Manual de Usuario y Operación — Emisor Visa

Proyecto: CCVI – Emisor de Tarjetas (Visa) **Repositorio:** `ccvi-visa-emisor` **Estado:** Versión inicial formal (candidata a PDF) **Última actualización:** 2025-10-14

---

1. Propósito

Este documento describe el funcionamiento integral del sistema Emisor Visa: arquitectura, instalación, uso operativo (frontend), consumo de la API REST, modelo de datos, seguridad, mantenimiento y CI/CD. Su formato sigue un estilo formal similar a `normalizacion-entrada.md`.

El objetivo es servir como referencia única para:

- Operadores / Área de negocio
- Integradores (consumo de API)
- Desarrolladores internos
- Equipo de Infra / DevOps
- Soporte y mantenimiento

---

2. Alcance

Incluye los componentes:

1. **API Backend** (Express 5, PostgreSQL) – Rutas de tarjetas, autorizaciones, pagos, salud.
2. **Frontend Web** (React + Vite + Nginx) – UI para gestión de tarjetas y transacciones.
3. **Base de Datos** (PostgreSQL, esquema `emisor`) – Tablas `tarjetas` y `transacciones`, triggers de negocio.
4. **Infraestructura** (Docker Compose) – Orquestación de servicios `db`, `api`, `web`.
5. **Registro y Observabilidad** – Logs con `pino/pino-http`, health & readiness.
6. **Seguridad** – API Key, HMAC de CVV, redacción de logs.
7. **Idempotencia** – Claves para operaciones repetibles (autorizaciones/pagos).
8. **CI/CD** – GitHub Actions (lint + unit, integración con DB).

Fuera de alcance (actual):

- Mecanismos avanzados antifraude
- Escalado horizontal automático
- Sistema de alertas externas

---

3. Visión General de Arquitectura

```
[Browser] --(HTTP :8080)--> [Nginx SPA] --(proxy /api/*)--> [API Express :3000]
                                          \--> /healthz, /readyz, /metadata
[API Express] --(pg pool)--> [PostgreSQL :5432]
```

- El frontend compila a estáticos (`dist/`) servidos por Nginx.
- Proxy Nginx reescribe `/api/` → `http://api:3000/VISA/api/` respetando `BASE_PATH=/VISA`.
- La API usa `Pool` de PostgreSQL para operaciones transaccionales.
- Triggers en DB gobiernan lógica de autorización / saldo.

---

4. Instalación y Puesta en Marcha

4.1 Requisitos

Componente	Versión recomendada	Notas
Node.js	24.x (LTS)	Backend, scripts, tests
npm	≥ 9.x	Gestión dependencias
Docker	Latest	Ejecución orquestada
Postgres	16 (si fuera local)	Usado vía contenedor

4.2 Clonado del Repositorio

```
git clone https://github.com/bitvalo34/ccvi-visa-emisor.git
cd ccvi-visa-emisor
```

4.3 Ejecución Rápida (Docker Compose)

```
docker compose up -d --build
```

Servicios:

- Web (SPA): http://localhost:8080
- API (BASE_PATH=/VISA): http://localhost:3000/VISA
- Salud: http://localhost:3000/VISA/healthz

4.4 Ejecución Local (sin Docker)

Backend:

```
npm install
npm run dev   # usa .env y arranca en :3000
```

Frontend:

```
cd web
npm install
npm run dev   # :5173 por defecto (Vite)
```

4.5 Scripts Útiles

Script	Descripción
`npm run db:init`	Ejecuta scripts SQL iniciales (001, 002) en contenedor DB
`npm run db:reset`	Dropea esquema y reconstruye
`npm run seed` / `seed:docker`	Carga datos de ejemplo
`npm run test:integration`	Pruebas integradas con DB
`npm run format` / `lint`	Formato y linting de código

---

5. Variables de Entorno Principales

Variable	Propósito	Ejemplo
BASE_PATH	Prefijo de rutas API	/VISA
PORT	Puerto de la API	3000
API_KEY	Clave requerida en x-api-key	dev_api_key_123
CVV_PEPPER	Pepper HMAC para CVV	dev-pepper-123
POSTGRES_*	Conexión a DB	POSTGRES_HOST=db
LOG_LEVEL	Verbosidad (info,warn)	info
PUBLIC_BASE_URL	URL pública (ngrok, etc.)	https://.../VISA
RATE_LIMIT_*	Límite de peticiones (si aplica)	60 / 60000

Plantilla .env mínima:

```
NODE_ENV=development
PORT=3000
BASE_PATH=/VISA
API_KEY=dev_api_key_123
CVV_PEPPER=dev-pepper-123
POSTGRES_HOST=db
POSTGRES_PORT=5432
POSTGRES_DB=ccvi
POSTGRES_USER=app
POSTGRES_PASSWORD=app
LOG_LEVEL=info
```

---

6. Frontend (Uso Operativo)

6.1 Pantallas

Pantalla	Propósito
Listado de Tarjetas	Visualiza tarjetas, filtros, ordenamiento, export CSV
Nueva Tarjeta	Form para emisión (validación Luhn, fecha futura, CVV)
Detalle de Tarjeta	Estado, límites, transacciones, acciones (activar/bloquear/vencida)
Registrar Pago	Incrementa disponible (hasta límite)

6.2 Validaciones UI

- **Número:** 16 dígitos, Luhn.
- **Nombre:** longitud > 2, normalizado a mayúsculas.
- **Fecha venc:** mes/año futuro.
- **CVV:** 3 dígitos.
- **Límite:** número positivo hasta dos decimales.

6.3 Exportación CSV

Genera encabezados: `numero,nombre_titular,fecha_venc,estado,monto_autorizado,monto_disponible`.

---

7. API REST (Integradores)

Prefijo efectivo de las rutas: `BASE_PATH=/VISA` → `/VISA/...` Cabecera de autenticación: `x-api-key: <API_KEY>`

7.1 Endpoints Principales

Método	Ruta	Descripción
GET	`/VISA/healthz`	Estado básico
GET	`/VISA/readyz`	Verifica DB
GET	`/VISA/metadata`	Metadatos del servicio
GET	`/VISA/api/v1/cards`	Listar tarjetas
POST	`/VISA/api/v1/cards`	Crear tarjeta
GET	`/VISA/api/v1/cards/{numero}`	Detalle tarjeta
PATCH	`/VISA/api/v1/cards/{numero}`	Actualizar estado / disponible
POST	`/VISA/api/v1/cards/{numero}/payments`	Registrar pago
GET	`/VISA/api/v1/cards/{numero}/transactions`	Listar transacciones
DELETE	`/VISA/api/v1/cards/{numero}`	Eliminar tarjeta (hard delete)
GET	`/VISA/autorizacion`	Legacy: autorización (query params)
POST	`/VISA/api/v1/authorizations`	Autorización moderna

7.2 Integración con Aerolíneas y Sistemas de Reservas

Algunos integradores (GDS, motores de reserva, aerolíneas) utilizan todavía un flujo legado basado en **GET** con parámetros en la URL. Este flujo se mantiene para compatibilidad pero se recomienda migrar al endpoint moderno POST `/VISA/api/v1/authorizations`.

7.2.1 Endpoint Legacy GET de Autorización Formato base:

`https://<BASE_PUBLICA>/VISA/autorizacion?tarjeta=<PAN>&nombre=<NOMBRE>&fecha_venc=<AAAAMM>&num_seguridad`

Donde `<BASE_PUBLICA>` en entornos de prueba expuestos vía ngrok es:

`https://nonmutinously-unadduceable-sharice.ngrok-free.dev`

Ejemplo completo (JSON):

`https://nonmutinously-unadduceable-sharice.ngrok-free.dev/VISA/autorizacion?tarjeta=4111111111111111&nom`

Parámetros: | Parámetro | Obligatorio | Descripción | |----------|----------|----------| | tarjeta | Sí | PAN de 16 dígitos | | nombre | Sí | Nombre titular normalizado (sin espacios opcional) | | fecha_venc | Sí | AAAAMM (debe ser futuro) | | num_seguridad | Sí | CVV 3 dígitos | | monto | Sí | Importe de la autorización (decimal) | | tienda | Sí | Identificador comercio / canal (ej: MYBOOKING) | | formato | Opcional | `JSON` (default) o `XML` para respuesta |

Cabecera `x-api-key` también puede ser requerida (según configuración). Si se exige y no se envía se devuelve 401.

Respuesta (aprobada JSON):

```
{
  "status": "APROBADO",
  "tarjeta": "4111111111111111",
  "monto": 600.00,
  "autorizacion_numero": "083452",
  "disponible_restante": 600.00,
```

```
  "tienda": "MYBOOKING"
}
```

Respuesta (denegada JSON):

```
{
  "status": "DENEGADO",
  "motivo": "TARJETA_VENCIDA"
}
```

Si `formato=XML` se retorna un documento `<autorizacion>` con campos equivalentes.

Ejemplo respuesta aprobada XML:

```
<autorizacion>
  <status>APROBADO</status>
  <tarjeta>4111111111111111</tarjeta>
  <monto>600.00</monto>
  <autorizacion_numero>083452</autorizacion_numero>
  <disponible_restante>600.00</disponible_restante>
  <tienda>MYBOOKING</tienda>
</autorizacion>
```

Ejemplo respuesta denegada XML:

```
<autorizacion>
  <status>DENEGADO</status>
  <motivo>TARJETA_VENCIDA</motivo>
</autorizacion>
```

Errores comunes legacy: | Código | Motivo | |--------|--------| | 400 | Parámetro faltante o formato inválido | | 401 | Falta `x-api-key` (si requerido) | | 422 | Validación de PAN / fecha / CVV | | 503 | Base de datos no disponible |

Recomendación: Migrar al endpoint POST moderno que permite body JSON, Idempotency-Key y extensiones futuras.

7.2.2 Endpoint Moderno POST de Autorización

Ruta: `/VISA/api/v1/authorizations` Ejemplo:

```
curl -X POST https://nonmutinously-unadduceable-sharice.ngrok-free.dev/VISA/api/v1/authorizations \
  -H 'Content-Type: application/json' \
  -H 'x-api-key: dev_api_key_123' \
  -H 'Idempotency-Key: 55c9ef1b-1c3b-4b1b-9e55-987654321000' \
  -d '{
    "numero":"4111111111111111",
    "monto":600.00,
    "comercio":"MYBOOKING",
    "moneda":"USD"
  }'
```

Ventajas:

- Body estructurado (fácil expansión de campos: moneda, canal, geolocalización).
- Idempotencia robusta.
- Respuestas consistentes JSON/XML administradas vía negociación de `Accept`.
- Mejor control de errores y versionamiento.

Respuesta aprobada:

```
Ejemplo aprobado XML (enviar `Accept: application/xml`):
```xml
```
```

```
<authorization>
  <status>APROBADO</status>
  <autorizacion_numero>412209</autorizacion_numero>
  <numero>4111111111111111</numero>
  <monto>600.00</monto>
  <comercio>MYBOOKING</comercio>
  <monto_disponible_restante>600.00</monto_disponible_restante>
</authorization>
```

Ejemplo denegado XML:

```
<authorization>
  <status>DENEGADO</status>
  <motivo>SALDO_INSUFICIENTE</motivo>
</authorization>
```

{ "status": "APROBADO", "autorizacion_numero": "412209", "numero": "4111111111111111", "monto": 600.00, "comercio": "MYBOOKING", "monto_disponible_restante": 600.00 }

```
#### 7.2.3 Migración Recomendada
1. Validar conectividad a `BASE_PATH` público (ngrok / dominio propio).
2. Replicar parámetros legacy dentro de body POST.
3. Implementar manejo de `Idempotency-Key` para reintentos seguros.
4. Usar códigos de estado HTTP para control de flujo (200, 422, 503) en lugar de parseo de strings.
5. Ajustar logs y monitoreo para diferenciar `authorizations` modernas vs legacy.

#### 7.2.4 Consideraciones de Seguridad
- Evitar registrar PAN completo en sistemas intermedios GDS.
- Usar HTTPS obligatoriamente (ngrok ya cifra).
- Rotar `x-api-key` semestralmente.
- Validar que la fecha de vencimiento no sea cacheada por proxies.

### 7.3 Ejemplo Creación de Tarjeta

### 7.2 Ejemplo Creación de Tarjeta
```bash
curl -X POST http://localhost:3000/VISA/api/v1/cards \
  -H 'Content-Type: application/json' \
  -H 'x-api-key: dev_api_key_123' \
  -d '{
    "numero":"4111111111111111",
    "nombre_titular":"JUAN PEREZ",
    "fecha_venc":"202701",
    "cvv":"123",
    "monto_autorizado":1200
  }'
```

7.4 Respuesta Exitosa (201)

```
{
  "emisor": "VISA",
  "numero": "4111111111111111",
  "nombre_titular": "JUANPEREZ",
  "fecha_venc": "202701",
  "monto_autorizado": 1200,
```
```

```
  "monto_disponible": 1200,
  "estado": "activa",
  "creada_en": "2025-10-14T10:22:01.123Z"
}
```

7.5 Errores Comunes

Código	Motivo
400	Formato general inválido / header fuera de límites
401	Falta o inválida `x-api-key` (si se implementa)
404	Recurso inexistente (tarjeta, transacción)
409	Reuso Idempotency-Key con parámetros distintos
422	Validación de entrada (formato monto, PAN, fecha)
503	Base de datos no lista

7.6 Idempotencia

7.7 Eliminación de Tarjetas

La eliminación se realiza con **hard delete** vía `DELETE /VISA/api/v1/cards/{numero}`:

```
curl -X DELETE http://localhost:3000/VISA/api/v1/cards/4111111111111111 \
  -H 'x-api-key: dev_api_key_123'
```

Respuesta (200):

```
{ "numero": "4111111111111111", "action": "deleted" }
```

Si la tarjeta no existe: **404 CARD_NOT_FOUND**.

Notas:

- Las transacciones históricas permanecen (referencian el número ya eliminado). Esto permite auditoría; el modelo actual no usa constraint ON DELETE CASCADE restrictivo sobre `transacciones.tarjeta_numero`.
- No se admite "soft delete"; el estado `eliminada` no se utiliza.
- Intentar operaciones (PATCH, pagos, autorizaciones) sobre un número ya eliminado resultará en 404.

Recomendación operativa: Exportar / respaldar transacciones antes de campañas masivas de limpieza si se requiere correlación posterior.

Enviar `Idempotency-Key` en headers (pagos/autorizaciones). Repetir la misma clave con el mismo payload → misma respuesta sin duplicar efectos.

---

8. Modelo de Datos

8.1 Tabla `emisor.tarjetas`

Campo	Tipo	Descripción
numero	CHAR(16)	PAN (clave primaria)
nombre_titular	TEXT	Texto original ingresado
nombre_titular_normalizado	TEXT	Versión normalizada (mayúsculas, sin tildes)
fecha_venc	CHAR(6)	Formato `YYYYMM`
cvv_hmac	CHAR(64)	HMAC-SHA256 del CVV + pepper

Campo	Tipo	Descripción
monto_autorizado	NUMERIC(14,2)	Límite asignado
monto_disponible	NUMERIC(14,2)	Saldo disponible
estado	TEXT	`activa` / `bloqueada` / `vencida`
creada_en	TIMESTAMP	Fecha creación
actualizada_en	TIMESTAMP	Fecha última modificación

8.2 Tabla `emisor.transacciones`

Campo	Tipo	Descripción
id	BIGSERIAL	Identificador
tarjeta_numero	CHAR(16)	FK a tarjeta
tipo	TEXT	`consumo` / `pago`
monto	NUMERIC(14,2)	Importe
comercio	TEXT	Comercio / referencia
idempotency_key	TEXT	Clave idempotencia opcional
status	TEXT	`APROBADO` / `DENEGADO`
autorizacion_numero	CHAR(6)	Número único (cuando APROBADO)
detalle_denegacion	TEXT	Motivo si DENEGADO
creada_en	TIMESTAMP	Fecha creación

8.3 Triggers Relevantes

- `tg_transacciones_apply`: Aplica lógica de autorización (saldo, vencimiento, estado, número de autorización).
- `tg_tarjetas_touch`: Actualiza `actualizada_en` en modificaciones.

---

9. Seguridad

Aspecto	Implementación
CVV	Nunca se almacena plano; se HMAC con Pepper
Logs	Redacción de campos sensibles (headers, CVV, PAN)
API Key	`x-api-key` requerida para rutas críticas (tarjetas, pagos, autorizaciones)
Idempotencia	Previene duplicados en pagos / autorizaciones
Vencimiento	Rechazo si `fecha_venc < mes actual`
Estado	Tarjeta no activa → DENEGADO transacción

Recomendaciones futuras: Rate limiting formal, rotación de claves, monitoreo de fraude.

---

10. Logs y Observabilidad

- Logging con `pino` y middleware `pino-http`.
- Redacta campos: CVV, PAN completo, API key.
- Health: `/VISA/healthz` (rápido), `/VISA/readyz` (consulta DB).
- Para ver logs en Docker:

```
docker compose logs -f api
```

---

11. CI/CD (GitHub Actions)

Jobs: | Job | Función | |-----|---------| | `lint_unit` | Instala deps, formatea, lint, tests unitarios (`healthz.test.js`) | | `integration_with_db` | Levanta stack con DB, aplica SQL y corre tests de integración |

Artifacts: no se generan binarios; imágenes Docker se construyen bajo demanda.

---

12. Mantenimiento y Solución de Problemas

Síntoma	Causa probable	Acción
404 tarjeta	Número incorrecto / no creada	Verificar PAN, crear nuevamente
404 delete	Tarjeta ya eliminada o inexistente	Confirmar que no se repitió operación
422 creación	Campo inválido (fecha, PAN, CVV)	Revisar formato y validaciones UI/API
409 pago	Idempotency-Key reutilizada con datos distintos	Usar nueva clave o repetir mismos datos
503 readyz	DB no inicializada	Ejecutar `npm run db:init` o esperar healthcheck
Saldo no cambia	Trigger no aplicado	Verificar scripts 002 y logs de DB

12.1 Reset del Entorno

```
npm run db:reset
npm run seed
```

12.2 Limpieza de Contenedores

12.3 Aplicar Cambios Recientes (DELETE + Frontend)

1. Asegurar que la API esté reconstruida con los cambios:

```
docker compose build api web
docker compose up -d
```

2. Verificar endpoint DELETE manualmente:

```
curl -X DELETE http://localhost:3000/VISA/api/v1/cards/<PAN> -H "x-api-key: $API_KEY" -i
```

3. En frontend (SPA), usar botón "Eliminar" en el listado; confirmación vía dialog nativo `window.confirm`.
4. Post eliminación se invalida la caché React Query y la tarjeta desaparece del listado.
5. Si es necesario revertir (recuperar tarjeta), se debe recrear con POST (no hay papelera de reciclaje).

```
docker compose down -v && docker compose up -d --build
```

---

13. Extensiones Futuras

- Autenticación con JWT / OAuth2
- Encriptación adicional para PAN
- Métricas Prometheus / Grafana
- Webhooks de eventos (creación tarjeta, pago registrado)
- Rate limiting global y por IP

---

14. Generación a PDF

Puede convertir este documento a PDF usando herramientas como:

```
# Requiere pandoc instalado
pandoc docs/manual/README.md -o manual-usuario.pdf
```

Opcional: incluir hoja de estilo personalizada (`--css` o plantilla tipo Eisvogel).

---

15. Anexos

15.1 OpenAPI

El contrato OpenAPI se encuentra en `docs/openapi.yaml`. La API sirve documentación interactiva en `/VISA/docs`.

15.2 Referencias Internas

- Normalización de entrada: `docs/normalizacion-entrada.md`
- SQL esquema y negocio: `scripts/sql/001_init.sql`, `scripts/sql/002_business.sql`
- Nginx proxy SPA: `web/nginx.conf`
- Dockerfile frontend: `web/Dockerfile`

Fin del documento