Verifica Entità Anagrafiche

SmartMaintenance

Piattaforma SaaS multi-tenant per la gestione manutenzioni

AgriSolution S.R.L.

Generato automaticamente da BUSINESS_LOGIC.md

Piano di verifica — Entità anagrafiche

Aggiornato: 2026-04-11 **Scopo:** verificare che ogni entità anagrafica del sistema sia **creabile, leggibile, modificabile, cancellabile** (CRUD) e che rispetti l'isolamento multi-tenant.

Compagnia di PIANO_VERIFICA_LOGICHE.md e BUSINESS_LOGIC.md.

Cos'è un'entità anagrafica

In questo software un'entità è "anagrafica" quando rappresenta un **soggetto, oggetto fisico o catalogo** che esiste indipendentemente da una transazione o evento. Sopravvive all'arco di vita di tickets e work orders. È il **dato master** su cui si appoggiano le entità logiche (transazionali).

Esempi: un cliente esiste prima e dopo un intervento; una caldaia esiste prima e dopo il primo guasto; un articolo di magazzino esiste a prescindere dalle vendite.

Inventario delle 16 entità anagrafiche

#	Entità	Tabella	Modulo	Multi-tenant
1	**Tenant**	`tenants`	Core	■ (è il tenant stesso)
2	**User**	`users`	Core / Auth	■
3	**Role**	`roles`	Auth (lookup globale)	■
4	**Customer**	`customers`	CRM	■
5	**CustomerInstallation**	`customer_installations`	CRM	■
6	**Asset**	`assets`	Asset management	■
7	**AssetCategory**	`asset_categories`	Asset management	■
8	**Supplier**	`suppliers`	Supply chain	■
9	**Agent**	`agents`	CRM (agenti commerciali)	■
10	**WarehouseItem**	`warehouse_items`	Magazzino	■
11	**ProductFamily**	`product_families`	Magazzino (catalogo)	■
12	**SerializedProduct**	`serialized_products`	Tracciamento seriali	■
13	**PriceList**	`price_lists`	Listini	■
14	**PriceCoefficient**	`price_coefficients`	Listini (coefficienti)	■
15	**BomHeader**	`bom_headers`	Bill of Materials	■
16	**Sensor**	`sensors`	IoT	■

Checklist di verifica per ogni entità

Per ciascuna delle 16 entità verifica i seguenti **8 punti**. Se anche uno fallisce, l'entità è "non OK" e va sistemata prima di considerare quella anagrafica funzionante.

Block A — CRUD UI (web Thymeleaf)

- [ ] **A.1 List** — `GET /<entity>` ritorna 200, mostra paginazione, filtra solo i record del proprio tenant
- [ ] **A.2 New form** — `GET /<entity>/new` ritorna 200, mostra form vuoto
- [ ] **A.3 Save (create)** — `POST /<entity>/save` con dati validi crea il record, redirect 302, messaggio success
- [ ] **A.4 Save (validation error)** — POST con dati invalidi mostra messaggio errore, NON crea record
- [ ] **A.5 View detail** — `GET /<entity>/{id}` ritorna 200, mostra il record
- [ ] **A.6 Edit form** — `GET /<entity>/{id}/edit` ritorna 200 con dati pre-compilati
- [ ] **A.7 Update** — `POST /<entity>/save` su record esistente lo aggiorna
- [ ] **A.8 Delete (soft)** — `POST /<entity>/{id}/delete` disattiva il record

Block B — Multi-tenant isolation

- [ ] **B.1 Cross-tenant read denied** — utente del tenant B non vede record del tenant A nella lista
- [ ] **B.2 Cross-tenant detail denied** — utente del tenant B prova `GET /<entity>/{id_di_A}` → 404 o 403
- [ ] **B.3 Cross-tenant update denied** — POST update su record di altro tenant fallisce
- [ ] **B.4 SQL probe** — `SELECT count(*) FROM <table> GROUP BY tenant_id` ritorna conteggi corretti per ciascun tenant

Block C — REST API (per consumer mobile / integrazioni)

- [ ] **C.1 GET list** `/api/v1/<entity>` — 200 con auth, 401 senza
- [ ] **C.2 GET detail** `/api/v1/<entity>/{id}` — 200 con auth
- [ ] **C.3 POST create** — 201 con body valido (se l'entità è creabile via API)
- [ ] **C.4 PUT update** — 200 con body valido
- [ ] **C.5 DELETE** — 204 (o 200 se soft delete)

Block D — Audit + integrità

- [ ] **D.1 Audit create** — `audit_log` ha riga `action=CREATE` dopo creazione
- [ ] **D.2 Audit update** — `audit_log` ha riga `action=UPDATE` dopo modifica
- [ ] **D.3 Audit delete** — `audit_log` ha riga `action=DELETE` dopo cancellazione
- [ ] **D.4 FK integrity** — eventuali FK verso questa entità non lasciano orphan (es. cancellando un Customer i suoi installations devono essere riassegnati o cancellati in cascade)
- [ ] **D.5 Bean validation** — `@NotBlank`, `@Email`, `@Size` enforced lato service prima del salvataggio

Block E — Specifico per entità sensibili

Per **User** + **Customer** (PII):

- [ ] **E.1 Password policy enforced** (User) — password <8 char, senza numeri o senza lettere → rifiuto con messaggio

- [ ] **E.2 Email univoca** (User) — duplicato → errore 409
- [ ] **E.3 Right-to-erasure** (Customer) — `POST /customers/{id}/erase-gdpr` anonimizza correttamente

---

Piano di esecuzione

Fase 1 — Setup ambiente di test

1. Database di test pulito (H2 in-memory o PostgreSQL su porta dev)
2. Seed di 2 tenant + 2 utenti per tenant + dati minimi (1 customer, 1 asset, ecc. per ciascuno)
3. Sessione MockMvc o browser (Selenium / Playwright) loggata come tenant 2

Fase 2 — Esecuzione checklist (16 entità × 8 + 5 + 5 + 5 + 3 = ~430 check)

Tempi stimati con automazione MockMvc: ~6h. Tempi stimati manuali via browser: ~2 giorni.

Fase 3 — Output

Per ogni entità produrre una tabella tipo:

Check	Stato	Note
A.1 list	■	
A.2 new	■	
A.3 save	■	manca @Valid (vedi Fase 4 audit)
...		

E un riepilogo finale:

```
Entità OK:           N / 16
Entità con warning:  N / 16
Entità rotte:        N / 16
```

Strumenti consigliati

MockMvc + JUnit

Crea un test class per ogni entità (es. `CustomerCrudTest.java`) che esegue i check A/B/C/D/E in modo automatizzato. Riusabili come regression suite.

```
@SpringBootTest
@AutoConfigureMockMvc
@ActiveProfiles("test")
class CustomerCrudTest {

    @Autowired MockMvc mvc;

    @Test @WithUserDetails("test-tenant2@test.local")
    void list_returnsOnlyTenant2Records() throws Exception {
        mvc.perform(get("/customers"))
           .andExpect(status().isOk())
           .andExpect(model().attributeExists("customers"));
```

```
    }

    @Test @WithUserDetails("test-tenant2@test.local")
    void detail_crossTenantReturns404() throws Exception {
        // ID di un customer di tenant 1
        mvc.perform(get("/customers/{id}", customerId_tenant1))
           .andExpect(status().is4xxClientError());
    }

    // ... altri check
}
```

Probe SQL (read-only)

Per i check `B.4` esegui:

```
ssh root@93.186.254.111 "sudo -u postgres psql -d smartmaintenance -c \"
SELECT 'tenants' AS t, count(*) FROM tenants
UNION ALL SELECT 'users', count(*) FROM users GROUP BY tenant_id
UNION ALL SELECT 'customers', count(*) FROM customers GROUP BY tenant_id
UNION ALL SELECT 'assets', count(*) FROM assets GROUP BY tenant_id
-- ... per ogni entità
;\""
```

Strumento GUI: Postman / Bruno

Per i check REST `C.*` crea una collection con:

- variabile `{{baseUrl}}` (https://lean.fattorecreativo.it)
- variabile `{{token}}` (cookie o JWT futuro)
- una request per ogni endpoint × ogni entità

Definition of done

L'entità anagrafica è **OK** quando:

1. Tutti i check A/B/C/D passano (E solo per entità sensibili)
2. Esiste almeno 1 test automatizzato regression suite
3. È documentata in BUSINESS_LOGIC.md con il suo posto nel flusso end-to-end

---

Stato attuale (snapshot 2026-04-11)

In base agli audit Fasi 2/3/4/5:

Entità	Coverage CRUD	Test esistenti	Note
User	■■ parziale	■ 0	password policy ora enforced (post-fix), no test
Customer	■ CRUD ok	■ 0	duplicati massivi (Fase 3.3) — dedup script pronto
CustomerInstallation	■ CRUD ok	■ 0	
Asset	■ CRUD ok	■ 0	QR scanner mobile ha 2 TODO
AssetCategory	■ CRUD ok	■ 0	
Supplier	■ CRUD ok	■ SupplierServiceTest	unico con test esistente
Agent	■ CRUD ok	■ 0	
WarehouseItem	■ CRUD ok	■ WarehouseServiceTest	
ProductFamily	■ CRUD ok	■ 0	
SerializedProduct	■ CRUD ok	■ 0	19% seq scan (Fase 6.2)
PriceList	■ CRUD ok	■ 0	92% seq scan
PriceCoefficient	■ CRUD ok	■ 0	
BomHeader	■■ CRUD parziale	■ 0	99.6% seq scan, FK senza indice
Sensor	■ CRUD ok	■ 0	sensor_readings senza tenant_id (Fase 3.5)
Tenant	■■ admin only	■ 0	gestione via SuperAdmin only
Role	■■ admin only	■ 0	lookup statico, raramente toccato

→ Il **test coverage** è il bottleneck principale: l'audit Fase 5 stima ~5% e questo piano richiede di portarla almeno al 50% sulle 16 anagrafiche.

Priorità di intervento

Sprint	Entità da coprire prima
1	Customer, User (PII + GDPR)
2	Asset, Supplier, WarehouseItem
3	PdaContract _(entità logica, vedi piano logiche)_, CustomerInstallation
4	Sensor, BomHeader, PriceList, PriceCoefficient
5	Agent, ProductFamily, SerializedProduct
6	Tenant, Role, AssetCategory

Definition of done del piano stesso: tutti i 16 × 26 ≈ 430 check eseguiti e tracciati in `docs/qa/verifica-anagrafiche-RISULTATO.md` (da generare al completamento dell'esecuzione).