Resumen Técnico del Swagger – API Zetti / Biotrack (T&SWeb)

Este documento consolida **toda la información relevante del Swagger** de la API Zetti / Biotrack, filtrada y reinterpretada para uso **real en producción**, evitando confusiones habituales del Swagger UI.

---

1. Información General

- **Nombre API:** Zetti / Biotrack / T&SWeb API
- **Tipo:** API REST on-premise
- **Formato:** JSON
- **Swagger / OpenAPI:** v3.x
- **Orientación:** consumo *server-to-server* (NO browser)

---

2. Servidor / Base URL

En el Swagger:

```
servers:
  - url: /api-rest
```

Interpretación correcta

- El Swagger **NO fija host ni protocolo**.
- `/api-rest` es una **ruta relativa**.
- El host real depende del entorno:

Ejemplos válidos:

```
http://190.15.199.103:8089/api-rest
http://192.168.54.200:8080/api-rest
http://192.168.50.200:8080/api-rest
```

👉Todos apuntan al **mismo backend lógico**.

---

3. Seguridad (visión global)

El Swagger declara:

```
security:
  - bearerAuth: []
```

Esto induce a error.

En la práctica existen DOS mecanismos distintos:

---

4. Autenticación Tipo A – OAuth clásico (Bearer)

Características

- Endpoint: `/oauth/token`
- Método: `POST`
- Autenticación: **Basic Auth** (usuario + clave)
- Tipo de token: **Bearer**
- Uso: endpoints generales de la API

Puntos críticos

- **NO existe refresh_token**
- El token **NO se renueva**
- Cuando expira → se vuelve a pedir desde cero

Flujo correcto

1. POST `/oauth/token`
2. Header:

   ```
   Authorization: Basic base64(usuario:clave)
   Content-Type: application/x-www-form-urlencoded
   ```

3. Se obtiene `access_token`
4. Se usa como:

   ```
   Authorization: Bearer <token>
   ```

5. Al expirar → repetir el flujo

👈No intentar `grant_type=refresh_token`

---

5. Autenticación Tipo B – Token V2 / HashedController

Swagger

Sección:

```
Token V2 - (HashedController)
```

Endpoint típico:

```
POST generateHash
```

Interpretación REAL

Esto **NO es OAuth**.

Es un **hash efímero**, normalmente: - dependiente del tiempo - dependiente de credenciales - de vida muy corta

Reglas obligatorias

- El hash **NO se cachea**
- El hash **NO se refresca**
- El hash **NO se reutiliza**

Uso correcto

```
1. generateHash
2. usar hash inmediatamente
3. descartar hash
```

Swagger funciona porque todo ocurre en la misma sesión y sin delay. El backend falla si intenta reutilizar el hash.

---

6. Identificadores de Nodo (MUY IMPORTANTE)

En los paths del Swagger aparecen ejemplos como:

```
/{idNodo}/...
/{idNode}/...
```

Punto crítico

- `idNodo` / `idNode` **NO es necesariamente numérico**
- Puede ser:
- string
- código lógico
- identificador interno

Consecuencia

Si se envía un ID incorrecto: - la API responde **200 OK** - pero devuelve **datos vacíos** - sin error explícito

👉Esto genera falsos diagnósticos.

---

7. Comportamiento HTTP

- La API **NO tiene CORS**
- La API **NO soporta consumo directo desde navegador**
- Está pensada para redes internas

Implicación

- HTTPS → HTTP desde browser = BLOQUEADO
- Uso obligatorio de **backend proxy**

---

8. Arquitectura Recomendada (obligatoria)

```
Frontend HTTPS (Firebase / Web)
   ↓
Backend Proxy (Node / Server)
   ↓
API Zetti / Biotrack (HTTP)
```

- El frontend **NO** conoce:
- IPs internas
- tokens
- lógica de autenticación

---

9. Errores comunes inducidos por el Swagger

- Creer que Token V2 es OAuth
- Intentar refresh_token
- Cachear hashes
- Usar ID de nodo incorrecto
- Llamar la API desde el browser
- Suponer que 200 = datos válidos

---

10. Conclusión operativa

El Swagger **describe endpoints**, pero **NO describe correctamente el modelo de seguridad real**.

Para que el sistema funcione de forma estable:

- OAuth: volver a pedir token al expirar
- Token V2: generar y descartar
- Nodo: validar identificador correcto
- Browser: nunca llamar directo

Este documento debe prevalecer sobre interpretaciones literales del Swagger UI.

Estado: Documento final de referencia técnica **Uso recomendado:** integración, mantenimiento y traspaso a terceros