Blockchain para Licenciamento Ambiental - Análise Estratégica

Data: 27 de Outubro de 2025
Projeto: #licenciamentoambiental
Status: Análise e Planejamento
Responsável deste doc: W.Maldonado

🌳 Por que Blockchain faz sentido aqui?

O licenciamento ambiental envolve:

- ☑ **Múltiplas partes interessadas** (órgãos ambientais, empresas, auditores, sociedade civil)
- ☑ **Necessidade de auditoria** e histórico imutável
- ☑ **Transparência pública** de processos
- ☑ **Conformidade regulatória** com rastreabilidade
- ☑ **Prevenção de fraudes** em documentação
- ☑ **Timestamping** confiável de etapas do processo

📊 Casos de Uso no Sistema

1. Registro de Processos CAR

Blockchain poderia registrar:

- Submissão inicial do CAR
- Mudanças de etapa (DADOS_GERAIS → ANÁLISE → APROVADO)
- Atualizações de situação
- Documentos anexados (hash)
- Pareceres técnicos
- Aprovações/reprovações

2. Histórico Imutável de Imóveis

- Transferências de propriedade
- Alterações cadastrais
- Áreas de preservação/reserva legal
- Incidentes ambientais
- Multas ou sanções

3. Rastreabilidade de Documentos

- Hash de PDFs, plantas, laudos técnicos
- Verificação de autenticidade
- Prova de existência em determinado momento

🏗️ Arquiteturas Recomendadas

Opção 1: Blockchain Público (Ethereum, Polygon)

Prós:

- Máxima transparência
- Descentralização total
- Comunidade e ferramentas maduras
- Timestamping público confiável

Contras:

- Custos de transação (gas fees)
- Dados públicos (privacidade limitada)
- Performance variável

Tecnologias:

- **Smart Contracts**: Solidity
- **Redes**: Polygon (custo baixo), Ethereum L2s (Arbitrum, Optimism)
- **Integração**: Web3.py, Ethers.js

Opção 2: Blockchain Privado/Permissionado (Hyperledger Fabric)

Prós:

- Controle de acesso granular
- Performance superior
- Sem custos de gas
- Conformidade com LGPD (dados sensíveis off-chain)

Contras:

- Requer infraestrutura própria
- Menos descentralizado
- Curva de aprendizado

Tecnologias:

- **Hyperledger Fabric** (mais usado em governos/empresas)
- **Quorum** (fork privado do Ethereum)
- **Corda** (focado em contratos legais)

Opção 3: Blockchain-as-a-Service (BaaS)

Prós:

- Setup rápido
- Manutenção gerenciada

- Conformidade built-in

Contras:

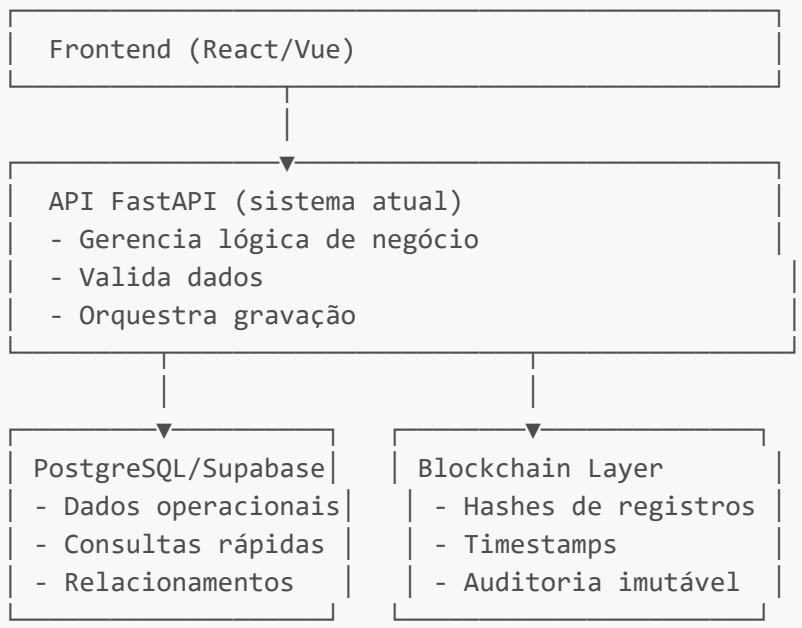
- Vendor lock-in
- Custos recorrentes

Provedores:

- **Amazon Managed Blockchain** (Hyperledger Fabric)
- **Azure Blockchain Service** (descontinuado, mas há alternativas)
- **IBM Blockchain Platform**
- **Oracle Blockchain**

🎯 Recomendação Arquitetural para Licenciamento Ambiental

Modelo Híbrido Recomendado:



Estratégia "Hash Anchoring":

1. **Dados completos** no PostgreSQL (performance)
2. **Hash criptográfico** no blockchain (prova de integridade)
3. **Eventos críticos** registrados on-chain:
    - Criação de CAR
    - Mudança de status críticos
    - Aprovações finais
    - Emissão de licenças

🔐 Dados que Devem Ir para Blockchain

ON-CHAIN (blockchain):

```
{
  "recordType": "CAR_SUBMISSION",
  "recordId": "CAR-2025-001",
  "timestamp": 1730001234,
  "dataHash": "0x3f7a9c2b...",
  "previousHash": "0x2e8b1a...",
  "actor": "CPF:12345678900",
  "action": "SUBMIT",
  "metadata": {
    "imovelId": 123,
    "areaTotal": 150.5
  }
}
```

Nota: Hash SHA-256 dos dados para garantir integridade.

OFF-CHAIN (PostgreSQL):

```
{
  "pkcar": 1,
  "numerocar": "CAR-2025-001",
  "pessoacadastrantenome": "João Silva",
  "pessoacadastrantedocumento": "***45678900",
  "documentosAnexos": [...],
  "blockchainTxHash": "0xabc123..."
}
```

Nota: Dados pessoais ficam off-chain (conformidade LGPD).

BR Conformidade e LGPD

Desafios:

- Blockchain é **imutável** → conflito com "direito ao esquecimento" (LGPD Art. 18)
- Dados pessoais **não podem** estar on-chain em texto claro

Soluções:

1. **Apenas hashes** on-chain (não são dados pessoais)
2. **Dados pessoais** off-chain (podem ser deletados)
3. **Criptografia homomórfica** (pesquisa avançada)
4. **Contratos inteligentes** que referenciam dados externos

Exemplo Conformidade:

```
# OFF-CHAIN: dados completos
pessoa = {
    "cpf": "12345678900",
    "nome": "João Silva"
}

# ON-CHAIN: hash + metadados não-pessoais
blockchain_record = {
    "dataHash": hashlib.sha256(json.dumps(pessoa).encode()).hexdigest(),
    "recordType": "PESSOA_FISICA",
    "timestamp": int(time.time()),
    "area": "LICENCIAMENTO_AMBIENTAL"
}
```

Stack Tecnológica Sugerida

Para Proof of Concept (PoC):

Opção 1: Polygon (baixo custo)

```
from web3 import Web3

w3 = Web3(Web3.HTTPProvider('https://polygon-rpc.com'))

# Smart Contract simples para registro
contract_abi = [...]
contract = w3.eth.contract(address=CONTRACT_ADDRESS, abi=contract_abi)

# Registrar hash
tx_hash = contract.functions.registerRecord(
    record_hash=data_hash,
    record_type="CAR_SUBMISSION",
    metadata=json.dumps(metadata)
).transact({'from': account})
```

Opção 2: Hyperledger Fabric (enterprise)

```
# SDK Python para Fabric
from hfc.fabric import Client

client = Client(net_profile="network.json")
user = client.get_user('org1', 'Admin')

# Invocar chaincode
response = client.chaincode_invoke(
    requestor=user,
```

```
    channel_name='licenciamento-channel',
    peers=['peer0.org1'],
    fcn='registerCAR',
    args=[car_id, data_hash, timestamp]
)
```

📈 Roadmap de Implementação Sugerido

Fase 1: Fundação (1-2 meses)

- [ ] Escolher blockchain (recomendo: **Polygon** para PoC ou **Hyperledger Fabric** para produção)
- [ ] Desenhar smart contract de auditoria
- [ ] Implementar geração de hashes no FastAPI
- [ ] Criar tabela `blockchain_records` no PostgreSQL

Fase 2: Integração (2-3 meses)

- [ ] Endpoint `/blockchain/register` no FastAPI
- [ ] Webhook para eventos automáticos (criação CAR, mudança status)
- [ ] Dashboard de auditoria
- [ ] Verificação de integridade

Fase 3: Produção (3-6 meses)

- [ ] Testes de carga
- [ ] Conformidade legal
- [ ] Portal público de consulta
- [ ] Integração com órgãos governamentais

🤔 Perguntas para Definir a Estratégia

1. **Transparência**: Os registros devem ser **públicos** ou apenas para auditores autorizados?
2. **Performance**: Quantos registros/dia esperamos? (isso afeta escolha público vs privado)
3. **Budget**: Há orçamento para infraestrutura blockchain própria?
4. **Stakeholders**: Quem mais participaria da rede (IBAMA, SEMA, ICMBio)?
5. **Prazo**: Quando isso deve estar em produção?

🎯 Recomendação Final

Para licenciamento ambiental governamental:

Short-term (6 meses):

- **Polygon** (Ethereum L2) para PoC
- Registrar apenas **hashes** e **eventos críticos**
- Manter dados completos no PostgreSQL

- Criar API de verificação pública

Long-term (1-2 anos):

- Migrar para **Hyperledger Fabric** se houver consórcio de órgãos ambientais
- Implementar rede permissionada multi-org
- Portal de transparência para cidadãos
- Integração com Receita Federal (validação CNPJ/CPF)

Recursos Adicionais

Documentação Técnica:

- Web3.py Documentation
- Hyperledger Fabric Documentation
- Polygon Developer Docs

Exemplos de Uso Governamental:

- **Blockchain.gov.br** - Iniciativas blockchain no governo brasileiro
- **Ethereum para registro de diplomas** - Casos de uso educacionais
- **Cartórios blockchain** - Registro de imóveis

Próximos Passos

1. **Validar** requisitos com stakeholders
2. **Definir** qual blockchain usar (público vs privado)
3. **Prototipar** smart contract básico
4. **Integrar** com API FastAPI atual
5. **Testar** em ambiente de desenvolvimento
6. **Escalar** para produção

Notas de Implementação

Tabela PostgreSQL para Tracking:

```
CREATE TABLE blockchain_records (
    id BIGSERIAL PRIMARY KEY,
    record_type VARCHAR(50) NOT NULL,
    record_id VARCHAR(100) NOT NULL,
    data_hash VARCHAR(66) NOT NULL, -- SHA-256 hash
    blockchain_tx_hash VARCHAR(66), -- Transaction hash
    blockchain_network VARCHAR(50), -- 'polygon', 'fabric', etc.
    timestamp TIMESTAMP DEFAULT NOW(),
    actor_id BIGINT,
    action VARCHAR(50),
    metadata JSONB,
```

```
    CONSTRAINT fk_actor FOREIGN KEY (actor_id) REFERENCES x_usr(pk_x_usr)
);

CREATE INDEX idx_blockchain_record_type ON blockchain_records(record_type);
CREATE INDEX idx_blockchain_record_id ON blockchain_records(record_id);
CREATE INDEX idx_blockchain_tx_hash ON blockchain_records(blockchain_tx_hash);
```

Função Helper para Gerar Hash:

```
import hashlib
import json

def generate_record_hash(data: dict) -> str:
    """
    Gera hash SHA-256 de um registro para blockchain.

    Args:
        data: Dicionário com dados do registro

    Returns:
        Hash hexadecimal (prefixado com 0x para compatibilidade Ethereum)
    """
    # Serializa de forma determinística
    json_str = json.dumps(data, sort_keys=True, ensure_ascii=False)

    # Gera hash
    hash_bytes = hashlib.sha256(json_str.encode('utf-8')).digest()

    # Retorna em formato hex com prefixo 0x
    return '0x' + hash_bytes.hex()
```

Exemplo de Endpoint FastAPI:

```
@app.post("/blockchain/register", tags=["blockchain"], summary="Registrar evento
em blockchain")
async def register_blockchain_event(
    record_type: str,
    record_id: str,
    data: dict,
    user_id: int
):
    """
    Registra um evento importante no blockchain para auditoria.

    Args:
        record_type: Tipo de registro (CAR_SUBMISSION, STATUS_CHANGE, etc.)
        record_id: ID do registro (ex: CAR-2025-001)
        data: Dados completos do registro
        user_id: ID do usuário que realizou a ação
```

```
    """
    try:
        # 1. Gera hash dos dados
        data_hash = generate_record_hash(data)

        # 2. Registra no blockchain (exemplo Polygon)
        tx_hash = await blockchain_service.register_record(
            record_type=record_type,
            record_id=record_id,
            data_hash=data_hash,
            actor=user_id
        )

        # 3. Salva referência no PostgreSQL
        with pool.connection() as conn:
            cur = conn.cursor()
            cur.execute("""
                INSERT INTO blockchain_records
                (record_type, record_id, data_hash, blockchain_tx_hash,
                 blockchain_network, actor_id, action, metadata)
                VALUES (%s, %s, %s, %s, %s, %s, %s, %s)
            """, (
                record_type,
                record_id,
                data_hash,
                tx_hash,
                'polygon',
                user_id,
                'REGISTER',
                json.dumps({"timestamp": int(time.time())})
            ))
            cur.close()

        return {
            "success": True,
            "data_hash": data_hash,
            "tx_hash": tx_hash,
            "network": "polygon"
        }

    except Exception as e:
        logger.exception("Erro ao registrar no blockchain")
        raise HTTPException(status_code=500, detail=str(e))
```

🔍 Verificação de Integridade

Endpoint para Verificar Hash:

```
@app.get("/blockchain/verify/{record_id}", tags=["blockchain"])
async def verify_blockchain_record(record_id: str):
```

```
    """
    Verifica se um registro foi alterado comparando hash atual com blockchain.
    """
    try:
        # 1. Busca registro atual no PostgreSQL
        with pool.connection() as conn:
            cur = conn.cursor()
            cur.execute("SELECT * FROM f_car WHERE numerocar = %s", (record_id,))
            current_data = dict(zip([desc[0] for desc in cur.description],
cur.fetchone()))
            cur.close()

        # 2. Gera hash dos dados atuais
        current_hash = generate_record_hash(current_data)

        # 3. Busca hash no blockchain
        blockchain_hash = await blockchain_service.get_record_hash(record_id)

        # 4. Compara
        is_valid = current_hash == blockchain_hash

        return {
            "record_id": record_id,
            "is_valid": is_valid,
            "current_hash": current_hash,
            "blockchain_hash": blockchain_hash,
            "message": "Registro íntegro" if is_valid else "ALERTA: Registro foi
modificado!"
        }

    except Exception as e:
        logger.exception("Erro ao verificar integridade")
        raise HTTPException(status_code=500, detail=str(e))
```

Documento criado em: 27/10/2025
Autor: Análise técnica para projeto #licenciamentoambiental
Versão: 1.0