ChatGPT

Open-source barcode scanning libraries and models

Several proven open-source tools exist for barcode detection. For example, **ZXing (“Zebra Crossing”)** is a mature, open-source 1D/2D barcode library (Java-based, with ports to C++ and other languages) that supports common formats (EAN-13, UPC, QR, etc.) [1]. Likewise, **ZBar** is an open-source barcode reader (C99) capable of reading many 1D barcodes [2]. In computer-vision toolkits, OpenCV now includes a `BarcodeDetector` that detects/decodes EAN/UPC codes [3]. (Note: OpenCV outputs energy in kJ by default [4].) On mobile, Google’s **ML Kit** provides an on-device barcode scanner for Android/iOS with high performance (no network needed) [5]. In Flutter specifically, plugins like **mobile_scanner** (using ML Kit on Android, Apple Vision on iOS, ZXing on web) offer real-time camera scanning across platforms [6]. Another Flutter option is **barcode_scan2**, which wraps iOS’s MTBBarcodeScanner and Android’s ZXing library [7].

- *Deep-learning models:* In addition to traditional libraries, research projects have trained neural nets (e.g. YOLOv3/Tiny-YOLO) to detect barcodes in images [8]. For instance, one open-source project uses Tiny YOLOv3 to localize barcodes in an image, then decodes them with PyZbar (ZBar) before querying OpenFoodFacts [9]. However, for most apps the standard libraries (ZXing/ZBar/MLKit) are faster and simpler.

Retrieving products by barcode (OpenFoodFacts)

To link a scanned barcode to a product, the **OpenFoodFacts** database can be used. OpenFoodFacts provides a JSON API where you query by EAN/UPC code. For example, a GET request to `https://world.openfoodfacts.net/api/v2/product/{barcode}` returns the full product data for that code [10]. For instance:

```
GET https://world.openfoodfacts.net/api/v2/product/3017624010701
```

will return all fields for the Nutella product with code 3017624010701 [10]. You can also add a `fields` parameter to limit the returned fields (e.g. `?fields=product_name,nutriments`) [11]. The JSON response includes a `"code"` field (the barcode) and a `"product"` object with details.

- *Offline data:* As an alternative, OpenFoodFacts offers daily data exports (CSV, JSONL, Parquet, or MongoDB dumps) that include barcodes and nutrition facts [12] [13]. Each record has a `code` field (the EAN/UPC barcode) [14]. One could import these into an SQL database and query `SELECT * FROM products WHERE code = ?`. However, using the API is easier for a first implementation and ensures up-to-date data (with the restriction that the API should only be used for real user scans [15]).

Extracting nutrition (calories, protein, carbs, fat)

The API's product JSON contains a `"nutriments"` object with detailed nutrition per 100g (and per serving). Key entries include `energy` (in kJ), `energy-kcal` (in kcal), `fat`, `saturated-fat`, `carbohydrates`, `sugars`, `proteins`, etc. [16] [17]. For example, a sample response snippet shows:

```
"nutriments": {
   "energy": 1221,          // kJ per serving
   "energy_unit": "kJ",
   "fat": 16,              // g per serving
   "carbohydrates": 26,
   "proteins": 11,
   // … plus _100g, _unit, _value fields for each
}
```

In this example, `"fat":16`, `"carbohydrates":26`, `"proteins":11` (all in grams) and `"energy": 1221` (kilojoules) [18]. To get calories, note that OpenFoodFacts returns energy in kJ by default; you can convert by multiplying by 0.239 (e.g. 1221 kJ ≈ 292 kcal) [4]. In practice, you may instead use the provided `energy-kcal` or `energy-kcal_100g` fields directly for calories. By parsing `product.nutriments`, the app can extract the calories, protein, carb, and fat values separately for display.

Implementation plan for Flutter app

1. **Choose a scanner library:** In Flutter, use a package like **mobile_scanner** or **barcode_scan2**. These plugins handle camera integration and real-time decoding on Android and iOS [6] [7]. Configure it to recognize 1D barcodes (EAN/UPC) commonly used on food products. Enable the device flashlight and auto-focus for reliability.
2. **Real-time vs static scanning:** Support both live camera scanning and static image upload. For real-time, start the camera and listen for a successful scan callback (the plugin reports the barcode string). For image uploads, pick or receive an image and feed it to the same decoding engine (some plugins like mobile_scanner provide an `analyzeImage` function, or you can use ZXing/MLKit on the static image). Generally, on-device scanning is fastest; uploading images requires network latency but can catch codes when the camera method fails.
3. **Query the product database:** Once a valid barcode is obtained, call the OpenFoodFacts API. For example, `GET /api/v2/product/{barcode}` [10]. If desired, add `?fields=product_name,nutriments` to reduce payload. Handle the JSON response: check `"status":1` and then access `response.product.nutriments`.
4. **Parse nutrition fields:** Extract `energy-kcal`, `proteins_100g`, `carbohydrates_100g`, `fat_100g` (or the serving values) from the `"nutriments"` object. Convert units as needed (most values are in grams; energy may need conversion to kcal). For robustness, account for cases where data is missing.
5. **Update UI:** Display the retrieved calories, protein, carbs, fats to the user. Optionally, show product name/brand for confirmation. If using an AI calorie estimator as well, you could reconcile or average the values, but barcode-derived values are usually more accurate.

6. **Fallbacks and caching:** Since the API has usage guidelines, consider caching recent lookups locally (e.g. in SQLite) to reduce repeated API calls. In the future, you could also download the CSV/JSON exports and do direct SQL queries (the database includes a `code` column for barcodes [14] and columns like `energy_100g`, `proteins_100g`, etc. as indicated in the data docs [12]).

By following these steps, the Flutter app can quickly decode barcodes on-device and fetch detailed nutrition data via OpenFoodFacts. This hybrid approach leverages open-source scanning libraries for speed and a well-maintained food database API for accuracy [10] [18].

Sources: OpenFoodFacts API documentation [10] [11], OpenFoodFacts developer wiki [4] [12], ZXing project README [1] [2], Flutter package docs [6] [7], and related barcode detection references [5] [8].

---

[1] [2] GitHub - zxing/zxing: ZXing ("Zebra Crossing") barcode scanning library for Java, Android
https://github.com/zxing/zxing

[3] OpenCV: Barcode Recognition
https://docs.opencv.org/4.x/d6/d25/tutorial_barcode_detect_and_decode.html

[4] [14] [16] API/Read/Product - Open Food Facts wiki
https://wiki.openfoodfacts.org/API/Read/Product

[5] Barcode scanning | ML Kit | Google for Developers
https://developers.google.com/ml-kit/vision/barcode-scanning

[6] GitHub - juliansteenbakker/mobile_scanner: A universal Flutter barcode and QR code scanner using CameraX/ML Kit for Android, AVFoundation/Apple Vision for iOS & macOS, and ZXing for web.
https://github.com/juliansteenbakker/mobile_scanner

[7] barcode_scan2 | Flutter package
https://pub.dev/packages/barcode_scan2

[8] [9] GitHub - dchakour/Barcode-detection: Barcode detection with Deep Learning (Tiny Yolo 3)
https://github.com/dchakour/Barcode-detection

[10] [11] Tutorial on using the Open Food Facts API - Product Opener (Open Food Facts Server)
https://openfoodfacts.github.io/openfoodfacts-server/api/tutorial-off-api/

[12] [13] [15] Reusing Open Food Facts Data - Open Food Facts wiki
https://wiki.openfoodfacts.org/Reusing_Open_Food_Facts_Data

[17] [18] API/Full JSON example - Open Food Facts wiki
https://wiki.openfoodfacts.org/API/Full_JSON_example