ChatGPT

Open Food Facts API Endpoints

Open Food Facts (OFF) provides a public API for product lookup and search. The relevant endpoints are:

- **Product lookup:** `GET https://world.openfoodfacts.org/api/v0/product/{barcode}.json` – fetches data for a given barcode.
- **Search (v1, legacy):** `GET https://world.openfoodfacts.org/cgi/search.pl` – an older search interface (returns HTML by default).
- **Search (v2, JSON):** `GET https://world.openfoodfacts.org/api/v2/search` – the newer search API, returning JSON.

These APIs are up-to-date and *not* officially deprecated. The legacy v1 search (`/cgi/search.pl`) is **still supported** "for years to come" alongside the new v2 search [1]. No major outages have been reported recently, and the OFF status page shows normal operation. In practice, issues often arise from missing query parameters or headers, not from endpoint shutdowns.

Below we summarize the current usage, parameters and requirements for each endpoint, plus general constraints (rate limits, headers, etc.), and suggestions for troubleshooting.

`/cgi/search.pl` (Search API v1)

- **Purpose:** Advanced search via query parameters (similar to the website's search form). Returns matching products.
- **URL:** `https://world.openfoodfacts.org/cgi/search.pl` (or `<country>.openfoodfacts.org/cgi/search.pl`).
- **Important parameters:**
- `search_terms=<terms>` – free-text search (e.g. `search_terms=cola`).
- `tagtype_0=<tagfield>&tag_contains_0=<contains>&tag_0=<value>` – advanced tag-based filters (e.g. categories, brands, ingredients). Multiple tag filters can be combined (`tagtype_1`, etc.).
- Always include `action=process` and `search_simple=1` when using `search_terms` (per examples) [2].
- **Format parameter:** Add `json=1` to get JSON output (or `xml=1` for XML). *By default the output is an HTML page,* so JSON=1 is required for machine use [3] [4].
- **Pagination:** You can set `page=<N>` (default 1) and `page_size=<N>` (up to 1000). If omitted, the default page size is 20–24. Example: `page_size=100`.
- **Fields selection:** Use `fields=code,product_name,brands,…` to limit which fields are returned (see examples in [39]). If omitted, OFF returns a full product JSON object for each hit.
- **Response:** A JSON object (when `json=1`) containing at least: `"page"`, `"page_size"`, `"products": […]`, and possibly `"count"` (total hits). Each element in `"products"` is a product object similar to the v0 product format. For example, calling `/cgi/search.pl?search_terms=cola&action=process&search_simple=1&json=1` returns a list of Coca-Cola products. (Without `json=1`, the response is HTML and cannot be parsed as JSON.)
- **Notes:** This endpoint is legacy but still active. It *will not* disappear soon [1]. However, it can be cumbersome, and many new apps use the v2 search API instead.

`/api/v2/search` (Search API v2, JSON)

- **Purpose:** Modern REST search API. Returns JSON results using the same query logic as the website's `/search` endpoint [5]. Supports structured filters and tags.
- **URL:** `GET https://world.openfoodfacts.org/api/v2/search` (or use a country subdomain like `fr.openfoodfacts.org/api/v2/search`).
- **Query parameters:** Uses tag-based parameters (often suffixed with `_tags` and optional language). Examples:
- `code=<barcode>` – restrict to specific barcode(s).
- `<field>_tags=<value>` – filter by tags. Common fields: `categories_tags`, `labels_tags`, `stores_tags`, `ingredients_tags`, `nutrition_grades_tags`, etc. Use the language code suffix if needed (e.g. `categories_tags_en=Chocolate`).
- Numeric filters: e.g. `nutriment_<nutrient>_<unit><op><value>` (e.g. `salt_100g>1` to get products with >1g salt/100g).
- `fields=` – specify which fields to return for each product (comma-separated). Example: `fields=code,product_name,nutrition_grades`. If omitted, the response includes the full product object for each result.
- `page=<N>` and `page_size=<N>` – for paging (defaults to page 1, size 24; max 1000) [6].
- `sort_by=<field>` – sort results (e.g. `last_modified_t`, `product_name`, etc. as supported) [7].
- `no_cache=1` – bypasses OFF's caching of search results (to get the most recent edits).
- **Response:** A JSON object like:

```
{
  "count": <totalHits>,
  "page": <currentPage>,
  "page_size": <pageSize>,
  "page_count": <totalPages>,
  "skip": <offset>,
  "products": [ {…}, {…}, … ]
}
```

The `"products"` array contains product objects (structure like in v0). For example, a request for orange juices with grade C returned 1629 hits (see tutorial [8]).
- **Notes:** This is the recommended search API. It always returns JSON (no extra flag needed) and generally uses more concise parameters than `/cgi/search.pl`. It is relatively new (released 2020) but stable [5].

`/api/v0/product/{barcode}.json` (Product lookup)

- **Purpose:** Retrieves the full data for a single product by barcode.
- **URL:** `GET https://world.openfoodfacts.org/api/v0/product/{barcode}.json` (replace `{barcode}` with the numeric code). Alternatively, you can use `{barcode}.xml` for XML. Example: `GET https://world.openfoodfacts.org/api/v0/product/737628064502.json` [9].
- **Query parameters:** Optional filters can be applied:
- `fields=…` – to limit output to specific fields (e.g. `fields=images` to return only image URLs [10]). This helps reduce bandwidth. By default, all fields for the product are returned.

- `lc=<language>&cc=<country>` – (less common) to override language/country context, but not usually needed.
- **Response:** A JSON object containing at least:

```
{
  "status": 1,
  "status_verbose": "product found",
  "code": "<barcode>",
  "product": { …product fields… }
}
```

If the barcode is not found, the API returns `{"status": 0, "status_verbose": "product not found", "code": "<barcode>"}` [11]. The `"product"` sub-object contains fields like `product_name`, `brands`, `ingredients_text`, `nutriments`, images URLs, etc.
- **Notes:** This endpoint is stable and widely used. No authentication is needed for reading (only a custom User-Agent is required). It's recommended to limit fields if you only need a few values, to speed up responses (e.g. `fields=nutriments,nutrition_grades` to get only nutrition data).

Usage Requirements and Constraints

All OFF read APIs require **no special authentication** beyond a proper User-Agent header. However, there are important usage rules:

- **User-Agent header:** OFF *requires* a custom User-Agent string that identifies your app (name, version, and contact/email). For example:

```
User-Agent: MyApp/1.0 (myapp@example.com)
```

This helps OFF distinguish real apps from bots. Failing to send a User-Agent may get your requests blocked [12] [13].
- **HTTPS and Base URL:** Use the official domains. Production queries should go to `https://world.openfoodfacts.org/...` (or a country-specific subdomain like `fr.openfoodfacts.org`). An older alias `https://ssl-api.openfoodfacts.org` also works but is equivalent to `world.` [14] [15]. (There are separate testing/staging servers like `*.openfoodfacts.net` which require basic auth, but these should be avoided in production.) All endpoints support HTTPS.
- **Rate limits:** To protect the service, OFF enforces limits per IP/user: **100 reads/minute** for product lookups, and **10 queries/minute** for any search (`/cgi/search.pl` or `/api/v2/search`) [16]. If you exceed these rates you may be temporarily blocked. Mobile apps count per user IP. Plan to throttle requests if needed.
- **Response format:** All listed endpoints return JSON (when requested properly). For `/cgi/search.pl`, you must specify `json=1` to get JSON [3]. The newer `/api/v2/search` returns JSON by default. The product API returns JSON when you use the `.json` suffix.
- **CORS:** The public API supports cross-origin GET requests (no special CORS restrictions), so web apps can fetch data directly. (OFF is open data, so `Access-Control-Allow-Origin: *` is typically sent.) However, modern browsers may require valid SSL and correct content-type.

Troubleshooting & Recommendations

If your app sees “Failed to load products” or empty results, check the following:

- **Missing parameters:** Ensure you include all required query parameters. For example, `/cgi/search.pl` **must** have `action=process`, `search_simple=1` (for text search), and `json=1`. An example working URL is:

  ```
  https://world.openfoodfacts.org/cgi/search.pl?
  search_terms=cola&action=process&search_simple=1&json=1
  ```

  Without `json=1`, the API will return HTML and your JSON parser will fail. Similarly, check that barcode calls use the correct path (e.g. `/api/v0/product/12345.json`).
- **Incorrect base URL:** Use `world.openfoodfacts.org` (or a language subdomain) *over* plain `openfoodfacts.org` or other aliases. Do **not** use experimental or old domains. If you are using `openfoodfacts.net`, remember it is the staging server requiring auth.
- **Headers:** Send a valid `User-Agent`. Some frameworks strip default user agents; you may need to set it explicitly. Also set `Accept: application/json` if possible (though `.json` suffix usually suffices).
- **Check rate limits:** If your app makes many queries (e.g. scanning barcodes in a loop), you may exceed the 100/min or 10/min limits. Try slowing requests or caching results locally. A blocked IP will see connection timeouts or HTTP 429 errors. If you suspect rate-limiting, try again after a minute or use the data dumps (CSV/JSONL) for large syncs.
- **Use fields filtering:** If you only need certain data (e.g. product name and nutri-score), use the `fields=` parameter. This reduces response size and can avoid parse errors from unexpectedly large JSON.
- **Alternate endpoints:** If `/cgi/search.pl` is problematic, try the v2 API (`/api/v2/search`) with equivalent parameters (for many filters, the v2 version may be simpler). Both should return matching results, but their parameters differ slightly. For example, filtering category = “Chocolate” is `categories_tags=Chocolate` in v2 versus a tag-based query in CGI.
- **Handling “not found”:** The product API returns `"status": 0` if a barcode is not in OFF. Make sure your code handles that gracefully (not as an error, just no product). Likewise, if search yields no hits, the `"products"` array will be empty.

By following the official documentation for parameter names and respecting the above constraints (custom User-Agent, HTTPS, rate limits), the OFF API should work reliably. In summary: **the endpoints themselves are not broken or removed**, so failures usually stem from client-side issues like missing `json=1`, wrong domain, or headers. Correcting those should restore functionality [5] [3].

Sources: Official Open Food Facts documentation and wiki (API docs) [16] [9] [5] [13], plus OFF community guides [6] [3]. These describe the endpoints, parameters, and usage rules in detail.

[1] [5] Open Food Facts Search API Version 2 - Open Food Facts wiki
https://wiki.openfoodfacts.org/Open_Food_Facts_Search_API_Version_2

[2] [4] [6] API/Read/Search - Open Food Facts wiki
https://wiki.openfoodfacts.org/API/Read/Search

[3] flutter - Beginner: How to combine a SearchView with Openfoodfacts_dart - Stack Overflow
https://stackoverflow.com/questions/74811106/beginner-how-to-combine-a-searchview-with-openfoodfacts-dart

[7] Implement Search V2 API to the fullest · Issue #515 · openfoodfacts/openfoodfacts-dart · GitHub
https://github.com/openfoodfacts/openfoodfacts-dart/issues/515

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

[9] [10] [11] [12] API/Read/Product - Open Food Facts wiki
https://wiki.openfoodfacts.org/API/Read/Product

[13] Open Food Facts Knowledge Base - Are there conditions to use the API?
https://support.openfoodfacts.org/help/en-gb/12-api-data-reuse/94-are-there-conditions-to-use-the-api

[14] [16] Introduction to Open Food Facts API documentation - Product Opener (Open Food Facts Server)
https://openfoodfacts.github.io/openfoodfacts-server/api/

[15] Use new HTTPS api · Issue #112 · openfoodfacts/openfoodfacts-androidapp · GitHub
https://github.com/openfoodfacts/openfoodfacts-androidapp/issues/112