UNIVERSIDAD LAICA ELOY ALFARO DE MANABÍ
FACULTAD DE CIENCIAS INFORMÁTICAS
CARRERA DE SOFTWARE

TALLER DE PRÁCTICA 2

ARQUITECTURA EVENT-DRIVEN CON WEBHOOKS Y SERVERLESS

Carrera:	**Software**
Nivel:	Quinto
Periodo Lectivo:	2025-2026 (2)
Asignatura:	Aplicación para el Servidor Web
Docente:	Ing. John Cevallos
Paralelos:	A
Número de Taller:	2p-2
Fecha:	15 de Diciembre 2025
Duración:	2 horas académicas
Modalidad:	Grupal (3 estudiantes)
Prerrequisito:	Taller 2p-1 completado y aprobado

1. DESCRIPCIÓN GENERAL

En este taller, los estudiantes evolucionarán la arquitectura híbrida desarrollada en el Taller 2p-1 (Microservicios + RabbitMQ) para integrar dos tecnologías empresariales fundamentales en sistemas distribuidos modernos:

1.1 WEBHOOKS (Event Publishing)

Sistema de notificación push que permite a los microservicios comunicar eventos de negocio críticos a sistemas externos de forma desacoplada y en tiempo real.

1.2 SERVERLESS COMPUTING (Edge Functions)

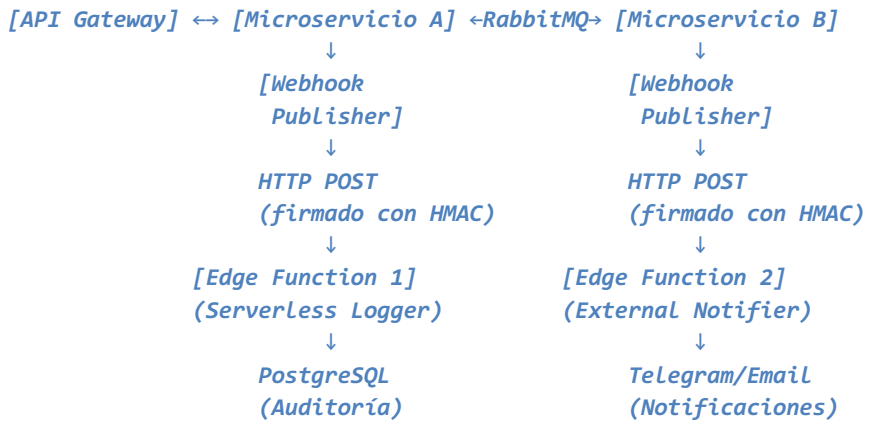
Funciones que se ejecutan bajo demanda en la nube sin gestión de infraestructura, actuando como receptores inteligentes de los webhooks publicados.

1.3 CONTEXTO DE EVOLUCIÓN

Taller 1 (Completado): **Modificar si es necesario la arquitectura del primer taller para que el Microservicio 1 se comunique con el Microservicio 2.**

```
[API Gateway] ↔ [Microservicio A] ←RabbitMQ→ [Microservicio B]
(Comunicación interna asíncrona)
```

Taller 2 (Este Taller):



1.4 VALOR REAL DEL PROYECTO

Este patrón arquitectónico es utilizado por empresas líderes como:

- **Stripe:** Notifica eventos de pagos vía webhooks
- **GitHub:** Notifica eventos de repositorios (push, pull request)
- **Shopify:** Notifica eventos de e-commerce (order.created, product.updated)
- **Twilio:** Notifica eventos de comunicación (sms.received, call.completed)

2. OBJETIVOS DE APRENDIZAJE

Al completar este taller, los estudiantes serán capaces de:

OBJETIVO 1: IMPLEMENTACIÓN DE WEBHOOKS EMPRESARIALES

- Diseñar y construir un sistema de publicación de webhooks con firma HMAC-SHA256
- Implementar retry logic con exponential backoff para garantizar entrega
- Gestionar múltiples suscriptores de webhooks de forma dinámica
- Crear payloads estructurados siguiendo estándares de la industria

OBJETIVO 2: SERVERLESS COMPUTING

- Desplegar funciones serverless en Supabase Edge Functions (Deno Runtime)
- Implementar validación de seguridad (firma HMAC, timestamp anti-replay)
- Procesar eventos de forma escalable sin gestión de servidores
- Integrar funciones serverless con bases de datos y APIs externas

OBJETIVO 3: PATRONES DE RESILIENCIA AVANZADOS

- Implementar idempotencia para prevenir procesamiento duplicado
- Configurar Dead Letter Queues para webhooks fallidos
- Implementar Circuit Breaker para proteger endpoints externos
- Crear sistemas de observabilidad con correlation IDs y logs estructurados

OBJETIVO 4: OBSERVABILIDAD DISTRIBUIDA

- Rastrear eventos a través de múltiples sistemas
- Implementar logging estructurado con formato JSON
- Crear dashboards de monitoreo de entregas de webhooks
- Diagnosticar fallos en sistemas distribuidos

3. COMPONENTES DEL SISTEMA

3.1 WEBHOOK PUBLISHER SERVICE

Ubicación: Módulo agregado a Microservicios A y B existentes

Responsabilidades:

- Escuchar eventos internos de RabbitMQ
- Transformar eventos internos a formato estándar de webhook
- Firmar payloads con HMAC-SHA256
- Enviar HTTP POST a URLs registradas
- Implementar retry con exponential backoff
- Registrar todos los intentos de entrega en base de datos

Tecnologías: NestJS, @nestjs/axios, Bull/BullMQ, crypto (Node.js)

3.2 EDGE FUNCTION 1 - EVENT LOGGER

Ubicación: Supabase Edge Functions (Deno Runtime)

Responsabilidades:

- Validar firma HMAC del webhook
- Verificar timestamp (anti-replay attack, máximo 5 minutos)
- Verificar idempotencia (deduplicar eventos duplicados)
- Guardar evento completo en tabla webhook_events
- Retornar 200 OK con event_id generado

Tecnologías: Deno, Supabase Client, PostgreSQL

3.3 EDGE FUNCTION 2 - EXTERNAL NOTIFIER

Ubicación: Supabase Edge Functions (Deno Runtime)

Responsabilidades:

- Validar firma HMAC del webhook
- Verificar idempotencia con PostgreSQL
- Enviar notificaciones a Telegram Bot / Email
- Registrar resultado de notificación
- Retornar 200 OK o 500 para retry

Tecnologías: Deno, Telegram Bot API, Email APIs

3.4 WEBHOOK REGISTRY

Ubicación: Nuevas tablas en PostgreSQL

Responsabilidades:

- webhook_subscriptions: URLs, secrets, configuración de retry
- webhook_deliveries: Auditoría completa de todos los envíos
- processed_webhooks: Control de idempotencia

Propósito: Gestionar suscripciones de webhooks dinámicamente

4. EL RETO: SELECCIÓN DE ESTRATEGIA AVANZADA

⚠️ **RESTRICCIÓN CRÍTICA:** Los estudiantes NO deben implementar webhooks simples sin resiliencia. Deben elegir, investigar e implementar UNA (1) de las siguientes estrategias avanzadas que garanticen robustez empresarial.

4.1 OPCIÓN A: WEBHOOK FANOUT CON DEAD LETTER QUEUE

EL PROBLEMA:

Cuando un webhook falla (receptor caído, timeout, error 500), el sistema tiene dos opciones malas: (1) Reintentar infinitamente → Bloquea la cola, o (2) Descartar el evento → Pérdida de datos críticos.

LA ESTRATEGIA:

Implementar un sistema de distribución (fanout) donde un evento interno genera múltiples webhooks hacia diferentes receptores, con un manejo robusto de fallos:

- FANOUT PUBLISHER: Un evento genera N webhooks (uno por suscriptor)
- RETRY CON EXPONENTIAL BACKOFF: 6 intentos (1min, 5min, 30min, 2h, 12h)
- DEAD LETTER QUEUE (DLQ): Después de 6 intentos → mensaje va a DLQ
- REPLAY MECHANISM: Edge Function para reenviar webhooks desde DLQ

4.2 OPCIÓN B: IDEMPOTENT CONSUMER (CONSUMIDOR IDEMPOTENTE)

EL PROBLEMA:

RabbitMQ garantiza "At-least-once delivery", no "Exactly-once". Si la red falla antes del ACK, el mensaje se duplica. Procesar un pago dos veces puede ser catastrófico.

LA ESTRATEGIA:

Implementar deduplicación estricta usando "Idempotency Keys" que garanticen que el efecto en la base de datos ocurra exactamente una vez, aunque el mensaje llegue múltiples veces.

- IDEMPOTENCY KEY GENERATION: Clave única basada en event_type + entity_id + action
- IDEMPOTENCY STORE: Almacenar claves procesadas en PostgreSQL
- VERIFICATION: Antes de procesar, verificar si la clave ya existe
- TTL: Configurar TTL de 7 días para limpiar claves antiguas

4.3 OPCIÓN C: WEBHOOK SIGNATURE CHAIN (CADENA DE CONFIANZA)

EL PROBLEMA:

En sistemas distribuidos complejos, un evento puede pasar por múltiples sistemas. ¿Cómo sabe el destino final que el mensaje pasó por todos los sistemas intermedios correctamente?

LA ESTRATEGIA:

Implementar una "Cadena de Firmas" donde cada sistema en la ruta agrega su propia firma HMAC al webhook, creando un registro inmutable.

- SIGNATURE CHAIN HEADER: Header HTTP con array de firmas [firma_A, firma_B, firma_C]
- SIGNATURE VERIFICATION: Cada sistema verifica la firma del sistema anterior
- PROVENANCE TRACKING: Registro completo de todos los sistemas que tocaron el mensaje
- AUDIT TRAIL: Trazabilidad inmutable de la ruta del evento

4.4 OPCIÓN D: SERVERLESS CIRCUIT BREAKER

EL PROBLEMA:

Cuando un sistema externo (ej. servicio de email) está caído, continuar enviando requests es contraproducente: desperdicio de recursos, acumulación en colas.

LA ESTRATEGIA:

Implementar un Circuit Breaker que monitorea la salud de cada endpoint externo y automáticamente "abre el circuito" cuando detecta fallos consecutivos.

- CLOSED (Cerrado): Sistema funcionando normalmente, requests fluyen
- OPEN (Abierto): Sistema detectado como caído, NO se envían requests
- HALF_OPEN (Semi-abierto): Periodo de prueba, se permite 1 request
- MONITORING: Estado persistido en Redis para coordinación distribuida

5. ESTRUCTURA DE PAYLOAD DEL WEBHOOK

Todo webhook enviado DEBE cumplir con la siguiente estructura JSON estándar:

```
{
  "event": "string",              // Tipo de evento (ej.
"order.created")
  "version": "string",            // Versión del schema (ej. "1.0")
  "id": "string",                 // ID único del webhook (UUID v4)
  "idempotency_key": "string",    // Clave para deduplicación
  "timestamp": "string",          // ISO 8601 (ej. "2025-12-
15T10:30:00Z")
  "data": {                       // Datos específicos del evento
    // Campos variables según el evento
  },
  "metadata": {                   // Información contextual
    "source": "string",           // Origen (ej. "microservice-b")
    "environment": "string",      // Entorno (ej. "production",
"staging")
    "correlation_id": "string"    // ID para rastrear a través de
sistemas
  }
}
```

5.1 EJEMPLOS POR DOMINIO

E-commerce - Order Created:

```
{
  "event": "order.created",
  "version": "1.0",
  "id": "evt_9f8e7d6c5b4a",
  "idempotency_key": "order-12345-created-20251215",
  "timestamp": "2025-12-15T10:30:45.123Z",
  "data": {
    "order_id": 12345,
    "customer_id": 67890,
    "status": "pending",
    "total": 159.99,
    "currency": "USD",
    "items_count": 3
  },
  "metadata": {
    "source": "microservice-orders",
    "environment": "production",
    "correlation_id": "req_abc123xyz"
  }
}
```

Healthcare - Appointment Scheduled:

```
{
  "event": "appointment.scheduled",
  "version": "1.0",
  "id": "evt_1a2b3c4d5e6f",
  "idempotency_key": "appointment-456-scheduled-20251215",
  "timestamp": "2025-12-15T14:15:00.000Z",
  "data": {
    "appointment_id": 456,
    "patient_id": 789,
    "doctor_id": 321,
    "specialty": "Cardiology",
    "scheduled_at": "2025-12-20T09:00:00.000Z",
    "duration_minutes": 30,
    "status": "confirmed"
  },
  "metadata": {
    "source": "microservice-appointments",
    "environment": "production",
    "correlation_id": "req_xyz789abc"
  }
}
```

6. SEGURIDAD: GENERACIÓN Y VALIDACIÓN DE FIRMA HMAC

6.1 GENERACIÓN DE FIRMA (Publisher - NestJS)

```
import * as crypto from 'crypto';

@Injectable()
export class WebhookSecurityService {
  generateSignature(payload: any, secret: string): string {
    // 1. Serializar payload a JSON string (sin espacios)
    const payloadString = JSON.stringify(payload);

    // 2. Crear HMAC con SHA256
    const hmac = crypto
      .createHmac('sha256', secret)
      .update(payloadString)
      .digest('hex');

    // 3. Retornar con prefijo estándar
    return `sha256=${hmac}`;
  }

  generateTimestamp(): string {
    return Math.floor(Date.now() / 1000).toString();
  }
}
```

6.2 VALIDACIÓN DE FIRMA (Consumer - Edge Function)

```
async function validateSignature(
  body: string,
  signature: string,
  secret: string
): Promise<boolean> {
  // 1. Extraer hash de la firma
  const receivedHash = signature.replace('sha256=', '');

  // 2. Calcular hash esperado
  const encoder = new TextEncoder();
  const keyData = encoder.encode(secret);
  const messageData = encoder.encode(body);

  const key = await crypto.subtle.importKey(
    'raw',
    keyData,
    { name: 'HMAC', hash: 'SHA-256' },
    false,
    ['sign']
  );

  const signatureBuffer = await crypto.subtle.sign(
```

```
    'HMAC',
    key,
    messageData
  );

  // 3. Convertir a hex
  const expectedHash = Array.from(new Uint8Array(signatureBuffer))
    .map(b => b.toString(16).padStart(2, '0'))
    .join('');

  // 4. Comparar de forma segura
  return timingSafeEqual(receivedHash, expectedHash);
}

function validateTimestamp(timestamp: string, maxAgeMinutes: number
= 5): boolean {
  const now = Math.floor(Date.now() / 1000);
  const requestTime = parseInt(timestamp);
  const age = now - requestTime;

  // Verificar que no sea muy antiguo (anti-replay)
  if (age > maxAgeMinutes * 60) return false;

  // Verificar que no sea del futuro (clock skew)
  if (age < -60) return false;

  return true;
}
```

7. ESQUEMA DE BASE DE DATOS

El sistema requiere las siguientes tablas en PostgreSQL para gestionar webhooks, auditoría y resiliencia:

```
-- Tabla: webhook_subscriptions
-- Gestión de URLs suscritas a eventos
CREATE TABLE webhook_subscriptions (
  id SERIAL PRIMARY KEY,
  event_type VARCHAR(100) NOT NULL,
  url VARCHAR(500) NOT NULL,
  secret VARCHAR(255) NOT NULL,
  is_active BOOLEAN DEFAULT true,
  retry_config JSONB DEFAULT '{
    "max_attempts": 6,
    "backoff_type": "exponential",
    "initial_delay_ms": 60000
  }'::jsonb,
  created_at TIMESTAMP DEFAULT NOW(),
  updated_at TIMESTAMP DEFAULT NOW(),
  last_triggered_at TIMESTAMP,
  CONSTRAINT unique_event_url UNIQUE(event_type, url)
);

-- Tabla: webhook_events
-- Registro de todos los eventos recibidos
CREATE TABLE webhook_events (
  id SERIAL PRIMARY KEY,
  event_id VARCHAR(255) UNIQUE NOT NULL,
  event_type VARCHAR(100) NOT NULL,
  idempotency_key VARCHAR(255) UNIQUE NOT NULL,
  payload JSONB NOT NULL,
  metadata JSONB,
  received_at TIMESTAMP NOT NULL DEFAULT NOW(),
  processed_at TIMESTAMP
);

-- Tabla: webhook_deliveries
-- Auditoría de todos los intentos de entrega
CREATE TABLE webhook_deliveries (
  id SERIAL PRIMARY KEY,
  subscription_id INTEGER REFERENCES webhook_subscriptions(id),
  event_id VARCHAR(255) REFERENCES webhook_events(event_id),
  attempt_number INTEGER NOT NULL,
  status_code INTEGER,
  status VARCHAR(20) NOT NULL CHECK (status IN ('success', 'failed',
'pending')),
  error_message TEXT,
  delivered_at TIMESTAMP NOT NULL DEFAULT NOW(),
  duration_ms INTEGER
);

-- Tabla: processed_webhooks
```

```
-- Control de idempotencia (deduplicación)
CREATE TABLE processed_webhooks (
  id SERIAL PRIMARY KEY,
  idempotency_key VARCHAR(255) UNIQUE NOT NULL,
  event_id INTEGER REFERENCES webhook_events(id),
  processed_at TIMESTAMP NOT NULL DEFAULT NOW(),
  expires_at TIMESTAMP NOT NULL DEFAULT (NOW() + INTERVAL '7 days')
);

-- Índices para performance
CREATE INDEX idx_event_type ON webhook_events(event_type);
CREATE INDEX idx_idempotency_key ON webhook_events(idempotency_key);
CREATE INDEX idx_delivered_at ON webhook_deliveries(delivered_at
DESC);
CREATE INDEX idx_subscription_status ON
webhook_deliveries(subscription_id, status);
```

---

8. DEPLOYMENT DE EDGE FUNCTIONS

Pasos para desplegar las funciones serverless en Supabase:

Paso 1: Inicializar proyecto Supabase

```
cd tu-proyecto
supabase init
```

Paso 2: Crear las funciones

```
supabase functions new webhook-event-logger
supabase functions new webhook-external-notifier
```

Paso 3: Vincular con proyecto

```
supabase link --project-ref tu-project-ref
```

Paso 4: Configurar secrets

```
supabase secrets set WEBHOOK_SECRET=tu-secret-compartido
supabase secrets set TELEGRAM_TOKEN=tu-token
supabase secrets set TELEGRAM_CHAT_ID=tu-chat-id
```

Paso 5: Desplegar funciones

```
supabase functions deploy webhook-event-logger
supabase functions deploy webhook-external-notifier
```

Paso 6: Verificar deployment

```
supabase functions list
```

URLs resultantes:

```
https://[tu-proyecto].supabase.co/functions/v1/webhook-event-logger
https://[tu-proyecto].supabase.co/functions/v1/webhook-external-
notifier
```

9. GUÍA DE IMPLEMENTACIÓN PASO A PASO

Esta sección provee una ruta clara y secuencial para completar el taller.

FASE 1: PREPARACIÓN Y DISEÑO (20 minutos)

- Definir 2 eventos de negocio de su dominio
- Diseñar diagrama de arquitectura completo
- Crear repositorio Git con .gitignore configurado
- Documentar estructura de payloads de webhook

FASE 2: CONFIGURACIÓN DE INFRAESTRUCTURA (30 minutos)

- Crear proyecto en Supabase
- Configurar base de datos (ejecutar schema SQL)
- Crear Bot de Telegram
- Configurar secrets en Supabase
- Inicializar proyecto de Edge Functions

FASE 3: IMPLEMENTACIÓN DE EDGE FUNCTIONS (40 minutos)

- Implementar webhook-event-logger con validación HMAC
- Implementar webhook-external-notifier con Telegram
- Desplegar ambas funciones a Supabase
- Probar manualmente con curl

FASE 4: EXTENDER MICROSERVICIOS (50 minutos)

- Agregar Webhook Publisher Service a microservicios
- Implementar generación de firma HMAC
- Configurar Bull/BullMQ para retry logic
- Integrar con eventos existentes de RabbitMQ
- Implementar estrategia avanzada elegida

FASE 5: PRUEBAS Y VALIDACIÓN (30 minutos)

- Prueba de flujo completo (end-to-end)
- Prueba de duplicados (idempotencia)
- Prueba de retry (apagar Edge Function)
- Prueba de estrategia avanzada específica
- Verificar logs en todos los componentes

FASE 6: DOCUMENTACIÓN (20 minutos)

- Actualizar README con diagramas
- Documentar eventos publicados
- Incluir instrucciones de setup
- Agregar ejemplos de uso con curl
- Documentar estrategia avanzada implementada

10. PRUEBAS DE VALIDACIÓN

Cada grupo debe ejecutar y documentar los siguientes escenarios de prueba:

PRUEBA 1: Happy Path - Flujo Completo

Crear una transacción que genere un evento → Webhook enviado → Edge Function procesa correctamente → Notificación recibida

```
# Ejemplo: Crear orden
curl -X POST http://localhost:3000/api/orders \
  -H "Content-Type: application/json" \
  -d '{"customer_id": 123, "total": 99.99, "items": [...]}'

# Verificar:
# 1. Logs del Webhook Publisher: "✅ Webhook sent"
# 2. Tabla webhook_events: Nuevo registro
# 3. Tabla webhook_deliveries: status='success'
# 4. Telegram: Mensaje recibido
```

PRUEBA 2: Validación de Firma HMAC

Modificar el payload antes de enviar → Edge Function rechaza con 401

```
# Enviar webhook con firma inválida
curl -X POST https://tu-proyecto.supabase.co/functions/v1/webhook-
event-logger \
  -H "Content-Type: application/json" \
  -H "X-Webhook-Signature: sha256=FIRMA_INVALIDA" \
  -d '{"event": "test", "data": {}}'

# Resultado esperado: HTTP 401 - Invalid signature
```

PRUEBA 3: Idempotencia - Duplicados

Enviar el mismo webhook 3 veces → Edge Function procesa solo una vez

```
# Enviar el mismo webhook 3 veces
for i in {1..3}; do
  curl -X POST https://tu-proyecto.supabase.co/functions/v1/webhook-
event-logger \
    -H "Content-Type: application/json" \
    -H "X-Webhook-Signature: sha256=..." \
    -d '{"idempotency_key": "test-001", ...}'
done

# Verificar:
# - webhook_events: Solo 1 registro
# - processed_webhooks: 1 entrada
# - Response 2 y 3: {"duplicate": true}
```

PRUEBA 4: Retry con Exponential Backoff

Apagar temporalmente la Edge Function → Webhook reintenta → Se entrega cuando vuelve

```
# 1. Pausar Edge Function (o simular con URL inválida)
# 2. Generar evento que dispare webhook
# 3. Observar reintentos en webhook_deliveries:
#    - attempt_number: 1, 2, 3...
#    - status: 'failed'
# 4. Reactivar Edge Function
# 5. Verificar entrega exitosa en próximo intento
```

PRUEBA 5: Estrategia Avanzada Específica

Demostrar funcionamiento de la estrategia elegida (DLQ, Circuit Breaker, etc.)

```
# Ejemplo para DLQ:
# 1. Configurar URL inválida en webhook_subscriptions
# 2. Generar evento
# 3. Observar 6 intentos fallidos
# 4. Verificar mensaje en Dead Letter Queue
# 5. Reenviar manualmente desde DLQ
# 6. Confirmar entrega exitosa
```

---

11. ENTREGABLES Y SISTEMA DE EVALUACIÓN

11.1 REPOSITORIO DE CÓDIGO (Obligatorio)

El repositorio debe contener:

- Proyecto NestJS con microservicios extendidos
- Edge Functions desplegadas (con URLs públicas)
- docker-compose.yml actualizado
- Variables de entorno documentadas (.env.example)
- README.md completo con instrucciones paso a paso
- Diagrama de arquitectura (imagen o Mermaid)
- Scripts SQL para crear esquema de base de datos
- Documentación de la estrategia avanzada implementada

11.2 MODALIDAD DE EVALUACIÓN

Opción A: Presentación en Clase (100% de la nota)

El estudiante debe presentar y demostrar en vivo al docente durante la clase el funcionamiento completo de su implementación. La presentación debe incluir:

- Explicación breve de la arquitectura implementada (5 minutos)
- Demostración funcional del flujo completo - Happy Path (5 minutos)
- Prueba de resiliencia en vivo: simulación del fallo específico que la estrategia resuelve y demostración de la recuperación (10 minutos)
- Respuesta a preguntas técnicas del docente sobre decisiones de diseño e implementación (5 minutos)

Opción B: Video Demostrativo (50% de la nota)

Si el estudiante NO logra demostrar exitosamente el funcionamiento en clase, deberá elaborar un video de 5-10 minutos que incluya los mismos elementos de la presentación en vivo. El video debe subirse a YouTube/Vimeo/Google Drive y la URL debe incluirse en el README.md del repositorio.

⚠️ **Importante:** Esta opción solo califica para el 50% de la nota total del taller.

12. RÚBRICA DE EVALUACIÓN

Criterio	Peso	Descripción
Integración Webhooks	25%	Correcta implementación del publisher con firma HMAC, retry logic y gestión de suscriptores. Webhooks enviados correctamente a Edge Functions.
Edge Functions	25%	Funciones serverless desplegadas correctamente. Validación de firma HMAC, timestamp y procesamiento idempotente implementados.
Estrategia Avanzada	30%	Implementación técnica completa y funcional de la opción elegida (DLQ, Idempotencia, Circuit Breaker o Signature Chain). Código robusto y bien documentado.
Observabilidad	10%	Logs estructurados, correlation IDs, métricas visibles. Capacidad de rastrear eventos a través de todo el sistema.
Demo de Resiliencia	10%	Prueba en vivo que demuestra convincentemente el manejo de fallos específico de la estrategia elegida. Sistema se recupera correctamente.

Nota: Se permite utilizar librerías de terceros para facilitar la implementación (ej. nestjs-throttler, bull, drivers de CDC), siempre que se explique su configuración y funcionamiento en la documentación.

13. RECURSOS Y ENLACES DE INTERÉS

13.1 DOCUMENTACIÓN OFICIAL

- **NestJS Microservices:** https://docs.nestjs.com/microservices/basics
- **RabbitMQ Tutorials:** https://www.rabbitmq.com/getstarted.html
- **Supabase Edge Functions:** https://supabase.com/docs/guides/functions
- **TypeORM Documentation:** https://typeorm.io/
- **Bull Queue:** https://docs.bullmq.io/

13.2 PATRONES DE RESILIENCIA

- **Transactional Outbox Pattern:** https://microservices.io/patterns/data/transactional-outbox.html
- **Idempotent Consumer:** https://microservices.io/patterns/communication-style/idempotent-consumer.html
- **Circuit Breaker Pattern:** https://martinfowler.com/bliki/CircuitBreaker.html
- **Saga Pattern:** https://microservices.io/patterns/data/saga.html

13.3 HERRAMIENTAS DE TESTING

- **Webhook.site** (https://webhook.site/): Inspector de webhooks online
- **ngrok** (https://ngrok.com/): Túnel para desarrollo local
- **RequestBin** (https://requestbin.com/): Capturar y analizar webhooks
- **Postman** (https://www.postman.com/): Testing y simulación de APIs

13.4 TELEGRAM BOT API

- **Crear Bot:** https://core.telegram.org/bots#creating-a-new-bot
- **Bot API Documentation:** https://core.telegram.org/bots/api
- **sendMessage:** https://core.telegram.org/bots/api#sendmessage

14. CONSEJOS Y MEJORES PRÁCTICAS

Comenzar Simple: Implementar primero el flujo básico (webhook sin retry) y luego agregar complejidad progresivamente.

Logs Estructurados: Usar console.log con objetos JSON estructurados que incluyan timestamp, correlation_id, event_type.

Secrets Management: NUNCA commitear secrets al repositorio. Usar .env y .env.example.

Testing Incremental: Probar cada componente individualmente antes de integrar todo el sistema.

Documentar Decisiones: Incluir comentarios en el código explicando POR QUÉ se tomaron decisiones arquitectónicas específicas.

Monitoreo Proactivo: Implementar dashboards simples para visualizar entregas exitosas vs fallidas.

Timeouts Apropiados: Configurar timeouts razonables (5-10 segundos) para evitar bloqueos.

Idempotencia Siempre: Incluso si no eligen la opción B, implementar idempotencia básica en todas las Edge Functions.

Git Commits Descriptivos: Hacer commits frecuentes con mensajes claros: "feat: add HMAC validation" en lugar de "update code".

README Como Tutorial: El README debe permitir que otro equipo replique su proyecto siguiendo los pasos sin necesidad de preguntar.

15. PREGUNTAS FRECUENTES (FAQ)

P: ¿Podemos usar otro proveedor de serverless que no sea Supabase?

R: No. El taller requiere específicamente Supabase Edge Functions para mantener consistencia en la evaluación. Supabase tiene tier gratuito suficiente.

P: ¿Es obligatorio usar Telegram para notificaciones?

R: No. Pueden usar Email (SendGrid, AWS SES) o SMS (Twilio). Telegram es recomendado por ser gratis y fácil de configurar.

P: ¿Podemos implementar más de una estrategia avanzada?

R: Sí, pero solo se evaluará una. Es mejor hacer una estrategia excelente que dos mediocres.

P: ¿Qué pasa si RabbitMQ está caído al momento de la presentación?

R: Tener docker-compose.yml preparado para levantar infraestructura rápidamente. Pueden usar RabbitMQ en CloudAMQP (tier gratuito) como respaldo.

P: ¿Es necesario implementar el API Gateway si ya tenemos uno del Taller 1?

R: Sí, deben extenderlo para que incluya los nuevos endpoints relacionados con webhooks.

P: ¿Cómo manejamos secrets en el repositorio?

R: Crear archivo .env.example con valores de ejemplo. El .env real debe estar en .gitignore. En el README explicar qué secrets se necesitan.

P: ¿Podemos usar servicios pagos como Temporal.io?

R: Solo si tienen tier gratuito suficiente. No se requiere que paguen por servicios. Temporal.io tiene tier gratuito.

P: Si implementamos Circuit Breaker, ¿necesitamos Redis?

R: Recomendado pero no obligatorio. Pueden usar PostgreSQL para guardar estado del circuit, aunque será menos performante.

P: ¿Cuánto tiempo debemos dedicar al taller de clase?

R: Estimado: 2-3 horas de trabajo en grupo. Distribuir el trabajo: una persona en Edge Functions, otra en Publishers, otra en estrategia avanzada.

P: ¿Es obligatorio que el microservicio1 se comunique con el microservicio2 en este taller?

R: Si, es parte importante de este taller.

¡Éxito en la implementación!