GitHub Issues Gateway - Design Note

Shilpa Yelkur Ramakrishnaiah - 019151782 (Team Leader)

Abhishek Darji – 019113471

Aniket Anil Naik – 019107114

Introduction

The GitHub Issues Gateway is designed as a lightweight HTTP service that acts as a proxy around the GitHub REST API for Issues. The purpose is to provide a simplified and consistent interface, while handling authentication, error mapping, webhook validation, and observability concerns.

Error Mapping Strategy

Errors returned by the GitHub API are normalized into clear HTTP responses by the service. Common cases include:
- 401/403 → Unauthorized (invalid or missing token).
- 404 → Not Found (e.g., issue not present).
- 422 → Validation error, mapped to 400 Bad Request with details.
- 5xx → Upstream GitHub errors mapped to 503 Service Unavailable.
- Rate limit exceeded (403 with rate limit headers) → converted into 429 Too Many Requests, with a Retry-After header derived from GitHub's reset time.

Pagination Handling

The /issues endpoint forwards GitHub's Link header for pagination. Clients can navigate through issues using next/prev/last links. Query parameters state, labels, page, and per_page are supported with validation. This ensures consistent behavior with GitHub while maintaining a clean interface.

Webhook Deduplication and Persistence

Incoming webhooks are validated using HMAC SHA-256 signatures with a shared secret. To ensure idempotency, the service constructs a unique key from the GitHub delivery ID, event type, and action. Events are persisted in a lightweight SQLite database for debugging. If the same delivery is redelivered, the INSERT operation is ignored to avoid duplicates.

Security Considerations

Secrets such as GitHub tokens and the webhook secret are never hardcoded. They are loaded from environment variables, which can be managed securely via .env files in development and external secret managers in production. Webhook signatures are verified using constant-time comparison to prevent timing attacks. No sensitive values (like tokens or raw signatures) are logged.

Observability and Reliability

The service includes a /healthz endpoint for container orchestration health checks. Structured logs include request IDs and GitHub delivery IDs for easier correlation. Uvicorn and FastAPI provide robust async request handling. The application design is stateless (aside from SQLite used for debugging webhooks), which aligns with 12-factor application principles.

Trade-offs and Decisions

- SQLite was chosen for webhook persistence because it is simple, lightweight, and sufficient for debugging and classroom use. In a production system, a more robust database or event queue would be appropriate.
- OpenAPI 3.1 is maintained as a separate contract file, rather than auto-generated, to encourage contract-first design. This makes the spec authoritative.
- Error handling prioritizes clarity for API consumers, even if it hides some of GitHub's low-level details.
- The service assumes one repository context, simplifying design but reducing flexibility.

Conclusion

The GitHub Issues Gateway demonstrates key practices in API gateway design: contract-first development, robust error handling, security via HMAC validation, observability, and packaging with Docker. While simplified for a course project, the design choices reflect patterns used in production-grade systems.