UpTrack: Sistema de Monitoreo de Disponibilidad de Servicios Web

Alexis Jara • César López

Carrera de Computación • Desarrollo Basado en Plataformas

alexis.jara@unl.edu.ec • cesar.e.lopez@unl.edu.ec

RESUMEN— UpTrack es una plataforma de monitoreo de disponibilidad HTTP/TCP que ejecuta verificaciones periódicas mediante un Worker Pool con hasta 100 goroutines concurrentes en Go 1.21+. Implementa un algoritmo de detección por consenso que realiza hasta 12 pings por sesión, alcanzando 3 resultados consecutivos iguales para determinar el estado (UP, DOWN, UNSTABLE, FLAPPING). La persistencia en PostgreSQL 15 utiliza UUID v7 como PK, con retención de 30 días para métricas de alta escritura y 90 días para resultados de cambios de estado. El frontend React 19 se integra mediante API REST con autenticación JWT, complementado por una app móvil React Native con widget Android nativo.

I. PROBLEMA

Los desarrolladores y administradores de sistemas necesitan verificar continuamente la disponibilidad de servicios críticos (APIs REST, sitios web). Las soluciones existentes presentan limitaciones: servicios comerciales costosos (Pingdom, UptimeRobot Pro), herramientas on-premise complejas (Nagios, Zabbix) que requieren expertise, y falta de integración móvil para consultas rápidas. UpTrack resuelve esto ofreciendo monitoreo multi-tenant con health checks HTTP/TCP configurables (60s-24h), detección inteligente de 6 estados, y notificaciones instantáneas vía Telegram con vinculación sin fricción mediante Magic Link.

II. ARQUITECTURA DEL SISTEMA

El sistema implementa Clean Architecture con separación de responsabilidades en 4 capas:

- **Presentation:** Handlers HTTP (Gin 1.9+), middleware JWT, DTOs Request/Response, documentación Swagger.
- **Application:** OrchestrationService coordina HealthChecker, ResultAnalyzer y StateUpdater.
- **Domain:** Entidades (MonitoringTarget, CheckResult), Value Objects (TargetId UUID v7, TargetStatus enum).
- **Infrastructure:** Repositorios PostgreSQL (GORM), Telegram Bot API, autenticación JWT (bcrypt cost 10).

III. DESARROLLO DE LA SOLUCIÓN

A. Stack Tecnológico

Backend: Go 1.21+, Gin, GORM, PostgreSQL 15, JWT, Worker Pool concurrente.
Frontend Web: React 19, TypeScript 5.9, Vite 7.2, Tailwind CSS 4.1, Recharts 3.5.
App Móvil: React Native 0.81, Expo 54, AsyncStorage 2.2, Widget Android nativo.

B. Componentes Principales

Scheduler: Polling cada 10s con query optimizada (índice idx_monitoring_targets_next_check_at).
Worker Pool: Hasta 100 goroutines concurrentes procesando checks en paralelo.
Algoritmo de Consenso: Hasta 12 pings por sesión, determina estado con 3 resultados consecutivos iguales.

```
// Detección de estado UP: 3 éxitos en ≤4 intentos DOWN: 3 fallos consecutivos UNSTABLE: 3 iguales en 5-9 intentos FLAPPING: Sin consenso en 12 intentos
```

IV. PERSISTENCIA DE DATOS

PostgreSQL 15 con UUID v7 como PK (ordenamiento temporal nativo). Estrategia de retención: Metrics (30 días, alta escritura), CheckResults (90 días, solo cambios de estado), Statistics (indefinida, agregados). Índices optimizados: idx_monitoring_targets_next_check_at, índices compuestos en (target_id, timestamp).

V. INTERFACES DE USUARIO

Web: React 19 + Vite, API REST con autenticación JWT, gráficos Recharts (latencia, heatmaps), deployment Nginx con SPA routing.
Móvil: Expo 54 + React Native, Widget Android 2x1 celdas con actualización periódica, caché offline AsyncStorage, navegación sin dependencias externas.

VI. DIAGRAMA DE COMPONENTES C4

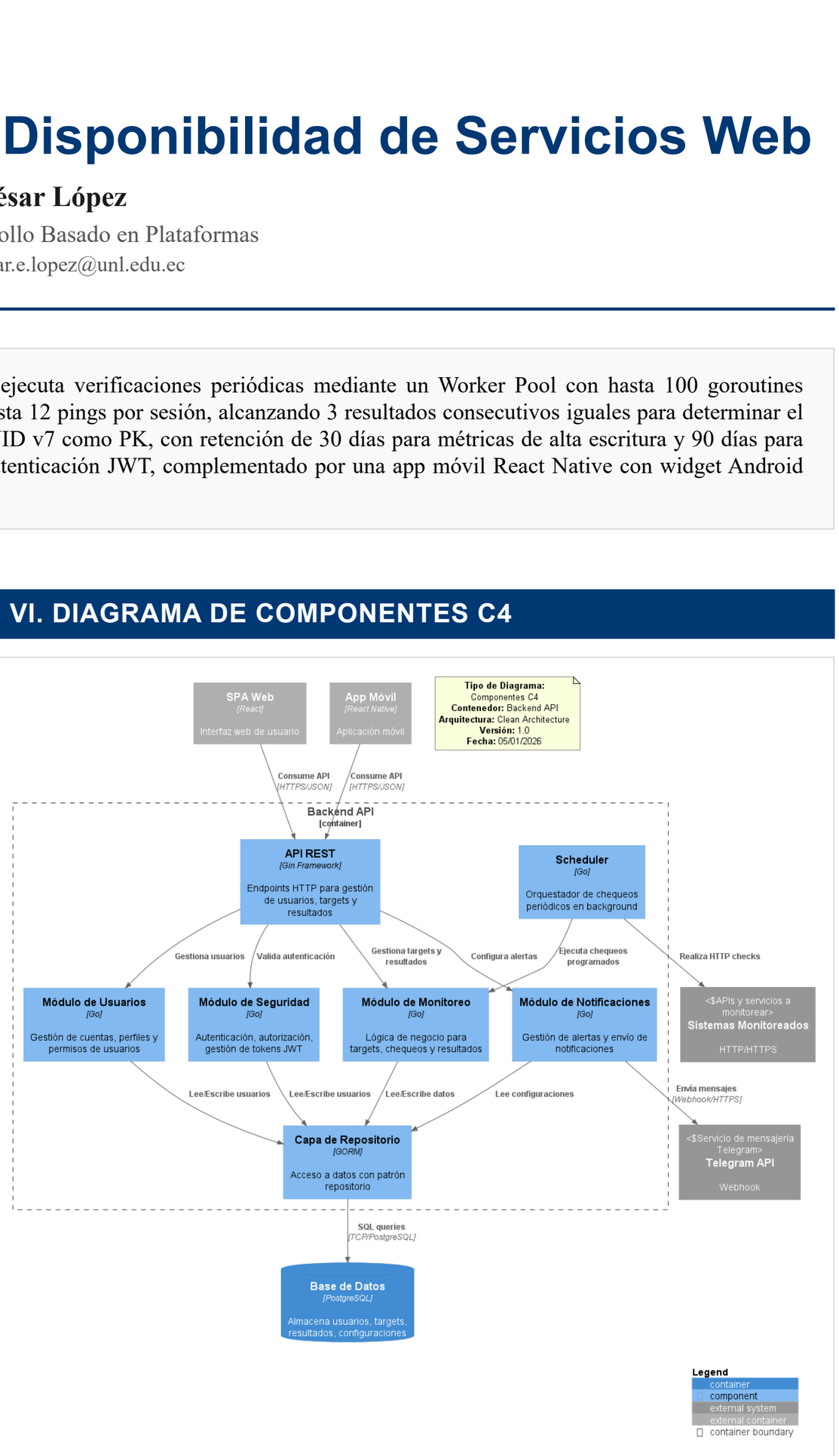

Figura 1: Diagrama de Componentes C4 - Backend UpTrack

VII. RESULTADOS

Funcionalidades Implementadas:
• Monitoreo multi-tenant con aislamiento por usuario (tabla users con FK en monitoring_targets).
• Health checks configurables: intervalo 60s-24h, timeout 10s, retry 3 intentos.
• Notificaciones Telegram con Magic Link (token TTL 15 min, vinculación sin fricción).
• Dashboard web: estadísticas en tiempo real, gráficos de latencia histórica, filtros por estado.
• App móvil: widget Android nativo, consulta offline con caché local.

Beneficios: Reducción de notificaciones espurias mediante consenso (3/12 pings), escalabilidad horizontal con Worker Pool (hasta 100 goroutines), persistencia optimizada con UUID v7 (orden temporal sin auto-increment).

VIII. CONCLUSIONES Y TRABAJO FUTURO

Lecciones: Clean Architecture facilita testing unitario por capas, Go channels permiten concurrencia sin race conditions, UUID v7 mejora rendimiento en índices B-tree vs auto-increment.

Mejoras Futuras: Push notifications Firebase/APNs, widget iOS, checks ICMP/DNS adicionales, exportación de reportes PDF, integración Slack/Email, dashboard de métricas agregadas por equipo.

IX. REFERENCIAS

[1] Go Team, "The Go Programming Language Specification," https://go.dev/ref/spec, 2024.
[2] PostgreSQL Global Development Group, "PostgreSQL 15 Documentation," https://www.postgresql.org/docs/15/, 2024.
[3] React Team, "React Documentation," https://react.dev/, 2024.
[4] Telegram, "Telegram Bot API Documentation," https://core.telegram.org/bots/api, 2024.