# 🌍 BlockReceipt.ai – Scalable Passport Stamp Integration

## 1. The Challenge

A city-based passport stamp theme is engaging, but there are far fewer city seals than potential receipts. To maintain uniqueness and scalability, we need to combine a fixed 'city emblem' with dynamic, generative overlays.

## 2. Scalable Stamp Architecture

Each NFT stamp will consist of:  
A. \*\*Base City Seal\*\* – a static graphic representing the city or region (e.g., Eiffel Tower for Paris).  
B. \*\*Dynamic Overlay\*\* – procedurally generated patterns, color schemes, and icons derived from the receipt data, merchant category, and timestamp, ensuring each stamp is unique.  
C. \*\*Promo Badge\*\* – vendor promo indicator (e.g., a colored border or icon) that changes or fades when expired.

## 3. Technical Implementation

### 3.1 Generative Stamp Service

File: `server/services/stampService.ts`

```ts  
import { createCanvas, loadImage } from 'canvas';  
import { pinFileToIPFS } from './ipfsService';  
  
// Deterministic PRNG for overlays  
function mulberry32(a: number): () => number {  
 return () => {  
 let t = a += 0x6D2B79F5;  
 t = Math.imul(t ^ t >>> 15, t | 1);  
 t ^= t + Math.imul(t ^ t >>> 7, t | 61);  
 return ((t ^ t >>> 14) >>> 0) / 4294967296;  
 };  
}  
  
interface StampOptions {  
 cityCode: string;  
 receiptHash: string;  
 merchantCategory: string;  
 timestamp: number;  
 promoActive: boolean;  
}  
  
export async function generatePassportStamp(options: StampOptions): Promise<string> {  
 const { cityCode, receiptHash, merchantCategory, timestamp, promoActive } = options;  
 const seedInput = receiptHash + merchantCategory + timestamp.toString();  
 const seed = parseInt(seedInput.slice(0, 8), 16);  
 const rand = mulberry32(seed);  
  
 const canvas = createCanvas(512, 512);  
 const ctx = canvas.getContext('2d');  
  
 // 1. Draw base city seal  
 const baseImage = await loadImage(`assets/cities/${cityCode}.png`);  
 ctx.drawImage(baseImage, 0, 0, 512, 512);  
  
 // 2. Draw dynamic overlay (circles, lines, patterns)  
 for (let i = 0; i < 5; i++) {  
 ctx.strokeStyle = `hsl(${rand() \* 360}, 70%, 50%)`;  
 ctx.lineWidth = 5 + rand() \* 10;  
 ctx.beginPath();  
 ctx.arc(256, 256, 50 + i \* 30, rand() \* Math.PI, 2 \* Math.PI \* rand());  
 ctx.stroke();  
 }  
  
 // 3. Optional merchant icon overlay  
 const icon = await loadImage(`assets/icons/${merchantCategory}.png`);  
 ctx.drawImage(icon, 400, 400, 64, 64);  
  
 // 4. Draw promo badge if active  
 if (promoActive) {  
 ctx.fillStyle = 'rgba(255, 215, 0, 0.6)';  
 ctx.font = 'bold 48px Sans';  
 ctx.fillText('PROMO', 350, 100);  
 }  
  
 // 5. Pin to IPFS  
 const buffer = canvas.toBuffer('image/png');  
 const { cid } = await pinFileToIPFS(buffer, `stamp-${options.receiptHash}.png`);  
 return `ipfs://${cid}`;  
}  
```

### 3.2 Mint Route Update

File: `server/routes/uploadAndMint.ts` (excerpt)

```ts  
import { generatePassportStamp } from '../services/stampService';  
// Inside upload-and-mint handler after receiptData extraction  
const cityCode = receiptData.cityCode; // via geocoding of merchant address  
const stampUri = await generatePassportStamp({  
 cityCode,  
 receiptHash: tx.hash,  
 merchantCategory: receiptData.category,  
 timestamp: Date.now(),  
 promoActive: promoData.expiresAt > Date.now()  
});  
  
// Build metadata  
const metadata = {  
 name: `Receipt #${tokenId}`,  
 description: `Minted in ${cityCode}`,  
 stamp: stampUri,  
 userData,  
 promoData  
};  
const metadataUri = await pinJSONToIPFS(metadata);  
  
// Mint NFT with metadataUri  
await contract.mintNewReceipt(walletAddress, metadataUri);  
```

### 3.3 Frontend Gallery Integration

Update `ReceiptGallery.tsx` to display the stamp image:

jsx  
<div className="nft-card">  
 <img src={nft.metadata.stamp} alt="Passport Stamp" className="w-full h-auto mb-2" />  
 <h3>{nft.name}</h3>  
 {/\* Reveal user and promo data buttons \*/}  
</div>  
```

## 4. Scaling to Millions

Even with a limited set of base city seals, we achieve effectively unlimited unique stamps by:  
1. \*\*Dynamic Overlays:\*\* Using a deterministic PRNG seed from receiptHash, merchantCategory, and timestamp to draw varied patterns.  
2. \*\*Geohash Buckets:\*\* Optionally derive more granular base images from geohash prefixes for micro-locations.  
3. \*\*Merchant Icons:\*\* Overlay category icons (e.g., shampoo, snacks) to diversify visuals.  
4. \*\*Time-Based Palettes:\*\* Use purchase time (morning/evening) to select color schemes.  
  
This seed space is vast, so every mint—even in the same city—yields a distinct artwork.

## 5. Optimization & Caching

- \*\*On-Demand Generation:\*\* Only generate and pin stamps when a user mints a receipt.  
- \*\*Seed Cache:\*\* Store `seed → cid` in a KV store to avoid re-pin duplicates (extremely rare).  
- \*\*Stateless Scaling:\*\* Deploy `stampService` as an autoscaled function or container for high throughput.

## 6. Replit Integration Instructions

1. Install `canvas` and `node-fetch`.  
2. Add city seal PNGs in `server/assets/cities/` and merchant icons in `server/assets/icons/`.  
3. Create `stampService.ts` with the generative code.  
4. Update `uploadAndMint.ts` to call `generatePassportStamp` and set `stamp` field in metadata.  
5. Pin city assets to IPFS or serve via static hosting.  
6. Modify `ReceiptGallery.tsx` to render `nft.metadata.stamp` images.  
7. Optionally implement seed caching in your database.