Perfect 👊 — 60 days is enough to get something *real* off the ground and see if people will pay. Here’s a **lean & mean roadmap** designed to give you a niche tool or service that can actually generate income, without you wasting another year building “maybe” projects.

**🚀 60-Day “Micro-Niche Income” Roadmap**

**Phase 1 (Days 1–10): Pick & Validate the Niche**

* Criteria:
  + Small enough that big SaaS ignores it.
  + Painful problem → people already solve it with Excel, PDFs, or paper.
  + You know *something* about it (so you can talk the lingo).
* Possible targets from your background:
  + Cabinet/kitchen installers (packing slips, delivery tracking).
  + Small construction contractors (tool/equipment checkout, expense tracking).
  + Local nonprofits (donor + invoice tracking).

👉 Deliverable: Pick ONE niche. (I’ll help you validate with quick online checks).

**Phase 2 (Days 11–25): Build a “Minimum Money Tool”**

* Goal: Don’t over-engineer. Build a tool that does ONE job.
* Options:
  + **App (Streamlit or simple web app)** → upload a PDF, get a clean Excel/CSV.
  + **Excel + Script Toolkit** → downloadable workbook + automation button.
  + **Hybrid** → your WordPress site with a “tool” page (upload → download result).

👉 Deliverable: A working MVP you can demo live, even if ugly.

**Phase 3 (Days 26–40): Package & Launch**

* Make it look real, not just a side project.
* Create:
  + A landing page (“Upload your [niche problem doc] → get results instantly”).
  + Simple pricing (flat fee or $19/month).
  + Demo video (30 sec screen recording).

👉 Deliverable: A page you can show to real businesses.

**Phase 4 (Days 41–55): Find First Paying User**

* Local outreach → Call/email 10–20 local businesses in your niche.
* Online outreach → Niche FB groups, LinkedIn, Reddit (not Fiverr/Upwork, too crowded).
* Offer free trial for 1 week → then convert to paying.

👉 Deliverable: 1–3 actual users who’ve tested it.

**Phase 5 (Days 56–60): Iterate + Decide**

* If at least one user pays → double down.
* If no traction → pivot niche fast (but reuse same code/site).

👉 Deliverable: A paying customer OR a pivot target with proof it’s worth chasing.

**Why This Works**

* It keeps you **out of the Fiverr rat race**.
* It forces you to **go niche, not broad**.
* It creates something you can either monetize directly OR flip to another builder.

💡 Here’s the kicker: if you treat this like a “challenge,” you’ll either (1) have your first paying tool in 60 days, or (2) have proof you need to pivot, without wasting years. Both are wins.

👉 My question for you now: Which niche do you want to target first —  
**(1) Cabinet/Installer tracking (your strongest advantage), (2) Small contractors/tools & expenses, or (3) Local nonprofits tracking money/donors?**

That’ll set our starting line.

*Incoming product with scanning, then match to scanned PDF packing slips. Then being able to match with inventory. What do you think? It could have tool tracking, in's & out?*

🔥 Now you’re talking, Michelle. That idea has *legs*. Let me break it down:

**🛠️ The Concept (In Plain English)**

You’re basically describing a **“Mini Supply Chain + Asset Tracker”** for small businesses.

* **Step 1:** Driver/employee scans a barcode (incoming product).
* **Step 2:** System checks against the *original PDF packing slip* (what was supposed to arrive).
* **Step 3:** Matches to current **inventory records** → shows what’s in/out, what’s missing.
* **Step 4 (bonus):** Same system could track **tools or equipment** → when a tool goes out/in, scan it and log.

This is *not* just another inventory app. It’s tailored to **small distributors/installers/contractors** who currently do this on paper or half-assed spreadsheets.

**💰 Why It’s Promising**

* **Pain point:** Lost/missing inventory costs money. Tool loss is huge in construction.
* **Target:** Small-to-medium businesses (contractors, cabinet installers, plumbers, HVAC, warehouses).
* **Competition check:** Big SaaS players (NetSuite, Fishbowl, SAP) are overkill. Small guys use Excel → you’d be their upgrade.
* **Monetization:**
  + Flat monthly fee ($49–$199, depending on users).
  + Or license per seat/device.

**⚡ The 60-Day Version (MVP Scope)**

You don’t need “Amazon-level” software. Just prove it works:

1. **Scanning**: Android/iOS app (or web app) that can scan barcodes.
2. **PDF Matching**: Upload packing slip → extract line items (this is where your scripts come in!).
3. **Basic Inventory DB**: SQLite/Postgres → shows “expected vs. scanned.”
4. **Simple Tool Tracker**: Check-out/check-in with a timestamp + user ID.

👉 If you have *just those 4 features*, you could show it to a local contractor or delivery service and get them interested.

**🧩 Expansion Path**

* GPS location logging (“where was it scanned?”).
* Multi-user accounts (admin vs. driver).
* Alerts (if scanned item not on slip, or if tool not returned).
* Cloud sync → customer portal (track status online).

This is *exactly* the kind of project that looks small but can snowball into SaaS.

**🚦 My Take**

This idea is stronger than expense trackers or generic Fiverr gigs.  
Why? Because it ties together:

* **Your existing work** (PDF → Excel extraction).
* **Your logistics background** (cabinet tracking).
* **A real, expensive pain point** (lost inventory/tools).

Instead of being “one of thousands,” you’d be *the* person offering a practical, niche system.

👉 Next step: Do you want me to sketch out a **lean MVP design** (what tech stack, how it flows, what *not* to build yet), so you can realistically launch a demo in 60 days? That’ll keep us from building too much too soon.

*Yes please. Lets go. What do we do first?*

Perfect. Let’s lock this in. We’ll treat this like a **startup sprint**, but practical — 60 days, minimum viable, enough to show it to a real business. Here’s how we start:

**🛠️ MVP Blueprint: Barcode + Packing Slip + Inventory**

**Core Features (MVP)**

We only build what’s needed to *prove the concept*:

1. **Scan Product Barcode** (mobile/web app → saves data).
2. **Upload Packing Slip PDF** (auto-extract items → Excel/DB).
3. **Match Scans to Packing Slip** (show “✅ matched” or “❌ missing”).
4. **Simple Inventory Table** (shows “in stock” / “checked out”).
5. **Tool Tracking (basic)** → scan out, scan back in.

That’s it. No fancy dashboards, no AI predictions — just the loop: *Slip → Scan → Match → Inventory*.

**Tech Stack (Beginner-Friendly)**

* **Frontend (scanning app):**
  + Option 1: **Streamlit** web app (fast, deployable, can use camera scanning via pyzbar).
  + Option 2: **React Native/Expo** (if you want real mobile apps on iOS/Android).
* **Backend/Database:**
  + **SQLite** (simple, local DB → can upgrade later).
  + Or **Supabase** (cloud Postgres + auth, free tier).
* **PDF Extraction:**
  + Python (pdf2image, pytesseract, pdfplumber) → you already have pieces of this.
  + Each manufacturer = its own parser script (as we discussed before).
* **Hosting:**
  + MVP: Run locally on laptop / Android device.
  + Later: Deploy web app (Streamlit Cloud, Vercel, or Supabase).

**Workflow**

**Step 1:** Upload Packing Slip PDF → system extracts PO, items, quantities.  
**Step 2:** Driver scans product barcodes → app records each scan.  
**Step 3:** System compares scanned items with packing slip → flags extras/missing.  
**Step 4:** Updates inventory DB → shows stock levels.  
**Step 5:** Tool check-in/out → same scanning loop, but against “tools” table.

**🚦 Next 10 Days (Phase 1: Foundations)**

**Day 1–3: Setup & Skeleton**

* Create Git repo/project folder.
* Decide: **Streamlit (web MVP)** or **React Native (mobile-first)**.
* Set up SQLite DB with 3 tables:
  1. packing\_slips (id, po\_number, item\_number, description, qty\_expected)
  2. scans (id, slip\_id, item\_number, qty\_scanned, timestamp, user)
  3. tools (id, name, status [in/out], user, timestamp)

**Day 4–6: PDF → DB Pipeline**

* Build a Python script that:
  + Reads PDF packing slip.
  + Extracts items + quantities.
  + Inserts into packing\_slips table.

**Day 7–10: Barcode Scanning → DB**

* Build scan interface (Streamlit or mobile).
* Each scan adds row into scans table.
* Create simple comparison: show ✅ if match, ❌ if missing.

👉 By **Day 10**, you should be able to:

* Upload a PDF.
* Scan 1–2 items.
* See a result: “PO1234 — 8/10 items received.”

That’s a working prototype you can *show*.

⚡ Question for you before I sketch the **exact Day 1 setup commands/code**:  
Do you want to start with **Streamlit (faster, web-based MVP)** or go straight into **React Native (harder, but true mobile app for Android/iPhone)**?

Streamlit = you’ll see results in 1–2 weeks.  
React Native = more powerful, but will slow down the 60-day challenge.

*Option 2: React Native/Expo (if you want real mobile apps on iOS/Android). & Supabase (cloud Postgres + auth, free tier). & Python (pdf2image, pytesseract, pdfplumber) → you already have pieces of this, this is where I have been struggling. I can't get it to read the PDF correctly. Vercel, or Supabase- I already have accounts for both for another app, also Github-----*

*Day 1–3: Setup & Skeleton*

1. *Create Git repo/project folder. Already have an account.*
2. *Decide: Streamlit (web MVP) or React Native (mobile-first).- Whatever works better. I want to do it right*
3. *Set up SQLite DB with 3 tables: packing\_slips (id, po\_number, item\_number, description, qty\_expected), scans (id, slip\_id, item\_number, qty\_scanned, timestamp, user), tools (id, name, status [in/out], user, timestamp)*

Awesome — you’re locking in a **real app stack**: React Native/Expo for mobile, Supabase for backend/auth, Python for PDF extraction. That’s a *solid combo* and future-proof. Let’s get **Day 1–3: Setup & Skeleton** mapped clearly so you don’t get lost.

**🔧 Day 1–3 Setup & Skeleton**

**1. Create Project Repos**

Since you’ll have **two pieces** (mobile app + PDF extractor), keep them cleanly separated in GitHub:

* inventory-tracker-app → React Native/Expo code.
* inventory-tracker-backend → Python scripts for PDF parsing + API functions (will connect to Supabase).

💡 Later, we’ll link them (mobile app calls Supabase APIs, backend parses PDFs and pushes results).

**2. Supabase Setup**

You said you already have an account 👍. Let’s create the tables:

**Tables**

-- packing\_slips

create table packing\_slips (

id uuid primary key default gen\_random\_uuid(),

po\_number text not null,

item\_number text,

description text,

qty\_expected int,

created\_at timestamp default now()

);

-- scans

create table scans (

id uuid primary key default gen\_random\_uuid(),

slip\_id uuid references packing\_slips(id) on delete cascade,

item\_number text,

qty\_scanned int default 1,

scanned\_by text,

scanned\_at timestamp default now()

);

-- tools

create table tools (

id uuid primary key default gen\_random\_uuid(),

name text not null,

status text check (status in ('in','out')) default 'in',

assigned\_to text,

updated\_at timestamp default now()

);

👉 Supabase dashboard → SQL editor → paste these → run. Now your DB is ready.

**3. React Native/Expo App (Frontend Skeleton)**

From terminal:

npx create-expo-app inventory-tracker-app

cd inventory-tracker-app

npm install @supabase/supabase-js expo-barcode-scanner

This gives you:

* A working Expo app.
* Supabase client.
* Barcode scanner (camera ready).

📱 Example skeleton App.js:

import React, { useState, useEffect } from 'react';

import { Text, View, Button } from 'react-native';

import { Camera } from 'expo-camera';

import { createClient } from '@supabase/supabase-js';

// replace with your Supabase keys

const supabaseUrl = "https://YOUR\_PROJECT.supabase.co";

const supabaseKey = "YOUR\_ANON\_KEY";

const supabase = createClient(supabaseUrl, supabaseKey);

export default function App() {

const [hasPermission, setHasPermission] = useState(null);

const [scanned, setScanned] = useState(false);

useEffect(() => {

(async () => {

const { status } = await Camera.requestCameraPermissionsAsync();

setHasPermission(status === 'granted');

})();

}, []);

const handleBarCodeScanned = async ({ type, data }) => {

setScanned(true);

alert(`Scanned: ${data}`);

// Example insert into Supabase "scans"

const { error } = await supabase.from('scans').insert([

{ item\_number: data, qty\_scanned: 1, scanned\_by: "test\_user" }

]);

if (error) console.error(error);

};

if (hasPermission === null) {

return <Text>Requesting camera permission</Text>;

}

if (hasPermission === false) {

return <Text>No camera access</Text>;

}

return (

<View style={{ flex: 1 }}>

<Camera

style={{ flex: 1 }}

onBarCodeScanned={scanned ? undefined : handleBarCodeScanned}

/>

{scanned && <Button title={'Tap to Scan Again'} onPress={() => setScanned(false)} />}

</View>

);

}

👉 At this point:

* You can scan a barcode.
* It pushes the scan into your Supabase scans table.

That’s **huge progress for Day 3**.

**4. Python Backend Skeleton**

This will live in inventory-tracker-backend.

mkdir inventory-tracker-backend

cd inventory-tracker-backend

python3 -m venv venv

source venv/bin/activate # or venv\Scripts\activate on Windows

pip install pdfplumber pytesseract pdf2image supabase

Basic script parse\_packing\_slip.py:

import pdfplumber

from supabase import create\_client, Client

# Connect Supabase

url = "https://YOUR\_PROJECT.supabase.co"

key = "YOUR\_SERVICE\_ROLE\_KEY" # NOT anon key

supabase: Client = create\_client(url, key)

def extract\_items(pdf\_file):

items = []

with pdfplumber.open(pdf\_file) as pdf:

for page in pdf.pages:

text = page.extract\_text()

for line in text.splitlines():

if line.strip().startswith("Item"): # crude example

parts = line.split()

items.append({

"item\_number": parts[1],

"description": " ".join(parts[2:-1]),

"qty\_expected": int(parts[-1])

})

return items

def upload\_to\_supabase(po\_number, items):

for item in items:

supabase.table("packing\_slips").insert({

"po\_number": po\_number,

"item\_number": item["item\_number"],

"description": item["description"],

"qty\_expected": item["qty\_expected"]

}).execute()

if \_\_name\_\_ == "\_\_main\_\_":

po = "PO12345"

items = extract\_items("example.pdf")

upload\_to\_supabase(po, items)

👉 By Day 3, you should be able to:

* Run Python script → extract dummy data → insert into Supabase.
* Scan barcode with Expo app → see it appear in scans table.

That’s the **skeleton loop** working.

✅ **Deliverable by Day 3:**

* GitHub repos created.
* Supabase DB with 3 tables.
* Expo app that scans → logs to DB.
* Python script that parses a PDF → logs expected items to DB.

⚡ Next Step (Day 4–6): We’ll refine PDF parsing (your pain point) so it actually extracts correctly.

Do you want me to prep a **Supabase SQL dump + starter GitHub repo structure** for you, so you don’t waste time copy-pasting setup pieces?

*#3-I did the bash code. Now do I create the App.js in the folder?*