Prompt Pirates — HW2 Design Document

---

1. Error Mapping

Upstream (GitHub)	Our API	Rationale
201 (Created)	201 Created + `Location: /issues/{number}`	Correct REST semantics for creates.
200	200 OK	Happy path.
304 (If-None-Match)	304 Not Modified *(optional extra credit)*	Conditional GET passthrough when ETag caching enabled.
401 / 403	401 Unauthorized	Token missing/invalid or repo access denied; normalize to 401 for clarity.
404	404 Not Found	Missing issue or resource.
422 (Validation)	400 Bad Request	Invalid input (state/title/body); map GitHub's validation error to client.
429 or secondary rate limit	429 Too Many Requests (or 503) + Retry-After	Surface backoff signal; caller can retry later.
5xx	502 Bad Gateway	Upstream fault; our gateway is healthy but GitHub failed.

---

2. Pagination Strategy

Inputs: `state=open|closed|all`, `labels`, `page`, `per_page` (≤100).
Behavior:

- Forward `page` / `per_page` directly to GitHub.
- Clamp `per_page` to 100.
- Preserve GitHub's `Link` header so clients can follow `rel="next"` / `rel="prev"`.

Example:

```
Link: <https://api.github.com/...&page=2>; rel="next",
      <...&page=4>; rel="last"
```

Rationale: Keeps client logic GitHub-compatible and avoids re-encoding pagination math.

---

3. Webhook Dedupe & Handling

- **Events:** `ping`, `issues` (opened/edited/closed/reopened), `issue_comment` (created/edited/deleted).
- **Security:** HMAC SHA-256 over raw body with `WEBHOOK_SECRET`; verify via `hmac.compare_digest`; reject invalid requests with 401; never log raw secrets.
- **Idempotency:** Use `X-GitHub-Delivery` as a unique ID. Maintain in-memory `seen_ids`; if duplicate → respond `204` immediately.
- **Processing:** Validate + record summary, respond `204` quickly; heavy work off-thread if necessary.

- **Debugging:** `/events` endpoint returns last N deliveries, e.g.:

```
[
  { "id": "...", "event": "issue_comment", "action": "created",
    "issue_number": 42, "timestamp": "..." }
]
```

---

4. Security Trade-offs

Chosen:

- Auth via fine-grained PAT (Issues: Read & Write, repo-scoped).
- Env vars (`.env`, ignored by git).
- Secrets masked in logs.
- Webhook HMAC with constant-time compare.
- CORS not broadly opened (service is server-to-server).

Not Implemented (for balance):

- GitHub App (more secure, but more setup required).
- Persistent event store (in-memory buffer suffices).
- Role-based auth (not needed for single-consumer demo).

---

5. Reliability & Rate Limits

- **Backoff:** Inspect `X-RateLimit-Remaining`, `X-RateLimit-Reset`, `Retry-After`. On exhaustion → return `429`/`503` with helpful `detail`, forward `Retry-After`.
- **Timeouts:** `httpx` client defaults used. No auto-retry on writes (to preserve idempotency).
- **Health Check:** `/healthz` returns `{ "status": "ok" }` (used by Docker/Compose).
- **Logs:** Structured, include path & upstream status; secrets always masked.

---

6. Data Models & OpenAPI

Schemas:

- `IssueIn`: `{ title, body?, labels?[] }`
- `IssueOut`: pass-through GitHub fields (`number`, `html_url`, `state`, `labels`, timestamps)
- `IssueUpdate`: `{ title?, body?, state=open|closed }`
- `CommentIn`: `{ body }`
- `CommentOut`: `{ id, body, created_at?, html_url? }`
- `Error`: `{ detail }`

Contract: `openapi.yaml` (v3.1) defines all routes, params, examples, error models, and bearer security scheme.

---

7. Extra Credit

ETag Support:

- Cache ETag/body from GET responses.
- On next request → send `If-None-Match`.
- If GitHub returns 304 → return 304 (or cached 200).
- Saves quota and bandwidth.

Pipeline (GitHub Actions):

- **Stages:**
  1. **Lint** — `flake8` for Python style compliance
  2. **Test** — `pytest` with coverage (≥80%)
  3. **Build** — Docker image build verification
  4. **Security** — Dependency vulnerability scan
- **Triggers:** push to `main`, pull requests.
- **Artifacts:** Coverage reports, Docker image (main branch only).
- **Environment:** Uses repo secrets for `GITHUB_TOKEN` (test scope).

---

8. Known Limitations / Future Work

- Webhook events lost on restart (could add SQLite/Redis).
- No background queue (all inline, though fast).
- PAT rotation is manual (GitHub App would improve).

---

9. Testing Summary

- **Unit Tests (pytest + respx):** happy & negative paths; webhook valid/invalid; pagination header; error mapping.
- **Integration (with `LIVE=1`):** end-to-end flow — create → get → close/open → comment against real repo.
- **Coverage:** ≥80% across `app/` (achieved ~88%).
