# 🔄 BlockReceipt.ai – Audit & Fix: Wallet Integration & NFT Gallery

## 1. Overview of Issues

After reviewing \*\*BlockReceiptai21\*\*, we found critical UX/blocking issues:  
1. \*\*Wallet integration fallback\*\*: Without MetaMask or WalletConnect extension, users cannot connect, and the dev/test wallet fallback is missing.  
2. \*\*Mint endpoint failures\*\*: The minting process never triggers real contract calls because no valid walletAddress or provider is passed.  
3. \*\*Empty NFT Gallery Component\*\*: `ReceiptGallery.tsx` is empty, so users cannot see or select NFTs.  
4. \*\*Frontend does not pass walletAddress to upload or gallery\*\* – causing 400 errors.

## 2. Proposed Code Changes

### 2.1 Enhanced Wallet Hook with Dev Fallback

File: `frontend/src/hooks/useWalletConnect.ts`  
  
Update your hook to use a mock dev wallet if `window.ethereum` is unavailable:

```ts  
import { useState } from 'react';  
import { ethers } from 'ethers';  
  
export function useWalletConnect() {  
 const [provider, setProvider] = useState<ethers.providers.Provider | null>(null);  
 const [signer, setSigner] = useState<ethers.Signer | null>(null);  
 const [walletAddress, setWalletAddress] = useState<string | null>(null);  
  
 const connectMetaMask = async () => {  
 if (window.ethereum) {  
 const web3Provider = new ethers.providers.Web3Provider(window.ethereum);  
 await web3Provider.send('eth\_requestAccounts', []);  
 const web3Signer = web3Provider.getSigner();  
 const address = await web3Signer.getAddress();  
 setProvider(web3Provider);  
 setSigner(web3Signer);  
 setWalletAddress(address);  
 } else if (process.env.REACT\_APP\_DEV\_PRIVATE\_KEY) {  
 console.warn('No MetaMask found. Falling back to dev wallet.');  
 const devProvider = new ethers.providers.JsonRpcProvider(process.env.REACT\_APP\_POLYGON\_RPC\_URL);  
 const devWallet = new ethers.Wallet(process.env.REACT\_APP\_DEV\_PRIVATE\_KEY, devProvider);  
 setProvider(devProvider);  
 setSigner(devWallet);  
 setWalletAddress(await devWallet.getAddress());  
 } else {  
 alert('Please install MetaMask or provide a dev private key.');  
 }  
 };  
  
 // WalletConnect logic similar...  
 return { provider, signer, walletAddress, connectMetaMask };  
}  
```

### 2.2 Pass `walletAddress` to Upload and Mint

File: `frontend/src/components/ReceiptUpload.tsx`  
  
Ensure `walletAddress` is included in the upload form:

```tsx  
// Inside handleFileChange or submit function:  
const formData = new FormData();  
formData.append('receipt', file);  
formData.append('walletAddress', walletAddress || '');  
  
// Then POST to /api/upload-receipt  
```

### 2.3 Real Contract Minting Logic

File: `server/routes/nfts.ts`  
  
Ensure the backend mints using the signer from provider:

```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: 'Connect wallet first' });  
 }  
 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);  
  
 try {  
 const tx = await contract.mint(walletAddress, nftId);  
 const receipt = await tx.wait();  
 return res.json({ success: true, tokenId: nftId, txHash: receipt.transactionHash });  
 } catch (error) {  
 console.error('Mint error:', error);  
 return res.status(500).json({ success: false, msg: 'Minting failed', error: error.message });  
 }  
});  
```

### 2.4 Implement NFT Gallery Component

File: `frontend/src/components/ReceiptGallery.tsx`  
  
Replace empty file with NFTGallery logic to display minted NFTs:

```tsx  
import React, { useEffect, useState } from 'react';  
  
export default function ReceiptGallery({ walletAddress }: { walletAddress: string }) {  
 const [nfts, setNfts] = useState<any[]>([]);  
  
 useEffect(() => {  
 if (!walletAddress) return;  
 fetch(`/api/gallery/${walletAddress}`)  
 .then(res => res.json())  
 .then(data => setNfts(data.nfts || []));  
 }, [walletAddress]);  
  
 if (!walletAddress) {  
 return <p>Please connect your wallet to view your NFT Receipts.</p>;  
 }  
  
 return (  
 <div className="grid grid-cols-3 gap-4">  
 {nfts.map(nft => (  
 <div key={nft.tokenId} className="border p-2 rounded">  
 <img src={nft.image} alt={`NFT ${nft.tokenId}`} />  
 <p className="font-semibold">Token ID: {nft.tokenId}</p>  
 <p>Tier: {nft.tier}</p>  
 <button  
 className="mt-2 bg-blue-500 text-white px-3 py-1 rounded"  
 onClick={() => {/\* decrypt or view metadata \*/}}  
 >  
 Unlock Metadata  
 </button>  
 </div>  
 ))}  
 </div>  
 );  
}  
```

### 2.5 Place Gallery on Main Page

File: `frontend/src/App.tsx` or main layout:  
  
Import and render the gallery component:

```tsx  
import ReceiptGallery from './components/ReceiptGallery';  
...  
function App() {  
 const { walletAddress } = useWalletConnect();  
 return (  
 <div>  
 {/\* Your header, upload component, selection... \*/}  
 <ReceiptGallery walletAddress={walletAddress} />  
 </div>  
 );  
}  
```

## 3. Next Steps

1. Merge these changes and restart the dev server.  
2. Test wallet fallback by removing MetaMask and verifying devKey path works.  
3. Upload receipts and select NFTs; confirm on-chain mint via logs or Etherscan.  
4. Verify gallery populates correctly on the main page.  
5. Iterate on UI and error messaging.