REST, Swagger/OpenAPI, and GraphQL — Detailed Guide with Implementations

Overview

This document summarizes best practices and provides runnable examples for: • REST API development (FastAPI) • Swagger/OpenAPI specification and tooling (Express + swagger-jsdoc) • GraphQL implementations (Apollo Server, Graphene)

REST API Development

REST Essentials
• Resource-oriented URIs (/items, /items/{id}), stateless design
• HTTP methods: GET (read), POST (create), PUT (replace), PATCH (partial), DELETE (remove)
• Status codes: 200/201/204 for success, 400/401/403/404/409 for client errors, 5xx for server errors
• Caching & Performance: ETag, Cache-Control, pagination, filtering
• Versioning: /v1/ or Accept headers; backward compatibility encouraged
• Security: TLS, OAuth2/OIDC + JWT, RBAC/ABAC, validation, rate limiting
• Observability: structured logs, correlation ids, tracing, metrics

FastAPI Endpoints (pseudo)
GET /items -> list (auth required, supports skip/limit)
POST /items -> create (201)
GET /items/{id} -> fetch single (404 if not found)
PUT /items/{id} -> replace with If-Match ETag (412 on mismatch)
DELETE /items/{id} -> remove (204)

OpenAPI / Swagger

OpenAPI / Swagger
• Code-first vs Contract-first workflows
• Interactive docs via Swagger UI; JSON at /openapi.json
• Governance: lint with Spectral, contract tests (Dredd/Newman), SDK generation

Useful commands
• Python/FastAPI: pip install fastapi uvicorn pydantic jwt pytest
• Node/Swagger: npm i express swagger-ui-express swagger-jsdoc

GraphQL

GraphQL Highlights
• Single endpoint with schema-defined types, queries, mutations, subscriptions
• Resolvers map to data sources (DB/REST)
• Mitigate N+1 via DataLoader

• Secure with query depth/complexity limits and JWT in context

Sample operations
query One($id: ID!) { item(id: $id) { id name price } }
query All { items { id name price } }
mutation Create($input: ItemIn!) { createItem(input: $input) { id name } }