MyGest

Conservazione Digitale a Norma

Analisi Completa e Progettazione

Versione: 1.0

Data: 17 Novembre 2025

Documento: Analisi Tecnica Completa

Autore: Team Sviluppo MyGest

Conservazione Digitale a Norma – Analisi e Progettazione

Data: 17 Novembre 2025

PARTE 1: ANALISI NORMATIVA

1.1 Quadro Normativo Italiano

Normative di Riferimento

1. **CAD (Codice Amministrazione Digitale)** - D.Lgs. 82/2005
2. Art. 43: Copie informatiche di documenti analogici
3. Art. 44: Requisiti per la conservazione dei documenti informatici
4. **DPCM 3 dicembre 2013**
5. Regole tecniche in materia di sistema di conservazione
6. **Linee Guida AgID**
7. Linee guida sulla formazione, gestione e conservazione dei documenti informatici
8. Versione corrente: 2021
9. **Regolamento eIDAS** (UE) n. 910/2014
10. Firma elettronica qualificata
11. Marcatura temporale

Requisiti Obbligatori

1. **Integrità**: Documento non modificabile dopo versamento in conservazione
2. **Autenticità**: Certezza dell'autore e della data
3. **Leggibilità**: Permanenza nel tempo
4. **Immodificabilità**: Hash + firma digitale + marcatura temporale
5. **Conformità**: Metadati obbligatori secondo standard

1.2 Soggetti e Ruoli

Responsabile della Conservazione

Figura obbligatoria che: - Definisce e attua le politiche di conservazione - Verifica periodicamente l'integrità degli archivi - Genera il rapporto di versamento - Effettua il versamento nel sistema di conservazione - Garantisce la conformità normativa

Conservatore Accreditato (Opzionale)

Per servizi in outsourcing, il conservatore deve essere: - Accreditato AgID - Conforme ISO 14721 (OAIS - Open Archival Information System) - Certificato secondo standard di sicurezza

PARTE 2: ARCHITETTURA PROPOSTA

2.1 Nuova App: `conservazione`

Struttura Directory

```
conservazione/
├── __init__.py
├── admin.py
├── apps.py
├── models.py              # Modelli principali
├── views.py               # API e views
├── urls.py
├── forms.py
├── signals.py
├── tasks.py               # Celery tasks per processi asincroni
├── utils.py
├── managers.py            # Manager per PdV e lotti
├── validators.py          # Validazione conformità
├── migrations/
├── templates/
│   └── conservazione/
│       ├── dashboard.html
│       ├── pacchetto_list.html
│       ├── pacchetto_detail.html
│       ├── lotto_list.html
│       └── rapporto_versamento.html
├── services/
│   ├── __init__.py
│   ├── firma_digitale.py      # Integrazione firma digitale
│   ├── marcatura_temporale.py # Integrazione marca temporale
│   ├── hash_service.py         # Calcolo hash SHA-256
│   ├── pdv_builder.py          # Costruzione PdV
│   ├── export_service.py       # Export in formati standard
│   └── verifica_service.py     # Verifica integrità
├── formatters/
│   ├── __init__.py
│   ├── pdf_a.py               # Conversione PDF/A
│   ├── xml_indice.py          # XML indice conservazione
│   └── metadata_builder.py     # Costruzione metadati
└── tests/
    ├── __init__.py
    ├── test_models.py
    ├── test_services.py
    └── test_conservazione.py
```

2.2 Modelli Principali

2.2.1 PacchettoVersamento (PdV)

```
from django.db import models
from django.contrib.auth import get_user_model
import uuid

User = get_user_model()

class PacchettoVersamento(models.Model):
    """
    Pacchetto di Versamento (PdV) - Unità base di conservazione
    Conforme alle specifiche AgID
    """

    class Stato(models.TextChoices):
        BOZZA = 'bozza', 'Bozza'
        IN_PREPARAZIONE = 'preparazione', 'In Preparazione'
        PRONTO = 'pronto', 'Pronto per Versamento'
        VERSATO = 'versato', 'Versato'
        ACCETTATO = 'accettato', 'Accettato in Conservazione'
        RIFIUTATO = 'rifiutato', 'Rifiutato'
        ERRORE = 'errore', 'Errore'

    # Identificativo univoco
    uuid = models.UUIDField(
        default=uuid.uuid4,
        editable=False,
        unique=True,
        db_index=True
    )

    # Codice identificativo (es: PdV_2025_001)
    codice = models.CharField(
        max_length=50,
        unique=True,
        db_index=True,
        help_text="Codice identificativo del PdV"
    )

    # Stato
    stato = models.CharField(
        max_length=20,
        choices=Stato.choices,
        default=Stato.BOZZA
    )

    # Date
    data_creazione = models.DateTimeField(auto_now_add=True)
    data_chiusura = models.DateTimeField(
        null=True,
        blank=True,
        help_text="Data chiusura del PdV"
```

```
    )
    data_versamento = models.DateTimeField(
        null=True,
        blank=True,
        help_text="Data invio al sistema di conservazione"
    )
    data_accettazione = models.DateTimeField(
        null=True,
        blank=True,
        help_text="Data accettazione dal sistema di conservazione"
    )

    # Responsabile
    responsabile = models.ForeignKey(
        User,
        on_delete=models.PROTECT,
        related_name='pacchetti_versamento'
    )

    # Metadati conservazione
    produttore = models.CharField(
        max_length=200,
        help_text="Soggetto produttore (es: nome azienda)"
    )

    periodo_inizio = models.DateField(
        help_text="Inizio periodo di riferimento documenti"
    )
    periodo_fine = models.DateField(
        help_text="Fine periodo di riferimento documenti"
    )

    # Hash del pacchetto
    hash_algoritmo = models.CharField(
        max_length=20,
        default='SHA-256',
        help_text="Algoritmo di hash utilizzato"
    )
    hash_valore = models.CharField(
        max_length=128,
        blank=True,
        help_text="Hash dell'intero pacchetto"
    )

    # Firma digitale
    firma_file = models.FileField(
        upload_to='conservazione/firme/',
        null=True,
        blank=True,
        help_text="File firma digitale P7M"
    )
```

```
    firma_data = models.DateTimeField(
        null=True,
        blank=True
    )
    firma_certificato_cn = models.CharField(
        max_length=200,
        blank=True,
        help_text="Common Name del certificato"
    )

    # Marca temporale
    marca_temporale_file = models.FileField(
        upload_to='conservazione/marche_temporali/',
        null=True,
        blank=True,
        help_text="File marca temporale TSR"
    )
    marca_temporale_data = models.DateTimeField(
        null=True,
        blank=True
    )
    marca_temporale_tsa = models.CharField(
        max_length=200,
        blank=True,
        help_text="TSA (Time Stamping Authority)"
    )

    # File del pacchetto
    file_indice_xml = models.FileField(
        upload_to='conservazione/pdv/indici/',
        null=True,
        blank=True,
        help_text="Indice XML del PdV"
    )
    file_zip = models.FileField(
        upload_to='conservazione/pdv/zip/',
        null=True,
        blank=True,
        help_text="Archivio ZIP del PdV"
    )

    # Rapporto di versamento
    rapporto_versamento = models.FileField(
        upload_to='conservazione/rapporti/',
        null=True,
        blank=True,
        help_text="Rapporto di versamento PDF"
    )

    # Note e messaggi errore
    note = models.TextField(blank=True)
```

```
    messaggio_errore = models.TextField(blank=True)

    class Meta:
        db_table = 'conservazione_pacchetto_versamento'
        verbose_name = 'Pacchetto di Versamento'
        verbose_name_plural = 'Pacchetti di Versamento'
        ordering = ['-data_creazione']
        indexes = [
            models.Index(fields=['stato', 'data_creazione']),
            models.Index(fields=['produttore', 'periodo_inizio']),
        ]

    def __str__(self):
        return f"{self.codice} - {self.get_stato_display()}"
```

2.2.2 DocumentoConservazione

```
class DocumentoConservazione(models.Model):
    """
    Documento inserito in un PdV per la conservazione
    """

    # Relazione con PdV
    pacchetto = models.ForeignKey(
        PacchettoVersamento,
        on_delete=models.PROTECT,
        related_name='documenti'
    )

    # Relazione con documento originale
    documento = models.ForeignKey(
        'documenti.Documento',
        on_delete=models.PROTECT,
        related_name='versioni_conservate'
    )

    # Identificativo nel PdV
    id_documento_pdv = models.CharField(
        max_length=100,
        help_text="ID univoco del documento nel PdV"
    )

    # File conservato (PDF/A)
    file_conservazione = models.FileField(
        upload_to='conservazione/documenti/',
        help_text="File in formato conservativo (PDF/A)"
    )

    # Hash del file
    hash_algoritmo = models.CharField(
        max_length=20,
        default='SHA-256'
    )
    hash_valore = models.CharField(
        max_length=128,
        help_text="Hash del file conservato"
    )

    # Dimensione file
    dimensione_byte = models.BigIntegerField(
        help_text="Dimensione in byte"
    )

    # Formato
    formato_originale = models.CharField(
        max_length=50,
        help_text="Formato originale (es: application/pdf)"
```

```
    )
    formato_conservazione = models.CharField(
        max_length=50,
        default='application/pdf',
        help_text="Formato di conservazione (PDF/A-2b)"
    )

    # Metadati minimi obbligatori
    oggetto = models.CharField(
        max_length=500,
        help_text="Oggetto/titolo del documento"
    )
    data_documento = models.DateField(
        help_text="Data del documento"
    )
    soggetto_produttore = models.CharField(
        max_length=200,
        help_text="Soggetto che ha prodotto il documento"
    )
    destinatario = models.CharField(
        max_length=200,
        blank=True,
        help_text="Destinatario del documento"
    )

    # Classificazione
    tipo_documento = models.CharField(
        max_length=100,
        help_text="Tipologia documentale"
    )

    # Firma digitale originale (se presente)
    ha_firma_originale = models.BooleanField(
        default=False,
        help_text="Il documento originale era firmato digitalmente"
    )

    # Timestamp
    data_inserimento = models.DateTimeField(auto_now_add=True)

    class Meta:
        db_table = 'conservazione_documento'
        verbose_name = 'Documento in Conservazione'
        verbose_name_plural = 'Documenti in Conservazione'
        unique_together = [['pacchetto', 'id_documento_pdv']]
        indexes = [
            models.Index(fields=['pacchetto', 'documento']),
            models.Index(fields=['hash_valore']),
        ]
```

```
    def __str__(self):
        return f"{self.id_documento_pdv} - {self.oggetto}"
```

2.2.3 LottoConservazione

```
class LottoConservazione(models.Model):
    """
    Lotto di conservazione - Insieme di PdV versati insieme
    """

    class Stato(models.TextChoices):
        APERTO = 'aperto', 'Aperto'
        CHIUSO = 'chiuso', 'Chiuso'
        VERSATO = 'versato', 'Versato'

    codice = models.CharField(
        max_length=50,
        unique=True,
        help_text="Codice lotto (es: LOTTO_2025_001)"
    )

    descrizione = models.CharField(max_length=200)

    stato = models.CharField(
        max_length=20,
        choices=Stato.choices,
        default=Stato.APERTO
    )

    data_apertura = models.DateTimeField(auto_now_add=True)
    data_chiusura = models.DateTimeField(null=True, blank=True)
    data_versamento = models.DateTimeField(null=True, blank=True)

    responsabile = models.ForeignKey(
        User,
        on_delete=models.PROTECT
    )

    # Pacchetti nel lotto
    pacchetti = models.ManyToManyField(
        PacchettoVersamento,
        related_name='lotti'
    )

    # Statistiche
    numero_documenti = models.IntegerField(default=0)
    dimensione_totale_mb = models.DecimalField(
        max_digits=12,
        decimal_places=2,
        default=0
    )

    class Meta:
        db_table = 'conservazione_lotto'
        verbose_name = 'Lotto di Conservazione'
```

```
        verbose_name_plural = 'Lotti di Conservazione'
        ordering = ['-data_apertura']
```

2.2.4 MetadatoConservazione

```
class MetadatoConservazione(models.Model):
    """
    Metadati aggiuntivi per documenti in conservazione
    Sistema estendibile per metadati custom
    """

    documento = models.ForeignKey(
        DocumentoConservazione,
        on_delete=models.CASCADE,
        related_name='metadati'
    )

    chiave = models.CharField(
        max_length=100,
        help_text="Nome del metadato"
    )

    valore = models.TextField(
        help_text="Valore del metadato"
    )

    tipo = models.CharField(
        max_length=20,
        choices=[
            ('string', 'Stringa'),
            ('date', 'Data'),
            ('number', 'Numero'),
            ('json', 'JSON'),
        ],
        default='string'
    )

    class Meta:
        db_table = 'conservazione_metadato'
        unique_together = [['documento', 'chiave']]
        indexes = [
            models.Index(fields=['chiave', 'valore']),
        ]
```

2.2.5 VerificaIntegrita

```
class VerificaIntegrita(models.Model):
    """
    Log delle verifiche periodiche di integrità
    Richiesto dalla normativa
    """

    class Esito(models.TextChoices):
        SUCCESSO = 'successo', 'Successo'
        FALLITO = 'fallito', 'Fallito'
        PARZIALE = 'parziale', 'Parziale'

    pacchetto = models.ForeignKey(
        PacchettoVersamento,
        on_delete=models.CASCADE,
        related_name='verifiche'
    )

    data_verifica = models.DateTimeField(auto_now_add=True)

    eseguita_da = models.ForeignKey(
        User,
        on_delete=models.PROTECT
    )

    esito = models.CharField(
        max_length=20,
        choices=Esito.choices
    )

    # Risultati verifica
    hash_verificato = models.BooleanField(default=False)
    firma_valida = models.BooleanField(default=False)
    marca_temporale_valida = models.BooleanField(default=False)
    file_accessibili = models.BooleanField(default=False)

    # Dettagli
    dettagli = models.JSONField(
        default=dict,
        help_text="Dettagli tecnici della verifica"
    )

    note = models.TextField(blank=True)

    class Meta:
        db_table = 'conservazione_verifica_integrita'
        verbose_name = 'Verifica di Integrità'
        verbose_name_plural = 'Verifiche di Integrità'
        ordering = ['-data_verifica']
```

2.3 Servizi di Integrazione

2.3.1 Firma Digitale

```
# conservazione/services/firma_digitale.py

import subprocess
from pathlib import Path
from typing import Optional, Tuple
import logging

logger = logging.getLogger(__name__)

class FirmaDigitaleService:
    """
    Servizio per firma digitale tramite token USB o HSM
    """

    def __init__(self, config: dict):
        """
        config = {
            'tipo': 'aruba|infocert|namirial|custom',
            'pin': 'PIN del token',
            'driver_path': '/path/to/driver',
            'certificate_cn': 'CN del certificato',
        }
        """
        self.config = config

    def firma_file(self, file_path: Path) -> Tuple[bool, Optional[Path], str]:
        """
        Firma digitalmente un file

        Returns:
            (successo, path_file_firmato, messaggio)
        """
        try:
            # Esempio con ArubaPEC
            if self.config['tipo'] == 'aruba':
                return self._firma_aruba(file_path)

            # Esempio con InfoCert
            elif self.config['tipo'] == 'infocert':
                return self._firma_infocert(file_path)

            # Firma con OpenSSL (per testing)
            elif self.config['tipo'] == 'openssl':
                return self._firma_openssl(file_path)

            else:
                return False, None, "Tipo firma non supportato"

        except Exception as e:
            logger.error(f"Errore firma digitale: {e}")
```

```
            return False, None, str(e)

    def _firma_openssl(self, file_path: Path) -> Tuple[bool, Optional[Path],
str]:
        """
        Firma con OpenSSL (per sviluppo/test)
        """
        output_path = file_path.with_suffix(file_path.suffix + '.p7m')

        cmd = [
            'openssl', 'smime',
            '-sign',
            '-in', str(file_path),
            '-out', str(output_path),
            '-signer', self.config['cert_path'],
            '-inkey', self.config['key_path'],
            '-outform', 'DER',
            '-nodetach',
        ]

        result = subprocess.run(cmd, capture_output=True, text=True)

        if result.returncode == 0:
            return True, output_path, "Firma applicata con successo"
        else:
            return False, None, f"Errore: {result.stderr}"

    def verifica_firma(self, file_p7m: Path) -> Tuple[bool, dict]:
        """
        Verifica la validità di una firma digitale

        Returns:
            (valida, dettagli)
        """
        try:
            cmd = [
                'openssl', 'smime',
                '-verify',
                '-in', str(file_p7m),
                '-inform', 'DER',
                '-noverify',  # Non verifica la CA (opzionale)
            ]

            result = subprocess.run(cmd, capture_output=True, text=True)

            dettagli = {
                'valida': result.returncode == 0,
                'output': result.stdout,
                'errori': result.stderr,
            }
```

```
            return result.returncode == 0, dettagli

        except Exception as e:
            logger.error(f"Errore verifica firma: {e}")
            return False, {'errore': str(e)}
```

2.3.2 Marcatura Temporale

```
# conservazione/services/marcatura_temporale.py

import requests
import hashlib
from pathlib import Path
from typing import Optional, Tuple
from datetime import datetime
import logging

logger = logging.getLogger(__name__)

class MarcaturaTemporaleService:
    """
    Servizio per applicare marche temporali RFC 3161
    """

    def __init__(self, config: dict):
        """
        config = {
            'tsa_url': 'https://tsa.example.com',
            'username': 'user',
            'password': 'pass',
            'cert_path': '/path/to/cert.pem',
        }
        """
        self.config = config

    def applica_marca(self, file_path: Path) -> Tuple[bool, Optional[Path],
dict]:
        """
        Applica marca temporale a un file

        Returns:
            (successo, path_file_tsr, info_marca)
        """
        try:
            # 1. Calcola hash del file
            file_hash = self._calcola_hash(file_path)

            # 2. Genera richiesta TSR
            tsr_request = self._genera_tsr_request(file_hash)

            # 3. Invia a TSA
            tsr_response = self._invia_tsa(tsr_request)

            # 4. Salva TSR
            tsr_path = file_path.with_suffix(file_path.suffix + '.tsr')
            tsr_path.write_bytes(tsr_response)

            # 5. Estrai info marca
```

```
            info = self._estrai_info_marca(tsr_response)

            return True, tsr_path, info

        except Exception as e:
            logger.error(f"Errore marcatura temporale: {e}")
            return False, None, {'errore': str(e)}

    def _calcola_hash(self, file_path: Path) -> bytes:
        """Calcola SHA-256 del file"""
        sha256 = hashlib.sha256()
        with open(file_path, 'rb') as f:
            for chunk in iter(lambda: f.read(4096), b''):
                sha256.update(chunk)
        return sha256.digest()

    def _genera_tsr_request(self, file_hash: bytes) -> bytes:
        """
        Genera richiesta TSR secondo RFC 3161
        In produzione usare libreria come 'python-tsp'
        """
        # Implementazione semplificata
        # In produzione: usare pyasn1 o cryptography
        raise NotImplementedError("Usare libreria python-tsp")

    def _invia_tsa(self, tsr_request: bytes) -> bytes:
        """Invia richiesta a TSA"""
        response = requests.post(
            self.config['tsa_url'],
            data=tsr_request,
            headers={'Content-Type': 'application/timestamp-query'},
            auth=(self.config.get('username'), self.config.get('password')),
            timeout=30,
        )
        response.raise_for_status()
        return response.content

    def verifica_marca(self, file_path: Path, tsr_path: Path) -> Tuple[bool,
dict]:
        """
        Verifica validità marca temporale

        Returns:
            (valida, dettagli)
        """
        try:
            # Verifica con OpenSSL
            cmd = [
                'openssl', 'ts',
                '-verify',
                '-in', str(tsr_path),
```

```
                '-data', str(file_path),
                '-CAfile', self.config['cert_path'],
            ]

            result = subprocess.run(cmd, capture_output=True, text=True)

            dettagli = {
                'valida': result.returncode == 0,
                'output': result.stdout,
            }

            return result.returncode == 0, dettagli

        except Exception as e:
            logger.error(f"Errore verifica marca: {e}")
            return False, {'errore': str(e)}
```

2.3.3 Conversione PDF/A

```
# conservazione/formatters/pdf_a.py

import subprocess
from pathlib import Path
from typing import Tuple, Optional
import logging

logger = logging.getLogger(__name__)

class PDFAConverter:
    """
    Conversione documenti in PDF/A-2b
    Formato richiesto per la conservazione
    """

    @staticmethod
    def converti_in_pdfa(
        input_path: Path,
        output_path: Optional[Path] = None
    ) -> Tuple[bool, Optional[Path], str]:
        """
        Converte un PDF in PDF/A-2b usando Ghostscript

        Returns:
            (successo, path_output, messaggio)
        """
        if output_path is None:
            output_path = input_path.with_name(
                input_path.stem + '_PDFA.pdf'
            )

        try:
            # Comando Ghostscript per PDF/A-2b
            cmd = [
                'gs',
                '-dPDFA=2',  # PDF/A-2
                '-dBATCH',
                '-dNOPAUSE',
                '-dNOOUTERSAVE',
                '-dUseCIEColor',
                '-sColorConversionStrategy=RGB',
                '-sDEVICE=pdfwrite',
                f'-sOutputFile={output_path}',
                '-dPDFACompatibilityPolicy=1',
                str(input_path),
            ]

            result = subprocess.run(
                cmd,
                capture_output=True,
```

```
                text=True,
                timeout=60
            )

            if result.returncode == 0 and output_path.exists():
                # Verifica conformità PDF/A
                is_valid, msg = PDFAConverter.verifica_pdfa(output_path)
                if is_valid:
                    return True, output_path, "Conversione PDF/A completata"
                else:
                    return False, None, f"PDF/A non conforme: {msg}"
            else:
                return False, None, f"Errore conversione: {result.stderr}"

        except Exception as e:
            logger.error(f"Errore conversione PDF/A: {e}")
            return False, None, str(e)

    @staticmethod
    def verifica_pdfa(pdf_path: Path) -> Tuple[bool, str]:
        """
        Verifica conformità PDF/A con VeraPDF
        """
        try:
            # Usa VeraPDF per validazione
            cmd = [
                'verapdf',
                '--format', 'text',
                str(pdf_path),
            ]

            result = subprocess.run(
                cmd,
                capture_output=True,
                text=True,
                timeout=30
            )

            is_valid = 'PASS' in result.stdout
            return is_valid, result.stdout

        except FileNotFoundError:
            # VeraPDF non installato, usa verifica basic
            logger.warning("VeraPDF non trovato, verifica basic")
            return True, "Verifica basic"
        except Exception as e:
            logger.error(f"Errore verifica PDF/A: {e}")
            return False, str(e)
```

2.4 Processo di Conservazione

2.4.1 Workflow Completo

```
1. SELEZIONE DOCUMENTI
   ↓
2. CREAZIONE PdV (Bozza)
   ↓
3. CONVERSIONE PDF/A
   ↓
4. CALCOLO HASH (SHA-256)
   ↓
5. GENERAZIONE METADATI XML
   ↓
6. CREAZIONE INDICE.XML
   ↓
7. PACCHETTIZZAZIONE (ZIP)
   ↓
8. FIRMA DIGITALE (.p7m)
   ↓
9. MARCATURA TEMPORALE (.tsr)
   ↓
10. RAPPORTO DI VERSAMENTO
   ↓
11. VERSAMENTO IN CONSERVAZIONE
   ↓
12. ACCETTAZIONE
   ↓
13. CONSERVAZIONE PERMANENTE
```

2.4.2 Implementazione Service

```
# conservazione/services/pdv_builder.py

from pathlib import Path
from typing import List, Dict
import zipfile
import hashlib
from datetime import datetime
from django.db import transaction
import logging

logger = logging.getLogger(__name__)

class PdVBuilder:
    """
    Costruttore di Pacchetti di Versamento
    """

    def __init__(self, pdv: PacchettoVersamento):
        self.pdv = pdv
        self.temp_dir = Path('/tmp/conservazione') / str(pdv.uuid)
        self.temp_dir.mkdir(parents=True, exist_ok=True)

    @transaction.atomic
    def aggiungi_documento(self, documento: 'Documento') ->
DocumentoConservazione:
        """
        Aggiunge un documento al PdV
        """
        # 1. Converti in PDF/A
        pdf_a_path = self._converti_pdfa(documento.file.path)

        # 2. Calcola hash
        hash_valore = self._calcola_hash(pdf_a_path)

        # 3. Crea record DocumentoConservazione
        doc_cons = DocumentoConservazione.objects.create(
            pacchetto=self.pdv,
            documento=documento,
            id_documento_pdv=self._genera_id_documento(),
            file_conservazione=pdf_a_path,
            hash_algoritmo='SHA-256',
            hash_valore=hash_valore,
            dimensione_byte=pdf_a_path.stat().st_size,
            formato_originale=documento.file.name.split('.')[-1],
            formato_conservazione='application/pdf',
            oggetto=documento.titolo,
            data_documento=documento.data_documento,
            soggetto_produttore=documento.cliente.ragione_sociale,
            tipo_documento=documento.tipo.descrizione if documento.tipo else
'Documento',
```

```
        )

        logger.info(f"Documento {documento.pk} aggiunto a PdV {self.pdv.codice}")
        return doc_cons

    def chiudi_e_sigilla(self) -> bool:
        """
        Chiude il PdV e applica firma e marca temporale
        """
        try:
            # 1. Genera indice XML
            xml_path = self._genera_indice_xml()

            # 2. Crea ZIP
            zip_path = self._crea_zip()

            # 3. Calcola hash del ZIP
            hash_zip = self._calcola_hash(zip_path)

            # 4. Firma digitale
            firma_service = FirmaDigitaleService(config=settings.FIRMA_CONFIG)
            success, p7m_path, msg = firma_service.firma_file(zip_path)
            if not success:
                raise Exception(f"Errore firma: {msg}")

            # 5. Marca temporale
            marca_service =
MarcaturaTemporaleService(config=settings.MARCA_CONFIG)
            success, tsr_path, info = marca_service.applica_marca(p7m_path)
            if not success:
                raise Exception(f"Errore marca: {info.get('errore')}")

            # 6. Aggiorna PdV
            self.pdv.stato = PacchettoVersamento.Stato.PRONTO
            self.pdv.data_chiusura = datetime.now()
            self.pdv.hash_valore = hash_zip
            self.pdv.file_indice_xml = xml_path
            self.pdv.file_zip = zip_path
            self.pdv.firma_file = p7m_path
            self.pdv.firma_data = datetime.now()
            self.pdv.marca_temporale_file = tsr_path
            self.pdv.marca_temporale_data = info.get('timestamp')
            self.pdv.save()

            logger.info(f"PdV {self.pdv.codice} sigillato con successo")
            return True

        except Exception as e:
            logger.error(f"Errore sigillatura PdV: {e}")
            self.pdv.stato = PacchettoVersamento.Stato.ERRORE
            self.pdv.messaggio_errore = str(e)
```

```
            self.pdv.save()
            return False

    def _genera_indice_xml(self) -> Path:
        """
        Genera l'indice XML del PdV secondo standard AgID
        """
        from .xml_indice import IndiceXMLGenerator

        generator = IndiceXMLGenerator(self.pdv)
        xml_content = generator.genera()

        xml_path = self.temp_dir / 'IndiceConservazione.xml'
        xml_path.write_text(xml_content, encoding='utf-8')

        return xml_path

    def _crea_zip(self) -> Path:
        """
        Crea archivio ZIP del PdV
        """
        zip_path = self.temp_dir / f'{self.pdv.codice}.zip'

        with zipfile.ZipFile(zip_path, 'w', zipfile.ZIP_DEFLATED) as zf:
            # Aggiungi indice
            zf.write(
                self.temp_dir / 'IndiceConservazione.xml',
                'IndiceConservazione.xml'
            )

            # Aggiungi documenti
            for doc in self.pdv.documenti.all():
                zf.write(
                    doc.file_conservazione.path,
                    f'documenti/{doc.id_documento_pdv}.pdf'
                )

        return zip_path

    def _calcola_hash(self, file_path: Path) -> str:
        """Calcola SHA-256 di un file"""
        sha256 = hashlib.sha256()
        with open(file_path, 'rb') as f:
            for chunk in iter(lambda: f.read(4096), b''):
                sha256.update(chunk)
        return sha256.hexdigest()

    def _genera_id_documento(self) -> str:
        """Genera ID univoco per documento nel PdV"""
        count = self.pdv.documenti.count() + 1
        return f"{self.pdv.codice}_DOC_{count:04d}"
```

2.5 Generazione XML Indice

```
# conservazione/formatters/xml_indice.py

from xml.etree.ElementTree import Element, SubElement, tostring
from xml.dom import minidom
from typing import List

class IndiceXMLGenerator:
    """
    Genera l'indice XML del PdV secondo standard AgID
    """

    def __init__(self, pdv: PacchettoVersamento):
        self.pdv = pdv

    def genera(self) -> str:
        """Genera XML completo"""
        root = Element('PacchettoVersamento')
        root.set('xmlns', 'http://www.agid.gov.it/conservazione')
        root.set('version', '1.0')

        # Header
        self._aggiungi_header(root)

        # Descrizione PdV
        self._aggiungi_descrizione_pdv(root)

        # Indice documenti
        self._aggiungi_indice_documenti(root)

        # Pretty print
        xml_str = minidom.parseString(
            tostring(root, encoding='utf-8')
        ).toprettyxml(indent='  ', encoding='utf-8')

        return xml_str.decode('utf-8')

    def _aggiungi_header(self, root: Element):
        """Sezione header"""
        header = SubElement(root, 'Header')

        SubElement(header, 'Identificativo').text = str(self.pdv.uuid)
        SubElement(header, 'Codice').text = self.pdv.codice
        SubElement(header, 'DataCreazione').text = \
            self.pdv.data_creazione.isoformat()
        SubElement(header, 'Produttore').text = self.pdv.produttore
        SubElement(header, 'Responsabile').text = \
            self.pdv.responsabile.get_full_name()

    def _aggiungi_descrizione_pdv(self, root: Element):
        """Descrizione del PdV"""
```

```
        desc = SubElement(root, 'DescrizionePdV')

        SubElement(desc, 'PeriodoInizio').text = \
            self.pdv.periodo_inizio.isoformat()
        SubElement(desc, 'PeriodoFine').text = \
            self.pdv.periodo_fine.isoformat()

        # Hash
        hash_elem = SubElement(desc, 'Hash')
        SubElement(hash_elem, 'Algoritmo').text = self.pdv.hash_algoritmo
        SubElement(hash_elem, 'Valore').text = self.pdv.hash_valore

        # Firma
        if self.pdv.firma_file:
            firma = SubElement(desc, 'FirmaDigitale')
            SubElement(firma, 'Data').text = \
                self.pdv.firma_data.isoformat()
            SubElement(firma, 'Certificato').text = \
                self.pdv.firma_certificato_cn

        # Marca temporale
        if self.pdv.marca_temporale_file:
            marca = SubElement(desc, 'MarcaTemporale')
            SubElement(marca, 'Data').text = \
                self.pdv.marca_temporale_data.isoformat()
            SubElement(marca, 'TSA').text = self.pdv.marca_temporale_tsa

    def _aggiungi_indice_documenti(self, root: Element):
        """Indice dei documenti"""
        indice = SubElement(root, 'IndiceDocumenti')

        for doc in self.pdv.documenti.all():
            doc_elem = SubElement(indice, 'Documento')

            SubElement(doc_elem, 'ID').text = doc.id_documento_pdv
            SubElement(doc_elem, 'Oggetto').text = doc.oggetto
            SubElement(doc_elem, 'Data').text = \
                doc.data_documento.isoformat()
            SubElement(doc_elem, 'TipoDocumento').text = doc.tipo_documento
            SubElement(doc_elem, 'Produttore').text = doc.soggetto_produttore

            if doc.destinatario:
                SubElement(doc_elem, 'Destinatario').text = doc.destinatario

            # File
            file_elem = SubElement(doc_elem, 'File')
            SubElement(file_elem, 'Nome').text = \
                f'{doc.id_documento_pdv}.pdf'
            SubElement(file_elem, 'Formato').text = doc.formato_conservazione
            SubElement(file_elem, 'Dimensione').text = \
                str(doc.dimensione_byte)
```

```
            # Hash
            hash_elem = SubElement(file_elem, 'Hash')
            SubElement(hash_elem, 'Algoritmo').text = doc.hash_algoritmo
            SubElement(hash_elem, 'Valore').text = doc.hash_valore

            # Metadati aggiuntivi
            if doc.metadati.exists():
                metadati = SubElement(doc_elem, 'Metadati')
                for meta in doc.metadati.all():
                    meta_elem = SubElement(metadati, 'Metadato')
                    SubElement(meta_elem, 'Chiave').text = meta.chiave
                    SubElement(meta_elem, 'Valore').text = meta.valore
```

PARTE 3: INTEGRAZIONI NECESSARIE

3.1 Modifiche al Progetto Esistente

3.1.1 Settings.py

```
# mygest/settings.py

# App conservazione
INSTALLED_APPS = [
    # ... esistenti ...
    'conservazione',
]

# Configurazione firma digitale
FIRMA_CONFIG = {
    'tipo': env('FIRMA_TIPO', default='openssl'),
    'cert_path': env('FIRMA_CERT_PATH', default=''),
    'key_path': env('FIRMA_KEY_PATH', default=''),
    'pin': env('FIRMA_PIN', default=''),
}

# Configurazione marca temporale
MARCA_CONFIG = {
    'tsa_url': env('TSA_URL', default=''),
    'username': env('TSA_USERNAME', default=''),
    'password': env('TSA_PASSWORD', default=''),
    'cert_path': env('TSA_CERT_PATH', default=''),
}

# Configurazione conservazione
CONSERVAZIONE = {
    'responsabile_default': env('CONSERVAZIONE_RESPONSABILE', default=''),
    'produttore': env('CONSERVAZIONE_PRODUTTORE', default=''),
    'verifica_periodica_giorni': 90,  # Verifica ogni 90 giorni
    'formato_conservazione': 'PDF/A-2b',
}

# Celery per task asincroni
CELERY_BROKER_URL = env('CELERY_BROKER_URL', default='redis://localhost:6379/0')
CELERY_RESULT_BACKEND = env('CELERY_RESULT_BACKEND', default='redis://localhost:
6379/0')
```

3.1.2 Modello Documento - Aggiunte

```
# documenti/models.py

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

    # Stato conservazione
    in_conservazione = models.BooleanField(
        default=False,
        help_text="Documento inserito in conservazione"
    )

    data_invio_conservazione = models.DateTimeField(
        null=True,
        blank=True,
        help_text="Data invio in conservazione"
    )

    # Relazione con conservazione
    @property
    def versione_conservata(self):
        """Ultima versione conservata del documento"""
        return self.versioni_conservate.filter(
            pacchetto__stato=PacchettoVersamento.Stato.ACCETTATO
        ).first()

    def puo_essere_conservato(self) -> Tuple[bool, str]:
        """
        Verifica se il documento può essere inviato in conservazione

        Returns:
            (può_essere_conservato, motivo)
        """
        if self.in_conservazione:
            return False, "Documento già in conservazione"

        if not self.file:
            return False, "Documento senza file"

        if not self.data_documento:
            return False, "Data documento mancante"

        if not self.cliente:
            return False, "Cliente non specificato"

        # Altri controlli...

        return True, "OK"
```

3.2 Celery Tasks

```
# conservazione/tasks.py

from celery import shared_task
from django.utils import timezone
from datetime import timedelta
import logging

logger = logging.getLogger(__name__)

@shared_task
def verifica_integrita_periodica():
    """
    Task periodico per verifica integrità pacchetti
    Da eseguire ogni 90 giorni come richiesto dalla normativa
    """
    from conservazione.models import PacchettoVersamento, VerificaIntegrita
    from conservazione.services.verifica_service import VerificaService

    # Trova pacchetti da verificare
    giorni_limite = timezone.now() - timedelta(
        days=settings.CONSERVAZIONE['verifica_periodica_giorni']
    )

    pacchetti = PacchettoVersamento.objects.filter(
        stato=PacchettoVersamento.Stato.ACCETTATO,
        verifiche__data_verifica__lt=giorni_limite
    ).distinct()

    logger.info(f"Verifica integrità: {pacchetti.count()} pacchetti da
verificare")

    service = VerificaService()

    for pdv in pacchetti:
        try:
            risultato = service.verifica_pacchetto(pdv)
            logger.info(f"Verifica PdV {pdv.codice}: {risultato['esito']}")
        except Exception as e:
            logger.error(f"Errore verifica PdV {pdv.codice}: {e}")

@shared_task
def chiudi_e_sigilla_pdv(pdv_id: int):
    """Task asincrono per sigillare un PdV"""
    from conservazione.models import PacchettoVersamento
    from conservazione.services.pdv_builder import PdVBuilder

    try:
        pdv = PacchettoVersamento.objects.get(pk=pdv_id)
        builder = PdVBuilder(pdv)
        success = builder.chiudi_e_sigilla()
```

```
        if success:
            logger.info(f"PdV {pdv.codice} sigillato con successo")
        else:
            logger.error(f"Errore sigillatura PdV {pdv.codice}")

        return success

    except Exception as e:
        logger.error(f"Errore task sigillatura: {e}")
        return False
```

3.3 API REST

```
# conservazione/views.py

from rest_framework import viewsets, status
from rest_framework.decorators import action
from rest_framework.response import Response
from rest_framework.permissions import IsAuthenticated
from django.shortcuts import get_object_or_404

class PacchettoVersamentoViewSet(viewsets.ModelViewSet):
    """
    API per gestione Pacchetti di Versamento
    """
    permission_classes = [IsAuthenticated]

    @action(detail=False, methods=['post'])
    def crea_da_documenti(self, request):
        """
        Crea un nuovo PdV da lista documenti

        POST /api/conservazione/pacchetti/crea_da_documenti/
        {
            "documenti_ids": [1, 2, 3],
            "periodo_inizio": "2025-01-01",
            "periodo_fine": "2025-03-31",
            "descrizione": "Primo trimestre 2025"
        }
        """
        from conservazione.services.pdv_builder import PdVBuilder

        documenti_ids = request.data.get('documenti_ids', [])

        # Crea PdV
        pdv = PacchettoVersamento.objects.create(
            codice=self._genera_codice_pdv(),
            responsabile=request.user,
            produttore=settings.CONSERVAZIONE['produttore'],
            periodo_inizio=request.data['periodo_inizio'],
            periodo_fine=request.data['periodo_fine'],
            stato=PacchettoVersamento.Stato.IN_PREPARAZIONE,
        )

        # Aggiungi documenti
        builder = PdVBuilder(pdv)
        for doc_id in documenti_ids:
            documento = Documento.objects.get(pk=doc_id)
            puo, motivo = documento.puo_essere_conservato()
            if puo:
                builder.aggiungi_documento(documento)
            else:
                return Response(
```

```
                    {'error': f'Documento {doc_id}: {motivo}'},
                    status=status.HTTP_400_BAD_REQUEST
                )

        return Response({
            'id': pdv.id,
            'codice': pdv.codice,
            'num_documenti': pdv.documenti.count()
        })

    @action(detail=True, methods=['post'])
    def sigilla(self, request, pk=None):
        """
        Sigilla il PdV (firma + marca temporale)

        POST /api/conservazione/pacchetti/{id}/sigilla/
        """
        from conservazione.tasks import chiudi_e_sigilla_pdv

        pdv = self.get_object()

        if pdv.stato != PacchettoVersamento.Stato.IN_PREPARAZIONE:
            return Response(
                {'error': 'PdV non in stato corretto'},
                status=status.HTTP_400_BAD_REQUEST
            )

        # Avvia task asincrono
        task = chiudi_e_sigilla_pdv.delay(pdv.id)

        return Response({
            'message': 'Sigillatura avviata',
            'task_id': task.id
        })

    @action(detail=True, methods=['get'])
    def verifica_integrita(self, request, pk=None):
        """
        Verifica integrità del PdV

        GET /api/conservazione/pacchetti/{id}/verifica_integrita/
        """
        from conservazione.services.verifica_service import VerificaService

        pdv = self.get_object()
        service = VerificaService()
        risultato = service.verifica_pacchetto(pdv)

        return Response(risultato)
```

PARTE 4: DEPLOYMENT E MANUTENZIONE

4.1 Dipendenze Aggiuntive

```
# requirements.txt - Aggiunte per conservazione

# PDF/A
ghostscript==10.0.0

# Firma digitale e crittografia
cryptography==41.0.0
pyOpenSSL==23.2.0

# Marca temporale
python-tsp==0.2.0

# XML
lxml==4.9.3

# Celery per task asincroni
celery==5.3.4
redis==5.0.1

# API
djangorestframework==3.14.0
```

4.2 Installazione Strumenti

```
# Ghostscript per PDF/A
sudo apt install ghostscript

# VeraPDF per validazione PDF/A
wget https://software.verapdf.org/releases/verapdf-installer.zip
unzip verapdf-installer.zip
sudo ./verapdf-install

# OpenSSL (già presente di solito)
sudo apt install openssl
```

4.3 Configurazione Produzione

```
# .env.production - Aggiunte

# Firma digitale
FIRMA_TIPO=aruba
FIRMA_CERT_PATH=/srv/mygest/certs/firma.crt
FIRMA_KEY_PATH=/srv/mygest/certs/firma.key
FIRMA_PIN=your_pin_here

# Marca temporale
TSA_URL=https://tsa.example.com/tsa
TSA_USERNAME=your_username
TSA_PASSWORD=your_password
TSA_CERT_PATH=/srv/mygest/certs/tsa_ca.pem

# Conservazione
CONSERVAZIONE_RESPONSABILE=nome.cognome@azienda.it
CONSERVAZIONE_PRODUTTORE=Azienda S.r.l.

# Celery
CELERY_BROKER_URL=redis://localhost:6379/0
CELERY_RESULT_BACKEND=redis://localhost:6379/0
```

4.4 Celery Worker

```
# /etc/systemd/system/mygest-celery.service

[Unit]
Description=MyGest Celery Worker
After=network.target redis.service

[Service]
Type=forking
User=mygest
Group=mygest
WorkingDirectory=/srv/mygest/app
Environment="PATH=/srv/mygest/venv/bin"
ExecStart=/srv/mygest/venv/bin/celery -A mygest worker \
    --loglevel=info \
    --logfile=/srv/mygest/logs/celery.log
Restart=on-failure

[Install]
WantedBy=multi-user.target
```

```
# Attiva servizio
sudo systemctl enable mygest-celery
sudo systemctl start mygest-celery
```

4.5 Celery Beat (Task Periodici)

```
# /etc/systemd/system/mygest-celerybeat.service

[Unit]
Description=MyGest Celery Beat
After=network.target redis.service

[Service]
Type=simple
User=mygest
Group=mygest
WorkingDirectory=/srv/mygest/app
Environment="PATH=/srv/mygest/venv/bin"
ExecStart=/srv/mygest/venv/bin/celery -A mygest beat \
    --loglevel=info \
    --logfile=/srv/mygest/logs/celerybeat.log
Restart=on-failure

[Install]
WantedBy=multi-user.target
```

PARTE 5: CONFORMITÀ E CERTIFICAZIONI

5.1 Checklist Conformità AgID

- [ ] **Identificazione univoca**: Ogni documento ha UUID
- [ ] **Integrità**: Hash SHA-256 di ogni documento
- [ ] **Immodificabilità**: Firma digitale + marca temporale
- [ ] **Autenticità**: Certificato digitale qualificato
- [ ] **Leggibilità**: Formato PDF/A-2b
- [ ] **Metadati obbligatori**: Presenti e validati
- [ ] **Indice conservazione**: XML conforme
- [ ] **Rapporto versamento**: Generato per ogni PdV
- [ ] **Verifiche periodiche**: Ogni 90 giorni
- [ ] **Responsabile conservazione**: Nominato e registrato
- [ ] **Manuale conservazione**: Documento le procedure
- [ ] **Log eventi**: Tracciamento completo
- [ ] **Piano disaster recovery**: Backup e restore

5.2 Audit Trail

```
# conservazione/models.py - Da aggiungere

class EventoConservazione(models.Model):
    """
    Log eventi per audit trail
    Richiesto dalla normativa
    """

    class TipoEvento(models.TextChoices):
        CREAZIONE_PDV = 'creazione_pdv', 'Creazione PdV'
        AGGIUNTA_DOCUMENTO = 'aggiunta_doc', 'Aggiunta Documento'
        SIGILLATURA = 'sigillatura', 'Sigillatura'
        VERSAMENTO = 'versamento', 'Versamento'
        ACCETTAZIONE = 'accettazione', 'Accettazione'
        VERIFICA = 'verifica', 'Verifica Integrità'
        ESPORTAZIONE = 'esportazione', 'Esportazione'
        ERRORE = 'errore', 'Errore'

    timestamp = models.DateTimeField(auto_now_add=True, db_index=True)

    tipo = models.CharField(
        max_length=20,
        choices=TipoEvento.choices
    )

    pacchetto = models.ForeignKey(
        PacchettoVersamento,
        on_delete=models.CASCADE,
        null=True,
        blank=True,
        related_name='eventi'
    )

    utente = models.ForeignKey(
        User,
        on_delete=models.SET_NULL,
        null=True
    )

    ip_address = models.GenericIPAddressField(null=True)

    descrizione = models.TextField()

    dettagli = models.JSONField(default=dict)

    class Meta:
        db_table = 'conservazione_evento'
        ordering = ['-timestamp']
        indexes = [
            models.Index(fields=['tipo', 'timestamp']),
```

```
            models.Index(fields=['pacchetto', 'timestamp']),
        ]
```

CONCLUSIONI

Riepilogo Implementazione

Componenti Obbligatori

1. **App** `conservazione`: Nuova applicazione Django dedicata
2. **Modelli**: PacchettoVersamento, DocumentoConservazione, LottoConservazione, VerificaIntegrita
3. **Servizi**: Firma digitale, marca temporale, conversione PDF/A, hash
4. **Workflow**: Processo completo da selezione a conservazione
5. **API REST**: Interfaccia per frontend
6. **Celery**: Task asincroni e verifiche periodiche
7. **Audit**: Log eventi completo

Costi e Fornitori

Firma Digitale: - ArubaPEC: ~€50/anno - InfoCert: ~€40/anno - Namirial: ~€45/anno

Marca Temporale: - ArubaPEC: ~€0.10 per marca - InfoCert: ~€0.08 per marca - Pacchetti da 1000+ marche disponibili

Conservatore Accreditato (opzionale): - ArchiWorld: da €500/anno - Namirial: da €600/anno - InfoCert: da €700/anno

Timeline Sviluppo

- **Fase 1** (2 settimane): Modelli e database
- **Fase 2** (2 settimane): Servizi base (hash, PDF/A)
- **Fase 3** (1 settimana): Integrazione firma e marca
- **Fase 4** (1 settimana): Workflow completo
- **Fase 5** (1 settimana): API e frontend
- **Fase 6** (1 settimana): Testing e validazione
- **TOTALE**: ~8 settimane

Manutenzione Continua

- Verifiche integrità automatiche (ogni 90 giorni)
- Aggiornamento certificati firma
- Backup pacchetti conservati
- Audit periodici conformità
- Aggiornamento normativa

Documento creato il: 17 Novembre 2025
Versione: 1.0
Autore: Sistema MyGest - Analisi Conservazione Digitale

MyGest - Sistema di Gestione Documentale

Conservazione Digitale a Norma secondo normativa AgID