Verifica Entità Logiche

SmartMaintenance

Piattaforma SaaS multi-tenant per la gestione manutenzioni

AgriSolution S.R.L.

Generato automaticamente da BUSINESS_LOGIC.md

Piano di verifica — Entità logiche

Aggiornato: 2026-04-11 **Scopo:** verificare che ogni entità logica (transazionale, derivata, di processo) funzioni nei suoi flussi di stato + validazioni + side-effect.

Compagnia di PIANO_VERIFICA_ANAGRAFICHE.md e BUSINESS_LOGIC.md.

Cos'è un'entità logica

Un'entità è "logica" (transazionale / di processo) quando:

- **Ha uno stato** che evolve nel tempo (OPEN → IN_PROGRESS → CLOSED)
- **Riferisce** entità anagrafiche (un Ticket riferisce Customer, Asset, User)
- **Genera side-effect** (notifiche, audit log, mail, push, eventi WhatsApp)
- **Esiste come conseguenza** di un evento, non come stato permanente
- **Ha una scadenza** (SLA, due date, expiry)

Esempi: un ticket nasce da una segnalazione e muore quando viene chiuso; un work order è figlio di un ticket; una fattura è figlia di un work order chiuso; un alert IoT nasce da una soglia superata.

Inventario delle 24 entità logiche

Block 1 — Workflow operativo

#	Entità	Tabella	Stati possibili
1	**Ticket**	`tickets`	OPEN → ASSIGNED → IN_PROGRESS → RESOLVED → CLOSED → CANCELLED
2	**TicketComment**	`ticket_comments`	(immutable, append-only)
3	**TicketAttachment**	`ticket_attachments`	(immutable)
4	**TicketStatusHistory**	`ticket_status_history`	(immutable, audit)
5	**WorkOrder**	`work_orders`	DRAFT → SCHEDULED → ASSIGNED → IN_PROGRESS → COMPLETED → INVOICED → CLOSED
6	**WorkOrderMaterial**	`work_order_materials`	(parte di WO)
7	**WorkOrderPhoto**	`work_order_photos`	(immutable)

Block 2 — Manutenzione preventiva e contratti

#	Entità	Tabella	Stati
8	**MaintenancePlan**	`maintenance_plans`	ACTIVE / PAUSED / EXPIRED
9	**PdaContract**	`pda_contracts`	DRAFT → ACTIVE → EXPIRED → RENEWED
10	**PdaContractProduct**	`pda_contract_products`	(riga di contratto)
11	**Contract**	`contracts`	DRAFT → ACTIVE → SUSPENDED → EXPIRED

Block 3 — Magazzino e movimenti

#	Entità	Tabella	Stati
12	**WarehouseMovement**	`warehouse_movements`	(immutable, append)
13	**PurchaseOrder**	`purchase_orders`	DRAFT → SENT → CONFIRMED → PARTIAL → RECEIVED
14	**DdtDocument**	`ddt_documents`	DRAFT → SENT → DELIVERED

Block 4 — Fatturazione

#	Entità	Tabella	Stati
15	**Invoice**	`invoices`	DRAFT → ISSUED → SENT_SDI → ACCEPTED → PAID
16	**SdiInvoice**	`sdi_invoices`	(XML FatturaPA) PENDING → SENT → ACCEPTED → REJECTED
17	**FinancingRecord**	`financing_records`	ACTIVE → CLOSED

Block 5 — Quality / Compliance

#	Entità	Tabella	Stati
18	**HaccpCheck**	`haccp_checks`	OK / FAIL
19	**HaccpNonConformity**	`haccp_non_conformities`	OPEN → ACTION_TAKEN → CLOSED

Block 6 — IoT e AI

#	Entità	Tabella	Stati
20	**SensorReading**	`sensor_readings`	(immutable, telemetria)
21	**AiPrediction**	`ai_predictions`	PENDING → ACK → DISMISSED

Block 7 — Comunicazione

#	Entità	Tabella	Stati
22	**Notification**	`notifications`	UNREAD → READ → ARCHIVED
23	**UserInvitation**	`user_invitations`	PENDING → SENT → ACCEPTED → EXPIRED
24	**AuditLog**	`audit_log`	(immutable, append)

Checklist di verifica per ogni entità logica

Le entità logiche **NON** si verificano solo con CRUD (è il piano anagrafiche). Si verificano i **flussi di stato + side-effect + integrità referenziale**.

Block S — State machine

- [ ] **S.1** — Stato iniziale corretto al momento del create
- [ ] **S.2** — Transizioni valide funzionano (es. OPEN → ASSIGNED)
- [ ] **S.3** — Transizioni invalide rifiutate (es. CLOSED → OPEN senza riapertura esplicita)
- [ ] **S.4** — Storia degli stati tracciata in `*_status_history` o `audit_log`
- [ ] **S.5** — Stato finale (CLOSED/CANCELLED) protegge da modifiche successive (read-only)

Block X — Cross-entity links

- [ ] **X.1** — FK verso entità anagrafiche valide (es. ticket.customer_id punta a un Customer del **proprio tenant**)
- [ ] **X.2** — FK cross-tenant rifiutate (Ticket di tenant 2 NON può puntare a Customer di tenant 1)
- [ ] **X.3** — Cancellazione cascade controllata (cancellando Ticket vengono cancellati ticket_comments e ticket_attachments? auto/manuale?)
- [ ] **X.4** — Promozione tra entità funziona (Ticket → WorkOrder; WorkOrder → Invoice)

Block N — Notifiche / Side effect

- [ ] **N.1** — Notification creata su evento (es. nuovo Ticket → notifica admin)
- [ ] **N.2** — Email inviata se `notification_email` attivo
- [ ] **N.3** — WhatsApp inviato se `notification_whatsapp` attivo
- [ ] **N.4** — Push FCM inviato se `notification_push` attivo + token presente
- [ ] **N.5** — Audit log riga `action=...` per ogni cambio stato significativo

Block V — Validazione business

- [ ] **V.1** — Campi obbligatori validati (es. WO senza customer fallisce)
- [ ] **V.2** — Date coerenti (es. completed_at > started_at > created_at)
- [ ] **V.3** — Quantità sensate (qty > 0, prezzo >= 0)
- [ ] **V.4** — Stati incoerenti rifiutati (es. ticket CLOSED senza resolution_text)

- [ ] **V.5** — SLA / due date calcolato correttamente
- [ ] **V.6** — Lock ottimistico contro modifiche concorrenti

Block T — Tenant integrity

- [ ] **T.1** — Counter mensile/annuale per tenant rispettato (es. `ticket_number = TK2026-00042` univoco per tenant per anno)
- [ ] **T.2** — Statistiche corrette per tenant (`SELECT count(*) WHERE status='OPEN'` torna solo i ticket del proprio tenant)

Block W — Webhook / API esterne

Per le entità che nascono o vanno verso l'esterno:

- [ ] **W.1** — Webhook WhatsApp 2Chat → crea ticket correttamente (vedi WhatsAppWebhookControllerTest)
- [ ] **W.2** — IoT data → crea SensorReading + eventuale alarm + ticket automatico (richiede API key valida)
- [ ] **W.3** — SDI invoice → genera XML FatturaPA conforme + invio Aruba
- [ ] **W.4** — Mobile API `/api/v1/work-orders/my` → ritorna solo i WO assegnati all'utente loggato

---

Verifica per stato (state machine testing)

Per ogni entità con state machine, testa esplicitamente la **matrice di transizione**:

FROM \ TO	DRAFT	OPEN	ASSIGNED	IN_PROGRESS	RESOLVED	CLOSED	CANCELLED
(new)	■	■	■	■	■	■	■
DRAFT	—	■	■	■	■	■	■
OPEN	■	—	■	■	■	■	■
ASSIGNED	■	■	—	■	■	■	■
IN_PROGRESS	■	■	■	—	■	■	■
RESOLVED	■	■	■	■	—	■	■
CLOSED	■	■	■	■	■	—	■
CANCELLED	■	■	■	■	■	■	—

→ ogni cella ■ = test che verifica la transizione, ogni ■ = test che verifica il rifiuto.

L'esempio sopra è per `Ticket` ma va replicato per `WorkOrder`, `PdaContract`, `Invoice`, `SdiInvoice`, `HaccpNonConformity`.

Side-effect map (cosa scatena cosa)

Per essere sicuri di non perdere coperture:

Trigger	Side-effect attesi
Ticket created	+ Notification per ADMIN tenant; + audit_log; eventuale auto-assign
Ticket assigned	+ Notification al tecnico assegnato; + push FCM; + email
Ticket WhatsApp	source=WHATSAPP; ticket_number formato TK2026-NNNNN
WorkOrder completed	Asset.last_maintenance_at aggiornato; warehouse_movements per i materiali usati; eventuale generazione Invoice draft
WorkOrder photo uploaded	Storage filesystem; thumbnail generated
PdaContract activated	MaintenancePlan generato per ogni installation collegata
PdaContract expiring (cron)	Notification ADMIN 30 giorni prima
MaintenancePlan due (cron)	Genera WorkOrder DRAFT
WarehouseItem stock < reorder_point	Notification se `notify_stock_below_reorder = true` su tenant
PurchaseOrder received	warehouse_stock incrementato; warehouse_movements riga IN
Invoice issued	SdiInvoice generata; XML inviato a SDI via Aruba
SdiInvoice rejected	Notification ADMIN; status SDI invoice = REJECTED
HaccpCheck failed	HaccpNonConformity OPEN; Notification responsabile
HaccpCheck temperatura fuori soglia	Ticket auto-creato se sensore configurato per autoCreateTicket
SensorReading > soglia MAX	Notification ADMIN; eventuale Ticket auto se threshold.autoCreateTicket = true
AiPrediction generated	Notification + dashboard alert
UserInvitation accepted	User created + active; password set; notification welcome
Login success	audit_log LOGIN
Login failed	audit_log LOGIN_FAILED + counter rate limit

Strumenti

Test integration con `@SpringBootTest`

```
@SpringBootTest
@ActiveProfiles("test")
@Transactional
class TicketWorkflowTest {

    @Autowired TicketService ticketService;
    @Autowired WorkOrderService woService;
    @Autowired NotificationService notifService;

    @Test
    void ticketCreated_notifiesAdmins() {
        // given
        Ticket t = Ticket.builder()
            .title("test").description("test")
            .source(Ticket.Source.WEB)
            .priority(Ticket.Priority.HIGH)
            .build();
        t.setTenantId(2L);

        // when
        ticketService.createTicket(t);

        // then
        // verifica audit log
        // verifica notifications.count > 0 per admin
    }

    @Test
    void ticketResolved_canBeReopened() { ... }

    @Test
    void ticketClosed_isReadOnly() {
        // CLOSED → modifica priority deve fallire
    }

    // ...
}
```

Tests da scrivere (priorità)

Sprint	Entità	Test stimati
1	Ticket (state machine + webhook)	15
1	WorkOrder (state machine + materials)	12
2	PdaContract + MaintenancePlan + auto-WO	10
2	Invoice + SdiInvoice (mock SDI)	8
3	Notification multi-channel + UserInvitation	8
3	HaccpCheck + NonConformity	6
4	SensorReading + alarm + auto-ticket	6
4	WarehouseMovement + reorder alert	5

Totale ~70 test sulle 24 entità logiche.

Definition of done

Un'entità logica è OK quando:

1. Tutti i check S/X/N/V/T (e W se applicabile) passano
2. La sua state machine ha una matrice di test esplicita
3. I side-effect attesi sono tutti coperti (tabella sopra)
4. È documentata in BUSINESS_LOGIC.md con il suo posto nel flusso end-to-end
5. Esiste almeno 1 integration test per ciascun side-effect critico

---

Stato attuale (snapshot 2026-04-11)

Coverage attuale dei test sulle 24 entità logiche:

Entità	Test esistente
Ticket	■ TicketServiceTest (parziale) + WhatsAppWebhookControllerTest (post-fix)
WorkOrder	■ WorkOrderServiceTest (parziale)
PurchaseOrder	■ PurchaseOrderServiceTest
WarehouseMovement	■ WarehouseServiceTest
Tutte le altre 20	■

→ **20 entità logiche su 24 senza test**. Il piano qui sopra ne richiede ~70. Il bottleneck operativo principale è scrivere e mantenere questa suite, non la complessità della logica in sé.