Analisi Sistema di Protocollo e Tracciabilità - MyGest

Data analisi: 21 Novembre 2025
Versione sistema: Main branch
Moduli analizzati: `protocollo/`, `documenti/`, `fascicoli/`, `archivio_fisico/`, `api/v1/protocollo/`

---

0. DISTINZIONE CONCETTUALE FONDAMENTALE

0.1 Protocollazione vs Tracciabilità

È cruciale distinguere due concetti spesso confusi nel sistema:

PROTOCOLLAZIONE

> Atto puntuale e definitivo che certifica l'esistenza di un documento/fascicolo

Caratteristiche:
- **Puntuale**: avviene **una sola volta** nella vita del documento
- **Certificativa**: attribuisce identità ufficiale e data certa
- **Immutabile**: congela i metadati principali (numero, data, mittente/destinatario)
- **Legalmente rilevante**: valore probatorio per fini amministrativi/legali

Elementi:
- Numero di protocollo progressivo annuale
- Data e ora di protocollazione
- Mittente (IN) / Destinatario (OUT)
- Oggetto/descrizione
- Classificazione (titolario)
- Ubicazione fisica iniziale

Direzioni:
- **IN**: Documenti/fascicoli **ricevuti** dall'esterno
- **OUT**: Documenti/fascicoli **prodotti e spediti** dall'organizzazione

Gestita da: `app protocollo/` → Modello `MovimentoProtocollo`

TRACCIABILITÀ

> Processo continuo che registra la vita e i movimenti del documento/fascicolo

Caratteristiche:
- **Continua**: dura per tutta la vita del documento (dalla creazione allo scarto)
- **Dinamica**: registra ogni passaggio, modifica, consultazione
- **Analitica**: traccia chi, cosa, quando, dove, perché
- **Operativa**: supporta gestione quotidiana e audit trail

Eventi tracciati:
- Passaggi di mano tra uffici/utenti (movimenti interni)
- Consultazioni e accessi
- Prestiti temporanei e restituzioni
- Spostamenti fisici tra ubicazioni
- Cambio stato (bozza → definitivo → archiviato → scaricato)
- Conservazione e scarto finale

Finalità:
- Ricostruire la storia completa dell'oggetto
- Garantire trasparenza e responsabilità (chi ha fatto cosa)
- Supportare ricerche e ritrovamento fisico
- Compliance normativa (es. privacy, conservazione documenti fiscali)

Gestita da:
- `app archivio_fisico/` → Modello `OperazioneArchivio` (solo documenti **cartacei**)
- ⚠ **MANCANTE**: Tracciabilità documenti **digitali** (da implementare)

0.2 Relazione tra Protocollazione e Tracciabilità

Punti chiave:
1. La **protocollazione** è **prerequisito** per la tracciabilità fisica
2. Un documento può essere protocollato **senza** essere tracciato (carte di lavoro)
3. Un documento **non può** essere tracciato senza essere protocollato
4. La protocollazione registra il **primo movimento** (IN o OUT)
5. La tracciabilità registra **tutti i movimenti successivi**

0.3 Implementazione Attuale nel Sistema

Aspetto	Protocollazione	Tracciabilità Cartacei	Tracciabilità Digitali
App Django	`protocollo/`	`archivio_fisico/`	**MANCANTE**
Modello principale	`MovimentoProtocollo`	`OperazioneArchivio`	-
Scope	Tutti documenti/fascicoli	Solo cartacei	-
Requisito	Campo `tracciabile=True`	+ Protocollo esistente	-
Frontend	React popup	Django templates	-
API REST	DRF ViewSet	Mancante	-
Stato	Funzionante*	Funzionante	Da implementare

*Con bug validazione univocità (vedi sezione 6.1)

---

1. ARCHITETTURA DEL SISTEMA

1.1 Struttura Moduli

Il sistema di protocollo è organizzato in:

```
protocollo/
├── models.py              # Modelli dati: MovimentoProtocollo, ProtocolloCounter
├── services.py            # Servizio wrapper protocolla()
├── forms.py               # Form Django per UI web
├── views.py               # Viste Django per protocollazione
├── admin.py               # Amministrazione Django
└── templates/             # Template HTML

api/v1/protocollo/
├── serializers.py         # Serializers REST API
├── views.py               # ViewSet DRF
└── urls.py                # Routes API

documenti/models.py        # Documento.tracciabile, out_aperto, metodi protocollo
fascicoli/models.py        # Fascicolo.ubicazione, metodi protocollo
```

1.2 Modello Dati

MovimentoProtocollo (modello centrale)

```
MovimentoProtocollo:
    - documento (FK nullable, XOR fascicolo)
    - fascicolo (FK nullable, XOR documento)
    - cliente (FK Anagrafica)
    - direzione (IN/OUT)
    - data (datetime)
    - anno (int, da data)
    - numero (int progressivo)
    - operatore (FK User)
    - destinatario (str) # "da chi" per IN, "a chi" per OUT
    - ubicazione (FK UnitaFisica)
    - chiuso (bool) # True quando OUT rientra
    - rientro_di (FK self) # IN che chiude un OUT
    - data_rientro_prevista (date)
    - causale (str)
    - note (text)
```

Vincoli DB:
- `unique_together` : (cliente, anno, direzione, numero)
- `check_constraint` : esattamente uno tra documento/fascicolo
- `check_constraint` : rientro_di solo per direzione IN
- `check_constraint` : numero > 0

ProtocolloCounter (contatore progressivi)

```
ProtocolloCounter:
    - cliente (FK Anagrafica)
    - anno (int)
    - direzione (IN/OUT)
    - last_number (int)

unique_together: (cliente, anno, direzione)
```

Gestione atomica con `F('last_number') + 1` e retry su `IntegrityError`.

---

2. FLUSSI OPERATIVI

2.1 Registrazione USCITA (OUT)

Precondizioni:
- Documento: `tracciabile = True`
- Nessun movimento protocollo esistente (validazione univocità)
- Nessuna uscita aperta (`out_aperto = False` per documenti)

Procedura:
1. Validazione target (documento XOR fascicolo)
2. Risoluzione ubicazione:
- **Documento digitale**: ubicazione = None (errore se fornita)
- **Documento cartaceo fascicolato**: ubicazione = ubicazione fascicolo protocollo o fascicolo.ubicazione
- **Documento cartaceo non fascicolato**: ubicazione obbligatoria
- **Fascicolo**: ubicazione obbligatoria se cartaceo (fascicolo.ubicazione_id presente)
3. Normalizzazione cliente (da documento/fascicolo)
4. Generazione progressivo atomico (cliente, anno, direzione)
5. Creazione movimento con `chiuso = False`
6. Per documenti: set `documento.out_aperto = True` (lock applicativo)
7. Per fascicoli: aggiornamento `fascicolo.ubicazione` se fornita

Vincoli:
- Solo un'uscita aperta per documento (gestito via `out_aperto`)
- Fascicoli: nessun vincolo OUT multipli aperti (da verificare se necessario)
- Ubicazione coerente con tipo (digitale/cartaceo)

Codice:

```
MovimentoProtocollo.registra_uscita(
    documento=doc,          # XOR fascicolo
    quando=datetime,
    operatore=user,
    a_chi="Nome Destinatario",
    data_rientro_prevista=date,
    ubicazione=unita_fisica,
    causale="...",
    note="..."
)
```

2.2 Registrazione ENTRATA (IN)

Precondizioni:
- Documento: `tracciabile = True`
- Nessun movimento protocollo esistente (validazione univocità)

Procedura:
1. Validazione target (documento XOR fascicolo)
2. Ricerca OUT aperto (più recente con `chiuso=False`)
3. Risoluzione ubicazione (stesse regole OUT)
4. Normalizzazione cliente
5. Generazione progressivo atomico
6. Creazione movimento IN
7. Se esiste OUT aperto:
- Set `out_aperto.chiuso = True`
- Link `movimento_in.rientro_di = out_aperto`
- Reset `documento.out_aperto = False`
8. Aggiornamento ubicazione per fascicoli

Vincoli:
- L'entrata può essere registrata anche senza uscita precedente (primo ingresso)
- Se esiste OUT aperto, lo chiude automaticamente
- Ubicazione obbligatoria per cartacei

Codice:

```
MovimentoProtocollo.registra_entrata(
    documento=doc,          # XOR fascicolo
    quando=datetime,
    operatore=user,
    da_chi="Nome Mittente",
    ubicazione=unita_fisica,
    causale="...",
    note="..."
)
```

---

3. LOGICHE DI UBICAZIONE

3.1 Documenti

Documenti Digitali (`documento.digitale = True`)

- **Ubicazione**: sempre `None`
- **Errore**: se ubicazione fornita → `ValidationError`

Documenti Cartacei NON Fascicolati

- **Ubicazione**: **OBBLIGATORIA**
- **Errore**: se ubicazione `None` → `ValidationError`

Documenti Cartacei Fascicolati

- **Ubicazione**: risoluzione automatica da fascicolo
- **Priorità**:
  1. MovimentoProtocollo del fascicolo (primo per data)
  2. `fascicolo.ubicazione`
  3. Se nessuna trovata → obbligatorio fornirla
- **Vincolo**: ubicazione documento deve coincidere con fascicolo
- **Errore**: se fornita diversa da fascicolo → `ValidationError`

Codice risoluzione:

```
def _resolve_documento_ubicazione(cls, documento, ubicazione):
    if documento.digitale:
        if ubicazione is not None:
            raise ValidationError("Documenti digitali non prevedono ubicazione")
        return None

    if documento.fascicolo_id:
        fascicolo_mov = cls._get_protocollo_fascicolo(documento.fascicolo)
        fascicolo_ubicazione = None

        if fascicolo_mov and fascicolo_mov.ubicazione:
            fascicolo_ubicazione = fascicolo_mov.ubicazione
        elif getattr(documento.fascicolo, "ubicazione", None):
            fascicolo_ubicazione = documento.fascicolo.ubicazione

        if fascicolo_ubicazione:
            if ubicazione and ubicazione != fascicolo_ubicazione:
                raise ValidationError("Ubicazione deve coincidere con fascicolo")
            return fascicolo_ubicazione

        if ubicazione is None:
            raise ValidationError("Ubicazione obbligatoria per doc cartaceo fascicolato senza protocol
        return ubicazione

    if ubicazione is None:
        raise ValidationError("Ubicazione obbligatoria per doc cartacei non fascicolati")
    return ubicazione
```

3.2 Fascicoli

Fascicoli Digitali (nessuna `fascicolo.ubicazione_id`)

- **Ubicazione**: sempre `None`
- **Logica**: se `ubicazione_id` vuoto → digitale

Fascicoli Cartacei (`fascicolo.ubicazione_id` presente)

- **Ubicazione**: **OBBLIGATORIA**
- **Aggiornamento**: `fascicolo.ubicazione` viene aggiornato con ubicazione del movimento

Codice risoluzione:

```
def _resolve_fascicolo_ubicazione(cls, fascicolo, ubicazione, *, direzione: str):
    ha_ubicazione = bool(fascicolo.ubicazione_id)

    if ha_ubicazione or ubicazione is not None:
        if ubicazione is None:
            raise ValidationError("Ubicazione obbligatoria per fascicoli cartacei")
        return ubicazione

    # fascicolo digitale
    return None
```

---

4. VINCOLI E VALIDAZIONI

4.1 Vincoli Modello Documento

```
Documento:
    tracciabile (bool) = True  # default
    out_aperto (bool) = False  # lock applicativo
    digitale (bool) = True     # default
```

Regole:
- Solo documenti `tracciabile=True` possono essere protocollati
- `out_aperto` gestisce lock per una sola uscita alla volta
- Campo `digitale` determina gestione ubicazione

4.2 Vincoli Modello Fascicolo

```
Fascicolo:
    ubicazione (FK UnitaFisica) nullable
```

Regole:
- `ubicazione_id` presente → fascicolo cartaceo
- `ubicazione_id` null → fascicolo digitale
- Ubicazione aggiornata durante protocollazione

4.3 Validazioni Applicative

Univocità Protocollo

```
def _ensure_univoco(cls, *, documento=None, fascicolo=None):
    if documento and cls.objects.filter(documento=documento).exists():
        raise ValidationError("Documento già protocollato")
    if fascicolo and cls.objects.filter(fascicolo=fascicolo).exists():
        raise ValidationError("Fascicolo già protocollato")
```

⚠ **PROBLEMA IDENTIFICATO**: Questa validazione impedisce movimenti multipli sullo stesso oggetto!

Lock Uscita Documenti

```
# Registrazione OUT
updated = DocumentoModel.objects.filter(
    pk=documento.pk, out_aperto=False
).update(out_aperto=True)

if updated == 0:
    raise ValidationError("Esiste già una uscita aperta")
```

CORRETTO: Impedisce uscite multiple contemporanee.

Normalizzazione Cliente

```
def _normalize_cliente(cls, cliente):
    # Gestisce SimpleLazyObject, Anagrafica diretta, Cliente con .anagrafica
    # Forza valutazione lazy e recupera Anagrafica
    # Solleva ValidationError se non trovata
```

ROBUSTO: Gestisce varie forme di riferimento cliente.

4.4 Generazione Progressivi

```
@transaction.atomic
def _next_progressivo(cliente, anno: int, direzione: str) -> int:
    for _ in range(5):  # retry
        try:
            counter, _ = ProtocolloCounter.objects.get_or_create(
                cliente=cliente, anno=anno, direzione=direzione,
                defaults={"last_number": 0}
            )
            ProtocolloCounter.objects.filter(pk=counter.pk).update(
                last_number=F("last_number") + 1
            )
            counter.refresh_from_db(fields=["last_number"])
            return counter.last_number
        except IntegrityError:
            continue
    raise RuntimeError("Impossibile generare progressivo")
```

ATOMICO: Usa `F()` expression e retry per race conditions.

---

5. INTERFACCE

5.1 API REST (DRF)

Endpoints:

```
POST /api/v1/protocollo/movimenti/protocolla-documento/{id}/
POST /api/v1/protocollo/movimenti/protocolla-fascicolo/{id}/
GET  /api/v1/protocollo/movimenti/
GET  /api/v1/protocollo/movimenti/{id}/
```

Input Serializer (`ProtocollazioneInputSerializer`):

```
{
    "direzione": "IN" | "OUT",  # required
    "quando": "2025-11-21T10:30:00Z",  # optional (default now)
    "da_chi": "...",     # required per IN
    "a_chi": "...",      # required per OUT
    "ubicazione_id": 123,  # optional
    "data_rientro_prevista": "2025-12-01",  # optional per OUT
    "causale": "...",
    "note": "..."
}
```

Validazione:

```
def validate(self, attrs):
    direzione = attrs.get('direzione')
    if direzione == 'IN' and not attrs.get('da_chi'):
        raise ValidationError({"da_chi": "Obbligatorio per entrate"})
    if direzione == 'OUT' and not attrs.get('a_chi'):
        raise ValidationError({"a_chi": "Obbligatorio per uscite"})
    return attrs
```

5.2 Web UI (Django Views)

Views:
- `protocolla_documento(request, pk)` → `documenti/protocollo.html`
- `protocolla_fascicolo(request, pk)` → `fascicoli/protocolla.html`

Form (`ProtocolloForm`):
- Form dinamico con campi condizionali
- Validazione ubicazione basata su tipo target
- Gestione disabilitazione campi pre-popolati

5.3 Frontend React

Componenti:
- `ProtocollazionePopupPage.tsx` (popup window standalone)
- `openProtocollazionePopup()` utility

Flusso:
1. Click pulsante “Protocolla” in detail page
2. Apertura popup window (650x800px) con form
3. Submit → POST API `/api/v1/protocollo/movimenti/protocolla-{tipo}/{id}/`
4. Success → postMessage parent + refresh + chiusura popup

---

6. PROBLEMI E CRITICITÀ IDENTIFICATE

6.1 CRITICITÀ ALTA: Validazione Univocità Errata

Problema:

```
def _ensure_univoco(cls, *, documento=None, fascicolo=None):
    if documento and cls.objects.filter(documento=documento).exists():
        raise ValidationError("Documento già protocollato")
```

Impatto:
- Impedisce registrare movimenti multipli (IN/OUT ciclici)
- Dopo prima protocollazione, impossibile registrare uscite/rientri successivi
- Sistema inutilizzabile per gestione movimenti reiterati

Causa: Validazione intesa per “primo protocollo” applicata a ogni movimento

Soluzione necessaria: Rimuovere o modificare questa validazione

6.2 CRITICITÀ MEDIA: Lock Uscite Fascicoli

Problema: Nessun lock `out_aperto` per fascicoli (solo documenti)

Impatto:
- Possibili uscite multiple contemporanee per fascicolo
- Ambiguità su quale OUT chiudere al rientro

Comportamento attuale:
- `registra_entrata()` cerca OUT più recente con `order_by("-data")`
- Chiude solo quello, lasciando altri aperti (se presenti)

Rischio: Stato inconsistente se uscite multiple non intenzionali

6.3 CRITICITÀ MEDIA: Ambiguità Fascicoli Digitali/ Cartacei

Problema: Distinzione basata su presenza `ubicazione_id`

Impatto:
- Non esiste campo `fascicolo.digitale` esplicito
- Fascicolo inizialmente digitale può diventare cartaceo post-protocollazione
- Logica implicita fonte di confusione

Miglioramento: Campo `digitale` booleano esplicito come per documenti

6.4 CRITICITÀ BASSA: Messaggio Destinatario Ambiguo

Problema: Campo `destinatario` usato per “da_chi” (IN) e “a_chi” (OUT)

Impatto:
- Naming confuso nel modello
- Necessaria logica condizionale per interpretare valore

Miglioramento: Campi separati `da_chi` e `a_chi` (nullable)

6.5 CRITICITÀ BASSA: Gestione OUT Multipli Chiusi

Problema: Solo ultimo OUT chiuso da IN

Scenario:
1. Fascicolo OUT a “Tizio” (id=1)
2. Fascicolo OUT a “Caio” (id=2) ← errore per documenti, ok per fascicoli
3. Fascicolo IN da “Caio” → chiude solo id=2

Impatto: OUT id=1 resta aperto indefinitamente

Soluzione attuale: Lock documenti impedisce scenario, fascicoli no

---

7. REGOLE DI BUSINESS ATTUALI

7.1 Documenti

Tipo	Tracciabile	Protocollo	Ubicazione	OUT Multipli
Digitale tracciabile			None	Lock
Cartaceo tracciabile			Obbl	Lock
Non tracciabile			N/A	N/A

Casi speciali:
- Carte di lavoro → `tracciabile=False` → esclusi da protocollo
- Documento in fascicolo → ubicazione ereditata
- Documento standalone → ubicazione obbligatoria

7.2 Fascicoli

Tipo	Ubicazione	Protocollo	OUT Multipli
Digitale	None		⚠ Nessun lock
Cartaceo	Obbligatoria		⚠ Nessun lock

Regole:
- Tutti i fascicoli sono protocollabili (nessun flag `tracciabile`)
- Ubicazione aggiornata durante protocollazione
- Nessun vincolo OUT unico

7.3 Progressivi Protocollo

Schema numerazione:

```
{CLIENTE}/{ANNO}/{DIREZIONE}/{NUMERO}

Esempio:
  Anagrafica: ROSSI001
  Anno: 2025
  Direzione: IN
  Numero: 00042
  → Protocollo: ROSSI001/2025/IN/00042
```

Caratteristiche:
- Progressivo per cliente/anno/direzione
- Numerazione indipendente IN/OUT
- Reset automatico ogni anno
- Nessun reset per cliente
- Formato display: `{anno}/{numero:06d} ({direzione})`

7.4 Ciclo Vita Movimento

```
OUT aperto → chiuso=False
    ↓
   IN registrato → chiuso OUT → chiuso=True
    ↓
IN.rientro_di → FK al OUT chiuso
```

Stati:
- **OUT aperto**: `direzione=OUT, chiuso=False`
- **OUT chiuso**: `direzione=OUT, chiuso=True`
- **IN** (rientro): `direzione=IN, rientro_di=<OUT_id>`
- **IN** (primo ingresso): `direzione=IN, rientro_di=NULL`

---

8. PROPOSTE DI MIGLIORAMENTO

8.1 PRIORITÀ ALTA: Rimozione Validazione Univocità

Obiettivo: Permettere movimenti multipli sullo stesso documento/fascicolo

Modifica:

```
# FILE: protocollo/models.py

# RIMUOVERE O MODIFICARE:
@classmethod
def _ensure_univoco(cls, *, documento=None, fascicolo=None):
    # OPZIONE A: Rimuovere completamente
    pass

    # OPZIONE B: Validare solo prima protocollazione
    if documento:
        # Verifica che non esista già un OUT aperto
        if cls.objects.filter(documento=documento, direzione='OUT', chiuso=False).exists():
            raise ValidationError("Esiste già un'uscita aperta per questo documento")

    if fascicolo:
        # Opzionale: stessa logica per fascicoli
        if cls.objects.filter(fascicolo=fascicolo, direzione='OUT', chiuso=False).exists():
            raise ValidationError("Esiste già un'uscita aperta per questo fascicolo")
```

Impatto:
- Permette cicli IN→OUT→IN→OUT illimitati
- Mantiene lock singola uscita (se opzione B)
- Storico completo movimenti

Rischi:
- ⚠ Rimuovere codice potrebbe lasciare comportamenti inattesi altrove

Test necessari:
- Registrare OUT su documento già con movimento IN
- Verificare storico movimenti multipli
- Testare chiusura OUT con IN successivi

8.2 PRIORITÀ ALTA: Lock Uscite Fascicoli

Obiettivo: Impedire uscite multiple contemporanee per fascicoli

Modifica:

```
# FILE: fascicoli/models.py

class Fascicolo(models.Model):
    # ... campi esistenti ...
    out_aperto = models.BooleanField(default=False, db_index=True)
```

```
# FILE: protocollo/models.py

@classmethod
@transaction.atomic
def registra_uscita(cls, *, documento=None, fascicolo=None, ...):
    # ... validazioni esistenti ...

    if fascicolo:
        # Aggiungi lock come per documenti
        from fascicoli.models import Fascicolo as FascicoloModel
        updated = FascicoloModel.objects.filter(
            pk=fascicolo.pk, out_aperto=False
        ).update(out_aperto=True)

        if updated == 0:
            raise ValidationError("Esiste già un'uscita aperta per questo fascicolo")

        try:
            # ... creazione movimento ...
        except Exception:
            FascicoloModel.objects.filter(
                pk=fascicolo.pk
            ).update(out_aperto=False)
            raise

@classmethod
@transaction.atomic
def registra_entrata(cls, *, documento=None, fascicolo=None, ...):
    # ... logica esistente ...

    if fascicolo and out_aperto:
        # Reset flag
        from fascicoli.models import Fascicolo as FascicoloModel
        FascicoloModel.objects.filter(pk=fascicolo.pk).update(out_aperto=False)
```

Migrazione:

```
# migrations/000X_add_fascicolo_out_aperto.py

from django.db import migrations, models

class Migration(migrations.Migration):
    dependencies = [
        ('fascicoli', '000X_previous'),
    ]

    operations = [
        migrations.AddField(
            model_name='fascicolo',
            name='out_aperto',
            field=models.BooleanField(default=False, db_index=True),
        ),
    ]
```

Impatto:
- Coerenza con comportamento documenti
- Previene ambiguità OUT multipli
- Logica uniforme tra documenti/fascicoli

8.3 PRIORITÀ MEDIA: Campo Fascicolo.digitale Esplicito

Obiettivo: Rendere esplicita natura digitale/cartacea fascicolo

Modifica:

```
# FILE: fascicoli/models.py

class Fascicolo(models.Model):
    # ... campi esistenti ...
    digitale = models.BooleanField(
        default=True,
        help_text="Se disattivo il fascicolo è cartaceo e richiede ubicazione fisica"
    )
    ubicazione = models.ForeignKey(
        UnitaFisica,
        on_delete=models.SET_NULL,
        null=True,
        blank=True,
        related_name="fascicoli",
    )

    class Meta:
        # ... existing ...
        constraints = [
            # Cartaceo → ubicazione obbligatoria
            models.CheckConstraint(
                check=(
                    models.Q(digitale=True, ubicazione__isnull=True) |
                    models.Q(digitale=False, ubicazione__isnull=False)
                ),
                name='fascicolo_ubicazione_coerente'
            )
        ]
```

```
# FILE: protocollo/models.py

@classmethod
def _resolve_fascicolo_ubicazione(cls, fascicolo, ubicazione, *, direzione: str):
    if fascicolo.digitale:
        if ubicazione is not None:
            raise ValidationError("Fascicoli digitali non prevedono ubicazione")
        return None

    # Fascicolo cartaceo
    if ubicazione is None:
        raise ValidationError("Ubicazione obbligatoria per fascicoli cartacei")
    return ubicazione
```

Migrazione dati:

```
# migrations/000X_add_fascicolo_digitale.py

def set_digitale_from_ubicazione(apps, schema_editor):
    Fascicolo = apps.get_model('fascicoli', 'Fascicolo')

    # Cartacei: hanno ubicazione
    Fascicolo.objects.filter(ubicazione__isnull=False).update(digitale=False)

    # Digitali: senza ubicazione
    Fascicolo.objects.filter(ubicazione__isnull=True).update(digitale=True)

class Migration(migrations.Migration):
    operations = [
        migrations.AddField(
            model_name='fascicolo',
            name='digitale',
            field=models.BooleanField(default=True),
        ),
        migrations.RunPython(set_digitale_from_ubicazione),
        migrations.AddConstraint(...),
    ]
```

Impatto:
- Chiarezza logica digitale/cartaceo
- Prevenzione errori configurazione
- Coerenza con modello Documento

8.4 PRIORITÀ MEDIA: Separazione Campi da_chi/a_chi

Obiettivo: Eliminare ambiguità campo `destinatario`

Modifica:

```
# FILE: protocollo/models.py

class MovimentoProtocollo(models.Model):
    # SOSTITUIRE:
    # destinatario = models.CharField(max_length=255, blank=True)

    # CON:
    da_chi = models.CharField(
        max_length=255,
        blank=True,
        help_text="Mittente (per entrate)"
    )
    a_chi = models.CharField(
        max_length=255,
        blank=True,
        help_text="Destinatario (per uscite)"
    )

    class Meta:
        # ... existing ...
        constraints = [
            # ... existing ...
            models.CheckConstraint(
                check=(
                    models.Q(direzione='IN', a_chi='') |
                    models.Q(direzione='OUT', da_chi='')
                ),
                name='direzione_destinatario_coerente'
            )
        ]
```

Migrazione dati:

```
def split_destinatario(apps, schema_editor):
    MovimentoProtocollo = apps.get_model('protocollo', 'MovimentoProtocollo')

    # IN: destinatario → da_chi
    MovimentoProtocollo.objects.filter(direzione='IN').update(
        da_chi=models.F('destinatario'),
        a_chi=''
    )

    # OUT: destinatario → a_chi
    MovimentoProtocollo.objects.filter(direzione='OUT').update(
        a_chi=models.F('destinatario'),
        da_chi=''
    )
```

Impatto:
- Chiarezza semantica
- Validazione constraint DB
- Codice più leggibile
- ⚠ Breaking change per API/form esistenti

8.5 PRIORITÀ BASSA: Metodi Property per Stato Movimento

Obiettivo: Facilitare query stato movimenti

Modifica:

```
# FILE: protocollo/models.py

class MovimentoProtocollo(models.Model):
    # ... campi esistenti ...

    @property
    def is_uscita_aperta(self) -> bool:
        """Verifica se è un'uscita aperta"""
        return self.direzione == 'OUT' and not self.chiuso

    @property
    def is_rientro(self) -> bool:
        """Verifica se è un rientro (IN che chiude OUT)"""
        return self.direzione == 'IN' and self.rientro_di_id is not None

    @property
    def is_primo_ingresso(self) -> bool:
        """Verifica se è il primo ingresso (IN senza OUT precedente)"""
        return self.direzione == 'IN' and self.rientro_di_id is None

    @property
    def giorni_fuori(self) -> Optional[int]:
        """Calcola giorni trascorsi dall'uscita (se OUT aperto)"""
        if not self.is_uscita_aperta:
            return None
        from django.utils import timezone
        delta = timezone.now() - self.data
        return delta.days

    @property
    def giorni_ritardo_rientro(self) -> Optional[int]:
        """Calcola giorni di ritardo rispetto a data_rientro_prevista"""
        if not self.is_uscita_aperta or not self.data_rientro_prevista:
            return None
        from django.utils import timezone
        oggi = timezone.now().date()
        if oggi <= self.data_rientro_prevista:
            return None
        delta = oggi - self.data_rientro_prevista
        return delta.days
```

Uso:

```
# Query uscite scadute
uscite_in_ritardo = MovimentoProtocollo.objects.filter(
    direzione='OUT',
    chiuso=False,
    data_rientro_prevista__lt=timezone.now().date()
)

for movimento in uscite_in_ritardo:
    print(f"{movimento} - Ritardo: {movimento.giorni_ritardo_rientro} giorni")
```

Impatto:
- Codice più espressivo
- Facilitato reporting ritardi
- Base per dashboard/alert

8.6 PRIORITÀ BASSA: Gestione Automatica Ubicazione Documenti

Obiettivo: Automatizzare risoluzione ubicazione documenti fascicolati

Modifica:

```
# FILE: documenti/models.py

class Documento(models.Model):
    # ... campi esistenti ...

    def get_ubicazione_effettiva(self) -> Optional['UnitaFisica']:
        """
        Restituisce l'ubicazione effettiva del documento:
        - Digitale: None
        - Cartaceo fascicolato: ubicazione fascicolo
        - Cartaceo standalone: None (da protocollare)
        """
        if self.digitale:
            return None

        if not self.fascicolo_id:
            return None

        # Priorità: protocollo fascicolo > campo fascicolo
        from protocollo.models import MovimentoProtocollo
        mov = MovimentoProtocollo.objects.filter(
            fascicolo=self.fascicolo
        ).order_by('data').first()

        if mov and mov.ubicazione:
            return mov.ubicazione

        return getattr(self.fascicolo, 'ubicazione', None)

    def richiede_ubicazione_protocollo(self) -> bool:
        """Verifica se serve ubicazione per protocollazione"""
        if self.digitale:
            return False

        if not self.fascicolo_id:
            return True  # Cartaceo standalone

        return self.get_ubicazione_effettiva() is None
```

Frontend:

```
# API Serializer
class DocumentoDetailSerializer(serializers.ModelSerializer):
    ubicazione_effettiva = serializers.SerializerMethodField()
    richiede_ubicazione = serializers.SerializerMethodField()

    def get_ubicazione_effettiva(self, obj):
        ub = obj.get_ubicazione_effettiva()
        return UbicazioneSerializer(ub).data if ub else None

    def get_richiede_ubicazione(self, obj):
        return obj.richiede_ubicazione_protocollo()
```

Impatto:
- UX migliore: form pre-popolato o campo disabilitato
- Meno errori utente
- Logica centralizzata

---

9. SISTEMA DI TRACCIABILITÀ

9.1 Architettura Attuale (Archivio Fisico)

Il sistema di tracciabilità per documenti cartacei è implementato nell'app `archivio_fisico/`:

```
archivio_fisico/
├── models.py
│   ├── OperazioneArchivio      # Operazione di movimentazione
│   ├── RigaOperazioneArchivio  # Dettaglio oggetti movimentati
│   ├── UnitaFisica             # Struttura gerarchica ubicazioni
│   └── CollocazioneFisica      # Collocazione corrente oggetto
├── services.py                  # Logica process_operazione_archivio()
├── views.py                     # Viste Django (non REST)
└── templates/                   # Template HTML + generazione verbali
```

Modello OperazioneArchivio

```
OperazioneArchivio:
    - tipo_operazione: "entrata" | "uscita" | "interna"
    - data_ora: datetime (auto_now_add)
    - referente_interno: FK User
    - referente_esterno: FK Anagrafica (opzionale)
    - note: text
    - verbale_scan: File PDF/IMG (facoltativo)

Relations:
    - righe: [RigaOperazioneArchivio]  # dettaglio movimenti
```

Tipi operazione:
1. **Entrata**: Ricezione documento/fascicolo in archivio
2. **Uscita**: Rimozione per scarto, prestito esterno
3. **Interna**: Spostamento tra ubicazioni interne

Modello RigaOperazioneArchivio

```
RigaOperazioneArchivio:
    - operazione: FK OperazioneArchivio
    - documento: FK Documento (nullable, XOR fascicolo)
    - fascicolo: FK Fascicolo (nullable, XOR documento)
    - movimento_protocollo: FK MovimentoProtocollo (opzionale)

    # Ubicazioni
    - unita_fisica_sorgente: FK UnitaFisica
    - unita_fisica_destinazione: FK UnitaFisica

    # Stati
    - stato_precedente: CharField
    - stato_successivo: CharField

    - note: CharField
```

Vincoli:
- Solo documenti **cartacei** (`documento.digitale=False`)
- Solo documenti **tracciabili** (`documento.tracciabile=True`)
- Deve esistere protocollo (`documento.movimenti.exists()`)
- Fascicoli devono avere ubicazione fisica

Modello UnitaFisica

Struttura gerarchica per definire ubicazioni fisiche:

```
UnitaFisica:
    - tipo: UFFICIO | STANZA | SCAFFALE | MOBILE | ANTA | RIPIANO | CONTENITORE | CARTELLINA
    - parent: FK self (nullable)
    - codice: Univoco (auto-generato)
    - nome: str
    - full_path: str (es. "UFF/U01/ST/01/SF/A/RIP/003")
    - attivo: bool
    - archivio_fisso: bool  # Flag per archivio definitivo
```

Gerarchia ammessa:

```
UFFICIO
  └─ STANZA
      ├─ SCAFFALE → RIPIANO → CONTENITORE → CARTELLINA
      └─ MOBILE
          └─ ANTA → RIPIANO → CONTENITORE → CARTELLINA
```

9.2 Flusso Operativo Tracciabilità

Registrazione Operazione

Procedura (in `services.process_operazione_archivio()`):

1. **Validazione righe**:
   - Ogni riga deve avere documento XOR fascicolo
   - Documenti: solo cartacei (`digitale=False`) e tracciabili
   - Fascicoli: devono avere ubicazione fisica
   - Deve esistere movimento protocollo collegato

2. **Risoluzione movimento protocollo**:
   - Se non fornito, cerca primo movimento per data
   - Solleva errore se oggetto non protocollato

3. **Elaborazione per tipo operazione**:

ENTRATA:
`python - Risolve unita_destinazione (riga o protocollo.ubicazione) - Valida stato_successivo obbligatorio - Aggiorna obj.stato - Salva riga con stati risolti`

USCITA:
`python - Risolve unita_sorgente (riga o protocollo.ubicazione) - Risolve unita_destinazione (riga o unita_scarico) - Imposta stato_successivo (default: SCARICATO) - Aggiorna obj.stato`

INTERNA:
`python - Risolve unita_sorgente (riga o protocollo.ubicazione) - Richiede unita_destinazione obbligatoria - Aggiorna fascicolo.ubicazione se fascicolo - Aggiorna protocollo.ubicazione - Mantiene o aggiorna stato`

1. **Persistenza**:
   - Salva stati precedente/successivo su riga
   - Aggiorna stato oggetto (documento/fascicolo)
   - Aggiorna ubicazioni (fascicoli e protocollo)

Generazione Verbali

Il sistema genera verbali Word/PDF per le operazioni:

Template disponibili:
- `VERBALE_CONSEGNA_ARCHIVIO.docx`
- `VERBALE_RITIRO_ARCHIVIO.docx`
- `VERBALE_MOVIMENTAZIONE_INTERNA.docx`

Placeholder supportati:

```
{DATA_OPERAZIONE}           # Data operazione
{TIPO_OPERAZIONE}           # Entrata/Uscita/Interna
{REFERENTE_INTERNO}         # Utente interno
{REFERENTE_ESTERNO}         # Anagrafica esterna (se presente)
{ELENCO_DOCUMENTI}          # Tabella righe
{NOTE_GENERALI}             # Note operazione
{FIRMA_REFERENTE_INTERNO}   # Placeholder firma
{FIRMA_REFERENTE_ESTERNO}   # Placeholder firma
```

9.3 Punti di Forza Sistema Tracciabilità

Architettura Solida

1. **Separazione responsabilità**: App dedicata distinta da protocollo
2. **Modellazione gerarchica**: UnitaFisica con full_path automatico
3. **Audit trail completo**: Ogni operazione tracciata con date/utenti
4. **Verbali formali**: Generazione automatica documenti certificativi

5. **Validazioni robuste**: Vincoli su cartacei/tracciabili/protocollati

Flessibilità Operativa

1. **Operazioni batch**: Movimentazione multipla con singola operazione
2. **Gestione stati**: Tracciamento automatico transizioni stato
3. **Collegamenti protocollo**: Link diretto a MovimentoProtocollo
4. **Ubicazioni dinamiche**: Aggiornamento automatico su movimenti interni
5. **Referenti esterni**: Supporto prestiti/consultazioni terze parti

Completezza Funzionale

1. **Tutti i tipi movimento**: Entrata, uscita, interna
2. **Gestione scarichi**: Unità dedicata per dismissioni
3. **Archivio fisso**: Flag per ubicazioni permanenti
4. **Storage verbali**: Upload scansioni verbali firmati
5. **Storico completo**: Tutte le operazioni conservate

9.4 Criticità Sistema Tracciabilità

CRITICITÀ ALTA: Mancanza Tracciabilità Digitale

Problema:
- Sistema traccia solo documenti **cartacei**
- Documenti digitali esclusi da OperazioneArchivio
- Nessun modello alternativo per tracciabilità digitale

Impatto:
- Impossibile tracciare consultazioni documenti digitali
- Nessuno storico accessi/download
- Perdita audit trail per documenti non cartacei
- Compliance GDPR/privacy compromessa

Scenario reale:

```
# Documento digitale
doc = Documento(digitale=True, tracciabile=True, ...)
doc.save()

# Protocollazione:   OK
MovimentoProtocollo.registra_entrata(documento=doc, ...)

# Tracciabilità:   IMPOSSIBILE
OperazioneArchivio.clean() → ValidationError:
    "I documenti digitali non possono essere movimentati in archivio fisico"
```

Conseguenze:
- Non si sa chi ha consultato il documento
- Non si sa quando è stato scaricato
- Non si sa se è stato modificato
- Non si sa se è stato condiviso

CRITICITÀ MEDIA: Interfaccia Solo Django Templates

Problema:
- Nessuna API REST per `OperazioneArchivio`
- Viste Django tradizionali (no SPA)
- Frontend React non integrato

Impatto:
- UX inconsistente (protocollo in React, tracciabilità in Django)
- Impossibile usare tracciabilità da app mobile
- Difficile integrazione con sistemi esterni
- Duplicazione logica form validation

CRITICITÀ MEDIA: Dipendenza da Protocollo Esistente

Problema:
- Validazione richiede `movimento_protocollo` esistente
- Se non fornito, cerca automaticamente il primo per data
- Assunzione: oggetto sempre protocollato prima di tracciatura

Impatto:
- Impossibile tracciare oggetti non protocollati
- Limite per documenti interni/bozze
- Accoppiamento forte tra moduli

Codice:

```
# RigaOperazioneArchivio.clean()
if self.documento and not self.documento.movimenti.exists():
    raise ValidationError("Il documento non risulta protocollato.")
if self.fascicolo and not self.fascicolo.movimenti_protocollo.exists():
    raise ValidationError("Il fascicolo non risulta protocollato.")
```

CRITICITÀ BASSA: Complessità UnitaFisica

Problema:
- Gerarchia rigida con regole tipo_figli_ammessi
- Generazione automatica codici/full_path
- Progressivi manuali vs automatici

Impatto:
- Curva apprendimento alta per utenti
- Errori configurazione struttura archivio
- Manutenzione gravosa per riorganizzazioni

CRITICITÀ BASSA: Nessun Lock Concorrenza

Problema:
- Nessun lock su oggetti in movimento
- Possibili operazioni sovrapposte sullo stesso oggetto

Scenario:

```
Utente A: Registra uscita documento D
Utente B: Registra movimento interno documento D (contemporaneo)
→ Entrambe le operazioni salvate
→ Stato finale ambiguo
```

Nota: Bassa priorità per uso tipico single-user/low-concurrency

9.5 Lacune Funzionali

1. Nessuna Tracciabilità Consultazioni

Mancante:
- Registrazione accessi/visualizzazioni
- Log download documenti
- Storico modifiche metadati

Uso previsto:

```
# Scenario: Utente apre documento in detail page
ConsultazioneDocumento.objects.create(
    documento=doc,
    utente=request.user,
    data_ora=now(),
    tipo='visualizzazione',
    ip_address=get_client_ip(request)
)
```

2. Nessuna Gestione Prestiti Temporanei

Mancante:
- Prestiti con data restituzione prevista
- Alert scadenze rientro
- Solleciti automatici

Uso previsto:

```
PrestitoDocumento.objects.create(
    documento=doc,
    destinatario=anagrafica,
    data_prestito=now(),
    data_rientro_prevista=now() + timedelta(days=30),
    causale="Verifica contabile"
)
```

3. Nessuna Reportistica Analitica

Mancante:
- Report movimenti per periodo
- Statistiche utilizzo archivio
- Indicatori prestazioni (tempi medi, rotazione)

4. Nessuna Integrazione Notifiche

Mancante:
- Email notifica operazioni
- Alert scadenze rientro
- Dashboard movimenti real-time

9.6 Proposte Miglioramento Tracciabilità

PRIORITÀ P0: Tracciabilità Documenti Digitali

Obiettivo: Estendere tracciabilità a tutti i documenti, non solo cartacei

Soluzione: Nuovo modello `EventoTracciabilita` generale

```
# FILE: tracciabilita/models.py (nuova app)

class EventoTracciabilita(models.Model):
    """
    Evento di tracciabilità generico per documenti e fascicoli
    (cartacei e digitali)
    """

    class TipoEvento(models.TextChoices):
        # Eventi fisici (cartacei)
        MOVIMENTO_ENTRATA = 'mov_entrata', 'Movimento Entrata'
        MOVIMENTO_USCITA = 'mov_uscita', 'Movimento Uscita'
        MOVIMENTO_INTERNO = 'mov_interno', 'Movimento Interno'

        # Eventi digitali
        CONSULTAZIONE = 'consultazione', 'Consultazione'
        DOWNLOAD = 'download', 'Download'
        MODIFICA = 'modifica', 'Modifica'
        CONDIVISIONE = 'condivisione', 'Condivisione'

        # Eventi comuni
        CAMBIO_STATO = 'cambio_stato', 'Cambio Stato'
        PRESTITO = 'prestito', 'Prestito'
        RESTITUZIONE = 'restituzione', 'Restituzione'
        ELIMINAZIONE = 'eliminazione', 'Eliminazione'

    # Target
    documento = models.ForeignKey(
        'documenti.Documento',
        on_delete=models.CASCADE,
        null=True, blank=True,
        related_name='eventi_tracciabilita'
    )
    fascicolo = models.ForeignKey(
        'fascicoli.Fascicolo',
        on_delete=models.CASCADE,
        null=True, blank=True,
        related_name='eventi_tracciabilita'
    )

    # Evento
    tipo_evento = models.CharField(max_length=20, choices=TipoEvento.choices)
    data_ora = models.DateTimeField(auto_now_add=True, db_index=True)
```

```
    utente = models.ForeignKey(
        settings.AUTH_USER_MODEL,
        on_delete=models.SET_NULL,
        null=True,
        related_name='eventi_tracciabilita'
    )

    # Dettagli contestuali
    ubicazione_precedente = models.ForeignKey(
        'archivio_fisico.UnitaFisica',
        on_delete=models.SET_NULL,
        null=True, blank=True,
        related_name='eventi_sorgente'
    )
    ubicazione_successiva = models.ForeignKey(
        'archivio_fisico.UnitaFisica',
        on_delete=models.SET_NULL,
        null=True, blank=True,
        related_name='eventi_destinazione'
    )

    stato_precedente = models.CharField(max_length=30, blank=True)
    stato_successivo = models.CharField(max_length=30, blank=True)

    # Metadati
    referente_esterno = models.ForeignKey(
        'anagrafiche.Anagrafica',
        on_delete=models.SET_NULL,
        null=True, blank=True
    )
    ip_address = models.GenericIPAddressField(null=True, blank=True)
    user_agent = models.CharField(max_length=255, blank=True)

    note = models.TextField(blank=True)
    metadata_json = models.JSONField(default=dict, blank=True)

    # Link operazione archivio fisico (retrocompatibilità)
    operazione_archivio = models.ForeignKey(
        'archivio_fisico.OperazioneArchivio',
        on_delete=models.SET_NULL,
        null=True, blank=True,
        related_name='eventi_generati'
    )
```

```
    class Meta:
        verbose_name = 'Evento Tracciabilità'
        verbose_name_plural = 'Eventi Tracciabilità'
        ordering = ['-data_ora']
        indexes = [
            models.Index(fields=['documento', '-data_ora']),
            models.Index(fields=['fascicolo', '-data_ora']),
            models.Index(fields=['tipo_evento', '-data_ora']),
            models.Index(fields=['utente', '-data_ora']),
        ]
        constraints = [
            models.CheckConstraint(
                check=(
                    models.Q(documento__isnull=False, fascicolo__isnull=True) |
                    models.Q(documento__isnull=True, fascicolo__isnull=False)
                ),
                name='evento_target_xor'
            )
        ]

    def __str__(self):
        target = self.documento or self.fascicolo
        return f"{self.get_tipo_evento_display()} - {target} - {self.data_ora:%Y-%m-%d %H:%M}"
```

Middleware per tracciamento automatico:

```
# FILE: tracciabilita/middleware.py

class TracciabilitaMiddleware:
    """
    Middleware per tracciare automaticamente accessi a documenti/fascicoli
    """

    def __init__(self, get_response):
        self.get_response = get_response

    def __call__(self, request):
        response = self.get_response(request)

        # Traccia solo se autenticato
        if request.user.is_authenticated:
            self._traccia_accesso(request, response)

        return response

    def _traccia_accesso(self, request, response):
        # Individua documenti/fascicoli visualizzati
        if 'documenti/' in request.path and response.status_code == 200:
            doc_id = self._extract_id(request.path, 'documenti')
            if doc_id:
                EventoTracciabilita.objects.create(
                    documento_id=doc_id,
                    tipo_evento=EventoTracciabilita.TipoEvento.CONSULTAZIONE,
                    utente=request.user,
                    ip_address=self._get_client_ip(request),
                    user_agent=request.META.get('HTTP_USER_AGENT', '')[:255]
                )
```

Impatto:
- Tracciabilità unificata cartacei/digitali
- Audit trail completo accessi
- Compliance GDPR/privacy
- Retrocompatibilità con OperazioneArchivio

Migrazione dati:

```
# Genera EventoTracciabilita da OperazioneArchivio esistenti
for op in OperazioneArchivio.objects.all():
    for riga in op.righe.all():
        EventoTracciabilita.objects.create(
            documento=riga.documento,
            fascicolo=riga.fascicolo,
            tipo_evento=map_tipo_operazione(op.tipo_operazione),
            data_ora=op.data_ora,
            utente=op.referente_interno,
            ubicazione_precedente=riga.unita_fisica_sorgente,
            ubicazione_successiva=riga.unita_fisica_destinazione,
            stato_precedente=riga.stato_precedente,
            stato_successivo=riga.stato_successivo,
            referente_esterno=op.referente_esterno,
            note=riga.note or op.note,
            operazione_archivio=op
        )
```

PRIORITÀ P1: API REST Tracciabilità

Obiettivo: Esporre tracciabilità via API REST per frontend React

Implementazione:

```
# FILE: api/v1/tracciabilita/serializers.py

class EventoTracciabilitaSerializer(serializers.ModelSerializer):
    tipo_evento_display = serializers.CharField(
        source='get_tipo_evento_display',
        read_only=True
    )
    utente_display = serializers.SerializerMethodField()
    target_display = serializers.SerializerMethodField()

    class Meta:
        model = EventoTracciabilita
        fields = [
            'id', 'tipo_evento', 'tipo_evento_display',
            'data_ora', 'utente', 'utente_display',
            'documento', 'fascicolo', 'target_display',
            'ubicazione_precedente', 'ubicazione_successiva',
            'stato_precedente', 'stato_successivo',
            'referente_esterno', 'note', 'metadata_json'
        ]

    def get_utente_display(self, obj):
        if obj.utente:
            return f"{obj.utente.get_full_name()} ({obj.utente.username})"
        return None

    def get_target_display(self, obj):
        if obj.documento:
            return f"Documento: {obj.documento.codice}"
        if obj.fascicolo:
            return f"Fascicolo: {obj.fascicolo.codice}"
        return None


class RegistraEventoInputSerializer(serializers.Serializer):
    tipo_evento = serializers.ChoiceField(
        choices=EventoTracciabilita.TipoEvento.choices
    )
    note = serializers.CharField(required=False, allow_blank=True)
    metadata = serializers.JSONField(required=False)


# FILE: api/v1/tracciabilita/views.py
```

```
class EventoTracciabilitaViewSet(viewsets.ReadOnlyModelViewSet):
    """
    API per consultazione storico tracciabilità
    """
    serializer_class = EventoTracciabilitaSerializer
    permission_classes = [permissions.IsAuthenticated]
    filter_backends = [DjangoFilterBackend, OrderingFilter]
    filterset_fields = ['tipo_evento', 'documento', 'fascicolo', 'utente']
    ordering_fields = ['data_ora']
    ordering = ['-data_ora']

    def get_queryset(self):
        return EventoTracciabilita.objects.select_related(
            'documento', 'fascicolo', 'utente',
            'ubicazione_precedente', 'ubicazione_successiva',
            'referente_esterno'
        )

    @action(detail=False, methods=['post'], url_path='registra-documento/(?P<doc_id>[^/.]+)')
    def registra_evento_documento(self, request, doc_id=None):
        """
        Registra evento tracciabilità per documento
        POST /api/v1/tracciabilita/eventi/registra-documento/{id}/
        """
        documento = get_object_or_404(Documento, pk=doc_id)

        serializer = RegistraEventoInputSerializer(data=request.data)
        serializer.is_valid(raise_exception=True)

        evento = EventoTracciabilita.objects.create(
            documento=documento,
            tipo_evento=serializer.validated_data['tipo_evento'],
            utente=request.user,
            note=serializer.validated_data.get('note', ''),
            metadata_json=serializer.validated_data.get('metadata', {}),
            ip_address=self._get_client_ip(request),
            user_agent=request.META.get('HTTP_USER_AGENT', '')[:255]
        )

        return Response(
            EventoTracciabilitaSerializer(evento).data,
            status=status.HTTP_201_CREATED
```

```
        )

    @action(detail=False, methods=['post'], url_path='registra-fascicolo/(?P<fasc_id>[^/.]+)')
    def registra_evento_fascicolo(self, request, fasc_id=None):
        """
        Registra evento tracciabilità per fascicolo
        POST /api/v1/tracciabilita/eventi/registra-fascicolo/{id}/
        """
        # Implementazione simile a registra_evento_documento
        ...
```

Frontend React:

```
// FILE: frontend/src/components/TracciabilitaTimeline.tsx

interface TracciabilitaTimelineProps {
  documentoId?: number;
  fascicoloId?: number;
}

export const TracciabilitaTimeline: React.FC<TracciabilitaTimelineProps> = ({
  documentoId,
  fascicoloId,
}) => {
  const [eventi, setEventi] = useState<EventoTracciabilita[]>([]);
  const [loading, setLoading] = useState(true);

  useEffect(() => {
    const fetchEventi = async () => {
      const params = new URLSearchParams();
      if (documentoId) params.append('documento', documentoId.toString());
      if (fascicoloId) params.append('fascicolo', fascicoloId.toString());

      const response = await fetch(`/api/v1/tracciabilita/eventi/?${params}`);
      const data = await response.json();
      setEventi(data.results);
      setLoading(false);
    };

    fetchEventi();
  }, [documentoId, fascicoloId]);

  if (loading) return <div>Caricamento...</div>;

  return (
    <div className="tracciabilita-timeline">
      <h3>Storico Tracciabilità</h3>
      {eventi.map(evento => (
        <div key={evento.id} className="evento-item">
          <div className="evento-header">
            <span className={`badge badge-${getTipoBadgeClass(evento.tipo_evento)}`}>
              {evento.tipo_evento_display}
            </span>
            <span className="evento-data">
              {formatDateTime(evento.data_ora)}
            </span>
```

```
          </div>
          <div className="evento-body">
            <p><strong>Utente:</strong> {evento.utente_display}</p>
            {evento.ubicazione_precedente && (
              <p><strong>Da:</strong> {evento.ubicazione_precedente.full_path}</p>
            )}
            {evento.ubicazione_successiva && (
              <p><strong>A:</strong> {evento.ubicazione_successiva.full_path}</p>
            )}
            {evento.note && <p><strong>Note:</strong> {evento.note}</p>}
          </div>
        </div>
      ))}
    </div>
  );
};
```

Impatto:
- Frontend React completo
- UX coerente con protocollo
- Real-time updates possibili (WebSocket futuro)
- Mobile-ready

PRIORITÀ P2: Gestione Prestiti con Scadenze

Obiettivo: Tracciare prestiti temporanei con alert rientro

```
# FILE: tracciabilita/models.py (aggiunta)

class PrestitoDocumento(models.Model):
    """
    Prestito temporaneo di documento/fascicolo
    """
    documento = models.ForeignKey(
        'documenti.Documento',
        on_delete=models.CASCADE,
        null=True, blank=True,
        related_name='prestiti'
    )
    fascicolo = models.ForeignKey(
        'fascicoli.Fascicolo',
        on_delete=models.CASCADE,
        null=True, blank=True,
        related_name='prestiti'
    )

    destinatario = models.ForeignKey(
        'anagrafiche.Anagrafica',
        on_delete=models.PROTECT,
        related_name='prestiti_ricevuti'
    )

    data_prestito = models.DateTimeField(default=timezone.now)
    data_rientro_prevista = models.DateField()
    data_rientro_effettiva = models.DateField(null=True, blank=True)

    richiedente_interno = models.ForeignKey(
        settings.AUTH_USER_MODEL,
        on_delete=models.SET_NULL,
        null=True,
        related_name='prestiti_richiesti'
    )

    causale = models.CharField(max_length=255)
    note = models.TextField(blank=True)

    # Stati
    aperto = models.BooleanField(default=True, db_index=True)
    solleciti_inviati = models.PositiveIntegerField(default=0)
```

```
    # Eventi collegati
    evento_prestito = models.ForeignKey(
        'EventoTracciabilita',
        on_delete=models.SET_NULL,
        null=True,
        related_name='prestito_generato'
    )
    evento_rientro = models.ForeignKey(
        'EventoTracciabilita',
        on_delete=models.SET_NULL,
        null=True,
        related_name='rientro_generato'
    )

    class Meta:
        verbose_name = 'Prestito Documento'
        verbose_name_plural = 'Prestiti Documenti'
        ordering = ['-data_prestito']
        indexes = [
            models.Index(fields=['aperto', 'data_rientro_prevista']),
        ]

    @property
    def giorni_ritardo(self) -> int:
        if not self.aperto or not self.data_rientro_prevista:
            return 0
        oggi = timezone.now().date()
        if oggi <= self.data_rientro_prevista:
            return 0
        return (oggi - self.data_rientro_prevista).days

    @property
    def in_ritardo(self) -> bool:
        return self.giorni_ritardo > 0

    def registra_rientro(self, *, data_rientro=None, utente=None):
        """Registra rientro prestito"""
        if not self.aperto:
            raise ValidationError("Prestito già chiuso")

        data_rientro = data_rientro or timezone.now().date()

        # Crea evento rientro
```

```
        evento = EventoTracciabilita.objects.create(
            documento=self.documento,
            fascicolo=self.fascicolo,
            tipo_evento=EventoTracciabilita.TipoEvento.RESTITUZIONE,
            utente=utente,
            referente_esterno=self.destinatario,
            note=f"Rientro prestito del {self.data_prestito:%Y-%m-%d}"
        )

        self.aperto = False
        self.data_rientro_effettiva = data_rientro
        self.evento_rientro = evento
        self.save(update_fields=['aperto', 'data_rientro_effettiva', 'evento_rientro'])

        return evento
```

Task Celery per alert:

```
# FILE: tracciabilita/tasks.py

@shared_task
def invia_alert_prestiti_scaduti():
    """
    Task giornaliero: invia alert per prestiti scaduti
    """
    oggi = timezone.now().date()

    prestiti_scaduti = PrestitoDocumento.objects.filter(
        aperto=True,
        data_rientro_prevista__lt=oggi
    ).select_related('destinatario', 'richiedente_interno')

    for prestito in prestiti_scaduti:
        giorni_ritardo = prestito.giorni_ritardo

        # Invia email sollecito
        send_mail(
            subject=f"Sollecito rientro prestito - {giorni_ritardo} giorni di ritardo",
            message=f"""
            Gentile {prestito.destinatario.display_name()},

            Le ricordiamo che il prestito del documento/fascicolo:
            {prestito.documento or prestito.fascicolo}

            è scaduto il {prestito.data_rientro_prevista:%d/%m/%Y}.
            Ritardo attuale: {giorni_ritardo} giorni.

            La preghiamo di restituire quanto prima.

            Cordiali saluti
            """,
            from_email=settings.DEFAULT_FROM_EMAIL,
            recipient_list=[prestito.destinatario.email]
        )

        prestito.solleciti_inviati += 1
        prestito.save(update_fields=['solleciti_inviati'])
```

Dashboard prestiti:

```
// Frontend React component
export const DashboardPrestiti: React.FC = () => {
  const [prestiti, setPrestiti] = useState<Prestito[]>([]);

  const prestitiInRitardo = prestiti.filter(p => p.in_ritardo);
  const prestitiInScadenza = prestiti.filter(p =>
    !p.in_ritardo && daysUntil(p.data_rientro_prevista) <= 7
  );

  return (
    <div className="dashboard-prestiti">
      <div className="alert alert-danger">
        <h4>⚠ Prestiti in Ritardo: {prestitiInRitardo.length}</h4>
        {prestitiInRitardo.map(p => (
          <PrestitoCard key={p.id} prestito={p} />
        ))}
      </div>

      <div className="alert alert-warning">
        <h4>  In Scadenza (7 giorni): {prestitiInScadenza.length}</h4>
        {prestitiInScadenza.map(p => (
          <PrestitoCard key={p.id} prestito={p} />
        ))}
      </div>
    </div>
  );
};
```

Impatto:
- Gestione prestiti completa
- Alert automatici scadenze
- Dashboard prestiti in ritardo
- Solleciti automatici

PRIORITÀ P3: Reportistica e Analytics

Obiettivo: Dashboard e report analitici movimenti archivio

```
# FILE: tracciabilita/reports.py

class ReportTracciabilita:
    """
    Generatore report analitici tracciabilità
    """

    @staticmethod
    def movimenti_per_periodo(data_inizio, data_fine, tipo_evento=None):
        """Report movimenti raggruppati per tipo"""
        qs = EventoTracciabilita.objects.filter(
            data_ora__gte=data_inizio,
            data_ora__lte=data_fine
        )

        if tipo_evento:
            qs = qs.filter(tipo_evento=tipo_evento)

        return qs.values('tipo_evento').annotate(
            totale=Count('id'),
            documenti=Count('documento', distinct=True),
            fascicoli=Count('fascicolo', distinct=True)
        ).order_by('-totale')

    @staticmethod
    def top_utenti_attivi(data_inizio, data_fine, limit=10):
        """Utenti più attivi nel periodo"""
        return EventoTracciabilita.objects.filter(
            data_ora__gte=data_inizio,
            data_ora__lte=data_fine
        ).values(
            'utente__username',
            'utente__first_name',
            'utente__last_name'
        ).annotate(
            totale_eventi=Count('id')
        ).order_by('-totale_eventi')[:limit]

    @staticmethod
    def utilizzo_ubicazioni(data_inizio, data_fine):
        """Statistiche utilizzo ubicazioni fisiche"""
        return EventoTracciabilita.objects.filter(
            data_ora__gte=data_inizio,
```

```
            data_ora__lte=data_fine,
            tipo_evento__in=['mov_entrata', 'mov_interno']
        ).values(
            'ubicazione_successiva__full_path',
            'ubicazione_successiva__nome'
        ).annotate(
            ingressi=Count('id')
        ).order_by('-ingressi')

    @staticmethod
    def tempo_medio_archiviazione(cliente=None):
        """Tempo medio tra protocollazione e archiviazione"""
        from protocollo.models import MovimentoProtocollo

        qs = EventoTracciabilita.objects.filter(
            tipo_evento='mov_entrata'
        ).select_related('documento', 'fascicolo')

        if cliente:
            qs = qs.filter(
                Q(documento__cliente=cliente) |
                Q(fascicolo__cliente=cliente)
            )

        tempi = []
        for evento in qs:
            target = evento.documento or evento.fascicolo
            protocollo = MovimentoProtocollo.objects.filter(
                Q(documento=evento.documento) | Q(fascicolo=evento.fascicolo)
            ).order_by('data').first()

            if protocollo:
                delta = evento.data_ora - protocollo.data
                tempi.append(delta.total_seconds() / 3600)  # ore

        if not tempi:
            return None

        return {
            'media_ore': sum(tempi) / len(tempi),
            'mediana_ore': sorted(tempi)[len(tempi) // 2],
            'campione': len(tempi)
        }
```

API endpoint report:

```
# FILE: api/v1/tracciabilita/views.py (aggiunta)

@action(detail=False, methods=['get'])
def report_movimenti(self, request):
    """
    Report movimenti per periodo
    GET /api/v1/tracciabilita/eventi/report-movimenti/
        ?data_inizio=2025-01-01&data_fine=2025-12-31
    """
    data_inizio = parse_date(request.query_params.get('data_inizio'))
    data_fine = parse_date(request.query_params.get('data_fine'))

    if not data_inizio or not data_fine:
        return Response(
            {'error': 'Parametri data_inizio e data_fine obbligatori'},
            status=status.HTTP_400_BAD_REQUEST
        )

    report = ReportTracciabilita.movimenti_per_periodo(
        data_inizio, data_fine
    )

    return Response({
        'periodo': {
            'inizio': data_inizio,
            'fine': data_fine
        },
        'dati': report
    })
```

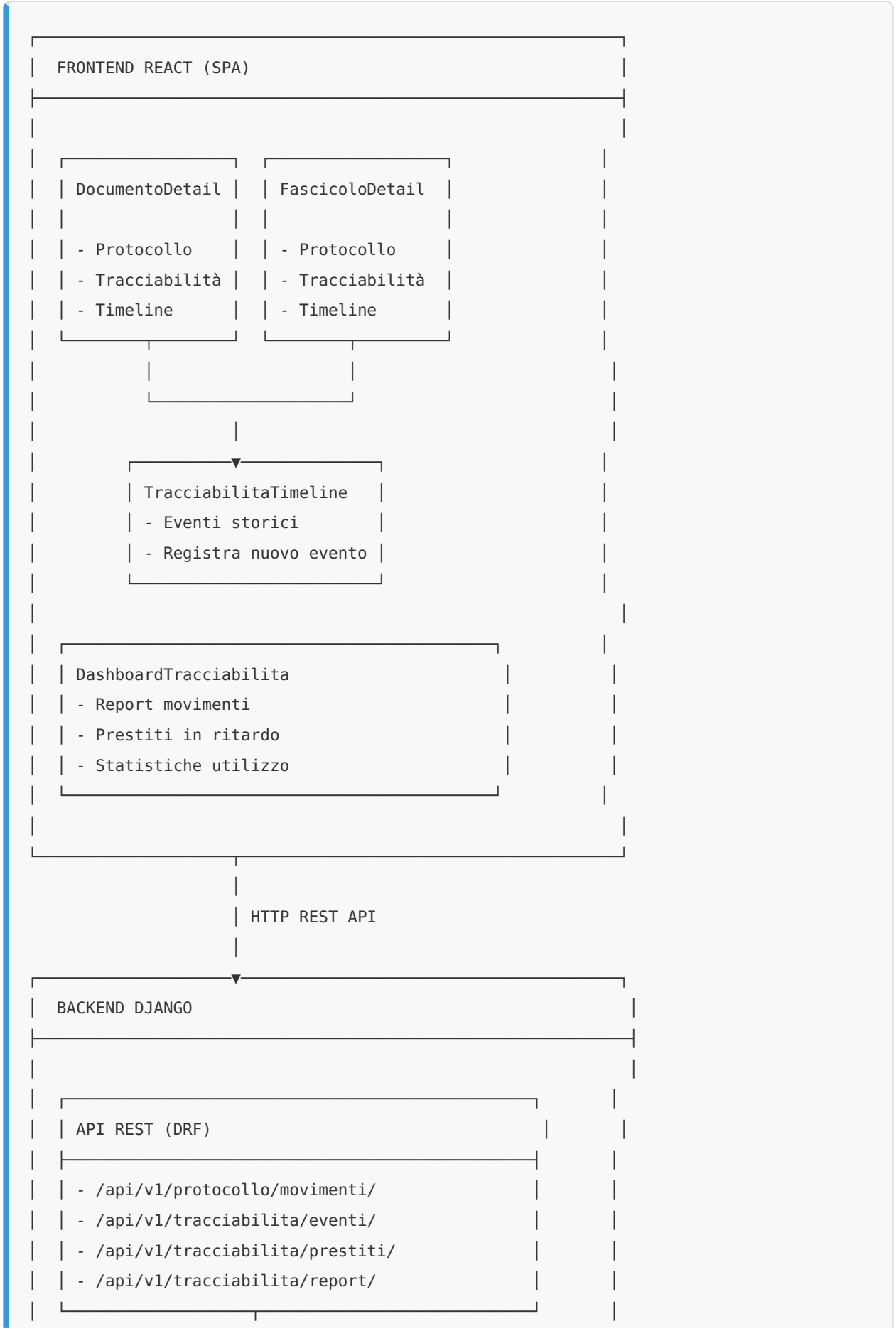
Dashboard React:

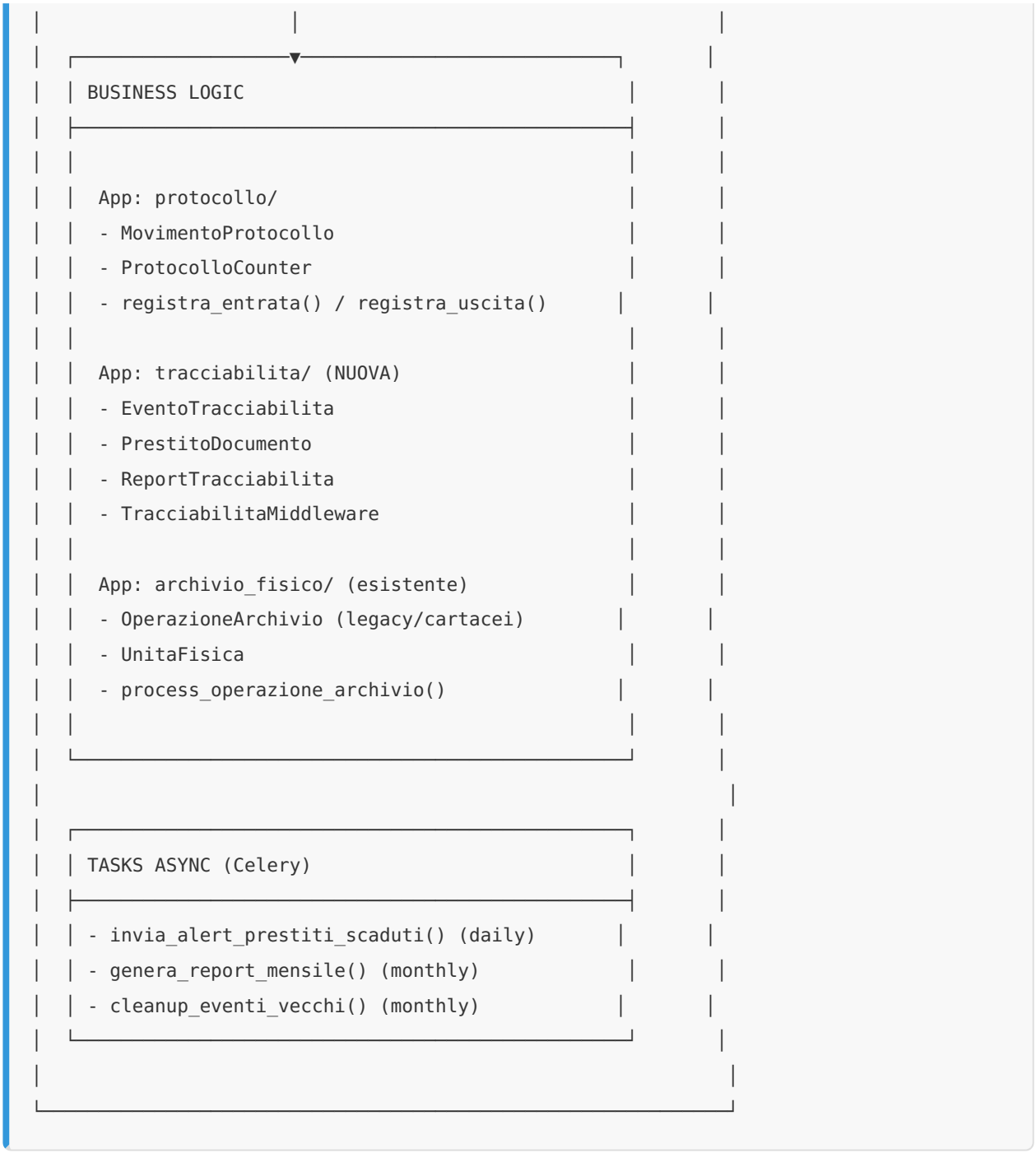
```
// FILE: frontend/src/pages/DashboardTracciabilita.tsx

export const DashboardTracciabilita: React.FC = () => {
  const [reportMovimenti, setReportMovimenti] = useState(null);
  const [topUtenti, setTopUtenti] = useState([]);

  useEffect(() => {
    // Carica dati dashboard
    fetchReportMovimenti();
    fetchTopUtenti();
  }, []);

  return (
    <div className="dashboard-tracciabilita">
      <h1>Dashboard Tracciabilità</h1>

      {/* Statistiche generali */}
      <div className="stats-grid">
        <StatCard
          title="Movimenti Oggi"
          value={reportMovimenti?.oggi || 0}
          icon=" "
        />
        <StatCard
          title="Prestiti Aperti"
          value={reportMovimenti?.prestiti_aperti || 0}
          icon=" "
        />
        <StatCard
          title="In Ritardo"
          value={reportMovimenti?.in_ritardo || 0}
          icon="⚠"
          variant="danger"
        />
      </div>

      {/* Grafico movimenti per tipo */}
      <div className="chart-container">
        <h3>Movimenti per Tipo</h3>
        <BarChart data={reportMovimenti?.per_tipo || []} />
      </div>

      {/* Top utenti attivi */}
```

```
      <div className="top-users">
        <h3>Utenti Più Attivi</h3>
        <table className="table">
          <thead>
            <tr>
              <th>Utente</th>
              <th>Eventi</th>
            </tr>
          </thead>
          <tbody>
            {topUtenti.map(u => (
              <tr key={u.username}>
                <td>{u.nome_completo}</td>
                <td>{u.totale_eventi}</td>
              </tr>
            ))}
          </tbody>
        </table>
      </div>
    </div>
  );
};
```

Impatto:
- Visibilità completa movimenti archivio
- KPI e metriche prestazioni
- Decision support per ottimizzazione
- Audit e compliance facilitati

9.7 Architettura Proposta Completa


FRONTEND REACT (SPA)
DocumentoDetail
- Protocollo
- Tracciabilità
- Timeline
FascicoloDetail
- Protocollo
- Tracciabilità
- Timeline
TracciabilitaTimeline
- Eventi storici
- Registra nuovo evento
DashboardTracciabilita
- Report movimenti
- Prestiti in ritardo
- Statistiche utilizzo
HTTP REST API
BACKEND DJANGO
API REST (DRF)
- /api/v1/protocollo/movimenti/
- /api/v1/tracciabilita/eventi/
- /api/v1/tracciabilita/prestiti/
- /api/v1/tracciabilita/report/


BUSINESS LOGIC
App: protocollo/
- MovimentoProtocollo
- ProtocolloCounter
- registra_entrata() / registra_uscita()
App: tracciabilita/ (NUOVA)
- EventoTracciabilita
- PrestitoDocumento
- ReportTracciabilita
- TracciabilitaMiddleware
App: archivio_fisico/ (esistente)
- OperazioneArchivio (legacy/cartacei)
- UnitaFisica
- process_operazione_archivio()
TASKS ASYNC (Celery)
- invia_alert_prestiti_scaduti() (daily)
- genera_report_mensile() (monthly)
- cleanup_eventi_vecchi() (monthly)

9.8 Roadmap di Adeguamento Complessivo

Fase	Durata stimata	Obiettivo sintetico
0	1 sprint	Protocollo universale IN/OUT per qualunque oggetto, pronto per email/WhatsApp
1	1-2 sprint	Nuova app `tracciabilita/` con causali configurabili e automazioni stato
2	1-2 sprint	Frontend React unico (timeline, modali, dashboard) e API simmetriche
3	1 sprint	Integrazione archivio_fisico e verbali collegati alla tracciabilità
4	1 sprint	Reporting, monitoraggio e preparazione future comunicazioni digitali

Fase 0 – Protocollo universale (Sprint 1)

- Rinforzare il vincolo di **univocità IN/OUT** su base (cliente, anno, direzione, numero) garantendo movimenti ripetuti sullo stesso oggetto.
- Estendere `MovimentoProtocollo` con un campo `content_type` / `generic_object_id` per permettere la protocollazione di **qualsiasi oggetto**, comprese future entità `ComunicazioneEmail` o `ComunicazioneWhatsApp`.
- Aggiornare API DRF e componenti React di protocollazione per accettare il nuovo schema (selezione tipo oggetto, autodetect direzione).
- Preparare template di protocollazione specifici per comunicazioni digitali (allegati, riferimenti messaggio) pur mantenendo il principio IN/OUT.
- Aggiornare test suite e migrazioni garantendo retrocompatibilità con documenti e fascicoli.

Stato Sprint 1 – Protocollo universale (21 Nov 2025)

- **Schema dati**: `MovimentoProtocollo` ora salva `target_content_type`, `target_object_id`, `target_tipo`, `target_label` e GFK (`target_object`) per rappresentare qualsiasi oggetto protocollabile, mantenendo i FK legacy per documenti/fascicoli.
- **Migrazione 0005**: riempie i nuovi campi partendo dallo storico e sostituisce il vecchio vincolo `_ensure_univoco`, consentendo cicli IN/OUT multipli sullo stesso oggetto ma preservando l'unicità `(cliente, anno, direzione, numero)`.

- **API**: nuovo endpoint `POST /api/v1/protocollo/movimenti/protocolla-target/` + serializer esteso (`target_type`, `target_id`, `target_label`) per protocollare future entità (email, WhatsApp, ecc.).
- **Admin/UI**: elenco admin e listing REST espongono `target_label` e `target_tipo`; la `ProtocolloModal` React può invocare l'endpoint generico quando riceve `targetType/targetId`.
- **Test**: suite Django `protocollo` e build Vite in esecuzione (coprono target fields, nuova API, regressioni IN/OUT); log `venv/bin/python manage.py test protocollo` + `npm run build` allegati in report sprint.

Fase 1 – App `tracciabilita/` dedicata (Sprint 2-3)

- Creare l'app Django separata con i modelli `EventoTracciabilita`, `OggettoTracciabile` e `CausaleTracciabilita` (gestite in admin per definire comportamento, impatto stato, obbligo verbale, trigger notifiche).
- Introdurre un flag esplicito `is_tracciabile` su documenti/fascicoli/comunicazioni: solo gli oggetti dichiarati tali generano eventi.
- Implementare servizi di automazione stato (ingresso, movimento interno, uscita verso esterno) basati sulle causali configurate.
- Esporre API REST CRUD e azioni rapide (`registra-movimento`, `chiudi-prestito`) con permessi granulari.
- Script di sincronizzazione iniziale: importare gli storici `OperazioneArchivio` e protocolli attivi.

Fase 2 – Esperienza React end-to-end (Sprint 4-5)

- Unificare tutte le interfacce in React (React Router + modali) eliminando template Django: protocollazione, timeline, dashboard tracciabilità, gestione causali.
- Componenti chiave: `ProtocolloModal`, `TracciabilitaTimeline`, `DashboardMovimenti`, `GestioneCausaliPage`, tutti con stato centralizzato (es. Zustand/Redux) e chiamate alle nuove API.
- Implementare pattern di **optimistic update** e notifiche real-time (WebSocket/fetch interval) per evidenziare movimenti appena registrati.
- Hardening UX: filtri avanzati, ricerca full-text, badge direzione IN/OUT coerenti.

Fase 3 – Integrazione archivio_fisico e verbali (Sprint 6)

- Mantenere `archivio_fisico` come fonte della verità per ubicazioni fisiche e generazione dei verbali di consegna/ritiro.
- Allineare gli eventi `OperazioneArchivio` con `EventoTracciabilita` tramite segnali o job programmato (ogni operazione fisica crea/aggiorna un evento di tracciabilità).

- Adeguare i verbali (PDF, firma, allegati) per puntare al nuovo ID evento, garantendo storicità e link bidirezionale.
- Validare i processi di consegna in entrata/uscita con casi misti (documenti cartacei, fascicoli, kit documentali).

Fase 4 – Reporting e readiness comunicazioni digitali (Sprint 7)

- Costruire report e dashboard (React) basati su `ReportTracciabilita` con KPI: movimenti per causale, SLA, prestiti aperti, oggetti non rientrati.
- Automatizzare alert e webhook verso sistemi terzi (es. notifiche Teams) per ritardi o anomalie.
- Preparare i moduli `ComunicazioneEmail` / `ComunicazioneWhatsApp`: definizione dei modelli, mapping con protocollazione e tracciabilità, mock API per future integrazioni.
- Documentare end-to-end (manuali, diagrammi sequenza, FAQ) e chiudere con revisione di sicurezza/compliance.

Output finale: protocollo unico per ogni oggetto (IN/OUT), nuova app di tracciabilità governata da causali configurabili, archivio_fisico focalizzato su ubicazioni e verbali ma collegato agli eventi, frontend interamente React e pronto ad includere email/WhatsApp.

---

10. RIEPILOGO ESECUTIVO ESTESO

Fase 1: Correzioni Critiche (Sprint 1)

Durata: 1-2 settimane

1. **Rimozione validazione univocità** (#8.1)
   - Modifica `_ensure_univoco()`
   - Test movimenti multipli
   - Aggiornamento documentazione

2. **Lock uscite fascicoli** (#8.2)
   - Migrazione campo `out_aperto`
   - Modifica logica registrazione
   - Test coerenza con documenti

Deliverable: Sistema protocollo funzionante per movimenti ripetuti

Fase 2: Miglioramenti Strutturali (Sprint 2-3)

Durata: 2-3 settimane

1. **Campo** `fascicolo.digitale` (#8.3)
   - Migrazione con conversione dati
   - Constraint DB
   - Aggiornamento form/API

2. **Separazione** `da_chi` / `a_chi` (#8.4)
   - Migrazione con split campo
   - Aggiornamento serializers
   - Deprecazione API vecchia (se pubblica)

Deliverable: Modello dati più robusto e chiaro

Fase 3: Enhancement UX (Sprint 4)

Durata: 1 settimana

1. **Property metodi stato** (#8.5)
   - Aggiunta property al modello
   - Dashboard uscite in ritardo
   - Alert automatici (opzionale)

2. **Gestione automatica ubicazione** (#8.6)
   - Metodi helper Documento
   - Aggiornamento serializers
   - Form auto-compilazione

Deliverable: UX ottimizzata, meno errori utente

---

10. CONSIDERAZIONI AGGIUNTIVE

10.1 Storico Movimenti

Attuale: Tutti i movimenti conservati, nessuna cancellazione

Pro:
- Audit trail completo
- Tracciabilità legale
- Analisi storiche

Cons:
- ⚠ Crescita DB nel tempo
- ⚠ Nessuna archiviazione/compattazione

Suggerimento:
- Report periodici movimenti vecchi
- Esportazione dati storici (CSV/PDF)
- Soft delete con campo `archiviato` (futuro)

10.2 Performance

Query critiche:

```
# Uscite aperte per documento
MovimentoProtocollo.objects.filter(
    documento=doc, direzione='OUT', chiuso=False
)  #   Indice: (documento, direzione, chiuso)

# Ultimo movimento fascicolo
MovimentoProtocollo.objects.filter(
    fascicolo=fasc
).order_by('-data')  #   Indice: (fascicolo, data DESC)

# Progressivo counter
ProtocolloCounter.objects.get(
    cliente=cli, anno=2025, direzione='IN'
)  #   unique_together index
```

Ottimizzazioni presenti:
- Index su (documento, direzione, chiuso)
- Index su (fascicolo, direzione, chiuso)
- Index su (cliente, anno, direzione, numero)
- Atomic counter con F() expression

Nessun problema performance rilevato

10.3 Sicurezza e Permessi

Attuale: Permessi gestiti da Django `@login_required`

Mancante:
- ⚠ Permessi granulari per protocollazione
- ⚠ Audit log modifiche (chi/quando)
- ⚠ Restrizioni per cliente (multi-tenant)

Suggerimento (se necessario):

```
# FILE: protocollo/permissions.py

class CanProtocollaDocumento(BasePermission):
    def has_permission(self, request, view):
        return request.user.has_perm('protocollo.protocolla_documento')

    def has_object_permission(self, request, view, obj):
        # Verifica accesso al cliente del documento
        return request.user.can_access_cliente(obj.cliente)
```

10.4 Integrazione Google Calendar

Attuale: File `GOOGLE_CALENDAR_DISABLED.md` presente → disabilitato

Potenziale: Eventi calendario per:
- Uscite con `data_rientro_prevista`
- Alert rientri scaduti
- Reminder pre-scadenza

Priorità: Bassa (funzionalità nice-to-have)

---

10. RIEPILOGO ESECUTIVO ESTESO

10.1 Distinzione Fondamentale

PROTOCOLLAZIONE vs **TRACCIABILITÀ**:
- Protocollazione: atto **puntuale** e **certificativo** (una sola volta)
- Tracciabilità: processo **continuo** per tutta la vita del documento

Entrambi necessari ma con finalità diverse:
- Protocollo → valore legale, certezza data, identificazione univoca
- Tracciabilità → gestione operativa, audit trail, trasparenza

10.2 Sistema Protocollo: Punti di Forza

Architettura modulare ben organizzata
Generazione progressivi atomica e robusta
Gestione ubicazioni articolata e completa
Lock uscite per documenti (previene inconsistenze)

Integrazione API REST moderna
Storico movimenti completo (audit trail)
Frontend React con popup window

10.3 Sistema Protocollo: Criticità

P0: Validazione univocità blocca movimenti multipli **(BLOCCO OPERATIVO)**
P1: Nessun lock uscite per fascicoli (rischio inconsistenze)
P2: Campo `destinatario` ambiguo (semantica da_chi/a_chi)
P2: Distinzione digitale/cartaceo fascicoli implicita

10.4 Sistema Tracciabilità: Punti di Forza

Architettura solida con separazione responsabilità
Modellazione gerarchica UnitaFisica
Audit trail completo operazioni cartacei
Verbali formali auto-generati
Validazioni robuste (cartacei/tracciabili/protocollati)
Operazioni batch multi-documento
Gestione stati automatica

10.5 Sistema Tracciabilità: Criticità

P0: **Nessuna tracciabilità documenti digitali (LACUNA CRITICA)**
P1: Interfaccia solo Django templates (no API REST)
P1: Dipendenza obbligatoria da protocollo esistente
P2: Nessuna gestione prestiti con scadenze
P2: Nessuna reportistica analitica
P3: Complessità configurazione UnitaFisica
P3: Nessun lock concorrenza movimenti

10.6 Azioni Prioritarie Integrate

Priority	Area	Azione	Effort	Risk	Impact
P0	Protocollo	Rimuovere validazione univocità	1d	Low	High
P0	Tracciabilità	Implementare `EventoTracciabilita` (digitali)	5d	Med	High
P1	Protocollo	Aggiungere lock `out_aperto` fascicoli	2-3d	Med	Med
P1	Tracciabilità	API REST + Frontend React	5d	Med	High
P2	Tracciabilità	Gestione prestiti con alert	3d	Low	Med
P2	Protocollo	Campo `fascicolo.digitale` esplicito	3-4d	Med	Low
P3	Tracciabilità	Dashboard analytics	3d	Low	Low
P3	Protocollo	Split `da_chi` / `a_chi`	3-4d	High	Low

10.7 Vision Architettura Target

10.8 Roadmap Consigliata (7 Sprint, ~14 settimane)

Sprint	Focus	Deliverable chiave
1	Protocollo universale	Nuovo schema oggetti protocollabili (documenti, fascicoli, comunicazioni) con direzione IN/OUT e frontend React aggiornato
2-3	App `tracciabilita/`	Modelli eventi + causali configurabili, API DRF e servizi di automazione stato per oggetti dichiarati tracciabili
4-5	Esperienza React	Timeline, dashboard e gestione causali 100% React, sostituzione template Django, notifiche in tempo quasi reale
6	Archivio fisico + verbali	Allineamento eventi fisici, generazione verbali collegati, sincronizzazione ubicazioni, regressione completa
7	Reporting & comunicazioni future	KPI, alert, readiness per protocollare/tracciare email e WhatsApp, documentazione finale

Nota frontend: tutti i nuovi flussi lato utente (protocollazione, timeline, dashboard, causali, verbali) devono vivere esclusivamente nell'app React esistente, riutilizzando gli stessi pattern di stato e component library per assicurare continuità UX.

10.9 Soluzioni Alternative Valutate

Alternativa 1: Estendere OperazioneArchivio per Digitali

Pro:
- Nessuna nuova app
- Logica esistente riutilizzata

Contro:
- Modello pensato per movimenti fisici
- Campi ubicazione_sorgente/destinazione non applicabili
- Verbali cartacei non servono per digitali
- Accoppiamento concetti diversi

Verdict: **Sconsigliata** - forzatura concettuale

Alternativa 2: App Audit Generica

Pro:
- Applicabile a tutti i modelli Django
- Tracciabilità universale

Contro:
- Troppo generica (perdita semantica dominio)
- Nessuna logica business specifica
- Overhead per casi d'uso semplici

Verdict: ⚠ **Futura evoluzione** - dopo tracciabilità base

Alternativa 3: App Tracciabilita Dedicata

Pro:
- Separazione netta protocollo/tracciabilità
- Estensibile per eventi futuri
- Modello dati semanticamente corretto
- Retrocompatibilità con OperazioneArchivio

Contro:
- ⚠ Nuova app (complessità iniziale)
- ⚠ Migrazione dati necessaria

Verdict: **RACCOMANDATA** - soluzione ottimale

10.10 Metriche di Successo

KPI Tecnici:
- Copertura test ≥ 80%
- Tempo risposta API < 200ms (p95)
- Zero regressioni funzionali
- Downtime migrazione < 1h

KPI Funzionali:
- 100% documenti tracciabili (cartacei + digitali)
- Audit trail completo ≥ 95% eventi
- Prestiti in ritardo < 5%
- Tempo medio archiviazione < 7 giorni

KPI Utente:
- Soddisfazione utenti ≥ 4/5
- Tempo medio registrazione movimento < 2min
- Errori utente < 1% operazioni

10.11 Rischi e Mitigazioni

Rischio	Probabilità	Impatto	Mitigazione
Migrazione dati fallita	Media	Alto	Backup completo, rollback plan, testing staging
Performance query eventi	Media	Medio	Indici DB, caching, pagination
Resistenza utenti nuova UI	Bassa	Medio	Training, documentazione, supporto
Regressioni funzionali	Media	Alto	Test suite estesa, QA manuale, feature toggle
Complessità implementazione	Alta	Medio	Suddivisione sprint, code review, pair programming

10.12 Conclusioni Finali

Il sistema MyGest presenta:

1. **Protocollo**: Base solida ma con bug critico (univocità) che blocca operatività. Una volta risolto, sistema pienamente funzionale.
2. **Tracciabilità**: Lacuna critica per documenti digitali. Sistema cartacei ben progettato ma isolato (no API, no React).
3. **Opportunità**: Unificare architettura con app `tracciabilita/` dedicata permette:
   - Tracciabilità universale (cartacei + digitali)
   - UX coerente React end-to-end
   - Compliance normativa completa
   - Scalabilità futura (audit generica, ML analytics)
4. **Priorità assoluta**:
   - **Protocollo**: Fix univocità (1 giorno)
   - **Tracciabilità**: Implementazione digitali (1-2 sprint)
5. **Investimento complessivo**: 14-16 settimane per sistema completo

Raccomandazione: Procedere con roadmap proposta. ROI elevato per compliance, efficienza operativa e qualità audit trail.

Fine Analisi Estesa
Generato automaticamente da GitHub Copilot - 21 Novembre 2025

Sistema Attuale: Punti di Forza

Architettura modulare ben organizzata
Generazione progressivi atomica e robusta
Gestione ubicazioni articolata e completa
Lock uscite per documenti (previene inconsistenze)
Integrazione API REST moderna
Storico movimenti completo (audit trail)

Sistema Attuale: Punti Critici

Validazione univocità blocca movimenti multipli **(BLOCCO OPERATIVO)**
Nessun lock uscite per fascicoli (rischio inconsistenze)
Campo `destinatario` ambiguo (semantica da_chi/a_chi)
Distinzione digitale/cartaceo fascicoli implicita

Azioni Prioritarie

1. **Immediato**: Rimuovere/modificare validazione univocità
2. **Breve termine**: Aggiungere lock `out_aperto` fascicoli
3. **Medio termine**: Campo `digitale` esplicito fascicoli
4. **Opzionale**: Separazione campi `da_chi` / `a_chi`

Stima Impatto Modifiche

Modifica	Effort	Risk	Impact	Priority
Fix univocità	1d	Low	High	**P0**
Lock fascicoli	2-3d	Med	Med	**P1**
Campo digitale	3-4d	Med	Low	P2
Split da_chi/a_chi	3-4d	High	Low	P3

Conclusioni

Il sistema di protocollo MyGest è ben progettato ma presenta **un bug critico** che impedisce l'uso normale (movimenti multipli bloccati). Una volta risolto questo problema prioritario, il sistema sarà pienamente funzionale. I miglioramenti successivi sono incrementali e possono essere pianificati in base alle priorità operative.

Fine Analisi
Generato automaticamente da GitHub Copilot - 21 Novembre 2025