# 🛠 BlockReceipt.ai – Fixing Receipt Upload + NFT Minting Flow

## 📣 Overview

This document addresses the broken user experience following receipt upload. While Replit implemented backend parsing, encryption, and task creation, the user sees no visible result. This plan removes ambiguity, automates the NFT lifecycle, and delivers front-end and back-end fixes with complete working code examples.

## 🔧 Backend Fix – Update `uploadReceipt.ts`

Store encrypted metadata and return the full NFT lifecycle flow from upload → encrypt → mint → store:

import { storeEncryptedMetadata } from '../services/metadataService';  
import { createNFTPurchaseTask } from '../services/nftPurchaseBot';  
  
...  
  
// After encryption  
const encryptedMetadataInfo = encryptedData ? {  
 capsule: encryptedData.capsule,  
 ciphertext: encryptedData.ciphertext,  
 policyPublicKey: encryptedData.policyPublicKey  
} : undefined;  
  
// Mint NFT  
const purchaseTask = await createNFTPurchaseTask(  
 walletAddress,  
 receiptId,  
 receiptData,  
 encryptedMetadataInfo  
);  
  
// Store metadata (if minting succeeded and returned a tokenId)  
if (purchaseTask.tokenId && encryptedMetadataInfo) {  
 await storeEncryptedMetadata({  
 tokenId: purchaseTask.tokenId,  
 capsule: encryptedMetadataInfo.capsule,  
 ciphertext: encryptedMetadataInfo.ciphertext,  
 policyPublicKey: encryptedMetadataInfo.policyPublicKey,  
 wallet: walletAddress  
 });  
}

## 🖼 Frontend Fix – React Upload Component

Auto-upload file and show a success state when backend confirms receipt:

import { useState } from 'react';  
  
export default function ReceiptUpload({ wallet }) {  
 const [status, setStatus] = useState('');  
 const [nftResult, setNftResult] = useState(null);  
  
 const handleFileChange = async (e) => {  
 const file = e.target.files[0];  
 if (!file || !wallet) return;  
  
 setStatus('Uploading and processing...');  
  
 const formData = new FormData();  
 formData.append('receipt', file);  
 formData.append('wallet', wallet);  
  
 const res = await fetch('/api/upload-receipt', {  
 method: 'POST',  
 body: formData  
 });  
  
 const result = await res.json();  
  
 if (result.success) {  
 setStatus('✅ Receipt uploaded. Your NFT is being minted.');  
 setNftResult(result.data);  
 } else {  
 setStatus('❌ Failed to process receipt.');  
 }  
 };  
  
 return (  
 <div>  
 <input type="file" onChange={handleFileChange} />  
 <p>{status}</p>  
 {nftResult && (  
 <div className="p-4 border mt-4">  
 <p><strong>Tier:</strong> {nftResult.tier?.title}</p>  
 <p><strong>Encrypted:</strong> {nftResult.isEncrypted ? 'Yes' : 'No'}</p>  
 <p><strong>Task:</strong> {nftResult.nftGift?.message}</p>  
 </div>  
 )}  
 </div>  
 );  
}

## 🎉 Optional Success Message Modal

After file upload and response, show user something like:

<div className="bg-green-100 p-4 rounded">  
 <h3>🎉 Receipt received!</h3>  
 <p>Your NFT is being minted. You’ll see it in your gallery soon.</p>  
</div>

## 🔁 Optional – Task Polling (If Needed)

Add a route like `/task/:taskId/status` to fetch tokenId once mint is complete:

router.get('/task/:taskId/status', async (req, res) => {  
 const { taskId } = req.params;  
 const status = await taskService.getTaskStatus(taskId); // e.g. { status: 'completed', tokenId: 1234 }  
 res.json({ success: true, data: status });  
});

## ✅ Summary Instruction to Replit

Please:  
1. Ensure receipt upload auto-triggers the full backend process (OCR, categorize, encrypt, mint)  
2. Store metadata persistently via metadataService  
3. Expose tokenId in upload response (or poll via taskId)  
4. Show user a confirmation message immediately post-upload  
5. Auto-refresh gallery if possible