# BlockReceipt.ai – Comprehensive Integration & Implementation Blueprint

## 1. Introduction

This document serves as an exhaustive blueprint for BlockReceipt.ai, summarizing the current prototype state, audited gaps, and detailed code implementations required to bring the application to a production-ready MVP. It consolidates all previous guidance, audit findings, and code snippets into a single, cohesive reference.

## 2. Current Architecture Snapshot

The current system comprises multiple layers:  
- Authentication & Wallet Management  
- Receipt Upload & OCR Pipeline  
- Proxy Re-Encryption (Threshold TACo)  
- NFT Pool & Selection Service  
- Blockchain Minting Service (Polygon)  
- Frontend Components (React + Tailwind)

Each layer has been implemented to varying degrees of completeness. The following sections detail the status of each layer and outline required next steps.

## 3. Detailed Audit Findings

### 3.1 Authentication & Wallet Management

- Email/password signup with optional wallet generation is implemented, but UX flows are fragmented.  
- TACo encryption service in place for user private key, but no frontend prompt to backup keys.  
- Wallet enforcement on upload needs strengthening.

### 3.2 Receipt Upload & OCR Pipeline

- ReceiptUpload.ts handles multer file upload.  
- ExtractReceiptData returns merchant, date, total, items[].  
- Tier calculation works; encryption triggers.

### 3.3 Proxy Re-Encryption (TACo)

- encryptLineItems() module active.  
- MetadataService stores capsule and ciphertext.  
- Decryption routes exist but need integration with frontend unlock button.

### 3.4 NFT Pool & Selection Service

- nft\_pool.json or DB table not present; previous guidance created mock JSON.  
- NFTPoolService code to retrieve 5 random NFTs per tier should be implemented.

### 3.5 Blockchain Minting Service

- Polygon RPC provider and dev wallet configured.  
- Mint endpoint stub present but calls are mocks; needs real contract integration.

### 3.6 Frontend Components

- useWalletConnect hook built for MetaMask & WalletConnect.  
- ReceiptUpload, NFTSelection, NFTGallery, WalletButton exist.  
- Polling logic and UI messaging need full wiring.

## 4. Smart Contract Deployment

Deploy the ERC-1155 contract on Polygon network:

```solidity  
// contracts/BlockReceiptCollection.sol  
// SPDX-License-Identifier: MIT  
pragma solidity ^0.8.24;  
  
import "@openzeppelin/contracts/token/ERC1155/ERC1155.sol";  
import "@openzeppelin/contracts/access/Ownable.sol";  
  
contract BlockReceiptCollection is ERC1155, Ownable {  
 constructor(string memory baseURI) ERC1155(baseURI) {}  
 function mint(address to, uint256 id) external onlyOwner {  
 \_mint(to, id, 1, "");  
 }  
}  
```

Deployment script (Hardhat):

```js  
// scripts/deploy.js  
async function main() {  
 const [deployer] = await ethers.getSigners();  
 const baseURI = "ipfs://<BASE\_CID>/";  
 const Factory = await ethers.getContractFactory("BlockReceiptCollection");  
 const contract = await Factory.deploy(baseURI);  
 await contract.deployed();  
 console.log("Contract deployed at:", contract.address);  
}  
main();  
```

## 5. Backend Implementation

### 5.1 Enforce Wallet Connection

In `routes/uploadReceipt.ts` at top of endpoint:

```ts  
if (!req.body.walletAddress) {  
 return res.status(400).json({ success:false, message:'Connect your wallet first.' });  
}  
```

### 5.2 Mint Endpoint Wiring

In `routes/nfts.ts`, implement real mint:

```ts  
import ERC1155\_ABI from '../abi/BlockReceiptCollection.json';  
import { ethers } from 'ethers';  
  
router.post('/mint', async (req, res) => {  
 const { walletAddress, nftId } = req.body;  
 if (!walletAddress) {  
 return res.status(400).json({ success:false, msg:'Wallet required' });  
 }  
 const provider = new ethers.providers.JsonRpcProvider(process.env.POLYGON\_RPC\_URL);  
 const devWallet = new ethers.Wallet(process.env.PRIVATE\_KEY, provider);  
 const contract = new ethers.Contract(process.env.CONTRACT\_ADDRESS, ERC1155\_ABI, devWallet);  
 const tx = await contract.mint(walletAddress, nftId);  
 const receipt = await tx.wait();  
 return res.json({ success:true, tokenId: nftId, txHash: receipt.transactionHash });  
});  
```

## 6. NFT Pool Service & Database

Create `data/nft\_pool.json` or DB collection with schema:

```json  
[  
 {  
 "id": "1",  
 "name": "Basic Badge",  
 "tier": "basic",  
 "image": "ipfs://<CID>/1.png",  
 "metadataUri": "ipfs://<CID>/1.json"  
 },  
 ...  
]  
```

Implement service `NFTPoolService.ts`:

```ts  
import db from '../db';  
  
export async function getNFTOptionsForTier(tier: string) {  
 const pool = await db.nftPool.find({ tier }).toArray();  
 return shuffle(pool).slice(0, 5);  
}  
```

## 7. Frontend Integration

### 7.1 WalletConnect Hook

```js  
// client/hooks/useWalletConnect.js  
import { useState } from 'react';  
import { ethers } from 'ethers';  
import WalletConnectProvider from '@walletconnect/web3-provider';  
  
export function useWalletConnect() {  
 const [provider, setProvider] = useState(null);  
 const [signer, setSigner] = useState(null);  
 const [walletAddress, setWalletAddress] = useState(null);  
  
 async function connectMetaMask() {  
 if (window.ethereum) {  
 const inst = new ethers.providers.Web3Provider(window.ethereum);  
 await inst.send('eth\_requestAccounts', []);  
 const sign = inst.getSigner();  
 setProvider(inst);  
 setSigner(sign);  
 setWalletAddress(await sign.getAddress());  
 }  
 }  
 async function connectWalletConnect() {  
 const wc = new WalletConnectProvider({ rpc: { 137: process.env.NEXT\_PUBLIC\_POLYGON\_RPC } });  
 await wc.enable();  
 const inst = new ethers.providers.Web3Provider(wc);  
 const sign = inst.getSigner();  
 setProvider(inst);  
 setSigner(sign);  
 setWalletAddress(await sign.getAddress());  
 }  
 return { connectMetaMask, connectWalletConnect, provider, signer, walletAddress };  
}  
```

### 7.2 Receipt Upload Component

```jsx  
// client/components/ReceiptUpload.jsx  
import { useState } from 'react';  
  
export default function ReceiptUpload({ walletAddress }) {  
 const [status, setStatus] = useState('');  
 const [taskId, setTaskId] = useState(null);  
  
 const handleChange = async e => {  
 const file = e.target.files[0];  
 setStatus('Uploading...');  
 const data = new FormData();  
 data.append('receipt', file);  
 data.append('walletAddress', walletAddress);  
 const res = await fetch('/api/upload-receipt', { method:'POST', body:data });  
 const json = await res.json();  
 if (json.success) {  
 setStatus('Processing...');  
 setTaskId(json.data.taskId);  
 }  
 };  
 return (  
 <div>  
 <input type="file" onChange={handleChange} />  
 <p>{status}</p>  
 </div>  
 );  
}  
```

### 7.3 NFT Selection Component

```jsx  
// client/components/NFTSelection.jsx  
import { useState, useEffect } from 'react';  
  
export default function NFTSelection({ tier, onSelect }) {  
 const [options, setOptions] = useState([]);  
 useEffect(() => {  
 fetch(`/api/nfts/pool?receiptTier=${tier}`)  
 .then(r => r.json())  
 .then(setOptions);  
 }, [tier]);  
 return (  
 <div className="grid grid-cols-2 gap-4">  
 {options.map(nft=>(  
 <div key={nft.id} onClick={()=>onSelect(nft)} className="p-2 border rounded cursor-pointer">  
 <img src={nft.image} alt={nft.name}/>  
 <p>{nft.name}</p>  
 </div>  
 ))}  
 </div>  
 );  
}  
```

## 8. Gallery & Task Polling

```jsx  
// client/hooks/useGalleryPoll.js  
import { useState, useEffect } from 'react';  
  
export function useGalleryPoll(taskId, refresh) {  
 const [ready, setReady] = useState(false);  
 useEffect(()=>{  
 if(!taskId) return;  
 const interval = setInterval(async()=>{  
 const res = await fetch(`/api/task/${taskId}/status`);  
 const { data } = await res.json();  
 if(data.status==='completed'){  
 setReady(true);  
 clearInterval(interval);  
 refresh();  
 }  
 }, 5000);  
 return ()=>clearInterval(interval);  
 }, [taskId, refresh]);  
 return ready;  
}  
```

Use in upload page to auto-refresh gallery once wallet mints.

## 9. Pinning Metadata to IPFS

Use NFT.Storage or Pinata to pin each metadata JSON + image. Update `data/nft\_pool.json` URIs accordingly.  
Example JSON:

```json  
{  
 "id": "1",  
 "tier": "basic",  
 "name": "Basic Badge",  
 "image": "https://gateway.pinata.cloud/ipfs/<CID>/1.png",  
 "metadataUri": "ipfs://<CID>/1.json"  
}  
```

## 10. Final Checklist Before Go-Live

- [ ] Deploy ERC-1155 contract and set CONTRACT\_ADDRESS  
- [ ] Wire backend mint endpoint and test on Amoy  
- [ ] Enforce walletAddress in upload endpoint  
- [ ] Create/pin NFT pool JSON to IPFS  
- [ ] Integrate NFTSelection & ReceiptUpload flows  
- [ ] Implement polling & auto-refresh gallery  
- [ ] Add UI messaging and success toasts  
- [ ] QA end-to-end on Polygon testnet

© BlockReceipt.ai – comprehensive blueprint for MVP completion