UNIVERSIDAD NACIONAL DE INGENIERÍA

Manual API de Integración Técnica: EUDR-BlockChain (v1.0)

Blockchain y Validación Geoespacial para la Trazabilidad Agrícola en Cumplimiento Normativo EUDR

PROYECTO: PROCIENCIA – PE501093703-2024

2026

Contenido

1. Introducción

El presente documento establece las directrices, reglas criptográficas y la referencia de *endpoints* necesarios para integrar aplicaciones móviles (Android/iOS) con el **Backend Blockchain EUDR**.

El objetivo de esta API no es solo recibir datos, sino garantizar que la información recolectada en zonas rurales mantenga su integridad criptográfica desde el dispositivo del usuario hasta su inserción en la cadena de bloques.

2. Arquitectura Offline-First y Flujo de Trabajo

La API Móvil está diseñada bajo una arquitectura estrictamente **Offline-First**. Esto significa que la aplicación móvil debe estar preparada para operar en zonas agrícolas sin conexión a internet.

El flujo de trabajo obligatorio para la App Móvil es el siguiente:

1. **Captura Local:** Almacenar todos los datos transaccionales (parcelas, cosechas, acopios) en la base de datos local del dispositivo (ej. SQLite / Room).
2. **Firma Criptográfica Local:** La aplicación móvil es la única encargada de generar los **Hashes criptográficos** en el momento exacto de la operación en campo.
3. **Sincronización Asíncrona (Lotes):** Cuando el dispositivo detecte una conexión a internet estable, enviará los paquetes de datos al servidor. El servidor recalculará los hashes recibidos; si coinciden, el bloque se considerará válido y se guardará en la Blockchain.

3. Autenticación y Seguridad

La API utiliza dos capas de seguridad distintas dependiendo del nivel de criticidad del *endpoint*:

3.1. Uso de API Key (x-api-key)

Utilizada para autorizar a la aplicación móvil en sí misma para consumir recursos base. Es requerida para operaciones de inicialización o donde el usuario aún no tiene una sesión activa (ej. Login, Registro Offline de usuarios, Descarga de Catálogos).

- **Implementación:** Debe enviarse obligatoriamente en las cabeceras (*Headers*) de la petición HTTP.
- **Formato:** x-api-key: <tu_api_key_proporcionada>

3.2. Uso de JWT Token (Authorization: Bearer)

Utilizada para identificar al usuario final que está realizando una transacción. Es requerida para todas las operaciones de escritura en la Blockchain (Sincronización de parcelas, lotes y acopios).

- **Implementación:** Se obtiene tras consumir exitosamente el *endpoint* de Login.
- **Formato:** Debe enviarse en el Header como Authorization: Bearer <token_jwt>.

4. Algoritmo de Hashing Criptográfico (SECCIÓN CRÍTICA)

Para que el servidor acepte y mine los bloques enviados por la aplicación móvil, **el Hash SHA-256 generado en el celular (ya sea en Kotlin, Swift o Flutter) debe ser matemáticamente idéntico al que calculará el servidor**.

Dado que ambos sistemas operan en lenguajes distintos, es **obligatorio** que el móvil sanitice (limpie) el JSON antes de encriptarlo.

4.1. Reglas estrictas de limpieza (Sanitización del Payload)

Antes de aplicar la función de Hash, el diccionario de datos (JSON) debe pasar por el siguiente filtro exacto:

- **Regla 1 (Sin IDs Locales):** Elimine cualquier clave que se llame exactamente id o que termine en los sufijos _id o _ids (ej. parcela_id, producto_id). Los IDs de SQLite del celular no existen en la nube y romperán la Blockchain.
- **Regla 2 (Sin Hashes):** Elimine las claves hash o nft_hash, ya que son el resultado del cálculo y no pueden ser parte del input.

- **Regla 3 (Sin Nulos ni Vacíos):** Elimine cualquier clave cuyo valor sea null o un *string* vacío "".
- **Regla 4 (Normalización de Números):** Si su lenguaje móvil tipa un número entero como flotante añadiendo un decimal cero (ej. 68.0), debe serializarlo estrictamente como entero (68).

4.2. Fórmula Final de Hashing (SHA-256)

Una vez aplicado el filtro anterior, convierta el diccionario a un *String* JSON aplicando la **Regla 5 (Orden y Espacios)**:

- Las llaves del JSON deben estar ordenadas alfabéticamente.
- **No deben existir espacios** después de las comas o los dos puntos (Formato: {"a":1,"b":"texto"}).
- No escape los caracteres Unicode (tildes, letras ñ). Deben enviarse crudos (UTF-8).

Finalmente, aplique la siguiente fórmula para obtener el Hash de 64 caracteres:

Hash = SHA256 (JSON_LIMPIO_STRING_ORDENADO + PREVIOUS_HASH)

(Nota: El valor del PREVIOUS_HASH dependerá del endpoint que esté consumiendo y se especificará en la Referencia de la API).

5. Referencia de Endpoints (API REST)

Todas las peticiones deben enviarse con el encabezado Content-Type: application/json.

5.1. Endpoints de Inicialización

5.1.1. Obtener Catálogos Base (GET /mobile/catalogs)

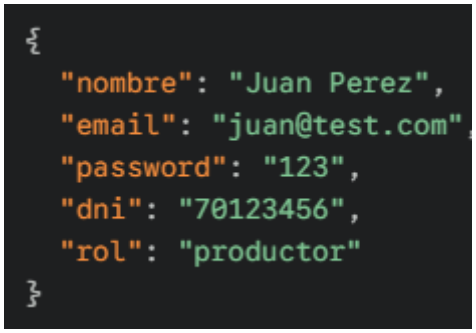
Descarga la lista maestra de productos (ej. Café, Cacao) para poblar la base de datos local (SQLite/Room) del dispositivo móvil.

- **Método y Ruta:** GET /mobile/catalogs
- **Headers requeridos:** x-api-key: <tu_api_key>
- **Respuesta Exitosa (200 OK):**

5.1.2. Registro de Usuario Offline (POST /mobile/sync-user)

Sincroniza un usuario (Productor o Acopiador) que fue creado en el campo sin conexión a internet.

- **Método y Ruta:** POST /mobile/sync-user
- **Headers requeridos:** x-api-key: <tu_api_key>
- **Payload (Cuerpo):**



- **Respuesta Exitosa (201 Created):** Retorna status, server_id y el codigo_generado (ej. PR-0001).

5.1.3. Login Móvil (POST /mobile/login)

Autentica al usuario contra el servidor y devuelve el Token JWT necesario para transacciones Blockchain.

- **Método y Ruta:** POST /mobile/login
- **Headers requeridos:** x-api-key: <tu_api_key>
- **Payload (Cuerpo):**

```
{
  "credential": "juan@test.com",
  "password": "123"
}
```

- **Respuesta Exitosa (200 OK):** Retorna el objeto user y el string token.

5.2. Endpoints Transaccionales (Blockchain)

5.2.1. Sincronización de Parcela / NFT (POST /mobile/sync-parcela)

Registra el polígono geográfico. El servidor evaluará la deforestación y validará el Hash NFT.

- **Método y Ruta:** POST /mobile/sync-parcela
- **Headers requeridos:** Authorization: Bearer <token>
- **Previous Hash para cálculo local:** String vacío "".
- **Payload (Cuerpo):**

```
{
  "nombre_parcela": "Mi Finca",
  "departamento": "San Martín",
  "provincia": "Tocache",
  "distrito": "Zapatero",
  "poligono": [[-6.55, -76.50], [-6.56, -76.51], [-6.54, -76.51]],
  "productos_nombres": ["CACAO"],
  "nft_hash": "hash_calculado_localmente"
}
```

- **Respuesta Exitosa (201 Created):** Retorna deforestacion_status (booleano indicando si es legal) y match (booleano validando si su hash coincide con el del servidor).

5.2.2. Sincronización de Cosechas / Lotes (POST /mobile/sync-lotes)

Sincroniza uno o múltiples bloques génesis de cosecha. **Este endpoint es idempotente**: soporta reintentos seguros si se cae la red.

- **Método y Ruta:** POST /mobile/sync-lotes
- **Headers requeridos:** Authorization: Bearer <token>

- **Previous Hash para cálculo local:** 64 ceros ("0000000000000000000000000000000000000000000000000000000000000000").
- **Payload (Cuerpo):**

- **Respuesta Exitosa (201 Created):** Devuelve un array de detalles. Si el status es "already_synced" o "ok", usted debe marcar ese registro como "subido" en su BD local.

5.2.3. Sincronización de Acopio / Merkle Root (POST /mobile/sync-acopio)

Sincroniza el bloque contenedor que agrupa múltiples tickets escaneados mediante el Árbol de Merkle.

- **Método y Ruta:** POST /mobile/sync-acopio
- **Headers requeridos:** Authorization: Bearer <token>
- **Previous Hash para cálculo local:** 64 ceros ("0000000000000000000000000000000000000000000000000000000000000000").
- **Payload (Cuerpo):**

6. Guía de Depuración de Hashes (Troubleshooting)

6.1. ¿Qué hacer si obtiene "hash_match": false?

Si el servidor responde con un estado HTTP 201 pero el campo "hash_match" es false, significa que los datos se guardaron, pero **la firma criptográfica de su aplicación móvil es distinta a la del servidor**.

6.2. Uso del parámetro debug_string_server

Para solucionar esto inmediatamente, el servidor le enviará una variable de ayuda llamada "debug_string_server" dentro del detalle del error.

1. Ese campo contiene el **String exacto (letra por letra)** que el servidor de Python utilizó antes de pasarlo por la función SHA-256.
2. Imprima en la consola de su entorno móvil (LogCat / Xcode Console) el String que usted generó.
3. Compare ambos textos frente a frente.

6.3. Errores comunes de serialización

Generalmente, la discrepancia entre el texto del móvil y del servidor radica en:

- **Espacios residuales:** Un espacio extra después de una coma o dos puntos generados por su librería JSON.
- **IDs Locales infiltrados:** Olvidó excluir un campo _id (ej. parcela_id) de su objeto antes de convertirlo a texto.

- **Tipos de datos:** Un número con punto decimal (.0) que no fue normalizado a entero antes de hashear.

7. Checklist de Integración Rápida

- Tengo la x-api-key configurada en las constantes de mi App.
- Desarrollé una función cleanPayload() que elimina IDs locales, nulos y la propia llave hash antes de encriptar.
- Mi función convierte campos Float como 68.0 a Int 68.
- Mi serializador JSON ordena las llaves alfabéticamente y no deja espacios en blanco.
- Almaceno el Token JWT y lo envío como Bearer en los endpoints de Sincronización.
- Implementé lógica para marcar como "Sincronizado" un lote si el servidor me devuelve "status": "already_synced".