علامة التبويب 4

# **Zahraah — ملف معياري لبنية وبيئة RESTful API (v1.0)**

**الغرض:** هذا المستند هو المرجع الرسمي الذي يلتزم به فريق التطوير عند تصميم وبناء وتشغيل واجهات RESTful الخاصة بتطبيق *زهراء*. يتضمن مبادئ التصميم، معايير التسمية، الأمن، الاختبارات، التشغيل، و”تعريف الاكتمال" (DoD) لكل Endpoint.

**النطاق:** جميع خدمات الـ API العامة والداخلية التي تُخدم تطبيقات الموبايل، لوحة التحكم، والتكاملات مع الشركاء.

## **0) نظرة معمارية مختصرة (Architecture Overview)**

* **طبقات أساسية:**
  1. **API Gateway / Edge** (Cloudflare/NGINX/Kong): مصادقة أولية، Rate Limit، CORS، مراقبة، توقيع الروابط.
  2. **Auth Service**: OAuth2/JWT، إدارة المستخدمين/الأدوار/الصلاحيات/المفاتيح.
  3. **API Services** (Microservice أو Modular Monolith): موارد Products/Orders/Customers/Payments…
  4. **Data Layer**: قاعدة بيانات رئيسية (SQL)، Cache (Redis)، Queue (RabbitMQ/SQS)، تخزين ملفات (S3-compatible).
  5. **Observability**: Logging/Tracing/Metrics، تنبيهات، لوحات مراقبة.
* **البيئات:** dev → staging → prod مع فصل للشبكات والأسرار والبيانات، وخط إطلاق مُدار عبر CI/CD.

## **1) مبادئ التصميم الأساسية (REST Principles)**

* **Resource-oriented**: كل كيان أعمال هو مورد بــ URI واضح.
* **Stateless**: كل طلب مستقل يحمل الاعتماد والسياق.
* **JSON/UTF-8** افتراضيًا؛ Content-Type: application/json.
* **Backward Compatibility** قدر الإمكان؛ التغييرات الكاسرة عبر إصدار جديد.

## **2) الإصدار والمسارات (Versioning & URLs)**

* الإصدار في المسار: /v1/... (افتراضي).
* صيغة الجمع للمورد: /v1/products, /v1/orders/{id}.
* بدون أفعال في المسار؛ الأفعال عبر HTTP verbs.
* **التسمية:**
  + **المسارات:** kebab-case مثل /order-items.
  + **حقول JSON:** snake\_case مثل created\_at, total\_amount.
* **العلاقات:** /v1/orders/{id}/items، والتوسيع عبر ?expand=.

## **3) أساليب HTTP وحالات الاستجابة (Methods & Status Codes)**

* **Methods**: GET للاستعلام، POST للإنشاء، PUT للاستبدال الكامل، PATCH للتحديث الجزئي، DELETE للحذف.
* **Idempotency**: لـ POST المؤثرة ماليًا عبر Idempotency-Key (فريد لكل محاولة من العميل).
* **Status Codes (مختارة):**
  + نجاح: 200, 201 (+Location), 202 (Async), 204.
  + أخطاء عميل: 400, 401, 403, 404, 409, 422, 429.
  + أخطاء خادم: 500, 502, 503, 504.

## **4) نموذج الردود والأخطاء (Response & Error Shape)**

**شكل الأخطاء الموحّد:**

{

"error": {

"code": "E1001",

"message": "Invalid email format",

"details": [{"field": "email", "issue": "format"}],

"trace\_id": "c9b1f3a0…"

}

}

* إرسال trace\_id أيضًا كهيدر (مثل X-Trace-Id).
* تدوين **كتالوج أكواد الأخطاء** لكل مورد (ملحق أ في آخر المستند).

## **5) التصفية والفرز والتقسيم (Filtering/Sorting/Pagination)**

* **Pagination (Cursor-based افتراضيًا):**
  + الطلب: GET /v1/products?limit=50&cursor=abc
  + الرد: { "data": [...], "page": { "next\_cursor": "def", "limit": 50 } }
  + حد أقصى limit = 100.
* **تصفية:** ?filter[status]=active&filter[category]=dresses.
* **فرز:** ?sort=-created\_at,price (السالب للترتيب التنازلي).
* **اختيار الحقول:** ?fields=sku,name,price.
* **توسيع العلاقات:** ?expand=category,images.

## **6) التحقق من البيانات والعقود (Validation & Schemas)**

* تعريف **JSON Schemas** لكل: إنشاء، تحديث، عرض.
* التواريخ بتنسيق **ISO-8601 UTC** (created\_at, updated\_at).
* القيم المالية كسلاسل أو Decimal لتجنب مشاكل الفاصلة العائمة.
* رفض أي مفاتيح غير معرّفة (strict schemas) وتطبيع القيم.

## **7) المصادقة والتخويل (AuthN/Z)**

* **OAuth2/JWT**: Authorization: Bearer <token>.
* **Scopes** صريحة: orders.read, orders.write, products.read…
* **RBAC**: أدوار وصلاحيات على مستوى الموارد والسجلات (عند الحاجة).
* مفاتيح شركاء **API Keys** مع حدود استخدام منفصلة.

## **8) الأمان (Security)**

* TLS إجباري؛ رفض HTTP العاري على prod.
* **CORS** بنهج allowlist للمصادر الموثوقة.
* إدارة الأسرار عبر Secret Manager (وليس في الكود/Env المكشوفة).
* حماية من إعادة الإرسال (Replay) عبر Idempotency-Key + مهلة صلاحية.
* **Rate Limit** وإرجاع هيدرز قياسية: RateLimit-Limit, RateLimit-Remaining, RateLimit-Reset.
* **ETag/If-Match** لمنع Lost Updates (قفل تفاؤلي).
* الالتزام بـ **OWASP API Security Top 10** (ملحق ب: قائمة تحقق).
* حجب/تنقية المدخلات ومنع حقن SQL/NoSQL.

## **9) التخزين المؤقت والتوافق (Caching)**

* Cache-Control, ETag, Last-Modified للقرّاء العامين.
* دعم 304 Not Modified.
* طبقة Edge (Cloudflare) للموارد القابلة للتخزين المؤقت بعناية.

## **10) الملفات والروابط الموقّتة (Uploads/Downloads)**

* رفع عبر POST /v1/uploads (multipart/form-data).
* الرد يحتوي file\_id, mime\_type, وحجم الملف.
* تنزيل عبر Signed URLs بصلاحيات ووقت صلاحية.

## **11) العمليات غير المتزامنة (Async Jobs)**

* عند طول التنفيذ: 202 Accepted + Location: /v1/jobs/{id}.
* مورد Job يعرض: status, progress, result, error.

## **12) Webhooks (للتكامل مع الشركاء)**

* توقيع كل webhook (HMAC-SHA256) مع X-Signature + X-Timestamp.
* إعادة المحاولة مع Backoff وتجنّب التكرار عبر event\_id.
* إدارة الاشتراكات وأنواع الأحداث (مثال: order.created, order.canceled).

## **13) الملاحظة والقياس (Observability)**

* **Structured Logging (JSON)**: مستوى، خدمة، endpoint، مدة، status، trace\_id, user\_id.
* **Tracing** متوافق W3C (traceparent)، ربط عبر Gateway.
* **Metrics**: p50/p95 latency، معدّل الأخطاء، QPS، 4xx/5xx، 429.
* **SLOs مبدئية**: قراءات p95 ≤ 300ms؛ كتابات p95 ≤ 700ms؛ توافر شهري ≥ 99.9%.

## **14) البيانات والمعاملات (Data & Persistence)**

* Migrations مُدارة (أدوات مثل Flyway/Liquibase) بجداول schema\_migrations.
* معاملات (Transactions) حول العمليات متعددة السجلات.
* **قفل تفاؤلي** بحقل نسخة version أو عبر ETag.
* جداول **Audit** للتغييرات الحساسة (من؟ ماذا؟ متى؟ قبل/بعد).

## **15) الاختبارات (Testing Policy)**

* **وحدة**: منطق الأعمال.
* **تكامل**: طبقة HTTP والـ DB.
* **عقود** (Contract) مقابل OpenAPI 3.1.
* **أداء/تحميل**: k6/JMeter؛ أهداف p95 أعلاه.
* **أمن**: SAST/DAST + فحوص ZAP/OWASP؛ منع التسريبات.
* بيئات dev/staging ببيانات وهمية؛ **Smoke Tests** بعد كل نشر.

## **16) التوثيق وتغيير الإصدارات (Docs & Change Management)**

* **OpenAPI 3.1** هو مصدر الحقيقة.
* أمثلة طلب/رد لكل Endpoint وحالات الأخطاء.
* **Changelog** واضح؛ إعلانات Deprecation قبل 90 يومًا على الأقل.
* Swagger UI/Redoc للوصول الداخلي والخارجي.

## **17) CI/CD لواجهات الـ API**

* **خط الأنابيب (Pipeline) — كحد أدنى:**
  1. Build & Lint & Unit Tests (فشل يعني إيقاف).
  2. SAST + فحص الأسرار (gitleaks) + Dependency Scan.
  3. Contract Tests مقابل OpenAPI.
  4. نشر إلى staging + Smoke/Integration.
  5. Performance Sanity (عينات).
  6. موافقة يدوية مبررة (إن لزم) → نشر إلى prod.
* استرجاع سريع (Rollback) عبر إصدارات مُمكنة التتبع.

## **18) Non-Functional Requirements (NFRs)**

* الأداء، التوافر، القابلية للتوسع، الأمان، الامتثال.
* **Timeouts** و**Circuit Breakers** عند استدعاء خدمات خارجية.
* **Backpressure/Retry** بسياسات واضحة لتجنب العواصف.

## **19) بوابة واجهات/API Gateway (Edge)**

* توحيد المصادقة، الـ Rate Limit، CORS، التتبع، وإصدار المفاتيح.
* القواعد على الحافة (Cloudflare): WAF، Bot Management، TLS، والـ Caching الانتقائي.

## **20) معايير الترميز والتنظيم (Coding & Structure)**

* نمط وحدات/Packages لكل مورد (Orders/Products…)
* فصل Domain/Infra/Transport قدر الإمكان.
* اتفاقيات أسماء الأخطاء، الرسائل، والأكواد.

# **ملاحق (للتنفيذ العملي)**

## **ملحق أ) قالب مواصفة مورد (Endpoint Spec Template)**

* **Resource Name:** Orders
* **Base Path:** /v1/orders
* **Endpoints:**
  + GET /v1/orders — قائمة (Pagination, Filters: status, created\_at\_from/to)
  + POST /v1/orders — إنشاء (Idempotent via Idempotency-Key)
  + GET /v1/orders/{id} — عرض
  + PATCH /v1/orders/{id} — تحديث جزئي + If-Match/ETag
  + DELETE /v1/orders/{id} — حذف (نقاش: soft delete؟)
* **Auth/Scopes:** orders.read, orders.write
* **Request Schemas:** (Create/Update)
* **Response Schemas:** (Item/List)
* **Errors:** E2001 invalid\_item, E2002 stock\_unavailable, …
* **Rate Limits:** Partner 600 rpm, App 60 rpm
* **Webhooks ذات الصلة:** order.created, order.canceled

## **ملحق ب) قائمة تحقق أمنية OWASP (مختصرة)**

## **ملحق ت) الرؤوس القياسية (Common Headers)**

* **Request:** Authorization, Idempotency-Key, If-Match, X-Request-Id.
* **Response:** ETag, RateLimit-\*, X-Trace-Id, Cache-Control, Location.

## **ملحق ث) قالب متغيرات البيئة (Env Template)**

| **المفتاح** | **الوصف** |
| --- | --- |
| DATABASE\_URL | اتصال قاعدة البيانات |
| READ\_REPLICA\_URL | نسخة القراءة (إن وجدت) |
| CACHE\_REDIS\_URL | اتصال Redis |
| QUEUE\_URL | نظام الصفوف |
| STORAGE\_BUCKET | حاوية التخزين S3-compatible |
| JWT\_SECRET/OAUTH\_KEYS | مفاتيح التوقيع |
| API\_GATEWAY\_KEY | مفتاح البوابة |
| LOG\_LEVEL | مستوى التسجيل |
| ALLOWED\_ORIGINS | قائمة CORS |
| WEBHOOK\_SECRET | مفتاح توقيع Webhooks |

## **ملحق ج) تعريف الاكتمال لكل Endpoint (DoD Checklist)**

## **ملحق ح) سياسة إدارة التغيير (Deprecation & Breaking Changes)**

* إعلان واضح في الـ Changelog + هيدر Deprecation وLink: rel="deprecation".
* مدة إشعار لا تقل عن 90 يومًا قبل إزالة سلوك/حقل.
* إتاحة الإصدار الجديد /v2 بالتوازي خلال فترة الانتقال.

## **ملحق خ) أمثلة عملية مختصرة**

**إنشاء طلب (idempotent):**

POST /v1/orders HTTP/1.1

Authorization: Bearer <token>

Idempotency-Key: 9b38c8d1-…

Content-Type: application/json

{

"customer\_id": "cus\_123",

"items": [{"sku": "SKU-1001", "qty": 2}],

"payment\_method": "cod"

}

**رد نجاح:**

{

"id": "ord\_789",

"status": "pending",

"total\_amount": "154.00",

"currency": "YER",

"created\_at": "2025-08-31T18:00:00Z"

}

### **ملاحظات تشغيلية (Runbook مختصر)**

* **الصحة:** GET /health (جاهزية/صلاحية)، GET /metrics (Prometheus).
* **الدارات:** مهلات خارجية واضحة + Circuit Breaker للخدمات الحرجة.
* **النسخ الاحتياطي/استعادة:** سياسات يومية + اختبارات استعادة دورية.

**هذا المستند معتمد كمعيار أولي v1.0.** أي إضافة أو تعديل يجب أن يمر عبر مراجعة تقنية ويتم تسجيله في الـ Changelog.

علامة التبويب 5

# **1) الفكرة باختصار**

* **API Gateway**: طبقة على الحافة أمام الـAPI توحّد الدخول، الأمان، المعدّلات، التحويلات، الكاش، والمراقبة.
* **OpenAPI (OAS 3.x)**: **عقد** (Specification) يُعرِّف *بالضبط* كيف تتكلّم واجهاتك: المسارات، المدخلات، المخرجات، الأخطاء، الأمان.

القاعدة الذهبية: **OpenAPI هو مصدر الحقيقة**، و**Gateway يفرض هذا العقد ويحميه**.

# **2) لماذا نحتاجهما؟**

* **أمان** موحّد (JWT/API Keys/mTLS، Rate Limiting، WAF).
* **حوكمة** وتوافق عكسي (Versioning/Deprecation).
* **تجربة مطوّر** (بوابة توثيق، مفاتيح شركاء، SDKs).
* **تحكّم بالأداء والتكلفة** (Cache، ضغوط، سياسات).
* **قابلية الملاحظة** (Logs/Metrics/Tracing موحّدة).

# **3) أين يُوضَع الـGateway؟ (مرجعية)**

* على نطاق: https://api.zahraah.com أمام Laravel (مونوليث/خدمات).
* خلفه: **Cloudflare/WAF** ثم **Gateway** (Kong/NGINX/OpenResty/Cloudflare Gateway/AWS API GW… اختر ما يناسبكم) → **Laravel API** → **MySQL/Redis/S3**.
* داخليًا: يمكن فصل **بوابة شركاء** على نطاق فرعي آخر (مثلاً partners.api…) بمفاتيح وسCOPEs مختلفة.

# **4) ماذا يفعل الـGateway فعليًا؟**

**أ. أمن وولادة الطلب**

* TLS و**Full (Strict)**، **mTLS** اختياري لشركاء حسّاسين.
* **مصادقة**: Bearer JWT / OAuth2 (Client Credentials) / API Key للشركاء.
* **إجازة/صلاحيات (Scopes/RBAC)** على المسارات.
* **Rate Limiting/Quotas** لكل IP/مستخدم/مفتاح (اندماج مع ما طبّقناه).
* **CORS** موحّد، **Request Size Limits**، **IP Allowlist** لمسارات معيّنة.

**ب. توجيه وتهيئة**

* Routing بحسب **المسار/الإصدار** (/v1/...).
* **تحويلات** (Request/Response Transform): رؤوس قياسية، إخفاء رؤوس حسّاسة، تطبيع أخطاء.
* **حقن تتبّع**: X-Request-Id, W3C traceparent.

**ج. أداء وكاش**

* **Caching** لـ GET العامة وفق Cache-Control.
* ضغط Gzip/Brotli للـ JSON.
* **Circuit Breaker/Timeouts/Retry** لمسارات داخلية محددة.

**د. مراقبة**

* **Logs موحّدة** (JSON) لكل طلب.
* **Metrics**: RPS, Latency P95/P99, 5xx%, 4xx%, Hit Ratio.
* **Tracing** موزّع.

# **5) OpenAPI — كيف نبنيه ونستخدمه؟**

**ماذا يصف؟**

* info/servers/tags، **paths** (المدخلات/المخرجات/الأخطاء)، **components/schemas** (نماذج البيانات)، **securitySchemes**، **webhooks** (إن لديكم أحداث واردة).

**أسلوب العمل**

* إمّا **Spec-first**: نكتب الـOAS ثم نولّد Stubs/SDKs.
* أو **Code-first**: نولّده من التعليقات، *لكن* نُراجعه ونُلَمِّطه (Lint) وننشره.

**التوافق العكسي**

* إضافات **غير كاسرة**: إضافة حقل اختياري/مسار جديد.
* تغييرات **كاسرة**: إزالة/تعديل نوع/إلزام حقل → **إصدار جديد /v2** + سياسة Deprecation.

**التوثيق والبوابة**

* تقديم **Swagger UI/Redoc** للمطوّرين + أمثلة + Sandbox + مفاتيح.

# **6) حوكمة الأسلوب (Style Guide مختصر)**

* **المسارات**: أسماء موارد جمع: /products, /orders/{id}.
* **الأفعال**: GET/POST/PUT/PATCH/DELETE وفق REST.
* **الترقيم**: **Cursor Pagination** (?cursor=&limit=).
* **الفرز/الفِلتر**: ?sort=-price&filter[brand]=....
* **الأسماء**: JSON **snake\_case** أو **camelCase**—اختاروا واحدًا وثبّتوه.
* **الأخطاء**: RFC 7807 (application/problem+json) أو كائن موحّد { error: { code, message, details } }.
* **الأمان**: securitySchemes واضحة (Bearer JWT، apiKey، oauth2).
* **النسخ**: /v1 في المسار + رؤوس Deprecation/Sunset عند إنهاء مسار.

# **7) سير العمل (Workflow) المقترح**

1. **تصميم العقد** (OpenAPI) + أمثلة/حالات حافة.
2. **Lint** (أدوات مثل Spectral) + **فحص تغييرات كاسرة** (oasdiff) في CI.
3. **نشر التوثيق** (بوابة المطوّرين).
4. **بوابة المفاتيح**: إنشاء API keys/OAuth Clients مع Scopes.
5. **Gateway تحديث**: ربط المسارات، سياسات الأمن/الحدود.
6. **اختبارات عقد** (Contract Tests) ضد الـOAS قبل الدمج.
7. **مراقبة/قياس** بعد الإطلاق + **Deprecation** وفق خطة عند تغييرات كاسرة.

# **8) أمثلة سريعة**

## **8.1 مقتطف OpenAPI 3.1 (YAML)**

openapi: 3.1.0

info:

title: Zahraah Public API

version: 1.0.0

servers:

- url: https://api.zahraah.com/v1

tags:

- name: products

components:

securitySchemes:

bearerAuth:

type: http

scheme: bearer

bearerFormat: JWT

schemas:

Product:

type: object

required: [id, name, price\_cents, stock\_status]

properties:

id: { type: integer, format: int64 }

name: { type: string }

brand\_id: { type: integer }

category\_id: { type: integer }

price\_cents: { type: integer }

price\_after\_discount\_cents: { type: integer, nullable: true }

stock\_status: { type: integer, description: "0 out,1 low,2 in" }

rating\_avg: { type: number, format: float }

rating\_count: { type: integer }

security:

- bearerAuth: []

paths:

/products:

get:

summary: List products

tags: [products]

parameters:

- in: query

name: cursor

schema: { type: string }

- in: query

name: limit

schema: { type: integer, minimum: 1, maximum: 100, default: 20 }

- in: query

name: filter[brand]

schema: { type: integer }

- in: query

name: sort

schema: { type: string, enum: ["price","-price","rating","-rating"] }

responses:

"200":

description: ok

content:

application/json:

schema:

type: object

properties:

data:

type: array

items: { $ref: "#/components/schemas/Product" }

meta:

type: object

properties:

next\_cursor: { type: string, nullable: true }

"401":

description: unauthorized

## **8.2 سياسات Gateway (مثال تصوري مبسّط)**

* **Auth**: Require bearerAuth على /v1/\* (باستثناء /v1/auth/\*).
* **Rate Limit**: /v1/products = 120 req/min/IP، /v1/search = 30 req/min/User.
* **Headers**: حقن X-Request-Id إذا لم يُرسَل، وتمرير traceparent.
* **Transform**: إخفاء X-Powered-By، توحيد أخطاء 4xx/5xx إلى JSON موحّد.
* **Cache**: GET /v1/products **مسموح بالكاش** حسب Cache-Control من الأصل.
* **CORS**: أصول محدّدة (app.zahraah.com) وأساليب/رؤوس مقيّدة.

## **8.3 Laravel + OAS (ملاحظة تنفيذية)**

* استخدم تعليقات/Attributes لتوليد الـOAS، ثم **Lint** و**نشر** ملف openapi.yaml مع الإصدار.
* فعّل Middleware يقرأ الـOAS للتحقق **Validation** (اختياري) في بيئات Staging.

# **9) أمن متقدّم**

* **OAuth2 Client Credentials** لشركاء B2B مع **Scopes** (read:products, write:orders).
* **mTLS** لواجهات داخلية.
* **قيود حجم/زمن**: max\_body\_size, read\_timeout, send\_timeout.
* **HMAC Signatures** لمسارات حسّاسة (بدون جلسة).
* **Anti-replay** عبر طابع زمني وتوقيع.
* **Threat Modeling** دوري (OWASP API Top 10).

# **10) قياس النجاح (KPIs)**

* **Availability (شهر)**: ≥ 99.9% للـAPI العامة.
* **P95 Latency (GET شائعة)**: < 300ms.
* **5xx%**: < 0.3%.
* **Docs Health**: درجة Lint ≥ 90/100، وقت تحديث الوثائق مع الكود ≤ 24h.
* **Onboarding شريك**: إصدار مفتاح/عميل + وصول ناجح ≤ 2 أيام.
* **Contract Tests Pass Rate**: ≥ 99%.

# **11) قوائم تحقق (Ready-to-use)**

**11.1 إضافة Endpoint جديد**

* تعريف في **OpenAPI** (مسار، بارامترات، مخطط، أمثلة، أخطاء).
* **Lint + oasdiff** (لا تغييرات كاسرة بلا /v2).
* اختبارات عقد + وحدة/تكامل.
* تحديث **Gateway**: Auth/Rate/CORS/Cache/Transform.
* نشر التوثيق + أمثلة + SDKs (إن لزم).

**11.2 إصدار جديد (Breaking)**

* إعلان /v2 + خطة **Deprecation** لـ/v1 (Headers: Deprecation, Sunset).
* فترة تداخل وتشغيل كلا الإصدارين.
* أدوات هجرة وأمثلة للعمّال/الشركاء.
* إيقاف /v1 بعد المدة المتفق عليها.

**11.3 شريك جديد**

* إنشاء **API Key/OAuth Client** + Scopes.
* تطبيق Rate/Quota مخصّص.
* توفير Sandbox + أمثلة Postman/SDK.
* مراقبة الاستهلاك والشذوذ.

## **خلاصة**

* **OpenAPI** يحدّد *كيف* نتواصل (عقد دقيق، موثّق، قابل للفحص).
* **API Gateway** يضمن *الالتزام والأمان والأداء* لهذا العقد (مصادقة، حدود، كاش، مراقبة).
* اعتمدوا **OAS كمصدر الحقيقة**، واربِطوه بخط CI/CD (Lint/Breaking-check)، واسمحوا للـGateway بفرض السياسات—بهذا تحصلون على واجهات قابلة للتوسع، آمنة، سهلة التكامل، ومع تجربة رائعة للمطوّرين.

علامة التبويب 3

# **1) الأهداف (Objectives)**

* **ثبات العقد**: واجهات واضحة لا تتغير دون خطة ترحيل.
* **الأمن أولًا**: مصادقة قوية، صلاحيات دقيقة، حماية من الهجمات.
* **قابلية التوسّع**: خدمات عديمة الحالة + تخزين/كاش + طوابير.
* **جاهزية التكامل**: مفاتيح/Scopes، Webhooks، Sandbox، وثائق حيّة.

# **2) الإصدار والتوافق (Versioning & Compatibility)**

* المسار: /v1/...، ولا يُزال أي حقل قبل دورة ترحيل معلنة.
* **إضافة حقول جديدة مسموح** دون كسر التوافق.
* سياسة إيقاف: إعلان ≥ 90 يومًا، ثم إغلاق تدريجي.

# **3) تصميم (REST (Resources & Conventions**

* **تسمية الموارد** بصيغة جمع: /v1/products, /v1/orders.
* **طرق HTTP**:
  + GET قراءة، POST إنشاء، PATCH تعديل جزئي (JSON Merge Patch)، PUT استبدال، DELETE حذف.
* **الحالة**: استخدم رموزًا قياسية (200, 201, 204, 400, 401, 403, 404, 409, 422, 429, 500).
* **الترقيم/الفرز/الفلترة**:
  + ?page=1&per\_page=50
  + ?sort=-created\_at,price (سالب = تنازلي)
  + فلاتر صريحة: ?status=active&category\_id=...&price\_min=...&price\_max=...
* **الإرجاع**: JSON UTF-8، الطوابع الزمنية ISO-8601 UTC (2025-08-27T17:00:00Z).
* **الحقول**: snake\_case للمفاتيح. استخدم fields= للانتقاء الجزئي: ?fields=id,name,price.
* **الارتباطات**: روابط HATEOAS اختيارية (self, next, prev).

### **قالب استجابة ناجحة**

{

"data": {...},

"meta": {"page":1,"per\_page":50,"total":132},

"links": {"self":"...","next":"..."}

}

### **قالب خطأ موحّد**

{

"error": {

"code": "VALIDATION\_ERROR",

"message": "price must be >= 0",

"details": [{"field":"price","rule":"min","value":-1}],

"trace\_id": "req\_7f3c..."

}

}

# **4) الهوية والصلاحيات (AuthN/Z)**

* **الموبايل/الويب**: JWT قصير العمر (15–30 دقيقة) + **Refresh token** آمن.
* **التكاملات (Server-to-Server)**:
  + **OAuth 2.1 Client Credentials** مع **Scopes** (مثال: orders:read, orders:write).
  + أو **HMAC Signature** للطلبات الحساسة (توقيع X-Signature على method+path+timestamp+body مع X-Timestamp؛ رفض إن > 5 دقائق).
* **RBAC/ABAC**: أدوار (admin, ops, support) + سياسات على مستوى الحقول/السجلات.
* **تخزين الأسرار**: في Secret Manager/ENV فقط، لا تُضمّن في الكود/العميل.
* **CORS**: قائمة بيضاء للمجالات الموثوقة، وHeaders لازمة فقط.

# **5) الأمان (Security Hardening)**

* TLS 1.2+ فقط.
* **معدلات**: Rate Limit افتراضي (مثال: 100 req/min لكل مفتاح) مع رؤوس:
  + X-RateLimit-Limit, X-RateLimit-Remaining, X-RateLimit-Reset.
* **حماية إعادة الإرسال**: رأس Idempotency-Key في أي POST قابل للتكرار (إنشاء طلبات، دفع COD، …)، يُخزَّن 24 ساعة.
* **التحقق من المُدخلات**: طول، نوع، نطاق؛ رفض أي حقل غير معروف (strict mode لبوابة API).
* **الملفات**: فحص نوع/حجم، تخزين على S3/Cloudinary، روابط موقَّعة قصيرة العمر.
* **الرؤوس الأمنية**: Content-Security-Policy, X-Content-Type-Options: nosniff.
* **التدقيق**: سجلات JSON مُهيكلة بلا PII، مع **trace\_id**/X-Request-Id.
* **Web Application Firewall** (Cloudflare WAF) + حظر/Rate Limit لمصادر مشبوهة.

# **6) الأداء والـCaching**

* **عديم الحالة** (Stateless) + تدرّج أفقي.
* **كاش**:
  + Responses قابلة للتخزين: Cache-Control, ETag/If-None-Match.
  + Redis لقراءات ثقيلة (منتجات، فئات) مع TTL.
* **SLO**: زمن استجابة p95 < 300ms للقراءة، < 800ms للكتابة.
* **صفوف/وظائف غير متزامنة**: للعمليات الثقيلة (إرسال بريد/واتساب، تسعير، تقارير).

# **7) قابلية التوسّع والموثوقية**

* **Outbox & Retry** لنشر أحداث مؤكدة عبر MQ (SQS/RabbitMQ/Kafka).
* **Circuit Breaker + Exponential Backoff** عند استدعاء أطراف ثالثة.
* **Bulk & Batching**: دعم طلبات مجمعة عند الحاجة (/bulk).
* **Pagination فقط** للقراءات الكثيفة؛ لا تعيد كل البيانات دفعة واحدة.
* **Soft Deletes** مع حقول deleted\_at لتجنّب فقدان دائم للبيانات.

# **8) نمط البيانات (Schema Standards)**

* مفاتيح: id كسلسلة غير متوقّعة (UUID/ULID).
* مبالغ مالية: **أصغر وحدة** (فلس/هللة) كأعداد صحيحة + حقل currency (ISO-4217).
* الألوان/المقاسات/السمات: جداول مرجعية؛ لا تجعلها نصًا حرًا بدون ضبط.
* الطوابع الزمنية: created\_at, updated\_at, (وstatus\_changed\_at للطلبات).
* التوطين: name\_ar, name\_en إن لزم، أو بنية translations.

# **9) طبقة التكامل (3rd-Party Readiness)**

* **API Keys** لكل شريك، صلاحيات بنطاقات (Scopes) دقيقة.
* **Sandbox** منفصل /sandbox ببيانات مزيّفة.
* **Webhooks** ثابتة + توقيع:
  + ترويس: X-Webhook-Signature, X-Webhook-Id, X-Webhook-Timestamp.
  + **إعادة المحاولة**: Backoff حتى 24 ساعة؛ كل حدث ذو event\_id لمنع الازدواج.
  + **أحداث موصى بها**:
    - order.created, order.cod\_confirmed, order.shipped, order.delivered, order.returned
    - stock.adjusted, product.updated, customer.created
* **توثيق حي**: OpenAPI 3.1 + Portal (أمثلة + curl + SDKs).
* **اتفاقية استخدام** + حدود + سياسة بيانات + SLA.

### **مثال Webhook (جسم الرسالة)**

{

"id": "evt\_01J...9X",

"type": "order.cod\_confirmed",

"created\_at": "2025-08-27T17:10:02Z",

"data": {

"order\_id": "ZHR-000123",

"customer\_id": "CUS-8ab1...",

"cod\_risk\_score": 0.37,

"confirmed\_by": "otp"

}

}

# **10) واجهات أساسية (نطاق زهراء)**

* **Auth**: /v1/auth/login, /v1/auth/refresh, /v1/auth/logout.
* **Users/Addresses**: /v1/users, /v1/addresses.
* **Catalog**: /v1/products, /v1/categories, /v1/attributes, /v1/media.
* **Cart/Checkout**: /v1/carts, /v1/carts/{id}/items, /v1/checkout.
* **Orders/Payments/COD**: /v1/orders, /v1/orders/{id}/confirm\_cod, /v1/payments.
* **Shipping**: /v1/shipments, /v1/shipments/{id}/events.
* **Returns**: /v1/returns.
* **Coupons**: /v1/coupons/apply.
* **Webhooks (config)**: /v1/webhooks/endpoints.

# **11) نماذج مهمّة**

### **PATCH (Merge Patch) لتعديل جزئي**

PATCH /v1/orders/ZHR-000123

Content-Type: application/merge-patch+json

{

"shipping\_address\_id": "ADDR-9d3...",

"note": "اتصال قبل التوصيل"

}

### **Idempotency-Key**

POST /v1/orders

Idempotency-Key: c5a8bd76-...

### **رؤوس معدل الاستهلاك**

X-RateLimit-Limit: 100

X-RateLimit-Remaining: 72

X-RateLimit-Reset: 1693152000

# **12) التتبع والملاحظة (Observability)**

* **Logging**: JSON، بدون PII؛ تضمين trace\_id, span\_id, user\_id?.
* **Metrics**: طلبات/ثانية، أخطاء حسب الرمز، زمن p50/p95، مهلات، إعادة محاولات.
* **Tracing**: OpenTelemetry بين الخدمات.
* **تنبيهات**: 5xx تتجاوز عتبة، 429 غير طبيعي، ارتفاع زمن p95.

# **13) الاختبارات والجودة**

* **Contract Tests** من OpenAPI (Prism/Stoplight).
* **Integration & E2E** (Postman/Newman أو K6).
* **Security Tests**: فحص OWASP API Top 10 دوريًا.
* **Smoke on deploy**: Healthcheck إلزامي قبل التحويل إلى الإنتاج.
* **بيئات**: dev → staging → prod، مع بيانات مزيفة في staging.

# **14) السياسات التشغيلية**

* **Health**: /health (جاهزية/حياة) + /metrics.
* **الترخيص والخصوصية**: DPA مع الشركاء، حذف بيانات العميل عند الطلب.
* **سجل التغييرات**: changelog عام، وتبعيات/Breaking Changes معلّمة بوضوح.
* **النماذج/المفاتيح**: تدوير المفاتيح كل 90–180 يومًا.

# **15) قوائم تدقيق (مختصرة)**

**قبل نشر أي إصدار API:**

* تحديث OpenAPI + أمثلة اختبارية.
* تغطية اختبارات ≥ النسبة المستهدفة.
* مراجعة أمنية (Secrets, CORS, Rate Limit, Validation).
* مراقبة مفعّلة (Logs/Metrics/Tracing).
* خطة تراجع (Rollback) جاهزة.

علامة التبويب 2

# **1) وظيفي (GUI/CLI بلا كود)**

* **Postman Collection Runner** (واجهة رسومية).

**Newman** (سطر أوامر لتشغيل Postman في الـCI):  
newman run Zahraah.postman\_collection.json -e staging.postman\_environment.json \

--reporters cli,html --reporter-html-export reports/postman.html

**Insomnia + inso (CLI)**:  
inso run test --env "staging" # يشغّل اختبارات Insomnia مباشرة

# **2) مطابقة المواصفة (Contract) من ملف OpenAPI**

**Dredd** (يقارن الاستجابات مع الـOpenAPI):  
npx dredd openapi.yaml https://api.staging.example.com

**Schemathesis** (توليد حالات واكتشاف حواف):  
pipx install schemathesis

schemathesis run openapi.yaml --base-url=https://api.staging.example.com --checks all

# **3) الأمن (DAST) للـAPI**

**OWASP ZAP – API Scan**:  
docker run -t owasp/zap2docker-stable zap-api-scan.py \

-t https://api.staging.example.com/openapi.yaml -f openapi -r zap-report.html

* (اختياري) **Burp Suite** للاختبار التفاعلي اليدوي المتقدّم.

# **4) الأداء/الضغط (Load & Performance)**

**k6** (خفيف وسهل الدمج):  
k6 run -e TOKEN=$TOKEN checkout.js

* *(استخدم سكربت checkout.js الذي يحاكي تصفّح → إضافة للسلة → إنشاء طلب)*

**Artillery** (أمر واحد سريع):  
npm i -g artillery

artillery quick --count 20 --num 10 https://api.staging.example.com/v1/products

# **5) التصفية/القواعد/الجودة للـOpenAPI**

**Spectral** (Lint للمواصفة):  
npx @stoplight/spectral lint openapi.yaml

# **6) محاكاة وProxy للتحقق/العزل (Mock/Proxy)**

**Prism** (Proxy يتحقق من التوافق مع OpenAPI أثناء مرور الطلبات):  
npx @stoplight/prism proxy openapi.yaml --port 4010

# وجّه العميل عبر http://localhost:4010 ليرصد/يحقق الطلبات

**WireMock** (Mock سريع عبر Docker):  
docker run -p 8080:8080 -v $(pwd)/mappings:/home/wiremock/mappings wiremock/wiremock

## **“باقة سريعة” تُشغَّل اليوم (مقترحة)**

1. **وظيفي:**newman run collection.json -e staging.json --reporters cli,html
2. **تعاقد:**npx dredd openapi.yaml https://api.staging.example.com
3. **أمني:**ZAP سطر الأوامر أعلاه ويُحفظ zap-report.html
4. **أداء (دخاني):**artillery quick --count 10 --num 5 https://api.staging.example.com/v1/products
5. **جودة مواصفة:**npx @stoplight/spectral lint openapi.yaml

## **إدماج مختصر داخل CI (مراحل)**

* **test:functional** → Newman
* **test:contract** → Dredd/Schemathesis
* **test:security** → ZAP baseline
* **test:perf-smoke** → Artillery/k6 خفيف
* **lint:openapi** → Spectral  
  (اجعل فشل أي مرحلة يوقف الـPipeline.)

علامة التبويب 1

# **1) التحضير**

* المطلوب: تفعيل بيئة **Staging** ببيانات اختبار وحسابات أدوار (عميل/مشرف/مندوب)، وفصل المفاتيح واللوغ.
* المخرجات: رابط بيئة Staging + حسابات اختبار + ملف متغيرات البيئة.
* معايير القبول: لا أسرار في الكود، كل طلبات الاختبار تمر عبر Staging فقط.

# **2) المواصفة (OpenAPI)**

* المطلوب: توحيد وتحديث ملف **OpenAPI** لكل المسارات مع أمثلة واقعية.
* المخرجات: ملف openapi محدث + صفحة Swagger قابلة للعرض.
* القبول: لا اختلافات بين الواقع والمواصفة في الأسماء والأنواع والأكواد.

# **3) الفحص الدخاني (Smoke)**

* المطلوب: التحقق السريع من الموارد الأساسية (صحة/نسخة، منتجات، سلة، طلبات، مصادقة).
* المخرجات: تقرير مختصر (حالة كل مسار وزمن الاستجابة).
* القبول: جميعها ترجع 2xx، وزمن القراءة معقول، والأخطاء مهيكلة بنفس الصيغة.

# **4) اختبار التعاقد (Contract)**

* المطلوب: تشغيل أداة مقارنة التنفيذ بالمواصفة (مثل Dredd/Schemathesis) وإصلاح الفروقات.
* المخرجات: تقرير نتائج + قائمة الفروقات وخطة الإغلاق.
* القبول: صفر فروقات حرجة، والفروقات الطفيفة مغلقة أو مبررة.

# **5) الاختبارات الوظيفية (Happy/Negative/Edge)**

## **أ) المنتجات (Products)**

* المطلوب: فحص التصفية/الفرز/الترقيم وحقول مختارة.
* المخرجات: جدول حالات (قيمة مدخلة → نتيجة متوقعة).
* القبول: يعمل limit/offset أو cursor، sort يقبل الحقول المسموح بها فقط، ورسالة 400 واضحة عند الخطأ.

## **ب) السلة (Cart)**

* المطلوب: إضافة/تعديل/حذف عنصر، تحديث الكميات، إعادة الحساب.
* المخرجات: سيناريوهات قبل/بعد (المجموع والضرائب والشحن).
* القبول: تطابق المجموع وتنعكس التغييرات فورًا بدون حالات سباق.

## **ج) الطلبات (Orders)**

* المطلوب: **Idempotency** للإنشاء + إرجاع Location ووسم نسخة (ETag) للموارد.
* المخرجات: لقطات ردود تُظهر عدم التكرار عند إعادة نفس الطلب.
* القبول: طلب مكرر بنفس المفتاح لا ينشئ Order جديدًا، ويتم إرجاع نفس الاستجابة.

## **د) المدفوعات/التسليم/الإرجاع**

* المطلوب: تدفق COD وتدفق إلكتروني إن وجد + إنشاء شحنة/إرجاع.
* المخرجات: جدول حالات النجاح والفشل ورسائل الخطأ.
* القبول: حالات 2xx للحالات الصحيحة، و4xx لمدخلات خاطئة مع تفاصيل مفهومة.

# **6) التزامن والتعارضات**

* المطلوب: تحديث متزامن لنفس المورد والتحقق من إدارة التعارض.
* المخرجات: تجربة موثّقة تُظهر ظهور 409 أو فشل If-Match عند تضارب النسخ.
* القبول: لا تُفقد بيانات، والتعارضات تُدار برسائل واضحة.

# **7) نموذج الأخطاء الموحد**

* المطلوب: توحيد جسم الخطأ ليحتوي type/title/status/detail/errors/trace\_id.
* المخرجات: دليل أخطاء + أمثلة من الإنتاج التجريبي.
* القبول: كل 4xx/5xx تتبع نفس الشكل، ولا وجود لـ PII داخل الأخطاء.

# **8) الأمان (Auth/CORS/Rate Limit/Secrets)**

* المطلوب: صلاحيات Roles/Scopes واضحة، انتهاء توكنات قصير، CORS مقيد، حدود طلبات مع رد 429، منع Mass Assignment.
* المخرجات: مصفوفة صلاحيات لكل دور + سياسة CORS + سياسة الحدود.
* القبول: 401 عند عدم المصادقة، 403 عند نقص الصلاحيات، أصول غير مصرح بها تُمنع، الحقول الحساسة تُتجاهل.

# **9) الكاش ووسوم النسخ (ETag)**

* المطلوب: تفعيل ETag على GET، وIf-None-Match يرجع 304، وIf-Match مطلوب في PATCH الحسّاسة.
* المخرجات: أمثلة ردود توضح السلوك.
* القبول: تقليل نقل البيانات على القوائم المتكررة، ومنع الكتابة على نسخة قديمة.

# **10) الأداء والتحمّل**

* المطلوب: خطة قياس لثلاث رحلات: (قائمة منتجات) و(تفاصيل منتج) و(إضافة للسلة → إنشاء طلب).
* المخرجات: تقرير أزمنة p95، ومعدلات الأخطاء تحت حمل متوسط.
* القبول: p95 للقراءات ≤ 300ms، للإنشاء/التعديل ≤ 500ms، وأخطاء 5xx ≤ 0.5%.

# **11) فحص أمني ديناميكي (DAST)**

* المطلوب: فحص آلي ضد مواصفة الـAPI (مثل ZAP baseline) ومراجعة الرؤوس الأمنية والحقن.
* المخرجات: تقرير HTML + قائمة العِلل مرتبة بالأولوية.
* القبول: لا ثغرات عالية غير مغلقة، والرؤوس الأمنية مضبوطة (TLS/HSTS/CSP…).

# **12) الرصد والتتبّع (Observability)**

* المطلوب: لوغ مُهيكل يحتوي request\_id/trace\_id/user\_id، وعدادات (QPS، 4xx/5xx، زمن)، وتتبع موزّع.
* المخرجات: لوحتان (Dashboards) للأخطاء والأداء + تنبيهات واضحة.
* القبول: القدرة على تتبّع أي طلب من اللوغ إلى قاعدة البيانات، وتنبيه تلقائي عند تجاوز العتبات.

# **13) الأتمتة في CI**

* المطلوب: تشغيل مجموعة الاختبارات آليًا (وظيفية/تعاقد/أداء خفيف/DAST بسيط) وتوليد تقارير كـArtifacts.
* المخرجات: روابط تقارير يومية + بوابة قبول (فشل البايبلاين عند تجاوز الحدود).
* القبول: أي تراجع في المقاييس يوقف الإطلاق تلقائيًا.

# **14) تسليم نهائي**

* المطلوب: ملف “ملخص جودة الـAPI” بجميع الروابط (Swagger، تقارير، لوحات، قرارات).
* المخرجات: مستند واحد شامل + جدول مهام مغلقة/مفتوحة وتواريخها.
* القبول: كل البنود أعلاه مرفقة بأدلة موثّقة وروابط فعّالة.

## **مؤشرات قبول كلية (OKRs تقنية للـAPI)**

* تحذيرات المواصفة/التعاقد: **صفر حرجة**.
* التغطية الاختبارية (الخدمة/المكتبات): **≥ 60%** كبداية.
* p95 للقراءات: **≤ 300ms**، للكتابة: **≤ 500ms**.
* Crash/5xx: **≤ 0.5%** من إجمالي الطلبات تحت حمل الاختبار.
* أخطاء موحّدة بنسبة **100%** (لا أجسام عشوائية).
* تفعيل ETag/If-None-Match على القوائم الرئيسية.
* سياسات Auth/CORS/Rate-Limit مفعّلة ومُوثّقة.

CI/CD – API (Laravel)

## 

# **CI/CD – API (Laravel) Execution Standard — Zahraah**

A practical, end‑to‑end blueprint for building and running a safe, fast CI/CD pipeline for the Laravel API. Includes goals, branching, stages & gates, security, rollback, example GitHub Actions, and checklists.

## **1) Goals & Scope**

* **Goals:** Safe and frequent deployments with no/low downtime, quality gates enforced, and instant rollback.
* **Scope:** Laravel 10+ (PHP 8.2), GitHub Actions, Docker image build & scan, environments staging and prod, observability & alerts.

## **2) Branching Strategy & Releases**

* **Model:** Trunk‑Based Development.
  + Short‑lived branches: feature/<slug>, hotfix/<slug>.
  + Protect main: mandatory review + green CI.
* **Versioning:** SemVer (X.Y.Z). Every production deployment = **Git tag**.
* **Release cadence:** Weekly release train by default; urgent hotfixes allowed.

## **3) Pipeline Stages & Health Gates**

**Stage 0 — Triggers & Context**

* Trigger on pull\_request to main, and on push to main or v\*.\*.\* tags.

**Stage 1 — Lint & Static Analysis**

* Tools: PHPStan (agreed level), PHPCS (PSR‑12), composer validate.
* **Gate:** any failure ⇒ PR cannot merge.

**Stage 2 — Tests & Coverage**

* Tools: Pest/PHPUnit (Unit + Feature) with a real test DB.
* **Coverage thresholds:** overall ≥ **65%** (critical paths ≥ **80%**).
* **Gate:** below threshold ⇒ fail.

**Stage 3 — Build (Docker)**

* Multi‑stage Dockerfile (dependencies builder → slim prod image).
* Tags: :sha-<GIT\_SHA>, :vX.Y.Z, :latest (protected).

**Stage 4 — Security Scans**

* composer audit + image scan (Trivy/Grype).
* **Gate:** unapproved Critical/High vulnerabilities ⇒ fail.

**Stage 5 — Publish**

* Push image to private registry (GHCR/ECR). Optionally attach SBOM.

**Stage 6 — Deploy to Staging**

* Safe rollout: php artisan migrate --force (forward‑compatible), cache config/routes, warm application.
* **Smoke tests:** /healthz, /readyz, one GET + one POST happy path.

**Stage 7 — Canary to Production**

* Progressive traffic: **5% → 25% → 100%**.
* **Health gates** per step (10–15 min observation): 5xx% < 0.3%, P95 < 300ms on key endpoints.
* Gate fail ⇒ **Immediate rollback** to last stable tag.

**Stage 8 — Post‑Release Checks**

* Watch errors/latency, DB/CPU, queues. Confirm feature flags as planned.

**Stage 9 — Tag & Release Notes**

* Create tag; publish release notes (changes/migrations/risks/rollback).

## **4) Secrets & Configuration Management**

* Store secrets in **GitHub Environments** (STAGING, PROD) or a Secret Manager.
* Examples: APP\_KEY, DB\_\*, REDIS\_\*, QUEUE\_CONNECTION, S3\_\*, CLOUDFLARE\_API\_TOKEN.
* **Never** hard‑code secrets or store them in YAML. Quarterly rotation. Least privilege.

## **5) Database Migrations (Zero/Low Downtime)**

* **Forward‑compatible:** add nullable columns first → backfill in batches → enforce NOT NULL.
* Indexes with ALGORITHM=INPLACE/INSTANT when available.
* Run migrations **before** sending 100% traffic to the new release.
* Rollback only if the migration is safe; otherwise **hotfix forward**.

## **6) Deployment Mechanics**

**A) Kubernetes**

* Deployment with readinessProbe: /readyz, livenessProbe: /healthz.
* HPA driven by CPU/RPS/P95. Rolling updates with maxUnavailable: 0.

**B) Docker/VM**

* docker compose pull && up -d with container healthchecks.
* LB (NGINX/HAProxy) using slow‑start and connection draining.

## **7) Observability & Operational Gates**

* **SLOs:** API availability 99.9% monthly; P95 < 300ms; 5xx% < 0.3%.
* **Automated gates:** pipeline step queries monitoring (Prometheus/Datadog) to assert thresholds during canary.
* Dashboards: RPS, latency P95/P99, errors, queue depth, cache hit, deployment markers.

## **8) Example GitHub Actions (YAML)**

name: api-ci-cd

on:

pull\_request:

branches: [main]

push:

tags: ["v\*.\*.\*"]

branches: [main]

jobs:

lint-test:

runs-on: ubuntu-latest

services:

mysql:

image: mysql:8

env:

MYSQL\_DATABASE: test

MYSQL\_ROOT\_PASSWORD: root

ports: ["3306:3306"]

options: >-

--health-cmd="mysqladmin ping -proot" --health-interval=10s --health-timeout=5s --health-retries=5

steps:

- uses: actions/checkout@v4

- uses: shivammathur/setup-php@v2

with: { php-version: "8.2", coverage: xdebug }

- uses: actions/cache@v4

with:

path: vendor

key: ${{ runner.os }}-composer-${{ hashFiles('\*\*/composer.lock') }}

- run: composer install --no-interaction --prefer-dist

- run: composer validate --no-check-publish

- run: vendor/bin/phpcs -n || true # enable as blocking if desired

- run: vendor/bin/phpstan analyse --no-progress

- run: php -d xdebug.mode=coverage vendor/bin/pest --coverage-clover=coverage.xml

- name: Enforce coverage gate

run: php scripts/coverage-gate.php coverage.xml 0.65 # 65%

build-scan-push:

needs: [lint-test]

if: github.ref\_type == 'tag' || github.ref == 'refs/heads/main'

runs-on: ubuntu-latest

permissions:

contents: read

packages: write

steps:

- uses: actions/checkout@v4

- name: Build Docker

run: |

docker build -t ghcr.io/org/api:${{ github.sha }} -f Dockerfile .

- name: Composer audit

run: composer audit || true # fail on High/Critical per policy

- name: Trivy scan

uses: aquasecurity/trivy-action@0.20.0

with:

image-ref: ghcr.io/org/api:${{ github.sha }}

format: 'table'

exit-code: '1'

severity: 'CRITICAL,HIGH'

- name: Login GHCR

uses: docker/login-action@v3

with:

registry: ghcr.io

username: ${{ github.actor }}

password: ${{ secrets.GITHUB\_TOKEN }}

- name: Push tags

run: |

docker tag ghcr.io/org/api:${{ github.sha }} ghcr.io/org/api:latest

docker push ghcr.io/org/api:${{ github.sha }}

docker push ghcr.io/org/api:latest

deploy-staging:

needs: [build-scan-push]

runs-on: ubuntu-latest

environment: STAGING

steps:

- name: Deploy to Staging

run: ./scripts/deploy.sh staging ${{ github.sha }}

- name: Smoke tests

run: ./scripts/smoke.sh https://staging.api.zahraah.com

canary-prod:

needs: [deploy-staging]

runs-on: ubuntu-latest

environment: PROD

steps:

- name: Deploy 5%

run: ./scripts/canary.sh prod ${{ github.sha }} 5

- name: Health gate

run: ./scripts/health-gate.sh prod --max-5xx=0.3 --max-p95=300

- name: Roll to 25%

run: ./scripts/canary.sh prod ${{ github.sha }} 25

- name: Health gate

run: ./scripts/health-gate.sh prod --max-5xx=0.3 --max-p95=300

- name: Roll to 100%

run: ./scripts/canary.sh prod ${{ github.sha }} 100

**Notes:**

* Replace org/api with your organization/repository name.
* deploy.sh / canary.sh / health-gate.sh are repo scripts that execute K8s/VM commands and query monitoring to enforce gates.

## **9) Rollback & Incident Plan**

* **App rollback:** deploy last known‑good tag (deploy.sh prod <LAST\_TAG>).
* **DB:** avoid data‑losing rollbacks; prefer forward hotfix migrations.
* **Feature disable:** use Feature Flags / Kill Switch.
* **Documentation:** incident record + blameless postmortem within 72h.

## **10) Software Supply Chain Security**

* Lock dependencies (composer.lock).
* Continuous dependency scanning + Dependabot.
* Sign images (Cosign) and optionally enforce **signed‑images only** pull policy.
* Generate SBOM (Syft) to assess vulnerability impact.

## **11) CI/CD Success KPIs**

* **Lead Time** (commit → prod): ≤ 24h.
* **Deployment Frequency:** ≥ 2/week.
* **Change Failure Rate:** ≤ 10%.
* **MTTR:** ≤ 60 min.
* **Pipeline P95 (build/test only):** ≤ 15 min.

## **12) RACI (Summary)**

| **Activity** | **R** | **A** | **C** | **I** |
| --- | --- | --- | --- | --- |
| YAML/scripts ownership | Backend Lead | CTO | SRE/QA | Product |
| Security & secrets policy | SRE | CTO | Backend | All |
| Coverage & tests | QA Lead | CTO | Backend/Mobile | All |
| Deploy & release | SRE | CTO | Backend/QA | Support |

## **13) Checklists**

**PR Gate — Assignment Matrix**

| Check | Owner | Evidence/Link | Status |
| --- | --- | --- | --- |
| CI green (lint/tests/coverage) | QA Lead | CI run URL | [ ] |
| No breaking changes | Backend Lead | oasdiff report / ADR | [ ] |
| DB migration reviewed | Backend + SRE | migration PR / runtime estimate | [ ] |
| Security review | SRE | dependency scan report / secrets check | [ ] |
| Release notes & flags | Product + Backend | draft notes / flags plan | [ ] |

**Release — Assignment Matrix**

| Check | Owner | Evidence/Link | Status |
| --- | --- | --- | --- |
| Tag & image pushed | Backend Lead | tag URL / registry image | [ ] |
| Staging smoke tests | QA Lead | smoke report / screenshots | [ ] |
| Canary plan & on‑call | SRE | runbook / on‑call schedule | [ ] |
| Secrets/config updated | SRE | env diff / change request | [ ] |
| Flags toggled & rollback cmd | Backend | flags file / rollback note | [ ] |

**Post‑release — Assignment Matrix**

| Check | Owner | Evidence/Link | Status |
| --- | --- | --- | --- |
| Health within targets | SRE | dashboards snapshot / gate logs | [ ] |
| Incidents & postmortem | SRE + Owners | incident ticket / postmortem link | [ ] |
| Changelog & docs updated | Backend + Product | PR link / doc page | [ ] |

**Automation hooks (to codify the checklists):**

* Coverage gate ➜ scripts/coverage-gate.php fails PR if below threshold.
* Breaking‑change gate ➜ oasdiff in CI fails PR on contract breaks.
* Security gate ➜ composer audit & container scan (Trivy) block on Critical/High.
* Health gate ➜ scripts/health-gate.sh queries monitoring during canary.

**Per‑environment quick checks**

* **Staging:** /healthz, /readyz, endpoints smoke, background jobs running, feature flags in intended state.
* **Prod Canary:** traffic % applied, P95/5xx stable for 10–15 min before next step, rollback command tested.

## **14) Appendices**

* deploy.sh (K8s): kubectl set image ... && kubectl rollout status ....
* health-gate.sh: query Prom/Datadog metrics and compare against thresholds.
* Docker hardening: official base image, non‑root user, least privileges, minimal layers.

Idempotency & Pagination

# **Idempotency & Pagination**

العميل (Flutter) ⟶ API Gateway/Reverse Proxy ⟶ خدمات الدومين (Orders/Cart/Payments/…)  
حولها: **Redis/DB** لحفظ مفاتيح عدم التكرار، **Logs/Tracing**، و**محدد المعدّل**.  
ترقيم الصفحات يعتمد على **Cursor** (مفضل للداتا المتغيرة) أو **Offset** (لأدمن/تقارير صغيرة) أو **Keyset** (استعلامات ضخمة).

# **أولًا: Idempotency (عدم تكرار التنفيذ)**

## **التعريف (مختصر)**

ضمان أن **تكرار نفس الطلب** (بسبب انقطاع/تأخير/إعادة محاولة) **لا يخلق أثراً جديدًا** غير المقصود (مثل إنشاء طلب مكرر أو سحب دفعة مرتين).

## **أين نستخدمه**

* كل **POST** غير آمن بطبيعته: إنشاء **order**, **payment**, **refund**, **coupon-redemption**, **webhook ingestion**.
* عمليات **PATCH/PUT/DELETE** الحسّاسة التي قد يُعاد إرسالها تلقائيًا.

## **كيف نطبّقه (المعيار)**

1. يرسل العميل هيدر **Idempotency-Key: <UUIDv4>** فريد لكل عملية “نية عمل”.
2. تحفظ الـ API **سِجلًا** للمفتاح يتضمن:
   * idempotency\_key, request\_fingerprint (هاش جسم الطلب + المسار + الـ user\_id),
   * status النهائي، **payload** الاستجابة، و**expiry\_at**.
3. عند تكرار الطلب بنفس المفتاح:
   * إذا كان **نفس البصمة** ⟶ أعِد **نفس الاستجابة** والـ **status code** (مع هيدر Idempotent-Replayed: true).
   * إذا كانت **بصمة مختلفة** ⟶ **409 Conflict** (طلبان مختلفان بمفتاح واحد).
4. **TTL** المفتاح:
   * عمليات الدفع/الطلبات: **72 ساعة** (أقل أو أكثر حسب العمل).
   * عمليات إدارية أقل حساسية: **24 ساعة**.
5. التخزين:
   * **Redis** (سرعة) + **persist** أو **DB** بعمود **unique** على (idempotency\_key, user\_id).

### **مخطط تدفق سريع**

* استقبل الطلب ⟶ اقرأ Idempotency-Key
* إن لم يوجد: **400 Bad Request** (إن كان endpoint يتطلبه)
* ابحث عن المفتاح:
  + **غير موجود**: أنشئ سجلًا *in-progress* ثم نفّذ العملية.
  + **موجود**:
    - in-progress ⟶ **409/425** (أعد المحاولة لاحقًا) أو انتظر قليلًا ثم أعد.
    - succeeded/failed ⟶ أعد نفس الرد المخزّن.

### **جَدول (مقترح) idempotency\_keys**

id (pk), idempotency\_key (unique per user), user\_id, request\_fingerprint\_sha256,  
status (in\_progress/succeeded/failed), response\_status, response\_body\_json,  
expires\_at, created\_at, updated\_at.

### **أخطاء ومعالجات**

* **409 Conflict**: مفتاح مستَخدم لطلب مختلف (بصمتان مختلفتان).
* **422 Unprocessable Entity**: فشل تحقق تجاري لكن المفتاح **يظل محجوزًا** للّحظة نفسها (لنفس الجسم).
* **425 Too Early / 429**: إذا كان التنفيذ جارٍ أو معدل المحاولات مرتفع.
* **202 Accepted** (للعمليات الطويلة): أعِد operation\_id وتتبّع التقدم؛ ربط النتيجة بنفس المفتاح.

### **ملاحظات تنفيذ مهمة**

* **Fingerprint** = method + path + normalized(body) + user\_id لمنع إعادة استخدام المفتاح ببيانات أخرى.
* **Bulk**: عند POST دفعي (مثلاً إنشاء 100 عنصر)، استخدم **مفتاحًا واحدًا** للدفعة، وخزّن **قائمة نتائج**، واجعل الإعادة تُعيد نفس القائمة.
* **Outbox/Eventing**: عند نشر حدث (مثلاً “OrderCreated”) استخدم **مفتاحًا داخليًا** لمنع النشر المكرر.
* **Optimistic Locking**: لـ PATCH/PUT استخدم **ETag/Version** + If-Match لمنع “ضياع التعديلات” (انظر أدناه).

### **مثال (إنشاء طلب)**

POST /v1/orders

Idempotency-Key: 9d0f8f2c-9a76-4b8f-8a2e-0a2b6f7d8c3f

Body: { "cart\_id": "c\_123", "address\_id": "a\_9", "payment\_method": "cod" }

→ 201 Created

Headers: Idempotent-Replayed: false

Body: { "id":"ord\_789", "status":"placed", ... }

إعادة نفس الطلب (انقطع النت):

→ 201 Created

Headers: Idempotent-Replayed: true

Body: { "id":"ord\_789", "status":"placed", ... } // نفس المحتوى

# 

# **ثانيًا: Pagination (ترقيم الصفحات)**

## **الهدف**

* تقليل زمن الاستجابة وحجم البيانات.
* **تحديدية** النتائج عبر **ترتيب ثابت** و**مؤشر صفحة** واضح.
* دعم تمرير آمن في وجود إضافات/حذف مستمر.

## **الأنماط (اختَر حسب الحالة)**

1. **Cursor-based (مُفضّل للداتا الحيّة):**
   * معاملات: limit, cursor
   * ترتيب ثابت: مثل created\_at DESC, id DESC (tie-breaker).
   * الاستجابة: next\_cursor, has\_more (+ اختياري prev\_cursor).
   * **لا تُظهر** offset. الكيرسر يشفّر آخر مفتاح/قيمة شوهدت.
2. **Offset/Limit (بسيط):**
   * مناسب لقوائم صغيرة/إدارية.
   * مشاكل: القفزات/التكرار مع بيانات تتغير بسرعة.
3. **Keyset Pagination (للقوائم الضخمة):**
   * يعتمد على شروط WHERE key < last\_key مع فهارس مناسبة؛ الأسرع على جداول كبيرة.

## **معايير ثابتة**

* **Default limit = 20**، **Max = 100** (قابلة للتعديل لكل مورد).
* **Sort** معلَن وثابت لكل مورد (مثل created\_at DESC, id DESC).
* تغيير **filters/sort** يبطل الكيرسر (أعد **400** مع رسالة: “الـ cursor لا يطابق معايير الاستعلام الحالية”).
* **Deterministic**: عند تساوي created\_at استخدم **id** كحاسم نهائي.
* **Partial fields**: وفّر fields لاختيار الحقول لتقليل الحجم.
* **Counts**: لا تُرجِع total افتراضيًا إن كان مُكلفًا؛ وفر **HEAD /resource** أو **/resource/count**.

## **واجهة موحّدة (مقترحة)**

### **طلب**

GET /v1/orders?limit=20&cursor=eyJjcmVhdGVkX2F0IjoiMjAyNS0wOS0xM1QxMzo0Mjo1WiIsImlkIjoiYWJjMTIzIn0

### **استجابة**

200 OK

Link: <...cursor=eyJ...> rel="next"

Body: {

"data": [ { "id":"ord\_789", "created\_at":"2025-09-13T12:40:05Z", ... }, ... ],

"page\_info": {

"next\_cursor": "eyJjcmVhdGVkX2F0Ijoi...IiwiaWQiOiJhYmMxMjMifQ",

"has\_more": true

}

}

### **باراميترات موحّدة**

* limit (1..100)، الافتراضي 20.
* cursor (اختياري) — base64/URL-safe JSON يحوي **المفاتيح الحدّية** (مثلاً created\_at, id).
* sort (قائمة بيضاء فقط: created\_at\_desc|asc, …).
* **Filters** واضحة بأسماء ثابتة (status, customer\_id, from, to …).

## **أخطاء وتصرّفات**

* **400 Bad Request**: كيرسر غير صالح/قديم/لا يطابق الفلاتر/الترتيب.
* **416 Range Not Satisfiable**: إذا استُخدم range غير مدعوم.
* **422**: فلاتر متعارضة.
* **429**: عند إساءة الاستعمال (limit كبير جدًا/معدل طلبات عالٍ).

## **أداء وفهارس**

* أنشئ فهارس مركبة توافق **الترتيب والفلاتر** الأكثر استخدامًا، مثل:
  + orders (created\_at DESC, id DESC)
  + orders (customer\_id, created\_at DESC, id DESC)
* تجنّب ORDER BY RAND() أو الفرز على عمود بلا فهرس.

# **تكامل Idempotency + Pagination + Concurrency**

* **PUT/PATCH**: استخدم **ETag** أو حقل version + هيدر If-Match.
  + إن تغيّر الإصدار ⟶ **412 Precondition Failed**.
* **GET مع ETag**: ادعم If-None-Match ⟶ **304** لتقليل النقل.
* **POST طويلة الأمد**: أعِد **202 + operation\_id**، ووفّر GET /operations/{id}؛ اربطها بـ Idempotency-Key.

# **قوائم تنفيذ سريعة**

**Idempotency**

* إلزام Idempotency-Key في POST الحسّاسة.
* تخزين request\_fingerprint + ردّ نهائي (body + status).
* TTL مناسب (72h للطلبات/الدفعات).
* Idempotent-Replayed عند الإعادة الناجحة، و**409** عند تغيّر البصمة.
* مراقبة: معدل تكرار المفاتيح، تضارب البصمات، زمن كتابة/قراءة المفتاح.

**Pagination**

* تبنّي **Cursor** افتراضيًا، **Offset** فقط للإدمن/تقارير صغيرة.
* توثيق limit/max, sort, filters، وربطها بفهرسة فعلية.
* تضمين page\_info.next\_cursor + has\_more وLink: rel="next".
* 400 عند كيرسر لا يطابق الفلاتر/الترتيب.
* لوحات مراقبة: p95 زمن القائمة، متوسط حجم الرد، نسبة استخدام prev/next.

دليل العمليات — API Operations Playbook

# **دليل العمليات — API Operations Playbook**

## **0) الهدف والنطاق**

* **الهدف:** تشغيل الـAPI كمنتج مؤسسي موثوق 24/7 مع استجابة سريعة للحوادث، تغييرات آمنة، وتوافر عالي.
* **النطاق:** المراقبة، الحوادث، التغيير والإصدارات، السعة والأداء، الاستمرارية (DR/Backup)، الأمان التشغيلي، التكلفة، التكاملات الخارجية، والجاهزية التشغيلية.

## **1) نموذج التشغيل (Operating Model)**

* **On-call 24/7:** مناوبة أولى (Primary) وثانية (Secondary) لكل نطاق خدمة.
* **NOC/SRE/Backend DRI:**
  + **NOC/SRE:** مراقبة وتنبيه، إدارة الحوادث، السعة، الموثوقية.
  + **Backend DRI:** القرار الفني أثناء الحوادث على خدمة معيّنة.
* **مسارات التصعيد:** زمن استجابة فوري للحالات الحرجة (S0/S1)، سلّم تصعيد واضح حتى Incident Commander.

**2) العمليات الأساسية (Core Processes)**

### **2.1 إدارة الحوادث (Incident Management)**

* **تصنيف الشدة:**
  + **S0** انقطاع شامل/أثر مالي كبير،
  + **S1** أثر مرتفع/شريحة كبيرة،
  + **S2** أثر متوسط،
  + **S3** منخفض.
* **SLA الاستجابة/الحل (إرشادي):**
  + S0: **استجابة ≤ 5 د** / احتواء ≤ 30 د / حل ≤ 4 س.
  + S1: ≤ 10 د / ≤ 60 د / ≤ 8 س.
* **دورة الحادث:** رصد → تصنيف → تعبئة War Room → احتواء → علاج جذري → **Postmortem بلا لوم** خلال 72 ساعة.
* **تواصل:** قالب إعلان داخلي/خارجي، وتحديث حالة دوري (Status Update Cadence).

### **2.2 إدارة المشكلة (Problem Management)**

* **RCA** موحّد (5 Whys / Ishikawa).
* **KEDB** (قاعدة الأخطاء المعروفة) + عناصر إجراء (Action Items) بمالكين ومواعيد.

### **2.3 إدارة التغيير والإصدارات (Change & Release)**

* أنواع التغيير: **قياسي** (مسبق الاعتماد)، **عادي** (يتطلب مراجعة)، **طارئ**.
* **Release Calendar** + نوافذ صيانة، استراتيجيات نشر: **Canary/Blue-Green**.
* **بوابات صحّة (Health Gates):** p95/5xx وError Budget توقف النشر تلقائيًا عند التجاوز.

### **2.4 الجاهزية التشغيلية (Operational Readiness Review – ORR)**

* قبل الإنتاج: **مراقبة، تنبيهات، Runbooks، SLOs، خط رجوع (Rollback)، اختبارات عقد/E2E، خطة DR** — كلها مُثبتة.

### **2.5 إدارة السعة والأداء (Capacity & Performance)**

* **توقع الطلب (Forecast):** QPS/RT/Throughput ربع سنوي.
* **Headroom ≥ 30%** للخدمات الحرجة.
* **Load/Stress Tests** قبل كل إطلاق رئيسي.
* Auto-Scaling بقواعد واضحة (CPU/RT/Queue Depth).

### **2.6 الاستمرارية والتعافي (BCP/DR & Backups)**

* **RPO/RTO** لكل مكوّن (مثال: RPO ≤ 24س، RTO ≤ 4س).
* تمارين **Failover/Failback** ربع سنوية.
* نسخ احتياطي مشفّر + اختبارات استعادة دورية.

### **2.7 الأمان التشغيلي (SecOps)**

* **Vuln Mgmt & Patching** بمواعيد ثابتة.
* **SIEM/Alerts** لمحاولات هجوم/WAF/Rate Abuse.
* **تدوير أسرار** ومفاتيح وفق سياسة (90–180 يوم).

### **2.8 تكاليف التشغيل (FinOps Ops)**

* تنبيهات **Cost Anomaly**، تنظيف موارد يتيمة، تتبّع تكلفة/طلب.
* تقارير شهرية حسب الفريق/الخدمة/البيئة.

### **2.9 الطرف الثالث (Third-Party Ops)**

* مراقبة **SLAs** للمزوّدين، خطة **Fallback/Circuit Breaker**، ومصفوفة الاتصال.

## **3) الأدوار (RACI مختصر)**

* **SRE/NOC:** تنبيه، حوادث، سعة، DR.
* **Backend DRI:** قرارات تقنية/إصلاحات.
* **Security:** SecOps، مفاتيح، ثغرات.
* **Data/BI:** جودة بيانات تشغيلية، تنبيهات Freshness.
* **Product/PM:** أولوية/تواصل خارجي، Go/No-Go.
* **Support/DevRel:** واجهة العملاء، تحديثات الحالة.

**4) المناوبات (On-call)**

* تناوب أسبوعي، **Primary + Secondary**، تسليم مناوبة مع **Handover Checklist**.
* **Quiet Hours** لغير الحرِج، لكن S0/S1 دائمًا page.
* اختبارات أسبوعية لتأكد وصول التنبيه (Paging Drill).

## **5) لوحات المراقبة الأساسية (Dashboards)**

* **منظور المنصة:** QPS، p95، 5xx/4xx، 429، توافر.
* **Gateway/API:** معدلات WAF/Rate-Limit، أعلى المسارات.
* **الخدمات:** Latency/Errors/Throughput لكل خدمة.
* **البنية:** DB/Cache/Queues/Storage.
* **Webhooks/Integrations:** نجاح/فشل/إعادة محاولات.
* **أعمال:** أوامر/دقيقة، Conversion Overlay عند الحاجة.

**6) مؤشرات الأداء التشغيلية (Ops KPIs)**

* **MTTA** (متوسط زمن الاستجابة للتنبيه).
* **MTTR** (متوسط زمن الحل).
* **Incident Rate** شهريًا حسب الشدة.
* **Change Failure Rate** و**Deployment Frequency**.
* **SLO Compliance** و**Error Budget Burn**.
* **Cost per 1k requests** واتجاهه.

## **7) كُتيّبات تشغيل إلزامية (Runbooks) — ماذا نكتب أولاً؟**

1. **API Gateway Down / 5xx Spike**
2. **DB Saturation / Connection Storm**
3. **Cache Cluster Fail / High Miss**
4. **Queue Backlog / Consumer Lag**
5. **Auth/JWT Failure / JWKS Fetch Error**
6. **Webhooks Delivery Fail / DLQ**
7. **Third-Party Timeout / Circuit Open**
8. **Regional Failover / DNS Cutover  
   هيكل كل Runbook:** سياق → كيفية الاكتشاف → التشخيص → التخفيف/الالتفاف → التحقّق → الرجوع → المالك → زمن الحل المتوقع.

**8) قوالب جاهزة (Templates)**

* **Incident Announcement (داخلي/خارجي)**
* **Postmortem**: التأثير، الخط الزمني، السبب الجذري، ما الذي كان جيدًا/سوءًا، إجراءات منع التكرار + ملاك ومواعيد.
* **Change Request**: الوصف، المخاطر، خطة التراجع، بوابات الصحّة.
* **ORR Checklist**: مراقبة، تنبيهات، SLOs، Runbooks، DR، اختبارات عقد/E2E.
* **Shift Handover**: أحداث مفتوحة، مخاطر، تنبيهات متكررة، عناصر متابعة.

**9) طقوس تشغيلية (Cadence)**

* **يوميًا:** مراجعة تنبيهات الليلة، صحة الـDashboards، تذاكر مفتوحة.
* **أسبوعيًا:** مراجعة SLO/Incidents، أهم 3 عناصر إجراء، تنبيهات مزعجة (Alert Fatigue).
* **شهريًا:** مراجعة Error Budget، تغييرات عالية المخاطر، تقارير FinOps، اختبارات استعادة عيّنية.
* **ربع سنوي:** تمرين DR/Game Day، اختبار اختراق، مراجعة سعة/تكلفة.

## **10) خطة تنفيذ العمليات**

* نشر تنبيهات أساسية (p95/5xx/429/DB/Queue).
* اعتماد تصنيف الشدة + مصفوفة التصعيد.
* نشر قوالب Incident/Postmortem/Change/ORR.
* إنتاج **5 Runbooks حرِجة** وتشغيل ORR لخدمتين مرجعيتين.
* تفعيل Release Calendar + Health Gates + Canary.
* تمرين **Paging Drill** أسبوعي و**Load Test** رئيسي.
* تمرين **DR** (Failover/Failback) على بيئة staging/جزئيًا على prod.
* إطلاق لوحات منصة موحّدة + مراجعة تنبيهات مزعجة.
* بدء **Problem Mgmt** رسميًا (RCA + KEDB).
* مراجعة KPIs (MTTA/MTTR/Change Failure/SLO).
* سدّ الفجوات في Runbooks (الـ8 أعلاه).
* تقرير عمليات تنفيذي وتحديث سياسات.

## **11) تشيك**

* مصفوفة الشدة والتصعيد منشورة.
* On-call Primary/Secondary وتناوب مُفعّل + Handover.
* Dashboards أساسية + Alerts فعّالة ومجرَّبة.
* 5+ Runbooks حرِجة منشورة ومُختبرة.
* Release Calendar + Health Gates + Canary/Blue-Green.
* ORR مُنَفَّذ لكل خدمة قبل الإنتاج.
* DR/Backups مُختبرة وفق RPO/RTO.
* SecOps: Patching، SIEM، تدوير أسرار.
* FinOps: تنبيهات تكلفة، تنظيف موارد يتيمة.
* Problem Mgmt + KEDB قيد العمل.

## **12) مخاطر متوقعة وكيف نمنعها**

* **تنبيهات كثيرة/مزعجة:** تنظيف قواعد التنبيه أسبوعيًا، اعتماد SLO-based alerts.
* **نشر يسبب أعطال:** Health Gates + Canary + Rollback جاهز.
* **اعتماد مفرط على فرد:** Runbooks مُفصّلة + تدريب مناوبات.
* **ثغرات أمنية تشغيلية:** Patch Cadence + SIEM + WAF Tuning + Rotation.
* **تكلفة غير متوقعة:** FinOps تنبيهات + تقارير أسبوعية + Rightsizing.

علامة التبويب 7

# **البنية والمعايير**

## **1) العمارة والبيئات**

* Gateway على الحافة (Cloudflare/Kong/NGINX) + خدمات الـAPI + طبقة البيانات (DB/Redis/Queues/Storage) + المراقبة.
* مسار بيئات واضح: **dev → staging (بيانات وهمية) → prod**.

## **2) المصدر الوحيد للحقيقة (OpenAPI)**

* مواصفة **OAS 3.x** محدثة مع أمثلة واقعية.
* أدوات فحص: **Spectral (lint)** + **oasdiff** (لكشف تغييرات كاسرة) ضمن الـCI.

## **3) الحوكمة والإصدار (Versioning)**

* الإصدار في المسار: **/v1/** …
* سياسة **Deprecation بإنذار ≥ 90 يومًا** وتشغيل متوازٍ عند /v2 قبل الإيقاف.

## **4) تصميم REST للموارد**

* صيغة جمع في المسارات، الأفعال عبر HTTP.
* JSON UTF-8، اختيار حقول (?fields=)، توسيع علاقات (?expand=).
* ترقيم Cursor/Limit (حد أقصى 100).

## **5) نماذج الاستجابة والأخطاء**

* خطأ موحّد: **code + message + details + trace\_id**.
* لا تُضمَّن أي PII، وحالات HTTP قياسية (200, 201, 400, 401…).

## **6) الهوية والصلاحيات (AuthN/Z)**

* **JWT قصير العمر** + Refresh.
* **OAuth2 Client Credentials** للشركاء.
* **RBAC/Scopes** للأدوار.
* الأسرار مخزنة في **Secret Manager** فقط.

## **7) الأمان الصلب**

* TLS فقط، CORS Allowlist.
* Rate Limit مع رؤوس قياسية.
* **Idempotency-Key** للعمليات المالية/القابلة للتكرار.
* منع Mass Assignment + تحقق صارم من المدخلات.

## **8) الأداء والتخزين المؤقت**

* **SLO p95** للقراءة ≤ 300ms، للكتابة ≤ 500–800ms.
* Cache-Control + ETag/If-None-Match/If-Match.
* Redis للقراءات الثقيلة.

## **9) الملفات والروابط**

* رفع عبر **multipart**.
* تنزيل عبر **Signed URLs** قصيرة العمر مع صلاحيات.

## **10) الوظائف غير المتزامنة**

* 202 + Location لمهام طويلة.
* مورد /jobs يعرض **status/progress/result**.

## **11) التكاملات وWebhooks**

* Sandbox منفصل.
* Webhooks موقَّعة (HMAC + طابع زمني + event\_id).
* إعادة المحاولة مع Backoff.
* أحداث موصى بها: **order.created, order.shipped, stock.adjusted…**

## **12) معايير البيانات والمخططات**

* **ISO-8601 UTC** للطوابع الزمنية.
* مبالغ مالية بالأصغر (cents/هللات).
* UUID/ULID للمعرّفات.
* جداول مرجعية للسمات (ألوان/مقاسات).

## **13) الملاحظة والتتبّع (Observability)**

* Logs مُهيكلة (JSON) مع **trace\_id/request\_id/user\_id**.
* Metrics: QPS, Latency P95, 4xx/5xx.
* Tracing عبر **OpenTelemetry**.
* Alerts عند تجاوز العتبات.

## **14) الجودة والاختبارات**

* **Contract Tests** من OpenAPI.
* اختبارات وظيفية/تكامل/E2E.
* **DAST (ZAP)** دوري.
* أداء: k6/Artillery لرحلات أساسية.

## **15) بوابة الـAPI (Gateway)**

* توحيد **Auth, Rate Limit, CORS, Headers, Caching, Tracing**.
* قواعد WAF على الحافة (Cloudflare).

## **16) CI/CD — مراحل وبوابات**

* Lint/Static → Tests/Coverage → Build (Docker) → Security Scans → Staging+Smoke → Canary → Prod.
* Rollback سريع.
* تغطية ≥ 60% كبداية، حظر الثغرات الحرجة.

## **17) مؤشرات القبول الكلية (SLOs/OKRs)**

* تحذيرات العقد الحرجة = صفر.
* P95 القراءة ≤ 300ms.
* Crash/5xx ≤ 0.5%.
* أخطاء موحّدة 100%.
* ETag مفعّل للقوائم.

## **18) السياسات التشغيلية**

* /health و/metrics إلزامية.
* إدارة الخصوصية وحذف بيانات العملاء عند الطلب.
* Changelog واضح.
* تدوير مفاتيح كل 90–180 يومًا.

## **19) إدارة التغيير والإصدارات الكاسرة**

* إعلان **Deprecation + Headers** (Deprecation, Sunset).
* فترة تداخل مع الإصدار الجديد.
* أدوات هجرة واضحة.

## **20) جاهزية الشركاء**

* إصدار مفاتيح/عملاء + Scopes.
* Sandbox جاهز.
* أمثلة Postman/SDKs.
* Onboarding ≤ يومين.

## **21) قوائم التحقق العملية**

* إضافة Endpoint جديد → OpenAPI → Lint/oasdiff → Tests → تحديث Gateway → نشر Docs/SDKs.
* قبل النشر: تحديث OAS، مراجعة أمنية، مراقبة مفعّلة، خطة Rollback جاهزة.

# **تحسينات مطلوبة**

1. **حوكمة البيانات والامتثال**: سياسات تصنيف، احتفاظ، مسح PII من اللوغ.
2. **أمن متقدّم**: mTLS داخلي، مكافحة احتيال COD (adaptive rate limits، OTP score).
3. **التعافي من الكوارث**: RPO/RTO محددين + اختبارات Restore فعلية.
4. **خطة الأداء والسعة**: Stress/Spike/Soak tests + هندسة مضادة لـN+1.
5. **كتالوج أخطاء رسمي**: مع أمثلة حلول لعملاء الموبايل.
6. **تجربة العملاء**: سياسات Retry/Timeout موصى بها + دعم Offline-safe للسلة.
7. **Error Budgets & Threat Modeling**: ربط SLOs بإيقاف الإطلاق + مراجعة OWASP ربع سنوية.
8. **Dev Experience**: بوابة مطورين + SDKs تولَّد آليًا + Workspace عام لـPostman.

(1) Architecture & Environments

# **العمارة والبيئات — Architecture & Environments**

## **1) آلية العمل & الاعتماديات (Mechanism & Dependencies)**

* **Gateway** (Cloudflare/Kong/NGINX) على الحافة: مسؤول عن TLS، WAF، Rate Limiting، CORS، مراقبة.
* **خدمات الـAPI (Laravel/Microservices)**: تعالج الموارد (Products, Orders, Users…).
* **طبقة البيانات**: MySQL/Postgres، Redis للكاش، Queues (RabbitMQ/SQS) للمهام غير المتزامنة، S3 أو ما يعادله للتخزين.
* **Observability Layer**: Logging، Metrics، Tracing، Alerts.
* **بيئات**: dev → staging (بيانات وهمية) → prod (بيانات حقيقية).
* الاعتمادية: أي طلب يمر عبر الـGateway → يوجّه للخدمات → يقرأ/يكتب في DB أو Queue أو Storage → يصدّر بيانات للمراقبة.

## **2) الغرض والنطاق (Purpose & Scope)**

* تحقيق فصل كامل بين البيئات (dev/staging/prod) لتجنّب أي تسريب بيانات حقيقية.
* ضمان وجود بنية مرنة وقابلة للتوسع.
* تمكين نشر سريع وآمن عبر CI/CD مع مراقبة كاملة.

## **3) المدخلات & المخرجات (Inputs/Outputs)**

**مدخلات:**

* نطاق api.zahraah.com، حسابات سحابة (DNS/TLS/WAF).
* إعدادات Gateway، ملفات بيئة (ENV) لكل بيئة.

**مخرجات:**

* روابط بيئات (Staging/Prod).
* حسابات اختبار (عميل/مشرف/مندوب).
* ملف متغيرات البيئة (DB URL، Redis URL، …).

## **4) المسؤوليات & الأدوار (R&R)**

* **Backend Team:** إعداد خدمات Laravel وربطها بالـDB/Redis.
* **SRE/Infra:** إعداد Gateway، CI/CD، المراقبة.
* **Security:** سياسات WAF، إدارة الأسرار، التحقق من TLS.
* **QA:** تشغيل Smoke/Contract Tests على dev/staging.

## **5) السياسات والمعايير (Policies & Standards)**

* Stateless API.
* JSON UTF-8 افتراضي.
* الإصدار في المسار /v1/.
* فصل صارم بين بيئة Staging والإنتاج.
* سرية تامة للأسرار (ENV/Secret Manager).

## **6) إجراءات التنفيذ (Implementation Steps)**

1. إعداد نطاق api.zahraah.com وربط TLS + WAF.
2. نشر Gateway وربط المسارات /v1/\* بسياسات Auth/Rate/CORS.
3. نشر خدمات Laravel API وربطها بـ DB/Redis/Queue/S3.
4. تفعيل Logging/Tracing/Metrics وربطها بلوحات مراقبة.
5. إنشاء بيئة dev (للتطوير)، Staging (بيانات وهمية + اختبارات)، Prod (بيانات فعلية).
6. تفعيل CI/CD لنشر تلقائي من GitHub Actions إلى كل بيئة.

## **7) قبول الجودة (Acceptance Criteria / DoD)**

* روابط بيئات فعّالة (staging, prod).
* حسابات اختبار تعمل.
* جميع طلبات الاختبار تمر عبر Staging فقط.
* لا توجد أسرار مخزّنة في الكود.

## **8) القياس والمراقبة (Metrics/SLIs/SLOs)**

* **SLIs:** Latency (P95 ≤ 300ms)، Availability ≥ 99.9%، معدل أخطاء 5xx ≤ 0.5%.
* **Dashboards:** أخطاء/زمن استجابة/معدل الطلبات.
* **Alerts:** عند تجاوز P95 أو ارتفاع 5xx أو زيادة غير طبيعية لـ429.

## **9) التكاملات (Integrations)**

* Gateway يحقن **X-Request-Id, traceparent**.
* OpenAPI ملف مواصفة لجميع المسارات.
* CI/CD Pipeline ينشر إلى dev/staging/prod ويربط مع مراقبة.

## **10) الأمان (Security)**

* TLS فقط (رفض HTTP العاري).
* CORS قائمة بيضاء لأصول محددة.
* أسرار مخزنة في Secret Manager.
* RBAC/Scopes للأدوار (admin, support, customer…).

## **11) الأداء والكاش (Performance & Caching)**

* Cache-Control, ETag, If-None-Match على GET.
* Redis للقراءات الثقيلة (قائمة منتجات، فئات).
* قواعد SLO: قراءة P95 ≤ 300ms، كتابة ≤ 500–800ms.

## **12) المخاطر & الضوابط (Risks & Controls)**

* خطر سوء تهيئة CORS/Rate/Secrets → ضوابط: تدقيق دوري + فحص ZAP.
* خطر نشر إلى Prod بدل Staging → ضوابط: بوابة مراجعة إلزامية في CI/CD.

## **13) برنامج الاختبارات (Testing Plan)**

* **Smoke Tests:** /health, /ready + GET/POST أساسي.
* **Contract Tests:** مقارنة OpenAPI مع التنفيذ (Dredd/Schemathesis).
* **Security baseline:** ZAP.

## **14) الوثائق والتحويلات (Docs & Change Mgmt)**

* OpenAPI منشور عبر Swagger/Redoc.
* أي تعديل كاسر → إصدار /v2 + إشعار Deprecation ≥ 90 يومًا.

## **15) الأدوات & المراجع (Tools & References)**

* Gateway: Cloudflare/Kong/NGINX.
* OpenAPI: Stoplight/Spectral، Dredd/Schemathesis.
* مراقبة: Prometheus, Grafana, Datadog.
* CI/CD: GitHub Actions.

## **16) Checklist**

* نطاق api.zahraah.com + TLS/WAF جاهز.
* Gateway مع سياسات Auth/Rate/CORS.
* روابط dev/staging/prod مفعلة.
* ملف ENV لكل بيئة.
* لوحات Logs/Metrics/Tracing جاهزة.
* Smoke Tests تعمل على Staging.

(2) OpenAPI

# **(2) OpenAPI كمصدر الحقيقة — OpenAPI as the Single Source of Truth**

## **1) آلية العمل والاعتماديات**

* OpenAPI 3.x/3.1 هو العقد الرسمي بين العميل (موبايل/ويب/شركاء) والخدمة.
* كل مسار/مدخل/مخرج/أخطاء/أمن معرف في ملف OAS واحد، وهو المرجع عند التصميم، الكود، الاختبارات، والتوثيق.
* أدوات تحقق آلية: Spectral (lint) + فحص تغييرات كاسرة (oasdiff) داخل CI.

## **2) الغرض والنطاق**

* توحيد العقد ومنع أي اختلاف بين المواصفة والتنفيذ.
* النطاق يشمل جميع واجهات /v1… العامة والداخلية، بما فيها Webhooks.

## **3) المدخلات والمخرجات**

* المدخلات: نماذج البيانات، قرارات التصميم (الإصدار/الحالات/الأخطاء)، سياسات الأمن.
* المخرجات:
  + ملف openapi.yaml محدث.
  + Swagger/Redoc منشور داخليًا وخارجيًا.
  + تقارير Lint/Contract محفوظة كـ Artifacts في CI.

## **4) المسؤوليات والأدوار**

* Backend: تصميم وتحديث المخططات والمسارات.
* QA: تنفيذ اختبارات العقد (Dredd/Schemathesis).
* SRE: استضافة Swagger/Redoc ونشر التقارير.
* Security: مراجعة securitySchemes ونماذج الأخطاء.

## **5) السياسات والمعايير**

* OAS هو مصدر الحقيقة؛ أي كود يخالف المواصفة مرفوض.
* كل Endpoint يحتوي أمثلة لطلب/رد ناجح وأخطاء شائعة.
* تسمية ثابتة: kebab-case للمسارات، snake\_case للحقول.
* توحيد آليات الترقيم، الفرز، الفلترة.

## **6) إجراءات التنفيذ**

1. تحديث أو إنشاء openapi.yaml.
2. تشغيل Spectral lint و oasdiff محليًا.
3. في PR: CI يشغل Lint + Contract Tests (Dredd/Schemathesis) ضد بيئة Staging.
4. نشر Swagger/Redoc تلقائيًا عند الدمج.
5. توليد SDKs (JS/Swift/Kotlin/PHP) من OAS وربطها بالـCI.

## **7) قبول الجودة**

* لا توجد فروقات حرجة بين التنفيذ والمواصفة.
* Lint ينجح بدون أخطاء حرجة.
* Swagger/Redoc يعمل ويعرض كل المسارات.
* أي تغيير كاسر موثق بخطة ترحيل (إصدار /v2 أو بدائل متوافقة للخلف).

## **8) القياس والمراقبة**

* صحة التوثيق: درجة Lint ≥ 90/100.
* زمن تحديث التوثيق مع الكود ≤ 24 ساعة.
* Contract Tests Pass Rate ≥ 99%.

## **9) التكاملات**

* ربط المواصفة ببوابة المطورين + إدارة مفاتيح الشركاء.
* توليد Postman/Insomnia Collections تلقائيًا من OAS.

## **10) الأمان**

* تعريف واضح لـ securitySchemes (Bearer JWT, apiKey, oauth2).
* توثيق أخطاء 401/403/429 باستخدام النموذج الموحد.

## **11) الأداء والكاش**

* تضمين أمثلة رؤوس الكاش (Cache-Control, ETag) في المواصفة.
* توثيق حدود pagination الافتراضية والقصوى.

## **12) المخاطر والضوابط**

* خطر انفصال المواصفة عن الواقع: يمنع بالـContract Tests الإلزامية.
* خطر تغييرات كاسرة غير معلنة: يمنع بفحص breaking changes وخطة Deprecation.

## **13) برنامج الاختبارات**

* Contract Tests: Dredd/Schemathesis ضد openapi.yaml.
* Lint: Spectral ضمن CI.
* وظيفي/E2E: Postman/Newman/Insomnia من Collections المولدة.

## **14) الوثائق وإدارة التغيير**

* نشر Swagger/Redoc مع كل إصدار.
* عند كسر التوافق: إعلان Deprecation ≥ 90 يومًا + نافذة تشغيل متوازٍ بين /v1 و /v2.

## **15) الأدوات والمراجع**

* Spectral (lint).
* Dredd/Schemathesis (اختبارات عقد).
* Stoplight/Redoc/Swagger UI.
* Postman/Insomnia.

## **16) Checklist**

* openapi.yaml محدث ويشمل جميع المسارات.
* Spectral يمر بدون أخطاء حرجة.
* Dredd/Schemathesis = 0 فروقات حرجة.
* Swagger/Redoc منشور ومتاح للفريق.
* SDKs و Postman/Insomnia Collections مولدة من OAS.
* فحص breaking changes وإرفاق خطة Deprecation عند الحاجة.

## **الملحقات التنفيذية (Artifacts)**

* Style Guide قصير للمواصفة (تسمية/ترقيم/أخطاء).
* Ruleset خاص بـ Spectral مرفوع في المستودع.
* أداة فحص breaking changes (oasdiff) ضمن CI.
* سكربت توليد SDKs من OAS ونشرها.
* قالب Error Catalog يربط أكواد الأخطاء بالـschemas.

(3) Versioning & Deprecation

# **(3) الحوكمة والإصدار — Versioning & Deprecation**

## **1) آلية العمل والاعتماديات**

* كل واجهة API لها إصدار محدد في المسار مثل: /v1/....
* عند وجود تغييرات كاسرة (Breaking Changes)، يتم فتح إصدار جديد /v2/... وتشغيله بالتوازي مع القديم لفترة انتقالية.
* تعتمد هذه السياسة على OpenAPI كمرجع للعقد، وCI/CD كأداة للتحقق من عدم وجود تغييرات غير معلنة.

## **2) الغرض والنطاق**

* الغرض: ضمان استقرار العملاء (الموبايل، الويب، الشركاء) أثناء تطور الـAPI.
* النطاق يشمل جميع الخدمات العامة والداخلية التي تُستهلك من خارج الفريق (تطبيق، لوحة تحكم، تكاملات شركاء).

## **3) المدخلات والمخرجات**

* المدخلات: المواصفة (openapi.yaml)، سجل التغييرات (Changelog)، خطة الترحيل.
* المخرجات: إصدارات API موثقة، إشعارات Deprecation، نافذة تشغيل متوازي.

## **4) المسؤوليات والأدوار**

* Backend: مسؤول عن إدارة الإصدارات داخل الكود والـspec.
* QA: مسؤول عن تشغيل اختبارات العقد على الإصدارات الجديدة.
* Product/PM: يحدد سياسة الإيقاف ومدة الـDeprecation.
* DevRel/Docs: ينشر التحديثات للمطورين والشركاء.

## **5) السياسات والمعايير**

* الإصدار يجب أن يظهر في المسار (path versioning).
* لا تغييرات كاسرة في نفس الإصدار (Backward Compatibility إلزامي).
* أي إصدار جديد يجب أن يمر بفترة انتقال ≥ 90 يومًا قبل إيقاف القديم.
* استخدام رؤوس HTTP لإشعارات الإيقاف:
  + Deprecation: true
  + Sunset: <date>

## **6) إجراءات التنفيذ**

1. عند الحاجة لتغيير كاسر → إنشاء /v2/... مع نسخة كاملة من الـspec.
2. تشغيل كل من /v1 و/v2 بالتوازي في Staging وProd.
3. توثيق التغييرات في Changelog + نشر أمثلة ترحيل.
4. إرسال إشعارات للعملاء (رسائل بريد/بوابة مطورين) مع تاريخ الإيقاف.
5. إيقاف الإصدار القديم بعد انتهاء فترة الـDeprecation.

## **7) قبول الجودة**

* أي إصدار جديد يملك توثيق كامل ومواصفة OpenAPI مستقلة.
* وجود Changelog واضح يربط التغييرات بين الإصدارات.
* نشر إشعارات Deprecation مسبقة ≥ 90 يومًا.
* اختبارات العقد تغطي كلا الإصدارين أثناء الفترة الانتقالية.

## **8) القياس والمراقبة**

* متابعة نسب استهلاك الإصدارات (v1 vs v2) عبر لوحات مراقبة.
* نسبة استهلاك الإصدار الجديد ≥ 90% قبل إيقاف القديم.
* إنذارات عند استمرار استخدام إصدار منتهي.

## **9) التكاملات**

* تكامل مع CI لفحص breaking changes (oasdiff).
* تكامل مع Gateway لإدارة المسارات /v1 و/v2.
* إشعارات آلية عبر بوابة المطورين أو Webhooks عن Deprecation.

## **10) الأمان**

* كل إصدار يحتفظ بنفس سياسات Auth/CORS/Rate.
* لا يسمح بترك إصدار قديم غير مؤمن (TLS/WAF/Secrets محدثة).

## **11) الأداء والكاش**

* يمكن تمييز الاستجابات بالإصدار في الـETag أو الـCache-Control.
* مراقبة الأداء لكل إصدار بشكل منفصل لتفادي مشاكل في الترحيل.

## **12) المخاطر والضوابط**

* خطر نسيان إشعار العملاء → ضابط: نشر Deprecation Headers + بريد + بوابة مطورين.
* خطر استمرار الاعتماد على إصدار قديم → ضابط: مراقبة الاستخدام + قطع تدريجي (Rate Limit مرتفع → متوسط → منخفض → إيقاف).

## **13) برنامج الاختبارات**

* اختبارات العقد على كل إصدار نشط.
* اختبار End-to-End للعميل (mobile/web) ضد الإصدار الجديد قبل الإطلاق.
* فحص Regression لضمان التوافق الخلفي.

## **14) الوثائق وإدارة التغيير**

* Changelog إلزامي مع كل إصدار.
* توثيق الفرق بين الإصدارات (Breaking vs Non-Breaking).
* سياسة Deprecation منشورة علنًا (مثلاً: 90 يومًا كحد أدنى).

## **15) الأدوات والمراجع**

* oasdiff لفحص التغييرات الكاسرة.
* Redoc/Swagger لتوثيق كل إصدار.
* CI/CD مع Stage مخصص لفحص الإصدارات.

## **16) تشيك**

* الإصدار الجديد معرف في المسار (/v2).
* مواصفة OpenAPI جديدة جاهزة.
* إشعارات Deprecation منشورة (Headers + Docs + Portal).
* Changelog محدث ومتوفر.
* اختبارات العقد تعمل على الإصدارين.
* مراقبة استهلاك الإصدارات فعالة.

## **الملحقات التنفيذية (Artifacts)**

* جدول سياسة Deprecation (مدة، خطوات، قنوات تواصل).
* قالب Changelog موحد (Added/Changed/Deprecated/Removed).
* سكربت CI لتوليد تقرير breaking changes عند كل PR.
* Dashboard مراقبة استهلاك الإصدارات.

(4) RESTful Resources & Endpoints

# **(4) تصميم REST للموارد — RESTful Resources & Endpoints**

## **1) آلية العمل والاعتماديات**

* الـAPI مبني على REST: الموارد (Resources) تمثل كيانات الأعمال (Products, Orders, Users…).
* العمليات تتم عبر أفعال HTTP القياسية (GET, POST, PUT, PATCH, DELETE).
* يعتمد على OpenAPI لتوثيق كل مورد ومساره، وعلى Gateway لتطبيق السياسات (CORS, Auth, Rate).

## **2) الغرض والنطاق**

* الغرض: توفير واجهات موحدة وسهلة الاستخدام ومتوافقة مع المعايير العالمية.
* النطاق: جميع الموارد الأساسية والفرعية (products, orders, customers, inventory, payments…).

## **3) المدخلات والمخرجات**

* المدخلات: طلبات HTTP بصيغة JSON.
* المخرجات: ردود JSON UTF-8 متوافقة مع المخططات (schemas).

## **4) المسؤوليات والأدوار**

* Backend: تصميم المسارات والمخططات.
* QA: التحقق من التوافق مع OpenAPI وتشغيل اختبارات العقد.
* Security: مراجعة البيانات المكشوفة في الردود.

## **5) السياسات والمعايير**

* تسمية المسارات **بصيغة الجمع**: /products, /orders.
* استخدام **kebab-case** في المسارات، و **snake\_case** في حقول JSON.
* لا يتم تضمين الأفعال في أسماء المسارات (مثلاً /getProducts ممنوع).
* المعرفات دائمًا ضمن المسار: /products/{id}.
* الترشيح عبر query params: /products?category=dress&color=red.
* الفرز عبر sort: /products?sort=-price.
* الترقيم (Pagination): Cursor أو Limit/Offset (مع توثيق الحدود القصوى).
* اختيار الحقول (Sparse Fieldsets): /products?fields=id,name,price.
* توسيع العلاقات (Expand): /orders?expand=items,customer.

## **6) إجراءات التنفيذ**

1. تحديد الموارد الأساسية والفرعية.
2. إنشاء المسارات بصيغة الجمع مع الالتزام بالNaming Convention.
3. إضافة التصفية والفرز والاختيار والتوسيع كخيارات قياسية.
4. تعريف الردود في OpenAPI بمخططات واضحة.
5. اختبار جميع العمليات CRUD لكل مورد.

## **7) قبول الجودة**

* كل مورد يحتوي على CRUD كاملة موثقة.
* وجود ترقيم، فرز، فلترة، اختيار حقول، توسيع علاقات.
* الردود تتوافق مع JSON Schema بدون أخطاء.
* أي مسار يلتزم بالNaming Convention.

## **8) القياس والمراقبة**

* متابعة أداء المسارات الأساسية (مثل /products, /orders).
* مراقبة زمن الاستجابة P95 لكل مورد.
* مراقبة استخدام الترقيم والفرز لاكتشاف queries ثقيلة.

## **9) التكاملات**

* ربط الموارد مع بوابة المطورين لتوفير أمثلة جاهزة.
* توليد Postman/Insomnia Collections من OpenAPI.

## **10) الأمان**

* لا يتم كشف بيانات حساسة في الردود (PII, أسرار).
* تحقق صارم من إدخال query params.
* RBAC لتحديد من يمكنه الوصول لكل مورد.

## **11) الأداء والكاش**

* الكاش على المسارات العامة (مثل /products).
* دعم ETag وCache-Control للطلبات GET.
* حدود قصوى (Limit Max = 100).

## **12) المخاطر والضوابط**

* خطر queries مكلفة (N+1) → حل: استخدام Expand مع ضوابط + فهارس DB.
* خطر تسريب بيانات زائدة → حل: فرض Sparse Fieldsets.

## **13) برنامج الاختبارات**

* اختبارات عقد لكل مورد.
* اختبارات وظيفية لكل عملية CRUD.
* اختبارات أداء على المسارات الأساسية.

## **14) الوثائق وإدارة التغيير**

* كل مورد موثق في OpenAPI بمخططات وأمثلة.
* أي تغيير في الموارد يوثق في Changelog.

## **15) الأدوات والمراجع**

* OpenAPI/Swagger/Redoc.
* أدوات اختبار العقد (Dredd/Schemathesis).
* Postman/Newman.

## **16) تشيك**

* جميع المسارات بصيغة الجمع.
* المعرفات في المسار وليس كـ query.
* Naming ثابت: kebab-case للمسارات، snake\_case للحقول.
* الترقيم/الفرز/الفلترة/الاختيار/التوسيع مدعومة.
* الردود JSON Schema صحيحة.
* Cache-Control وETag مفعلة للمسارات العامة.

## **الملحقات التنفيذية (Artifacts)**

* دليل Naming Convention (مسارات وحقول).
* جدول Pagination/Filtering/Sorting مع أمثلة.
* قوالب JSON Schema للموارد الأساسية (Product, Order, Customer).
* أمثلة OpenAPI جاهزة (Request/Response) لكل مورد.

(5) Error Handling & Response Shape

# **(5) نماذج الاستجابة والأخطاء — Error Handling & Response Shape**

## **1) آلية العمل والاعتماديات**

* جميع الأخطاء (4xx/5xx) تُرجع **غلاف JSON موحد**.
* يحتوي دائمًا على الحقول: code, message, details, trace\_id.
* يتم توثيق الشكل والأمثلة داخل OpenAPI.

## **2) الغرض والنطاق**

* توحيد شكل الأخطاء لتسهيل الفهم والمعالجة من قبل العملاء والتطبيقات.
* ينطبق على جميع المسارات العامة والداخلية بما فيها Webhooks.

## **3) المدخلات والمخرجات**

* المدخلات: أي طلب خاطئ أو فاشل (تحقق، صلاحيات، معدل طلبات، …).
* المخرجات: استجابة JSON + كود HTTP صحيح + رأس X-Trace-Id.

## **4) المسؤوليات والأدوار**

* فريق Backend: تنفيذ معالج أخطاء مركزي يعيد الشكل الموحّد.
* فريق QA: التحقق من مطابقة الأخطاء للمخططات عبر اختبارات العقد.
* فريق SRE: ربط الأخطاء بالـ trace\_id في اللوغ واللوحات.

## **5) السياسات والمعايير**

* استخدام شكل واحد فقط (غلاف JSON موحد أو RFC7807).
* الحقول إلزامية: code ثابت وموثق، message نص مفهوم، details للأخطاء التفصيلية، trace\_id للربط.
* عدم تضمين أي بيانات حساسة (PII) في الاستجابات.
* الالتزام بأكواد HTTP القياسية:
  + نجاح: 200 / 201 / 204
  + أخطاء عميل: 400 / 401 / 403 / 404 / 409 / 422 / 429
  + أخطاء خادم: 500 / 502 / 503

## **6) إجراءات التنفيذ**

1. إنشاء معالج أخطاء مركزي في الخدمة.
2. ربط الاستثناءات بالكود المناسب والـHTTP Status.
3. إنشاء **كتالوج أخطاء** (Error Catalog) يوثق الأكواد والرسائل.
4. تضمين trace\_id في الرأس والجسم دائمًا.

## **7) قبول الجودة**

* جميع الأخطاء تعود بالشكل الموحّد.
* لا توجد PII داخل الرسائل.
* كل خطأ موثق في OpenAPI + Error Catalog.
* اختبارات العقد تمر بدون فروقات.

## **8) القياس والمراقبة**

* تتبع معدل الأخطاء (4xx, 5xx) في اللوحات.
* مراقبة أكثر الأكواد تكرارًا (Top Errors).
* إنذار عند ارتفاع نسبة الأخطاء عن الحد المسموح.

## **9) التكاملات**

* ربط Error Catalog مع بوابة المطورين.
* أمثلة أخطاء جاهزة في Postman/Insomnia.

## **10) الأمان**

* منع أي كشف بيانات حساسة داخل الرسائل.
* إرجاع أكواد مناسبة تمنع التسريب (لا تستخدم 500 عام دائمًا).

## **11) الأداء والكاش**

* استجابات الأخطاء لا يتم تخزينها في الكاش.
* يجب أن تحتوي على رأس Cache-Control: no-store.

## **12) المخاطر والضوابط**

* خطر اختلاف الشكل بين الخدمات → حل: معالج أخطاء مركزي + اختبار عقد إلزامي.
* خطر تضارب الأكواد أو ازدواجها → حل: كتالوج رسمي للأخطاء.

## **13) برنامج الاختبارات**

* اختبارات العقد للتحقق من الشكل.
* اختبارات وظيفية للأخطاء المتوقعة (صلاحيات، تحقق…).
* اختبارات أداء: قياس سرعة إرجاع الخطأ في P95.

## **14) الوثائق وإدارة التغيير**

* توثيق جميع الأخطاء في OpenAPI.
* تحديث Changelog عند إضافة/تغيير كود خطأ.

## **15) الأدوات والمراجع**

* OpenAPI + Swagger/Redoc لعرض أمثلة الأخطاء.
* Dredd/Schemathesis لاختبارات العقد.
* Postman/Newman لمحاكاة سيناريوهات الخطأ.

## **16) تشيك**

* جميع الأخطاء ترجع JSON موحد.
* الأكواد ثابتة وموثقة.
* لا يوجد PII في الرسائل.
* Error Catalog محدث.
* OpenAPI يحتوي على أمثلة أخطاء.
* Cache-Control: no-store مضاف للأخطاء.

## **الملحقات التنفيذية (Artifacts)**

* **كتالوج أخطاء رسمي** (Error Catalog) مع كود/رسالة/HTTP Status/تفاصيل.
* **قالب JSON Error** جاهز للاستخدام في جميع الخدمات.
* **قائمة أكواد HTTP المسموحة** مرتبطة بالـError Catalog.

(6) Authentication & Authorization

# **(6) الهوية والصلاحيات — Authentication & Authorization**

## **1) آلية العمل والاعتماديات**

* الاعتماد الأساسي على **JWT قصير العمر** (Access Token) + **Refresh Token** للتجديد.
* تكامل مع **OAuth2 Client Credentials** لتطبيقات الشركاء أو التكاملات الآلية.
* الصلاحيات تُدار عبر **RBAC** (Role-Based Access Control) أو **Scopes**.
* جميع الأسرار تُخزّن في **Secret Manager** وليس داخل الكود أو الملفات.

## **2) الغرض والنطاق**

* حماية جميع موارد الـAPI من الوصول غير المصرح.
* ضمان أن كل طلب له هوية مرتبطة ويمكن تتبعها.
* النطاق يشمل: تطبيق الموبايل، لوحة التحكم، بوابة الشركاء، وأي خدمة داخلية تحتاج وصول.

## **3) المدخلات والمخرجات**

* المدخلات: بيانات اعتماد (credentials)، رموز JWT، أو مفاتيح API للشركاء.
* المخرجات: استجابة مصادقة ناجحة مع Token أو خطأ موحّد عند الفشل.

## **4) المسؤوليات والأدوار**

* Backend: تنفيذ التدفقات (login، token refresh، revocation).
* Security: إدارة الأسرار والمفاتيح، مراجعة سياسات الوصول.
* QA: اختبار التدفقات الصحيحة والخاطئة.

## **5) السياسات والمعايير**

* **JWT قصير العمر** (مثلاً 15 دقيقة) لتقليل المخاطر.
* **Refresh Token** أطول عمرًا مع إمكانية الإلغاء (revocation).
* **OAuth2** لتكامل الشركاء، باستخدام Client ID + Secret.
* **RBAC/Scopes** لتعريف صلاحيات دقيقة (admin, support, customer).
* لا يسمح أبداً بإرسال كلمة مرور في الاستجابات أو اللوغ.

## **6) إجراءات التنفيذ**

1. إعداد خادم هوية (Identity Provider) أو وحدة Auth مدمجة.
2. تعريف الأدوار (Roles) والصلاحيات (Scopes).
3. إنشاء نقاط نهاية:
   * /auth/login (الحصول على JWT + Refresh).
   * /auth/refresh (تجديد الـJWT).
   * /auth/logout (إلغاء الـRefresh Token).
4. تفعيل تحقق JWT في كل مسار محمي عبر Middleware.
5. ربط RBAC/Scopes مع الموارد (مثلاً: admin فقط يمكنه /users).

## **7) قبول الجودة**

* كل مسار محمي لا يقبل إلا طلبات مع JWT صالح.
* أي طلب بدون صلاحية يُرجع خطأ 401 أو 403.
* كل دور/Scope يعمل كما هو معرف في المواصفة.

## **8) القياس والمراقبة**

* مراقبة محاولات تسجيل الدخول الفاشلة.
* نسبة استخدام الرموز المنتهية.
* اكتشاف محاولات brute force أو إساءة استخدام.

## **9) التكاملات**

* OAuth2 Client Credentials لتكامل الشركاء.
* دعم API Keys فقط في حالات الاستخدام المحدودة وبصلاحيات مقيدة.

## **10) الأمان**

* تخزين جميع الأسرار في Secret Manager.
* تشفير الاتصالات عبر TLS.
* تدوير المفاتيح بشكل دوري (كل 90–180 يومًا).
* حماية Refresh Tokens من التسريب (HTTP-only, Secure Cookies عند الحاجة).

## **11) الأداء والكاش**

* التحقق من JWT يتم محليًا عبر مفاتيح عامة (Public Keys).
* توزيع المفاتيح عبر JWKS Endpoint لزيادة الكفاءة.

## **12) المخاطر والضوابط**

* خطر سرقة Refresh Token → ضابط: إلغاء فوري عند الشك + تخزين آمن.
* خطر إساءة استخدام API Keys → ضابط: تحديد صلاحيات، معدل طلبات منخفض، مراقبة.

## **13) برنامج الاختبارات**

* اختبارات وظيفية لتدفق login/refresh/logout.
* اختبارات أمنية: brute force، token replay.
* اختبارات عقد: استجابات الخطأ (401/403).

## **14) الوثائق وإدارة التغيير**

* جميع التدفقات موثقة في OpenAPI.
* توثيق أمثلة Auth في Swagger/Redoc.
* تحديث الوثائق عند إضافة Role/Scope جديد.

## **15) الأدوات والمراجع**

* OAuth2 / OpenID Connect.
* JWT Libraries (Auth0, Keycloak, Laravel Passport).
* Secret Manager (AWS, GCP, HashiCorp Vault).

## **16) تشيك**

* جميع المسارات محمية بـJWT أو OAuth2.
* Refresh Tokens قابلة للإلغاء.
* RBAC/Scopes معرفة ومطبقة.
* Secret Manager مستخدم لتخزين الأسرار.
* تدوير المفاتيح دوري.
* لا يتم إرجاع أي بيانات اعتماد في الاستجابات أو اللوغ.

## **الملحقات التنفيذية (Artifacts)**

* جدول Roles/Scopes مع الصلاحيات المسموح بها.
* قالب JWT Payload (الحقول الأساسية: sub, role, exp, iat).
* سكربت CI يتحقق من صحة تكوين JWKS ومدة المفاتيح.
* سياسة مكتوبة لتدوير الأسرار والمفاتيح.

(7) Security Hardening

# **(7) الأمان الصلب — Security Hardening**

## **1) آلية العمل والاعتماديات**

* جميع الطلبات تمر عبر **TLS** (HTTPS فقط).
* يتم تطبيق **WAF** (Web Application Firewall) و**Rate Limiting** على مستوى الـGateway.
* **CORS** مفعّل باستخدام قائمة بيضاء (Allowlist) للمصادر الموثوقة فقط.
* يتم فرض **Idempotency-Key** في العمليات الحرجة (مثل الطلبات والدفع).
* التحقق الصارم من المدخلات عبر Validation على جميع الحقول.

## **2) الغرض والنطاق**

* حماية الـAPI من الهجمات الشائعة (Injection, XSS, CSRF, DDoS).
* منع إساءة الاستخدام (Spam, Bots, Brute Force).
* ينطبق على جميع البيئات (dev, staging, prod) مع تفعيل أقوى في prod.

## **3) المدخلات والمخرجات**

* المدخلات: جميع الطلبات الواردة (Headers, Params, Body).
* المخرجات: رد آمن بدون أي تسريب بيانات حساسة.

## **4) المسؤوليات والأدوار**

* Security Team: وضع السياسات ومراجعة الإعدادات.
* Backend: الالتزام بالتحقق من المدخلات.
* SRE: مراقبة تطبيق WAF وRate Limits.

## **5) السياسات والمعايير**

* TLS إلزامي (رفض HTTP العادي).
* WAF مفعل بقواعد OWASP ModSecurity.
* CORS يسمح فقط للنطاقات المحددة (web app, mobile app).
* Idempotency-Key إلزامي في POST للمدفوعات والطلبات.
* إدخال البيانات يخضع للتحقق (validation) والـSanitization.

## **6) إجراءات التنفيذ**

1. تفعيل HTTPS مع شهادات TLS محدثة وتجديد تلقائي.
2. ضبط WAF في Cloudflare أو Gateway محلي (NGINX/Kong).
3. تحديد قائمة CORS (النطاقات فقط).
4. إضافة Middleware للتحقق من Idempotency-Key.
5. إضافة طبقة Validation على مستوى API Layer لكل حقول JSON.

## **7) قبول الجودة**

* أي طلب غير HTTPS يتم رفضه.
* أي مصدر غير مسموح في CORS يتم رفضه.
* أي POST بدون Idempotency-Key في العمليات الحرجة يتم رفضه.
* أي مدخل غير صحيح يتم رفضه مع خطأ 400.

## **8) القياس والمراقبة**

* تتبع عدد الطلبات المحظورة عبر WAF.
* مراقبة معدلات 429 (Rate Limit).
* مراقبة عدد هجمات brute force/requests الفاشلة.

## **9) التكاملات**

* تكامل مع Cloudflare لحماية DDoS.
* تكامل مع أدوات Security Scanners (OWASP ZAP, Burp Suite).

## **10) الأمان**

* جميع الأسرار مخزنة في Secret Manager.
* مراجعة دورية للأذونات والمفاتيح.
* تفعيل سياسة تدوير المفاتيح كل 90–180 يومًا.

## **11) الأداء والكاش**

* الكاش يُستخدم فقط للبيانات العامة، ليس للأخطاء أو بيانات حساسة.
* استخدام Redis بدلاً من الاعتماد على Headers مخصصة.

## **12) المخاطر والضوابط**

* خطر هجمات DDoS → ضابط: Cloudflare WAF + Rate Limit.
* خطر تسريب بيانات → ضابط: عدم تضمين PII في الأخطاء.
* خطر هجمات إدخال (Injection) → ضابط: Validation صارم على الحقول.

## **13) برنامج الاختبارات**

* اختبارات أمنية دورية (OWASP Top 10).
* اختبارات اختراق داخلية/خارجية.
* اختبار تحميل (Load Test) لقياس مقاومة DDoS.

## **14) الوثائق وإدارة التغيير**

* سياسة أمنية منشورة للفريق.
* سجل تغييرات الأمان (تحديث CORS، مفاتيح، أسرار).
* مراجعة أمان كل 3 أشهر.

## **15) الأدوات والمراجع**

* Cloudflare WAF, NGINX/Kong Plugins.
* OWASP Top 10 Checklist.
* OWASP ZAP, Burp Suite.

## **16) تشيك**

* TLS مفعّل بالكامل.
* WAF مفعّل ومحدّث.
* CORS يسمح فقط بالنطاقات المعتمدة.
* Idempotency-Key إلزامي في POST الحرجة.
* Validation مفعل لجميع المدخلات.
* لا يتم تسريب أي بيانات حساسة في الردود.

## **الملحقات التنفيذية (Artifacts)**

* جدول CORS Allowed Origins (web, mobile).
* جدول Rate Limits لكل مسار.
* قالب Middleware لفحص Idempotency-Key.
* خطة مراجعة دورية للأسرار والمفاتيح.

(8) Performance & Caching

# **(8) الأداء والتخزين المؤقت — Performance & Caching**

## **1) آلية العمل والاعتماديات**

* تحديد أهداف أداء واضحة (SLOs) لكل عملية قراءة وكتابة.
* استخدام التخزين المؤقت (Caching) بعدة مستويات:
  + **Edge/Gateway** باستخدام رؤوس HTTP (Cache-Control).
  + **Redis** للبيانات المتكررة أو الثقيلة.
  + **ETag و If-None-Match** للطلبات GET لتقليل النقل.
  + **If-Match** مع PATCH لمنع التعارض في التحديثات.

## **2) الغرض والنطاق**

* تحسين زمن الاستجابة وتجربة المستخدم.
* تقليل الحمل على قواعد البيانات والخوادم.
* ينطبق على المسارات الأكثر استخدامًا (مثل قائمة المنتجات، تفاصيل منتج).

## **3) المدخلات والمخرجات**

* المدخلات: طلبات القراءة والكتابة للموارد.
* المخرجات: استجابة أسرع مع بيانات مخزنة مؤقتًا عند توفرها.

## **4) المسؤوليات والأدوار**

* Backend: إضافة الكاش في المستوى المناسب وتفعيل الـETag.
* SRE/Infra: إدارة Redis/Edge Cache ومراقبة الأداء.
* QA: اختبار سلوك الكاش (GET مكرر، تحديث مع If-Match).

## **5) السياسات والمعايير**

* **SLO**: زمن استجابة p95 للقراءة ≤ 300ms، للكتابة ≤ 800ms.
* أقصى قيمة للـlimit في الترقيم = 100.
* جميع GET العامة يجب أن تدعم Cache-Control + ETag.
* الأخطاء لا يتم تخزينها (Cache-Control: no-store).

## **6) إجراءات التنفيذ**

1. تحديد المسارات الأكثر استخدامًا.
2. تفعيل الكاش على GET العامة عبر رؤوس HTTP.
3. تفعيل Redis للقوائم الثقيلة (مثلاً /products).
4. إضافة ETag على GET و If-Match على PATCH.
5. مراقبة مؤشرات الأداء (latency, error rate).

## **7) قبول الجودة**

* زمن استجابة p95 ضمن الهدف المحدد.
* استجابات GET المكررة تُرجع 304 Not Modified عند عدم التغيير.
* تحديث بدون If-Match يُرفض بخطأ 412.
* Redis يعمل ويقلل الحمل على DB.

## **8) القياس والمراقبة**

* مراقبة زمن الاستجابة (p50/p95).
* مراقبة نسب cache hit/miss.
* إنذارات عند تجاوز SLO أو انخفاض الكفاءة.

## **9) التكاملات**

* تكامل مع CDN/Cloudflare لتوزيع الكاش على مستوى الحافة.
* تكامل مع أدوات المراقبة (Prometheus/Grafana أو Datadog).

## **10) الأمان**

* البيانات الحساسة أو الخاصة لا تُخزّن في الكاش العام.
* استخدام Cache-Control: private أو no-store عند الحاجة.

## **11) المخاطر والضوابط**

* خطر بيانات قديمة (Stale) → حل: تقليل TTL وتحديث بعد التغييرات الحرجة.
* خطر تخزين بيانات حساسة بالخطأ → حل: قوائم استثناء واضحة.

## **12) برنامج الاختبارات**

* اختبار GET مكرر → الأول 200 والثاني 304.
* اختبار PATCH بدون If-Match → خطأ 412.
* اختبار تحميل (Load Test) للتأكد من استقرار الأداء.

## **13) الوثائق وإدارة التغيير**

* توثيق سياسات الكاش لكل مسار في OpenAPI.
* تحديث Changelog عند تغيير TTL أو سياسة الكاش.

## **14) الأدوات والمراجع**

* Redis كذاكرة تخزين سريعة.
* k6/Artillery لاختبارات الأداء.
* أدوات مراقبة مثل Prometheus/Grafana.

## **15) تشيك**

* تحديد SLOs ومشاركتها مع الفريق.
* Cache-Control و ETag مفعّلين على جميع GET العامة.
* If-Match مفعّل على عمليات PATCH.
* Redis مستخدم للمسارات الثقيلة.
* تقارير p95 و 5xx ضمن الحدود.
* اختبارات الأداء مدمجة في خط CI/CD.

## **الملحقات التنفيذية (Artifacts)**

* جدول يحدد لكل مسار: هل يُخزَّن؟ أين؟ وما مدة TTL؟
* أمثلة رؤوس كاش في OpenAPI (Cache-Control, ETag).
* سكربت اختبارات أداء جاهز (k6 أو Artillery).

(9) Uploads & Signed URLs

# **(9) الملفات والروابط الموقَّعة — Uploads & Signed URLs**

## **1) آلية العمل والاعتماديات**

* الرفع يتم عبر واجهات multipart/form-data إلى خدمة التخزين (S3 أو ما يعادلها) بمسار مؤقت أو مباشر.
* التنزيل يتم عبر روابط موقَّعة قصيرة العمر (Signed URLs) بصلاحيات محددة ونطاق زمني مضبوط.
* يعتمد على الطبقة الأمنية في الـGateway (TLS, Auth, Rate) وعلى إدارة صلاحيات التخزين (IAM/Policies).

## **2) الغرض والنطاق**

* تمكين رفع/تنزيل آمن وفعّال للصور والمرفقات دون كشف الخادم للأصول الحساسة.
* ينطبق على: صور المنتجات، فواتير، وثائق شحن، مرفقات دعم فني، وأي ملفات عميل/تاجر.

## **3) المدخلات والمخرجات**

* المدخلات: ملف واحد أو عدة ملفات (حجم/نوع محدد)، طلب إنشاء رابط موقَّع.
* المخرجات:
  + عند الرفع: كائن Metadata (المسار، الحجم، النوع، checksum، معرّف).
  + عند التنزيل: رابط موقَّع بزمن انتهاء محدد.

## **4) المسؤوليات والأدوار**

* Backend: توليد الروابط الموقَّعة والتحقق من الميتاداتا والـMIME.
* SRE/Infra: سياسات التخزين والـIAM، ضبط دورة حياة الملفات (Lifecycle).
* Security: فحص الامتدادات وسياسات المحتوى الآمن ومنع التنفيذ.
* QA: اختبار القيود (أحجام، أنواع، انتهاء صلاحية الرابط).

## **5) السياسات والمعايير**

* أحجام قصوى واضحة لكل نوع (مثلاً: صور ≤ 5MB، وثائق ≤ 10MB).
* أنواع مسموحة فقط (قائمة بيضاء: jpg/png/webp/pdf).
* روابط موقَّعة قصيرة العمر (مثلاً: 5–15 دقيقة) وبصلاحيات أقل ما يمكن (read أو write فقط).
* ملفات المستخدم الخاصّة لا تُخزّن في كاش عام؛ استخدام Cache-Control: private أو no-store حسب الحساسية.
* حفظ checksum (مثل SHA-256) والتحقق منه عند الرفع (سلامة الملف).
* منع امتدادات خطرة (مثل .exe, .js في مسارات قابلة للتنفيذ).
* أسماء ملفات نظيفة وآمنة (توليد اسم داخلي أو استخدام UUID/ULID).

## **6) إجراءات التنفيذ**

1. نقطة نهاية للحصول على رابط رفع موقَّع: يَستقبل النوع والحجم المتوقع ويُرجع URL + headers المطلوبة.
2. العميل يرفع الملف مباشرة للتخزين باستخدام الرابط الموقَّع.
3. نقطة نهاية لتأكيد الرفع: تُخزّن الميتاداتا (النوع، الحجم، checksum، المالك، الصلاحيات).
4. نقطة نهاية لإنشاء رابط تنزيل موقَّع قصير العمر عند الحاجة.
5. تطبيق سياسات Lifecycle: أرشفة/حذف تلقائي بعد مدد محددة للفئات.

## **7) قبول الجودة**

* رفض أي ملف خارج النوع/الحجم المسموح به.
* الروابط الموقَّعة تنتهي صلاحيتها وفق الزمن المحدد ولا تعمل بعده.
* لكل ملف Metadata مكتمل (المسار، المالك، الأذونات).
* لا توجد ملفات يتيمة (بدون تسجيل) في التخزين.

## **8) القياس والمراقبة**

* معدلات نجاح/فشل الرفع، أحجام إجمالية، أكثر الأنواع استخدامًا.
* إنذارات عند ارتفاع أخطاء 4xx/5xx في الرفع أو زمن استجابة غير طبيعي.
* مراقبة استهلاك التخزين والتكلفة لكل فئة ملفات.

## **9) التكاملات**

* فحص فيروسات/برمجيات خبيثة (إن أمكن) بخط معالجة غير متزامن.
* سياسة CDN للملفات العامة فقط (صور المنتجات)، مع احترام الكاش والـETag.
* تكامل مع نظام صلاحيات التطبيق (المالك، فرق العمل، الشركاء).

## **10) الأمان**

* جميع الروابط عبر HTTPS فقط.
* الروابط الموقَّعة للقراءة لا تمنح كتابة والعكس صحيح.
* تقييد النطاق (Bucket/Path) في التوقيع لمنع الوصول خارج المسار المصرّح به.
* إزالة أي EXIF حسّاس من الصور عند الحاجة (الخصوصية).

## **11) الأداء والكاش**

* صور المنتجات العامة: Cache-Control: public, max-age=… + ETag، مع تحديث الـETag عند التبديل.
* مرفقات العملاء الخاصة: Cache-Control: private, no-store أو private, max-age=0.
* ضغط تلقائي للصور (WebP/AVIF) عبر مسار معالجة إن تطلّب.

## **12) المخاطر والضوابط**

* خطر رفع ملفات ضارة: فحص MIME/الامتداد + قيود قائمة بيضاء + فحص مضاد للبرمجيات.
* خطر كشف روابط طويلة العمر: فرض مدة قصيرة وتجديد عند الطلب.
* خطر بقاء ملفات قديمة: سياسات Lifecycle للحذف/الأرشفة.

## **13) برنامج الاختبارات**

* اختبارات وظيفية: رفع ملف صالح/مرفوض، تنزيل قبل/بعد انتهاء الصلاحية.
* اختبارات أمنية: محاولة رفع نوع محظور/حجم زائد، وصول عبر رابط منتهي.
* اختبارات أداء: قياس زمن رفع/تنزيل لملفات بأحجام نموذجية.

## **14) الوثائق وإدارة التغيير**

* توثيق نقاط النهاية للروابط الموقَّعة والأمثلة في OpenAPI.
* توثيق أنواع الملفات المسموحة والحدود القصوى وسياسات الكاش في بوابة المطوّرين.
* تسجيل أي تغيير في القيود أو الأعمار في سجل التغييرات.

## **15) الأدوات والمراجع**

* خدمات التخزين: Amazon S3 أو مكافئ (GCS, Azure Blob).
* توليد التوقيع: SDK الرسمي لمزوّد التخزين.
* معالجة صور: خدمة تحويل/ضغط إن لزم.

## **16) تشيك**

* قائمة بيضاء للأنواع + حدود أحجام ثابتة.
* روابط موقَّعة قصيرة العمر ومقيّدة الصلاحيات.
* Metadata محفوظة لكل ملف (مالك/نوع/حجم/checksum/مسار).
* سياسات كاش مميزة بين العام والخاص.
* سياسات Lifecycle مفعّلة للتنظيف والأرشفة.
* اختبارات رفع/تنزيل/انتهاء صلاحية تعمل بنجاح.

## **الملحقات التنفيذية (Artifacts)**

* مصفوفة أنواع الملفات والأحجام القصوى وقيم TTL للرابط الموقَّع.
* قوالب استجابات OpenAPI لنقاط: إنشاء رابط رفع، تأكيد الرفع، إنشاء رابط تنزيل.
* سكربت تنظيف دوري لملفات يتيمة أو منتهية الصلاحية.

(10) Asynchronous Jobs

# **(10) المهام غير المتزامنة — Asynchronous Jobs**

## **1) آلية العمل والاعتماديات**

* أي عملية طويلة (توليد تقرير، معالجة صورة، فحص بيانات…) لا تُنفَّذ مباشرة بل توضع في **Queue** (RabbitMQ, SQS, Redis Queue).
* عند استلام الطلب: الخادم يُرجع **HTTP 202 Accepted** مع رأس Location يشير إلى مورد /jobs/{id}.
* العميل يستعلم عن حالة المهمة عبر /jobs/{id} حتى تكتمل أو تفشل.

## **2) الغرض والنطاق**

* تجنب حجب العميل أو تجاوزه مهلة الشبكة (Timeout).
* تحسين تجربة المستخدم عبر تتبع تقدم المهام.
* ينطبق على العمليات الثقيلة مثل: إنشاء تقارير، مزامنة بيانات خارجية، ضغط أو تحويل ملفات.

## **3) المدخلات والمخرجات**

* المدخلات: طلب API لإنشاء مهمة (مثلاً /reports أو /bulk-upload).
* المخرجات:
  + عند الإنشاء: استجابة 202 + Location: /jobs/{id}.
  + عند الاستعلام: JSON يحتوي id, status, progress, result\_url أو error.

## **4) المسؤوليات والأدوار**

* Backend: إنشاء المهام وتخزين حالتها في DB/Redis.
* Worker Service: تنفيذ المهمة وتحديث حالتها.
* QA: اختبار تدفق الإنشاء → الاستعلام → النتيجة.
* SRE: مراقبة استهلاك الـQueue وأداء الـWorkers.

## **5) السياسات والمعايير**

* الحالات الأساسية: pending, in\_progress, completed, failed.
* progress يُعبر بنسبة مئوية أو مراحل واضحة.
* النتيجة الناجحة تُشير إلى رابط/مورد جديد (مثلاً تقرير جاهز).
* المهام الفاشلة تُرجع خطأ موحّد في error.

## **6) إجراءات التنفيذ**

1. إنشاء نقطة نهاية (POST) تُنشئ مهمة وتُرجع 202 + Location.
2. تسجيل المهمة في DB/Redis مع الحالة pending.
3. Worker يلتقط المهمة من Queue → ينفذها → يحدّث الحالة.
4. العميل يستعلم عبر GET /jobs/{id}.
5. عند انتهاء المهمة → تحديث الحالة إلى completed أو failed مع النتيجة.

## **7) قبول الجودة**

* أي مهمة تُرجع 202 مع Location صحيح.
* /jobs/{id} يعرض الحالة الحالية دائمًا.
* الحالات محدثة بشكل متسق ولا تبقى معلّقة.
* جميع الأخطاء تُوثق وتُرجع بشكل موحّد.

## **8) القياس والمراقبة**

* زمن المعالجة المتوسط لكل نوع مهمة.
* عدد المهام الفاشلة مقابل الناجحة.
* معدل استهلاك Queue (processing rate).
* إنذار عند تراكم المهام أو تجاوز SLA.

## **9) التكاملات**

* تكامل مع إشعارات Webhooks لإبلاغ العميل عند انتهاء المهمة بدل الاستعلام المتكرر.
* تكامل مع نظام التنبيهات الداخلي عند فشل المهام الحرجة.

## **10) الأمان**

* لا يمكن الوصول إلى /jobs/{id} إلا بواسطة المالك أو من لديه الصلاحية.
* البيانات الحساسة داخل نتيجة المهمة تُخزن مؤقتًا مع صلاحيات محددة.
* صلاحية روابط النتائج قصيرة العمر (مثل Signed URL).

## **11) الأداء والكاش**

* تخزين نتائج المهام المكتملة مؤقتًا في Redis أو DB لتقليل الحمل.
* نتائج قديمة يتم تنظيفها بشكل دوري (Retention Policy).

## **12) المخاطر والضوابط**

* خطر تراكم المهام في Queue → مراقبة + Auto Scaling للـWorkers.
* خطر فقدان المهمة عند فشل Worker → سياسة إعادة المحاولة (Retry with backoff).
* خطر ازدواج التنفيذ → Idempotency-Key للمهمة.

## **13) برنامج الاختبارات**

* اختبار تدفق كامل: إنشاء مهمة → استعلام متكرر → اكتمال.
* اختبار مهمة تفشل وتُرجع خطأ.
* اختبار سيناريوهات تحميل (آلاف المهام بالتوازي).

## **14) الوثائق وإدارة التغيير**

* توثيق /jobs في OpenAPI مع حالات status والأمثلة.
* تحديث Changelog عند إضافة نوع مهمة جديدة.

## **15) الأدوات والمراجع**

* Queues: RabbitMQ, Kafka, AWS SQS, Redis Queue.
* Workers: Laravel Queues, Sidekiq, Celery.
* مراقبة: Grafana, Prometheus, Datadog.

## **16) تشيك**

* جميع العمليات الطويلة تستخدم Queue.
* استجابة الإنشاء = 202 + Location صحيح.
* حالات pending/in\_progress/completed/failed معرفة ومستخدمة.
* الأخطاء موحدة في الحقل error.
* مراقبة أداء Queue والـWorkers فعّالة.
* نتائج قديمة يتم تنظيفها بشكل دوري.

## **الملحقات التنفيذية (Artifacts)**

* قالب JSON استجابة /jobs/{id} مع الحقول الأساسية.
* جدول SLA لكل نوع مهمة (المدة المتوقعة).
* سكربت تنظيف دوري للمهام المنتهية أو القديمة.

(11) Webhooks — Integrations & Webhooks

# **(11) التكاملات والـWebhooks — Integrations & Webhooks**

## **1) آلية العمل والاعتماديات**

* النظام ينشر أحداثًا (events) عند تغيّر الحالة (مثل: order.created, order.paid, stock.adjusted).
* الإرسال يتم عبر Webhooks إلى عناوين URL يحددها الشريك/النظام المتكامل.
* كل حدث يُوقَّع بتوقيع HMAC مع طابع زمني لمنع التلاعب وإعادة الإرسال الخبيث.
* توجد بيئتان للتكامل: Sandbox (اختبار) و Production (حي).

## **2) الغرض والنطاق**

* تمكين الأنظمة الخارجية من التفاعل فورًا دون الاستطلاع (polling).
* دعم تكاملات الشركاء، المحاسبة، المستودعات، أنظمة الإشعارات.

## **3) المدخلات والمخرجات**

* المدخلات: تكوين Webhook (العنوان، الأحداث، السر).
* المخرجات: POST إلى عنوان المستقبِل يتضمن JSON للحدث + رؤوس التوقيع + الطابع الزمني.
* المستقبِل يعيد 2xx للاعتراف (ack). غير ذلك يُعد فشلًا ويؤدي لإعادة المحاولة.

## **4) المسؤوليات والأدوار**

* Backend: توليد الأحداث، التوقيع، الإرسال، تسجيل المحاولات.
* Integrations/Partner: استلام/تحقق/معالجة الحدث وإرجاع 2xx سريع.
* SRE/Security: إدارة الأسرار، الدوران (rotation)، المراقبة، قوائم السماح (IP Allowlist إن لزم).

## **5) السياسات والمعايير**

* التوقيع: X-Signature (HMAC-SHA256 على الجسم مع سر مشترك) + X-Timestamp.
* الحماية من إعادة التشغيل (Replay): رفض الطلب إذا انقضى على الطابع الزمني أكثر من نافذة محددة (مثل 5 دقائق).
* منع الازدواج: event\_id فريد يجب ألا يُعالج أكثر من مرة لدى المستقبِل.
* حجم الجسم الأقصى محدد (مثلاً ≤ 256KB)؛ لما هو أكبر تُرسل روابط لاسترجاع التفاصيل عبر API.
* زمن استجابة المستقبِل ≤ 5 ثوانٍ؛ المعالجة الطويلة تُنقل داخليًا لديه.

## **6) إجراءات التنفيذ**

1. إنشاء/تحديث تكوين Webhook: URL، الأسرار، الأحداث، الوضع (sandbox/prod).
2. عند وقوع الحدث: إنشاء Payload وفق المخطط المتفق عليه + إضافة event\_id, type, created\_at.
3. حساب التوقيع HMAC وإرساله مع الرؤوس.
4. انتظار رد المستقبِل:
   * 2xx → نجاح.
   * 4xx/5xx أو مهلة → إعادة محاولة وفق سياسة backoff.
5. تسجيل كل محاولة مع الحالة والأزمنة لأغراض التدقيق والمراقبة.

## **7) قبول الجودة**

* إرسال صحيح مع رؤوس التوقيع والطابع الزمني.
* إعادة محاولة تلقائية عند الفشل حتى حد أقصى، ثم نقل إلى Dead-Letter Queue.
* إمكانية إعادة الإرسال اليدوي (replay) من لوحة المراقبة لحدث محدد.
* Sandbox يعمل بنفس المخطط مع أسرار ووجهات منفصلة.

## **8) القياس والمراقبة**

* معدل نجاح التسليم، زمن الاستجابة المتوسط، أعلى المستلمين خطأً.
* عدد الأحداث في الانتظار/قائمة الرسائل المتعثرة (DLQ).
* إنذارات عند ارتفاع الفشل فوق عتبة محددة أو تجاوز زمن الاستجابة.

## **9) التكاملات**

* تكامل مع بوابة المطورين: إدارة الأسرار، إضافة/تعطيل Webhooks، عرض السجلات وإعادة الإرسال.
* تكامل مع API العامة لتمكين المستقبِل من جلب تفاصيل إضافية عند الحاجة (pattern: send minimal → fetch detail).

## **10) الأمان**

* HTTPS إلزامي؛ يُفضَّل mTLS للتكاملات الحساسة.
* دوران الأسرار دوريًا (كل 90–180 يومًا) مع دعم مفاتيح متعددة خلال فترة الانتقال.
* خيار IP Allowlist على مستوى الـGateway.
* التحقق من X-Timestamp ضمن نافذة زمنية قصيرة.

## **11) الأداء والكاش**

* لا كاش لاستدعاءات Webhook.
* على المستقبِل إرجاع 2xx سريعًا ثم المعالجة داخليًا (Queue)، لتجنب إعادة المحاولات غير الضرورية.

## **12) المخاطر والضوابط**

* خطر المعالجة المكررة → فرض idempotency في نظام المستقبِل باستخدام event\_id.
* خطر تسريب السر → تدوير فوري وتجميد الإرسال حتى التحقق.
* خطر حلقات الاستدعاء (loop) → تتبّع source system وإسقاط الأحداث المرتدة.

## **13) برنامج الاختبارات**

* اختبارات وظيفية لكل نوع حدث (payload + توقيع).
* اختبارات فشل: 500/timeout/شبكة → التأكد من backoff وDLQ.
* اختبارات أمن: توقيع غير صحيح/طابع زمني منتهي → رفض.

## **14) الوثائق وإدارة التغيير**

* توثيق مخططات الأحداث، الحقول الإلزامية، والرؤوس.
* سياسة إصدارات للأحداث: event.version داخل الجسم (مثلاً 1.0)، مع سجل تغييرات.
* أمثلة كاملة بالطلبات والاستجابات، وسيناريوهات فشل/نجاح.

## **15) الأدوات والمراجع**

* مكتبات HMAC متوافرة في جميع اللغات.
* DLQ عبر SQS/Kafka/Redis Streams حسب البنية.
* لوحات مراقبة (Grafana/Datadog) لمعدلات التسليم والأخطاء.

## **16) تشيك**

* تم تفعيل HTTPS والتوقيع HMAC + X-Timestamp.
* نافذة رفض إعادة التشغيل محددة (مثلاً 5 دقائق).
* سياسة إعادة المحاولة: backoff مع jitter + حد أقصى للمحاولات (مثلاً 8).
* DLQ مفعلة وإعادة إرسال يدوي متاح.
* event\_id فريد ومعرّف الإصدار داخل payload.
* Sandbox/Prod مفصولان بأسرار ووجهات مختلفة.

## **الملحقات التنفيذية (Artifacts)**

* قالب JSON موحّد للحدث: { event\_id, type, version, created\_at, data }.
* مثال توقيع HMAC: حسابه على الجسم الكامل مع X-Timestamp.
* مصفوفة إعادة المحاولة: أزمنة المحاولات (على سبيل المثال: 10s, 30s, 2m, 5m, 15m, 30m, 1h, 2h).
* جدول إدارة الأسرار: حقول، تواريخ الإنشاء/الدوران/الانتهاء، وحالة التفعيل.

(12) Data & Schema Standards

# **(12) معايير البيانات والمخططات — Data & Schema Standards**

## **1) آلية العمل والاعتماديات**

* جميع البيانات المرسلة والمستقبلة عبر API تتبع مخططات (JSON Schemas) معرفة مسبقًا داخل OpenAPI.
* التواريخ والأوقات تستخدم **ISO-8601** مع المنطقة الزمنية UTC.
* المعرفات (IDs) تستخدم **UUID/ULID** وليس أرقامًا متسلسلة.
* القيم المالية تحفظ دائمًا في أصغر وحدة عملة (مثال: سنت، هللة).

## **2) الغرض والنطاق**

* ضمان اتساق البيانات بين الأنظمة المختلفة.
* منع الأخطاء الناتجة عن اختلاف التنسيقات.
* ينطبق على جميع الموارد (Products, Orders, Users, Payments…).

## **3) المدخلات والمخرجات**

* المدخلات: بيانات الطلب من العميل.
* المخرجات: بيانات الاستجابة بنفس المعايير (التاريخ، المعرفات، القيم المالية).

## **4) المسؤوليات والأدوار**

* Backend: الالتزام بالمخططات أثناء التطوير.
* QA: اختبار الطلبات والردود للتحقق من الالتزام بالمخططات.
* Data/Analytics: استخدام نفس المعايير عند تخزين البيانات في المستودع.

## **5) السياسات والمعايير**

* التواريخ بصيغة YYYY-MM-DDTHH:MM:SSZ (UTC دائمًا).
* المبالغ المالية تحفظ كأعداد صحيحة (integer) تمثل أصغر وحدة، مع حقل العملة (currency).
* جميع IDs تكون UUID/ULID.
* النصوص UTF-8، بدون رموز تحكم أو تنسيقات غير قياسية.
* الحقول المرجعية (مثل الألوان أو المقاسات) تستخدم جداول مرجعية ثابتة.

## **6) إجراءات التنفيذ**

1. تعريف JSON Schemas لكل مورد داخل OpenAPI.
2. فرض التحقق من المدخلات عبر Validation Middleware.
3. رفض أي طلب يحتوي على قيم خارج النطاق أو غير مطابقة للمخطط.
4. إضافة جداول مرجعية للأبعاد والسمات (مقاسات، ألوان).
5. تحديث المواصفة عند أي تعديل في المخططات.

## **7) قبول الجودة**

* جميع الطلبات والردود متوافقة مع JSON Schema.
* لا يوجد أي تاريخ خارج UTC ISO-8601.
* جميع IDs بصيغة UUID/ULID.
* جميع المبالغ المالية أعداد صحيحة مع currency.

## **8) القياس والمراقبة**

* معدل فشل التحقق من المخططات (Schema Validation Errors).
* عدد الحقول غير القياسية المرفوضة.
* إنذارات عند تزايد معدل الأخطاء في التحقق.

## **9) التكاملات**

* تكامل مع Analytics/Data Lake باستخدام نفس التنسيقات.
* تكامل مع بوابة المطورين لتوفير جداول مرجعية للألوان والمقاسات.

## **10) الأمان**

* منع أي قيم مدخلة قد تُستغل في Injection عبر Validation.
* التأكد من أن النصوص لا تحتوي على أكواد تنفيذية أو Tags HTML.

## **11) الأداء والكاش**

* Validation يتم على مستوى Middleware قبل الوصول للمنطق الرئيسي.
* جداول مرجعية تُخزن في Redis لتسريع التحقق.

## **12) المخاطر والضوابط**

* خطر اختلاف التنسيقات بين الأنظمة → الحل: فرض التحقق المركزي.
* خطر استخدام IDs متسلسلة → الحل: UUID/ULID دائمًا.
* خطر تضارب العملات → الحل: إلزام currency code في جميع الحقول المالية.

## **13) برنامج الاختبارات**

* اختبارات إدخال خاطئ (Invalid JSON, تاريخ خاطئ، معرف غير UUID).
* اختبارات قبول صحيحة (Schema Pass).
* اختبارات وحدات لحقول المبالغ المالية والعملة.

## **14) الوثائق وإدارة التغيير**

* جميع المخططات منشورة في OpenAPI.
* أي تغيير في الحقول أو الصيغ يوثق في Changelog.
* توفير جداول مرجعية محدثة في Docs.

## **15) الأدوات والمراجع**

* JSON Schema Validator.
* UUID/ULID Libraries.
* ISO-8601 Libraries.

## **16) تشيك**

* جميع التواريخ بصيغة ISO-8601 UTC.
* جميع IDs UUID/ULID.
* المبالغ المالية كـ integer + currency.
* Validation Middleware مفعّل.
* جداول مرجعية للألوان والمقاسات مستخدمة.
* OpenAPI يحتوي على JSON Schemas كاملة.

## **الملحقات التنفيذية (Artifacts)**

* جدول Reference للقيم (مقاسات، ألوان، عملات).
* أمثلة JSON Schema (Product, Order, Payment).
* سكربت تحقق آلي من التوافق مع المخططات في CI.

(13) Observability & Tracing

# **(13) الملاحظة والتتبع — Observability & Tracing**

## **1) آلية العمل والاعتماديات**

* كل طلب يحصل على معرف تتبع فريد trace\_id (أو X-Request-Id) يتم تمريره عبر جميع الخدمات.
* اللوغ (Logs) منظم بصيغة JSON ويحتوي على الحقول الأساسية (trace\_id, user\_id, endpoint, status, latency).
* المراقبة (Metrics) تُجمع آليًا عبر Prometheus/Datadog أو ما يعادلها.
* التتبع الموزع (Distributed Tracing) عبر OpenTelemetry أو Jaeger.

## **2) الغرض والنطاق**

* تمكين تتبع أي طلب من لحظة وصوله حتى استجابته عبر جميع المكونات.
* مراقبة الأداء والأخطاء بشكل آلي وسريع.
* ينطبق على جميع البيئات (dev, staging, prod).

## **3) المدخلات والمخرجات**

* المدخلات: طلبات الـAPI والمعطيات التشغيلية (CPU, Memory, Latency).
* المخرجات: لوغ منظم، مقاييس (metrics)، وتتبع (traces) يمكن عرضها على لوحات المراقبة.

## **4) المسؤوليات والأدوار**

* Backend: إضافة الحقول اللازمة (trace\_id, user\_id) للوغ.
* SRE/Infra: إعداد النظام المركزي للوغ والمقاييس.
* Security: التأكد من عدم وجود بيانات حساسة في اللوغ.
* QA: التحقق من وجود trace\_id في الاستجابات.

## **5) السياسات والمعايير**

* كل استجابة يجب أن تحتوي على رأس X-Trace-Id.
* اللوغ بصيغة JSON فقط (Structured Logging).
* لا يتم تسجيل أي بيانات حساسة (Passwords, Tokens, PII).
* المقاييس الأساسية: QPS, Latency (p50, p95, p99), Error Rate (4xx, 5xx), Resource Utilization.

## **6) إجراءات التنفيذ**

1. تفعيل Middleware لإضافة trace\_id إذا لم يكن موجودًا.
2. تمرير trace\_id لجميع الاستدعاءات الداخلية والخارجية.
3. تفعيل Structured Logging عبر مكتبة مركزية.
4. ربط الخدمات بـOpenTelemetry أو Jaeger للتتبع الموزع.
5. إعداد Dashboards للـLatency, Error Rate, QPS.
6. إعداد Alerts على المؤشرات الحرجة (مثلاً ارتفاع 5xx فوق 0.5%).

## **7) قبول الجودة**

* كل طلب يحتوي على trace\_id يمكن تتبعه في اللوغ.
* كل استدعاء داخلي يورّث نفس trace\_id.
* المقاييس تظهر في Dashboard بدون فجوات.
* التتبع الموزع يظهر التسلسل الكامل للطلب.

## **8) القياس والمراقبة**

* Latency: p50, p95, p99 لكل مسار.
* Error Rate: 4xx, 5xx لكل خدمة.
* QPS (requests per second).
* معدل نجاح استدعاءات Webhooks/Integrations.
* إنذارات آلية عند أي انحراف عن SLO.

## **9) التكاملات**

* تكامل مع CI/CD لإرسال Deployment Markers إلى لوحات المراقبة.
* تكامل مع نظام التنبيهات (Slack, Email, PagerDuty).
* تكامل مع Logging Central System (ELK, Loki, Datadog).

## **10) الأمان**

* إزالة أو إخفاء أي بيانات حساسة من اللوغ (PII, Secrets, Tokens).
* التحكم في مستوى اللوغ (DEBUG في dev فقط، INFO/ERROR في prod).
* تطبيق سياسات الاحتفاظ بالبيانات (Retention Policy).

## **11) الأداء والكاش**

* أخذ عينات (Sampling) للتتبع لتقليل الحمل (مثلاً 10% من الطلبات).
* ضغط البيانات قبل إرسالها للـCollector.

## **12) المخاطر والضوابط**

* خطر تضخم اللوغ → حل: سياسات احتفاظ (7–30 يومًا).
* خطر كشف بيانات حساسة → حل: Masking, Redaction.
* خطر فقدان التتبع → حل: تمرير trace\_id إلزامي.

## **13) برنامج الاختبارات**

* اختبار وجود trace\_id في كل استجابة.
* اختبار ظهور جميع المقاييس في Dashboard.
* اختبار إنذارات تنطلق عند تخطي الحدود.

## **14) الوثائق وإدارة التغيير**

* دليل Logging & Tracing منشور للفريق.
* جميع الحقول المطلوبة موثقة.
* تحديث الوثائق عند إضافة مقاييس جديدة.

## **15) الأدوات والمراجع**

* OpenTelemetry, Jaeger, Zipkin.
* Prometheus, Grafana, Datadog.
* ELK Stack أو Loki.

## **16) تشيك**

* كل استجابة تحتوي على trace\_id.
* Structured Logging مفعّل بصيغة JSON.
* لا يوجد أي PII في اللوغ.
* Dashboards للأداء والأخطاء متوفرة.
* Alerts مفعّلة للمؤشرات الأساسية.
* Distributed Tracing مفعّل على جميع الخدمات.

## **الملحقات التنفيذية (Artifacts)**

* قالب Log JSON يحدد الحقول الأساسية (trace\_id, user\_id, endpoint, latency, status).
* Dashboard Template لعرض QPS, Latency, Error Rate.
* سكربت تنبيه (Alert Rule) لارتفاع الأخطاء أو تجاوز p95.

(14) Quality & Testing

# **(14) الجودة والاختبارات — Quality & Testing**

## **1) آلية العمل والاعتماديات**

* جميع التغييرات في الكود تمر عبر **CI/CD Pipeline** مع مراحل اختبارات واضحة.
* يتم الاعتماد على **OpenAPI Contract Tests** للتحقق من مطابقة التنفيذ للمواصفة.
* الاختبارات تنقسم إلى: وحدة (Unit)، تكامل (Integration)، شاملة (E2E)، وأمن (Security).

## **2) الغرض والنطاق**

* ضمان أن كل إصدار من الـAPI يعمل كما هو موثق.
* منع التغييرات الكاسرة (Breaking Changes) من الوصول إلى الإنتاج.
* يشمل جميع الخدمات والبيئات (dev, staging, prod).

## **3) المدخلات والمخرجات**

* المدخلات: الكود الجديد، مواصفة OpenAPI، بيانات الاختبار.
* المخرجات: تقارير اختبارات (نجاح/فشل)، تغطية الكود، تقارير أمان وأداء.

## **4) المسؤوليات والأدوار**

* Backend: كتابة اختبارات الوحدة والتكامل.
* QA: كتابة وتشغيل اختبارات العقد وE2E.
* Security: إجراء فحوص أمان دورية (DAST, SAST).
* SRE: مراقبة نتائج الأداء في الاختبارات.

## **5) السياسات والمعايير**

* جميع الـPR يجب أن تمر بجميع مراحل الاختبارات.
* تغطية الكود ≥ 60% كبداية، مع هدف تحسين تدريجي.
* اختبارات الأداء الأساسية إلزامية قبل أي نشر.
* اختبار أمني (DAST) شهري على الأقل.

## **6) إجراءات التنفيذ**

1. تشغيل **Linting & Static Analysis** في بداية CI.
2. تشغيل اختبارات الوحدة مع كل Build.
3. تشغيل اختبارات التكامل ضد قاعدة بيانات وهمية.
4. تشغيل اختبارات العقد باستخدام Dredd/Schemathesis ضد OpenAPI.
5. تشغيل اختبارات E2E على بيئة Staging.
6. اختبار أداء (Load/Stress) على المسارات الأساسية.
7. تشغيل DAST (OWASP ZAP) بشكل دوري.

## **7) قبول الجودة**

* جميع الاختبارات تمر بدون فشل.
* تغطية الكود ≥ 60%.
* اختبارات العقد = 0 فروقات.
* نتائج الأداء ضمن SLOs (p95 ≤ 300ms للقراءة، ≤ 800ms للكتابة).

## **8) القياس والمراقبة**

* معدل نجاح الاختبارات في CI.
* نسبة التغطية الكودية.
* مؤشرات الأداء الأساسية من اختبارات الحمل.
* إنذارات في حال فشل اختبارات العقد أو الأمان.

## **9) التكاملات**

* تكامل مع GitHub Actions/GitLab CI لتشغيل الاختبارات.
* تكامل مع أنظمة التقارير (Allure, SonarQube).
* تكامل مع مراقبة الإنتاج لمقارنة الأداء الفعلي بالاختبارات.

## **10) الأمان**

* تفعيل SAST (Static Analysis) عند كل دمج.
* تشغيل DAST بشكل دوري على بيئة Staging.
* اختبارات للتحقق من عدم وجود PII في الاستجابات.

## **11) الأداء والكاش**

* تشغيل Load Test عند كل نشر رئيسي.
* اختبار استجابات الكاش (GET + ETag, If-None-Match).

## **12) المخاطر والضوابط**

* خطر ضعف التغطية → إلزام حد أدنى وتتبّع التحسين.
* خطر تجاهل نتائج الأداء → إدراجها كشرط (Gate) في CI/CD.
* خطر تجاهل فحوص الأمان → جدولة دورية إلزامية.

## **13) برنامج الاختبارات**

* Unit Tests: التحقق من الدوال والمكونات الفردية.
* Integration Tests: التحقق من الربط مع DB/Redis/Queue.
* Contract Tests: مطابقة OpenAPI.
* E2E Tests: رحلة عميل كاملة (Login → Browse → Checkout).
* Performance Tests: k6/Artillery.
* Security Tests: ZAP, SAST.

## **14) الوثائق وإدارة التغيير**

* جميع أنواع الاختبارات موثقة في ملف QA Guide.
* تحديث الوثائق عند إضافة سيناريو جديد.
* تقارير الاختبار تحفظ تلقائيًا في CI كـArtifacts.

## **15) الأدوات والمراجع**

* Jest/PyTest/NUnit لاختبارات الوحدة.
* Postman/Newman أو Cypress لاختبارات E2E.
* Dredd/Schemathesis لاختبارات العقد.
* k6/Artillery للأداء.
* OWASP ZAP للأمان.

## **16) تشيك**

* Unit Tests محدثة وتغطي المنطق الأساسي.
* Integration Tests تعمل ضد DB/Redis وهمية.
* Contract Tests ضد OpenAPI ناجحة.
* E2E Tests تعمل على Staging.
* Performance Tests ضمن SLOs.
* Security Tests دورية ونتائجها محفوظة.

## **الملحقات التنفيذية (Artifacts)**

* قالب تقارير الاختبارات (نجاح/فشل/تغطية).
* ملف QA Guide يصف كل نوع اختبار.
* سكربت k6/Artillery للرحلات الأساسية.
* خطة فحص أمني شهرية.

(15) API — API Gateway

# **(15) بوابة الـAPI — API Gateway**

## **1) آلية العمل والاعتماديات**

* تعمل كبوابة موحّدة على الحافة أمام جميع الخدمات: إنهاء TLS، توجيه المسارات، سياسات الأمان، المراقبة.
* تطبّق سياسات مشتركة (Auth, CORS, Rate Limiting, WAF, Headers, Caching, Tracing) قبل وصول الطلب للخدمة.
* تتكامل مع OpenAPI لتوليد قواعد المسارات والتحقق الأساسي.

## **2) الغرض والنطاق**

* توحيد نقاط الدخول والتحكم المركزي في السياسات.
* تقليل التعقيد داخل الخدمات الخلفية وت标准ة السلوك عبر كل الـAPIs.

## **3) المدخلات والمخرجات**

* المدخلات: طلبات HTTP من العملاء/الشركاء.
* المخرجات: تمرير الطلب للخدمة المناسبة أو رد فوري بسياسة البوابة (مثل 429، 401).

## **4) المسؤوليات والأدوار**

* SRE/Infra: إدارة ونشر الـGateway وسياساته ومراعاة التوسّع.
* Security: ضبط WAF, CORS, Auth, Bot Management.
* Backend: تعريف المسارات والمطلوب من السياسات لكل خدمة.
* QA: اختبار سلوك السياسات عبر بيئات dev/staging.

## **5) السياسات والمعايير**

* TLS إلزامي، منع HTTP العاري.
* CORS Allowlist لمصادر موثوقة فقط.
* Rate Limiting متعدد الأبعاد (IP، User, Token, Partner).
* حقن/تمرير معرف التتبع X-Request-Id وtraceparent.
* توحيد رؤوس الاستجابات (Security Headers: X-Content-Type-Options, X-Frame-Options, Referrer-Policy…).
* دعم Cache-Control للموارد العامة على الحافة.
* توحيد أخطاء البوابة ضمن الغلاف القياسي للأخطاء.

## **6) إجراءات التنفيذ**

1. إعداد نطاقات البوابة (prod/staging/dev) مع شهادات TLS.
2. تعريف مخطط التوجيه: /v1/\* → خدمة X، /v2/\* → خدمة Y.
3. تمكين CORS وRate Limit وWAF وفق سياسات مكتوبة.
4. تفعيل Auth على مستوى البوابة (JWT verification/OAuth introspection) إن لزم.
5. تفعيل Proxy-timeouts, Retries مع سياسات معقولة.
6. تمكين Observability: Logs, Metrics, Traces على مستوى البوابة.

## **7) قبول الجودة**

* جميع الطلبات تمر عبر HTTPS وتظهر في لوحات المراقبة.
* سياسات CORS وRate Limiting تعمل وفق الجداول الموثّقة.
* أخطاء البوابة تعود بالشكل الموحّد.
* توجيه صحيح لكل مسار بدون تسرب.

## **8) القياس والمراقبة**

* مؤشرات: QPS، P95 latency عند البوابة، معدلات 4xx/5xx/429، نسبة cache hit على الحافة.
* إنذارات عند ارتفاع 5xx، أو تخطي حدود P95، أو زيادة 429 غير طبيعية.

## **9) التكاملات**

* التكامل مع أنظمة Auth (JWKS, OAuth2 Introspection).
* التكامل مع CDN/Bot Management على الحافة.
* التكامل مع CI/CD لنشر سياسات البوابة كـ Code (GitOps).

## **10) الأمان**

* تشغيل WAF بقواعد OWASP، وتمكين Bot Management عند الحاجة.
* تدوير مفاتيح/أسرار البوابة دوريًا.
* IP Allowlist للتكاملات الحساسة/لوحات التحكم.
* إخفاء عناوين الخدمات الداخلية (no direct origin exposure).

## **11) الأداء والكاش**

* تفعيل الكاش للموارد العامة على الحافة مع Cache-Control وETag.
* ضغط Brotli/Gzip على الاستجابات النصية.
* إعداد Keep-Alive وHTTP/2 أو HTTP/3 لتحسين زمن البدء.

## **12) المخاطر والضوابط**

* إسقاط خاطئ لطلبات سليمة بسبب قواعد WAF صارمة → ضابط: وضع “Learning Mode” قبل التشديد ومراجعة السجلات.
* سوء تهيئة CORS يؤدي لحجب الواجهات الأمامية → ضابط: اختبارات CORS في CI وبيئة Staging.
* عنق زجاجة في البوابة → ضابط: Auto-Scaling، قياس P95، توزيع أقاليم.

## **13) برنامج الاختبارات**

* اختبارات دُخانية لسياسات CORS/Rate/Headers.
* اختبارات Auth على مستوى البوابة (JWT مفقود/منتهي).
* اختبارات أداء على طبقة البوابة تحت حمل متدرّج.

## **14) الوثائق وإدارة التغيير**

* وثيقة سياسات البوابة (CORS, Rate, WAF, Headers, Timeouts).
* سجل تغييرات السياسات مع أسباب وتواريخ.
* نسخ قابلة للاسترجاع (versioned policies) عبر المستودع.

## **15) الأدوات والمراجع**

* Gateways: NGINX, Kong, Envoy, Cloudflare/APIM.
* سياسة كود: Declarative config/CRDs، GitOps (ArgoCD/Flux).
* مراقبة: Prometheus/Grafana/Datadog للبوابة.

## **16) تشيك**

* HTTPS مفعل، لا وصول عبر HTTP.
* CORS Allowlist مضبوطة ومختبرة.
* Rate Limiting مفعّل وجداول الحدود منشورة.
* WAF/Bot Management مفعّلان ومراقبان.
* Auth على مستوى البوابة (عند الحاجة) + JWKS متزامن.
* Logs/Metrics/Tracing مفعلة مع Dashboards.
* أخطاء البوابة تُرجع الشكل الموحّد.
* سياسات البوابة مُدارة ككود ومربوطة بـCI/CD.

## **الملحقات التنفيذية (Artifacts)**

* جداول CORS وRate Limits الفعلية (Origin → Allowed، Path → Limits، مفاتيح التخصيص).
* قالب سياسات البوابة كملف إعلاني (YAML/JSON) قابل للنشر.
* أمثلة رؤوس الإجابات القياسية من البوابة (Security Headers, Cache, Trace).
* دليل Timeouts/Retry (client ↔ gateway ↔ upstream) بالأرقام الموصى بها.

(16) CI/CD

# **(16) CI/CD — مراحل وبوابات الصحة**

## **1) آلية العمل والاعتماديات**

* خط أنابيب آلي يبدأ عند كل PR/merge/tag: يبني، يختبر، يفحص الأمان، ينشر إلى البيئات بالتدرّج.
* يعتمد على مستودع واحد/متعددة للخدمات، ويستخدم قوالب سير عمل موحّدة (Reusable Workflows).
* النشر يتم على مراحل: dev → staging (اختبارات دخانية) → canary (جزء من الحركة) → prod الكامل + مراقبة حيّة.

## **2) الغرض والنطاق**

* تسليم سريع وآمن وقابل للتكرار، مع إيقاف تلقائي عند أي فشل.
* يشمل جميع الخدمات والبوابة وسياسات البنية (Infrastructure as Code).

## **3) المدخلات والمخرجات**

* المدخلات: الكود، مواصفة OpenAPI، اختبارات، ملفات البنية (IaC)، أسرار/مفاتيح.
* المخرجات: صور حاويات موقّعة، تقارير اختبارات/تغطية/أمان/أداء، حزم SDKs، نشر مُوسوم (versioned release).

## **4) المسؤوليات والأدوار**

* Backend: كتابة الكود والاختبارات وإصلاح الفشل.
* QA: إدارة مجموعات الاختبارات (Contract/E2E/Perf-smoke).
* Security: قواعد SAST/DAST وسياسات الاعتمادات.
* SRE/Infra: قوالب CI، نشر، مراقبة ما بعد الإطلاق، rollback.

## **5) السياسات والمعايير**

* أي PR يجب أن يجتاز: Lint + Unit + Contract + SAST على الأقل.
* حد تغطية مبدئي ≥ 60% مع تحسين تدريجي.
* فشل تلقائي إذا وجدت ثغرات حرجة/عالية لم تُستثنَ صراحة.
* نشر مُعنون (SemVer أو تاريخي) مع Changelog آلي.

## **6) إجراءات التنفيذ**

1. Lint & Static Analysis (code style, secrets scan).
2. Unit Tests + Coverage.
3. Build Docker image + SBOM + توقيع (cosign/slsa).
4. SAST (تحليل ثابت) وDependencies Scan.
5. Contract Tests ضد OpenAPI، ثم Integration Tests.
6. نشر إلى Staging + Smoke/E2E.
7. Perf-smoke (عينات سريعة) وDAST على Staging.
8. Canary إلى نسبة صغيرة (مثلاً 5–10%) مع Health Gates.
9. توسيع النشر → 50% → 100% إذا مرّت القياسات.
10. وضع Deployment Marker وإطلاق مراقبة ما بعد الإطلاق.

## **7) قبول الجودة**

* كل المراحل السابقة خضراء (بدون فشل).
* تقارير التغطية ≥ الحد الأدنى.
* فحص الأمان بلا ثغرات حرجة/عالية غير مُعالجة.
* Perf-smoke ضمن SLOs قبل التوسيع إلى كامل المستخدمين.

## **8) القياس والمراقبة**

* عرض تلقائي لمقاييس P95/5xx/429 أثناء الـcanary.
* إنذارات فورية مع عتبات محددة توقف التوسيع (Auto-Fail).
* لوحات مقارنة قبل/بعد الإطلاق (baseline vs current).

## **9) التكاملات**

* ربط CI مع إدارة الحاويات (Registry خاص) وتوقيع الصور.
* ربط مع بوابة المطورين لنشر Swagger/SDKs بعد الدمج.
* تكامل مع نظام التذاكر لتوليد/تحديث المهام تلقائيًا عند الفشل.

## **10) الأمان**

* إدارة الأسرار عبر Secret Manager/CI Vault (لا أسرار في المتغيرات العادية).
* تفعيل السياسات: Least Privilege لحسابات النشر.
* توقيع الصور والتحقق من السلامة قبل السحب (admission policy).

## **11) الأداء والكاش**

* استخدام طبقات Cache للـdependencies والأبنية لتسريع الـCI.
* تشغيل اختبارات أداء خفيفة على كل نشر، وكاملة على الإصدارات الكبرى.

## **12) المخاطر والضوابط**

* نشر خاطئ إلى prod → ضابط: بوابات صحّة + موافقة إلزامية + canary.
* تسريب أسرار في الـlogs → ضابط: ماسك أسرار + تدقيق دوري.
* تجاهل نتائج الأمان → ضابط: فشل إجباري فوق عتبة الشدة.

## **13) برنامج الاختبارات**

* Unit/Integration/Contract/E2E في مراحل منفصلة واضحة.
* Perf-smoke في Staging، وأحيانًا على canary.
* DAST مجدول أسبوعيًا/شهريًا حسب الحساسية.

## **14) الوثائق وإدارة التغيير**

* دليل CI/CD موثّق (المراحل، العتبات، منطق الإيقاف).
* Changelog آلي من الـPRs والتاغات.
* سياسة Rollback موثّقة وخطوات تشغيل مختصرة.

## **15) الأدوات والمراجع**

* منصات CI: GitHub Actions/GitLab CI.
* حاويات: Docker/OCI + Registry خاص.
* أمان: Trivy/Grype/Snyk, ZAP.
* تتبع: Grafana/Datadog + OpenTelemetry.
* توقيع/سلسلة توريد: Cosign, SLSA, SBOM (Syft).

## **16) تشيك**

* Lint/Static/Secrets Scan ناجحة.
* Unit/Integration/Contract/E2E جميعها خضراء.
* Coverage ≥ 60%.
* SAST/Dependency Scan: لا ثغرات حرجة/عالية غير معالجة.
* Staging Smoke + Perf-smoke ناجحة.
* Canary مفعل مع Health Gates وBaseline مقارنة.
* نشر كامل مع Deployment Marker ومراقبة ما بعد الإطلاق.
* خطة Rollback مجرّبة وجاهزة.

## **الملحقات التنفيذية (Artifacts)**

* قوالب Workflows جاهزة (lint/test/build/scan/deploy) قابلة لإعادة الاستخدام.
* ملف عتبات صحّة (Health Gates) يحدد حدود P95/5xx/429 للإيقاف الآلي.
* سكربت توليد Changelog من الـPRs/tags.
* كتيّب Rollback مختصر (أوامر + شروط التفعيل + التحقق ما بعد التراجع).

(17) SLOs / OKRs

# **(17) مؤشرات القبول الكلية — SLOs / OKRs**

## **1) آلية العمل والاعتماديات**

* تحديد مؤشرات خدمة واضحة (Service Level Objectives - SLOs) تُبنى على مؤشرات رئيسية (SLIs).
* مراقبة مستمرة عبر أدوات مثل Prometheus, Datadog, Grafana.
* مؤشرات الأداء مرتبطة بأهداف العمل (OKRs) لضمان التوافق بين التقنية والأهداف التجارية.

## **2) الغرض والنطاق**

* ضمان أن تجربة المستخدم والجودة التقنية يمكن قياسها والتحكم بها.
* تمكين الفريق من معرفة متى تتدهور الخدمة ومتى يجب التدخل.
* النطاق يشمل جميع واجهات الـAPI، الأنظمة المساندة (Auth, DB, Cache)، والتكاملات الخارجية.

## **3) المدخلات والمخرجات**

* المدخلات: بيانات المراقبة (Requests, Latency, Errors, Availability).
* المخرجات: لوحات مؤشرات SLOs، تقارير شهرية/ربع سنوية عن الالتزام.

## **4) المسؤوليات والأدوار**

* Product/PM: تحديد الـOKRs المرتبطة بالأعمال (مثلاً: تقليل فشل الدفع إلى <1%).
* SRE: صياغة وقياس الـSLOs وربطها بالـSLIs.
* Backend: تحسين الكود لتلبية الأهداف.
* QA: التحقق من مؤشرات الأداء في البيئات قبل الإنتاج.

## **5) السياسات والمعايير**

* صياغة كل SLO بهذا الشكل: **[المؤشر] ≤/>= [الحد] خلال [المدة]**.
* أمثلة:
  + Latency: 95% من طلبات GET /products ≤ 300ms على مدار 30 يومًا.
  + Availability: 99.9% من الطلبات لا تفشل (5xx) خلال ربع سنة.
  + Error Rate: معدل 5xx ≤ 0.1% شهريًا.
  + Auth: زمن الاستجابة لمصادقة المستخدم ≤ 200ms في 95% من الحالات.

## **6) إجراءات التنفيذ**

1. تعريف الـSLIs (Latency, Availability, Error Rate, Throughput).
2. صياغة الـSLOs بناءً على الـSLIs.
3. ربط الـSLOs بـOKRs العمل (زيادة المبيعات، تحسين تجربة الدفع).
4. نشر الـDashboards والـAlerts بناءً على SLOs.
5. مراجعة ربع سنوية لتعديل الحدود.

## **7) قبول الجودة**

* جميع الـSLIs متوفرة في لوحات واضحة.
* الـSLOs قابلة للقياس خلال فترة زمنية (30 يوم، ربع سنة).
* تقارير الالتزام تُرسل دوريًا للإدارة.
* أي إخفاق في SLO يفعّل خطة استجابة (Error Budget Policy).

## **8) القياس والمراقبة**

* SLIs رئيسية: Latency, Error Rate, Availability, Throughput.
* Error Budget: النسبة المسموح بها من الإخفاقات قبل وقف النشر الجديد.
* OKRs مرتبطة (مثال: 99% من العملاء يكملون عملية الشراء بنجاح).

## **9) التكاملات**

* تكامل مع أدوات المراقبة (Datadog, Prometheus) لإرسال التنبيهات.
* تكامل مع CI/CD لإيقاف النشر عند تجاوز حدود الـError Budget.

## **10) الأمان**

* تضمين SLOs متعلقة بالأمان:
  + جميع الطلبات عبر HTTPS = 100%.
  + نتائج DAST الشهرية = 0 ثغرات حرجة.

## **11) الأداء والكاش**

* مؤشرات خاصة بالكاش: معدل Cache Hit ≥ 80% للقوائم العامة.
* زمن معالجة Redis ≤ 5ms في 95% من الطلبات.

## **12) المخاطر والضوابط**

* خطر وضع حدود غير واقعية → مراجعة دورية مع الفريق.
* خطر تجاهل الأهداف → إلزام بسياسة Error Budget توقف النشر عند الإخفاق.

## **13) برنامج الاختبارات**

* اختبار أن الـSLIs تُجمع بشكل صحيح.
* اختبار أن الـAlerts تنطلق عند تجاوز الحدود.
* اختبار تقارير الالتزام تُولد بشكل دوري.

## **14) الوثائق وإدارة التغيير**

* جميع الـSLOs موثقة في مستند SLO Handbook.
* تحديث الوثائق عند تغيير أي حد أو إضافة مؤشر.
* نشر ملخصات الالتزام كجزء من تقارير الإدارة.

## **15) الأدوات والمراجع**

* Prometheus/Grafana, Datadog.
* SLO frameworks مثل Google SRE Workbook.
* Error Budget Policies من Google SRE.

## **16) تشيك**

* SLIs معرفة ومراقبة (Latency, Availability, Error Rate, Throughput).
* SLOs مكتوبة بوضوح ومربوطة بمدة زمنية.
* Error Budget Policy مفعلة وتوقف النشر عند التجاوز.
* Dashboards وAlerts جاهزة وتعمل.
* مراجعة ربع سنوية للأهداف والحدود.

## **الملحقات التنفيذية (Artifacts)**

* جدول SLOs لكل خدمة (المؤشر، الهدف، الفترة، Error Budget).
* قوالب Dashboards جاهزة للعرض.
* تقرير التزام شهري/ربع سنوي.

(18) Incidents & Error Budget

# **(18) إدارة الأخطاء والأحداث — Incidents & Error Budget**

## **1) آلية العمل والاعتماديات**

* يتم قياس معدل الأخطاء والفشل عبر **SLIs** (latency, error rate, availability).
* يُحدد **Error Budget** (مثلاً: 0.1% من الطلبات يمكن أن تفشل خلال 30 يومًا).
* عند تجاوز الحد يتم تفعيل خطة إدارة الحوادث (Incident Management) وفق بروتوكول واضح (On-call, Escalation, Postmortem).
* يعتمد على أدوات المراقبة (Datadog, Grafana, Prometheus) + نظام إدارة الأحداث (PagerDuty, OpsGenie).

## **2) الغرض والنطاق**

* السيطرة على جودة الخدمة ومنع التدهور المستمر.
* تحديد متى يتم **إيقاف النشر الجديد** بسبب استهلاك الميزانية.
* يشمل جميع واجهات الـAPI، الخدمات الأساسية، والتكاملات.

## **3) المدخلات والمخرجات**

* المدخلات: بيانات الأخطاء (4xx, 5xx, Latency breaches).
* المخرجات: تقارير Error Budget، سجل الحوادث (Incidents Log)، Postmortems.

## **4) المسؤوليات والأدوار**

* SRE: متابعة مؤشرات الـError Budget، تفعيل خطة الحوادث عند التجاوز.
* On-call Engineer: الاستجابة الأولية ومعالجة المشكلة.
* Incident Commander (IC): قيادة الاستجابة، التنسيق مع الفرق.
* Product/PM: اتخاذ القرار بإيقاف النشر أو استئنافه بعد الحادث.

## **5) السياسات والمعايير**

* Error Budget = نسبة الأخطاء المسموح بها ضمن فترة زمنية (مثلاً: 0.1% شهريًا).
* عند تجاوز الميزانية:
  + إيقاف مؤقت لكل عمليات النشر.
  + التركيز على الإصلاحات بدل الميزات الجديدة.
* كل حادث يجب أن يُوثق بـPostmortem خلال 72 ساعة.
* لا عقوبات فردية (Blameless Culture).

## **6) إجراءات التنفيذ**

1. مراقبة الـSLIs باستمرار وتوليد تقارير Error Budget.
2. عند تجاوز الميزانية → تنبيه تلقائي للفريق.
3. إعلان حادث رسمي وتعيين Incident Commander.
4. توثيق جميع الخطوات والقرارات أثناء الحادث.
5. بعد الحل: كتابة Postmortem، مراجعة الأسباب الجذرية، وضع إجراءات منع تكرار.
6. مراجعة Error Budget في الاجتماعات الدورية للمنتج والفريق.

## **7) قبول الجودة**

* جميع الحوادث مسجلة برقم وحالة.
* وجود Postmortem لكل حادث كبير.
* خطط الإصلاح (Action Items) معرفة ومتابعة.
* الالتزام بمراجعة Error Budget بشكل دوري.

## **8) القياس والمراقبة**

* معدل استهلاك Error Budget (مثلاً: 50% مستخدم في أول 10 أيام).
* عدد الحوادث الشهرية والربع سنوية.
* متوسط زمن الحل (MTTR).
* متوسط زمن الاكتشاف (MTTD).

## **9) التكاملات**

* التكامل مع PagerDuty/OpsGenie لإدارة On-call.
* التكامل مع Slack/Email للتنبيهات الفورية.
* التكامل مع أنظمة إدارة المشاريع (Jira, Linear) لتوثيق Postmortems.

## **10) الأمان**

* مراجعة أي حادث أمني بشكل منفصل مع إجراءات خاصة.
* Postmortem أمني قد يكون محدود الوصول.
* مراقبة محاولات الهجمات (DDoS, Injection) كجزء من الحوادث.

## **11) الأداء والكاش**

* تتبع تأثير الحوادث على الأداء (Latency spikes, Cache misses).
* توثيق إن كانت الأعطال بسبب تخزين مؤقت أو قاعدة بيانات.

## **12) المخاطر والضوابط**

* خطر تجاهل Error Budget → حل: سياسة صارمة بإيقاف النشر عند التجاوز.
* خطر عدم وجود شفافية → حل: مشاركة تقارير Postmortems مع جميع الفرق.
* خطر إلقاء اللوم → حل: ثقافة Blameless مع التركيز على النظام لا الأفراد.

## **13) برنامج الاختبارات**

* محاكاة حوادث (Game Days/Chaos Engineering) لاختبار الاستجابة.
* اختبار تنبيهات PagerDuty/Slack تعمل عند الاختراق.
* مراجعة قدرة الفريق على الالتزام بدليل الحوادث.

## **14) الوثائق وإدارة التغيير**

* دليل Incident Response منشور للفريق.
* سياسة Error Budget موثقة ومربوطة بالـSLOs.
* سجل الحوادث محفوظ مع Postmortems وروابطها.

## **15) الأدوات والمراجع**

* PagerDuty, OpsGenie, VictorOps.
* Grafana, Datadog, Prometheus.
* Chaos Monkey, Gremlin لاختبار المتانة.
* Google SRE Workbook لإدارة الحوادث.

## **16) تشيك**

* Error Budget معرف ومربوط بالـSLOs.
* Alerts تلقائية عند تجاوز الميزانية.
* سياسة إيقاف النشر مفعلة عند التجاوز.
* Incident Commander محدد في كل حادث.
* Postmortem مكتوب لكل حادث كبير.
* Action Items موثقة ويتم متابعتها.
* مراجعة دورية لاستهلاك Error Budget.

## **الملحقات التنفيذية (Artifacts)**

* قالب Postmortem رسمي (الحادث، السبب الجذري، التأثير، الإصلاحات، الإجراءات المستقبلية).
* لوحة Dashboard لاستهلاك Error Budget.
* جدول Escalation Matrix (من المسؤول الأولي، من التالي، كيف يتم التصعيد).

(19) Governance & Change Management

# **(19) الحوكمة وإدارة التغيير — Governance & Change Management**

## **1) آلية العمل والاعتماديات**

* كل تغيير في الـAPI أو بنيته يجب أن يمر عبر عملية موحدة (RFC / ADR).
* يتم تسجيل القرارات المعمارية (Architecture Decision Records - ADRs) في المستودع.
* إدارة التغيير (Change Management) تتكامل مع Git (PRs, Reviews) و CI/CD.
* السياسات مكتوبة ومعلنة (Policies as Code).

## **2) الغرض والنطاق**

* ضمان أن التغييرات مدروسة، موثقة، وقابلة للتتبع.
* تقليل المخاطر الناتجة عن التغييرات المفاجئة أو غير المنسقة.
* ينطبق على الكود، البنية، مواصفات الـAPI، والعمليات.

## **3) المدخلات والمخرجات**

* المدخلات: طلب تغيير (PR, RFC, ADR).
* المخرجات: موافقة/رفض، تحديث مواصفات OpenAPI، تحديث المستندات.

## **4) المسؤوليات والأدوار**

* Developers: رفع PRs وطلبات التغيير مع وصف كامل.
* Tech Leads/Architects: مراجعة واعتماد ADRs.
* Product/PM: تقييم أثر التغيير على العملاء والمنتج.
* SRE/Security: مراجعة التغييرات ذات الأثر التشغيلي أو الأمني.

## **5) السياسات والمعايير**

* أي تغيير في الـAPI يتطلب تحديث OpenAPI + Changelog.
* كل تغيير معماري كبير يوثق في ADR.
* مراجعة أقران (Peer Review) إلزامية لكل PR.
* سياسة SemVer: أي Breaking Change → إصدار رئيسي جديد.
* أي تغيير إنتاجي يحتاج إلى مراجعة مخاطر وتوثيق.

## **6) إجراءات التنفيذ**

1. إنشاء RFC/ADR عند اقتراح تغيير مهم.
2. مناقشة ومراجعة التغيير عبر اجتماع تقني أو PR.
3. تحديث الوثائق (OpenAPI, Guides, Runbooks).
4. إدراج التغيير في Changelog.
5. نشر التغيير تدريجيًا عبر CI/CD مع مراقبة.

## **7) قبول الجودة**

* وجود ADR موثق لكل تغيير معماري.
* جميع PRs تمت مراجعتها من شخصين على الأقل.
* Changelog محدث عند كل إصدار.
* توافق التغيير مع سياسات SemVer.

## **8) القياس والمراقبة**

* معدل الالتزام بالـChange Process (عدد PRs بدون مراجعة = 0).
* عدد ADRs المنشورة شهريًا.
* نسبة التغييرات التي سببت حوادث (يجب أن تكون منخفضة جدًا).

## **9) التكاملات**

* ربط GitHub/GitLab مع أدوات التوثيق لنشر ADRs تلقائيًا.
* تكامل مع بوابة المطورين لتحديث التغييرات المهمة.
* تكامل مع أنظمة Incident Management لمراجعة تأثير التغييرات.

## **10) الأمان**

* مراجعة أمنية إلزامية للتغييرات في المصادقة/التخزين/البيانات الحساسة.
* سياسات Least Privilege عند تطبيق تغييرات في البنية.
* تتبّع Audit Log لكل تغيير.

## **11) الأداء والكاش**

* أي تغيير في آليات الكاش أو الأداء يجب أن يرافقه خطة اختبار أداء.
* مراجعة تأثير التغييرات على SLOs قبل دمجها.

## **12) المخاطر والضوابط**

* خطر تغييرات غير موثقة → حل: إلزام ADR وChangelog.
* خطر تغييرات غير مراجعة → حل: فحص آلي في CI يمنع الدمج بدون Review.
* خطر تجاهل الأثر التجاري → حل: إشراك PM دائمًا.

## **13) برنامج الاختبارات**

* اختبار أن التغييرات مرفقة بتحديث OpenAPI.
* اختبار أن CI يوقف الدمج عند غياب مراجعة أو اختبارات.
* اختبار أن ADRs محفوظة وقابلة للاسترجاع.

## **14) الوثائق وإدارة التغيير**

* جميع السياسات منشورة في مستند Governance Handbook.
* جميع ADRs في مجلد /docs/adr/ داخل المستودع.
* أي تغيير API يرافقه تحديث Changelog + Portal.

## **15) الأدوات والمراجع**

* GitHub/GitLab PR Reviews.
* ADR Templates (Markdown).
* Semantic Versioning Guidelines.
* Google SRE Change Management Principles.

## **16) تشيك**

* كل تغيير API → تحديث OpenAPI + Changelog.
* أي تغيير معماري → ADR جديد.
* كل PR → مراجعة من شخصين على الأقل.
* أي Breaking Change → إصدار رئيسي جديد (SemVer).
* سجل تغيير (Audit Log) محدث.
* Governance Handbook متوفر للفريق.

## **الملحقات التنفيذية (Artifacts)**

* قالب ADR (السياق، القرار، البدائل، التأثير).
* قالب Changelog (Added, Changed, Deprecated, Removed).
* Workflow CI يتحقق من وجود تحديث OpenAPI/Changelog عند أي تغيير API.
* Audit Dashboard يعرض جميع التغييرات الأخيرة.

(20) Versioning & Documentation Portal

# **(20) إدارة النسخ والتوثيق — Versioning & Documentation Portal**

## **1) آلية العمل والاعتماديات**

* كل إصدار جديد للـAPI موثق بوضوح في **OpenAPI Spec** مستقل.
* البوابة (API Portal/Developer Portal) تعرض الوثائق بشكل تفاعلي (Swagger/Redoc/Postman Collections).
* إدارة النسخ تعتمد على **Semantic Versioning (SemVer)**:
  + Major → تغييرات كاسرة.
  + Minor → ميزات جديدة متوافقة.
  + Patch → إصلاحات بدون تغيير على العقد.

## **2) الغرض والنطاق**

* توفير مصدر واحد وموحد للمطورين الداخليين والخارجيين.
* تمكين العملاء والشركاء من التكامل بسرعة وبدون ارتباك.
* ينطبق على جميع إصدارات الـAPI (عامة، داخلية، Webhooks).

## **3) المدخلات والمخرجات**

* المدخلات: OpenAPI YAML/JSON، Changelog، SDKs.
* المخرجات: بوابة توثيق غنية بالمخططات والأمثلة والأدلة.

## **4) المسؤوليات والأدوار**

* Backend: تحديث OpenAPI عند أي تغيير.
* DevRel/Docs: نشر الوثائق في البوابة وتحديث الأمثلة.
* Product/PM: التأكد من أن التغييرات موثقة للمستخدمين.
* QA: مراجعة دقة الأمثلة مقابل التنفيذ.

## **5) السياسات والمعايير**

* كل إصدار API يجب أن يكون له مواصفة مستقلة (v1, v2, …).
* Deprecation Policy منشورة (مثلاً: 90 يومًا قبل إيقاف النسخة القديمة).
* كل تغيير موثق في Changelog ضمن الأقسام: Added, Changed, Deprecated, Removed.
* البوابة توفر SDKs آلية (JS, Swift, Kotlin, PHP) مولدة من OpenAPI.
* توحيد شكل الأمثلة (Requests/Responses) في كل إصدار.

## **6) إجراءات التنفيذ**

1. تحديث ملف OpenAPI عند كل تغيير.
2. تشغيل CI لتوليد الوثائق (Redoc, Swagger UI) ورفعها للبوابة.
3. توليد SDKs و Postman Collections تلقائيًا.
4. تحديث Changelog وربطه بالإصدار الجديد.
5. إعلان الإصدار الجديد في البوابة مع توثيق خطة الترحيل.

## **7) قبول الجودة**

* كل إصدار API ظاهر بوضوح في البوابة.
* جميع الأمثلة في البوابة متطابقة مع التنفيذ.
* Changelog محدّث مع كل إصدار.
* وجود خطة ترحيل عند Deprecation.

## **8) القياس والمراقبة**

* عدد زيارات البوابة + تنزيلات SDKs.
* نسبة العملاء المنتقلين من النسخة القديمة للجديدة.
* إنذارات عند وجود اختلاف بين OpenAPI والبوابة.

## **9) التكاملات**

* CI/CD لنشر التوثيق مع كل دمج/إصدار.
* ربط البوابة مع نظام Auth لعرض الوثائق الداخلية بشكل مقيد.
* تكامل مع Analytics لمتابعة استخدام الإصدارات المختلفة.

## **10) الأمان**

* البوابة خلف HTTPS فقط.
* الوثائق الداخلية تتطلب Auth.
* إخفاء أي معلومات سرية/داخلية عن العملاء الخارجيين.

## **11) الأداء والكاش**

* Cache-Control لتسريع تحميل الوثائق والـSDKs.
* تحديث تلقائي للبوابة عند أي إصدار جديد.

## **12) المخاطر والضوابط**

* خطر عدم توثيق التغييرات → CI يتحقق من وجود تحديث OpenAPI + Changelog.
* خطر استمرار العملاء على إصدار قديم → نشر تحذيرات Deprecation في البوابة + إشعارات.
* خطر تعارض في SDKs → توليد آلي من نفس الـOpenAPI لتجنب الأخطاء اليدوية.

## **13) برنامج الاختبارات**

* اختبار نشر الوثائق في بيئة Staging قبل الإنتاج.
* اختبار روابط الأمثلة (Requests/Responses) تعمل فعليًا.
* اختبار أن جميع SDKs محدثة ومتوافقة.

## **14) الوثائق وإدارة التغيير**

* جميع التغييرات موثقة في Changelog.
* البوابة تعرض نسخة حية لكل إصدار API.
* خطة ترحيل واضحة منشورة مع كل إصدار رئيسي جديد.

## **15) الأدوات والمراجع**

* Swagger UI, Redoc للعرض.
* Postman/Insomnia Collections.
* OpenAPI Generator للـSDKs.
* GitHub Pages أو بوابات API مدفوعة (Kong Dev Portal, ReadMe.com).

## **16) تشيك**

* كل إصدار موثق بـOpenAPI مستقل.
* Changelog محدث ومرفوع.
* البوابة تعرض جميع الإصدارات.
* الأمثلة متطابقة مع التنفيذ.
* SDKs و Postman Collections محدثة.
* سياسة Deprecation منشورة.

## **الملحقات التنفيذية (Artifacts)**

* قالب Changelog (Added/Changed/Deprecated/Removed).
* ملف OpenAPI لكل إصدار (v1, v2…).
* قوالب SDKs مولدة من OpenAPI.
* دليل ترحيل (Migration Guide) بين الإصدارات.

(21) Compatibility & External Integrations

# **(21) التوافقية والتكامل مع الأنظمة الخارجية — Compatibility & External Integrations**

## **1) آلية العمل والاعتماديات**

* جميع التكاملات مع الأنظمة الخارجية (مثل: بوابات الدفع، أنظمة ERP، خدمات الشحن، أنظمة الهوية) تُنفّذ عبر واجهات **موثقة ومعزولة**.
* يتم استخدام **Adapters أو Proxies** لعزل التغييرات الخارجية عن الكود الأساسي.
* جميع التكاملات تمر عبر طبقة Gateway أو Service خاصة تراقب وتطبق السياسات الأمنية.

## **2) الغرض والنطاق**

* ضمان توافق الـAPI مع متطلبات شركاء خارجيين وخدمات طرف ثالث.
* تمكين التوسع عبر تكاملات جديدة دون إعادة بناء الأنظمة الداخلية.
* يشمل التكاملات مع: الدفع، الشحن، التحليلات، التحقق من الهوية، والأنظمة الداخلية للشركاء.

## **3) المدخلات والمخرجات**

* المدخلات: طلبات تكامل (API Calls) من/إلى الأنظمة الخارجية.
* المخرجات: استجابات موحدة عبر الـAPI الداخلي، مع تحويلات (Transformations) إذا اختلفت الصيغ.

## **4) المسؤوليات والأدوار**

* Backend: بناء Adapters للتكامل مع كل نظام خارجي.
* Security: مراجعة عقود الطرف الثالث والتأكد من التحكم بالوصول.
* QA: اختبار التكامل مع بيئات Sandbox للطرف الثالث.
* SRE: مراقبة الأداء والتوافر للتكاملات الخارجية.

## **5) السياسات والمعايير**

* كل تكامل خارجي يجب أن يكون معزول في Service/Module مستقل.
* تسجيل جميع استدعاءات الطرف الثالث مع trace\_id.
* توحيد الاستجابات للأخطاء الخارجية ضمن غلاف الخطأ الداخلي.
* التعامل مع معدلات محددة (Rate Limits) لكل مزود خارجي.
* دعم fallback أو graceful degradation عند فشل الخدمة الخارجية.

## **6) إجراءات التنفيذ**

1. تعريف العقد الخارجي (Swagger/Docs/SDK).
2. بناء Adapter يحول الطلبات/الاستجابات إلى الشكل الداخلي الموحد.
3. إعداد بيئة Sandbox للاختبار.
4. إضافة Retry + Circuit Breaker للتعامل مع الأعطال.
5. مراقبة الأخطاء والزمن مع تصنيفها لكل مزود.

## **7) قبول الجودة**

* جميع التكاملات تمر عبر Adapters.
* الاستجابات الخارجية تعود بالشكل الموحد داخليًا.
* وجود سيناريو Fallback لكل تكامل حساس.
* Sandbox تم اختباره قبل الإطلاق إلى الإنتاج.

## **8) القياس والمراقبة**

* Latency لكل تكامل خارجي.
* معدل الأخطاء لكل مزود.
* نسبة النجاح مقابل الفشل.
* تنبيهات عند تعطل مزود أو بطء زائد.

## **9) التكاملات**

* الدفع: PayPal, Stripe, أو مزود محلي.
* الشحن: DHL, Aramex, FedEx، أو شركات محلية.
* ERP/CRM: Odoo, SAP, Salesforce.
* الهوية: OAuth/OpenID, eKYC APIs.

## **10) الأمان**

* جميع الاستدعاءات عبر HTTPS فقط.
* إدارة أسرار/Merchant Keys في Secret Manager.
* سياسة Least Privilege للمفاتيح.
* فحص الاستجابات لتجنب إدخال بيانات غير موثوقة.

## **11) الأداء والكاش**

* الكاش للردود الخارجية الثابتة (مثل أسعار الصرف اليومية).
* Circuit Breaker لتجنب إغراق النظام عند فشل مزود خارجي.
* Retry بحدود (Exponential Backoff).

## **12) المخاطر والضوابط**

* خطر تعطل مزود خارجي → حل: Fallback أو تعطيل جزئي.
* خطر اختلاف في العقود الخارجية → حل: Adapters + Mapping.
* خطر تجاوز الحد اليومي للمكالمات → حل: مراقبة + Alerts + Queue.

## **13) برنامج الاختبارات**

* اختبارات تكامل (Integration Tests) ضد Sandbox لكل مزود.
* اختبارات أعطال (Simulated Failures) مع Fallback.
* اختبارات أداء تحت ضغط التكاملات.

## **14) الوثائق وإدارة التغيير**

* كل تكامل خارجي موثق في دليل Integrations Handbook.
* تحديث الوثائق عند تغيير واجهة المزود.
* Changelog لأي تعديل في العقود الخارجية.

## **15) الأدوات والمراجع**

* Circuit Breaker Libraries (Hystrix, Resilience4j).
* API Clients/SDKs للمزودين.
* Secret Manager لإدارة المفاتيح.
* Monitoring Tools: Grafana, Datadog.

## **16) تشيك**

* جميع التكاملات عبر Adapters مستقلة.
* أسرار ومفاتيح مخزنة في Secret Manager.
* Retry + Circuit Breaker مفعلان.
* Sandbox مُختبر قبل Prod.
* استجابات موحدة داخليًا للأخطاء الخارجية.
* مراقبة Latency/Error لكل مزود.

## **الملحقات التنفيذية (Artifacts)**

* مصفوفة تكامل: (المزود، نوع الخدمة، بيئة Sandbox، الأسرار، SLA).
* قوالب تحويل (Mapping) لطلب/استجابة لكل مزود.
* Runbook لإدارة أعطال مزود خارجي.

(22) Data Privacy & Compliance

# **(22) إدارة البيانات والخصوصية — Data Privacy & Compliance**

## **1) آلية العمل والاعتماديات**

* جميع البيانات الحساسة (PII, بيانات الدفع, كلمات المرور) تُخزن وتُنقل بشكل مشفر.
* الامتثال لمعايير الخصوصية والأمان مثل: **GDPR, CCPA, PCI-DSS** عند الحاجة.
* تطبيق سياسات **Data Classification** لتحديد مستوى حساسية كل نوع بيانات.
* الاعتماد على **DLP (Data Loss Prevention)** وآليات المراقبة لاكتشاف أي تسريب.

## **2) الغرض والنطاق**

* حماية بيانات العملاء من التسريب أو الوصول غير المصرح.
* ضمان الامتثال للتشريعات المحلية والعالمية.
* يشمل جميع مكونات النظام: API, قواعد البيانات, التخزين, النسخ الاحتياطي, التكاملات.

## **3) المدخلات والمخرجات**

* المدخلات: بيانات العملاء (اسم، بريد، هاتف، عنوان، دفع…).
* المخرجات: خدمات آمنة مع بيانات مخزنة وموزعة بشكل متوافق مع القوانين.

## **4) المسؤوليات والأدوار**

* Security Team: وضع السياسات ومراجعة الامتثال.
* Backend: تنفيذ التشفير والـMasking.
* SRE: مراقبة حركة البيانات وتطبيق سياسات النسخ الاحتياطي.
* Legal/Compliance: مراجعة التوافق مع القوانين.

## **5) السياسات والمعايير**

* **التشفير في الراحة (At Rest)**: قواعد البيانات والتخزين مشفرة (AES-256).
* **التشفير أثناء النقل (In Transit)**: TLS 1.2+ إلزامي.
* **الحد الأدنى من البيانات**: جمع فقط ما هو ضروري (Data Minimization).
* **الاحتفاظ بالبيانات (Retention)**: حذف/إخفاء البيانات بعد انتهاء الحاجة (مثلاً: حذف الحساب بعد 90 يومًا من التعطيل).
* **إخفاء البيانات (Masking)**: إخفاء PII في اللوغ وواجهات الإدارة.
* **إدارة الموافقات (Consent Management)**: العميل يحدد موافقاته (SMS, Email, Push).

## **6) إجراءات التنفيذ**

1. تحديد تصنيف البيانات (Public, Internal, Confidential, Restricted).
2. تفعيل التشفير في قواعد البيانات والتخزين السحابي.
3. فرض سياسات Masking في اللوغ وواجهات الإدارة.
4. إنشاء آليات حذف/محو للبيانات عند الطلب (Right to Erasure).
5. تطبيق Consent Management وربطه بجميع قنوات التواصل.

## **7) قبول الجودة**

* لا يوجد أي بيانات حساسة في اللوغ أو الردود العامة.
* جميع البيانات الحساسة مشفرة في DB/Storage.
* وجود آلية مؤكدة لحذف البيانات عند الطلب.
* وجود تدقيق (Audit Log) على كل وصول للبيانات.

## **8) القياس والمراقبة**

* نسبة البيانات غير المشفرة = 0%.
* تتبع جميع محاولات الوصول لبيانات حساسة.
* إنذارات عند محاولات قراءة غير مصرح بها.
* تقارير امتثال ربع سنوية.

## **9) التكاملات**

* تكامل مع أنظمة DLP وSIEM لمراقبة الحوادث.
* تكامل مع بوابات الدفع PCI-DSS.
* تكامل مع نظام إدارة الهوية (IAM) للتحكم في الوصول.

## **10) الأمان**

* سياسة Least Privilege للوصول إلى البيانات.
* مراجعة صلاحيات DB دورية.
* استخدام أسرار ديناميكية (Dynamic Secrets) بدلاً من ثابتة.
* مراقبة تسرب البيانات عبر أدوات DLP.

## **11) الأداء والكاش**

* البيانات الحساسة لا تُخزَّن في الكاش العام.
* بيانات PII تُعالج فقط عند الحاجة الفعلية.
* الحقول المجهولة (anonymized) تُستخدم لتحليلات الأداء.

## **12) المخاطر والضوابط**

* خطر تسريب بيانات حساسة → تشفير + DLP + Masking.
* خطر الاحتفاظ بالبيانات لفترة أطول → سياسة Retention وإتلاف تلقائي.
* خطر فقدان موافقات العملاء → نظام مركزي لإدارة الموافقات.

## **13) برنامج الاختبارات**

* اختبارات تحقق: PII لا تظهر في الردود أو اللوغ.
* اختبارات تشفير: الحقول الحساسة مشفرة فعليًا.
* اختبارات GDPR: تنفيذ حق النسيان (Right to Erasure).

## **14) الوثائق وإدارة التغيير**

* سياسة الخصوصية منشورة ومحدثة.
* دليل تصنيف البيانات (Data Classification Guide).
* مستند إدارة الموافقات متاح للفريق.

## **15) الأدوات والمراجع**

* DLP Systems (Google DLP, Azure Information Protection).
* SIEM (Splunk, ELK, Datadog Security).
* PCI-DSS Guidelines.
* GDPR/CCPA Compliance Frameworks.

## **16) تشيك**

* جميع البيانات الحساسة مشفرة At Rest وIn Transit.
* PII لا تظهر في اللوغ.
* سياسة Retention مفعلة.
* Consent Management مدمج مع القنوات.
* Audit Logs لجميع الوصول للبيانات.
* تقارير امتثال تصدر دوريًا.

## **الملحقات التنفيذية (Artifacts)**

* مصفوفة تصنيف البيانات (Data Classification Matrix).
* قالب Audit Log لتتبع وصول البيانات.
* نموذج تنفيذ Right to Erasure.
* تقرير امتثال ربع سنوي.

(23) Backup & Disaster Recovery

# **(23) إدارة النسخ الاحتياطية واستعادة الكوارث — Backup & Disaster Recovery**

## **1) آلية العمل والاعتماديات**

* جميع قواعد البيانات وملفات التخزين يتم نسخها احتياطيًا بشكل دوري (يومي/أسبوعي).
* النسخ تُخزن في مواقع متعددة (Multi-Region / Multi-AZ).
* الاعتماد على تشفير كامل للنسخ الاحتياطية (AES-256).
* خطة استعادة الكوارث (DR Plan) معرفة ومجربة بشكل دوري.

## **2) الغرض والنطاق**

* حماية البيانات من الفقدان أو التلف.
* ضمان استمرارية الأعمال عند حدوث أعطال جسيمة (انقطاع، هجوم، فقدان بيانات).
* يشمل جميع البيانات (Databases, File Storage, Configurations, Secrets).

## **3) المدخلات والمخرجات**

* المدخلات: بيانات الإنتاج.
* المخرجات: نسخ احتياطية قابلة للاستعادة، خطة واضحة للعودة إلى الخدمة.

## **4) المسؤوليات والأدوار**

* SRE/Infra: إدارة النسخ الاحتياطية، مراقبتها، اختبار استعادتها.
* Security: التأكد من تشفير النسخ وإدارة الصلاحيات.
* PM/Management: الموافقة على RPO/RTO والموارد اللازمة.

## **5) السياسات والمعايير**

* **RPO (Recovery Point Objective):** أقصى فقد بيانات مسموح (مثلاً ≤ 24 ساعة).
* **RTO (Recovery Time Objective):** أقصى وقت لاستعادة الخدمة (مثلاً ≤ 4 ساعات).
* جميع النسخ مشفرة ومخزنة في مواقع جغرافية مختلفة.
* صلاحيات الوصول للنسخ محدودة للغاية (Least Privilege).

## **6) إجراءات التنفيذ**

1. إعداد نسخ احتياطية أوتوماتيكية لقواعد البيانات (Snapshots).
2. نسخ احتياطي للملفات/الصور إلى تخزين منفصل (Cold Storage).
3. تشفير جميع النسخ الاحتياطية.
4. اختبار الاستعادة كل 3 أشهر.
5. إعداد خطة DR تشمل خطوات إعادة الخدمة، ترتيب الأولويات، وقائمة المسؤولين.

## **7) قبول الجودة**

* جميع النسخ الاحتياطية قابلة للاستعادة بنجاح.
* زمن الاستعادة ≤ RTO.
* فقد البيانات ≤ RPO.
* جميع الاختبارات الدورية ناجحة.

## **8) القياس والمراقبة**

* نسبة نجاح النسخ الاحتياطية.
* نسبة نجاح اختبارات الاستعادة.
* زمن الاستعادة الفعلي مقابل RTO.
* تقارير دورية عن حجم وتكرار النسخ.

## **9) التكاملات**

* تكامل مع أنظمة التخزين السحابي (AWS S3 Glacier, GCP Coldline).
* تكامل مع أنظمة المراقبة لإصدار إنذارات عند فشل النسخ.
* تكامل مع إدارة الأسرار لتشفير مفاتيح النسخ.

## **10) الأمان**

* جميع النسخ مشفرة أثناء النقل وفي الراحة.
* صلاحيات الوصول محدودة ومسجلة في Audit Logs.
* مراجعة دورية لمفاتيح التشفير وسياسات الوصول.

## **11) الأداء والكاش**

* النسخ الاحتياطية تُجرى خارج ساعات الذروة لتقليل الأثر.
* ضغط البيانات قبل النقل لتقليل التكلفة.

## **12) المخاطر والضوابط**

* خطر فشل النسخ → إنذارات فورية + إعادة المحاولة.
* خطر عدم إمكانية الاستعادة → اختبارات DR دورية.
* خطر وصول غير مصرّح به → تشفير + صلاحيات محدودة + مراقبة.

## **13) برنامج الاختبارات**

* اختبار استعادة جزئية (ملف/جدول).
* اختبار استعادة كاملة من نسخة حديثة.
* اختبار استعادة من نسخة قديمة (≥ 6 أشهر).

## **14) الوثائق وإدارة التغيير**

* خطة DR مكتوبة ومحدثة.
* توثيق جميع اختبارات الاستعادة مع نتائجها.
* Changelog لأي تعديل في سياسات النسخ.

## **15) الأدوات والمراجع**

* حلول النسخ الاحتياطي السحابي (AWS Backup, GCP Backup, Azure Recovery Vault).
* أدوات DR Automation مثل Velero, CloudEndure.
* معايير NIST وISO 27001 لاستعادة الكوارث.

## **16) تشيك**

* نسخ احتياطية يومية/أسبوعية مفعلة.
* جميع النسخ مشفرة.
* RPO/RTO محددين ومتفق عليهم.
* اختبارات استعادة تُنفذ كل 3 أشهر.
* خطة DR مكتوبة ومحدثة.
* تقارير نسخ/استعادة محفوظة.

## **الملحقات التنفيذية (Artifacts)**

* خطة DR رسمية (خطوات، أدوار، تسلسل زمني).
* جدول RPO/RTO لكل خدمة رئيسية.
* تقارير اختبارات الاستعادة.
* لوحة Dashboard لمتابعة النسخ الاحتياطية والفشل.

(24) Infrastructure as Code (IaC)

# **(24) إدارة البنية التحتية ككود — Infrastructure as Code (IaC)**

## **1) آلية العمل والاعتماديات**

* البنية التحتية تُدار بشكل كامل عبر ملفات **كود معلن (Declarative Code)** مثل Terraform, Pulumi, أو CloudFormation.
* جميع ملفات IaC مخزنة في مستودع Git مع مراجعات (PRs, Code Review).
* عمليات النشر تتم عبر **CI/CD Pipeline** مع خطوات التحقق والاختبار.

## **2) الغرض والنطاق**

* توفير بيئة قابلة للتكرار (Reproducible) ومتماسكة عبر جميع البيئات (Dev, Staging, Prod).
* ضمان توثيق جميع التغييرات وقابليتها للتدقيق (Auditable).
* يشمل الشبكات، قواعد البيانات، التخزين، الأمن، السياسات، وأي مكونات بنية تحتية.

## **3) المدخلات والمخرجات**

* المدخلات: ملفات IaC (Terraform/Pulumi/YAML).
* المخرجات: بنية تحتية منشورة تلقائيًا ومطابقة للتعريف في الكود.

## **4) المسؤوليات والأدوار**

* SRE/Infra: كتابة وصيانة ملفات IaC.
* Developers: استخدام وحدات (Modules) جاهزة دون تعديل مباشر للبنية.
* Security: مراجعة سياسات الأمن داخل IaC (IAM, Network).
* QA: اختبار البيئات المنشورة عبر IaC.

## **5) السياسات والمعايير**

* جميع التغييرات تمر عبر Git (Pull Requests + Reviews).
* بيئات متعددة (Dev, Staging, Prod) معرفة ومنفصلة.
* عدم وجود تغييرات يدوية (Manual Changes) في البنية.
* جميع الموارد معرفة بشكل صريح مع وسوم (Tags/Labels) للملكية والتكلفة.
* مراجعة دورية للـModules لإزالة التكرار وتحسين الأمان.

## **6) إجراءات التنفيذ**

1. إنشاء مستودع IaC مستقل أو مجلد فرعي مخصص.
2. تعريف الموارد كوحدات (Modules) قابلة لإعادة الاستخدام.
3. استخدام Workspaces/Environments للفصل بين البيئات.
4. إعداد CI/CD لتشغيل terraform plan أو ما يعادلها على كل PR.
5. نشر تلقائي بعد الموافقة، مع حفظ حالة (State) في Storage آمن (S3/GCS + Locking).

## **7) قبول الجودة**

* جميع الموارد منشورة من الكود فقط.
* نتائج terraform plan أو ما يعادلها مطابقة للتوقعات.
* لا يوجد أي تعديل يدوي خارج IaC.
* وجود وسوم/Labels موحدة لكل الموارد.

## **8) القياس والمراقبة**

* نسبة التغطية عبر IaC (100% من الموارد تُدار بالكود).
* عدد الانحرافات (Drift) المكتشفة بين IaC والبنية الفعلية.
* تقارير دورية عن استخدام التكاليف عبر الوسوم.

## **9) التكاملات**

* تكامل مع CI/CD للتحقق والنشر.
* تكامل مع أنظمة المراقبة لمطابقة التعريف مع الواقع (Drift Detection).
* تكامل مع أنظمة Security Scan لفحص Misconfigurations.

## **10) الأمان**

* تخزين حالة IaC (State) في تخزين مشفر مع قفل (Lock).
* مراجعة IAM Roles ضمن الكود بشكل دوري.
* سياسة Least Privilege في حسابات الخدمة.
* تشغيل أدوات Security Scan (Checkov, tfsec).

## **11) الأداء والكاش**

* استخدام وحدات جاهزة لتقليل وقت النشر.
* Parallelism مضبوط لتقليل وقت التنفيذ دون التسبب في أعطال.

## **12) المخاطر والضوابط**

* خطر Drift (تباين بين الكود والبنية) → تشغيل فحص دوري.
* خطر Misconfiguration أمني → استخدام Scanners + Code Review.
* خطر تدمير موارد حساسة → تطبيق سياسات حماية (Prevent Destroy).

## **13) برنامج الاختبارات**

* اختبار نشر بيئة مؤقتة (Ephemeral Environment) عبر IaC.
* اختبار Regression عبر مقارنة plan القديم والجديد.
* اختبارات أمان للتأكد من IAM/Networking.

## **14) الوثائق وإدارة التغيير**

* دليل IaC Handbook يشرح السياسات والمعايير.
* جميع التغييرات موثقة عبر PRs + ADRs عند تغييرات كبيرة.
* سجل Changelog للبنية محدث دوريًا.

## **15) الأدوات والمراجع**

* Terraform, Pulumi, CloudFormation.
* Vault/Secret Manager لإدارة الأسرار.
* Checkov, tfsec, Terrascan لفحص الأمان.
* Atlantis أو Spacelift لإدارة خطط IaC عبر PRs.

## **16) تشيك**

* 100% من البنية مُدارة بالكود.
* جميع التغييرات عبر PR + Code Review.
* بيئات منفصلة (Dev/Staging/Prod) معرفة.
* State مخزن في مكان آمن ومشفر.
* Drift Detection دوري.
* وسوم/Labels موحدة لكل الموارد.
* Security Scan مفعّل عند كل PR.

## **الملحقات التنفيذية (Artifacts)**

* قوالب Modules جاهزة (DB, VPC, Redis, CDN).
* ملف سياسات IaC (Prevent Destroy, Mandatory Tags).
* تقارير Drift Detection أسبوعية.
* Dashboard يوضح نسبة التغطية بالكود + التكاليف.

(25) Cost Management & FinOps

# **(25) إدارة التكاليف والمراقبة المالية — Cost Management & FinOps**

## **1) آلية العمل والاعتماديات**

* جميع الموارد السحابية والبنية التحتية مربوطة بوسوم (Tags/Labels) للملكية، البيئة، والفريق.
* المراقبة تتم عبر لوحات تحكم مالية (Cost Explorer, BigQuery Billing, Datadog, CloudHealth).
* الاعتماد على **FinOps Practices**: الموازنة بين الأداء والتكلفة، تحسين الاستهلاك، وخفض الهدر.

## **2) الغرض والنطاق**

* ضمان أن التكاليف تحت السيطرة ومبررة.
* توفير شفافية لكل فريق حول استهلاك موارده.
* يشمل: قواعد البيانات، التخزين، الشبكات، الخدمات المُدارة، والأدوات الخارجية (3rd Party SaaS).

## **3) المدخلات والمخرجات**

* المدخلات: فواتير مزود الخدمة السحابية + وسوم الموارد.
* المخرجات: تقارير مالية مفصلة، تنبيهات تجاوز الميزانية، توصيات تحسين.

## **4) المسؤوليات والأدوار**

* Finance/Management: تحديد الميزانية ومتابعة التقارير الشهرية.
* SRE/Infra: ضبط الوسوم، مراقبة الاستهلاك، تطبيق التحسينات.
* Backend: كتابة كود وكويري فعال يقلل التكلفة (DB, API Calls).
* Product/PM: الموازنة بين التكاليف ومتطلبات الميزات.

## **5) السياسات والمعايير**

* جميع الموارد يجب أن تحتوي على وسوم: env, team, owner, project.
* وضع حدود ميزانية لكل بيئة (Dev/Staging/Prod).
* تفعيل Alerts عند تجاوز 80% من الميزانية.
* مراجعة شهرية لتوزيع التكاليف ومناقشتها.

## **6) إجراءات التنفيذ**

1. تفعيل التتبع المالي على مستوى الحساب/المشروع.
2. إلزام الوسوم قبل إنشاء أي مورد جديد (Policy as Code).
3. إعداد تقارير يومية/أسبوعية عن الاستهلاك.
4. إعداد توصيات (Rightsizing, Spot Instances, Auto-scaling).
5. إجراء مراجعة FinOps ربع سنوية.

## **7) قبول الجودة**

* كل مورد مرتبط بفريق/بيئة عبر الوسوم.
* لا توجد موارد يتيمة (Orphaned Resources).
* التنبيهات تنطلق عند تجاوز الحد.
* التكاليف متسقة مع توقعات الميزانية.

## **8) القياس والمراقبة**

* التكلفة الشهرية الإجمالية مقابل الميزانية.
* تكلفة كل خدمة/فريق/بيئة.
* معدل النمو (MoM) في التكاليف.
* نسبة الهدر المكتشف والمحسَّن.

## **9) التكاملات**

* تكامل مع Google BigQuery/Looker لإنشاء لوحات تحليلية.
* تكامل مع أدوات FinOps مثل CloudHealth, Apptio.
* تكامل مع Slack/Email لإرسال تنبيهات مالية.

## **10) الأمان**

* الوصول لتقارير التكاليف مقيد للأدوار المالية والإدارية فقط.
* إخفاء أي بيانات دفع أو فواتير حساسة.

## **11) الأداء والكاش**

* استخدام Auto-scaling لتقليل الهدر.
* تفعيل التخزين البارد/الأرشفة للبيانات قليلة الاستخدام.
* تحسين استعلامات DB الثقيلة لتقليل استهلاك الموارد.

## **12) المخاطر والضوابط**

* خطر نمو غير مراقب في التكاليف → حل: Alerts + مراجعة أسبوعية.
* خطر موارد يتيمة → حل: سياسات تنظيف (Orphan Cleanup).
* خطر هدر الموارد (Over-provisioning) → حل: Rightsizing + Spot.

## **13) برنامج الاختبارات**

* اختبار إنشاء موارد بدون وسوم → رفض.
* اختبار Alerts عند تجاوز 80% من الميزانية.
* اختبار لوحات التحليل تعرض بيانات صحيحة.

## **14) الوثائق وإدارة التغيير**

* دليل FinOps منشور للفريق.
* تحديث شهري للتقارير وإرسالها للإدارة.
* Changelog عند تعديل سياسات الميزانية.

## **15) الأدوات والمراجع**

* Cloud Billing APIs (AWS, GCP, Azure).
* FinOps Framework.
* CloudHealth, Apptio.
* BigQuery + Looker Studio.

## **16) تشيك**

* جميع الموارد مربوطة بوسوم إلزامية.
* حدود الميزانية محددة لكل بيئة.
* Alerts عند 80% من الميزانية.
* تقارير أسبوعية/شهرية صادرة.
* توصيات Rightsizing مطبقة.
* مراجعة ربع سنوية FinOps منفذة.

## **الملحقات التنفيذية (Artifacts)**

* مصفوفة وسوم الموارد (env, team, owner, project).
* تقرير شهري بالتكاليف مقابل الميزانية.
* Dashboard تفاعلي يوضح التكاليف حسب الخدمة/الفريق.
* Runbook لتطبيق تحسينات Rightsizing و Spot Instances.

(26) Identity & Access Management (IAM)

# **(26) إدارة الوصول والهويات — Identity & Access Management (IAM)**

## **1) آلية العمل والاعتماديات**

* جميع الخدمات والمستخدمين يتعاملون مع النظام عبر **هوية رقمية** (User/Service Identity).
* يتم إصدار وتوثيق الوصول باستخدام **JWT/OAuth2/OpenID Connect** أو مفاتيح قصيرة العمر (Short-lived Tokens).
* الاعتماد على **Role-Based Access Control (RBAC)** أو **Attribute-Based Access Control (ABAC)** لإدارة الصلاحيات.
* الأسرار (Secrets, API Keys) تُدار عبر Secret Manager آمن.

## **2) الغرض والنطاق**

* ضمان أن كل عملية وصول مرتبطة بهوية معروفة وصلاحيات محددة.
* منع الوصول غير المصرح إلى الموارد أو البيانات الحساسة.
* يشمل: العملاء (Users), الشركاء (Partners), الأنظمة الداخلية (Services).

## **3) المدخلات والمخرجات**

* المدخلات: بيانات اعتماد (Credentials) مثل Token أو API Key.
* المخرجات: وصول مصرح أو رفض (401/403).

## **4) المسؤوليات والأدوار**

* Security Team: وضع السياسات ومراجعة الامتثال.
* Backend: تطبيق التحقق من الهوية والصلاحيات.
* SRE/Infra: إدارة أنظمة IAM، تدوير المفاتيح والأسرار.
* QA: اختبار السيناريوهات (مصرح، غير مصرح، صلاحيات ناقصة).

## **5) السياسات والمعايير**

* جميع الاتصالات عبر HTTPS مع مصادقة إلزامية.
* JWT Tokens موقعة ومحدودة العمر (مثلاً 15 دقيقة + Refresh Token).
* API Keys قصيرة العمر مع تجديد تلقائي.
* RBAC/ABAC مفعل مع أدوار معرفة مسبقًا (Admin, Support, User…).
* جميع الأحداث الأمنية تسجل في Audit Logs.
* تدوير المفاتيح/الأسرار بشكل دوري (90–180 يومًا).

## **6) إجراءات التنفيذ**

1. إنشاء نظام إصدار هوية (Auth Service) يدعم OAuth2/OpenID.
2. إضافة Middleware للتحقق من JWT/Token في كل API.
3. ربط الصلاحيات بالأدوار أو السمات.
4. تفعيل سياسات Least Privilege (أقل صلاحيات ممكنة).
5. إدارة المفاتيح عبر Secret Manager مع صلاحيات وصول محددة.

## **7) قبول الجودة**

* كل طلب بدون توثيق → 401.
* كل طلب مع صلاحيات ناقصة → 403.
* جميع الـTokens قصيرة العمر وموقعة بشكل صحيح.
* Audit Log يحتوي على جميع عمليات تسجيل الدخول ومحاولات الفشل.

## **8) القياس والمراقبة**

* عدد محاولات تسجيل الدخول الناجحة والفاشلة.
* استخدام كل دور (Role Usage).
* تنبيهات عند محاولات brute force أو استخدام مفاتيح منتهية.

## **9) التكاملات**

* التكامل مع مزودي الهوية (Google, Azure AD, Keycloak).
* التكامل مع أنظمة الشركاء عبر OAuth2 Client Credentials.
* التكامل مع إدارة الأجهزة (MDM) عند الحاجة.

## **10) الأمان**

* جميع المفاتيح مخزنة في Secret Manager فقط.
* 2FA إلزامي لحسابات المسؤولين.
* حسابات الخدمة مقيدة بصلاحيات دقيقة.
* مراقبة وصول غريب أو غير متوقع عبر SIEM.

## **11) الأداء والكاش**

* تخزين مؤقت (Caching) لنتائج التحقق من التوكنات لتقليل الحمل.
* استخدام JWKS Endpoint للتحقق من التوقيعات بشكل فعال.

## **12) المخاطر والضوابط**

* خطر سرقة Token → استخدام صلاحية قصيرة + تجديد + Revoke.
* خطر صلاحيات واسعة → مراجعة دورية للأدوار والصلاحيات.
* خطر تسريب مفاتيح → Secret Manager + تدقيق الوصول.

## **13) برنامج الاختبارات**

* اختبار تسجيل الدخول (صحيح/خاطئ).
* اختبار الوصول المصرح/غير المصرح.
* اختبار انتهاء التوكن وتجديده.
* اختبار RBAC/ABAC لكل دور.

## **14) الوثائق وإدارة التغيير**

* دليل IAM Handbook يشرح الأدوار والصلاحيات.
* توثيق جميع Flows (Auth, Refresh, Revoke).
* Changelog عند تعديل أي دور أو صلاحية.

## **15) الأدوات والمراجع**

* Keycloak, Auth0, Okta.
* HashiCorp Vault, AWS Secrets Manager, GCP Secret Manager.
* OAuth2 / OpenID Connect Standards.
* NIST IAM Guidelines.

## **16) تشيك**

* جميع الطلبات تتطلب Auth عبر Token.
* جميع الـTokens قصيرة العمر وموقعة.
* RBAC/ABAC مفعل ومحدود الصلاحيات.
* Audit Log يسجل كل عمليات Auth.
* 2FA مفعلة للحسابات الحساسة.
* جميع المفاتيح/الأسرار في Secret Manager.

## **الملحقات التنفيذية (Artifacts)**

* مصفوفة Roles & Permissions.
* مخطط تدفق OAuth2/OpenID.
* قالب Audit Log لمحاولات Auth.
* خطة تدوير المفاتيح والأسرار.

علامة التبويب 37

# **سياسة لإدارة الهوية والوصول (IAM)**

Flutter App ⇢ API Gateway/Reverse Proxy ⇢ **خدمة الهوية (IdP)** ⇢ مخزن الهوية (MySQL/Redis + مفاتيح JWKS) ⇢ **سياسات الوصول (RBAC/ABAC)** ⇢ خدمات الدومين (الطلبات/المنتجات/…)

* على الحافة: DNS + CDN + WAF + Load Balancer
* حول المنظومة: أمن الأسرار (KMS/HSM)، محدد معدّل، سجلات تدقيق، CMP للموافقات، تحليلات (Firebase/GA4 مع user\_id)، مستودع بيانات (BigQuery) ولوحات (Metabase/Looker).

# **- الغرض والنطاق**

**المادة 1 – الغرض:** ضمان هوية موحّدة وآمنة للمستخدمين والموظفين، وتمكين تحكم دقيق في الوصول، مع امتثال للخصوصية واستمرارية عالية.  
**المادة 2 – النطاق:** تسري هذه السياسة على: تطبيق زهراء (Flutter)، لوحة التحكم، الـ API/Gateway، خدمة الهوية، قاعدة البيانات، Redis، الأدوات والتكاملات (دفع، SMS/WhatsApp/Email/Push، الشحن، التحليلات)، عبر بيئات **Dev/Staging/Prod**.

# **- التعاريف**

* **IdP:** مزوّد الهوية (OIDC/OAuth 2.1).
* **AuthN/AuthZ:** التحقق من الهوية/التخويل.
* **RBAC/ABAC:** أدوار وصلاحيات / سمات وحالات.
* **JWT (Access/Refresh):** رموز الوصول/التجديد موقّعة بمفاتيح دورانية (JWKS).
* **2FA/TOTP/OTP:** عامل ثانٍ (رمز مؤقت عبر تطبيق أو رسالة).
* **Consent/CMP:** موافقات القنوات والخصوصية.
* **Audit Log:** سجل تدقيق غير قابل للتلاعب.

# **- المبادئ العامة**

* **أمان أولًا + تجربة سلسة.**
* **أقل صلاحية وفصل الواجبات.**
* **هوية موحّدة عبر كل القنوات.**
* **الخصوصية بالتصميم (لا نرسل PII غير لازمة للتحليلات).**
* **قابليّة التوسع والتغيير (Key Rotation، إصدار واجهات).**

# **- الأدوار والمسؤوليات**

**الأدوار (RBAC مقترح):**

* **customer:** إدارة حسابه/طلباته.
* **support\_agent:** خدمة العملاء والمرتجعات (بدون بيانات دفع كاملة).
* **merchandiser:** كتالوج/صور/قواعد عرض (بدون أسعار مالية).
* **ops\_logistics:** الشحن والتتبع.
* **finance:** تسويات/استردادات/تقارير.
* **marketing:** حملات/قسائم/قنوات.
* **admin:** إدارة المستخدمين والأدوار.
* **superadmin (مقيّد جدًا):** تغييرات أمنية/مفاتيح وسياسات.

**المسؤوليات:**

* **مالك خدمة الهوية (IAM Owner):** الامتثال والتغيير والإصدارات.
* **أمن المعلومات (SecOps):** المفاتيح/السجلات/التدقيق/الاستجابة للحوادث.
* **SRE:** الإتاحة والمراقبة وSLO.
* **التطوير (Backend/App):** التزام الواجهات والسياسات.
* **العمليات:** تفعيل/إلغاء صلاحيات الموظفين (On/Offboarding).

# **- ضوابط الهوية والوصول**

## **10 – التسجيل وتسجيل الدخول**

* قنوات دخول معتمدة: **OTP بالهاتف** (أساسي للأسواق المحلية)، **Password + TOTP**، وSSO للموظفين إن توفر.
* **محدد معدّل** لطلبات /auth/\*، تبريد بعد فشل متكرر.
* **تحقق ملكية القناة** (الهاتف/الإيميل) قبل تفعيلها.

## **11 – إدارة الاعتمادات**

* كلمات المرور: **Argon2id** (أو bcrypt ≥ 12)، قائمة منع كلمات شائعة، طول ≥ 10.
* **2FA إلزامي** لموظفي لوحة التحكم (TOTP مفضل).
* **OTP**: صلاحية 3–5 دقائق، 5 محاولات، قنوات بديلة (WhatsApp/اتصال آلي).

## **12 – الجلسات والرموز**

* **Access JWT** قصير (10–20 دقيقة)، **Refresh** 30 يومًا مع **دوران + كشف إعادة الاستخدام** (Token Family).
* تضمين sid وdevice\_id وroles في Claims المناسبة.
* **تجديد صامت** قبل انتهاء access (عند 70–80% من TTL).

## **13 – الترخيص (AuthZ)**

* **RBAC** افتراضي، و**ABAC** حيث يلزم (مثلاً نطاق منطقة/قناة).
* فصل الواجبات: إنشاء قسيمة ≠ اعتمادها.
* سياسات دقيقة للوحة التحكم (رؤية/تعديل/اعتماد).

## **14 – إدارة المفاتيح والأسرار**

* مفاتيح توقيع JWT داخل **KMS/HSM** مع **Key Rotation نصف سنوي**.
* نشر مفاتيح عامة عبر **JWKS** موقّع، مع **KID** للإصدارات.
* أسرار ومفاتيح تكاملات في مخزن أسرار؛ حظر وجودها في الكود/البيئات المشتركة.

## **15 – حدود المعدّل والحافة**

* Rate Limit على /auth/\*, /admin/\*, /checkout/\*.
* **TLS فقط** + **HSTS** + تحديثات بروتوكول (HTTP/2/3).
* **CORS** مقيّد: أصول تطبيق زهراء فقط، رؤوس/طرق محددة.

## **16 – الخصوصية والموافقات (CMP)**

* تخزين **consents** لكل قناة (Push/SMS/WA/Email) واحترامها في أي إرسال.
* عدم إرسال أرقام الهواتف/الإيميلات للتحليلات؛ فقط **user\_id**.
* سياسات الاحتفاظ: حذف/تصحيح/تصدير بيانات عند الطلب.

## **17 – السجلات والتدقيق**

* Logs مهيكلة (JSON) لكل: طلب OTP، فشل/نجاح دخول، إنشاء/إبطال جلسة، تغيير دور/صلاحية.
* **Trace ID** من الحافة حتى IdP؛ ساعات حفظ السجلات وفق الامتثال.
* تقارير شهرية للأنشطة عالية الحساسية.

## **18 – أمن لوحة التحكم**

* خلف **WAF + Allowlist IP/VPN + SSO + 2FA**.
* قفل تلقائي عند نشاط مشبوه؛ مراجعة أدوار ربع سنوية.

# **- التكاملات والاعتماديات**

* **Firebase/GA4:** تعيين user\_id = users.id بعد التحقق فقط.
* **قنوات الإشعار:** تفعيل بعد إثبات ملكية القناة وربطها بالموافقات.
* **الدفع:** عدم تخزين بيانات بطاقات خام؛ استخدام رموز/آخر 4 أرقام للعرض.

# **- SLO/SLI ومؤشرات الأداء**

| **المؤشر (SLI)** | **SLO مستهدف** | **طريقة القياس** |
| --- | --- | --- |
| إتاحة خدمة IdP | ≥ 99.9% شهريًا | Uptime من المراقبة الخارجية والداخلية |
| p95 زمن إصدار Access Token | ≤ 300ms | تتبع من بوابة الـ API/IdP |
| نجاح تسليم OTP | ≥ 98% | مزوّدو الرسائل + سجلات IdP |
| معدل نجاح تسجيل الدخول | ≥ 97% | نسبة نجاح/محاولات حسب القناة |
| معدل False Positives على WAF لمسارات auth | ≤ 1% | مقارنة الحظر بمحاولات شرعية |
| معدل إعادة استخدام Refresh المكتشفة | 0% | سجلات Token Family/Re-use |
| زمن JWKS fetch/cached | ≤ 50ms | مراقبة الحافة/الخدمات |

**إنذار:** إذا فشل SLO لأي مؤشرين متتالين، يلزم **خطة تحسين** ومراجعة ضبط.

# **- إدارة التغيير والإصدارات**

* إصدارات واجهات OIDC/OAuth مُعلَمة (semver) وتوافق خلفي حيث أمكن.
* **Feature Flags** لطرح تدريجي (Canary 5–10%) مع خط رجعة سريع.
* تغييرات مفاتيح/محددات معدّل تمر عبر **CAB** (Change Advisory Board) مصغّر.

# **- دورة حياة الوصول (موظفون)**

* **Onboarding:** SSO + 2FA، تعيين دور أدنى ممكن، تدريب إلزامي على السياسة.
* **Review ربع سنوي للصلاحيات** (Access Recertification).
* **Offboarding خلال 24 ساعة:** إبطال الجلسات والمفاتيح، سحب الأدوار والوصول إلى الأدوات.

# **- Runbooks (إجراءات جاهزة)**

## **. تسريب مفاتيح JWT (تعرّض KMS/HSM)**

1. تدوير فوري للمفاتيح (إنشاء KID جديد) ونشر JWKS.
2. إبطال كل الـ Refresh Tokens (أو العائلات المتأثرة).
3. فرض إعادة تسجيل الدخول على الجميع، ومراجعة السجلات للأضرار.
4. تقرير حادث + إجراءات منع مستقبلية.

## **. اكتشاف إعادة استخدام Refresh**

1. قفل **Token Family** المتأثر + إنهاء الجلسات.
2. إشعار المستخدم وإجباره على 2FA.
3. تحليل مصدر التسريب (جهاز/شبكة/تطبيق).
4. تحديث سياسات العميل (Storage/HttpOnly/Clear on logout).

## **. فشل واسع في OTP (مزود رسائل)**

1. تحويل تلقائي لقناة احتياطية (WA/اتصال آلي).
2. رفع معدّل إعادة المحاولة وإظهار رسالة توجيهية للمستخدم.
3. تواصل مع المزود + سجل إنذار؛ العودة عند الاستقرار.

## **. نشاط مشبوه على لوحة التحكم**

1. قفل حساب/جلسة المستخدم المتأثر.
2. مراجعة Audit Logs للعمليات الحساسة.
3. فرض إعادة تعيين كلمة مرور و2FA.
4. إرسال تقرير للمدير الأمني واعتماد الدروس المستفادة.

## **. انقطاع IdP**

1. تفعيل **وضع قراءة فقط** حيث أمكن، والسماح بجلسات صالحة قائمة.
2. تحويل المرور عبر مركز بديل إن معدّ.
3. التواصل الداخلي والخارجي بجدول زمني للاستعادة.
4. Postmortem خلال 72 ساعة.

# **- الامتثال والمعايير المرجعية**

* **OWASP ASVS/Top10، NIST 800-63** (الهوية الرقمية)، **OAuth 2.1 / OIDC**.
* سياسات الشركة الأخرى: أمن المعلومات، الخصوصية، إدارة البيانات، الاستمرارية.

# **- المراقبة والتقارير**

* لوحات مباشرة: Login Success, OTP Delivery, p95 Token Issue, Refresh Reuse, IdP Uptime.
* تقارير أسبوعية للفِرق التشغيلية، وشهرية للإدارة، وPostmortem للحوادث.

# **- الملاحق**

* **مخطط تدفق** (كما في الملخص).
* **نموذج جداول قاعدة البيانات**(users/emails/phones/credentials/identities/sessions/refresh\_tokens/roles/role\_permissions/permissions/consents/audit\_logs).
* **نماذج رسائل** للمستخدم عند الحوادث الأمنية.

## **ملاحظات سياقية (المنطقة/الشبكات)**

* شبكات متذبذبة ⇒ جدّد Access مبكرًا + نافذة سماح قصيرة.
* اعتمد **OTP متعدد القنوات** افتراضيًا.
* لا تربط امتيازات حساسة بنجاح الدفع؛ اربطها بهوية موثوقة وموافقة قنوات فقط.

المرحلة الثانية (Implementation & Ops Layer)

# **المرحلة الثانية — التنفيذ والمراقبة (Implementation & Ops Layer)**

## **1) معايير التطوير (Coding Standards)**

* توحيد Style Guides (Laravel, Java, Flutter).
* أسماء متسقة (متغيرات، ملفات، جداول DB).
* Git Workflow موحد (Branches, PRs, Commit Messages).
* قوالب جاهزة (Boilerplates) للخدمات الجديدة لبدء العمل بشكل ملتزم بالمعايير.

## **2) جودة الكود عبر CI/CD (Quality Gates)**

* جميع الـPR تمر عبر:
  + **Lint & Static Analysis**.
  + **Unit & Integration Tests**.
  + **Contract Tests ضد OpenAPI**.
  + **Security Scans (SAST, Dependency Scan)**.
* أي كود لا يجتاز هذه المراحل يُرفض تلقائيًا.

## **3) كتالوج الخدمات (Service Catalog & API Registry)**

* سجل مركزي لكل خدمة يوضح: الاسم، المالك، الإصدار، SLOs، روابط الوثائق وDashboards.
* ربط مع Developer Portal داخلي/خارجي.

## **4) المراقبة والتتبع (Monitoring & Observability)**

* Logging منظم بصيغة JSON مع trace\_id.
* Metrics رئيسية (Latency, Error Rate, QPS).
* Tracing عبر OpenTelemetry/Jaeger.
* Dashboards جاهزة + Alerts آلية عند تخطي SLO.

## **5) إدارة الحوادث (Runbooks & Incident Management)**

* Runbooks جاهزة لحالات الأعطال المتكررة (DB Down, Cache Fail, Payment Timeout).
* Incident Response Plan: On-call, Incident Commander, Escalation.
* ربط مع PagerDuty/Slack للتنبيهات.

## **6) التليمتري والبيانات (Telemetry & Data)**

* كل API يرسل بيانات موحدة: trace\_id, user\_id, event\_type.
* تكامل مع Data Warehouse (BigQuery, GA4, Firebase).
* آليات فحص جودة البيانات (Data Quality Checks).

## **7) تطبيق الأمان (Security Hardening)**

* IAM مفعل (RBAC/ABAC).
* إدارة الأسرار عبر Secret Manager فقط.
* تفعيل WAF, Rate Limiting, CORS على API Gateway.
* مراجعات أمنية واختبارات اختراق دورية.

## **8) إدارة البيئات (Environment Management)**

* فصل كامل بين Dev, Staging, Prod.
* استراتيجيات نشر آمنة (Blue-Green, Canary).
* بيانات وهمية في البيئات غير الإنتاجية.

## **9) الحوكمة التشغيلية (Operational Governance)**

* أي تغيير → ADR + Changelog إلزامي.
* تقارير Error Budget & SLOs أسبوعية/شهرية.
* اجتماعات مراجعة تشغيلية دورية مع الفريق.

### **الخلاصة**

المرحلة الثانية تهدف إلى:

* **الكود ملتزم** (Coding Standards + Quality Gates).
* **الخدمات موثقة ومراقبة** (Service Catalog + Observability).
* **التشغيل منظم** (Runbooks + Incident Management).
* **البيانات موحدة** (Telemetry + Data Quality).
* **الأمان مطبق فعليًا** (IAM, Secrets, WAF).
* **البيئات والنشر آمنة** (Env Separation + Canary/Blue-Green).
* **الحوكمة واضحة** (ADR + Error Budget Reports).

المرحلة الثالثة (Data & Analytics Layer)

# **المرحلة الثالثة — البيانات والتحليلات (Data & Analytics Layer)**

## **0) الهدف والنطاق**

* **الهدف:** تحويل كل عمليات الـAPI إلى بيانات موحّدة قابلة للقياس، التحليل، واتخاذ القرار (تشغيليًا وتجاريًا).
* **النطاق:** جمع البيانات، المنصة، الجودة، الحوكمة، النماذج/المقاييس، اللوحات/التقارير، التكاملات، ML/AI، التكلفة، الأمن، وخطة التنفيذ.

## **1) المبادئ الحاكمة**

* **Data as a Product:** لكل مجموعة بيانات مالك، تعريف، وSLA.
* **Schema/Contract First:** لا بيانات بدون مخطط/عقد مُعلن.
* **Privacy by Design:** PII مُخفاة/محدودة منذ المصدر.
* **Single Source of Truth:** المستودع التحليلي هو المرجع الرسمي للأرقام.
* **Observability for Data:** مراقبة الجودة والحداثة مثل مراقبة الخدمات.

**2) العمارة العامة (Architecture)**

* **المصادر:**
  + تشغيلية: API Logs/Events، قواعد المعاملات (OLTP).
  + خارجية: الدفع، الشحن، CRM/ERP، إعلانات.
  + تطبيقات العميل: GA4/Firebase/SDKs.
* **الاستخلاص والتحميل:**
  + **Streaming** (Kafka/PubSub) للأحداث اللحظية.
  + **Batch ETL/ELT** (Airflow/dbt/Dataform) للدُفعات.
  + **CDC** (Change Data Capture) لجداول OLTP الحرجة.
* **التخزين:**
  + **Data Lake** (Raw Files) + **Data Warehouse** (BigQuery/Snowflake/Redshift).
  + **Feature Store** لسمات ML (offline/online).
* **المناطق (Zones):** Raw → Staging/Processed → Curated/Analytics → Sandbox.
* **الميتاداتا والأنساب:** **Data Catalog** + **Lineage** + **Schema Registry**.
* **الوصول والتحليل:** BI/Notebooks/Reverse ETL.

## **3) عقود البيانات والمخططات (Data Contracts & Schemas)**

* **الأحداث (Events):** تسمية موحّدة domain.action.version (مثل order.created.v1).
* حقول إلزامية: event\_id, event\_name, trace\_id, user\_id (أو anon\_id)، timestamp (UTC), app\_version, device, region.
* **التوافقية:** السماح بإضافة حقول جديدة بدون كسر (backward-compatible)، والإصدارات الكبرى عند التغييرات الكاسرة.
* **Schema Registry:** تحقّق تلقائي قبل قبول الحدث (رفض/Quarantine عند الفشل).

**4) جمع البيانات (Instrumentation & Ingestion)**

* **Telemetry داخل الخدمات والواجهات:** تضمين trace\_id عبر الرحلة.
* **هوية المستخدم:** توحيد user\_id وحالات المجهول anon\_id + قواعد دمج الحسابات (Identity Resolution).
* **SLA للتغذية:**
  + لحظي (Streaming): ≤ دقيقة.
  + دفعي (Batch): 15–60 دقيقة حسب الجدول.
* **التعامل مع الفشل:** DLQ/Retry/Quarantine مع تنبيهات.

## **5) جودة البيانات (Data Quality)**

* **قواعد إلزامية:**
  + Completeness (لا Null في الحقول الأساسية).
  + Validity (تواريخ ISO-8601 UTC، عملة إلزامية للمبالغ).
  + Uniqueness (مفتاح مركّب للأحداث event\_id + وقت).
  + Freshness (تحديث ضمن SLA).
  + Referential Integrity (مفاتيح أجنبية صحيحة).
* **فحوص آلية:** dbt tests/Great Expectations + تقارير.
* **لوحات جودة:** درجات DQ لكل جدول/حدث + إنذارات عند تجاوز العتبات.
* **مسار أخطاء:** جداول عزل (Quarantine) + إعادة معالجة (Replay).

**6) الحوكمة والامتثال (Governance & Compliance)**

* **تصنيف البيانات:** Public / Internal / Confidential / Restricted.
* **الوصول:** RBAC/ABAC على مستوى الجداول والأعمدة (Column-Level Security).
* **الخصوصية:** Masking/Tokenization للـPII، وسياسات **Retention** (حذف/إخفاء بعد المدة).
* **Data Catalog إلزامي:** المالك، الوصف، الحساسية، المصدر، معدّل التحديث، أصحاب المصلحة.
* **التدقيق:** Audit Logs لجميع عمليات الوصول/الاستخراج.

## **7) النماذج التحليلية والطبقة الدلالية (Models & Semantic Layer)**

* **طبقة دلالية موحّدة (Semantic/Metric Layer):** تعريف مقاييس قياسية قابلة لإعادة الاستخدام:
  + **تشغيلية:** Latency p95, Error Rate, Availability, QPS.
  + **عملاء:** FRM، Retention، Churn، CLV (Cohort-based).
  + **مبيعات:** ARPU، Conversion Funnel، Return Rate، Gross/Net Revenue.
  + **تسويق:** ROAS, CAC, LTV/CAC، Attribution (Last/First/Multi-Touch).
* **نماذج بيانات:**
  + **Dimensions** (Customers, Products, Channels) + **Facts** (Orders, Events, Payments).
  + SCD Type 2 للأبعاد المتغيرة (مثل تغير فئة المنتج).

## **8) اللوحات والتقارير (Dashboards & Reporting)**

* **تشغيلية (قريبة للحظية):** Latency/Error/Availability/Orders per Minute (تحديث ≤ 5 دقائق).
* **تجارية/تسويقية:** مبيعات يومية/شهرية، Retention، Conversion، ROAS، CAC.
* **إدارية:** ملخص تنفيذي أسبوعي/شهري مع اتجاهات وForecast مبسّط.
* **حكم التحديث (Refresh SLAs):**
  + تشغيلية: ≤ 5 دقائق.
  + أعمال: ≤ ساعة.
  + تنفيذية: مرّة يوميًا.

**9) التكاملات التشغيلية (Operational & Business Integrations)**

* **Reverse ETL:** دفع الشرائح (Segments) إلى CRM/ESP/Ads (VIP, Churn-risk).
* **ERP/Inventory:** مزامنة الطلبات والمخزون لأتمتة القرارات.
* **التسويق:** تتبع الحملة → ربط مباشر بالمبيعات وCLV.

## **10) ML/AI (الجاهزية وخط التشغيل)**

* **MLOps Pipeline:** إعداد بيانات نظيفة، تقسيم Train/Validation، تتبّع التجارب، إدارة النماذج (Registry).
* **Feature Store:** سمات جاهزة (تفضيلات، تواتر الشراء، متوسط السلة).
* **سيناريوهات أولية:** توصية المنتجات، توقع الطلب، رصد الاحتيال، تحليل انطباعات النصوص (Sentiment).
* **المراقبة:** Drift, Performance, Bias + تنبيهات.

## **11) كفاءة التكلفة (FinOps for Data)**

* **تقسيم طبقات التخزين:** Raw رخيص، Curated للأعمال، أرشفة إلى Cold Storage.
* **تحسين الاستعلام:** Partitioning/Clustering + حدود تكلفة الاستعلام + تنبيهات.
* **TTL/Retention:** تنظيف تلقائي للجداول المؤقتة والـSandbox.

## **12) مؤشرات الأداء الرئيسية (KPIs)**

* **تشغيلية:** p95 ≤ 300ms، Error Rate ≤ 0.1%، Uptime ≥ 99.9%.
* **عملاء:** Retention D30 ≥ 25%، Churn ≤ 5%، CLV ≥ X.
* **مبيعات:** Conversion لكل مرحلة، ARPU ≥ X، Return Rate ≤ 2%.
* **تسويق:** ROAS ≥ 3x، CAC ≤ X، Experiment Uplift ≥ +10%.
* **جودة بيانات:** Data Quality Score ≥ 95%، PII Masking = 100%، Freshness ضمن SLA.

**13) خطة التنفيذ**

**أ. الأساسيات**

* تفعيل Schema Registry + Data Catalog + Lineage.
* تحديد Event Taxonomy + Data Contracts.
* إنشاء قنوات Streaming + جداول Raw.

**ب. النمذجة والجودة**

* بناء طبقات Staging/Curated بالدبتي/dbt.
* إنشاء قواعد DQ + لوحات جودة + تنبيهات.
* ربط الهوية (Identity Resolution) ومفاتيح المفاهيم.

**ج. المقاييس واللوحات**

* طبقة دلالية للمقاييس القياسية.
* لوحات تشغيلية/تجارية/إدارية وفق SLAs.
* إطلاق Reverse ETL لأهم الشرائح.

**د. ML/التكلفة/التدقيق**

* POC لتوصية أو Forecast.
* سياسات Retention/Masking + ضبط التكاليف + حدود الاستعلام.
* اختبارات قبول نهائية وتسليم Runbooks.

## **14) RACI (الأدوار والمسؤوليات)**

* **Data Engineering:** البنية، ETL/ELT، النمذجة، DQ، الكاتالوج.
* **Analytics/BI:** المقاييس، اللوحات، التقارير، التحليلات المتقدمة.
* **ML/DS:** النماذج، Feature Store، المراقبة.
* **Security/Compliance:** التصنيف، الوصول، الخصوصية، التدقيق.
* **Product/PM:** المتطلبات، الأولويات، قبول النتائج.
* **SRE/Infra:** المنصة، المراقبة، التكلفة، التوسّع.

## **15) معايير القبول (Definition of Done)**

* كل مصدر بيانات مغطّى بعقد (Contract) ومُوثّق.
* جودة البيانات ≥ 95% وFreshness ضمن SLA.
* المقاييس الأساسية متاحة في طبقة دلالية واحدة ومعتمدة.
* اللوحات الثلاث (تشغيلية/تجارية/إدارية) عاملة وتحدَّث في مواعيدها.
* Reverse ETL فعّال لشرائح حرجة (VIP/Churn).
* POC ML واحد على الأقل يعمل ببيانات حقيقية.
* سياسات Retention/Masking/Audit نافذة.

## **16) تشيك**

* Event Taxonomy + Contracts جاهزة.
* Streaming + Batch Ingestion تعمل.
* Raw/Staging/Curated/Sandbox مُفعّلة.
* DQ Rules + Dashboards + Alerts.
* Semantic Metrics Layer موثّقة.
* Dashboards (Ops/Business/Exec) وفق SLAs.
* Reverse ETL إلى CRM/Ads.
* Feature Store + POC ML.
* Retention/Masking/Access/Audit مطبّقة.
* Cost controls (Partition/Clustering/Alerts) مفعّلة.

## **17) المخاطر والضوابط**

* **فجوات في التعاريف:** طبقة دلالية مركزية + توثيق مقاييس.
* **جودة ضعيفة:** DQ إلزامي + عزل وإعادة معالجة.
* **تكلفة عالية:** Partition/Clustering + حدود وAlerts.
* **خصوصية:** Masking/Tokenization + وصول مقنن.
* **اعتمادية ضعيفة:** SLA/Retry/DLQ + مراقبة Freshness.
* **تشتت الملكية:** Data Owners محدّدون لكل منتج بيانات.

## **18) المخرجات المطلوبة (Artifacts)**

* **Data Contracts** لكل مصدر/حدث.
* **Data Catalog** مع Lineage ومالكي البيانات.
* **نماذج dbt** (Dimensions/Facts + Tests).
* **لوحات** (Ops/Business/Exec) وروابطها.
* **Runbooks** للتشغيل وإعادة المعالجة.
* **Policy Pack**: Retention/Masking/Access.
* **Cost Playbook** لتحسين الاستعلام والتخزين.
* **ML POC Doc**: البيانات، المقاييس، خطة المراقبة.