

MANUAL 2: GUÍA DE DESARROLLO Y API - SISTEMA CONA

INFORMACIÓN DEL DOCUMENTO

Fecha de Creación: 21 de Julio de 2025

Proyecto: Sistema CONA (Gestión CONAVEG)

Audiencia: Desarrolladores Frontend, Backend, Integradores de API

Nivel: Intermedio - Avanzado

Tiempo Estimado: 3-6 horas (estudio completo)

Última Actualización: 21 de Julio de 2025

OBJETIVOS DE APRENDIZAJE

Al finalizar este manual, serás capaz de:

- ☒ Comprender la arquitectura completa del Sistema CONA
 - ☒ Implementar autenticación JWT y manejo de roles
 - ☒ Consumir todas las APIs REST disponibles
 - ☒ Manejar DTOs, validaciones y respuestas de error
 - ☒ Integrar el frontend con el sistema de autorización
 - ☒ Utilizar el modo desarrollo sin autenticación
 - ☒ Probar APIs usando Swagger y herramientas externas
 - ☒ Implementar mejores prácticas de desarrollo
-

REQUISITOS PREVIOS

Conocimientos Necesarios:

- Conceptos de APIs REST y HTTP
- Experiencia con JSON y JavaScript/TypeScript
- Conocimientos de autenticación JWT
- Familiaridad con Spring Boot (para backend)
- Conceptos básicos de bases de datos relacionales

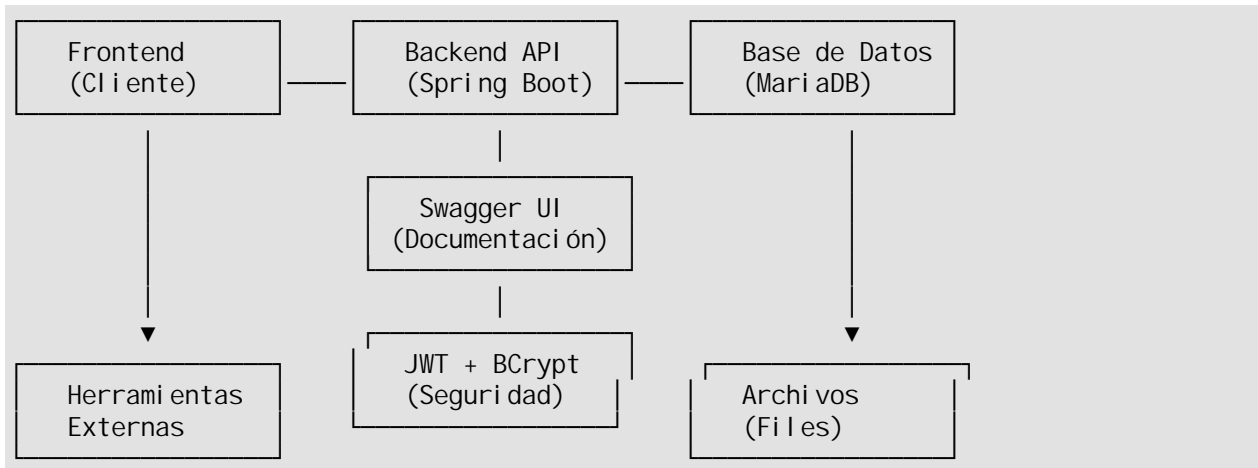
Herramientas Requeridas:

- Sistema CONA instalado y funcionando (ver Manual 1)
- Postman, Insomnia o similar para testing de APIs

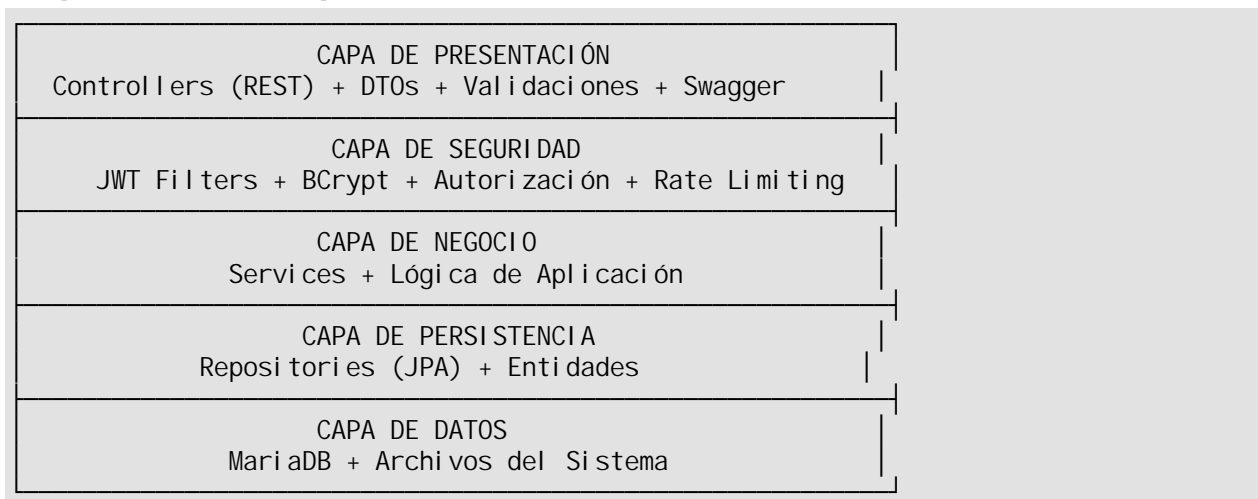
- IDE o editor de código
- Navegador web moderno

ARQUITECTURA DEL SISTEMA

Arquitectura General



Arquitectura de Capas Backend



Estructura del Código

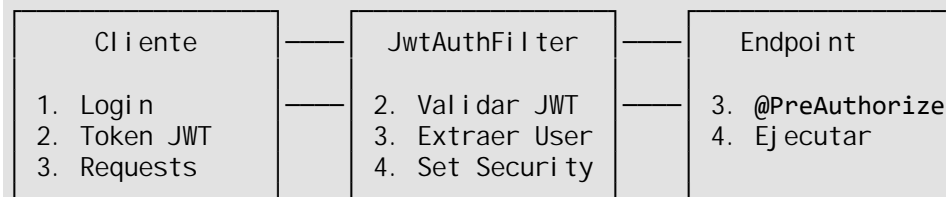
```

src/main/java/com/conaveg/cona/
├── config/           # Configuraciones (Security, Jackson, etc.)
├── controller/       # Endpoints REST
├── dto/              # Objetos de Transferencia de Datos
├── filter/           # Filtros de seguridad y auditoría
├── model/            # Entidades JPA
├── repository/       # Acceso a datos
└── service/          # Lógica de negocio
  
```

```
└─ util / # Utilidades (JWT, validaciones)
└─ ConaApplication.java # Clase principal
```

SISTEMA DE AUTENTICACIÓN Y AUTORIZACIÓN

Arquitectura de Seguridad JWT



Flujo de Autenticación Completo

1. Proceso de Login

```
// 1. Cliente envía credenciales
POST /api/auth/login
Content-Type: application/json
```

```
{
  "email": "usuario@ejemplo.com",
  "password": "Mi Password123!"
}
```

```
// 2. Backend valida con BCrypt y responde
HTTP/1.1 200 OK
```

```
{
  "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2Vybm91dCI6ImFkbGUiLCJpdiI6MX0",
  "tokenType": "Bearer",
  "expiresIn": 86400000,
  "user": {
    "id": 1,
    "userName": "usuario",
    "email": "usuario@ejemplo.com",
    "role": "GERENTE",
    "estado": "ACTIVO"
  }
}
```

2. Uso del Token en Requests Posteriores

```
// Cliente incluye token en header Authorization
GET /api/empleados
Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2Vybm91dCI6ImFkbGUiLCJpdiI6MX0
```

3. Endpoints de Autenticación Disponibles

```
# Login básico
POST /api/auth/login
```

```

# Información del usuario actual
GET /api/auth/me

# Renovar token JWT (dentro de ventana de 15 min antes de expirar)
POST /api/auth/refresh

# Recuperación de contraseña
POST /api/auth/forgot-password
POST /api/auth/reset-password
GET /api/auth/validate-reset-token?token=...

# Logout básico
POST /api/auth/logout

# Validar token
POST /api/auth/validate

```

Sistema de Roles y Permisos

Roles Implementados:

- ADMIN: Acceso total al sistema
- GERENTE: Gestión operativa + perfil propio
- EMPLEADO: Info de empleados + perfil propio
- USER: Solo lectura + perfil propio

Matriz de Permisos Resumida:

Recurso	ADMIN	GERENTE	EMPLEADO	USER
Usuarios	✓ CRUD	✗	✗	✗
Empleados	✓ CRUD	✗	✓ Read	✗
Proyectos	✓ CRUD	✓ CRUD	👁️ Read	👁️ Read
Inventario	✓ CRUD	✓ CRUD	👁️ Read	👁️ Read
Facturas	✓ CRUD	✓ CRUD	✗	✗
Perfil Propio	✓	✓	✓	✓

Nota: Todos los usuarios pueden ver y editar su propio perfil (ownership logic).

MODO DESARROLLO SIN AUTENTICACIÓN

Activación del Modo Desarrollo

Configurar Variable de Entorno:

```
# Activar modo desarrollo
export SPRING_PROFILES_ACTIVE=dev

# Iniciar aplicación
mvn spring-boot:run
```

Verificar Modo Activo:

```
# Endpoint para verificar estado
curl http://localhost:8080/conaveg/api/dev/status

# Respuesta esperada:
{
  "message": "Modo desarrollo activo - Autenticación deshabilitada",
  "warning": "NO USAR EN PRODUCCIÓN",
  "skipAuthentication": true,
  "availableEndpoints": ["/api/dev/login", "/api/dev/me", "/api/dev/status"]
}
```

Endpoints de Desarrollo Disponibles

Login Simulado:

```
// Cualquier credencial funciona
POST /api/dev/login
Content-Type: application/json

{
  "email": "cualquier@email.com",
  "password": "cualquier-password"
}

// Respuesta simulada:
{
  "token": "dev-mock-jwt-token-12345",
  "type": "Bearer",
  "expiresIn": 86400,
  "user": {
    "id": 1,
    "userName": "dev-user",
    "email": "dev@conaveg.com",
    "role": "ADMINISTRADOR",
    "estado": "ACTIVO"
  },
  "message": "Login simulado para desarrollo"
}
```


Usuario Actual Simulado:

```
GET /api /dev/me

// Respuesta:
{
  "id": 1,
  "userName": "dev-user",
  "email": "dev@conaveg.com",
  "role": "ADMINISTRADOR",
  "estado": "ACTIVO",
  "message": "Usuario simulado para desarrollo"
}
```

Ventajas del Modo Desarrollo

- ☒ Acceso directo a todos los endpoints sin token
- ☒ Desarrollo rápido sin implementar autenticación inicialmente
- ☒ Testing simplificado de APIs
- ☒ Datos consistentes para pruebas

 **IMPORTANTE:** Este modo SOLO funciona en desarrollo y se desactiva automáticamente en producción.

GUÍA COMPLETA DE APIs REST

URL Base del Sistema

```
http://localhost:8080/conaveg/api
```

Headers Comunes

```
// Headers básicos para todas las requests
{
  "Content-Type": "application/json",
  "Authorization": "Bearer <JWT_TOKEN>", // Excepto en modo dev
  "Accept": "application/json"
}
```

API DE USUARIOS

Base URL: `/api /users`

1. Listar Usuarios (Solo ADMIN)

```
GET /api /users
```

```
// Respuesta:
[
```

```
{
  "id": 1,
  "userName": "admin",
  "email": "admin@conaveg.com",
  "role": {
    "id": 1,
    "nombre": "Administrador"
  },
  "estado": "ACTIVO",
  "createdAt": "2025-07-21T10:30:00Z"
}
```

2. Obtener Usuario por ID

```
// Solo ADMIN o el propio usuario
GET /api/users/{id}

// Respuesta:
{
  "id": 1,
  "userName": "usuario",
  "email": "usuario@ejemplo.com",
  "role": {
    "id": 2,
    "nombre": "Gerente"
  },
  "estado": "ACTIVO",
  "createdAt": "2025-07-21T10:30:00Z",
  "updatedAt": "2025-07-21T15:45:00Z"
}
```

3. Crear Usuario (Solo ADMIN)

```
POST /api/users
Content-Type: application/json

{
  "userName": "nuevo_usuario",
  "email": "nuevo@ejemplo.com",
  "password": "Mi Password123!",
  "roleid": 2
}
```

// Validaciones de Password:
 // - Mínimo 8 caracteres, máximo 128
 // - Al menos 1 minúscula, 1 mayúscula, 1 número, 1 especial
 // - Caracteres especiales permitidos: @\$!%*?&

```
// Respuesta exitosa (201):
{
  "id": 5,
  "userName": "nuevo_usuario",
  "email": "nuevo@ejemplo.com",
  "role": {
```

```
    "id": 2,
    "nombre": "Gerente"
  },
  "estado": "ACTIVO",
  "createdAt": "2025-07-21T16:20:00Z"
}
```

4. Actualizar Usuario

```
// ADMIN puede actualizar cualquiera, otros solo su perfil
PUT /api/users/{id}
Content-Type: application/json

{
  "userName": "usuario_actualizado",
  "email": "nuevo_email@ejemplo.com",
  "password": "NuevaPassword456!", // OPCIONAL en actualizaciones
  "roleId": 3
}

// Nota: Si no se envía password, se mantiene la actual
```

5. Eliminar Usuario (Solo ADMIN)

```
DELETE /api/users/{id}

// Respuesta exitosa (204): Sin contenido
```



API DE EMPLEADOS

Base URL: `/api/empleados`

1. Listar Empleados (ADMIN y EMPLEADO)

```
GET /api/empleados

// Respuesta:
[
  {
    "id": 1,
    "nombre": "Juan",
    "apellido": "Pérez",
    "email": "juan.perez@empresa.com",
    "telefono": "+51987654321",
    "cargo": "Desarrollador Senior",
    "fechaIngreso": "2024-01-15",
    "estado": "ACTIVO",
    "salario": 5000.00
  }
]
```


2. Crear Empleado (Solo ADMIN)

POST /api/empleados

Content-Type: application/json

```
{
  "nombre": "María",
  "apellido": "García",
  "email": "maria.garcia@empresa.com",
  "telefono": "+51987654322",
  "cargo": "Analista de Sistemas",
  "fechaIngreso": "2025-07-21",
  "salario": 4500.00
}
```

// Campos opcionales: telefono, salario

// email debe ser único

3. Actualizar Empleado (Solo ADMIN)

PUT /api/empleados/{id}

Content-Type: application/json

```
{
  "nombre": "María Fernanda",
  "apellido": "García López",
  "email": "maria.garcia@empresa.com",
  "telefono": "+51987654322",
  "cargo": "Analista Senior",
  "fechaIngreso": "2025-07-21",
  "salario": 5200.00
}
```



API DE PROYECTOS

Base URL: </api/proyectos>

1. Listar Proyectos (Todos los roles autenticados)

GET /api/proyectos

// Respuesta:

```
[
  {
    "id": 1,
    "nombre": "Sistema de Inventarios",
    "descripcion": "Modernización del sistema de inventarios",
    "fechaInicio": "2025-01-01",
    "fechaFinPrevista": "2025-12-31",
    "estado": "EN_PROGRESO",
    "presupuesto": 150000.00,
    "responsable": "Juan Pérez"
  }
]
```

```
}  
]
```

2. Crear Proyecto (ADMIN y GERENTE)

POST /api/proyectos

Content-Type: application/json

```
{  
  "nombre": "Portal Web Corporativo",  
  "descripcion": "Desarrollo de nuevo portal web",  
  "fechalnicio": "2025-08-01",  
  "fechaFinPrevista": "2025-11-30",  
  "presupuesto": 80000.00,  
  "responsable": "María García"  
}
```

// Estados válidos: PLANIFICADO, EN_PROGRESO, PAUSADO, COMPLETADO, CANCELADO



API DE INVENTARIO

Base URL: /api/inventario

1. Listar Items de Inventario (Todos los roles)

GET /api/inventario

// Respuesta:

```
[  
  {  
    "id": 1,  
    "nombre": "Laptop Dell Latitude 7420",  
    "descripcion": "Laptop corporativa con 16GB RAM",  
    "categoria": {  
      "id": 1,  
      "nombre": "Equipos de Cómputo"  
    },  
    "cantidad": 25,  
    "precioUnitario": 3500.00,  
    "ubicacion": "Almacén Principal",  
    "estado": "DISPONIBLE"  
  }  
]
```

2. Crear Item de Inventario (ADMIN y GERENTE)

POST /api/inventario

Content-Type: application/json

```
{  
  "nombre": "Mouse Inalámbrico Logitech",  
  "descripcion": "Mouse óptico inalámbrico",  
  "categoriald": 1,  
  "cantidad": 1,  
  "precioUnitario": 10.00,  
  "ubicacion": "Almacén Principal",  
  "estado": "DISPONIBLE"  
}
```

```
"cantidad": 50,  
"precioUnitario": 45.00,  
"ubicacion": "Almacén Principal"  
}
```

API DE PROVEEDORES

Base URL: `/api/proveedores`

1. Listar Proveedores

GET `/api/proveedores`

// Respuesta:

```
[  
  {  
    "id": 1,  
    "nombre": "Tecnología Avanzada S.A.C.",  
    "ruc": "20123456789",  
    "direccion": "Av. Tecnológica 123, Lima",  
    "telefono": "+5114567890",  
    "email": "ventas@tecavanzada.com",  
    "contacto": "Carlos Mendoza",  
    "estado": "ACTIVO"  
  }  
]
```

2. Crear Proveedor

POST `/api/proveedores`

Content-Type: `application/json`

```
{  
  "nombre": "Suministros Industriales Lima",  
  "ruc": "20987654321",  
  "direccion": "Jr. Industrial 456, Lima",  
  "telefono": "+5119876543",  
  "email": "contacto@suministros.com",  
  "contacto": "Ana Rodríguez"  
}
```

// RUC debe ser único y tener formato válido peruano

API DE FACTURAS

Base URL: `/api/facturas`

1. Listar Facturas (ADMIN y GERENTE)

GET `/api/facturas`

// Respuesta:

```
[
  {
    "id": 1,
    "proveedorId": 1,
    "usuarioId": 2,
    "nroFactura": "F-2025-001",
    "tipoDocumento": "FACTURA",
    "fechaEmision": "2025-07-20",
    "fechaVencimiento": "2025-08-20",
    "montoTotal": 15000, // En centavos (150.00 soles)
    "moneda": "PEN",
    "descripcion": "Compra de equipos de oficina",
    "rutaArchivo": "facturas/FACTURA_20250720_143052_a1b2c3d4.pdf",
    "nombreArchivo": "FACTURA_20250720_143052_a1b2c3d4.pdf",
    "estadoFactura": "PENDIENTE"
  }
]
```

2. Crear Factura con Archivo PDF

// Multipart form-data request

POST `/api/facturas/with-file`

Content-Type: `multipart/form-data`

// Form fields:

file: [archivo PDF]

proveedorId: 1

usuarioId: 2

nroFactura: "F-2025-002"

tipoDocumento: "FACTURA"

fechaEmision: "2025-07-21"

fechaVencimiento: "2025-08-21"

montoTotal: 25000 // En centavos

moneda: "PEN"

descripcion: "Servicios de mantenimiento"

estadoFactura: "PENDIENTE"

// Validaciones:

// - Archivo debe ser PDF

// - Máximo 10MB

// - nroFactura debe ser único

3. Descargar Archivo de Factura

GET `/api/facturas/{id}/download`

```
// Respuesta: Archivo PDF para descarga
Content-Type: application/pdf
Content-Disposition: attachment; filename="FACTURA_20250720_143052_a1b2c3d4.pdf"
```

API DE ASISTENCIAS

Base URL: `/api/asistencias`

1. Listar Asistencias

GET `/api/asistencias`

```
// Respuesta:
[
  {
    "id": 1,
    "empleadoid": 1,
    "fecha": "2025-07-21",
    "horaEntrada": "08:30:00",
    "horaSalida": "17:30:00",
    "horasTrabajadas": 8.5,
    "observaciones": "Día normal de trabajo",
    "estado": "COMPLETO"
  }
]
```

2. Registrar Asistencia

POST `/api/asistencias`
Content-Type: `application/json`

```
{
  "empleadoid": 1,
  "fecha": "2025-07-21",
  "horaEntrada": "08:45:00",
  "horaSalida": "17:15:00",
  "observaciones": "Llegada tardía por tráfico"
}
```

// horasTrabajadas se calcula automáticamente
// Estados: COMPLETO, TARDANZA, FALTA, MEDIO_DIA

API DE ROLES

Base URL: `/api/roles` (Solo ADMIN)

1. Listar Roles

GET `/api/roles`

```
// Respuesta:
[
  {
    "id": 1,
    "nombre": "Administrador",
    "descripcion": "Acceso total al sistema"
  },
  {
    "id": 2,
    "nombre": "Gerente",
    "descripcion": "Gestión operativa"
  }
]
```

2. Crear Rol

```
POST /api/roles
Content-Type: application/json

{
  "nombre": "Supervisor",
  "descripcion": "Supervisión de equipos de trabajo"
}
```

API DE CATEGORÍAS DE INVENTARIO

Base URL: `/api/categorias-inventario`

1. Listar Categorías

```
GET /api/categorias-inventario

// Respuesta:
[
  {
    "id": 1,
    "nombre": "Equipos de Cómputo",
    "descripcion": "Laptops, PCs, servidores"
  },
  {
    "id": 2,
    "nombre": "Mobiliario",
    "descripcion": "Escritorios, sillas, archiveros"
  }
]
```

API DE MOVIMIENTOS DE INVENTARIO

Base URL: `/api/movimientos-inventario`

1. Listar Movimientos

GET `/api/movimientos-inventario`

// Respuesta:

```
[
  {
    "id": 1,
    "inventariold": 1,
    "tipoMovimiento": "ENTRADA",
    "cantidad": 10,
    "fechaMovimiento": "2025-07-21T14:30:00Z",
    "motivo": "Compra de nuevos equipos",
    "usuariold": 2
  }
]
```

2. Crear Movimiento

POST `/api/movimientos-inventario`

Content-Type: `application/json`

```
{
  "inventariold": 1,
  "tipoMovimiento": "SALIDA",
  "cantidad": 3,
  "motivo": "Asignación a proyecto XYZ",
  "usuariold": 2
}
```

// Tipos: ENTRADA, SALIDA, AJUSTE, TRANSFERENCIA

API DE ASIGNACIONES PROYECTO-EMPLEADO

Base URL: `/api/asignaciones-proyectos-empleado`

1. Listar Asignaciones

GET `/api/asignaciones-proyectos-empleado`

// Respuesta:

```
[
  {
    "id": 1,
    "proyectold": 1,
    "empleadold": 1,
    "fechaAsignacion": "2025-07-01",
    "fechaFinalizacion": null,
  }
]
```

```
"rol": "Desarrollador Principal",
"estado": "ACTIVO"
}
]
```

2. Crear Asignación

POST /api/asignaciones-proyectos-empleado
Content-Type: application/json

```
{
  "projectId": 1,
  "empleadoId": 2,
  "fechaAsignacion": "2025-07-21",
  "rol": "Analista de Requisitos"
}
```

// Un empleado puede estar asignado a múltiples proyectos



TRABAJANDO CON DTOs Y VALIDACIONES

Patrón DTO Implementado

El sistema utiliza DTOs (Data Transfer Objects) para separar la capa de presentación de la capa de persistencia:

Request → DTO (con validaciones) → Service → Entity → Repository → Database
Database → Entity → Service → DTO → Response

Ejemplo de DTO con Validaciones

UserCreatedDTO.java:

```
public class UserCreatedDTO {

    @NotBlank(message = "El nombre de usuario es obligatorio")
    @Size(min = 3, max = 50, message = "El nombre debe tener entre 3 y 50 caracteres")
    @Pattern(regexp = "^[a-zA-Z0-9._-]+$")
    private String userName;

    @NotBlank(message = "El email es obligatorio")
    @Email(message = "Formato de email inválido")
    @Size(max = 100)
    private String email;

    @NotBlank(message = "La contraseña es obligatoria")
    @Size(min = 8, max = 128)
    @Pattern(regexp = "^(?=.*[a-z])(?=.*[A-Z])(?=.*\\d)(?=.*[@$!%*?&])[A-Za-z\\d@$!%*?&]{8,}$")
    private String password;

    @NotNull(message = "El rol es obligatorio")
```



```
} private Long rolId;
```

Manejo de Errores de Validación

Error de Validación - Respuesta HTTP 400:

```
POST /api/users
{
  "userName": "ab",    // Muy corto
  "email": "email-invalido", // Formato inválido
  "password": "123",   // No cumple complejidad
  "rolId": null
}

// Respuesta de error:
{
  "timestamp": "2025-07-21T16:30:00Z",
  "status": 400,
  "error": "Bad Request",
  "message": "Validation failed",
  "errors": [
    {
      "field": "userName",
      "message": "El nombre debe tener entre 3 y 50 caracteres"
    },
    {
      "field": "email",
      "message": "Formato de email inválido"
    },
    {
      "field": "password",
      "message": "La contraseña debe contener al menos: 1 minúscula, 1 mayúscula, 1
número y 1 carácter especial"
    },
    {
      "field": "rolId",
      "message": "El rol es obligatorio"
    }
  ]
}
```

DOCUMENTACIÓN SWAGGER/OPENAPI

Acceso a Swagger UI

URL: <http://localhost:8080/conaveg/swagger-ui/index.html>

Características de la Documentación:

- ☒ Endpoints organizados por controlador
- ☒ Esquemas de DTOs con validaciones

- ☒ Ejemplos de requests y responses
- ☒ Códigos de respuesta HTTP documentados
- ☒ Autenticación JWT integrada (botón "Authorize")

Uso del Swagger para Testing:

1. Configurar Autenticación JWT:

```
// 1. Hacer login en /api/auth/login
// 2. Copiar el token de la respuesta
// 3. Click en "Authorize" en Swagger UI
// 4. Ingresar: Bearer <tu-token-aqui>
// 5. Todos los endpoints estarán autenticados
```

2. Testing de Endpoints:

- Try it out: Permite editar parámetros
- Execute: Ejecuta la request
- Response: Muestra código HTTP y body de respuesta
- cURL: Genera comando cURL equivalente

HERRAMIENTAS DE DESARROLLO

Postman Collection para CONA

Configurar Environment en Postman:

```
// Variables de entorno:
{
  "base_url": "http://localhost:8080/conaveg/api",
  "jwt_token": "", // Se actualiza automáticamente tras login
  "user_id": "" // Para testing de ownership
}
```

Pre-request Script para Auto-Login:

```
// Script para login automático si el token expira
pm.test("Auto-login if needed", function () {
  const token = pm.environment.get("jwt_token");

  if (!token) {
    pm.sendRequest({
      url: pm.environment.get("base_url") + "/auth/login",
      method: 'POST',
      header: {'Content-Type': 'application/json'},
      body: {
        mode: 'raw',
        raw: JSON.stringify({
          email: "admin@test.com",
```

```

        password: "admin123"
    })
  }
}, function (err, res) {
  if (res.code === 200) {
    const responseJson = res.json();
    pm.environment.set("jwt_token", responseJson.token);
    pm.environment.set("user_id", responseJson.user.id);
  }
});
}
});

```

Testing con cURL

Scripts de Testing Completo:

```

#!/bin/bash
# test_cona_api.sh

BASE_URL="http://localhost:8080/conaveg/api"

echo "🔍 Testing CONA API..."

# 1. Health Check
echo "1. Health Check:"
curl -s "$BASE_URL/.. /actuator/health" | jq

# 2. Login y obtener token
echo "2. Login:"
TOKEN=$(curl -s -X POST "$BASE_URL/auth/login" \
  -H "Content-Type: application/json" \
  -d '{"email": "admin@test.com", "password": "admin123"}' | \
  jq -r '.token')

echo "Token: $TOKEN"

# 3. Test endpoint protegido
echo "3. Test Usuarios:"
curl -s -H "Authorization: Bearer $TOKEN" "$BASE_URL/users" | jq

# 4. Test CRUD completo
echo "4. Test CRUD Roles:"
ROLE_ID=$(curl -s -X POST "$BASE_URL/roles" \
  -H "Authorization: Bearer $TOKEN" \
  -H "Content-Type: application/json" \
  -d '{"nombre": "Test Role", "descripcion": "Rol de prueba"}' | \
  jq -r '.id')

echo "Rol creado con ID: $ROLE_ID"

# 5. Cleanup
curl -s -X DELETE "$BASE_URL/roles/$ROLE_ID" \
  -H "Authorization: Bearer $TOKEN"

```

```
echo "✅ Testing completado"
```

💡 MEJORES PRÁCTICAS DE DESARROLLO

1. Manejo de Tokens JWT

Frontend/Cliente:

```
// Almacenar token de forma segura
localStorage.setItem('conajwt_token', response.token);

// Interceptor para agregar token automáticamente (Axios)
axios.interceptors.request.use(config => {
  const token = localStorage.getItem('conajwt_token');
  if (token) {
    config.headers.Authorization = `Bearer ${token}`;
  }
  return config;
});

// Manejo de expiración automática
axios.interceptors.response.use(
  response => response,
  error => {
    if (error.response?.status === 401) {
      // Token expirado - redirigir a login
      localStorage.removeItem('conajwt_token');
      window.location.href = '/login';
    }
    return Promise.reject(error);
  }
);
```

2. Manejo de Errores Estandarizado

Códigos de Respuesta HTTP del Sistema:

- 200 OK: Operación exitosa
- 201 Created: Recurso creado exitosamente
- 204 No Content: Eliminación exitosa
- 400 Bad Request: Error de validación
- 401 Unauthorized: Token inválido/expirado
- 403 Forbidden: Sin permisos para la operación
- 404 Not Found: Recurso no encontrado
- 409 Conflict: Duplicación de datos únicos

- 429 Too Many Requests: Rate limiting activado
- 500 Internal Server Error: Error interno del servidor

Estructura de Errores Estándar:

```
// Error de validación (400)
{
  "timestamp": "2025-07-21T16:30:00Z",
  "status": 400,
  "error": "Bad Request",
  "message": "Validation failed",
  "errors": [
    {
      "field": "email",
      "message": "Email ya está en uso"
    }
  ]
}

// Error de autorización (403)
{
  "timestamp": "2025-07-21T16:30:00Z",
  "status": 403,
  "error": "Forbidden",
  "message": "No tienes permisos para acceder a este recurso"
}

// Error de rate limiting (429)
{
  "timestamp": "2025-07-21T16:30:00Z",
  "status": 429,
  "error": "Too Many Requests",
  "message": "Demasiados intentos. Intenta nuevamente en 15 minutos",
  "retryAfter": 900
}
```

3. Paginación y Filtros

Parámetros de Query Estándar:

```
// Paginación
GET /api/empleados?page=0&size=20&sort=nombre,asc

// Filtros
GET /api/facturas?estado=PENDIENTE&fechaDesde=2025-01-01&fechaHasta=2025-12-31

// Búsqueda
GET /api/inventario?search=laptop&categoria=1
```

4. Versionado de API

Estructura de Versiones (Futuro):

```
// v1 (actual)
GET /api/users

// v2 (futura)
GET /api/v2/users

// Header de versión
Accept: application/vnd.cona.v1+json
```



TESTING DE APIS

Casos de Prueba Esenciales

1. Testing de Autenticación:

```
// Test 1: Login exitoso
POST /api/auth/login
{
  "email": "admin@test.com",
  "password": "admin123"
}
// Esperado: 200, token válido

// Test 2: Login fallido
POST /api/auth/login
{
  "email": "admin@test.com",
  "password": "wrong_password"
}
// Esperado: 401, mensaje de error

// Test 3: Acceso sin token
GET /api/users
// Esperado: 401 Unauthorized

// Test 4: Token expirado
GET /api/users
Authorization: Bearer <token_expirado>
// Esperado: 401 Unauthorized
```

2. Testing de Autorización:

```
// Test 1: ADMIN accediendo a usuarios - ☒ DEBE PASAR
GET /api/users
Authorization: Bearer <admin_token>
// Esperado: 200, lista de usuarios

// Test 2: USER accediendo a usuarios - ☐ DEBE FALLAR
GET /api/users
```

```

Authorization: Bearer <user_token>
// Esperado: 403 Forbidden

// Test 3: Usuario accediendo a su propio perfil - ☒ DEBE PASAR
GET /api/users/5
Authorization: Bearer <user_id_5_token>
// Esperado: 200, datos del usuario

// Test 4: Usuario accediendo a perfil ajeno - ☐ DEBE FALLAR
GET /api/users/3
Authorization: Bearer <user_id_5_token>
// Esperado: 403 Forbidden

```

3. Testing de Validaciones:

```

// Test datos inválidos
POST /api/users
{
  "userName": "", // Vacío
  "email": "invalid-email", // Formato inválido
  "password": "123", // Muy simple
  "roleId": 999 // ID inexistente
}
// Esperado: 400, errores de validación detallados

```

4. Testing de Rate Limiting:

```

# Script para probar rate limiting
for i in {1..12}; do
  echo "Intento $i:"
  curl -X POST http://localhost:8080/conaveg/api/auth/login \
    -H "Content-Type: application/json" \
    -d '{"email": "fake@test.com", "password": "wrong"}' \
    -w "HTTP Code: %{http_code}\n"
done

# A partir del intento 11 debería retornar 429

```

CONFIGURACIÓN PARA DIFERENTES ENTORNOS

Desarrollo Frontend con Backend Local

Configuración CORS (ya incluida):

```

// En SecurityConfig.java
@Bean
public CorsConfigurationSource corsConfigurationSource() {
  CorsConfiguration configuration = new CorsConfiguration();
  configuration.setAllowedOrigins(Arrays.asList(
    "http://localhost:3000", // React
    "http://localhost:4200", // Angular
    "http://localhost:8080", // Vue
    "http://127.0.0.1:5500" // Live Server
  ));
}

```

```

));
configuration.setAllowedMethods(Arrays.asList("*"));
configuration.setAllowedHeaders(Arrays.asList("*"));
configuration.setAllowCredentials(true);
return source;
}

```

Proxy para Desarrollo (Create React App):

```

// package.json
{
  "proxy": "http://localhost:8080"
}

// 0 con setupProxy.js
const { createProxyMiddleware } = require('http-proxy-middleware');

module.exports = function(app) {
  app.use(
    '/conaveg',
    createProxyMiddleware({
      target: 'http://localhost:8080',
      changeOrigin: true
    })
  );
};

```

Configuración para Producción

Variables de Entorno Críticas:

```

# Producción
export SPRING_PROFILES_ACTIVE=prod
export JWT_SECRET="clave_jwt_super_segura_de_256_bits_minimo_para_produccion_2025"
export DB_PASSWORD="password_muy_seguro_de_base_de_datos"
export MAIL_PASSWORD="app_password_del_email_corporativo"

```



MONITOREO Y MÉTRICAS

Endpoints de Actuator Disponibles:

```

// Health check
GET /actuator/health

// Métricas de la aplicación
GET /actuator/metrics

// Información de la aplicación
GET /actuator/info

// Configuración (solo en desarrollo)
GET /actuator/configprops

```


Métricas Personalizadas del Sistema:

```
// Métricas de autenticación
GET /actuator/metrics/authentication.success.total
GET /actuator/metrics/authentication.failure.total

// Métricas de rate limiting
GET /actuator/metrics/rate.limit.violations.total

// Métricas de performance
GET /actuator/metrics/http.server.requests
```

✓ CHECKLIST DE INTEGRACIÓN

Frontend con Backend CONA:

Preparación:

- ☐ Backend CONA ejecutándose correctamente
- ☐ Swagger UI accesible y funcional
- ☐ Variables de entorno configuradas
- ☐ CORS configurado para dominio del frontend

Autenticación:

- ☐ Implementar login con email/password
- ☐ Almacenar JWT token de forma segura
- ☐ Interceptor para incluir token en requests
- ☐ Manejo de expiración de token
- ☐ Logout que limpia token almacenado

Manejo de APIs:

- ☐ Cliente HTTP configurado (Axios, Fetch, etc.)
- ☐ Base URL configurada según entorno
- ☐ Headers estándar configurados
- ☐ Manejo de errores estandarizado
- ☐ Loading states para requests

Autorización:

- ☐ Roles de usuario implementados en UI
- ☐ Restricciones de acceso por rol




- [] Manejo de errores 403 Forbidden
- [] Validación de permisos en componentes

Testing:

- [] Tests de integración con API
- [] Manejo de estados de error
- [] Tests de autorización por rol
- [] Tests de flujo de autenticación completo

RECURSOS Y DOCUMENTACIÓN ADICIONAL

Documentación del Proyecto:

-  [Manual de Instalación](#)
-  [Manual de Testing](#)
-  [Manual de Seguridad](#)
-  [Manual de Monitoreo](#)

Recursos Técnicos:



- [Spring Boot Documentation](#)
- [JWT.io](#) - Para decodificar tokens JWT
- [Postman Learning Center](#)
- [REST API Best Practices](#)



Herramientas Recomendadas:

- Postman: Testing de APIs
- Insomnia: Alternativa a Postman
- httpie: Cliente HTTP de línea de comandos
- jq: Procesamiento de JSON en terminal

SOPORTE Y CONTACTO






Canales de Soporte:

-  Email Técnico: desarrollo@conaveg.com
-  Slack: #cona-desarrollo

-  Teams: Canal CONA APIs
-  Issues: GitHub Issues del proyecto

Horarios de Soporte:

- Lunes a Viernes: 9:00 AM - 6:00 PM
- Respuesta promedio: 4 horas laborales
- Emergencias: Contactar por Teams

 Fecha de Creación: 21 de Julio de 2025
  Responsable: Equipo de Desarrollo CONA
 Estado: Manual Completo y Validado
 Próxima Revisión: 21 de Agosto de 2025

NOTA FINAL

Este manual proporciona una guía completa para desarrolladores que deseen integrar con el Sistema CONA. Todas las APIs están documentadas con ejemplos reales y casos de uso típicos.

🎯 **Objetivo:** Facilitar la integración rápida y eficiente con el backend del Sistema CONA, manteniendo las mejores prácticas de seguridad y desarrollo.

¡Happy Coding! 