**Sistema de Gestión de Flota**

PepsiCo Chile

ARQUITECTURA DEL SISTEMA

*Diseño Técnico y Componentes*

Fecha: 14 de octubre de 2025

Equipo: Joaquín Marín y Benjamin Vilches

Duoc UC - Proyecto de Título 2025

# ARQUITECTURA DEL SISTEMA

Proyecto: Plataforma de Gestión de Ingreso de Vehículos - PepsiCo Chile

Fecha: 13/10/2025

Responsables: Joaquín Marín y Benjamin Vilches

Versión: 1.0

1. VISTA GENERAL DE LA ARQUITECTURA

Tipo de Arquitectura: Cliente-Servidor (3 Capas)

Patrón Arquitectónico: Model-View-Controller (MVC) + API REST

Estilo: Single Page Application (SPA) con Backend API

┌────────────────────────────────────┐
│ USUARIOS FINALES │
│ (Guardias, Mecánicos, Jefes, etc) │
└────────────────────────────────────┘
│
│ HTTPS
▼
┌───────────────────────────────────────────────────────────────┐
│ CAPA DE PRESENTACIÓN │
│ (Frontend - React SPA) │
│ │
│ ┌──────────────┐ ┌──────────────┐ ┌──────────────┐ │
│ │ Dashboard │ │ Ingresos │ │ Órdenes de │ │
│ │ │ │ de Vehículos│ │ Trabajo │ │
│ └──────────────┘ └──────────────┘ └──────────────┘ │
│ │
│ ┌──────────────┐ ┌──────────────┐ ┌──────────────┐ │
│ │ Inventario │ │ Reportes │ │ Usuarios │ │
│ │ Repuestos │ │ │ │ y Roles │ │
│ └──────────────┘ └──────────────┘ └──────────────┘ │
│ │
│ React Router | Zustand | React Query | Axios │
└───────────────────────────────────────────────────────────────┘
│
│ REST API (JSON)
│ JWT Authentication
▼
┌───────────────────────────────────────────────────────────────┐
│ CAPA DE APLICACIÓN │
│ (Backend - Node.js API) │
│ │
│ ┌─────────────────────────────────────────────────────┐ │
│ │ API Gateway / Routes │ │
│ │ /api/auth | /api/vehicles | /api/entries | /api/ot │ │
│ └─────────────────────────────────────────────────────┘ │
│ │ │
│ ┌─────────────────────────────────────────────────────┐ │
│ │ Middlewares │ │
│ │ Auth | CORS | Rate Limit | Validation | Logger │ │
│ └─────────────────────────────────────────────────────┘ │
│ │ │
│ ┌──────────────┐ ┌──────────────┐ ┌──────────────┐ │
│ │ Controllers │ │ Services │ │ Validators │ │
│ │ (HTTP Logic) │ │ (Business │ │ (Zod) │ │
│ │ │ │ Logic) │ │ │ │
│ └──────────────┘ └──────────────┘ └──────────────┘ │
│ │
│ Express.js | JWT | bcrypt | Prisma ORM │
└───────────────────────────────────────────────────────────────┘
│
│ SQL Queries
│ Prisma Client
▼
┌───────────────────────────────────────────────────────────────┐
│ CAPA DE DATOS │
│ (PostgreSQL Database) │
│ │
│ ┌──────────────┐ ┌──────────────┐ ┌──────────────┐ │
│ │ Usuarios │ │ Vehículos │ │ Ingresos │ │
│ │ y Roles │ │ y Talleres │ │ y OT │ │
│ └──────────────┘ └──────────────┘ └──────────────┘ │
│ │
│ ┌──────────────┐ ┌──────────────┐ ┌──────────────┐ │
│ │ Repuestos │ │ Notificaciones│ │ Auditoría │ │
│ │ e Inventario│ │ y Documentos │ │ y Logs │ │
│ └──────────────┘ └──────────────┘ └──────────────┘ │
│ │
│ PostgreSQL 15+ │
└───────────────────────────────────────────────────────────────┘

┌───────────────────────────────────────────────────────────────┐
│ SERVICIOS EXTERNOS │
│ │
│ ┌──────────────┐ ┌──────────────┐ ┌──────────────┐ │
│ │ Cloudinary │ │ Nodemailer │ │ Winston │ │
│ │ (Imágenes) │ │ (Email) │ │ (Logs) │ │
│ └──────────────┘ └──────────────┘ └──────────────┘ │
└───────────────────────────────────────────────────────────────┘

2. DIAGRAMA DE COMPONENTES DETALLADO

## 2.1. FRONTEND (React Application)

────────────────────────────────────────────────────────────────

┌─────────────────────────────────────────────────────────────────┐
│ FRONTEND LAYER │
│ │
│ ┌───────────────────────────────────────────────────────┐ │
│ │ App.tsx │ │
│ │ (Entry Point + Router) │ │
│ └───────────────────────────────────────────────────────┘ │
│ │ │
│ ┌─────────────────────────┴─────────────────────────────┐ │
│ │ │ │
│ ▼ ▼ ▼ │
│ ┌──────────────┐ ┌──────────────┐ ┌──────────────┐ │
│ │ Layout │ │ Protected │ │ Public │ │
│ │ Components │ │ Routes │ │ Routes │ │
│ │ │ │ │ │ │ │
│ │ - Header │ │ - Dashboard │ │ - Login │ │
│ │ - Sidebar │ │ - Vehicles │ │ - Forgot Pwd │ │
│ │ - Footer │ │ - Entries │ │ │ │
│ └──────────────┘ │ - WorkOrders │ └──────────────┘ │
│ │ - Inventory │ │
│ │ - Reports │ │
│ │ - Users │ │
│ └──────────────┘ │
│ │
│ ┌───────────────────────────────────────────────────────┐ │
│ │ STATE MANAGEMENT │ │
│ │ │ │
│ │ ┌──────────────┐ ┌──────────────┐ │ │
│ │ │ Zustand │ │ React Query │ │ │
│ │ │ │ │ │ │ │
│ │ │ - Auth State │ │ - Server │ │ │
│ │ │ - User Info │ │ Data Cache │ │ │
│ │ │ - UI State │ │ - Mutations │ │ │
│ │ │ - Filters │ │ - Queries │ │ │
│ │ └──────────────┘ └──────────────┘ │ │
│ └───────────────────────────────────────────────────────┘ │
│ │
│ ┌───────────────────────────────────────────────────────┐ │
│ │ SERVICES LAYER │ │
│ │ │ │
│ │ ┌──────────────┐ ┌──────────────┐ ┌──────────────┐│ │
│ │ │ API │ │ Auth │ │ Upload ││ │
│ │ │ Client │ │ Service │ │ Service ││ │
│ │ │ (Axios) │ │ │ │ ││ │
│ │ └──────────────┘ └──────────────┘ └──────────────┘│ │
│ └───────────────────────────────────────────────────────┘ │
│ │
│ ┌───────────────────────────────────────────────────────┐ │
│ │ SHARED COMPONENTS │ │
│ │ │ │
│ │ - Button | Input | Select | Modal | Table | Card │ │
│ │ - DatePicker | FileUpload | Notifications | Loader │ │
│ │ - Dropdown | Badge | Avatar | Tabs | Accordion │ │
│ └───────────────────────────────────────────────────────┘ │
│ │
│ ┌───────────────────────────────────────────────────────┐ │
│ │ HOOKS │ │
│ │ │ │
│ │ - useAuth | usePermissions | useVehicles | useOT │ │
│ │ - useInventory | useNotifications | useDebounce │ │
│ └───────────────────────────────────────────────────────┘ │
└─────────────────────────────────────────────────────────────────┘

## 2.2. BACKEND (Node.js API)

────────────────────────────────────────────────────────────────

┌─────────────────────────────────────────────────────────────────┐
│ BACKEND LAYER │
│ │
│ ┌───────────────────────────────────────────────────────┐ │
│ │ index.ts │ │
│ │ (Server Initialization) │ │
│ └───────────────────────────────────────────────────────┘ │
│ │ │
│ ┌─────────────────────────┴─────────────────────────────┐ │
│ │ MIDDLEWARE STACK │ │
│ │ │ │
│ │ 1. cors() - CORS Policy │ │
│ │ 2. helmet() - Security Headers │ │
│ │ 3. express.json() - Body Parser │ │
│ │ 4. rateLimit() - Rate Limiting │ │
│ │ 5. logger - Request Logging │ │
│ │ 6. authenticate - JWT Verification │ │
│ │ 7. authorize - RBAC Check │ │
│ │ 8. validate - Zod Validation │ │
│ └───────────────────────────────────────────────────────┘ │
│ │ │
│ ┌─────────────────────────┴─────────────────────────────┐ │
│ │ ROUTES │ │
│ │ │ │
│ │ /api/auth - Authentication │ │
│ │ /api/users - User Management │ │
│ │ /api/roles - Role Management │ │
│ │ /api/vehicles - Vehicle CRUD │ │
│ │ /api/entries - Vehicle Entries │ │
│ │ /api/work-orders - Work Orders │ │
│ │ /api/spare-parts - Inventory │ │
│ │ /api/workshops - Workshop Management │ │
│ │ /api/notifications - Notifications │ │
│ │ /api/reports - Report Generation │ │
│ │ /api/upload - File Upload │ │
│ └───────────────────────────────────────────────────────┘ │
│ │ │
│ ┌─────────────────────────┴─────────────────────────────┐ │
│ │ CONTROLLERS │ │
│ │ (HTTP Request/Response Handling) │ │
│ │ │ │
│ │ - AuthController - UserController │ │
│ │ - VehicleController - EntryController │ │
│ │ - WorkOrderController - InventoryController │ │
│ │ - ReportController - NotificationController │ │
│ └───────────────────────────────────────────────────────┘ │
│ │ │
│ ┌─────────────────────────┴─────────────────────────────┐ │
│ │ SERVICES │ │
│ │ (Business Logic Layer) │ │
│ │ │ │
│ │ - AuthService - UserService │ │
│ │ - VehicleService - EntryService │ │
│ │ - WorkOrderService - InventoryService │ │
│ │ - NotificationService - ReportService │ │
│ │ - EmailService - UploadService │ │
│ │ - AuditService - PermissionService │ │
│ └───────────────────────────────────────────────────────┘ │
│ │ │
│ ┌─────────────────────────┴─────────────────────────────┐ │
│ │ PRISMA ORM CLIENT │ │
│ │ (Database Access Layer) │ │
│ │ │ │
│ │ - prisma.user - prisma.vehicle │ │
│ │ - prisma.workOrder - prisma.sparePart │ │
│ │ - prisma.notification - prisma.auditLog │ │
│ │ - Transactions - Batch Operations │ │
│ └───────────────────────────────────────────────────────┘ │
│ │ │
│ ┌─────────────────────────┴─────────────────────────────┐ │
│ │ UTILITIES │ │
│ │ │ │
│ │ - JWT Utils - Password Utils │ │
│ │ - Date Utils - File Utils │ │
│ │ - Validation Utils - Error Handler │ │
│ │ - Response Builder - Logger │ │
│ └───────────────────────────────────────────────────────┘ │
└─────────────────────────────────────────────────────────────────┘

3. FLUJO DE DATOS (DATA FLOW)

## 3.1. Flujo de Autenticación

────────────────────────────────────────────────────────────────

┌────────┐ ┌──────────┐ ┌──────────┐ ┌──────────┐
│ User │ │ Frontend │ │ Backend │ │ Database │
└───┬────┘ └────┬─────┘ └────┬─────┘ └────┬─────┘
│ │ │ │
│ 1. Enter creds │ │ │
├──────────────────>│ │ │
│ │ │ │
│ │ 2. POST /api/auth/login │
│ ├────────────────────>│ │
│ │ │ │
│ │ │ 3. Query user │
│ │ ├────────────────────>│
│ │ │ │
│ │ │ 4. User data │
│ │ │<────────────────────┤
│ │ │ │
│ │ │ 5. Verify password │
│ │ │ (bcrypt) │
│ │ │ │
│ │ │ 6. Generate JWT │
│ │ │ │
│ │ │ 7. Log audit │
│ │ ├────────────────────>│
│ │ │ │
│ │ 8. Return token │ │
│ │<────────────────────┤ │
│ │ │ │
│ │ 9. Store in state │ │
│ │ (Zustand) │ │
│ │ │ │
│ 10. Redirect │ │ │
│<──────────────────┤ │ │
│ │ │ │

## 3.2. Flujo de Creación de Orden de Trabajo

────────────────────────────────────────────────────────────────

User → Frontend → Backend → Database

Guardia registra ingreso de vehículo

└─> POST /api/entries
└─> EntryService.create()
└─> prisma.vehicleEntry.create()
└─> NotificationService.notify(recepcionista)

Recepcionista valida documentos y crea OT

└─> POST /api/work-orders
└─> WorkOrderService.create()
└─> prisma.workOrder.create()
└─> Generate OT number
└─> NotificationService.notify(jefe\_taller)

Jefe de taller asigna mecánico

└─> PATCH /api/work-orders/:id/assign
└─> WorkOrderService.assignMechanic()
└─> prisma.workOrder.update()
└─> NotificationService.notify(mecanico)

Mecánico actualiza estado

└─> PATCH /api/work-orders/:id/status
└─> WorkOrderService.updateStatus()
└─> prisma.workOrderStatus.create()
└─> prisma.workOrder.update()
└─> AuditService.log()

## 3.3. Flujo de Gestión de Inventario

────────────────────────────────────────────────────────────────

Mecánico solicita repuesto

└─> POST /api/work-orders/:id/spare-parts
└─> InventoryService.requestPart()
└─> prisma.workOrderSparePart.create()
└─> NotificationService.notify(asistente\_repuestos)

Asistente entrega repuesto

└─> PATCH /api/work-orders/:id/spare-parts/:partId/deliver
└─> InventoryService.deliverPart()
└─> BEGIN TRANSACTION
└─> prisma.workOrderSparePart.update()
└─> prisma.sparePartMovement.create()
└─> prisma.sparePart.update() (stock)
└─> COMMIT TRANSACTION
└─> Check stock\_minimo
└─> If low: NotificationService.notify(coordinador)

4. PATRONES DE DISEÑO UTILIZADOS

## 4.1. Repository Pattern

────────────────────────────────────────────────────────────────

Prisma ORM actúa como Repository, abstrayendo el acceso a datos.

**Ejemplo:**

class VehicleService {

async getAll() {
return prisma.vehicle.findMany();
}

}

## 4.2. Service Layer Pattern

────────────────────────────────────────────────────────────────

Lógica de negocio separada de controllers.

**Ejemplo:**

Controller → Service → Prisma → Database

## 4.3. Middleware Pattern

────────────────────────────────────────────────────────────────

Express middleware para concerns transversales.

**Ejemplo:**

authenticate → authorize → validate → controller

## 4.4. Factory Pattern

────────────────────────────────────────────────────────────────

Generación de números de OT, códigos de ingreso.

**Ejemplo:**

# OT-2025-001

# ING-RM-20251013-001

## 4.5. Observer Pattern

────────────────────────────────────────────────────────────────

Sistema de notificaciones reactivo.

**Ejemplo:**

WorkOrder.create() → trigger → NotificationService.notify()

## 4.6. Singleton Pattern

────────────────────────────────────────────────────────────────

Prisma Client instancia única.

**Ejemplo:**

export const prisma = new PrismaClient();

5. SEGURIDAD

## 5.1. Autenticación

────────────────────────────────────────────────────────────────

✓ JWT (JSON Web Tokens)

* Access Token: 15 minutos
* Refresh Token: 7 días
* Stored in httpOnly cookies (XSS protection)

✓ Password Hashing

* bcrypt con 10 salt rounds
* Nunca se almacenan contraseñas en texto plano

## 5.2. Autorización (RBAC)

────────────────────────────────────────────────────────────────

✓ Role-Based Access Control

* Middleware authorize(permissions)
* Verificación en cada endpoint protegido

**Ejemplo:**

router.patch(

'/work-orders/:id',
authenticate,
authorize(['orders.update']),
workOrderController.update

);

## 5.3. Protección de API

────────────────────────────────────────────────────────────────

✓ Rate Limiting

* 100 requests por 15 minutos por IP
* 5 intentos de login por 15 minutos

✓ CORS

* Solo orígenes permitidos
* Credenciales habilitadas

✓ Helmet

* Security headers (XSS, clickjacking, etc.)

✓ Input Validation

* Zod schemas en todos los endpoints
* Sanitización de inputs

## 5.4. Auditoría

────────────────────────────────────────────────────────────────

✓ Audit Logs

* Todas las operaciones críticas
* IP, User Agent, timestamp
* Datos antes/después

6. MANEJO DE ERRORES

## 6.1. Estructura de Errores

────────────────────────────────────────────────────────────────

**Respuesta estándar de error:**

{

"success": false,

"error": {

"code": "VALIDATION\_ERROR",
"message": "Datos inválidos",
"details": [
{
"field": "email",
"message": "Email inválido"
}
]

},

"timestamp": "2025-10-13T10:30:00Z"

}

## 6.2. Tipos de Errores

────────────────────────────────────────────────────────────────

* ValidationError (400)
* AuthenticationError (401)
* AuthorizationError (403)
* NotFoundError (404)
* ConflictError (409)
* InternalServerError (500)

## 6.3. Error Handler Middleware

────────────────────────────────────────────────────────────────

app.use((err, req, res, next) => {

logger.error(err);

if (err instanceof ValidationError) {

return res.status(400).json({...});

}

// ... otros tipos de error

res.status(500).json({

success: false,
error: "Error interno del servidor"

});

});

7. RENDIMIENTO Y OPTIMIZACIÓN

## 7.1. Frontend

────────────────────────────────────────────────────────────────

✓ Code Splitting

* Lazy loading de rutas
* Dynamic imports

✓ React Query Caching

* Cache de datos del servidor
* Stale-while-revalidate

✓ Image Optimization

* Cloudinary CDN
* Lazy loading de imágenes
* Responsive images

✓ Bundle Optimization

* Tree shaking
* Minificación
* Compression (gzip/brotli)

## 7.2. Backend

────────────────────────────────────────────────────────────────

✓ Database Query Optimization

* Índices en campos frecuentes
* Select específico (no SELECT \*)
* Prisma query optimization

✓ Pagination

* Cursor-based pagination
* Límite de resultados

✓ Caching Strategy

* En memoria para datos estáticos
* Redis (futuro, si es necesario)

✓ Async Processing

* Background jobs para emails
* Generación de reportes asíncrona

## 7.3. Database

────────────────────────────────────────────────────────────────

✓ Indexing Strategy

* Primary keys
* Foreign keys
* Campos de búsqueda frecuente

✓ Connection Pooling

* Prisma connection pool
* Max connections: 10

✓ Query Optimization

* Explain analyze para queries lentas
* Evitar N+1 queries

8. ESCALABILIDAD

## 8.1. Horizontal Scaling

────────────────────────────────────────────────────────────────

✓ Stateless API

* JWT (no sessions en servidor)
* Múltiples instancias posibles

✓ Load Balancing

* Ready para Nginx/HAProxy
* Round-robin distribution

## 8.2. Vertical Scaling

────────────────────────────────────────────────────────────────

✓ Database

* PostgreSQL soporta hasta 1000+ conexiones
* Read replicas (futuro)

✓ Backend

* Node.js cluster mode
* PM2 process manager

## 8.3. Crecimiento Futuro

────────────────────────────────────────────────────────────────

✓ Microservices Ready

* Servicios bien separados
* Fácil extracción a microservicios

✓ Message Queue (Futuro)

* RabbitMQ/Redis para async tasks
* Event-driven architecture

9. DISPONIBILIDAD Y RESILIENCIA

## 9.1. Health Checks

────────────────────────────────────────────────────────────────

GET /health

* Database connection
* External services status
* Memory usage

GET /health/ready

* Application ready to serve

## 9.2. Graceful Shutdown

────────────────────────────────────────────────────────────────

process.on('SIGTERM', async () => {

await prisma.$disconnect();

server.close();

});

## 9.3. Error Recovery

────────────────────────────────────────────────────────────────

✓ Database reconnection

✓ Retry logic para servicios externos

✓ Circuit breaker pattern (futuro)

10. MONITOREO Y LOGGING

## 10.1. Application Logging

────────────────────────────────────────────────────────────────

**Winston Logger con niveles:**

* error: Errores críticos
* warn: Advertencias
* info: Información general
* debug: Debugging (solo desarrollo)

**Formato:**

[2025-10-13 10:30:00] [INFO] [UserService] User created: user@example.com

## 10.2. Métricas (Futuro)

────────────────────────────────────────────────────────────────

* Tiempo de respuesta promedio
* Requests por segundo
* Error rate
* CPU y memoria

## 10.3. Alertas

────────────────────────────────────────────────────────────────

* Error rate > 5%
* Response time > 3s
* Database connection failures
* Disk space < 10%

11. DEPLOYMENT ARCHITECTURE

┌─────────────────────────────────────────────────────────────────┐
│ INTERNET │
└────────────────────────────┬────────────────────────────────────┘
│
┌────────────┴────────────┐
│ │
▼ ▼
┌───────────────────┐ ┌───────────────────┐
│ Vercel CDN │ │ Railway/Render │
│ (Frontend) │ │ (Backend API) │
│ │ │ │
│ - React SPA │────>│ - Node.js │
│ - Static Assets │ │ - Express API │
│ - Auto SSL │ │ - Auto Deploy │
└───────────────────┘ └─────────┬─────────┘
│
▼
┌───────────────────┐
│ Neon/Supabase │
│ (PostgreSQL) │
│ │
│ - Managed DB │
│ - Auto Backup │
│ - Monitoring │
└───────────────────┘

┌───────────────────┐ ┌───────────────────┐
│ Cloudinary │ │ SMTP Service │
│ (Images/Files) │ │ (Emails) │
└───────────────────┘ └───────────────────┘

**Environments:**

* Development (local)
* Staging (preview deployments)
* Production (main branch)

CI/CD PIPELINE

**GitHub Actions Workflow:**

On Push to 'develop':

✓ Lint code (ESLint)

✓ Type check (TypeScript)

✓ Run tests (Jest/Vitest)

✓ Build frontend

✓ Build backend

✓ Deploy to Staging

On Push to 'main':

✓ All previous checks

✓ Integration tests

✓ Deploy to Production

✓ Run smoke tests

✓ Notify team

13. TECNOLOGÍAS POR CAPA

┌──────────────────┬────────────────────────────────────────────────┐
│ Capa │ Tecnologías │
├──────────────────┼────────────────────────────────────────────────┤
│ Presentación │ React, TypeScript, Vite, TailwindCSS, │
│ │ shadcn/ui, React Router, Zustand, React Query │
├──────────────────┼────────────────────────────────────────────────┤
│ Aplicación │ Node.js, Express, TypeScript, Prisma, │
│ │ JWT, bcrypt, Zod, Winston, Nodemailer │
├──────────────────┼────────────────────────────────────────────────┤
│ Datos │ PostgreSQL 15+ │
├──────────────────┼────────────────────────────────────────────────┤
│ Servicios │ Cloudinary (imágenes), SMTP (email) │
├──────────────────┼────────────────────────────────────────────────┤
│ DevOps │ GitHub Actions, Vercel, Railway/Render, Neon │
├──────────────────┼────────────────────────────────────────────────┤
│ Testing │ Vitest, React Testing Library, Jest, Supertest│
├──────────────────┼────────────────────────────────────────────────┤
│ Development │ Git, VS Code, Postman, Prisma Studio, Figma │
└──────────────────┴────────────────────────────────────────────────┘

14. CONFORMIDAD CON REQUERIMIENTOS NO FUNCIONALES

┌──────────────┬────────────────────────────────────────────────────┐
│ RNF │ Implementación │
├──────────────┼────────────────────────────────────────────────────┤
│ RNF-01 │ Vercel/Railway uptime 99%+ │
│ RNF-02 │ React Query cache + DB indexes < 3s │
│ RNF-03 │ Tailwind responsive + mobile-first │
│ RNF-04 │ JWT + bcrypt + helmet + CORS │
│ RNF-05 │ Stateless API, horizontal scaling ready │
│ RNF-06 │ Chrome, Firefox, Edge, Safari support │
│ RNF-07 │ Neon/Supabase automated daily backups │
└──────────────┴────────────────────────────────────────────────────┘

VALIDACIÓN Y APROBACIÓN

**Esta arquitectura cumple con:**

✓ Todos los requerimientos funcionales

✓ Todos los requerimientos no funcionales

✓ Mejores prácticas de la industria

✓ Escalabilidad y mantenibilidad

✓ Seguridad robusta

✓ Costo $0

**Elaborado por:**

Joaquín Marín - Gerente de Proyecto

Benjamin Vilches - Arquitecto de Software

Fecha: 13/10/2025

**Aprobado por:**

[ ] Alexis González - Patrocinador

[ ] Fabián Álvarez - Docente

Firma: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Fecha: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_