

Licenciatura en Sistemas de Información  
Arq. Programación Móvil - 2025

## Sistema de eventos bancarios con Kafka → ReactJS - NextJS

01

### Requerimientos

---

**Objetivo:** simular el ciclo de vida de una **transacción bancaria** (iniciar, reservar fondos, antifraude, enviar fondos, notificación) publicando/hearing eventos en Kafka y **mostrarlos en tiempo real** en una app Web de **React JS o NextJS** (vía un **gateway WebSocket**).

02

### Servicios

---

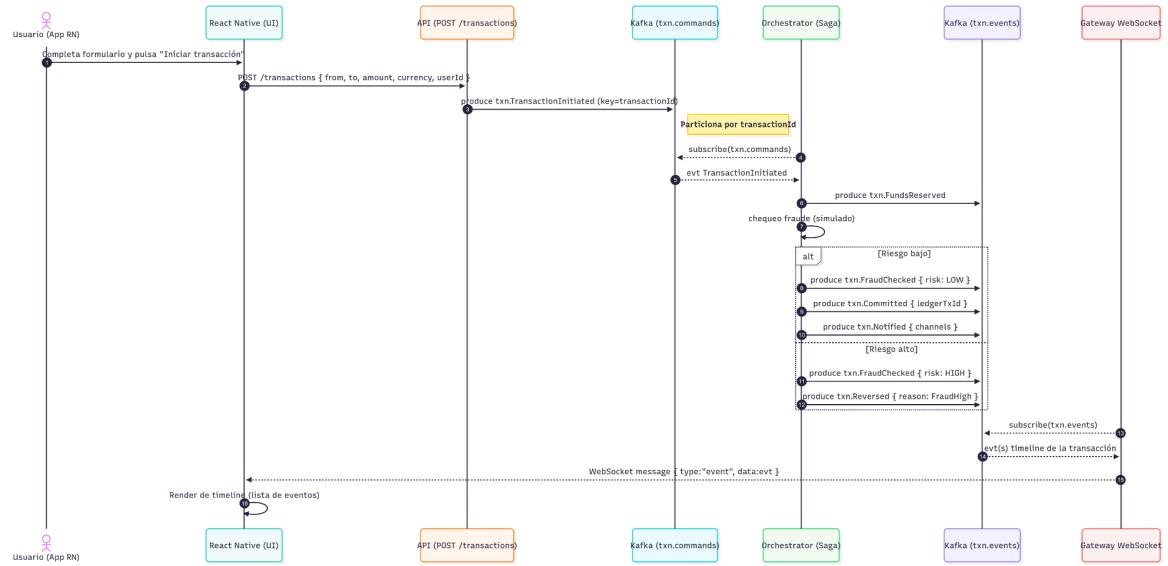
- **API** (`api.js`): recibe `POST /transactions` y publica `txn.commands`.
- **Orchestrator** (`orchestrator.js`): consume `txn.commands`, emite `txn.events` (FundsReserved, FraudChecked, Committed/Reversed, Notified) y maneja `txn.dlq` en errores.
- **Gateway WS** (`gateway.js`): consume `txn.events` y los reenvía por WebSocket a la app móvil (permite suscripción por `userId/transactionId`).
- **App RN** (`rn-txn`): inicia transacciones y visualiza el **timeline** en vivo.

**Tópicos Kafka:** `txn.commands`, `txn.events`, `txn.dlq`

**Clave de partición:** `transactionId` (garantiza orden por transacción).

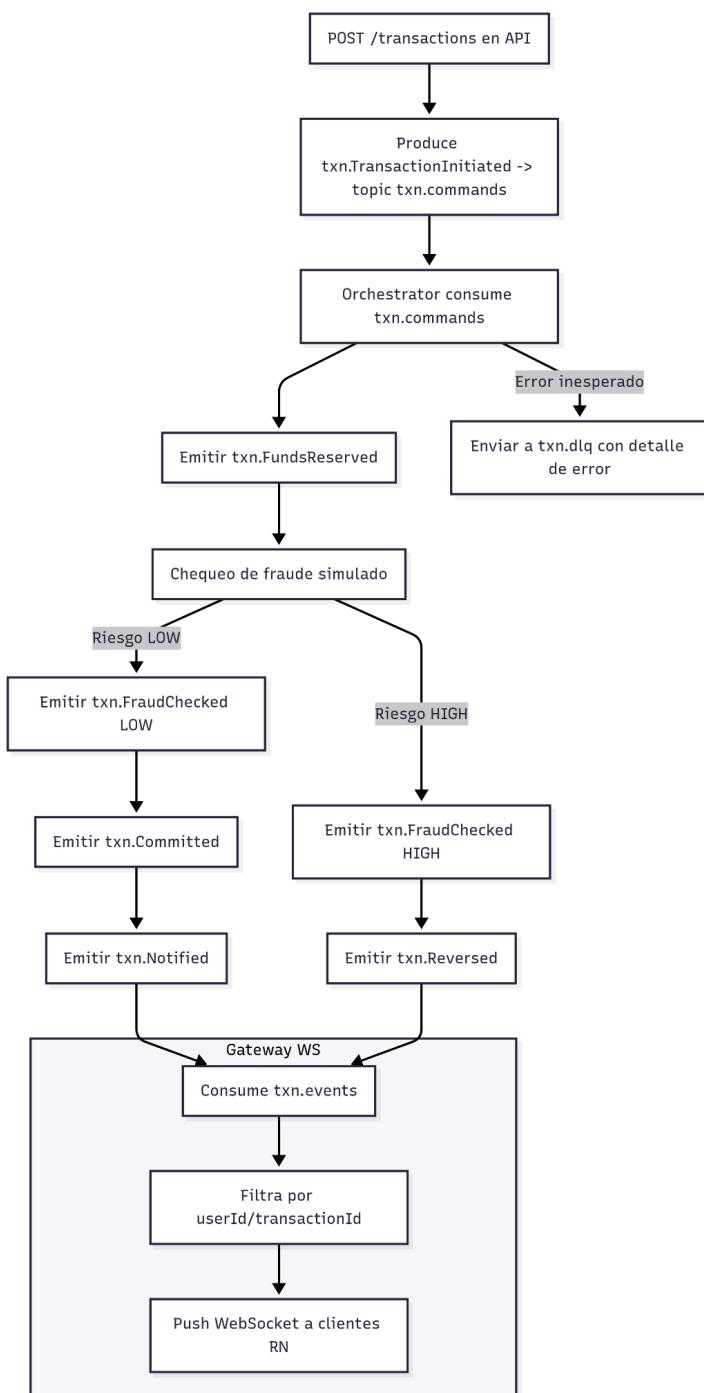
## 03

### Diagrama de secuencia



## 04

### Diagrama de Flujo



05

## Contratos del evento

---

```
type EventEnvelope<T> = {
    id: string;           // uuid v4
    type: string;         // ej: "txn.FundsReserved"
    version: number;      // 1
    ts: number;           // epoch ms
    transactionId: string; // partición
    userId: string;
    payload: T;
    correlationId?: string;
};
```

### Ejemplos de payload:

- TransactionInitiated: { fromAccount, toAccount, amount, currency, userId }
- FundsReserved: { ok: true, holdId, amount }
- FraudChecked: { risk: 'LOW' | 'HIGH' }
- Committed: { ledgerTxId }
- Reversed: { reason }
- Notified: { channels: string[] }

06

## Infra local

---

Docker

07

## Backend

---

Nodejs - NestJS - API Middleware NextJS

08

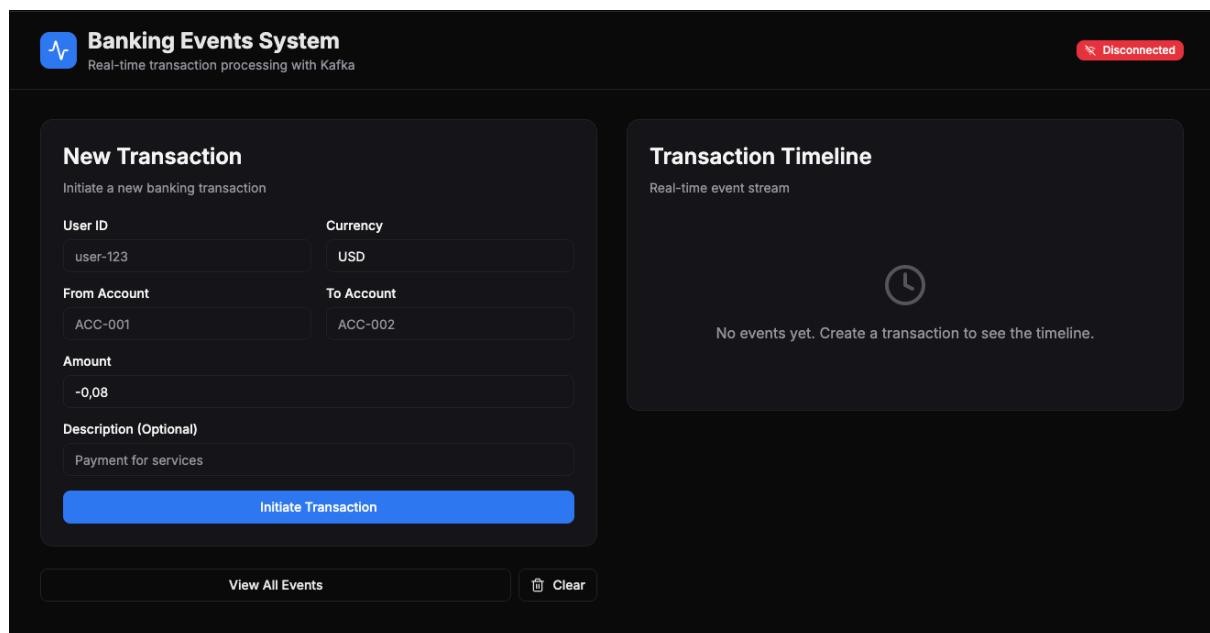
## App

---

ReactJS - NextJS

UI: Puede buscar ejemplos de diseño y flujo de diseño basados en MEILI o alguna otra operadora bancaria.

mockup - demo



The screenshot displays a dark-themed user interface for a banking events system. At the top, there's a header bar with a blue square icon containing a white downward arrow, the text "Banking Events System", and a red "Disconnected" status indicator. Below the header, the main area is divided into two sections: "New Transaction" on the left and "Transaction Timeline" on the right.

**New Transaction:** This section is titled "New Transaction" and includes the sub-instruction "Initiate a new banking transaction". It contains several input fields: "User ID" (user-123), "Currency" (USD), "From Account" (ACC-001), "To Account" (ACC-002), "Amount" (-0.08), and "Description (Optional)" (Payment for services). A prominent blue button at the bottom is labeled "Initiate Transaction".

**Transaction Timeline:** This section is titled "Transaction Timeline" and describes itself as a "Real-time event stream". It features a large clock icon and the message "No events yet. Create a transaction to see the timeline." There are also "View All Events" and "Clear" buttons at the bottom of this section.

## Add Environment Variables

This generation requires environment variables. Enter them below.

KAFKA\_BROKERS

Enter a key

KAFKA\_CLIENT\_ID

Enter a key

NEXT\_PUBLIC\_WS\_URL

Enter a key

Please fill out this field.

PORT

Enter a key

Submit

09

Qué se está practicando (buenas prácticas reales)

---

- **Orden por transacción:** usar `transactionId` como key ⇒ orden fuerte en `txn.events`.
- **Saga/Orquestación:** pasos con *retry simple*, *fraude simulado* y *rollback* (Reversed) si hace falta.
- **Contratos de evento:** un `envelope` estable evita dolores de versionado.
- **Aislamiento móvil:** RN no habla con Kafka; consume vía **WebSocket Gateway** (puede aplicar auth/JWT, rate-limit, wss, etc.).
- **Observabilidad:** podrías añadir un `/metrics` al gateway y contadores por tipo de evento.
- **DLQ:** errores “no recuperables” → `txn.dlq` (inspección offline).

- **Idempotencia** (siguiente paso): guarda *processed event ids* para no ejecutar dos veces efectos con side-effects (debito/credito).
- **Outbox pattern** (siguiente paso): si persistes estados en DB, utiliza Outbox para publicar eventos de forma transaccional.