de `CifradorSecretos` ni se registra; en su lugar se registra un bean que falla con un mensaje explícito (`SP_MASTER_KEY no configurada`) en vez del genérico `NoSuchBeanDefinitionException` que daría Spring por defecto.
- Un `secreto_cifrado` que no autentica (master key incorrecta o dato alterado) lanza `IllegalStateException` al descifrar — nunca decodifica basura en silencio.

Build del jar único

El perfil Maven `full` (activo por defecto — se desactiva con `-DskipFrontends=true`) bundlea la landing y el gestor **dentro** del jar del backend, vía `frontend-maven-plugin`:

```
cd backend
mvn package -Pfull
```



Las ejecuciones de npm y el copy de los `dist/` están atadas a la fase `prepare-package` (**después** de `test`), así que `mvn test` del día a día nunca paga el costo de construir los dos frontends. Node se descarga localmente (`v20.18.1`, vía `frontend-maven-plugin`), no hace falta tenerlo instalado en la máquina de build.

El resultado es un único artefacto: `backend/target/sistemapremios-backend-0.0.1-SNAPSHOT.jar`, que sirve la API, la landing (`/`, assets en `/assets/**`) y el gestor (`/gestor/`, assets en `/gestor/assets/**`) — ver Instalar en el servidor para el despliegue real.

QA y análisis estático

Perfil Maven `qa` (atado a `verify`, nunca a `test`, para no encarecer el ciclo diario): checkstyle + PMD + SpotBugs (con `findsecbugs`). Ninguno de los tres rompe el build por sí mismo (`failOnViolation` / `failOnError` en `false`) — el gate real lo aplica el pipeline de calidad del proyecto sobre los reportes XML que generan.

`jacoco-maven-plugin` (0.8.15) instrumenta la suite en cada `mvn test` y emite el reporte XML de cobertura.

Dependencias notables

`spring-boot-starter-data-jpa`	Persistencia (Hibernate 7)
`spring-boot-starter-validation`	Bean Validation en DTOs de entrada
`spring-boot-starter-security`	HTTP Basic para `/api/gestor/**`
`mssql-jdbc`	Driver SQL Server (runtime)
`bootui-spring-boot-starter` (1.14.1)	Consola de desarrollo (introspección salud/JPA/seguridad); solo activa en `dev` / `local`, rechaza requests no-loopback — ver ADR-018
`spring-boot-starter-flyway` + `flyway-sqlserver`	Migraciones versionadas
`com.google.zxing:core` / `javase` (3.5.3)	Genera el PNG del QR de los vouchers de prueba (solo `ModoPruebaService`, nunca en el camino de producción real)
`spring-boot-starter-webmvc-test` (test)	Soporte de `MockMvc` (módulo separado desde Boot 4)

Landing

La landing es la web del QR: la página anónima que se abre cuando un cliente escanea el voucher que imprime la caja. Es de un solo uso por voucher, pensada para cargar rápido en el **3G de salón** — la audiencia real entra casi siempre desde el celular. No tiene "home": solo se llega por `/p/{token}`.

Para el backend que la sirve y valida los tokens, ver Backend. Para el formato del token del QR, ver El token del QR.

Stack

Tecnología	Versión	Para qué
Preact (vía `preact/compat`)	^10.29	Runtime de UI — reemplaza a React solo acá, por peso (ver abajo)
Vite	^5.4	Build
`@preact/preset-vite`	^2.10	Plugin de build — alias `react` / `react-dom` → `preact/compat` en build time
`zxing-wasm`	^3.1	Decodificador PDF417 del DNI (carga diferida, fuera del bundle inicial)
`vite-plugin-compression2`	^2.5	Precomprime `.br` / `.gz` en build time
TypeScript	^5.6	Tipado

Por qué Preact y no React

El código fuente está escrito contra la API de React (hooks, `Component` de clase para el `ErrorBoundary`, etc.) y no cambió: `@preact/preset-vite` alias `react` / `react-dom` a `preact/compat` / `preact/jsx-runtime` en build time, sin tocar un import. El motivo es peso: el runtime de React (`react` + `react-dom`) representaba ~45 KB de los ~51 KB gzip del bundle inicial — la mayor parte no era código propio, era el framework. Preact pesa ~4 KB. Detalle completo, alternativas consideradas y riesgos de compatibilidad evaluados en ADR-019.

El gestor se queda en React 18 sin cambios — es una herramienta interna sin presupuesto de peso.

Pantallas

La landing es un único componente `App.tsx` con un `switch` sobre el nombre de la pantalla — sin router (no hace falta: solo hay una ruta de entrada, `/p/{token}`, y las transiciones son estados internos, no URLs distintas).

Pantalla	Cuándo aparece
Reconocido	El dispositivo ya tiene un secreto de reconocimiento guardado (ver más abajo) y el voucher todavía no está resuelto — saludo + confirmar de un toque
1a — Pedí tu DNI	Primera pantalla del flujo clásico: pide el DNI (a mano o escaneado)
1b — Alta	El DNI escaneado/tipeado no está en el padrón — formulario de alta (nombre, apellido, contacto, aceptación de bases)
2 — Confirmar	El DNI ya está en el padrón — confirma la participación con un toque, mostrando el nombre y (si aplica) permite completar/corregir email o celular
Éxito	Chance sumada — muestra el total acumulado
3 — Ya participó	Ese voucher específico ya se usó — muestra la fecha del registro original

4a — Padrón, fuera de vigencia	El voucher es válido pero la campaña ya no está vigente — ofrece sumarse solo al padrón
4b — Gracias	Cierre del alta al padrón (4a)
Error	Rechazo definitivo (voucher inválido, honeypot de bot, etc.) — mensaje genérico, nunca el motivo real
Error de red	Fallo de conexión (no de servidor) — pantalla de reintentar, con backoff
Tope	El cliente alcanzó el límite diario de participaciones por DNI

Estos nombres de pantalla siguen los "casos" que devuelve el backend (`A` – `E`, `PENDIENTE_DNI`, `TOPE`) — `estadoDesdeResolucion()` en `App.tsx` es la traducción caso → pantalla.

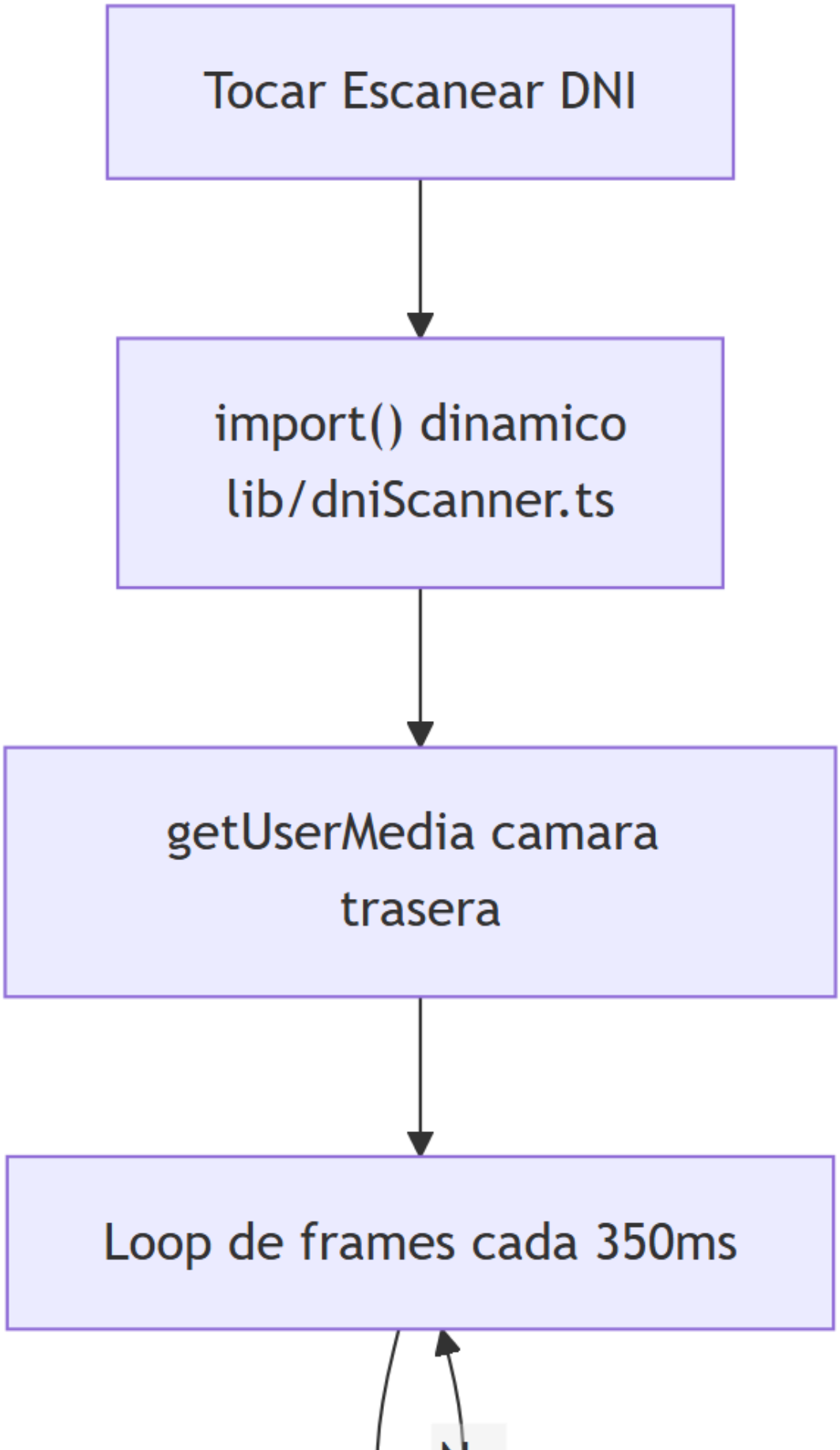
Escaneo de DNI (PDF417)

El botón "Escanear DNI" activa la cámara y decodifica el código de barras PDF417 del reverso del DNI argentino, autocompletando nombre/apellido/sexo/fecha de nacimiento en el alta.

La primera implementación usaba `BarcodeDetector` (Shape Detection API del navegador), pero una prueba en dispositivo real (Chrome de Android, el navegador más común del público objetivo) mostró que **no soporta PDF417** sin un módulo extra de Google Play Services — el botón de escaneo prácticamente no aparecía para la mayoría de la audiencia real. Por eso el sistema trae su **propio decodificador PDF417** (`zxing-wasm`), independiente del navegador:

- Funciona en cualquier navegador con `getUserMedia` (cámara), no solo donde hay `BarcodeDetector`.
- **Carga diferida**: el módulo (WASM, pesado) entra vía `import()` dinámico recién al tocar "Escanear" — nunca forma parte del bundle inicial.
- El `.wasm` se sirve desde el propio origen (no el CDN por defecto de `zxing-wasm`), porque el salón donde se escanea puede tener conectividad inestable.
- **Charset**: el payload del DNI argentino viene en Latin-1 (ISO-8859-1), no UTF-8. El decodificador expone los bytes crudos y la landing los reinterpreta explícitamente como Latin-1, así apellidos con Ñ o acentos (p. ej. "PEÑA") llegan correctos.

Detalle de la decisión y las alternativas descartadas en ADR-026.


Tocar Escanear DNI
import() dinamico
lib/dniScanner.ts
getUserMedia camara
trasera
Loop de frames cada 350ms

Reconocimiento de dispositivo

En la segunda participación desde el mismo teléfono, la landing puede saludar por nombre y ofrecer confirmar de un toque, sin re-pedir el DNI. El mecanismo es un **secreto opaco**, nunca el DNI:

- Al consumir una chance (alta o confirmar), el backend genera un secreto aleatorio (≥128 bits) y lo devuelve **una sola vez**; la landing lo guarda en `localStorage` junto al primer nombre — `{ secreto, nombre }`, nada más.
- El DNI **nunca** toca el navegador, ni en claro ni hasheado (un DNI son ~90 millones de valores posibles; un hash se revierte por fuerza bruta con la propia app).
- **"No soy yo"** borra el secreto guardado y cae al flujo clásico por DNI — protege el caso de un teléfono compartido (la caja, un familiar).
- Reconocer nunca participa solo: siempre exige el toque explícito de "Confirmar" además de un voucher vigente.

Persistencia en `lib/reconocimiento.ts` (`localStorage`, no `sessionStorage`: el secreto es permanente y cruza campañas). Configurable por cliente vía `sp.reconocimiento.habilitado` — con la feature apagada, la landing pide siempre el DNI. Detalle completo en ADR-027.

Cómo apunta al backend

```
const BASE = import.meta.env.VITE_API_BASE ?? "";
```

- **En producción**: `BASE` queda vacío — todas las llamadas son **same-origin**, porque el backend sirve la landing y la API desde el mismo jar (ver Backend → build del jar único). No hace falta CORS.
- **En desarrollo** (`vite dev`, puerto `5173`): el proxy de Vite reenvía `/api`, `/bases` y `/branding` a `http://localhost:8080` (el backend corriendo aparte) — mismo comportamiento same-origin desde el punto de vista del navegador, sin abrir CORS en el backend.
- `VITE_API_BASE` permite apuntar explícitamente a otro host si hiciera falta (no se usa en el flujo normal de dev ni de prod).

Presupuesto de peso

La Pantalla 1a tiene que pintar en menos de 1.5 s bajo el perfil "3G lento" de los tests E2E — la mayoría del público entra desde el celular. El presupuesto es **60 KB gzip** para el bundle inicial (`AC-24`), medido automáticamente (`npm run build:check`) sobre los `<script>` / `<link>` que `dist/index.html` referencia directo — los chunks que solo se alcanzan por `import()` dinámico (como el escáner de DNI) quedan **fuera** del presupuesto a propósito, porque no bloquean el primer render.

Con Preact + estado inicial embebido en el HTML (el backend inyecta `{token, campania, resolucion}` directo en `/p/{token}` , evitando dos viajes de red antes del primer render), el bundle inicial mide **14.29 KB gzip / 12.89 KB brotli** — bien por debajo del límite.

> **Compresión**
>
> `vite-plugin-compression2` genera `.br` / `.gz` en build time; el backend los sirve tal cual vía `EncodedResourceResolver` cuando el navegador los acepta, sin gastar CPU comprimiendo al vuelo en cada request.

Gestor

El gestor es el panel de administración interno del Sistema de Premios: donde el operador de Caracol carga la campaña, arma y ejecuta los sorteos, gestiona ganadores y entregas, administra el padrón de clientes y consulta reportes. A diferencia de la landing, no es anónimo — requiere login — y no tiene presupuesto de peso: es una herramienta de uso interno, no una página que un cliente abre desde un QR.

Se sirve bajo `/gestor/`, empaquetado dentro del mismo jar que el backend (ver Backend → build del jar único).

Stack

Tecnología	Versión	Para qué
React	18.3	Runtime de UI (sin cambios respecto al stack original — ver ADR-019, que aclara por qué el gestor *no* migró a Preact)
Vite	^5.4	Build
Tailwind CSS	^3.4	Estilos, con componentes propios en el estilo shadcn (`components/ui/`)
React Router	^6.26	Ruteo del lado del cliente, `basename="/gestor"`
Zustand	^5.0	Store de autenticación
TanStack Query	^5.59	Fetching/caching contra la API del gestor
Axios	^1.7	Cliente HTTP, con interceptors de auth y manejo de 401
React Hook Form + Zod	^7.53 / ^3.23	Formularios y validación

`vite-plugin-compression2`	^2.5	Precomprime `.br` / `.gz` en build (mismo tratamiento que la landing)

PWA

El gestor es instalable como **Progressive Web App** — pensado para la tarea genuinamente móvil del panel: registrar la entrega de un premio en sucursal desde el celular. El resto del gestor sigue siendo mayormente de escritorio, pero el sidebar colapsa a un drawer por debajo de 768px y las tablas scrollean dentro de su propio contenedor (nunca el body).

Piezas de la PWA (`public/manifest.webmanifest`, `public/sw.js`, íconos 192/512 + maskable):

- **Manifest**: nombre "Sistema Premios", `scope` / `start_url` = `/gestor/`, `display: standalone`, theme color = rojo de branding.
- **Service worker mínimo, a mano** (sin Workbox ni `vite-plugin-pwa` — sin librería nueva): precachea el shell de la app con *stale-while-revalidate*, pero **nunca** cachea respuestas de `/api/**` ni `/bases/**` — el gestor es una herramienta viva, sus datos siempre tienen que salir frescos de la red. El predicado que decide esto (`debeUsarSoloRed`) es una función pura, testeada evaluando el propio archivo del service worker (no una copia en TypeScript que se pueda desincronizar).
- El backend sirve `manifest.webmanifest` y `sw.js` con un controller dedicado (`GestorPwaController`), no por el resource handler estático genérico, porque necesita agregar el header `Service-Worker-Allowed: /gestor/` por request.

Detalle de la decisión (responsive + PWA, sin abrir un segundo codebase) en ADR-022.

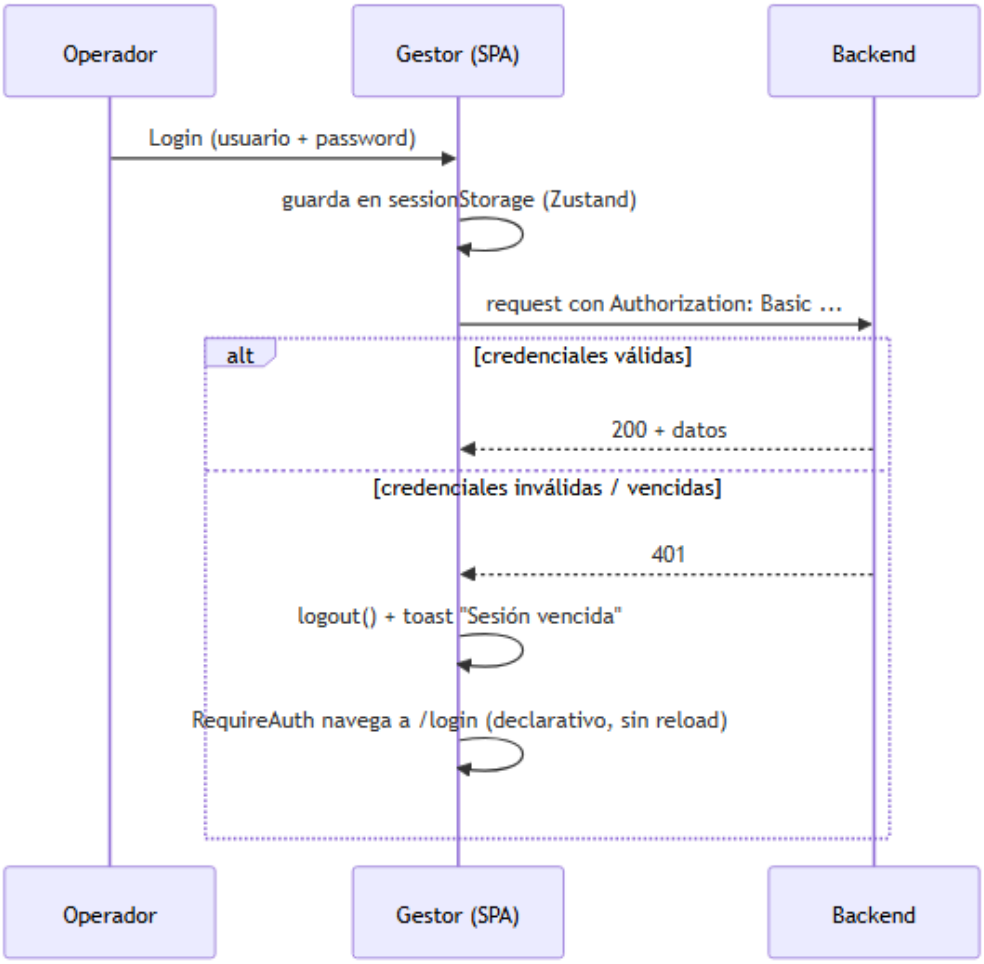
Autenticación

HTTP Basic contra `/api/gestor/**` (ver Backend → seguridad). El gestor no usa cookies ni sesión — cada request lleva el header `Authorization` armado en el cliente:

```
export function authHeader(credenciales: Credenciales | null): string | null {
  if (!credenciales) return null;
  return "Basic " + btoa(`${credenciales.usuario}:${credenciales.password}`);
}
```

- Las credenciales viven en un store de **Zustand** (`store/authStore.ts`), persistidas en **`sessionStorage`** — a propósito, no `localStorage`: se pierden al cerrar la pestaña, no quedan indefinidamente en el disco del puesto de trabajo.

- Un interceptor de request de Axios agrega el header `Authorization` a cada llamada.
- Un interceptor de response detecta cualquier `401`, desloguea (limpia el store) y muestra un toast de "Sesión vencida" — con un guard para no disparar el toast duplicado si llegan varios 401 concurrentes de una ráfaga de requests en vuelo.
- El redirect a `/login` es **declarativo**, no un `location.assign`: `RequireAuth` lee el store y renderiza `<Navigate to="/login">` en cuanto `credenciales` queda `null` — un reload duro destruiría el toast recién mostrado antes de que React lo pinte.



Secciones y rutas

Todas las rutas cuelgan de `basename="/gestor"` (React Router) y quedan detrás de `RequireAuth`, salvo `/login`:

Ruta	Sección
`/panel`	Panel — resumen operativo de la campaña
`/campania`	Datos de la campaña (vigencia, bases, reglas de chances)
`/premios`	Alta y administración de premios
`/sorteos`	Generación y ejecución de sorteos
`/sorteos/actas/:id`	Detalle del acta de un sorteo
`/ganadores/:actaId`	Ganadores de un acta — gestión de entrega
`/clientes`	Padrón de clientes
`/clientes/:id`	Detalle de un cliente
`/chances`	Consulta de chances (soporte/trazabilidad)
`/claves`	Gestión de claves de firma (`ClaveFirma`)
`/modo-prueba`	Generador de vouchers de prueba (ver Backend → modo prueba)

Cualquier ruta no reconocida (`*`) y la raíz (`/`) redirigen a `/panel` .

Cómo apunta al backend

Misma lógica que la landing: en producción, same-origin (`VITE_API_BASE_URL` ?? `/api` , servido por el mismo jar bajo `/gestor/`); en desarrollo (`vite dev` , puerto `5174`), el proxy de Vite reenvía `/api` y `/branding` a `http://localhost:8080` .

Configuración

Toda la configuración operativa del backend vive **fuera del jar**, en un `application.yml` junto al ejecutable. Esta página documenta cada variable/clave real, de dónde sale y qué pasa si falta. Para el detalle del proceso de arranque que la exige, ver Backend → cómo arranca. Para el despliegue paso a paso, ver Instalar en el servidor.

Config externa al jar y fail-fast

Decisión explícita del operador: *"el yaml de configuración por completo tiene que estar fuera del .jar, nada de sorpresas con yaml adentro del big fat jar"*. El jar empaquetado **no contiene** ningún `application*.yml` / `application*.properties` — Spring Boot busca `./config/` relativo al **directorio de trabajo del proceso**, sin ningún loader custom.

```
C:\work\sistemapremios\
├── sistemapremios-backend-0.0.1-SNAPSHOT.jar
├── config\
│   └── application.yml      ← toda la config real vive acá
├── bases\
├── branding\
│   └── logo.png
└── logs\
```

El repo versiona `backend/config/application.yml` como **referencia documentada** — resuelve dev y CI sin fricción (ambos corren con working directory `backend/`), y nunca contiene un valor secreto: los `${SP_*}` quedan sin resolver hasta que el entorno real los provee.

Sin `config/application.yml`, el proceso no arranca — a propósito

Un `EnvironmentPostProcessor` (`ConfigExternaEnvironmentPostProcessor`) corre antes que cualquier bean y exige:

1. La propiedad centinela `sp.config.externa: true` — solo existe en el archivo externo, nunca embebida. Si falta, la excepción nombra el archivo esperado, el working directory real del proceso y sugiere `--spring.config.additional-location=file:<ruta>/config/` si no se puede controlar el working directory.
2. Dos claves de contrato del código, con su valor exacto: `spring.jackson.property-naming-strategy: SNAKE_CASE` (si falta, toda la API cambia de formato y la landing/gestor rompen en silencio) y `spring.jpa.hibernate.ddl-auto: validate` (si falta, Hibernate podría alterar el esquema en vez de solo validarlo contra Flyway).

Nunca arranca con defaults embebidos. Mejor no arrancar que arrancar distinto a lo esperado. Detalle completo de la decisión en ADR-029.

`AppDirectory` / working directory es crítico en Windows Service

Spring busca `./config/` relativo al **working directory del proceso**, no a la ubicación del jar. Con NSSM, esto significa que `AppDirectory` tiene que apuntar a la carpeta que contiene `config/` — un desajuste típico de "Start in" deja el servicio sin encontrar la config aunque el jar esté en la ruta correcta.

Variables de entorno

Variable	Default	Obligatoria	Para qué
`SP_DB_URL`	`jdbc:sqlserver://localhost;databaseName=SistemaPremios;encrypt=false`	No	URL JDBC de SQL Server
`SP_DB_USER`	`sa`	No	Usuario de la base

`SP_DB_PASSWORD`	— (fallback a `SP_DB_PASS`)	**Sí**, en la práctica	Password de la base. `SP_DB_PASS` es el nombre viejo, obsoleto — sigue funcionando de fallback pero loguea un `WARN`
`SP_MASTER_KEY`	—	**Sí**	Base64 de 32 bytes (AES-256) para cifrar/descifrar el secreto HMAC de la clave de firma. Sin ella, el arranque falla con un mensaje explícito. Debe ser la **misma** clave usada al sembrar los datos, o los QR emitidos antes no vuelven a validar
`SP_GESTOR_USER`	`admin`	No	Usuario de login del gestor
`SP_GESTOR_PASSWORD`	— (fallback a `SP_GESTOR_PASS`)	**Sí**	Password del gestor. Sin ella, el arranque **falla rápido** — no hay camino de "arrancar igual y avisar" (la versión anterior generaba y logueaba una password aleatoria; eso viola la regla de secretos nunca en logs)
`SP_CLIENTE_NOMBRE`	`Cliente`	No	Nombre del cliente mostrado en landing/gestor (branding multi-cliente)
`SP_CLIENTE_COLOR`	`#E01020`	No	Color primario de marca, formato `#RRGGBB` estricto — un valor inválido hace fallar el arranque
`SP_BRANDING_DIR`	`./branding`	No	Carpeta donde vive `logo.png` del cliente

`SP_BASES_DIR`	—	No	Carpeta de las bases legales versionadas de la campaña (PDF subido desde el gestor)
`SP_MODO_PRUEBA`	`false`	No	Habilita el generador de vouchers de prueba. Ver guarda de prod más abajo
`SP_BASE_URL`	`http://localhost:8080`	**Sí**, en producción	URL pública que se **embebe dentro del contenido del QR**. Si queda en `localhost`, los QR generados apuntan a `localhost` y no sirven fuera de esa máquina

Las tres que muerden si las salteás

`SP_GESTOR_PASSWORD` (login del gestor — sin ella no arranca) · `SP_BASE_URL` (si queda en `localhost`, los QR apuntan a `localhost`) · `SP_MASTER_KEY` (debe ser la misma que se usó al sembrar los datos, o los QR no validan).

Claves de config (`sp.*`)

Estas viven en `application.yml`, no como variable de entorno directa (aunque casi todas se resuelven a partir de las de arriba vía `${SP_*:default}`):

Clave	Default	Para qué
`sp.config.externa`	—	Centinela obligatorio, ver arriba. No borrar.
`sp.cliente.nombre`	`${SP_CLIENTE_NOMBRE:Cliente}`	Branding
`sp.cliente.color-primario`	`${SP_CLIENTE_COLOR:#E01020}`	Branding, validado `#RRGGBB`

`sp.branding.dir`	`${SP_BRANDING_DIR:./branding}`	Carpeta del logo
`sp.modo-prueba.enabled`	`${SP_MODO_PRUEBA:false}`	Ver Backend → modo prueba
`sp.base-url`	`${SP_BASE_URL:http://localhost:8080}`	Base pública embebida en el QR
`sp.reconocimiento.habilitado`	`true`	Perilla de privacidad por cliente para el reconocimiento de dispositivo — en `false`, la landing pide siempre el DNI
`sp.rate-limit.enabled`	`true`	Ver Backend → rate limiting
`sp.rate-limit.limite`	`30`	Requests por ventana
`sp.rate-limit.ventana-ms`	`60000`	Tamaño de la ventana (ms)
`sp.rate-limit.cap-entradas`	`10000`	Tope de IPs trackeadas antes de purgar
`sp.seed.archivo`	—	Ver Backend → seed inicial. Sin esta property, `SeedMinimo` ni se registra como bean

Cambiar un flag operativo no reempaqueta nada

Por ejemplo, apagar el reconocimiento de dispositivo para un cliente puntual es editar `sp.reconocimiento.habilitado: false` en `config/application.yml` y reiniciar el servicio — sin tocar código ni volver a buildear el jar. Es el objetivo completo de ADR-029.

Perfiles

Perfil	Uso
(sin perfil) / `dev`, `local`	Desarrollo. `bootui` (consola de desarrollo) se activa sola en `AUTO`; modo prueba puede estar prendido sin problema
`e2e`	Suite end-to-end (Playwright contra el jar empaquetado). Config propia en `backend/config/application-e2e.properties`
`load`	Pruebas de carga. Config propia en `backend/config/application-load.properties`
`prod`	Producción — ver guarda de abajo

`prod` mata el modo prueba — no se pueden combinar

`ModoPruebaConfig` corre al arranque: si `sp.modo-prueba.enabled=true` **y** el perfil activo contiene `prod`, el contexto **no levanta**. Es intencional — modo prueba nunca debe existir en el ambiente real. La consecuencia práctica, documentada en el runbook de despliegue: si necesitás generar QR de prueba en el servidor de producción real (para el smoke inicial), corrés **sin** `spring.profiles.active=prod` — lo que también significa que `bootui` no queda bloqueado por código en ese momento. La mitigación es de infraestructura: un reverse proxy (Caddy) devuelve 404 en `/bootui` y `/bootui/*` de entrada, para que la consola de desarrollo nunca quede expuesta públicamente aunque el proceso la tenga activa.

Ejemplo completo

Este es el `config/application.yml` real de un despliegue (basado en el runbook de instalación), con los valores a completar marcados:

```
sp.config.externa: true          # centinela ADR-029 — NO borrar

SP_DB_PASSWORD: "CAMBIAR"
SP_MASTER_KEY: "CAMBIAR"         # base64 32 bytes — el MISMO usado al seed
SP_GESTOR_USER: "admin"
SP_GESTOR_PASSWORD: "CAMBIAR"    # obligatoria: sin esto no arranca
SP_CLIENTE_NOMBRE: "Supermercados Caracol"
SP_CLIENTE_COLOR: "#e30613"
SP_BRANDING_DIR: "C:/work/sistemapremios/branding"
SP_MODO_PRUEBA: "true"           # NO usar junto con perfil prod
SP_BASE_URL: "https://sistemapremios.tipre.com"   # URL pública real, nunca
localhost

management:
  endpoints:
    web:
      exposure:
        include: health
  endpoint:
    health:
      show-details: when-authorized

server:
  address: 127.0.0.1              # solo localhost — el reverse proxy es quien
queda expuesto
  port: 28880
  forward-headers-strategy: framework
  tomcat:
    allow-trace: false
  compression:
    enabled: true
    mime-types:
text/html,text/css,application/javascript,text/javascript,application/json
    min-response-size: 1024

spring:
  application:
    name: sistemapremios-backend
  datasource:
    url:
${SP_DB_URL:jdbc:sqlserver://localhost;databaseName=SistemaPremios;encrypt=false}
    username: ${SP_DB_USER:sa}
    password: ${SP_DB_PASSWORD:${SP_DB_PASS:}}
  jpa:
    hibernate:
      ddl-auto: validate
    open-in-view: false
    properties:
      hibernate:
        jdbc:
          time_zone: America/Argentina/Buenos_Aires
        query:
          in_clause_parameter_padding: true
  flyway:
```

```
    enabled: true  jackson:    property-naming-strategy: SNAKE_CASE  servlet:
multipart:      max-file-size: 10MB      max-request-size: 10MBsp:  cliente:
nombre: ${SP_CLIENTE_NOMBRE:Cliente}    color-primario:
${SP_CLIENTE_COLOR:#E01020}  branding:    dir: ${SP_BRANDING_DIR:./branding}
modo-prueba:    enabled: ${SP_MODO_PRUEBA:false}  base-url:
${SP_BASE_URL:http://localhost:8080}  reconocimiento:    habilitado: true
```

Este archivo no contiene secretos reales

El `application.yml` versionado en el repo (`backend/config/application.yml`) es la referencia sin valores reales — los `${SP_*}` quedan sin resolver. Los valores reales van en la carpeta de deploy, fuera de git. Detalle del despliegue completo (NSSM, Caddy, TLS) en Instalar en el servidor.

Montar el entorno de desarrollo

Esta guía te lleva de una máquina limpia a los tres proyectos —backend, landing y gestor— corriendo en local con hot reload, campaña de prueba cargada y un login funcionando en el gestor. Calculá unos 20-30 minutos si ya tenés el software base instalado.

El **Sistema de Premios** es un único backend Java que sirve dos frontends: la **landing** (donde el cliente final escanea su ticket y participa) y el **gestor** (el panel donde el equipo de Caracol administra la campaña). En desarrollo corrés los tres procesos por separado para tener hot reload; en producción, el backend empaqueta y sirve a los otros dos (ver instalación en servidor).

> **Orden recomendado**
>
> Arrancá siempre por la base de datos y el backend. La landing y el gestor son clientes del backend — sin él levantado, ninguno de los dos tiene con qué hablar.

Prerrequisitos

Necesitás	Versión	Notas
JDK	**21**	El bytecode del jar apunta a Java 17, así que cualquier JDK ≥17 corre el proceso — pero usá 21 para que coincida con lo que usa el equipo.
Maven	cualquiera reciente	Sin wrapper en el repo: `mvn` tiene que estar en el `PATH`.
SQL Server	2022	Con una base vacía llamada `SistemaPremios` ya creada.
Node	**20**	Lo pide el `frontend-maven-plugin` del build `full`; usalo también para correr landing y gestor sueltos.

1. Crear la base de datos

Necesitás una base vacía llamada `SistemaPremios` en tu instancia de SQL Server. Flyway se encarga de crear el esquema entero la primera vez que arranca el backend — vos solo creás el contenedor vacío.

```
CREATE DATABASE SistemaPremios;
```

2. Clonar el repo y preparar las variables de entorno

```
git clone <url-del-repo> SistemaPremios
cd SistemaPremios\backend
```

El repo trae `backend/dev-env.local.ps1` (gitignoreado) con las variables `SP_*` que el backend necesita en dev. Antes de arrancar nada, cargalas en tu sesión de PowerShell:

```
. .\dev-env.local.ps1
```

> ⚠ **Sin `SP_GESTOR_PASSWORD` el backend no arranca**
>
> No es un default silencioso: si esa variable falta, el arranque falla rápido y explícito. Si el script `dev-env.local.ps1` no existe todavía en tu clon, armate uno con al menos estas variables (con tus propios valores, nunca los reales de producción):
>
> ```
> $env:SP_DB_PASSWORD = "CAMBIAR"
> $env:SP_MASTER_KEY = "CAMBIAR"          # base64 de 32 bytes
> $env:SP_GESTOR_USER = "admin"
> $env:SP_GESTOR_PASSWORD = "CAMBIAR"
> $env:SP_MODO_PRUEBA = "true"
> ```

3. Arrancar el backend con hot reload

Parado en `backend/`:

```
mvn spring-boot:run
```

Esto levanta el backend en `http://localhost:8080`, corre las migraciones de Flyway y queda escuchando con reinicio automático ante cambios de código (`spring-boot-devtools`). La config operativa la toma de `backend/config/application.yml`, el archivo versionado de referencia (ver ADR-029 y el detalle completo de cada variable en Configuración).

Verificá que levantó bien:

```
curl.exe http://localhost:8080/api/salud
```

Debería responder `{"estado":"ok","base_datos":"ok"}`.

4. Arrancar la landing

En otra terminal:

```
cd SistemaPremios\landing
npm install
npm run dev
```

Queda en `http://localhost:5173`. El `vite.config.ts` de la landing ya trae un proxy de `/api`, `/bases` y `/branding` hacia `http://localhost:8080`, así que no hay CORS que configurar: la landing en dev habla con tu backend local como si fuera same-origin.

5. Arrancar el gestor

En una tercera terminal:

```
cd SistemaPremios\gestor
npm install
npm run dev
```

Queda en `http://localhost:5174`, sirviendo bajo la ruta `/gestor/` (igual que en producción, donde el jar lo sirve bajo esa misma ruta dentro del mismo puerto).

6. Cargar el seed mínimo

Con el backend abajo (`Ctrl+C`), arrancalo una vez apuntando al seed de ejemplo:

```
mvn spring-boot:run "-Dspring-boot.run.arguments=--sp.seed.archivo=../docs/seed-minimo.ejemplo.json"
```

Esto carga una campaña vigente con 6 sorteos (5 semanales + cierre) y una clave de firma. Una vez que veas en el log que el seed corrió, podés volver a arrancar con `mvn spring-boot:run` normal — el flag solo hace falta la primera vez.

El seed es idempotente respecto al bean

Sin la property `sp.seed.archivo`, el bean `SeedMinimo` ni se registra — así que dejar el flag afuera en los arranques siguientes no vuelve a insertar nada de más.

7. Verificar que todo funciona junto

1. **Backend solo**: `http://localhost:8080/api/salud` → `{"estado":"ok","base_datos":"ok"}`.
2. **Landing**: abrí `http://localhost:5173` — debería cargar sin errores de consola relacionados a `/api`.
3. **Login del gestor**: abrí `http://localhost:5174/gestor/` (o directamente `http://localhost:5174/` si tu router redirige) e ingresá con `SP_GESTOR_USER` / `SP_GESTOR_PASSWORD` — los mismos valores que cargaste en el paso 2.
4. **Campaña vigente**: dentro del gestor, la pantalla **Panel** debería mostrar la campaña "Sorteo Aniversario Caracol" con sus 6 sorteos.

Entorno listo

Si los cuatro puntos de arriba cierran, tenés el entorno de desarrollo completo: backend + landing + gestor + datos de prueba.

Iterar con los tres procesos a la vez

Para el día a día dejá las tres terminales abiertas en paralelo:

Terminal	Comando	Puerto
Backend	`mvn spring-boot:run` (en `backend/`)	8080
Landing	`npm run dev` (en `landing/`)	5173
Gestor	`npm run dev` (en `gestor/`)	5174

Cada una tiene su propio hot reload: tocás un `.java` y `spring-boot:run` reinicia el contexto solo; tocás un `.tsx` en landing o gestor y Vite actualiza el navegador sin recargar la página entera. No hace falta reiniciar nada manualmente salvo que cambies una variable de entorno del backend (esas sí requieren volver a correr `. .\dev-env.local.ps1` y reiniciar `spring-boot:run`).

Para generar vouchers de prueba y probar el circuito de punta a punta sin depender de una impresora de ticket real, seguí con Generar vouchers de prueba (QR).

Instalar en el servidor del cliente

Esta guía instala el Sistema de Premios como servicio de Windows en el servidor de destino: build del jar único, carpeta de deploy fuera del repo, `config/application.yml` con los valores reales del cliente, arranque a mano para verificar, y el servicio con **NSSM** para que quede corriendo solo y se levante con el sistema operativo.

No hace falta Docker: es un jar Java corriendo nativo en Windows, con SQL Server ya instalado en el mismo servidor o accesible por red.

> **Esta página deja la app corriendo en `127.0.0.1:28880`, sin TLS**
>
> A propósito: acá dejamos la aplicación escuchando solo localhost. El paso de exponerla al público por HTTPS real es aparte — seguí con Publicar con HTTPS (Caddy + NSSM) inmediatamente después de terminar esta guía.

Prerrequisitos del servidor

- **JDK 21** instalado (el jar es bytecode Java 17, corre en cualquier JDK ≥17 — usamos 21 para estar alineados con el build).
- **SQL Server 2022** con la base de datos `SistemaPremios` ya creada (vacía; Flyway arma el esquema al primer arranque).
- NSSM descargado en el servidor (para correr la app como servicio Windows).
- Acceso de red del servidor a la instancia de SQL Server (local o remota).

1. Buildear el jar completo

El build se hace en tu máquina de build (o en el mismo servidor, si tiene Maven/Node). El perfil `full` empaqueta la landing y el gestor **dentro** del jar del backend — no hay que copiar nada de esos dos proyectos por separado.

```
cd C:\Work\Tipre\SistemaPremios\backend
mvn package -DskipTests -Pfull
```

Esto genera:

```
backend\target\sistemapremios-backend-0.0.1-SNAPSHOT.jar
```

con la landing (`/`) y el gestor (`/gestor/`) ya adentro.

2. Crear la carpeta de deploy (fuera del repo)

La carpeta de deploy es donde vive todo lo operativo del servidor — el jar, la config real con secretos, las bases legales, el branding y los logs. **No es un checkout de git**: nunca se versiona.

```
mkdir C:\work\sistemapremios
mkdir C:\work\sistemapremios\config
mkdir C:\work\sistemapremios\bases
mkdir C:\work\sistemapremios\branding
mkdir C:\work\sistemapremios\logs
```

Copiá el jar recién buildeado y el logo del cliente:

```
copy backend\target\sistemapremios-backend-0.0.1-SNAPSHOT.jar
C:\work\sistemapremios\
```

```
copy docs\branding\caracol\logo.png C:\work\sistemapremios\branding\logo.png
```

3. Escribir `config\application.yml`

Este es el archivo que reemplaza al que viaja en el repo como referencia (ver ADR-029 y el detalle de cada clave en Configuración). Va **junto al jar**, en `C:\work\sistemapremios\config\application.yml`, y acá sí lleva los valores reales.

Creá el archivo con este contenido, reemplazando cada `CAMBIAR`:

```yaml
# config externa (ADR-029). Va junto al jar. NO en git (valores reales).

sp.config.externa: true          # centinela ADR-029 — NO borrar

# ── valores del cliente (resuelven los ${SP_*} de abajo) ──
SP_DB_PASSWORD: "CAMBIAR"                              # sa de SQL Server
(SistemaPremios)
SP_MASTER_KEY: "CAMBIAR"                               # base64 32 bytes — EL
MISMO usado al seed
SP_GESTOR_USER: "admin"                                # login del gestor
SP_GESTOR_PASSWORD: "CAMBIAR"                           # login del gestor
(obligatorio: sin esto NO arranca)
SP_CLIENTE_NOMBRE: "Supermercados Caracol"
SP_CLIENTE_COLOR: "#e30613"
SP_BRANDING_DIR: "C:/work/sistemapremios/branding"     # con logo.png adentro
SP_MODO_PRUEBA: "true"                                  # generador de QR de
prueba (NO usar perfil prod)
SP_BASE_URL: "https://sistemapremios.tipre.com"        # URL pública que va
DENTRO del QR (¡no localhost!)

management:
  endpoints:
    web:
      exposure:
        include: health
  endpoint:
    health:
      show-details: when-authorized

server:
  address: 127.0.0.1                    # solo localhost — solo el reverse proxy
entra
  port: 28880
  forward-headers-strategy: framework    # detrás de Caddy ya reescribe la IP
real (X-Forwarded-For) — NO cambiar
  tomcat:
    allow-trace: false
  compression:
    enabled: true
    mime-types:
text/html,text/css,application/javascript,text/javascript,application/json
    min-response-size: 1024

spring:
  application:
    name: sistemapremios-backend
  datasource:
    url:
${SP_DB_URL:jdbc:sqlserver://localhost;databaseName=SistemaPremios;encrypt=false}
    username: ${SP_DB_USER:sa}
    password: ${SP_DB_PASSWORD:${SP_DB_PASS:}}
  jpa:
    hibernate:
      ddl-auto: validate
```

```
    open-in-view: false    properties:      hibernate:        jdbc:
time_zone: America/Argentina/Buenos_Aires        query:
in_clause_parameter_padding: true  flyway:    enabled: true  jackson:
property-naming-strategy: SNAKE_CASE  servlet:    multipart:      max-file-size:
10MB      max-request-size: 10MBsp:  cliente:    nombre:
${SP_CLIENTE_NOMBRE:Cliente}    color-primario: ${SP_CLIENTE_COLOR:#E01020}
branding:    dir: ${SP_BRANDING_DIR:./branding}  modo-prueba:    enabled:
${SP_MODO_PRUEBA:false}  base-url: ${SP_BASE_URL:http://localhost:8080}
reconocimiento:    habilitado: true
```

Tres variables que muerden si las salteás

- `SP_GESTOR_PASSWORD` — sin ella el backend **no arranca**. No hay fallback silencioso.
- `SP_BASE_URL` — si queda en `localhost`, los QR que genera el sistema apuntan a `localhost` y ningún cliente fuera de esa máquina puede escanearlos.
- `SP_MASTER_KEY` — tiene que ser **la misma** que se usó al sembrar los datos (seed), o los QR emitidos antes no vuelven a validar.

Fallback LAN sin reverse proxy (self-signed, no recomendado para producción pública)

Si por algún motivo no vas a poner Caddy delante (por ejemplo, una prueba interna en la LAN del cliente sin dominio público), podés reemplazar el bloque `server` de arriba por:

```
server:
  port: 8443
  ssl:
    enabled: true
    key-store: "file:C:/work/sistemapremios/sp-dev.p12"
    key-store-password: changeit
    key-store-type: PKCS12
    key-alias: sp
```

Con Caddy delante (el camino recomendado, ver el paso siguiente) esto **no** hace falta.

4. Probar el arranque a mano

Antes de instalarlo como servicio, corré el jar directamente para confirmar que todo está en orden. Parado en la carpeta de deploy:

```
cd C:\work\sistemapremios
```

```
& "C:\Program Files\Java\jdk-21\bin\java.exe" -jar sistemapremios-backend-0.0.1-SNAPSHOT.jar
```

Flyway migra la base al arrancar. Si falta la carpeta `config\`, o si le falta alguna clave de contrato, el fail-fast de ADR-029 corta el arranque y te dice exactamente qué archivo esperaba y en qué working directory buscó — nunca arranca en silencio con defaults.

En otra terminal, verificá el health check:

```
curl.exe http://127.0.0.1:28880/api/salud
```

Debería responder `{"estado":"ok","base_datos":"ok"}`. Si respondió bien, cortá el proceso (`Ctrl+C`) y pasá al servicio.

5. Instalar el servicio con NSSM

```
nssm install SistemaPremios "C:\Program Files\Java\jdk-21\bin\java.exe" "-jar C:\work\sistemapremios\sistemapremios-backend-0.0.1-SNAPSHOT.jar"
```

```
nssm set SistemaPremios AppDirectory C:\work\sistemapremios
```

```
nssm set SistemaPremios AppStdout C:\work\sistemapremios\logs\out.log
```

```
nssm set SistemaPremios AppStderr C:\work\sistemapremios\logs\err.log
```

```
nssm set SistemaPremios Start SERVICE_AUTO_START
```

```
nssm start SistemaPremios
```

> **`AppDirectory` es CRÍTICO — no es un detalle cosmético**
>
> Spring Boot busca `./config/` **relativo al working directory del proceso**, no a la ubicación del jar. Si `AppDirectory` no apunta a `C:\work\sistemapremios` (la carpeta que contiene `config\`), el servicio arranca con el proceso corriendo en otro directorio y el chequeo de config externa falla — aunque el jar esté en la ruta correcta. Es el error más común al migrar de "corrí bien a mano" a "no arranca como servicio".

Confirmá que el servicio está arriba:

```
nssm status SistemaPremios
```

```
curl.exe http://127.0.0.1:28880/api/salud
```

6. Backup diario

La carpeta de deploy y la base de datos son lo único que no se recupera con un simple re-deploy del jar (el jar se regenera desde el repo; los datos, no). Como mínimo, programá dos backups diarios fuera de horario pico:

Backup de la base de datos (ejemplo con `sqlcmd`, ajustá credenciales y ruta):

```
sqlcmd -S localhost -U sa -P "CAMBIAR" -Q "BACKUP DATABASE SistemaPremios TO DISK
= 'C:\backups\SistemaPremios_$(Get-Date -Format yyyyMMdd).bak'"
```

Backup de la carpeta de deploy (config, branding y logs — no hace falta backupear el jar, ese sale del repo):

```
robocopy C:\work\sistemapremios\config C:\backups\sistemapremios\config /MIR
```

```
robocopy C:\work\sistemapremios\branding C:\backups\sistemapremios\branding /MIR
```

> **Programalo con el Programador de tareas de Windows**
>
> Envolvé estos comandos en un `.ps1` y agendalo con `schtasks` o el Programador de tareas de Windows, corriendo una vez por día fuera del horario de mayor tráfico de la campaña. `C:\backups\` es un ejemplo — usá la ruta de backup que ya tenga definida la infraestructura del cliente (idealmente en otro disco o servidor).

Siguiente paso

La app ya corre como servicio, pero solo responde en `127.0.0.1:28880` — todavía no es accesible desde afuera. Para publicarla con HTTPS real (necesario para que el escáner de DNI de la landing funcione en cualquier celular), seguí con Publicar con HTTPS (Caddy + NSSM).

Publicar con HTTPS (Caddy + NSSM)

Esta guía expone al público la app que dejaste corriendo en Instalar en el servidor (escuchando en `127.0.0.1:28880`), poniendo **Caddy** delante como reverse proxy que termina TLS con un certificado real de Let's Encrypt, y corriéndolo también como servicio Windows con NSSM.

```
Internet ──443/TLS──▶ Caddy (cert Let's Encrypt) ──HTTP──▶ app :28880 (solo
127.0.0.1)
```

> **Por qué HTTPS real y no un certificado autofirmado**
>
> La landing usa la **cámara del celular** para escanear el DNI del cliente (lector de código de barras PDF417). Los navegadores móviles solo habilitan `getUserMedia` (acceso a cámara) en un **secure context** — HTTPS con un certificado válido, o `localhost`. Con un certificado autofirmado, el navegador muestra un warning de seguridad y en muchos casos bloquea directamente el acceso a la cámara. Un dominio público con cert de Let's Encrypt es lo único que garantiza que el escaneo funcione en cualquier teléfono, de cualquier cliente, sin fricción.

Prerrequisitos

- Caddy (binario estándar, sin necesidad de módulos DNS extra).
- Un **registro DNS A** apuntando el dominio de la campaña a la IP pública/elástica del servidor.
- Puerto **443/tcp** abierto hacia el servidor (en el Security Group de AWS, o el firewall que corresponda). El puerto 80 **no** hace falta — Caddy emite el certificado por TLS-ALPN, sin pasar por HTTP.
- **Windows Firewall** con el puerto 443 permitido:

```
netsh advfirewall firewall add rule name="Caddy HTTPS 443" dir=in action=allow
protocol=TCP localport=443
```

1. Escribir el `Caddyfile`

Creá el `Caddyfile` junto al binario `caddy.exe` (por ejemplo, `D:\tipre\caddy\Caddyfile`):

```
{
    email admin@tipre.com
    auto_https disable_redirects
    storage file_system {
        root D:\tipre\caddy\data
    }
}

sistemapremios.tipre.com {
    tls {
        issuer acme {
            disable_http_challenge     # TLS-ALPN-01 por 443; Caddy nunca toca el
80
        }
    }

    @bootui path /bootui /bootui/*     # consola de dev — nunca pública
    respond @bootui 404

    reverse_proxy 127.0.0.1:28880 {
        header_up X-Forwarded-For {remote_host}   # pisa el XFF entrante: el
cliente no puede falsificar su IP
    }
}
```

Reemplazá `sistemapremios.tipre.com` por el dominio real de la campaña, y `admin@tipre.com` por el email que Let's Encrypt usa para avisos de renovación.

`header_up X-Forwarded-For {remote_host}` no es opcional

Detrás de un reverse proxy, todas las requests le llegan al backend con la misma IP de origen si no se reescribe el header — el rate limiting (30 req/60s por IP) colapsaría en un solo balde compartido por todos los clientes. Peor: sin este pisado, un cliente malicioso podría **falsificar su propia IP** mandando su propio `X-Forwarded-For` y esquivar el límite. Caddy tiene que pisar el header entrante con la IP real del socket (`{remote_host}`), nunca reenviar el que venga del cliente sin tocar.

El bloqueo de `/bootui` es la mitigación de infraestructura, no de código

La consola de desarrollo `boot-ui` se autodesactiva por código solo con el perfil `prod` — pero `prod` mata el modo prueba (fail-fast), y necesitamos modo prueba prendido para poder generar QRs de prueba en el propio servidor. Por eso corremos **sin** `spring.profiles.active=prod`, lo que significa que `boot-ui` sigue activo a nivel de la app. El `@bootui → 404` de este Caddyfile es lo que evita que esa consola quede expuesta públicamente.

2. Probar Caddy a mano antes de instalarlo como servicio

```
cd D:\tipre\caddy
```

```
caddy run --config Caddyfile
```

Mirá la consola: el primer arranque tarda unos segundos en emitir el certificado. Cuando veas que levantó sin errores, `Ctrl+C` y pasá al servicio.

3. Instalar Caddy como servicio con NSSM

```
nssm install caddy "D:\tipre\caddy\caddy.exe" "run --config
D:\tipre\caddy\Caddyfile"
```

```
nssm set caddy AppDirectory D:\tipre\caddy
```

```
nssm set caddy Start SERVICE_AUTO_START
```

```
nssm start caddy
```

4. Verificar

```
nssm status caddy
```

```
curl.exe -v https://sistemapremios.tipre.com/api/salud
```

El primer arranque puede tardar unos segundos en emitir el certificado — si el `curl` falla apenas arrancás el servicio, esperá un momento y reintentá antes de asumir que algo está mal.

5. Smoke test completo

Con los dos servicios (`SistemaPremios` y `caddy`) arriba, confirmá el circuito de punta a punta:

1. **Health por HTTPS**: `https://sistemapremios.tipre.com/api/salud` → `{"estado":"ok","base_datos":"ok"}`.

2. **Landing**: abrí el dominio público en el navegador — debería cargar sin warning de certificado.
3. **QR de prueba**: entrá al gestor (`https://sistemapremios.tipre.com/gestor/`, login `SP_GESTOR_USER` / `SP_GESTOR_PASSWORD`) → **Modo prueba** → generá un QR y escaneálo desde un celular real, fuera de la red del servidor. Ver el detalle en Generar vouchers de prueba (QR).
4. **`bootui` bloqueado**: `https://sistemapremios.tipre.com/bootui` debería devolver `404`.

> ✓ **Publicación completa**
>
> Si los cuatro puntos cierran, el sistema está publicado con HTTPS real, el rate limit funciona por IP real de cada cliente, y la consola de desarrollo no queda expuesta.

Seguridad — resumen de las guardas activas

- **Gestor bajo autenticación básica**: `/api/gestor/**` exige usuario y password en cada request.
- **Actuator cerrado** salvo `/health`, y ese health no muestra detalle a un caller anónimo.
- **Secretos nunca en el repo ni en el Caddyfile**: la carpeta de deploy con los valores reales (`config/application.yml`) vive fuera de git, según ADR-029. `SP_MASTER_KEY` en el YAML es un atajo operativo aceptado para este despliegue; si el cliente pide un manejo de secretos más estricto, se puede inyectar por `AppEnvironmentExtra` de NSSM en lugar de dejarla en el archivo.

Con la app publicada y el smoke test en verde, el siguiente paso operativo es cargar la campaña real desde el gestor — ver Preparar una campaña.

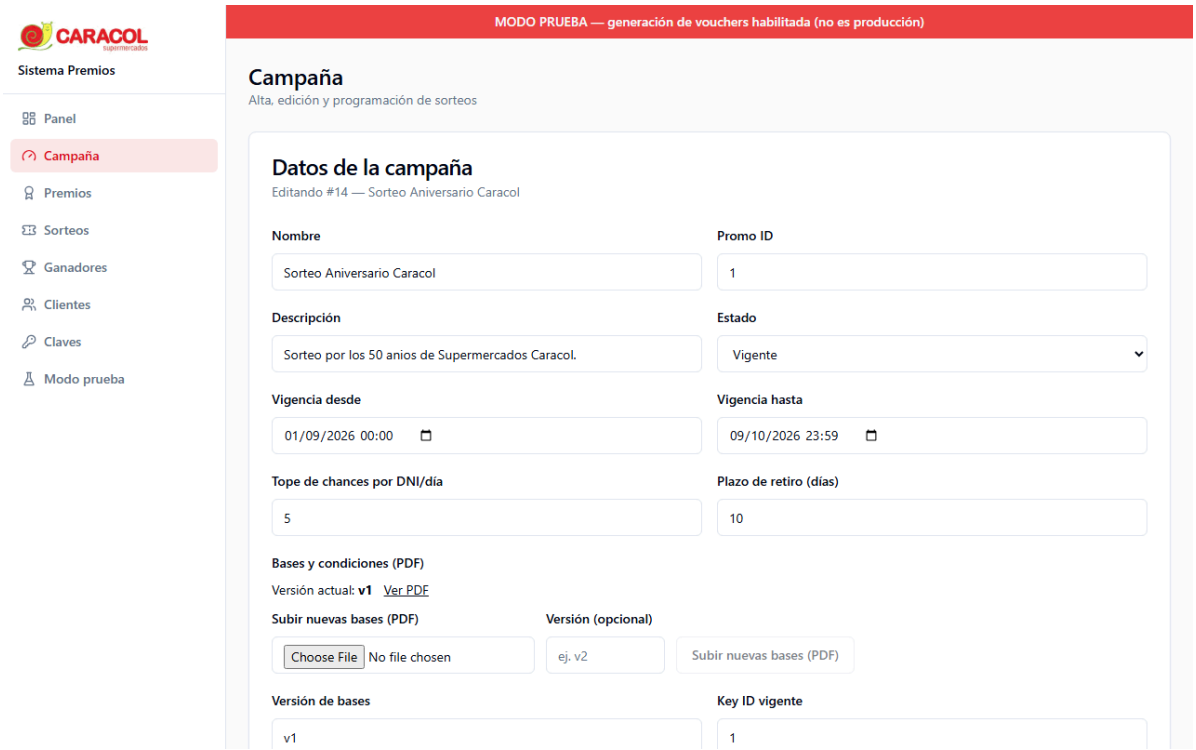
Preparar una campaña

Esta guía te lleva por la carga operativa de una campaña desde el gestor: revisar los datos generales, cargar los tipos de premio, generar las unidades y asignarlas a los sorteos. Al terminar, la campaña queda lista para que el equipo de sucursal empiece a validar tickets.

> **A quién apunta esta guía**
>
> Al equipo de Caracol que administra la campaña desde el gestor (`/gestor/`), con las credenciales `SP_GESTOR_USER` / `SP_GESTOR_PASSWORD` . Si todavía no tenés el sistema instalado y publicado, empezá por Instalar en el servidor y Publicar con HTTPS.

1. Revisar los datos de la campaña

Entrá a la sección **Campaña** del gestor y confirmá que los datos generales estén bien cargados: vigencia, textos que se muestran en la landing, la regla de chances (cuántas chances vale cada voucher) y el tope diario por DNI.



Solo puede haber una campaña vigente a la vez

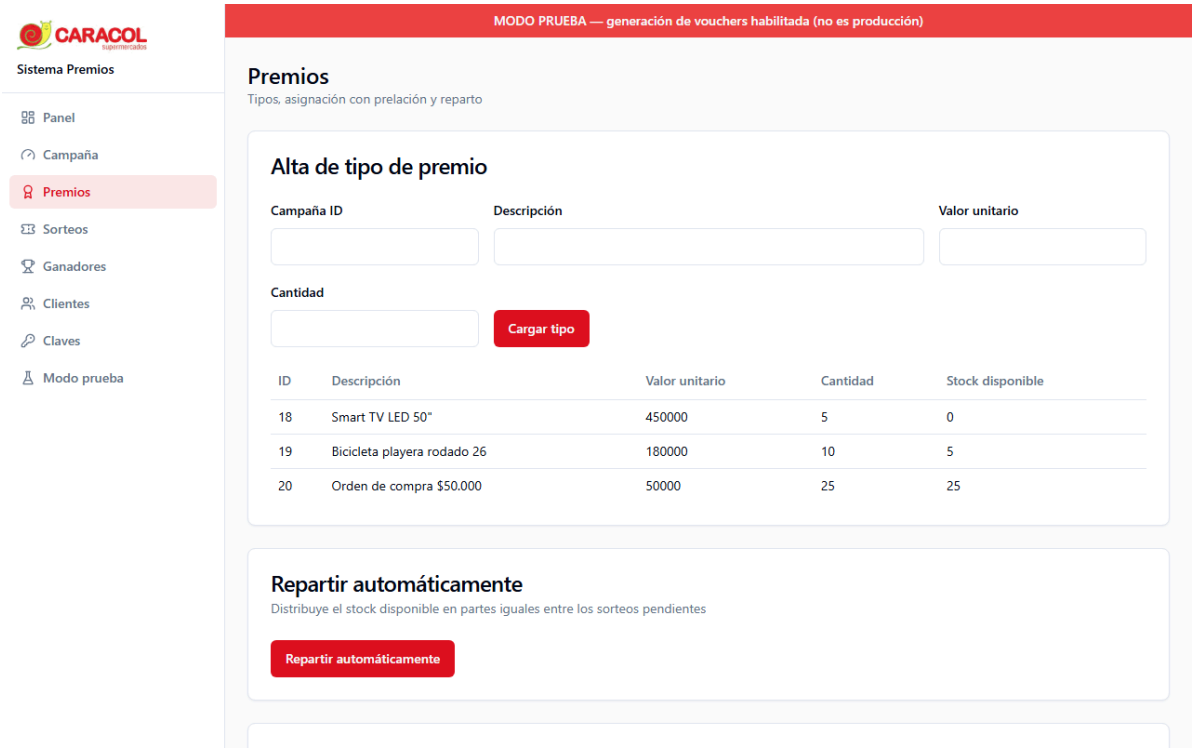
El sistema garantiza una única campaña activa. Los textos que cargués acá (nombre, descripción) se reflejan directo en la landing sin tocar código — así se puede ajustar la comunicación de la promo sin depender de un nuevo despliegue.

La vigencia se compara contra la fecha de emisión del ticket, no contra "hoy"

Un ticket con fecha de emisión dentro de la ventana de la campaña participa aunque se escanee después de cerrada la vigencia. Si necesitás el detalle completo de esta regla, ver Los casos de participación.

2. Cargar los tipos de premio

Andá a la sección **Premios**. Ahí se cargan los tipos de premio (por ejemplo, "Bicicleta playera × 10"), se controla el stock disponible y se asignan a los sorteos.



1. **Cargar un tipo de premio**: completá nombre y cantidad. El sistema **genera automáticamente** cada unidad individual del lote — no hace falta crear las unidades una por una.

2. Repetí para cada tipo de premio de la campaña, siguiendo la lista real que definió el cliente.

> ⚠ **Es un lote único para toda la campaña**
>
> No hay "recargar stock" a mitad de campaña como flujo estándar — la cantidad que cargás por tipo de premio es la que genera todas las unidades disponibles. Confirmá la lista completa de premios con el cliente antes de cargar, para no tener que hacer altas parciales sobre la marcha.

3. Asignar unidades a los sorteos

Con las unidades generadas, el siguiente paso es asignarlas a sorteos futuros, con su **prelación** — el orden en el que se reparten según el orden en que sale sorteada cada persona (el primer ganador sorteado se lleva el premio de prelación 1, y así sucesivamente).

Tenés dos caminos, no excluyentes:

- **Asignación manual**: elegís vos qué unidades van a qué sorteo y en qué prelación. Es el camino para cuando la distribución de premios entre semanas no es pareja (por ejemplo, un premio grande reservado para el sorteo de cierre).
- **Repartir automáticamente**: un atajo que distribuye el stock en partes iguales entre las semanas, como **punto de partida**. No es una decisión final — siempre podés ajustar la asignación y la prelación después de usarlo.

> 🔥 **El reparto automático es un punto de partida, no el resultado final**
>
> Usalo para no arrancar de cero, pero revisá siempre la asignación resultante antes de dar la campaña por lista — sobre todo si hay premios de valor muy distinto que conviene repartir a criterio y no en partes iguales.

Cada unidad de premio recorre un ciclo de vida propio: **Disponible** → **Asignado** → **Sorteado** → **Otorgado**, con un estado terminal aparte (**No otorgado**) para cuando un premio no se puede entregar. El detalle completo de ese ciclo, incluyendo devoluciones al stock y reasignaciones, está en Premios y su ciclo de vida.

4. Revisar la campaña antes de lanzar

Antes de dar por lista la campaña para que arranque la validación de tickets en sucursal, repasá:

- [ ] La **vigencia** de la campaña cubre todo el período real de la promo.
- [ ] Todos los tipos de premio del acuerdo comercial están cargados, con la cantidad correcta.
- [ ] Cada sorteo tiene unidades asignadas — ningún sorteo debería quedar sin premios el día que le toca ejecutarse.
- [ ] La prelación de cada sorteo tiene sentido (el premio más valioso, primero).
- [ ] El modo prueba generó al menos un QR de prueba y el circuito completo (caja → QR → landing → gestor) funcionó de punta a punta.

> ✓ **Campaña lista**
>
> Con la campaña vigente, los premios cargados y asignados, y el smoke test de QR en verde, el sistema está listo para que las cajas empiecen a emitir tickets reales.

Con la campaña corriendo, el siguiente paso operativo llega cuando se cumple la fecha de un sorteo — ver Ejecutar un sorteo.

Ejecutar un sorteo

Esta guía es el paso a paso para ejecutar un sorteo desde el gestor: qué revisar antes, cómo confirmar la ejecución y cómo descargar el acta resultante.

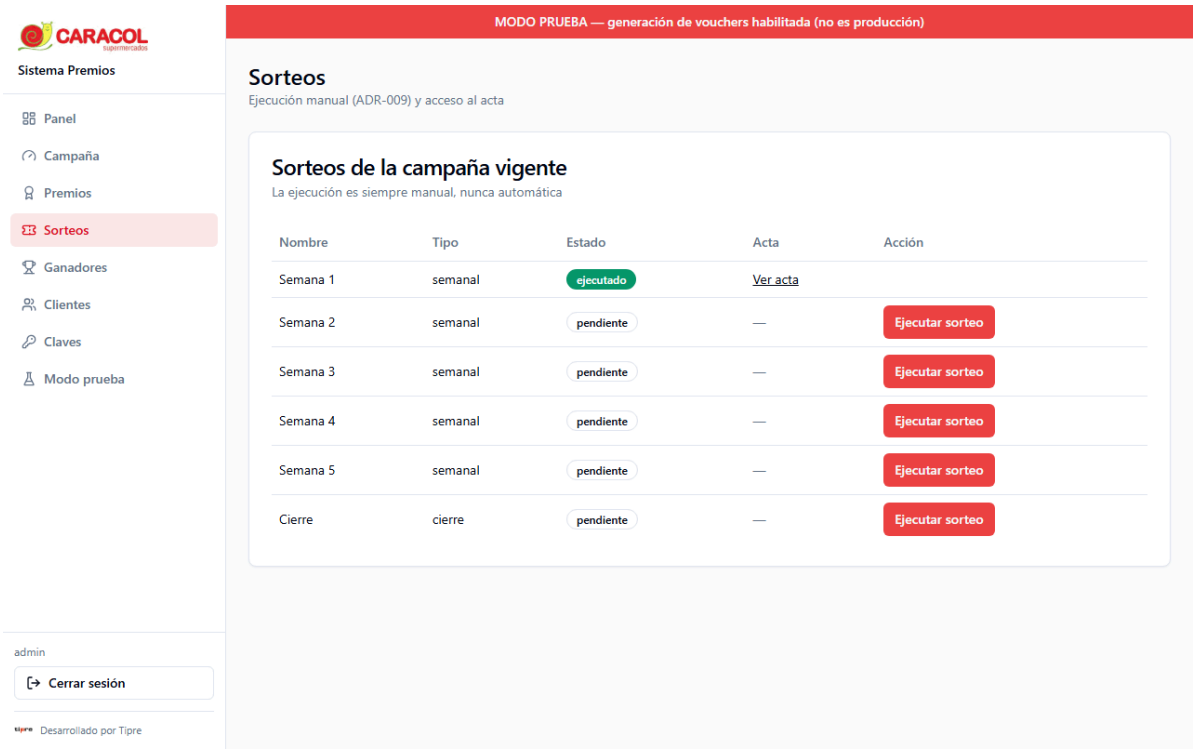
> **Ejecutar un sorteo es irreversible**
>
> Una vez ejecutado, el resultado queda en un **acta inmutable**: la lista de ganadores y suplentes de ese sorteo no se puede volver a generar ni "reejecutar". No existe un botón de deshacer. Si algo sale mal después de ejecutar (un error de campaña detectado tarde, por ejemplo), la corrección es una **decisión humana documentada** sobre esa acta — nunca un reintento del sorteo. Por eso, antes de tocar **Ejecutar**, asegurate de haber revisado todo lo de abajo.

Antes de ejecutar: qué revisar

1. Confirmá que el sorteo que vas a ejecutar es el que **corresponde a la fecha de hoy** — no hay disparo automático por reloj, así que la responsabilidad de elegir el sorteo correcto es tuya.
2. Confirmá que el sorteo tiene **unidades de premio asignadas** (ver Preparar una campaña). Si un sorteo no tiene premios asignados, el motor sortea lo que hay disponible, que podría ser nada.
3. Si tenés dudas sobre cuántas chances o clientes elegibles hay en juego, repasá el **Panel** del gestor antes de ejecutar.

1. Ir a la sección Sorteos

La lista de sorteos muestra cada sorteo semanal y el de cierre, con su estado y la acción de ejecutar.



Ubicá el sorteo correcto (el que corresponde a la fecha de hoy) y tocá **Ejecutar**.

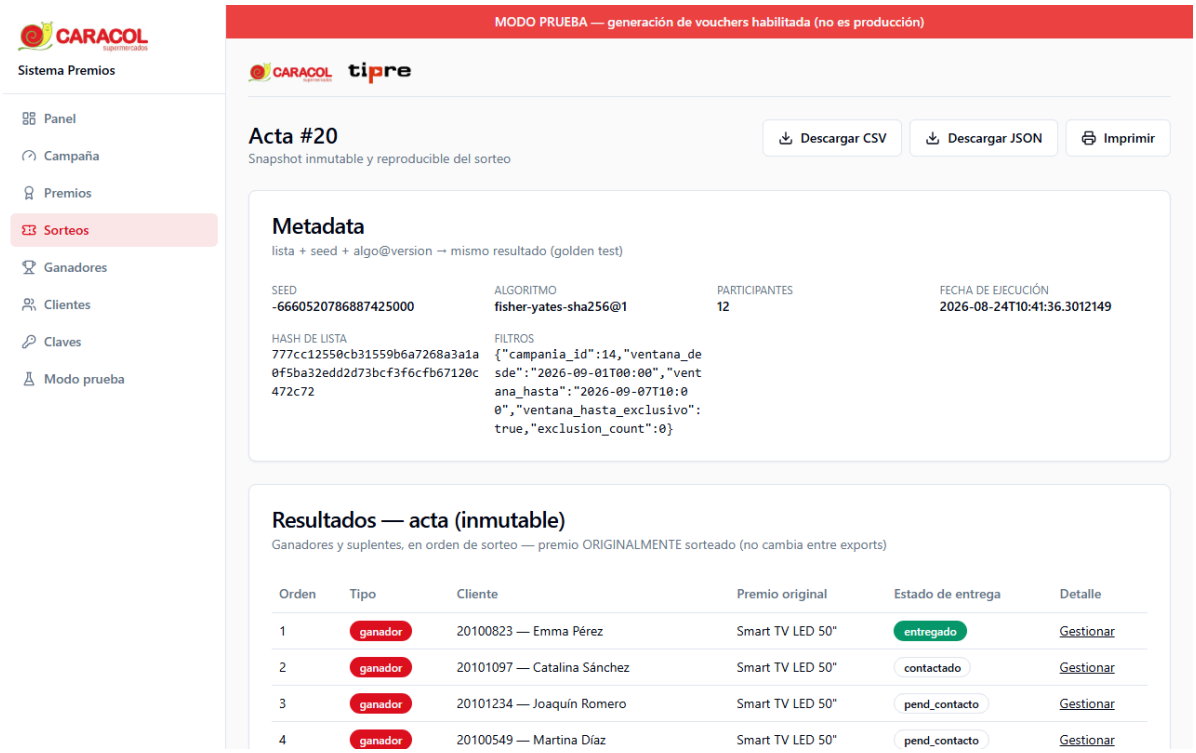
2. Confirmar la ejecución

El botón **Ejecutar** pide **doble confirmación** antes de correr el sorteo — un diálogo de confirmación primero, y después te va a pedir que **escribas la palabra** `EJECUTAR` para habilitar el botón final. Es intencional: la ejecución del sorteo siempre está a cargo de una persona en un momento concreto, nunca de un disparador automático (ver ADR-009).

> ⚠ **Escribí `EJECUTAR` (en mayúsculas, tal cual) para confirmar**
>
> Es una barrera deliberada contra el click accidental — no alcanza con confirmar el primer diálogo, tenés que tipear la palabra completa.

3. Ver el acta generada

Cada sorteo ejecutado genera un **acta**: el registro inmutable y reproducible de esa ejecución, con la semilla (`seed`) usada, el algoritmo y versión, un hash de la lista de elegibles, y el resultado ordenado con ganadores y suplentes.



El acta es reproducible, no solo un registro

La misma lista de elegibles + la misma semilla + el mismo algoritmo siempre dan el mismo resultado. Eso es lo que hace que el acta sea auditable: cualquiera puede reproducirla y llegar a los mismos ganadores. Ver el detalle técnico en Motor de sorteo y acta.

4. Descargar el acta

Desde la pantalla del acta podés bajarla en:

- **CSV** — para un reporte o cruce en planillas.
- **JSON** — para un respaldo estructurado o integración.
- **Imprimir** — para entregar en papel, si el cliente lo pide.

El acta no cambia, aunque cambie la entrega

Lo que salió sorteado (el premio original de cada ganador) queda **congelado para siempre** en el acta, aunque después ese premio pase a un suplente o se devuelva al stock. Si exportás la misma acta dos veces en momentos distintos, la parte del sorteo es idéntica; el estado de entrega puede diferir. Ver ADR-025.

Siguiente paso

Con el acta generada, el trabajo pasa a contactar a cada ganador y gestionar la entrega de su premio — ver Gestionar ganadores y entregas.

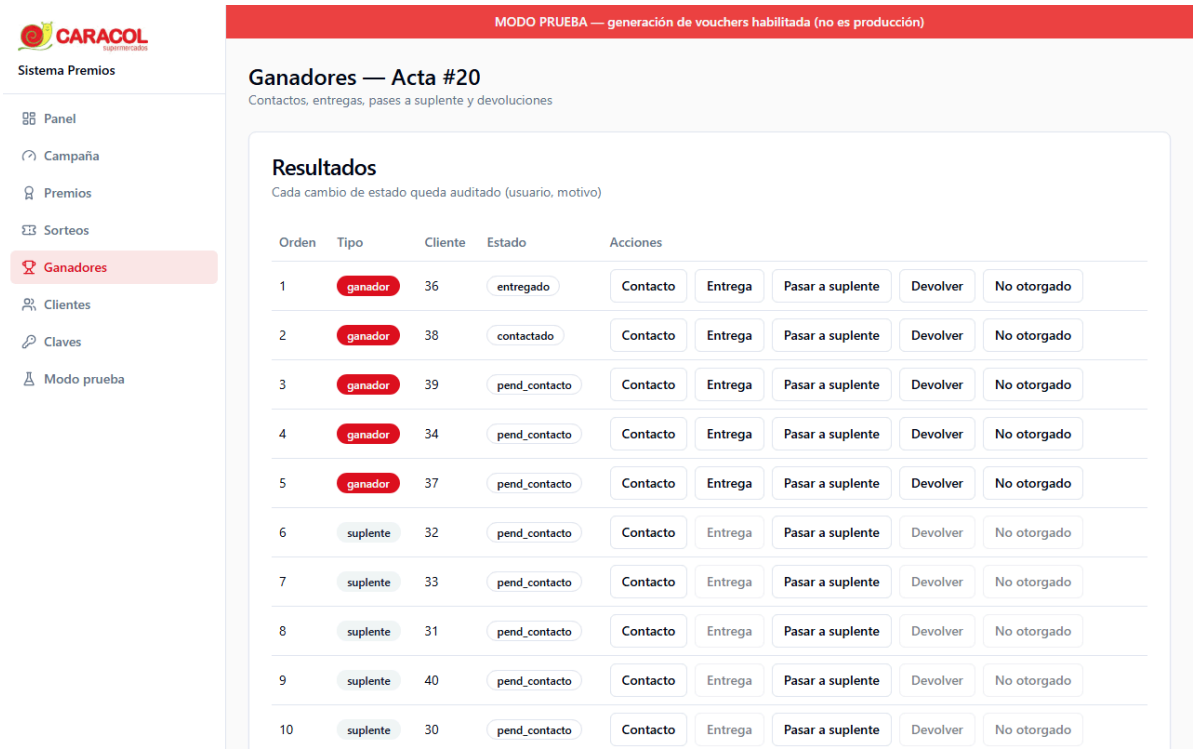
Gestionar ganadores y entregas

Esta guía cubre el flujo operativo completo después de ejecutar un sorteo: contactar a cada ganador, registrar la entrega verificando su identidad, y qué hacer cuando el titular no aparece o el premio no se puede entregar.

Si todavía no ejecutaste el sorteo que da origen a estos ganadores, empezá por Ejecutar un sorteo.

1. Abrir los ganadores de un acta

Desde **Sorteos**, entrá al sorteo ya ejecutado y abrí su **Acta**. Desde ahí accedés a la pantalla de **Ganadores**, donde gestionás todo el circuito de contacto y entrega de cada premio.



Cada ganador tiene un estado:

Estado	Qué significa

Pend. contacto	Todavía no se lo contactó.
Contactado	Ya se avisó al ganador; está pendiente de coordinar o concretar la entrega. Acá arranca a correr el plazo de retiro.
Entregado	El premio ya se le entregó. Estado terminal para ese ganador.

2. Registrar el contacto

Cuando avisás al ganador (por teléfono, mensaje, o el canal que use la operación), registrá la acción **Contacto** sobre su fila. Esto marca el estado como **Contactado** y arranca a correr el **plazo de retiro** de la campaña (10 días corridos desde el primer contacto).

3. Registrar la entrega — con verificación de DNI obligatoria

Cuando el ganador se presenta a retirar su premio, usá la acción **Entrega**.

Nunca se entrega un premio sin verificar el DNI contra el acta

El sistema siempre exige confirmar que el DNI de la persona que retira coincide con el DNI que figura en el acta como ganador. No hay excepción a este paso: es lo único que garantiza que el premio llega a quien realmente lo ganó, y no a otra persona que se presente con el mismo aviso de contacto.

Una vez verificado el DNI y confirmada la entrega, la unidad de premio pasa a **Otorgado** — estado terminal para esa unidad.

Si el plazo de retiro venció

El gestor te va a advertir si el plazo de 10 días desde el primer contacto ya pasó, y te va a pedir una **confirmación explícita** antes de dejarte continuar. No lo bloquea del todo: la decisión final de si igual corresponde entregar el premio queda en manos del operador, pero el sistema no te deja pasarlo por alto sin darte cuenta.

4. Cuando el titular no aparece

El circuito de entrega tiene desvíos previstos para cuando el ganador titular está incontactable, rechaza el premio, o no se presenta:

Pasar a suplente

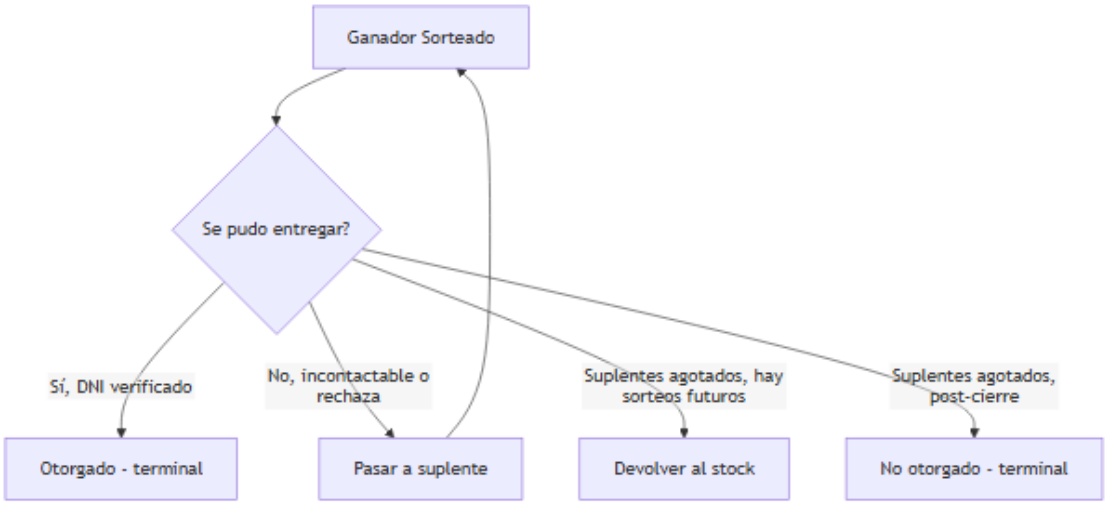
Usá la acción **Pasar a suplente** sobre la fila del ganador. El sistema avanza automáticamente al **siguiente suplente** de la misma acta, respetando el orden en que salieron sorteados (cada acta trae 10 suplentes de reserva).

Devolver al stock

Si ya se agotaron los suplentes de esa acta y todavía hay **sorteos futuros** en la campaña, usá **Devolver**: la unidad vuelve al stock disponible para poder asignarla a un sorteo posterior. Esta acción siempre pide un motivo y queda auditada.

Marcar No otorgado

Si el premio no se puede entregar y ya **no hay sorteos futuros** a los que devolverlo (por ejemplo, después del cierre de la campaña), usá **No otorgado**. Es un estado terminal: cierra la unidad de forma explícita, con motivo, para que el inventario final cierre exacto sin unidades en el limbo.



Por qué existe el estado No otorgado

Tras el cierre de campaña ya no hay "próximos sorteos" a los que devolver una unidad no entregada. Sin un estado terminal para ese caso, el stock arrastraría unidades fantasma para siempre. El detalle completo de este ciclo de vida está en Premios y su ciclo de vida y en ADR-008.

Resumen de acciones

Acción	Cuándo usarla	Efecto
Contacto	Avisaste al ganador	Pasa a Contactado, arranca el plazo de retiro
Entrega	El ganador retira el premio	Exige verificar DNI contra el acta; pasa a Otorgado
Pasar a suplente	Titular incontactable o rechaza	Avanza al siguiente suplente de la acta
Devolver	Suplentes agotados, hay sorteos futuros	Vuelve a Disponible en el stock
No otorgado	Suplentes agotados, post-cierre	Estado terminal, con motivo

Para preguntas frecuentes de clientes sobre su participación o sus premios, ver Dar soporte a clientes.

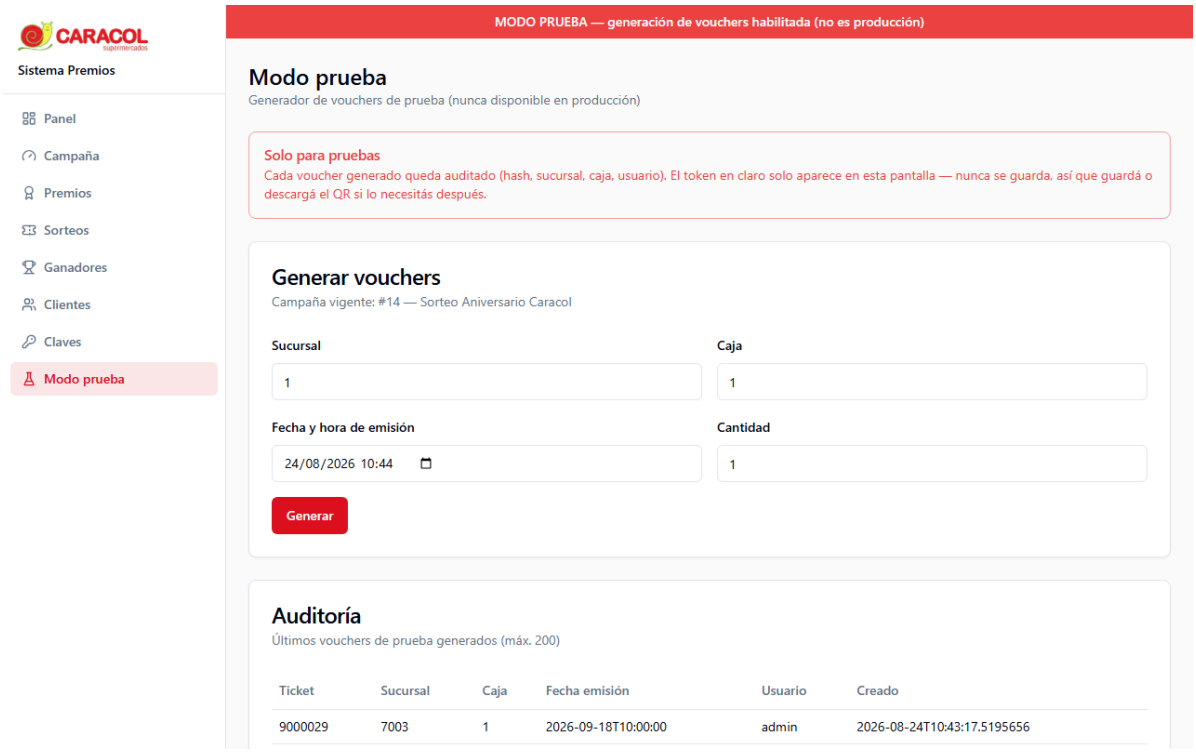
Generar vouchers de prueba (QR)

El **modo prueba** del gestor genera vouchers de prueba con su QR válido, sin necesidad de una impresora de ticket real ni de esperar a que una caja emita un ticket verdadero. Es la herramienta para validar el circuito completo (caja → QR → landing → gestor) antes de una salida en vivo, o para reproducir un caso puntual reportado por soporte.

> **El modo prueba nunca existe en producción real**
>
> Está protegido por dos guardas independientes: si `sp.modo-prueba.enabled` está apagado, la pantalla ni existe (404). Y si estuviera prendido **junto con** el perfil `prod`, el backend directamente **no arranca** (fail-fast) — no hay combinación posible de "modo prueba activo en el ambiente real de cara al cliente". Los QR que genera este modo están firmados con una **clave de prueba separada** de la clave de producción, así que nunca se confunden con participaciones reales.

1. Ir a Modo prueba

En el gestor, andá a la sección **Modo prueba**.



2. Completar los datos del voucher

Campo	Rango / formato	Para qué
Sucursal	1 a 65535	El número de sucursal que va a llevar el QR generado — simula desde qué local salió el ticket.
Caja	1 a 255	El número de caja emisora.
Fecha/hora de emisión	Dentro de la vigencia de la campaña	El sistema evalúa la vigencia y las reglas de tope diario contra **esta** fecha, no contra el momento real en que generás el QR — así podés simular tickets de cualquier día dentro de la campaña.
Cantidad	—	Cuántos vouchers de prueba generar de una vez.

> **La fecha de emisión tiene que caer dentro de la vigencia de la campaña**
>
> Si ponés una fecha fuera del período vigente, el voucher se genera igual pero al escanearlo el sistema lo va a rechazar por "fuera de vigencia" — el mismo comportamiento que tendría un ticket real con esa fecha. Es útil para probar justamente ese caso, pero confirmá que es lo que querés probar antes de reportarlo como un bug.

3. Generar y usar el resultado

Al confirmar, el sistema te devuelve, por cada voucher generado:

- Una **URL** con el formato `/p/{token}` — la misma URL que un cliente real abriría al escanear el QR de un ticket físico.
- Un **PNG** del código QR, listo para escanear con la cámara de un celular o abrir directamente la URL en el navegador.

4. Probar el circuito completo

Con el QR generado, seguí el mismo camino que seguiría un cliente real:

1. **Escaneá el QR** (o abrí la URL `/p/{token}` directo) desde un celular.
2. **Completá la participación** en la landing — carga de DNI (manual o por escaneo), consentimiento, confirmación.
3. Volvé al gestor y confirmá en **Panel** o en **Chances y vouchers** que la chance quedó registrada correctamente.

> **Es la forma más rápida de validar un despliegue nuevo**
>
> Después de instalar o publicar el sistema (Instalar en el servidor, Publicar con HTTPS), generar un QR de prueba y correr este circuito completo es el smoke test más directo: si el voucher de prueba se valida y la chance aparece en el gestor, el sistema está funcionando de punta a punta.

Los QR de prueba no cuentan como participación real

Al estar firmados con la clave de prueba, quedan claramente identificados como generados por este modo — no se mezclan con el padrón de clientes ni las chances reales de la campaña de cara al público.

Para el detalle del panel donde se ven las métricas de la campaña (incluidas las participaciones registradas por esta vía), ver Operar el gestor.

Manual de usuario

Este manual es para el **equipo de Caracol**: las personas que día a día administran la campaña "Sorteo 50 años" desde el gestor y las que atienden a un cliente que llama porque "el QR no le anda" o quiere saber si ganó. No hace falta ser técnico para seguirlo — cada paso está acompañado de una captura real del sistema.

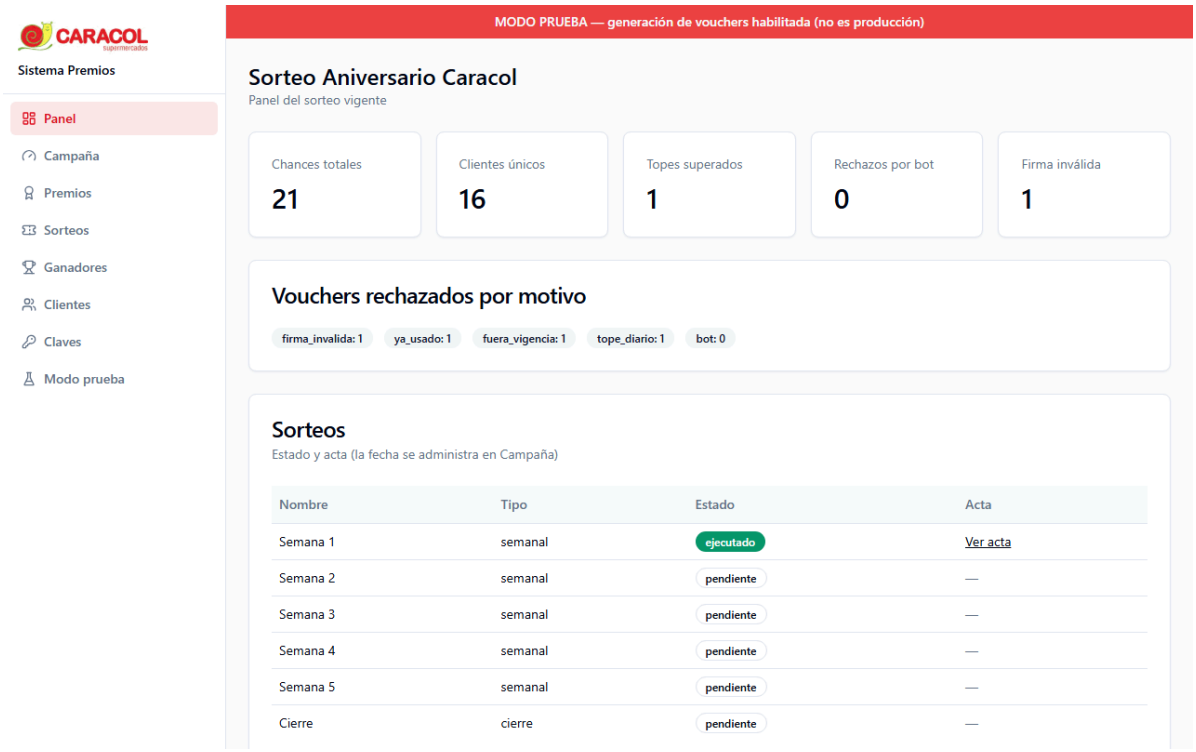
> **¿Buscás el porqué técnico?**
>
> Este manual explica **cómo usar** el sistema. Si en cambio querés entender **por qué** está diseñado así (arquitectura, decisiones de diseño, modelo de datos), esa documentación vive en Entender el sistema y Referencia técnica.

Las dos caras del sistema

Todo lo que hace el Sistema de Premios pasa por dos pantallas muy distintas, y este manual está organizado siguiendo esa misma división:

- **El celular del cliente.** Cuando alguien compra en Caracol, el ticket sale con un QR. Lo escanea, carga (o confirma) su DNI, y en segundos ya tiene una chance sumada para el sorteo de la semana. Esa experiencia — anónima, rápida, pensada para funcionar con mala señal en el salón — se llama la **landing**.
- **El gestor del operador.** Del otro lado, alguien de Caracol necesita cargar los premios, armar los sorteos, ejecutarlos, avisarle a los ganadores y entregarles el premio. Todo eso se hace desde el **gestor**, el panel web al que entrás con usuario y contraseña.

	Landing (celular del cliente)	**Gestor (panel del operador)**
¿Quién lo usa?	Cualquier cliente de Caracol	Personal de Caracol autorizado
¿Cómo se entra?	Escaneando el QR del ticket	Con usuario y contraseña
¿Qué se hace ahí?	Participar del sorteo	Administrar toda la campaña


CARACOL
Supermercados
Sistema Premios
Panel
Campaña
Premios
Sorteos
Ganadores
Clientes
Claves
Modo prueba
MODO PRUEBA — generación de vouchers habilitada (no es producción)
Sorteo Aniversario Caracol
Panel del sorteo vigente
Chances totales
21
Clientes únicos
16
Topes superados
1
Rechazos por bot
0
Firma inválida
1
Vouchers rechazados por motivo
firma_invalida: 1
ya_usado: 1
fuera_vigencia: 1
tope_diario: 1
bot: 0
Sorteos
Estado y acta (la fecha se administra en Campaña)
Nombre
Tipo
Estado
Acta
Semana 1
semanal
ejecutado
Ver acta
Semana 2
semanal
pendiente
—
Semana 3
semanal
pendiente
—
Semana 4
semanal
pendiente
—
Semana 5
semanal
pendiente
—
Cierre
cierre
pendiente
—

CARACOL
supermercados

Sorteo Aniversario Caracol

Sorteo por los 50 anios de Supermercados Caracol.

+1 chance

Ya tenés

1

chance acumuladas

Qué querés hacer → a qué página ir

Querés...	Andá a
Recorrer el gestor de punta a punta: premios, sorteos, ganadores, clientes	Operar el gestor
Entender qué ve el cliente en el celular en cada situación (QR vencido, ya usado, ganador reconocido, etc.)	La experiencia del cliente en el celular
Resolver una consulta de un cliente (su QR no anda, no sabe cuántas chances tiene, no le llegó el premio)	Dar soporte a clientes
Preparar una campaña nueva o programar los sorteos	Preparar una campaña
Ejecutar un sorteo semanal o el de cierre	Ejecutar un sorteo
Marcar entregas y gestionar ganadores	Gestionar ganadores y entregas
Generar vouchers de prueba para probar el circuito sin gastar tickets reales	Generar vouchers de prueba (QR)

Si sos nuevo en el equipo

Empezá por Operar el gestor: es un recorrido completo, sección por sección, con captura de cada pantalla. Después pasá por La experiencia del cliente en el celular para entender qué ve la persona del otro lado del mostrador — te va a servir muchísimo el día que tengas que dar soporte.

Operar el gestor

El **gestor** es el panel web donde el equipo de Caracol administra toda la campaña: la campaña en sí, los premios, los sorteos, los ganadores y sus entregas, el padrón de clientes y el rastreo de vouchers. Esta página lo recorre de punta a punta, sección por sección, con la captura de cada pantalla.

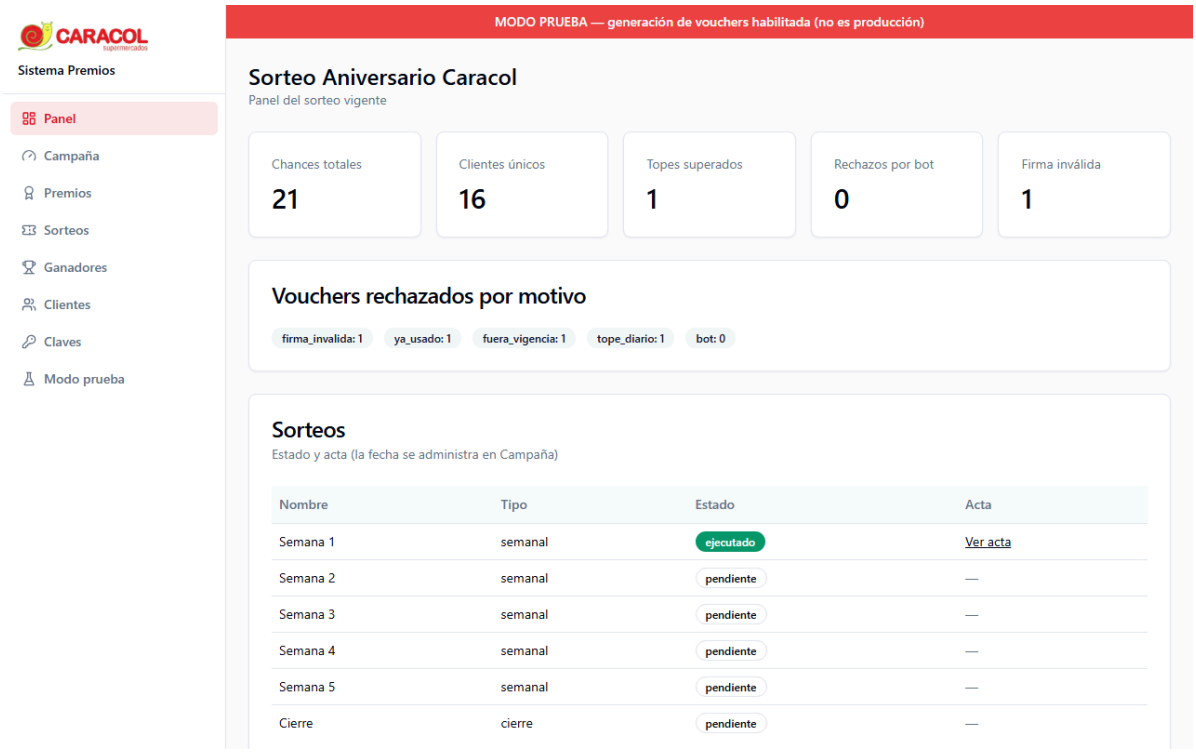
> **No hace falta memorizar nada**
>
> Cada sección de esta página corresponde a una pestaña o pantalla del gestor. Podés usarla como referencia: buscá la sección que necesitás en el momento en que la necesitás.

Acceso

Entrás al gestor con usuario y contraseña. Es un panel **autenticado** — no es la landing anónima que usa el cliente — así que solo entra personal de Caracol autorizado.

Panel

Al entrar, el panel te muestra un resumen del estado de la campaña de un vistazo:



Métrica	Qué significa
Chances totales	Cuántas participaciones válidas se registraron en toda la campaña (cada voucher escaneado y confirmado suma una).
Clientes únicos	Cuántas personas distintas (por DNI) participaron, sin importar cuántas chances tenga cada una.
Topes (alcanzados)	Cuántas veces un cliente llegó al límite diario de chances (5 por DNI por día de emisión) y un voucher se rechazó sin consumirse.
Rechazos por motivo	Desglose de los vouchers que no sumaron chance: firma inválida, ya usado, fuera de vigencia, tope diario. Es tu primer lugar para investigar un reclamo de "el QR no me anda".

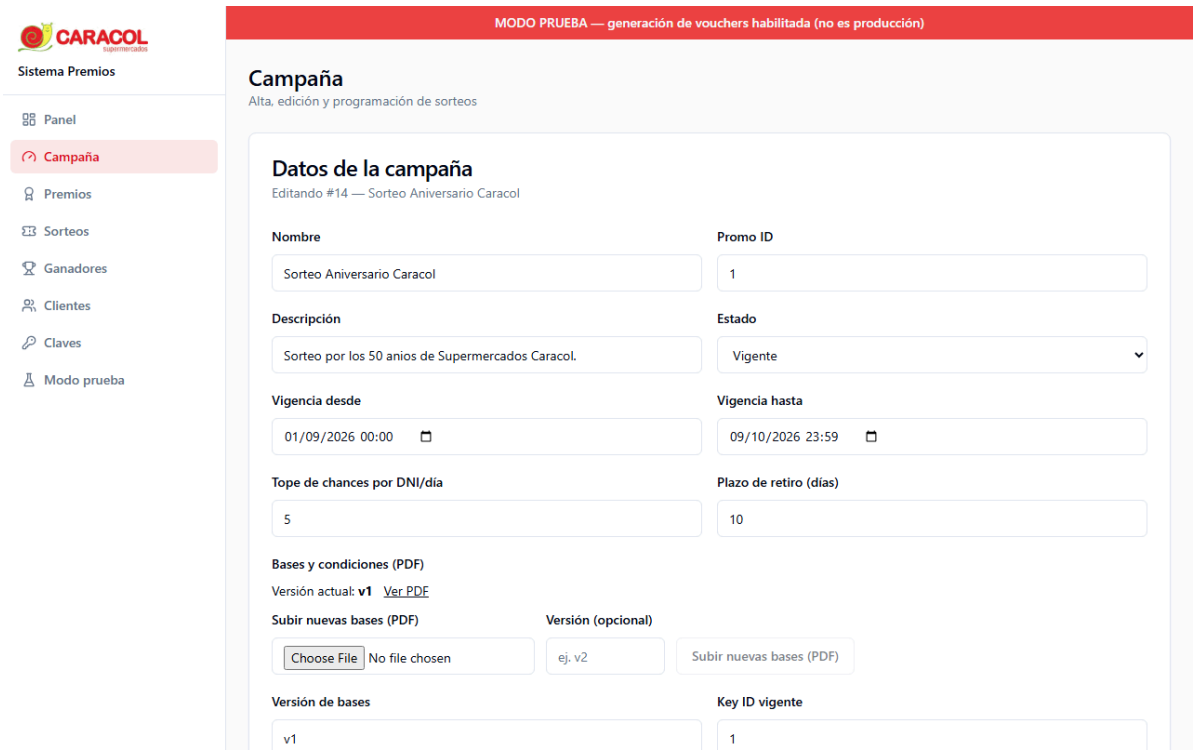
Tabla de sorteos	Los sorteos de la campaña con su estado (pendiente / ejecutado) y fecha.

Para dar soporte, empezá acá

Si un cliente dice que tuvo un problema al escanear, el desglose de **rechazos por motivo** te dice de entrada si fue un caso puntual o algo más generalizado. El detalle de cómo investigar un voucher específico está en Dar soporte a clientes.

Campaña

La pantalla de campaña muestra los datos generales de "Sorteo 50 años": vigencia, los textos que se ven en la landing, la regla de chances (1 voucher = 1 chance), el tope diario y la versión de las bases vigentes.

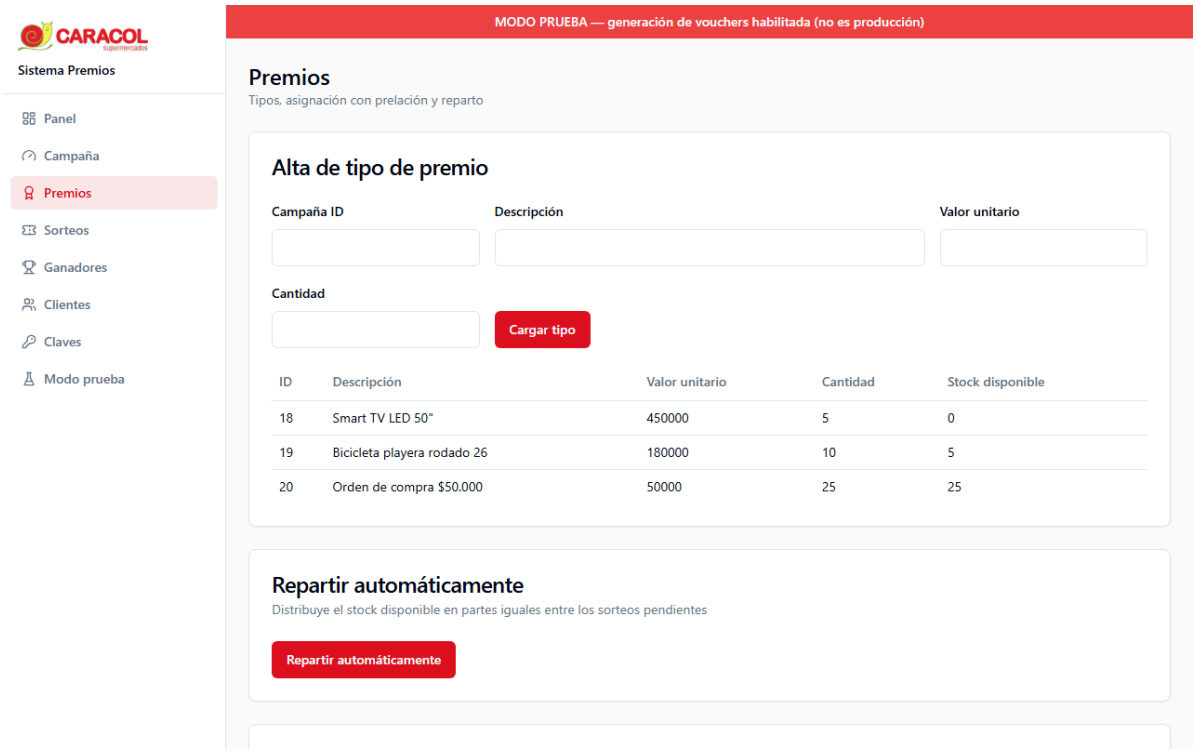


Solo puede haber una campaña vigente

El sistema garantiza que exista una única campaña activa a la vez. Los textos que cargués acá (nombre, descripción) se reflejan directo en la landing sin necesidad de tocar código — así se puede lanzar una promo nueva sin depender de un despliegue.

Premios

Acá se cargan los tipos de premio, se controla el stock disponible y se asignan a cada sorteo.



Los pasos habituales son:

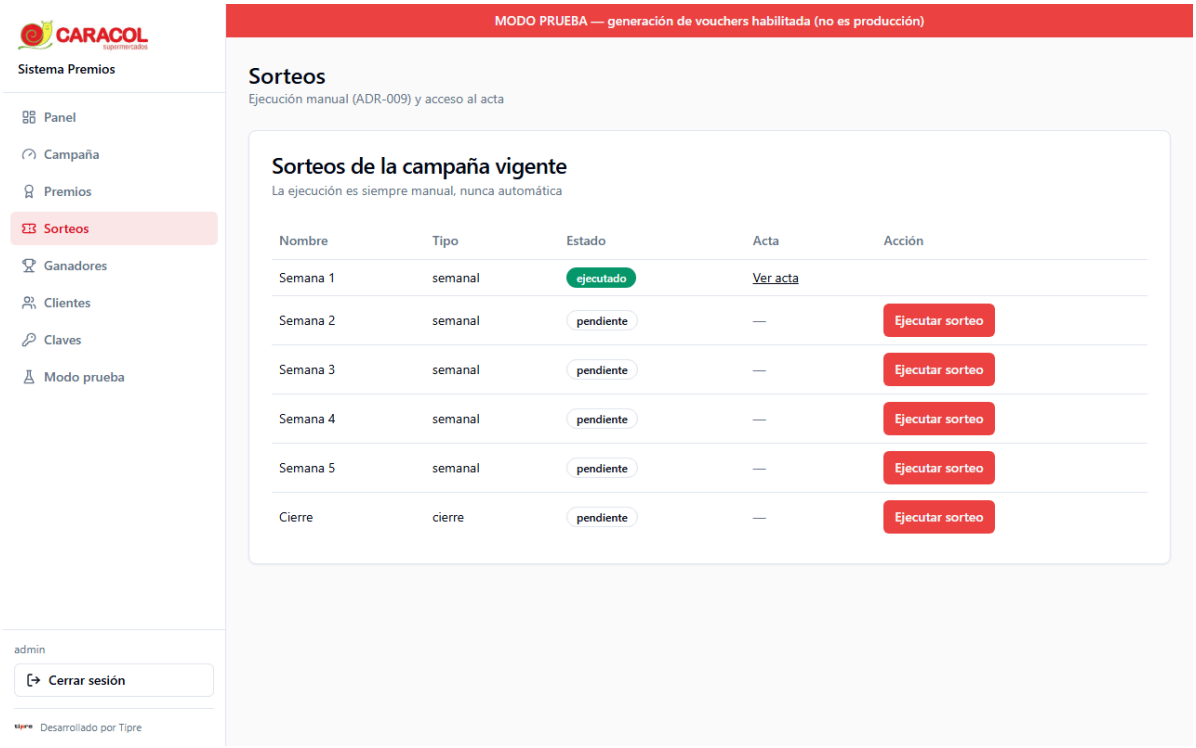
1. **Cargar un tipo de premio**: nombre y cantidad (por ejemplo, "Bicicleta playera × 10"). El sistema genera automáticamente cada unidad individual del lote.
2. **Asignar** unidades a un sorteo futuro, con su **prelación** — el orden en el que se van a repartir según el orden en que salga sorteada cada persona (primer sorteado se lleva el premio de prelación 1, y así sucesivamente).
3. **Repartir automáticamente**: reparte el stock en partes iguales entre las semanas como punto de partida. Es un atajo, no una decisión final — la asignación y la prelación definitiva siempre

las termina de ajustar el operador.

Cada unidad de premio recorre un ciclo de vida: **Disponible** → **Asignado** → **Sorteado** → **Otorgado** (entregado), con un estado terminal aparte, **No otorgado**, para cuando un premio no se puede entregar. El detalle completo de ese ciclo está en Premios y su ciclo de vida.

Sorteos

La lista de sorteos muestra cada sorteo semanal y el de cierre, con su estado y la acción de ejecutar.



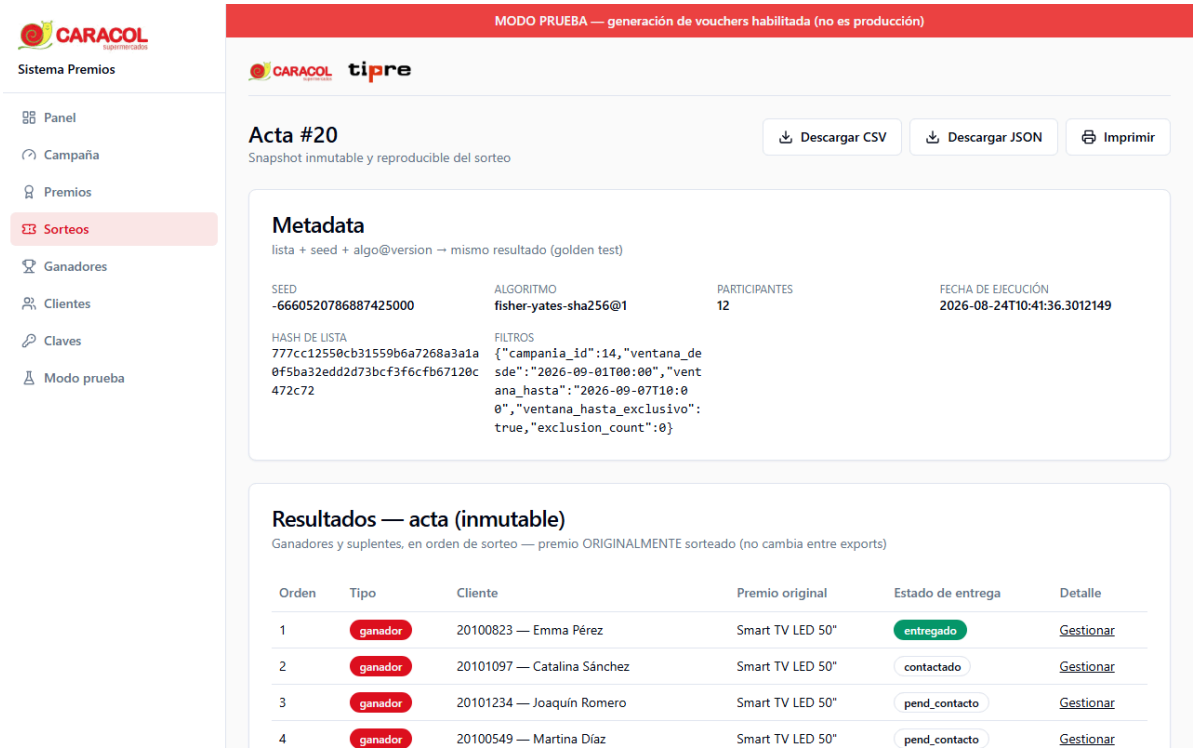
Ejecutar un sorteo es irreversible

El sorteo se ejecuta **manualmente** desde acá — nunca hay un disparador automático por hora o fecha, para que siempre haya una persona a cargo del momento exacto. Por eso el botón **Ejecutar** pide **doble confirmación** antes de correr: una vez ejecutado, el resultado queda en un acta inmutable. No existe un "reejecutar" — un error de campaña ya sorteada se corrige por decisión humana documentada, no reiniciando el sorteo.

Para el paso a paso completo (qué revisar antes de ejecutar, cómo elegir el sorteo correcto), ver Ejecutar un sorteo.

Acta

Cada sorteo ejecutado genera un **acta**: el registro inmutable y reproducible de esa ejecución. Contiene la semilla (`seed`) usada, el algoritmo y versión, un hash de la lista de elegibles, y el resultado ordenado con ganadores y suplentes.

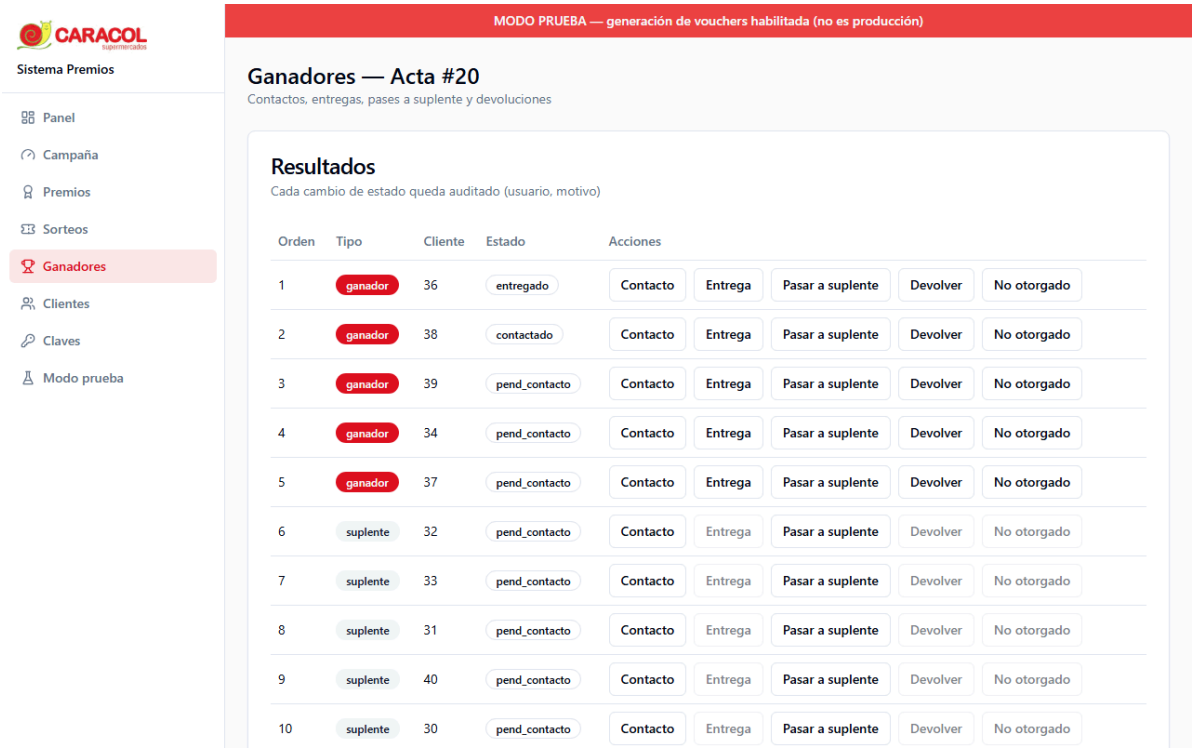


- **Reproducible**: la misma lista de elegibles + la misma semilla + el mismo algoritmo siempre dan el mismo resultado. Eso es lo que hace que el acta sea auditable — cualquiera puede reproducir el sorteo y llegar a los mismos ganadores.
- **Descargar**: podés bajar el acta en **CSV**, en **JSON** o mandarla directo a **imprimir**, según para qué la necesites (un reporte interno, un respaldo, o entregarla en papel).

> **El acta no cambia, aunque cambie la entrega**
>
> Lo que salió sorteado en el acta (el premio original de cada ganador) queda **congelado para siempre**, aunque después ese premio pase a un suplente o se devuelva al stock. El acta separa claramente "lo sorteado" (inmutable, tu registro legal del sorteo) de "el estado de entrega" (que sí evoluciona con el tiempo). Si exportás la misma acta dos veces en momentos distintos, la parte del sorteo es idéntica; la parte de entrega puede diferir.

Ganadores

Desde la pantalla de ganadores de un acta gestionás todo el circuito de contacto y entrega de cada premio.



Cada ganador tiene un estado:

Estado	Qué significa
Pend. contacto	Todavía no se lo contactó.
Contactado	Ya se avisó al ganador, está pendiente de coordinar o concretar la entrega.
Entregado	El premio ya se le entregó. Estado terminal para ese ganador.

Y las acciones disponibles sobre cada uno:

- **Contacto**: registra que se estableció contacto con el ganador (arranca a correr el plazo de retiro).

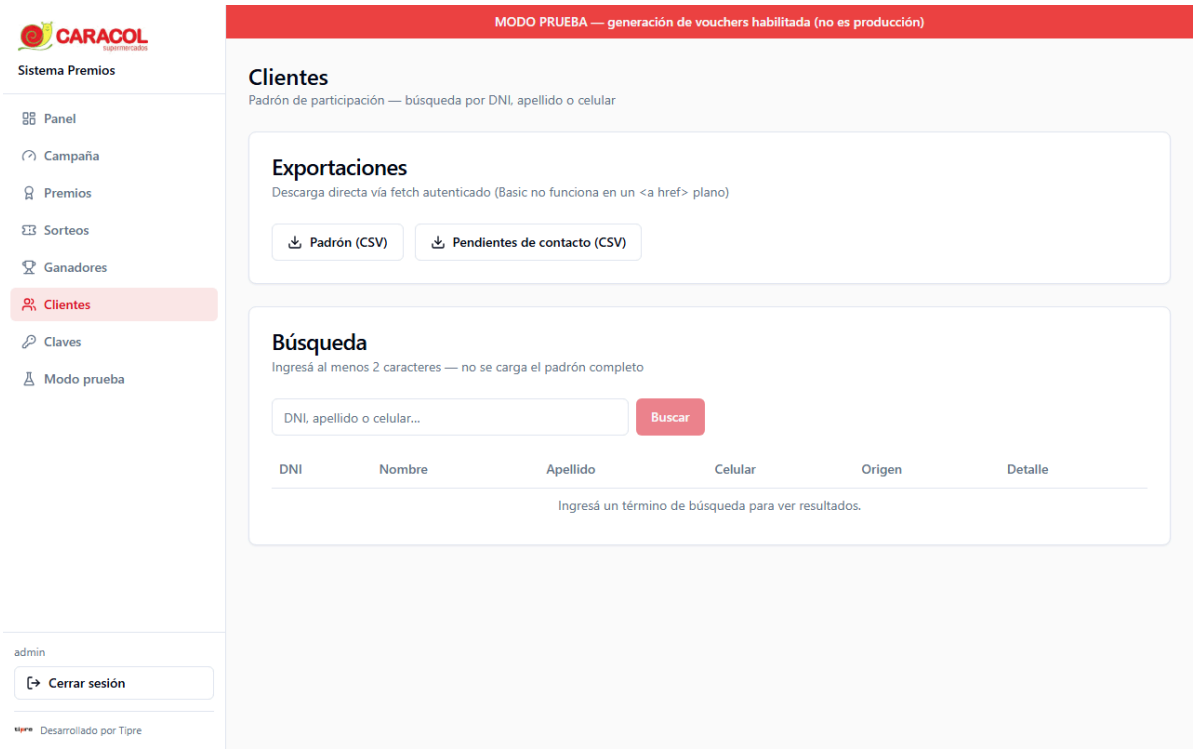
- **Entrega**: registra la entrega del premio. **Siempre exige verificar el DNI de la persona contra el DNI que figura en el acta** — nunca se entrega un premio sin confirmar que quien lo retira es efectivamente el ganador.
- **Pasar a suplente**: si el titular está incontactable o rechaza el premio, esta acción lo pasa al siguiente suplente de la misma acta, en el orden en que salieron sorteados.
- **Devolver**: devuelve la unidad de premio al stock disponible, para poder asignarla a un sorteo futuro (por ejemplo, si se agotaron los suplentes antes del cierre de la campaña).
- **No otorgado**: marca la unidad como definitivamente no entregada. Es un estado terminal, para usar cuando ya no tiene sentido seguir intentando (por ejemplo, después del cierre de la campaña).

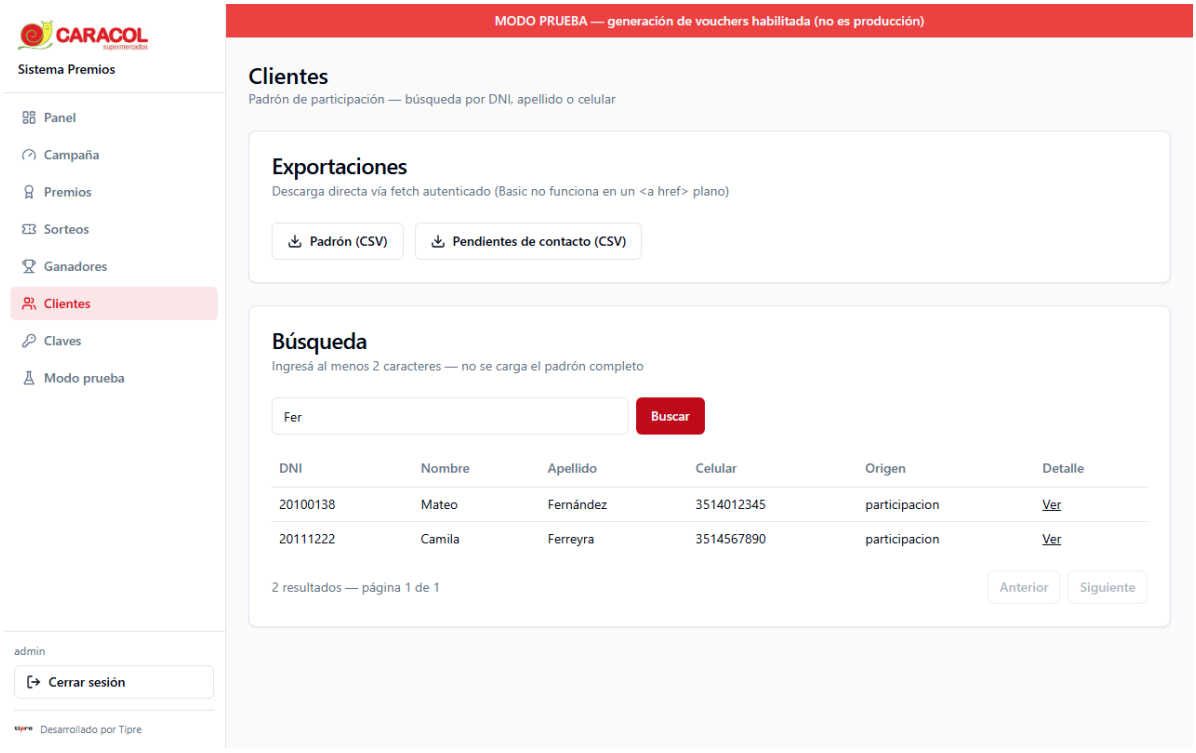
> ⚠ **Plazo de retiro vencido**
>
> Cada campaña define un plazo de retiro (10 días corridos desde el primer contacto). Si ese plazo venció, el gestor te lo va a advertir al intentar registrar una entrega y te va a pedir una confirmación explícita antes de dejarte continuar — nunca lo bloquea del todo, porque la decisión final es del operador, pero tampoco lo deja pasar por alto en silencio.

Para el flujo operativo completo de principio a fin, ver Gestionar ganadores y entregas.

Clientes

El buscador de clientes te deja encontrar a cualquier persona del padrón por DNI, apellido o celular.



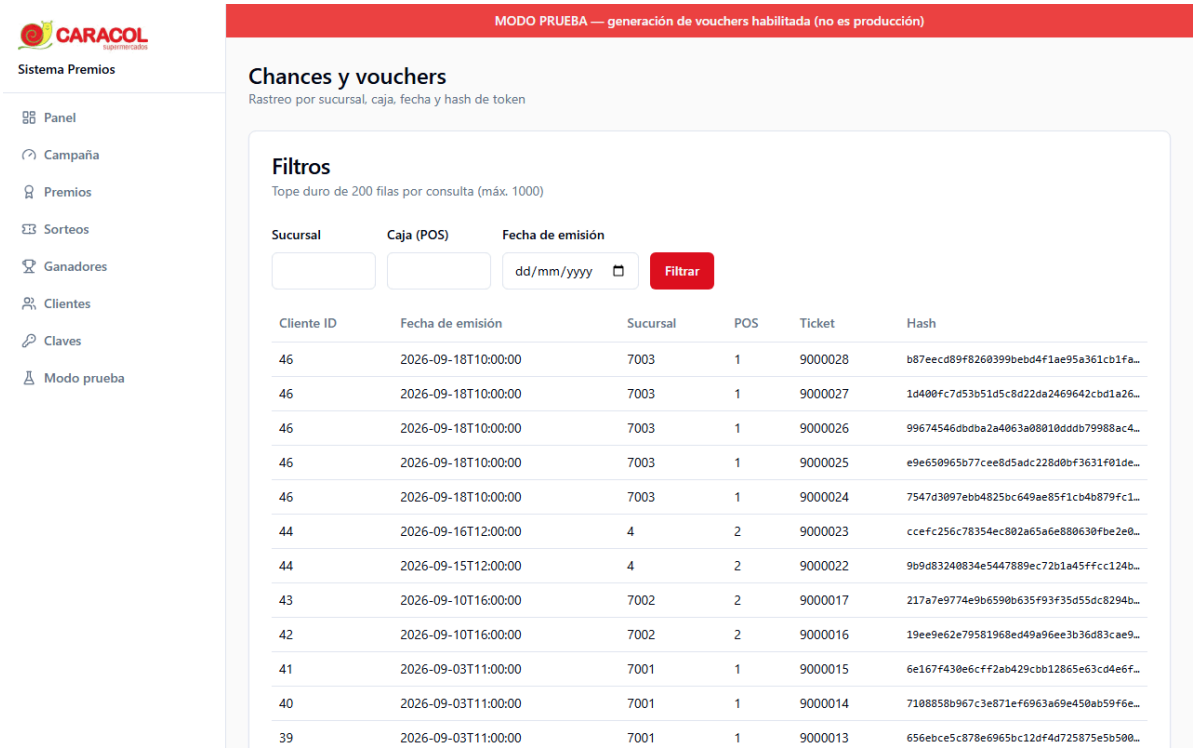


Desde acá podés:

- **Buscar** un cliente puntual (hace falta escribir un término de búsqueda; la pantalla no carga todo el padrón de una — con decenas de miles de clientes eso sería lento e innecesario).
- **Corregir datos** de un cliente cuando detectás un error (por ejemplo, un celular mal cargado que impide contactar a un ganador). Ver el detalle de qué se puede corregir y qué no en Dar soporte a clientes.
- **Exportar a CSV** el padrón o el resultado de una búsqueda, para reportes o cruces por fuera del sistema.

Chances y vouchers

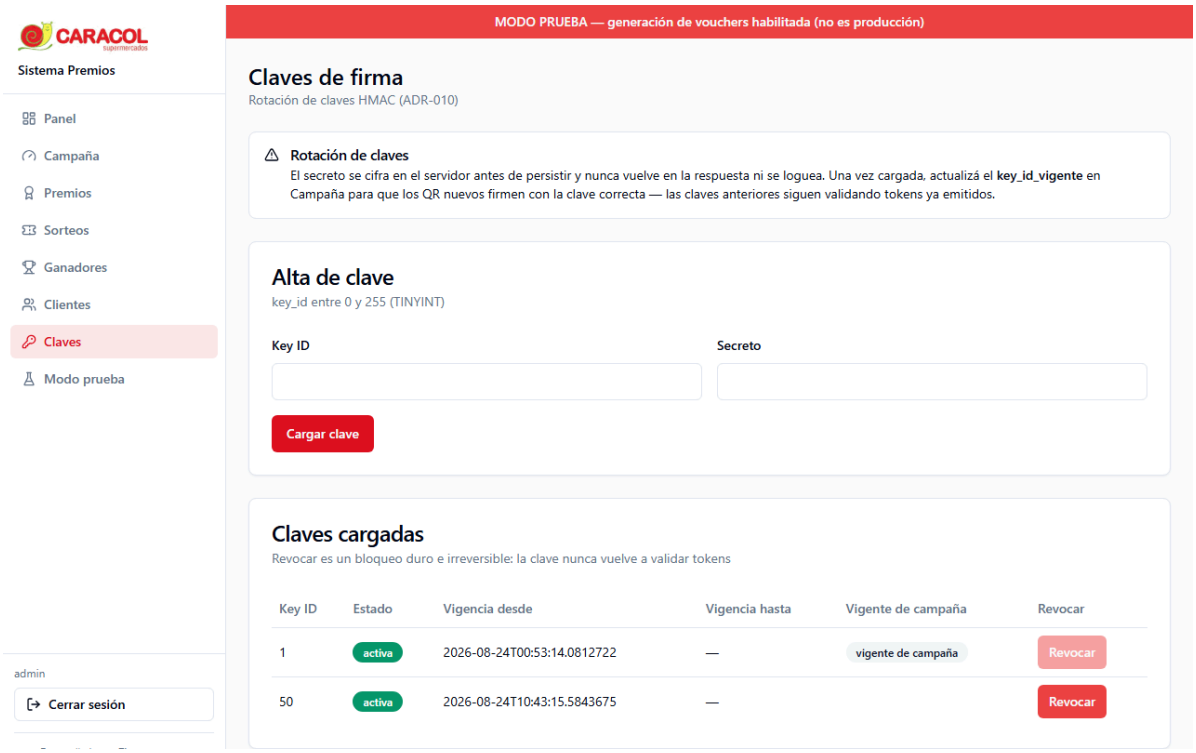
La pantalla de rastreo te deja seguir el rastro de cualquier chance o voucher: por sucursal, caja, fecha o estado.



Es la herramienta indicada cuando necesitás confirmar si un voucher puntual se validó, se rechazó (y por qué motivo) o directamente nunca llegó a procesarse — clave para responder consultas de soporte con precisión en vez de a ojo.

Claves

Las claves de firma son las que el sistema usa para validar que el QR de cada ticket sea genuino (HMAC). Cada clave tiene un identificador (`keyId`) y un estado de vigencia.

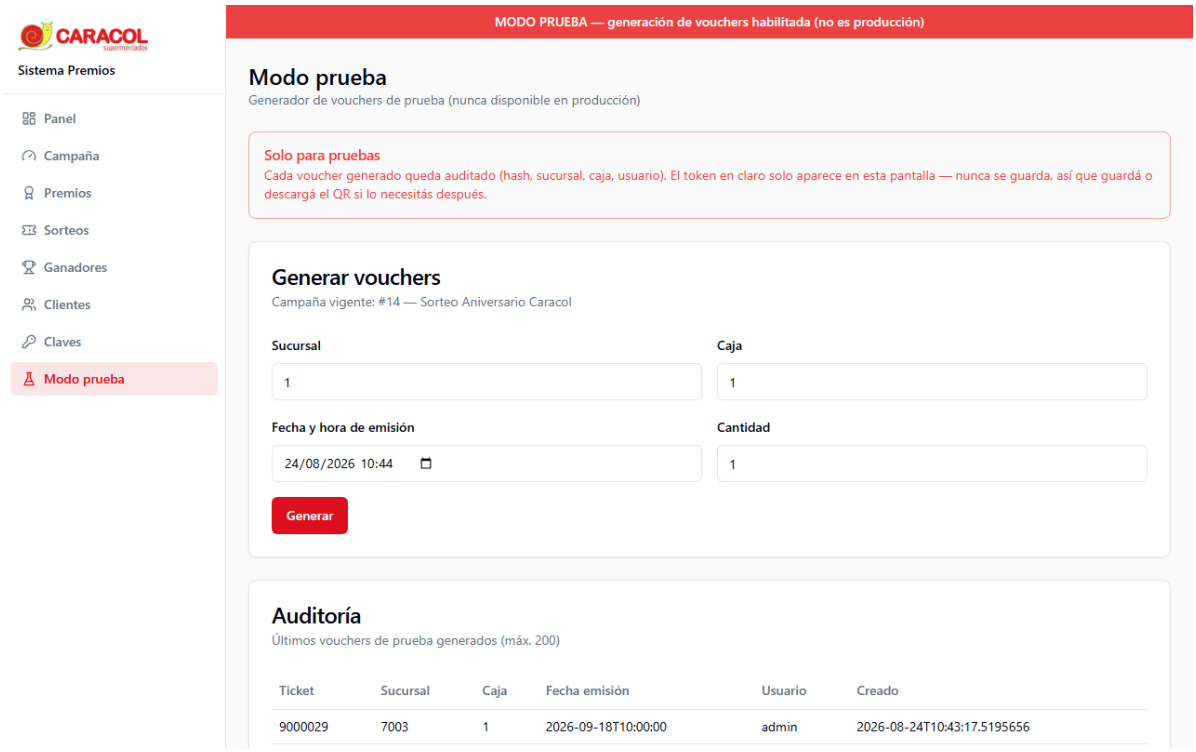


Rotación sin cortar el servicio

Se pueden tener varias claves registradas a la vez, cada una con su `keyId` . Esto permite rotar la clave de firma sin que los vouchers ya impresos con la clave anterior dejen de funcionar — el sistema valida cada voucher contra la clave que indica su propio `keyId` , no contra "la última".

Modo prueba

El generador de vouchers de prueba te permite crear QRs válidos sin necesidad de una impresora de ticket real, firmados con una clave de prueba separada de la clave de producción.



Es la herramienta para validar el circuito completo (caja → QR → landing → gestor) antes de una salida en vivo, o para reproducir un caso puntual sin usar un ticket de un cliente real. El paso a paso está en Generar vouchers de prueba (QR).

La experiencia del cliente en el celular

Cuando un cliente de Caracol escanea el QR de su ticket, entra a la **landing**: una página anónima, pensada para abrir rápido incluso con mala señal en el salón. Lo que ve ahí depende de la situación concreta de ese voucher. Esta página muestra, con captura real, **todos los casos conocidos** — para que el equipo de soporte sepa exactamente qué está viendo el cliente del otro lado del teléfono en cada situación.

> **Por qué existen estos casos**
>
> Cada pantalla responde a una pregunta que el sistema se hace, en orden, apenas se abre el QR: ¿la firma es válida?, ¿el voucher ya se usó?, ¿la promoción está vigente?, ¿el DNI ya está en el padrón?, ¿llegó al tope de hoy? El detalle técnico de esa cadena de decisiones está en Los casos de participación (A–E + tope); acá nos quedamos con qué ve el cliente y por qué.

Caso A — El QR no es válido

Si la firma del voucher no es válida (un QR dañado, mal impreso, o directamente falso), el cliente ve un error genérico. **A propósito no da ninguna pista** de cuál fue el problema real — esa información queda registrada internamente, nunca expuesta hacia afuera, para no darle señales útiles a alguien que intente falsificar un voucher.

CARACOL
supermercados

Sorteo Aniversario Caracol

Sorteo por los 50 anios de Supermercados Caracol.

No pudimos validar el voucher

No pudimos validar tu voucher.

Caso B — El voucher ya participó

Si ese mismo voucher ya se usó antes (aunque sea el mismo cliente reintentando), el sistema lo detecta y avisa que ya participó, mostrando la fecha y hora de esa participación original. **Nunca muestra datos personales** en esta pantalla.

CARACOL
supermercados

Sorteo Aniversario Caracol

Sorteo por los 50 anios de Supermercados Caracol.

Este voucher ya participó

Registrado el 24/8/2026, 10:43:11

Un doble toque no genera doble chance

Es normal que un cliente, con mala señal, toque "participar" dos veces. El sistema garantiza que un mismo voucher sume **una sola** chance aunque llegue el envío duplicado — el segundo intento cae directo en el Caso B.

Caso C — La promoción no está vigente

Si el voucher es válido y no fue usado, pero su fecha de emisión cae fuera de la vigencia de la campaña (por ejemplo, un ticket muy viejo, o escaneado antes de que arranque la promo), no suma chance — pero el sistema aprovecha para sumar al cliente al **padrón** para próximas promociones.

CARACOL
supermercados

Sorteo Aniversario Caracol

Sorteo por los 50 anios de Supermercados Caracol.

Esta promoción ya no está vigente

Dejanos tus datos y te avisamos de las próximas promociones.

DNI

Confirmá tu DNI

Nombre

Apellido

Celular

Si el cliente todavía no estaba en el padrón, completa un alta corta (DNI + datos básicos). Al confirmar, ve la pantalla de cierre:

CARACOL
supermercados

Sorteo Aniversario Caracol

Sorteo por los 50 anios de Supermercados Caracol.

¡Listo!

Ya quedaste registrado en nuestra base. Te vamos a avisar de las próximas promociones.

Caso D — Cliente nuevo

Si el voucher es válido, no fue usado y la promoción está vigente, el sistema pide el DNI:

CARACOL
supermercados

Sorteo Aniversario Caracol

Sorteo por los 50 anios de Supermercados Caracol.

Ingresá tu DNI

Lo necesitamos para sumar tu chance al sorteo.

DNI

Confirmá tu DNI

Confirmar

Escanear DNI

El DNI se puede **tipear** (con reingreso de confirmación para evitar errores de dedo) o **escanear con la cámara** el código de barras del documento físico, tocando "Escanear DNI". El escaneo es más rápido y evita errores de tipeo — conviene sugerirlo siempre que el cliente tenga el DNI físico a mano.

Si ese DNI todavía no está en el padrón, es un cliente nuevo: se le pide completar sus datos.

supermercados

Sorteo Aniversario Caracol

Sorteo por los 50 anios de Supermercados Caracol.

Ya casi terminás

Completá tus datos para sumar tu primera chance.

DNI

20111222

Nombre

Camila

Apellido

Ferreyra

Celular

3514567890

Fecha de nacimiento (opcional)

DD/MM/AAAA

El **celular es obligatorio** (es el canal para poder contactarlo si gana), el email es opcional. Al confirmar, el alta y la chance se registran juntas, y el cliente ve la confirmación:

CARACOL
supermercados

Sorteo Aniversario Caracol

Sorteo por los 50 anios de Supermercados Caracol.

+1 chance

Ya tenés

1

chance acumuladas

Caso E — Cliente ya registrado

Si el DNI que ingresa (o escanea) ya está en el padrón, no hace falta volver a pedirle los datos: el sistema lo reconoce y confirma su participación en un toque.

CARACOL
supermercados

Sorteo Aniversario Caracol

Sorteo por los 50 anios de Supermercados Caracol.

¡Hola Camila Ferreyra!

Ya te reconocimos. Confirmá para sumar tu chance.

Ticket N.º 9000023 · Sucursal 4

Emitido el 16/9/2026, 12:00:00

Revisá tus datos

Email: c*****@ejemplo.com

Celular: ******7890

Editar celular

¿Tu nombre está mal escrito? Escaneá tu DNI para corregirlo.

Al confirmar, se suma la chance:

CARACOL
supermercados

Sorteo Aniversario Caracol

Sorteo por los 50 anios de Supermercados Caracol.

+1 chance

Ya tenés

2

chances acumuladas

Por qué a veces lo saluda por su nombre sin pedirle el DNI

Hay una segunda forma de llegar a esta misma pantalla, y es la que más preguntas genera en soporte: en algunos casos el cliente ve el saludo "¡Hola {nombre}!" **antes** de que se le haya pedido el DNI. Esto es el **reconocimiento del dispositivo**: si esa persona ya participó antes desde ese mismo celular, el sistema recuerda quién es sin necesidad de volver a pedirle el DNI cada vez.

Importante para entender qué es y qué no es esta función:

- **El DNI nunca se guarda en el celular.** Lo que queda guardado en el teléfono es un identificador aleatorio sin significado propio, que solo el servidor puede traducir a una persona — nunca el documento en sí.
- **Reconocer no es participar.** Aunque el sistema lo salude por su nombre, siempre exige que el cliente confirme explícitamente y que el voucher escaneado sea válido. Nunca se suma una chance en automático solo por reconocer el dispositivo.
- **"No soy yo" existe por el celular compartido.** Como un teléfono puede ser de toda la familia o el de la caja del súper, la pantalla de saludo siempre tiene una opción para decir que no es esa persona, que borra el reconocimiento guardado y vuelve al flujo normal de pedir el DNI.
- **Es "memoria del dispositivo", no del DNI.** Una misma persona puede ser reconocida desde el celular de casa y desde el del trabajo por separado, sin que uno pise al otro.

Tope diario alcanzado

Cada DNI tiene un límite de **5 chances por día**, contado por la fecha de emisión del ticket (no por el día en que se escanea). Si un cliente ya llegó a ese límite y escanea un sexto voucher del mismo día, el sistema lo rechaza **sin gastar el voucher** — el ticket sigue disponible para escanearlo al día siguiente si corresponde a otra fecha de emisión.

CARACOL
supermercados

Sorteo Aniversario Caracol

Sorteo por los 50 anios de Supermercados Caracol.

Ya alcanzaste el límite de hoy

Alcanzaste el límite de participaciones de hoy con este DNI.

El tope protege, no penaliza

Como el voucher no se consume, no hay ninguna chance "perdida": el tope solo pausa nuevas participaciones de ese DNI por ese día. Es una medida antifraude para evitar que una sola persona concentre desproporcionadamente las chances de un sorteo.

Resumen rápido

Caso	Qué pasó con el voucher	Qué ve el cliente
A	Firma inválida	Error genérico, sin pistas
B	Ya se usó antes	"Ya participaste", con fecha y hora
C	Fuera de vigencia	Sin chance, pero se suma al padrón
D	Válido, DNI nuevo	Formulario de alta → chance
E	Válido, DNI ya registrado	Saludo + confirmación en un toque → chance
Tope	Válido, pero ya llegó al límite del día	Aviso de límite diario, sin gastar el voucher

Para resolver dudas de un cliente sobre alguno de estos casos usando el gestor, ver Dar soporte a clientes.

Dar soporte a clientes

Esta página es para cuando un cliente de Caracol te escribe, te llama o se acerca al mostrador con una duda o un problema sobre el sorteo. Cada pregunta frecuente incluye cómo resolverla **usando el gestor**.

> **Antes de responder, mirá el panel**
>
> El panel del gestor tiene el desglose de **rechazos por motivo** — muchas veces te ahorra tener que investigar caso por caso, porque te dice de entrada si un problema es puntual o algo más generalizado.

Mensaje que ve el cliente → qué significa → qué hacer

El cliente te dice / te muestra	Qué significa	Qué hacer
"No pudimos validar el voucher"	**Caso A** — la firma del QR no es válida (dañado, mal impreso, o falso)	Pedile el ticket físico y verificá el estado real del voucher en Chances y vouchers. Si el ticket es legítimo y sigue fallando, escalalo — puede ser un problema de impresión en esa caja.
"Este voucher ya participó"	**Caso B** — ese QR concreto ya sumó una chance antes	Es normal, incluso con el mismo cliente: pasa si tocó "participar" dos veces con mala señal. Buscá el voucher en Chances y vouchers para confirmar fecha y hora de esa participación.

"Dice que la promo no está vigente"	**Caso C** — el ticket tiene una fecha de emisión fuera del período de la campaña	Confirmá la fecha de emisión del ticket contra la vigencia de la campaña en Campaña. El cliente igual queda sumado al padrón para futuras promos, aunque esta vez no sume chance.
"Alcancé el límite de hoy"	Llegó al **tope diario** (5 chances por DNI por día de emisión)	Es una medida antifraude, no un error. El voucher **no se gastó**, así que si corresponde a otra fecha de emisión lo puede escanear sin problema.
"Me dice que ya tengo N chances"	Participación exitosa (caso D o E)	Todo en orden — podés confirmarle el total buscándolo en Clientes.

Preguntas frecuentes

Un cliente dice que su QR no anda. ¿Cómo lo reviso?

Primero pedile que te diga (o te muestre) **exactamente qué mensaje** le apareció — con eso ya sabés en qué caso cayó (mirá la tabla de arriba: A, B o C son los tres motivos por los que un voucher no suma chance).

Después, con el ticket físico a mano, andá a Chances y vouchers en el gestor y buscá ese voucher por sucursal, caja o fecha. Ahí vas a ver el estado real que quedó registrado y, si fue rechazado, el motivo interno — que suele ser más preciso que lo que el cliente recuerda haber leído en la pantalla.

Si el voucher aparece como rechazado por firma inválida (Caso A) y el ticket es evidentemente legítimo, puede tratarse de un problema de impresión de esa caja puntual — vale la pena mirar si hay más rechazos del mismo tipo concentrados en la misma sucursal o caja desde el panel.

Quiero saber cuántas chances tiene un cliente

Andá a Clientes y buscalo por DNI, apellido o celular. La ficha del cliente te muestra su total de chances acumuladas en la campaña.

Un ganador no aparece en la lista, o no sé cómo entregarle el premio

Entrá al acta del sorteo correspondiente desde Sorteos → el sorteo ya ejecutado → **Acta**, y ahí abrí Ganadores. Vas a ver a todos los ganadores y suplentes de esa acta con su estado (pend. contacto / contactado / entregado).

Para entregar un premio, usá la acción **Entrega** desde esa misma pantalla. El sistema te va a pedir **verificar el DNI** de la persona que retira contra el DNI que figura en el acta — nunca se entrega sin esa verificación, para asegurarnos de que el premio va a manos del ganador real y no de otra persona que se presente con el mismo aviso.

Si el titular está incontactable o no puede retirarlo, usá **Pasar a suplente**: el sistema avanza automáticamente al siguiente suplente sorteado en esa misma acta, respetando el orden.

El cliente cargó mal su celular o su nombre. ¿Se puede corregir?

Sí, hay dos caminos:

- **El cliente lo corrige él mismo desde el celular**, presentando su DNI de nuevo (por seguridad, nadie puede editar los datos de otra persona sin tener su DNI a mano — ni siquiera si el teléfono lo "reconoce"). El celular y el email siempre se pueden corregir así. El nombre, apellido, sexo y fecha de nacimiento **solo son editables si el alta original fue manual**; si el alta fue por **escaneo del DNI**, esos datos de identidad quedan de solo lectura (se toma el documento como la fuente de verdad) — para cambiarlos, el cliente tiene que volver a escanear su DNI.
- **Vos lo corregís directamente desde el gestor**, en Clientes: buscá al cliente y corregí el dato erróneo. Es el camino más rápido cuando el cliente te llama para avisar el error, en vez de mandarlo de vuelta a la landing.

El celular es crítico para los ganadores

Si el cliente ganó y su celular está mal cargado, el premio queda **imposible de entregar** por ese canal. Ante cualquier duda sobre un dato de contacto de un ganador, priorizá corregirlo cuanto antes.

El cliente no puede escanear su DNI con la cámara

El escaneo del código de barras del DNI es una **comodidad**, nunca un requisito: la carga manual del DNI (tipeado, con reingreso de confirmación) está siempre disponible como alternativa. Si el escaneo no anda, decile simplemente que lo tipee.

Motivos típicos de que el escaneo falle: poca luz, el DNI con la banda dañada o muy gastada, o la cámara del celular sucia o empañada. Sugerile acercar el documento, buscar mejor luz, o directamente pasar a carga manual si insiste sin éxito — no vale la pena que el cliente pierda tiempo con el escaneo si el tipeo funciona igual de bien.

¿Qué le digo a un cliente que no ganó?

Cada sorteo saca ganadores y suplentes de entre todas las chances elegibles de esa ventana — cuantas más chances acumuladas, más veces entra en juego, pero no hay garantía de premio en ningún sorteo puntual. Si el cliente pregunta, lo más simple y honesto es explicarle que el sorteo entre semana no fue su turno, pero que **sus chances acumuladas siguen sumando** para los sorteos siguientes de la campaña (salvo que ya haya sido ganador antes: quien ya ganó un premio en la campaña queda excluido de los sorteos siguientes, para repartir entre más gente).

Si quiere confirmar cuántas chances tiene acumuladas, buscalo en Clientes.

Sistema de Premios

Sorteos por compra que se integran a las cajas que ya tenés, sin tocar el POS.

Tipre presenta un sistema para montar campañas de sorteo por compra: cada ticket imprime un cupón con QR, el cliente lo escanea desde su celular y participa. Sin apps, sin sincronización entre cajas y web, y con un padrón de clientes que queda para el negocio después de la campaña.

El problema

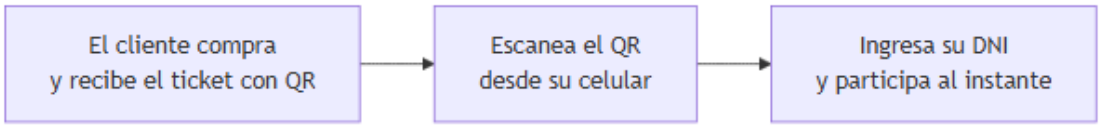
Montar un sorteo por compra suele exigir que cada caja "hable" en tiempo real con un sistema central: qué ticket es válido, si ya participó, si está vigente. Esa sincronización cajas ↔ web es frágil, cara y lenta de instalar. Un corte de red en una sucursal deja la promoción sin funcionar.

La solución

El QR del ticket es **autocontenido**: todo lo que la web necesita del ticket viaja firmado dentro del token. La caja no consulta nada; solo imprime. La página valida la firma y resuelve la participación por sí sola.

> ✓ **El diferencial**
>
> Un solo contrato de integración —el token firmado— y nada más. El sistema se enchufa a las cajas existentes sin modificar el punto de venta y sin depender de una conexión permanente entre la sucursal y el servidor.

Cómo funciona



Tres pasos, sin fricción: comprar, escanear, participar. Si es un cliente nuevo, se registra en un formulario breve; si ya está en el padrón, confirma en un toque.

Qué gana el negocio

Beneficio	Qué significa
Participación instantánea	El cliente juega desde su celular, sin descargar ninguna app, apenas sale de la caja.
Padrón propio de clientes	Cada participación construye una base de clientes con consentimiento: un activo de marketing que queda para el negocio cuando la campaña termina.
Sorteos transparentes y auditables	Cada sorteo genera un acta inmutable y reproducible, clave ante escribano o legales.
Antifraude incorporado	Tope de participaciones por documento y por día, verificación anti-robots y señales de concentración sospechosa en el panel.
Corre en tu servidor	Sin dependencia de la nube, con respaldo diario. Los datos de tus clientes quedan en casa.
Reutilizable	Branding configurable: el mismo sistema sirve para la próxima campaña, sin rehacerlo.

Por qué es confiable

- **Acta reproducible.** Cada sorteo queda registrado como una foto inmutable: la lista de participantes elegibles, el resultado y la semilla que lo generó. A partir de esos datos, el sorteo se puede volver a calcular y da exactamente el mismo resultado. Es la prueba que se necesita ante un escribano o un reclamo legal.
- **Antifraude de fábrica.** El sistema controla un tope de participaciones por documento y por día, valida criptográficamente cada cupón e informa señales de uso anómalo (un mismo documento, una misma conexión o una misma caja concentrando cupones).

- **Los datos, en casa del cliente.** El sistema corre en un servidor propio del negocio, sin depender de servicios de nube, con backup diario. La base de clientes es del negocio, no de un tercero.

El producto, hoy

No es una idea: es un sistema **ya construido y operativo**, en tres piezas:

1. **Backend de participación** — valida cada cupón, arma el padrón, registra las chances y ejecuta el motor de sorteo con su acta.
2. **Landing móvil** — la página que abre el QR, pensada para cargar al instante con la conexión de un salón de ventas.
3. **Gestor web** — el panel de administración: campañas, premios, sorteos, ganadores, clientes y reportes.

CARACOL
supermercados

Sorteo Aniversario Caracol

Sorteo por los 50 anios de Supermercados Caracol.

¡Hola Camila Ferreyra!

Ya te reconocimos. Confirmá para sumar tu chance.

Ticket N.º 9000023 · Sucursal 4

Emitido el 16/9/2026, 12:00:00

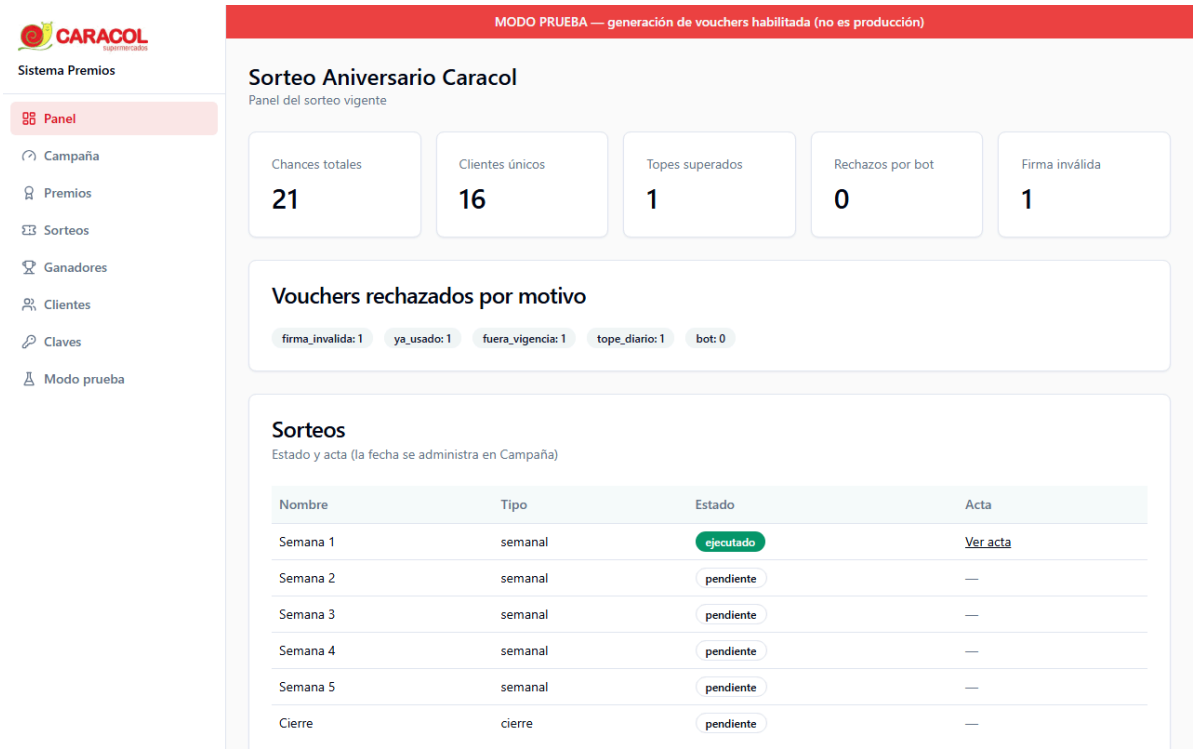
Revisá tus datos

Email: c*****@ejemplo.com

Celular: ******7890

Editar celular

¿Tu nombre está mal escrito? Escaneá tu DNI para corregirlo.



> **Campaña de referencia**
>
> El sistema está desplegado para el **Sorteo 50 años de Supermercados Caracol**: 100 premios, sorteos semanales y un sorteo de cierre, sobre las cajas existentes de la cadena.

Cierre

Tipre provee el sistema completo y lo pone a funcionar sobre la infraestructura de cajas que el negocio ya tiene. Terminada una campaña, queda el padrón de clientes como activo propio y el sistema listo para la siguiente: mismo motor, nuevo branding, sin volver a empezar.