# 🔒 BlockReceipt.ai – Comprehensive Code Audit & Improvement Plan

## 1. What’s Working Well

- \*\*Modular Services:\*\* OCR, TPRE encryption, IPFS pinning, metadata storage, and minting are separated into services.  
- \*\*End-to-End Flow Defined:\*\* `upload-and-mint` route covers upload → OCR → encrypt → pin → mint → store.  
- \*\*TPRE Integration:\*\* Threshold client SDK correctly encrypts payloads.  
- \*\*ERC-1155 Contract & Hardhat Suite:\*\* Solid contract with `onlyOwner` mint gating and deploy/verify scripts.

## 2. Critical Gaps & Robustness Risks

| Area | Issue | Impact |  
|------------------------------|---------------------------------------------------------------------------------------------|----------------------------------------------|  
| OCR Accuracy | Regex-only parsing breaks on varied receipt layouts | Garbage metadata → broken UX |  
| IPFS Pinning | Mock pinning lacks retries, timeouts, CID validation | Dead URIs → unusable NFTs |  
| Mint Function Signature | Current mint does not accept metadata URI | NFTs lack metadata link for wallets/markets |  
| TokenID Management | Using `Date.now()` risks ID collisions | Duplicate token IDs |  
| On-chain Metadata Recording | Encrypted payload only off-chain, no on-chain immutability guarantee | Trust vulnerability if backend is compromised|  
| Events & Visibility | Missing `ReceiptMinted` or `EncryptedData` events | Hard to track on-chain activity |  
| Error Handling & Retries | Single catch-all, no granular error feedback | Poor UX and debugging |  
| Frontend UX Guards | Upload enabled without wallet, no stage-by-stage feedback | User confusion & failed flows |

## 3. Recommended Improvements

1. \*\*Upgrade OCR:\*\* Integrate Google Cloud Vision or a specialized receipt-parser library with unit tests for sample images.  
2. \*\*Enhance Mint Function:\*\* Modify ERC-1155 to accept a `string uri` and emit `ReceiptMinted(address to, uint256 tokenId, string uri)`.  
3. \*\*TokenID Sequencing:\*\* Implement on-chain counter or backend sequence to avoid `Date.now()` collisions.  
4. \*\*On-chain Encrypted Data Events:\*\* Emit an `EncryptedData(uint256 tokenId, string capsule, string ciphertext)` event in the contract.  
5. \*\*Robust IPFS Pinning:\*\* Add retry logic and CID validation in `ipfsService`.  
6. \*\*Granular Error Handling:\*\* Separate try/catch per step; return user-friendly errors (e.g., ‘Receipt unreadable’).  
7. \*\*Frontend Feedback & Guards:\*\* Disable upload without wallet, show toasts for each stage, handle promise rejections.

## 4. Next Steps & Roadmap to Production

1. Revise and deploy ERC-1155 contract with URI support and events.  
2. Refactor `upload-and-mint` route into discrete steps with individual error handlers.  
3. Integrate Google Cloud Vision and add receipt-parser tests.  
4. Add event logging for encrypted data on-chain.  
5. Enhance IPFS pinning service with retries and CID checks.  
6. Polish frontend: enforce wallet connect, real-time toasts, auto-refresh gallery.  
7. Conduct end-to-end QA on Polygon testnet, including metadata, transactions, and decryption flow.

By addressing these gaps and following the roadmap, BlockReceipt.ai will become a resilient, user-friendly, and production-ready platform for secure, encrypted receipt-to-NFT experiences.