# DOCUMENTO DE ARQUITECTURA

# DEL SOFTWARE

# DE TAPTRACK

# 2025

Documento oficial del proyecto TAPTRACK, que tiene como fin brindar una vista macro de la estructura del proyecto que sirva como base para el entendimiento de los deberes del equipo y de la estructura interna para el usuario final.

**DOCUMENTO DE ARQUITECTURA DE SOFTWARE**

## Historial de versiones

| Versión | Fecha | Descripción | Autor |
|---|---|---|---|
| 1.0 | 2025-04-12 | Versión inicial- reléase 01 | Duvan Camilo Amaya Urrego |

# DOCUMENTO DE ARQUITECTURA DE SOFTWARE

**Contenido**



Documento oficial del proyecto TAPTRACK, que tiene como fin brindar una vista macro de la estructura del proyecto que sirva como base para el entendimiento de los deberes del equipo y de la estructura interna para el usuario final.

**DOCUMENTO DE ARQUITECTURA DE SOFTWARE**

## 1. Introducción

El presente documento describe la arquitectura, diseño y buenas prácticas de desarrollo del aplicativo **BarSync**, un sistema de control de inventario diseñado para el proyecto **TrapTrack**, que gestiona tres sedes de un bar. **BarSync** tiene como objetivo centralizar y automatizar el seguimiento de insumos, bebidas y productos en tiempo real, optimizando el flujo de operaciones, reduciendo pérdidas por mermas y facilitando la toma de decisiones basada en datos.

### 1.1 Propósito

Este documento sirve como guía técnica para el equipo de desarrollo, administradores y stakeholders, detallando:

- La estructura arquitectónica del sistema.
- Los módulos clave y su interacción.
- Los criterios de despliegue en las tres sedes.
- Las políticas de desarrollo y seguridad.

### 1.2 Alcance

**BarSync** cubrirá las siguientes funcionalidades principales:

1. **Gestión de inventario**: Registro de entradas, salidas y stock crítico.
2. **Sincronización multisede**: Actualización en tiempo real entre locales.
3. **Reportes automatizados**: Alertas de reposición, tendencias de consumo y auditorías.

### 1.3 Tecnologías Principales

- **Frontend**: *React.js* (interfaz intuitiva para dispositivos móviles y web).
- **Backend**: *Node.js con Express* (API REST para gestión de datos).
- **Base de Datos**: *Mysql* (escalabilidad para inventario dinámico) + *Redis* (caché para sincronización en tiempo real).
- **Infraestructura**: *AWS EC2* (servidores) + *Firebase* (notificaciones push para alertas).
- **Autenticación**: *JWT* (acceso seguro por roles: administrador, sede, empleado).

### 1.4 Audiencia

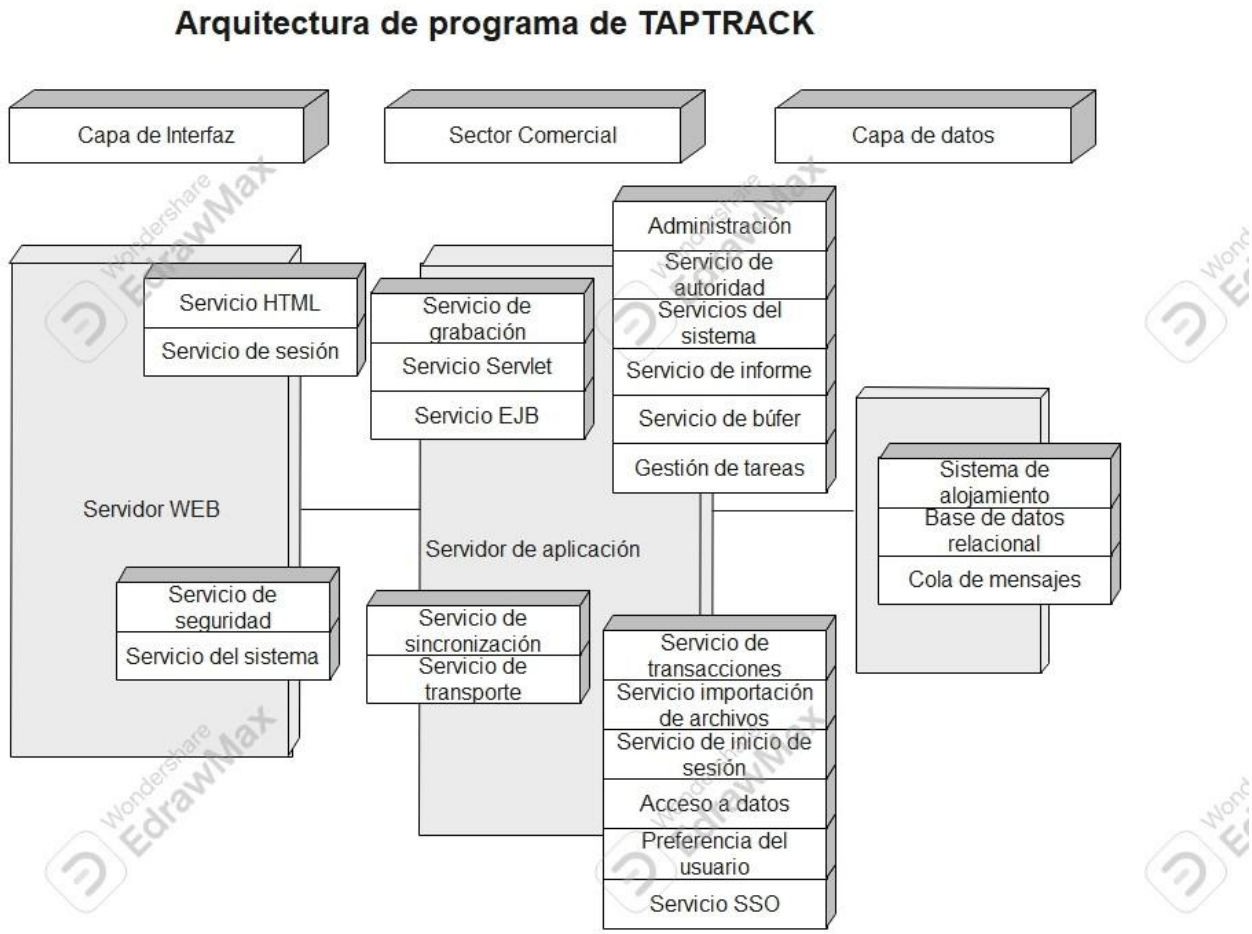
- **Dueños y gerentes**: Monitoreo consolidado de todas las sedes.

- **Personal de bar**: Registro rápido de insumos usados.
- **Equipo de desarrollo**: Implementación y mantenimiento.

**BarSync** busca reducir en un *40% las pérdidas por inventario no registrado* y mejorar la eficiencia en pedidos mediante integración con proveedores.

## 2. Diagrama de Arquitectura

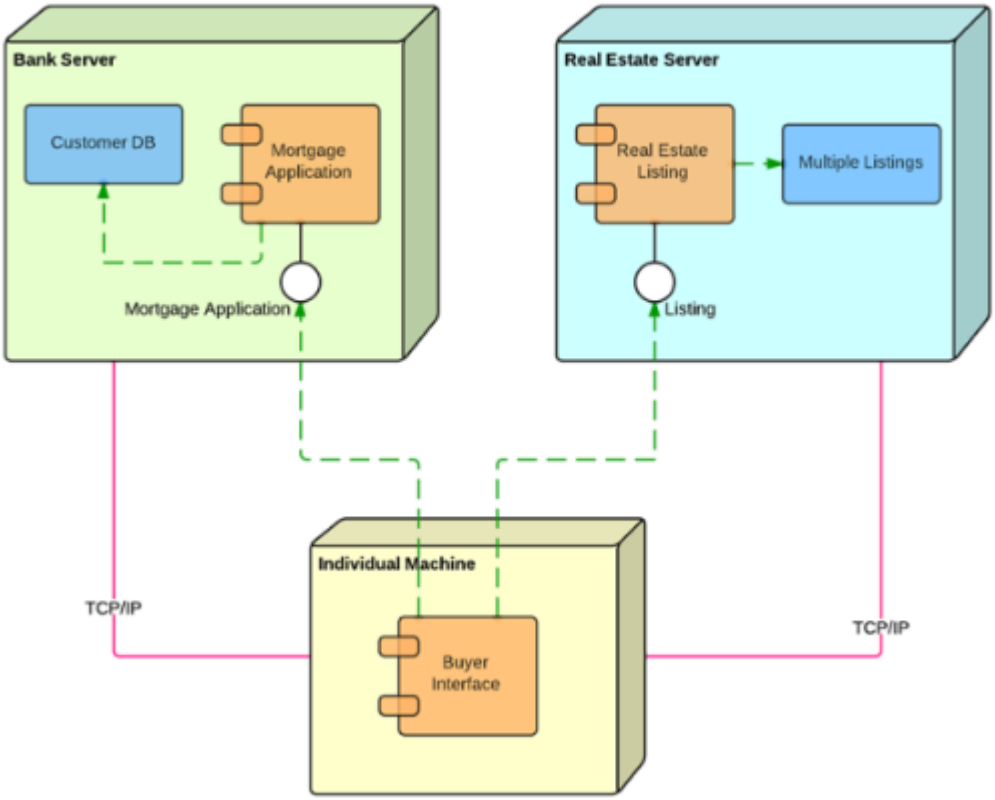
### 2.1 Diagrama



El diagrama de arquitectura es importante y necesario porque proporciona una representación visual clara y estructurada de los componentes clave de un sistema, sus interacciones y flujos de datos, lo que facilita la comprensión, el diseño y la comunicación entre equipos técnicos y no técnicos. Este diagrama actúa como un mapa que guía el desarrollo, ayuda a identificar posibles problemas de diseño antes de la implementación, asegura la consistencia en la estructura del proyecto y sirve como documentación de referencia para futuras actualizaciones o mantenimiento. En el caso específico de MVC, el diagrama clarifica la separación de responsabilidades entre modelo, vista y controlador, promoviendo un código más organizado, escalable y fácil de mantener.

## 3. Diagrama de despliegue

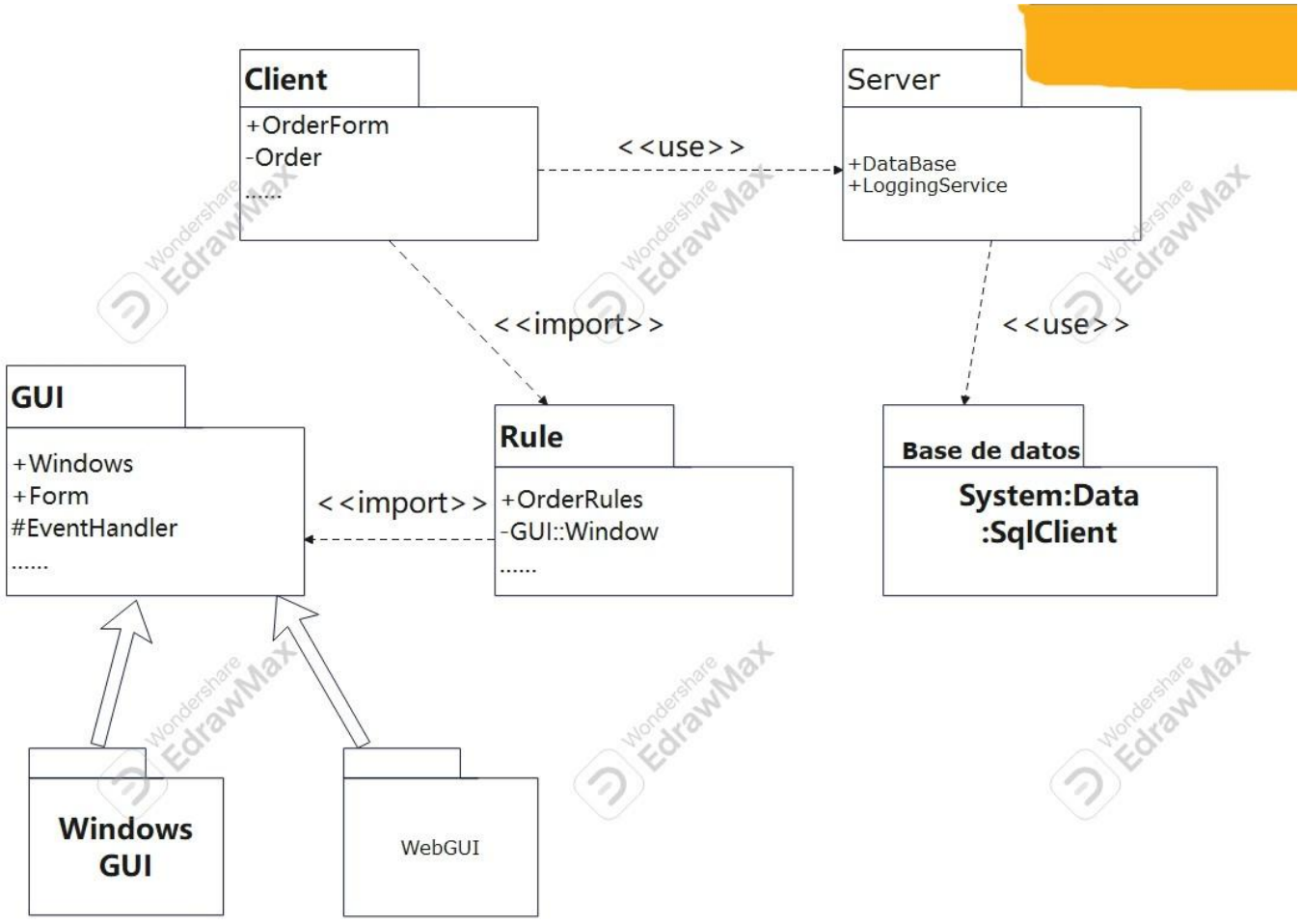
### 3.1 Diagrama



El diagrama de despliegue UML es crucial para mostrar cómo los componentes de un sistema MVC se distribuyen físicamente en servidores, dispositivos o entornos en la nube. A diferencia del diagrama de componentes (que es lógico), este se enfoca en la infraestructura física y las conexiones de red.

## 4. Diagrama de Componentes

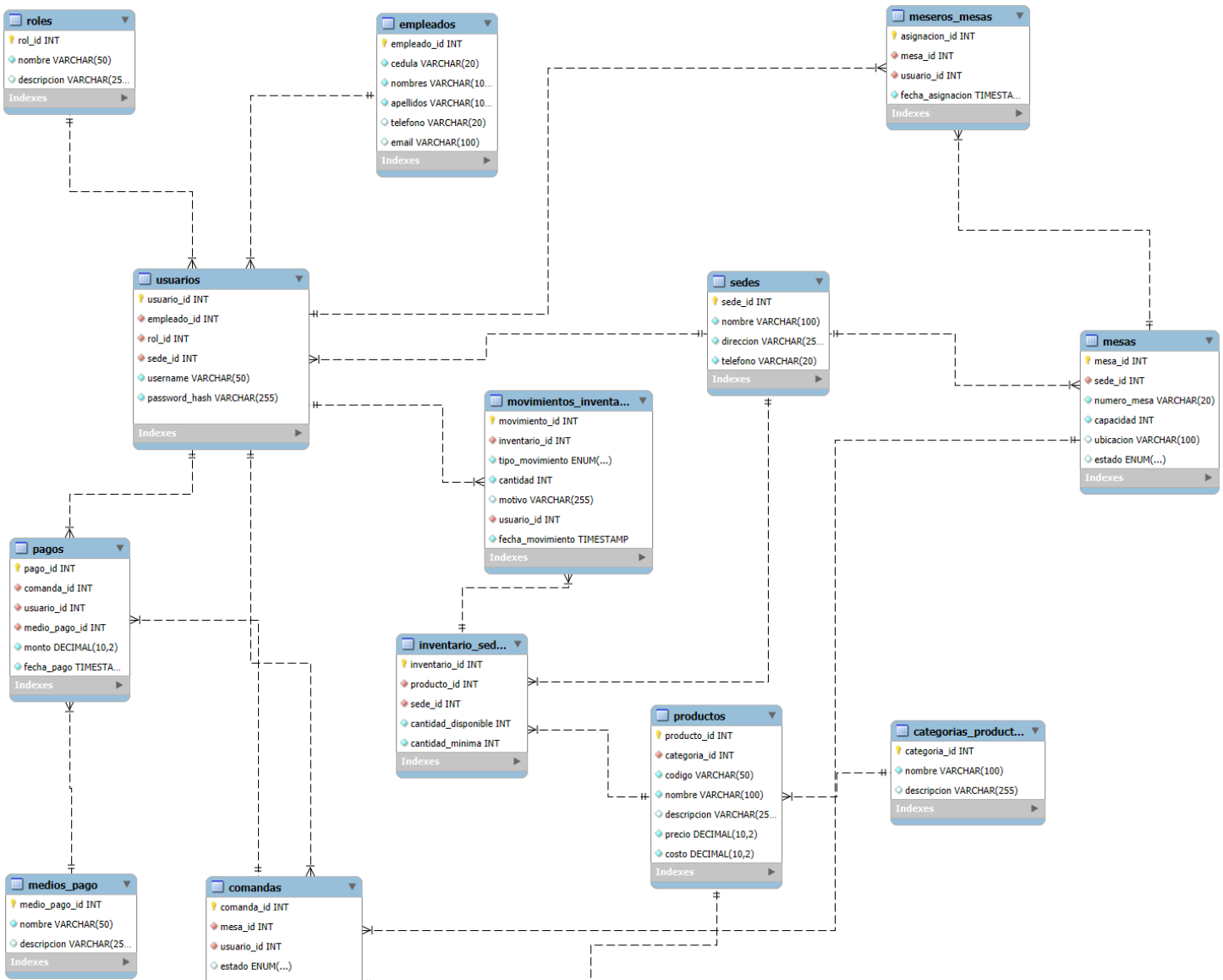
### 4.1 Diagrama



El diagrama de componentes en la arquitectura UML es esencial porque desglosa visualmente los módulos funcionales, sus dependencias y cómo interactúan entre sí. Este diagrama:

1. Clarifica el modularidad: Muestra cómo cada componente (ej: UserController, DatabaseModel, UI/Views) opera de forma independiente pero se integra mediante interfaces bien definidas.
2. Identifica reutilización: Destaca componentes compartidos (ej: servicios de autenticación o APIs) que pueden ser usados en múltiples partes del sistema.
3. Facilita el escalamiento: Ayuda a planificar la adición de nuevos módulos sin afectar el funcionamiento existente.
4. Documenta el sistema: Sirve como referencia técnica para desarrolladores y stakeholders, asegurando coherencia en el desarrollo y mantenimiento.

## 5. Diagrama de Modelo Entidad Relación



## 6. Buenas practicas de desarrollo

| ID | Nombre Buena Practica | Descripción | Aplica |
|---|---|---|---|
| **1** | Control de Versiones | Usar herramientas como Git para rastrear cambios y colaborar eficientemente. | Todos los proyectos |
| **2** | Código Limpio | Seguir convenciones de legibilidad, nombres descriptivos y principios SOLID. | Todos los proyectos |
| **3** | Pruebas Automatizadas | Implementar pruebas unitarias, de integración y E2E para garantizar calidad. | Proyectos críticos |
| **4** | Documentación | Mantener documentación actualizada (código, APIs, manuales de usuario). | Todos los proyectos |
| **5** | Integración Continua | Automatizar builds y pruebas al fusionar cambios (ej. Jenkins, GitHub Actions). | Equipos >1 persona |
| **6** | Seguridad básica | Validar inputs, evitar hardcodeo de credenciales, usar HTTPS. | Proyectos con datos sensibles |

| | | | |
|---|---|---|---|
| **7** | Patrones de Diseño | Aplicar patrones adecuados (ej. MVC, Singleton) para escalabilidad. | Proyectos complejos |
| **8** | Logging y Monitoreo | Registrar eventos clave y errores para diagnóstico (ej. ELK, Prometheus). | Entornos de producción |
| **9** | KISS y DRY | Evitar sobreingeniería ("Keep It Simple") y duplicación de código. | Todos los proyectos |
| **10** | Revisiones de Código | Realizar *code reviews* para detectar bugs y compartir conocimiento. | Equipos colaborativos |