* A **FastAPI** microservice that exposes your CleanScore logic over HTTPS
* A minimal **React Native (Expo)** screen that calls the API, supports paste-entry and CSV upload, and renders **three tiers**

# **1) FastAPI service (Python)**

## **a) app/main.py**

# app/main.py  
from \_\_future\_\_ import annotations  
from typing import List, Optional, Dict, Tuple  
from fastapi import FastAPI  
from fastapi.middleware.cors import CORSMiddleware  
from pydantic import BaseModel  
import re, unicodedata  
  
app = FastAPI(title="CleanScore API", version="1.0.0")  
  
# CORS for your mobile app domain(s)  
app.add\_middleware(  
 CORSMiddleware,  
 allow\_origins=["\*"], # tighten in prod: ["<https://yourapp.example>"]  
 allow\_credentials=True,  
 allow\_methods=["\*"],  
 allow\_headers=["\*"],  
)  
  
# ---------- heuristic lists (same spirit as MVP; keep transparent) ----------  
STRONG\_NEG = [  
 "high fructose corn syrup","corn syrup","partially hydrogenated","hydrogenated","trans fat","shortening",  
 "bht","bha","tbhq","propyl gallate","monosodium glutamate","msg","red 40","yellow 5","yellow 6",  
 "blue 1","blue 2","caramel color","sodium nitrite","sodium nitrate","nitrite","nitrate",  
 "polysorbate 80","propylene glycol","titanium dioxide","potassium bromate","bromated",  
 "aspartame","sucralose","acesulfame","saccharin","carboxymethylcellulose","carrageenan",  
 "artificial flavor","artificial flavours","artificial color","artificial colour",  
]  
MOD\_NEG = [  
 "sugar","cane sugar","glucose","dextrose","fructose","invert sugar","maltodextrin",  
 "enriched flour","bleached flour","white flour","refined flour",  
 "canola oil","soybean oil","vegetable oil","palm oil","corn oil","rapeseed oil",  
 "sodium benzoate","potassium sorbate","disodium phosphate","phosphate",  
 "xanthan gum","guar gum","natural flavor","natural flavour",  
]  
POSITIVE = [  
 "organic","non-gmo","no added sugar","unsweetened","unsalted","low sodium","sprouted","fermented","probiotic",  
 "grass-fed","pasture-raised","wild-caught","extra virgin olive oil","olive oil","avocado oil",  
 "whole grain","100% whole","whole wheat","steel-cut oats","brown rice","quinoa","lentils","beans","chickpeas",  
 "high fiber","plain yogurt","greek yogurt","whole milk yogurt","no nitrate","no nitrite","nitrite-free","nitrate-free",  
]  
ULTRA\_PROCESSED\_HINTS = [  
 "chips","soda","soft drink","energy drink","candy","cookie","cake","donut","instant noodle","ramen",  
 "frozen dinner","processed meat","hot dog","bacon","sausage","sugary cereal","sweetened cereal","ice cream",  
 "fries","fried",  
]  
WHOLE\_FOOD\_HINTS = [  
 "spinach","kale","broccoli","berries","apple","banana","carrot","lettuce","tomato","avocado","onion","garlic",  
 "pepper","orange","grapefruit","quinoa","oats","oatmeal","lentils","beans","chickpeas","nuts","almonds",  
 "walnuts","sunflower seeds","pumpkin seeds","yogurt","plain yogurt","greek yogurt","eggs","salmon","sardines",  
]  
  
def \_norm(s: str) -> str:  
 s = (s or "").lower()  
 s = unicodedata.normalize("NFKC", s)  
 return re.sub(r"\s+", " ", s).strip()  
  
def score\_item(item: str, ingredients: Optional[str]) -> Tuple[int, Dict[str, list]]:  
 base = 70  
 item\_t = \_norm(item)  
 ing\_t = \_norm(ingredients or item)  
  
 matched = {"positive": [], "moderate\_neg": [], "strong\_neg": [], "whole\_food": [], "ultra\_processed": []}  
 for kw in POSITIVE:  
 if kw in ing\_t: matched["positive"].append(kw)  
 for kw in MOD\_NEG:  
 if kw in ing\_t: matched["moderate\_neg"].append(kw)  
 for kw in STRONG\_NEG:  
 if kw in ing\_t: matched["strong\_neg"].append(kw)  
 for kw in WHOLE\_FOOD\_HINTS:  
 if kw in item\_t: matched["whole\_food"].append(kw)  
 for kw in ULTRA\_PROCESSED\_HINTS:  
 if kw in item\_t: matched["ultra\_processed"].append(kw)  
  
 score = base  
 score += 6 \* len(set(matched["positive"]))  
 score -= 7 \* len(set(matched["moderate\_neg"]))  
 score -= 15 \* len(set(matched["strong\_neg"]))  
 score += 5 \* min(2, len(set(matched["whole\_food"])))  
 score -= 10 \* min(2, len(set(matched["ultra\_processed"])))  
 if ("organic" in matched["positive"]) and any(s in ing\_t for s in ["cane sugar","sugar"]):  
 score -= 3  
 score = max(0, min(100, int(round(score))))  
 return score, matched  
  
def tier\_of(score: int, t1: int, t2: int) -> str:  
 if score >= t1:  
 return "Tier 1"  
 elif score >= t2:  
 return "Tier 2"  
 return "Tier 3"  
  
# ---------- Pydantic schemas ----------  
class ItemIn(BaseModel):  
 item: str  
 ingredients: Optional[str] = None  
  
class BatchIn(BaseModel):  
 items: List[ItemIn]  
 tier1\_threshold: int = 80  
 tier2\_threshold: int = 60  
  
class ItemOut(BaseModel):  
 item: str  
 ingredients: Optional[str]  
 clean\_score: int  
 tier: str  
 matched\_positive: List[str]  
 matched\_moderate\_neg: List[str]  
 matched\_strong\_neg: List[str]  
 hints\_whole\_food: List[str]  
 hints\_ultra\_processed: List[str]  
  
class BatchOut(BaseModel):  
 results: List[ItemOut]  
 summary: Dict[str, int] # counts per tier  
  
# ---------- Routes ----------  
@app.get("/healthz")  
def healthz():  
 return {"ok": True}  
  
@app.post("/score", response\_model=ItemOut)  
def score\_endpoint(payload: ItemIn, tier1\_threshold: int = 80, tier2\_threshold: int = 60):  
 score, m = score\_item(payload.item, payload.ingredients)  
 return ItemOut(  
 item=payload.item,  
 ingredients=payload.ingredients,  
 clean\_score=score,  
 tier=tier\_of(score, tier1\_threshold, tier2\_threshold),  
 matched\_positive=sorted(set(m["positive"])),  
 matched\_moderate\_neg=sorted(set(m["moderate\_neg"])),  
 matched\_strong\_neg=sorted(set(m["strong\_neg"])),  
 hints\_whole\_food=sorted(set(m["whole\_food"])),  
 hints\_ultra\_processed=sorted(set(m["ultra\_processed"])),  
 )  
  
@app.post("/score-batch", response\_model=BatchOut)  
def score\_batch(payload: BatchIn):  
 t1, t2 = payload.tier1\_threshold, payload.tier2\_threshold  
 results: List[ItemOut] = []  
 counts = {"Tier 1": 0, "Tier 2": 0, "Tier 3": 0}  
 for it in payload.items:  
 one = score\_endpoint(it, t1, t2)  
 results.append(one)  
 counts[one.tier] += 1  
 return BatchOut(results=results, summary=counts)

## **b) requirements.txt**

fastapi>=0.111,<1.0  
uvicorn[standard]>=0.29,<1.0

## **c) Local run**

pip install -r requirements.txt  
uvicorn app.main:app --reload --port 8080  
# API docs: <http://localhost:8080/docs>

## **d) (Optional) Dockerfile**

FROM python:3.11-slim  
WORKDIR /app  
COPY requirements.txt .  
RUN pip install --no-cache-dir -r requirements.txt  
COPY app ./app  
EXPOSE 8080  
CMD ["uvicorn","app.main:app","--host","0.0.0.0","--port","8080"]

# **2) React Native (Expo) client**

Quick start:

npm install -g expo-cli  
npx create-expo-app cleanscore-mobile  
cd cleanscore-mobile  
npm i papaparse

Put the screen below in app/index.tsx (Expo Router) **or** App.tsx (single-screen).

### **App.tsx (minimal)**

import React, { useState } from "react";  
import { SafeAreaView, View, Text, TextInput, Button, FlatList, TouchableOpacity, Alert } from "react-native";  
import \* as DocumentPicker from "expo-document-picker";  
import Papa from "papaparse";  
  
type Item = { item: string; ingredients?: string };  
type ApiItemOut = {  
 item: string; ingredients?: string; clean\_score: number; tier: "Tier 1"|"Tier 2"|"Tier 3";  
 matched\_positive: string[]; matched\_moderate\_neg: string[]; matched\_strong\_neg: string[];  
 hints\_whole\_food: string[]; hints\_ultra\_processed: string[];  
};  
  
const API\_BASE = "<http://YOUR-API-HOST:8080>"; // e.g., <https://api.yourdomain.com>export default function App() {  
 const [text, setText] = useState("");  
 const [items, setItems] = useState<Item[]>([]);  
 const [t1, setT1] = useState("80");  
 const [t2, setT2] = useState("60");  
 const [results, setResults] = useState<ApiItemOut[] | null>(null);  
 const [loading, setLoading] = useState(false);  
  
 const addLine = () => {  
 const line = text.trim();  
 if (!line) return;  
 // support: "Item - ingredients" | "Item, ingredients" | "Item"  
 let item = line, ings: string | undefined = undefined;  
 if (line.includes(" - ")) {  
 const [a,b] = line.split(" - ", 1+1);  
 item = a.trim(); ings = b.trim();  
 } else if (line.includes(",")) {  
 const [a,b] = line.split(",", 1+1);  
 item = a.trim(); ings = b.trim();  
 }  
 setItems(prev => [...prev, { item, ingredients: ings }]);  
 setText("");  
 };  
  
 const pickCSV = async () => {  
 const res = await DocumentPicker.getDocumentAsync({ type: "text/\*", copyToCacheDirectory: true });  
 if (res.canceled || !res.assets?.[0]) return;  
 const asset = res.assets[0];  
 const file = await fetch(asset.uri);  
 const blob = await file.blob();  
 const textData = await blob.text();  
 const parsed = Papa.parse(textData, { header: true });  
 if (parsed.errors.length) {  
 Alert.alert("CSV Parse Error", parsed.errors[0].message);  
 return;  
 }  
 // try to detect Item/Ingredients columns  
 const rows: Item[] = [];  
 const mapKey = (k: string) => k.toLowerCase().trim();  
 parsed.data.forEach((row: any) => {  
 const keys = Object.keys(row).reduce((acc: any, k: string) => { acc[mapKey(k)] = k; return acc; }, {});  
 const itemKey = keys["item"] || keys["product"] || keys["name"] || keys["title"] || Object.keys(row)[0];  
 if (!itemKey) return;  
 const ingKey = keys["ingredients"] || keys["ingredient"];  
 const itemVal = (row[itemKey] || "").toString().trim();  
 if (!itemVal) return;  
 rows.push({ item: itemVal, ingredients: ingKey ? (row[ingKey] || "").toString().trim() : undefined });  
 });  
 setItems(prev => [...prev, ...rows]);  
 };  
  
 const clearAll = () => { setItems([]); setResults(null); };  
  
 const scoreBatch = async () => {  
 if (!items.length) { Alert.alert("No items", "Add or upload at least one item."); return; }  
 setLoading(true);  
 try {  
 const payload = {  
 items: items.map(i => ({ item: i.item, ingredients: i.ingredients || null })),  
 tier1\_threshold: Number(t1) || 80,  
 tier2\_threshold: Number(t2) || 60,  
 };  
 const resp = await fetch(`${API\_BASE}/score-batch`, {  
 method: "POST",  
 headers: { "Content-Type": "application/json" },  
 body: JSON.stringify(payload),  
 });  
 if (!resp.ok) throw new Error(`HTTP ${resp.status}`);  
 const data = await resp.json();  
 setResults(data.results);  
 // Optional: use data.summary for quick counts  
 } catch (e: any) {  
 Alert.alert("Error", e.message || "Failed to score.");  
 } finally {  
 setLoading(false);  
 }  
 };  
  
 const renderItem = ({ item }: { item: ApiItemOut }) => (  
 <View style={{ padding: 10, borderBottomWidth: 1, borderColor: "#eee" }}>  
 <Text style={{ fontWeight: "600" }}>{item.item}</Text>  
 {item.ingredients ? <Text numberOfLines={2} style={{ color: "#555" }}>{item.ingredients}</Text> : null}  
 <Text>Score: {item.clean\_score} • {item.tier}</Text>  
 </View>  
 );  
  
 const tiered = (tier: "Tier 1"|"Tier 2"|"Tier 3") =>  
 results?.filter(r => r.tier === tier) ?? [];  
  
 return (  
 <SafeAreaView style={{ flex: 1, padding: 16 }}>  
 <Text style={{ fontSize: 20, fontWeight: "700", marginBottom: 8 }}>Grocery CleanScore</Text>  
  
 <View style={{ flexDirection: "row", gap: 8, alignItems: "center" }}>  
 <TextInput  
 value={text}  
 onChangeText={setText}  
 placeholder='e.g., "Oats - steel-cut oats"'  
 style={{ flex: 1, borderWidth: 1, borderColor: "#ccc", padding: 8, borderRadius: 6 }}  
 />  
 <Button title="Add" onPress={addLine} />  
 </View>  
  
 <View style={{ flexDirection: "row", marginTop: 10, gap: 10 }}>  
 <Button title="Upload CSV" onPress={pickCSV} />  
 <Button title="Clear" color="#999" onPress={clearAll} />  
 </View>  
  
 <View style={{ flexDirection: "row", marginTop: 10, gap: 10, alignItems: "center" }}>  
 <Text>Tier1 ≥</Text>  
 <TextInput value={t1} onChangeText={setT1} keyboardType="numeric"  
 style={{ width: 60, borderWidth: 1, borderColor: "#ccc", padding: 6, borderRadius: 6 }} />  
 <Text>Tier2 ≥</Text>  
 <TextInput value={t2} onChangeText={setT2} keyboardType="numeric"  
 style={{ width: 60, borderWidth: 1, borderColor: "#ccc", padding: 6, borderRadius: 6 }} />  
 <Button title={loading ? "Scoring..." : "Score"} onPress={scoreBatch} />  
 </View>  
  
 <Text style={{ marginTop: 16, fontWeight: "700" }}>Items: {items.length}</Text>  
  
 {results && (  
 <>  
 <Text style={{ marginTop: 16, fontSize: 16, fontWeight: "700" }}>Tier 1</Text>  
 <FlatList data={tiered("Tier 1")} keyExtractor={(\_, i) => "t1-"+i} renderItem={renderItem} />  
  
 <Text style={{ marginTop: 16, fontSize: 16, fontWeight: "700" }}>Tier 2</Text>  
 <FlatList data={tiered("Tier 2")} keyExtractor={(\_, i) => "t2-"+i} renderItem={renderItem} />  
  
 <Text style={{ marginTop: 16, fontSize: 16, fontWeight: "700" }}>Tier 3</Text>  
 <FlatList data={tiered("Tier 3")} keyExtractor={(\_, i) => "t3-"+i} renderItem={renderItem} />  
 </>  
 )}  
 </SafeAreaView>  
 );  
}

## **3) How these pieces fit (and why stores will accept it)**

* **FastAPI** is your backend “kitchen” — simple, stateless, and horizontally scalable.
* **Expo app** is a real native client using mobile controls (file picker, inputs), not a WebView wrapper.
* All **CleanScore** logic lives server-side → easier iteration, one source of truth, and avoids app resubmits for rule tweaks.
* Add a **Privacy Policy URL**, **support email**, and ensure you disclose how uploads are handled (transient vs. stored).

## **4) Deployment tips**

* Put FastAPI behind HTTPS (e.g., Cloud Run / ECS / App Service + a managed TLS cert).
* Lock CORS to your mobile domain/bundle ID in production.
* If you later need **offline** scoring, we can ship a **local rules-only mode** in the app for small lists.

If you’d like, I can bundle this into two ready-to-run repos (API + mobile) with README files and a GitHub Action that lints/builds both.