# 🧾 BlockReceipt.ai – NFT Market Query, Rate Limiting & Fallback Minting

## 🔎 1. NFT Query from Reservoir API (OpenSea Alternative)

Install Axios if not yet present:

```bash  
npm install axios  
```

File: `server/utils/fetchAffordableNFT.js`

const axios = require('axios');  
  
// Search Reservoir for cheap NFTs (Polygon only)  
async function fetchAffordableNFT() {  
 try {  
 const response = await axios.get(  
 'https://api.reservoir.tools/tokens/v5',  
 {  
 params: {  
 limit: 20,  
 sortBy: 'floorAskPrice',  
 normalizeRoyalties: true,  
 includeTopBid: false,  
 displayCurrency: '0x0000000000000000000000000000000000000000', // ETH  
 collectionSetId: 'polygon', // filter by chain  
 priceFilter: '0.001-0.10' // 0.1 ETH max  
 },  
 headers: {  
 'x-api-key': process.env.RESERVOIR\_API\_KEY  
 }  
 }  
 );  
  
 const tokens = response.data.tokens || [];  
 return tokens.length ? tokens[Math.floor(Math.random() \* tokens.length)] : null;  
 } catch (error) {  
 console.error("Failed to fetch NFT:", error.message);  
 return null;  
 }  
}  
  
module.exports = { fetchAffordableNFT };

## 🔒 2. Wallet Rate Limiting

Simple in-memory limiter (upgrade to Redis for production):

File: `server/utils/claimLimiter.js`

const claims = new Map();  
  
function canClaim(walletAddress) {  
 const now = Date.now();  
 const lastClaim = claims.get(walletAddress);  
 if (!lastClaim || now - lastClaim > 24 \* 60 \* 60 \* 1000) {  
 claims.set(walletAddress, now);  
 return true;  
 }  
 return false;  
}  
  
module.exports = { canClaim };

## 🧾 3. Fallback Minting from Local NFT Collection

Use this when no NFT is found on the market. Assumes you have local art + metadata.

File: `server/utils/fallbackMint.js`

const { ethers } = require("ethers");  
const CONTRACT\_ABI = require('./YourNFTContractABI.json');  
  
const provider = new ethers.JsonRpcProvider('https://polygon-rpc.com');  
const wallet = new ethers.Wallet(process.env.PRIVATE\_KEY, provider);  
const contract = new ethers.Contract(process.env.CONTRACT\_ADDRESS, CONTRACT\_ABI, wallet);  
  
async function fallbackMint(userWallet, metadataURI) {  
 try {  
 const tx = await contract.mintReceipt(userWallet, metadataURI);  
 return await tx.wait();  
 } catch (error) {  
 console.error("Fallback mint failed:", error.message);  
 return null;  
 }  
}  
  
module.exports = { fallbackMint };

## 📦 Integration in `uploadReceipt.ts`

const { canClaim } = require('../utils/claimLimiter');  
const { fetchAffordableNFT } = require('../utils/fetchAffordableNFT');  
const { fallbackMint } = require('../utils/fallbackMint');  
  
...  
  
if (!canClaim(userWallet)) {  
 return res.status(429).json({ success: false, message: "NFT already claimed in last 24h" });  
}  
  
const marketNFT = await fetchAffordableNFT();  
if (marketNFT) {  
 await purchaseAndTransferNFT(marketNFT, userWallet);  
} else {  
 const fallbackMeta = 'ipfs://...'; // assign from pool  
 await fallbackMint(userWallet, fallbackMeta);  
}

## ✅ Final Notes

- Add proper logging and a claim database for production  
- Rotate fallback NFTs to ensure uniqueness  
- Rate-limit via memory for now; replace with Redis for scaling  
- Confirm NFT contract supports `mintReceipt(address,string)` function