⚙️ Event-Driven Architecture (EDA) — Gamasa Properties (V1)

> الهدف من القسم ده: نخلي الـ **Booking / Payments / Availability / Notifications** ماشية بـ "Source of Truth" بتاعة النظام بدون ما نكسر (Produced/Consumed) واضحة **Events**: **Availability** **مشتقة من bookings فقط**.

---

1) مبادئ أساسية (Non-Negotiables)

1. **Availability** داخل property **لا تُخزّن**.
   بتتحدد وقت البحث/اختيار تواريخ عبر استبعاد أي bookings في حالات overlaps مع `confirmed/active` + جدول `property_unavailability`.
2. **State Machine** نتيجة = حالة للحجز: أي تغيير **هي الحقيقة** Event + Guard Rules.
3. **الأحداث لا تغيّر الحقيقة لوحدها**: الحدث "يعلن" إن شيء حصل، لكن **التحويل الفعلي** لازم يمر عبر قواعد الانتقال المسموح.

---

2) Why EDA هنا؟

- إشعارات فورية للمؤجر/المستأجر (Realtime) بدون coupling عالي.
- فصل الدفع عن الحجز (Payment proofs + verification).
- تشغيل Automations زي `auto_expire` و `auto_complete`.
- Audit/Tracing كامل لكل تغيير حالة/فلوس/جزاءات.

---

3) Domains & Event Topics

نقسّم الـ events حسب الدومين:

- **booking.***: دورة حياة الحجز (requested → … → completed)
- **payment.***: رفع إيصال/مراجعة/اعتماد/فشل/Refund
- **property.***: تغييرات إدارية فقط (approved/blocked/archived) + unavailability blocks
- **revenue.***: تحصيل عمولة بعد check-in فقط
- **notification.***: إرسال إشعار (push/in-app/realtime)
- **rating.***: تخفيض تقييم/إضافة Review بعد completion

ده متوافق مع الفلو والـ state machine الموجودة V1.

---

4) Event Contract (Schema موحّد)

كل event لازم يبقى له envelope ثابت:

- `event_id` (UUID) — **Idempotency key**
- `event_type` (مثال: `booking.requested`)
- `occurred_at` (ISO timestamp)
- `actor_type` (`tenant|landlord|admin|system`)
- `actor_id` (UUID nullable لو system)
- `entity_type` (`booking|payment|property|revenue|notification`)
- `entity_id` (UUID)
- `correlation_id` (UUID) — يربط سلسلة أحداث لنفس العملية
- `payload` (JSONB) — البيانات اللازمة فقط
- `version` (مثلاً 1)

> القاعدة: أي consumer لازم يشتغل **Idempotent**: لو نفس `event_id` اتكرر، مايحصلش side effects مرتين.

---

5) Producers & Consumers (مين بيطلّع ومين بيستهلك)

Producers (مصادر الأحداث)

- **Server Actions / API Routes** (Next.js) عند actions المستخدم
- **Supabase DB triggers** (للـ Outbox / Audit)
- **Edge Functions / Cron** لأحداث النظام (expire/complete)

Consumers (مستهلكين الأحداث)

- **Booking Service**: يطبّق transitions
- **Payment Service**: ينشئ payment records + يغير payment_status
- **Notification Service**: يرسل Realtime/Push/In-app
- **Revenue Service**: يسجل عمولة بعد `active` فقط
- **Admin/Moderation**: penalties + block property عند تكرار no-show

(كل ده مطابق لسيناريوهات V1: العمولة بعد check-in، والـ no-show، والـ expiry… إلخ)

---

6) Core Event Flows (V1)

A) Booking Request (Tenant → Landlord)

1. `booking.availability_checked` (اختياري للتتبع فقط)
2. `booking.verification_fee_paid` (50 جنيه)
3. `booking.requested` → إنشاء booking بحالة `requested`
4. `notification.sent` إلى landlord

(ده مطابق للفلو الأساسي).

B) Approve / Reject

- `booking.approved` (Landlord) → status = `approved`
- `booking.rejected` (Landlord) → status = `rejected`
    - side effect: `wallet.credit_added` (50 كـ credit) لو ماشيين بنظام رصيد

قواعد التحويل مسنودة بالـ state machine.

C) Payment

Cash on delivery

- `booking.payment_method_selected` (cash)
- `booking.confirmed` مباشرة

Electronic

- `booking.payment_pending`
- `payment.receipt_uploaded`
- `payment.verified` (landlord/admin) → `booking.confirmed`
- أو `payment.rejected` → يرجع لـ `payment_pending` أو يتحول `expired/cancelled` حسب policy

ده كله مطابق للـ lifecycle.

D) Check-in & Commission

- `booking.checkin_confirmed` (Landlord) → status = `active`
- ثم **حدث واحد واضح**:
    - `revenue.commission_collect_requested`
    - ينتج عنه:
        - `revenue.commission_collected` (10% من أول شهر من الطرفين)
        - `revenue.recorded`

مهم: مفيش تحصيل عمولة قبل `active`.

E) Auto Complete

- Cron/Edge Function يطلق:

- `booking.auto_completed` عند نهاية المدة → status = `completed`
- ثم `notification.sent` للطرفين

F) Expiry (No Payment)

- النظام يطلق:
  - `booking.auto_expired` بعد deadline → status = `expired`
- Side effects:
  - 50 (حسب V1) جنيه **غير مستردة**

G) Cancellation (Before active only)

- `booking.cancel_requested` (tenant أو landlord حسب الحالة)
- `booking.cancelled`
- Side effects (حسب policy):
  - قبل approval: 50 يرجع credit
  - بعد approval وقبل check-in: “Apply cancellation policy”
  - بعد `active`: **No cancellation** (خارج المنصة قانونياً)

H) Landlord No-Show

- `booking.landlord_no_show_reported`
- `booking.cancelled` (reason: landlord_no_show)
- Side effects:
  - `payment.refund_issued` (Refund tenant)
  - `revenue.penalty_applied` (خصم/جزاء)
  - `rating.landlord_decreased`
  - `property.blocked` لو تكررت (Admin)

---

7) التنفيذ المقترح داخل Stack المشروع (Next.js + Supabase)

A) Outbox Pattern (مهم لتجنب ضياع events)

- أي update (مثل تغيير مهم status) يتم في **Transaction**:
  1. update booking
  2. insert event في جدول `event_outbox`
- Worker/Edge Function تلتقط من outbox وتبعت Realtime/Notifications/Async jobs

كده نضمن “at-least-once delivery” مع idempotency.

B) Realtime

- استخدم Supabase Realtime Channels لبث:
  - `notification.sent`
  - `booking.status_changed`
  - `payment.status_changed`

C) Mock Mode

بما إن النظام عندك دلوقتي فيه `IS_MOCK_MODE = true`، الأحداث تشتغل "in-memory":

- نسجل event log في console / array
- ونشغّل نفس consumers (لكن بدون DB triggers)
  عشان لما نطفي mock mode، نفس الـ contracts تفضل ثابتة.

---

8) Ordering / Idempotency / Retries

- **Ordering**: مش هنفترض ترتيب مثالي على مستوى النظام كله.
  الضمان الحقيقي ييجي من **Guards + DB constraints** (مثلاً: ماينفعش confirmed → requested).
- **Idempotency**: أي consumer يحتفظ بـ `processed_events(event_id)` أو يستخدم unique constraint.
- **Retries**: exponential backoff للـ notifications / webhooks.
  لو فشل دائم: نحول event لـ `dead_letter` table.

---

9) Security & Permissions (Actors)

- Tenant ينتج: `booking.requested`, `payment.receipt_uploaded`, `booking.cancel_requested` (قبل active فقط)
- Landlord ينتج: `booking.approved/rejected`, `booking.checkin_confirmed`
- System ينتج: `booking.auto_expired`, `booking.auto_completed`
- Admin ينتج: `payment.verified/rejected` (اختياري)، `property.blocked`

"مين يغيّر الحالة؟" ده مطابق لجدول.

---

10) Observability (لازم لـ Production)

- **Event Log** (immutable) + **Audit Log** لتغييرات الحالة
- Correlation IDs لكل flow (من لحظة request لحد completion)
- Metrics:
  - time-to-approve
  - time-to-confirm

- payment verification latency
- cancellation rate قبل check-in
- no-show rate
