# 🚀 BlockReceipt.ai – Polish & Launchpad Sprint Plan

## 📆 Sprint Overview

This 7-day sprint plan is designed to finalize and polish your BlockReceipt.ai MVP, ensuring a seamless user experience, robust on-chain integration, and readiness for public demo or beta launch.

## 🎯 Sprint Goals

1. Deploy and verify the ERC-1155 contract on Polygon.  
2. Pin all NFT metadata and images to IPFS and update references.  
3. Implement and seed the NFT pool database/service.  
4. Complete and test the mint endpoint with real transactions.  
5. Finalize frontend gallery, polling, and user feedback flows.  
6. Conduct a UX and messaging audit for onboarding clarity.  
7. Perform QA and compile test cases for end-to-end flows.

## 📂 Day-by-Day Roadmap

### Day 1: Smart Contract Deployment & Verification

- Review and compile `contracts/BlockReceiptCollection.sol`.  
- Deploy to Polygon testnet (Amoy or Mumbai) via Hardhat.  
- Run verification script (`verify-contract.js`) and confirm on Polygonscan.  
- Update `.env` with `CONTRACT\_ADDRESS`.  
- Smoke-test contract using `scripts/check-contract-on-chain.js`.

### Day 2: Metadata Pinning to IPFS

- Gather all NFT images and JSON metadata files in `data/nft\_pool/` folder.  
- Use NFT.Storage or Pinata SDK to pin images and JSON to IPFS.  
- Capture returned CIDs and update `data/nft\_pool.json` with `ipfs://` URIs.  
- Validate URIs manually via gateway to ensure accessibility.

### Day 3: NFT Pool Service Implementation

- Create `server/services/NFTPoolService.ts` with functions:  
 ```ts  
 async function getNFTOptionsForTier(tier: string) {  
 const pool = await db.nftPool.find({ tier }).toArray();  
 return shuffle(pool).slice(0, 5);  
 }  
 ```  
- Seed the `nftPool` collection in MongoDB (or JSON file store) with pinned metadata entries.  
- Add `/api/nfts/pool` route to return options based on `receiptTier`.

### Day 4: Backend Mint Endpoint & Wallet Enforcement

- Update `server/routes/nfts.ts` mint endpoint to call real contract:  
 ```ts  
 const tx = await contract.mint(walletAddress, nftId);  
 await tx.wait();  
 ```  
- Add guard in `uploadReceipt.ts`:  
 ```ts  
 if (!req.body.walletAddress) throw Error('Connect wallet first');  
 ```  
- Test minting flow with sample `curl` or Postman scripts.  
- Ensure dev wallet has sufficient MATIC for gas.

### Day 5: Frontend Gallery, Polling & Feedback

- Integrate `useGalleryPoll` hook to monitor `/api/task/:taskId/status`.  
- In `client/pages/gallery.tsx`, fetch `/api/gallery/:walletAddress` on load and after mint.  
- Add success toasts using your UI library (e.g., shadcn/ui or Tailwind alerts):  
 ```jsx  
 toast.success('NFT minted! Check your gallery.');  
 ```  
- Style the `NFTGallery` component with locked/unlocked badges.  
- Ensure mobile responsiveness and loading states.

### Day 6: UX & Messaging Audit

- Review all user flows: signup, upload, selection, mint, gallery.  
- Simplify copy: use terms like \*\*MintVault\*\* or \*\*ProofStack\*\* for gallery header.  
- Add inline help/tooltips explaining TACo encryption and NFT process.  
- Include wallet backup reminder modal after wallet creation.

### Day 7: QA & Testing

- Execute end-to-end tests:  
 1. Sign up with email & wallet.  
 2. Upload receipt; verify automatic minting.  
 3. Choose NFT; confirm on-chain via Polygonscan.  
 4. View in gallery; test unlock metadata.  
- Write up test cases and record any bugs.  
- Prepare demo script with screenshots or recording.

## ✅ Sprint Completion Criteria

- ERC-1155 deployed and verified on testnet.  
- NFT metadata pinned, pool service functional.  
- Mint endpoint live and enforced with wallet guard.  
- Frontend gives real-time feedback and gallery refresh.  
- UX flows polished and all key messages in place.  
- End-to-end QA passed without critical bugs.