Design/Implementation Notes

1. Error mapping Per Function :

- _owner_repo() :
    1. Raises 500 if GITHUB_OWNER or GITHUB_REPO missing.

- _auth_headers()
    1. Raises 401 if GITHUB_TOKEN missing.

- create_issue()
    1. Delegates rate checks to map_rate_limit().
    2. GitHub 401/403 -> mapped to 401 (detail = r.text).
    3. Any other r.status_code >= 400 -> mapped to 400 (detail = r.text).
    4. On success -> returns r.json().

- list_issues()
    1. Delegates rate checks to map_rate_limit().
    2. If ETag returned and server replies 304 -> raises 304 ("cached").
    3. GitHub 401/403 -> mapped to 401.
    4. GitHub 404 -> mapped to 404 ("not found").
    5. Any other r.status_code >= 400 -> mapped to 400.
    6. On success -> returns (r.json(), headers).

- get_issue()
    1. Delegates rate checks to map_rate_limit().
    2. GitHub 404 -> mapped to 404 ("not found").
    3. GitHub 401/403 -> mapped to 401.
    4. Any other r.status_code >= 400 -> mapped to 400.
    5. On success -> returns r.json().

- update_issue()
    1. Same mapping as get_issue (404 -> 404, 401/403 -> 401, others -> 400).

- create_comment()
    1. Same mapping as get_issue/update_issue (404 -> 404, 401/403 -> 401, others -> 400).

2. Pagination strategy :

- list_issues(state, labels, page, per_page)
    1. Builds params = {"state": state, "page": page, "per_page": per_page} and adds labels if provided.
    2. Uses headers by calling _auth_headers() (includes Authorization and Accept).
    3. If a cached ETag exists for the key, sets If-None-Match header.

Design/Implementation Notes

4. ETag caching :
    a. In-process dict _etag_cache: Dict[Tuple[str, str, int, int], str]
    b. Cache key = (state, labels or "", page, per_page)
    c. On response, if r.headers contains "etag" it stores _etag_cache[key] = etag.
5. Response handling
    a. Calls map_rate_limit(r.headers) and raises its exception if present.
    b. If r.status_code == 304 -> raises HTTPException(status_code=304, detail="cached").
    c. If r.status_code >= 400 -> handles 401/403 -> 401, 404 -> 404, else -> 400 (via HTTPException).
    d. On success -> returns (r.json(), dict(r.headers)) — body plus raw GitHub response headers (so callers can read Link, X-RateLimit-*, etc.).
6. Transport
    a. Uses httpx.AsyncClient(timeout=20) for the request.
7. Implementation facts
    a. Pagination is page-based and forwarded directly to GitHub.
    b. ETag caching is per-process, keyed by (state, labels, page, per_page).
    c. The function returns raw headers alongside the JSON payload.

3. Webhook dedupe :

- Signature verification
    1. Verify X-Hub-Signature-256 (HMAC-SHA256) against WEBHOOK_SECRET before processing.
- Idempotency key
    1. Use X-GitHub-Delivery header as the canonical delivery id (fallback: payload hash).
- File-backed index check
    1. Load events_index.json (or similar index file) and check if delivery id is present.
    2. If present -> treat as duplicate and return 200 immediately.
- Atomic reservation + append
    1. If not present -> atomically mark delivery id as "processing" in the index (write tmp + os.replace or use file lock).
    2. Append full event (or minimal metadata) to events.jsonl (append-only line-delimited JSON).
    3. Update index entry to "processed" with timestamp/offset and persist atomically.
- Failure semantics
    1. On duplicate: 200 (idempotent).
    2. On transient processing failure: return 5xx so GitHub may retry.

Design/Implementation Notes

4. Security trade-offs :

- File-backed events_index.json with no strong locking/GC
    1. Risk: concurrent writes may cause index corruption or duplicate processing (integrity issue more than confidentiality).
- .env secrets on disk for dev
    1. Risk: accidental commit or backup exposure of GITHUB_TOKEN / WEBHOOK_SECRET.
- Returning upstream r.text in HTTPException details
    1. Risk: may leak GitHub error bodies or internal info to clients.