Analisi Progetto MyGest

Data Analisi:	17/11/2025 11:38
Progetto:	MyGest - Sistema di Gestione Documentale
Repository:	sandro6917/mygest
Branch:	main
Framework:	Django 4.2.16 + PostgreSQL

Executive Summary

Il progetto MyGest è un'applicazione Django modulare per la gestione documentale, protocollo, pratiche e scadenze. L'architettura è ben strutturata con 12 app Django separate. **L'applicazione è già in produzione su VPS Hostinger** con configurazione di sicurezza adeguata (variabili d'ambiente, HTTPS, etc). L'ambiente analizzato è quello di sviluppo. Il codice è di buona qualità ma può beneficiare di miglioramenti in testing, monitoring e performance optimization.

Categoria	Stato	Priorità
Sicurezza Prod	■ CONFIGURATO	N/A
Testing	■ MEDIO	ALTA
Performance	■ MEDIO	MEDIA
Architettura	■ BUONO	MEDIA
Monitoring	■ BASE	ALTA
Documentazione	■ BUONO	BASSA

■■ Nota Importante

Ambiente Analizzato: Questa analisi si basa sul codice dell'ambiente di sviluppo. L'applicazione è già **in produzione su VPS Hostinger** con configurazione di sicurezza adeguata (variabili d'ambiente tramite .env, DEBUG=False, ALLOWED_HOSTS configurato, HTTPS attivo). Le considerazioni di sicurezza riportate sono relative all'ambiente di sviluppo locale.

■ Aree di Miglioramento

1. Sicurezza - Best Practices Ambiente Dev

• **Separazione settings:** Creare settings/base.py, settings/dev.py, settings/prod.py per gestione più pulita

• **File .env nel repository:** Usare .env.example come template senza valori sensibili

• **csrf_exempt** in whatsapp/views.py - Valutare validazione signature webhook invece di disabilitare CSRF

• **Rate limiting:** Aggiungere django-ratelimit per proteggere login e API

• **Security headers:** Implementare django-csp per Content Security Policy

• **Audit log:** Tracciare accessi e modifiche critiche (protocollo, documenti)

2. Database

• Password database hardcoded in settings.py (linea 106): 'ScegliUnaPasswordSicura'

• Uso estensivo di on_delete=models.CASCADE in 20+ modelli - Rischio eliminazioni a cascata non volute

• Mancano indici su campi frequently queried (es. stato, data_creazione in vari modelli)

• Nessun backup automatico configurato (solo script manuale backup_mygest.sh)

• Connection pooling non configurato per ambienti ad alto carico

3. Gestione File e Storage

• Storage misto tra MEDIA_ROOT e NAS (NASPathStorage) può creare confusione

• Percorso temporaneo tmp/ in documento_upload_to non viene pulito automaticamente

• Path hardcoded: /mnt/archivio, /home/sandro/documenti - Non portabile

• Nessuna validazione dimensione massima file upload

• Mancano controlli antivirus sui file caricati

■ Miglioramenti Consigliati

1. Architettura & Codice

Settings Management:

• Separare settings per ambiente (base.py, dev.py, prod.py, test.py)

• Usare django-environ per TUTTE le variabili sensibili

• Implementare settings validation al startup

• Creare template .env.example senza valori sensibili

Performance:

• Implementare caching (Redis/Memcached) - attualmente assente

• Aggiungere Django Debug Toolbar in sviluppo per analisi query

• Ottimizzare query N+1 - select_related/prefetch_related presente ma non ovunque

• Implementare paginazione consistente su tutte le liste

• Database connection pooling (pgBouncer in produzione)

• Compressione statica (django-compressor, Brotli/Gzip)

Testing:

• Coverage attuale molto bassa - aumentare al 70%+ minimo

• Mancano test di integrazione tra app

• Nessun test per API GraphQL/REST

• Aggiungere test per validatori CF/PIVA

• Implementare CI/CD pipeline (GitHub Actions)

• Test di carico e stress testing

2. Code Quality

• Type hints più consistenti (mypy installato ma poco utilizzato)

• Docstrings mancanti nella maggior parte delle funzioni

• Logging strutturato (attualmente solo protocollazione.log)

• Exception handling più robusto e specifico

• Validazioni business logic centralizzate nei modelli

• Codice duplicato in vari admin.py e forms.py

3. Gestione Dipendenze

Problema	Descrizione	Azione
Duplicati	psycopg2-binary appare 2 volte	Rimuovere duplicato
Conflitto	psycopg e psycopg-binary insieme	Usare solo psycopg[binary]
Versioni	Versioni pinned ma vecchie	Aggiornare Django 4.2.16 → 5.1+
Sicurezza	Verificare vulnerabilità note	pip-audit o safety check

4. Task Asincroni

File **comunicazioni/tasks.py** presente ma **Celery/Redis NON configurato**. Questo è un problema significativo perché:

• Email import è potenzialmente bloccante (IMAP sync)

• Invio massivo comunicazioni blocca l'applicazione

• Generazione report PDF pesanti rallenta le richieste

• Sincronizzazione Google Calendar sincrona

• Nessuna gestione retry per operazioni fallite

5. API & Monitoring

API:

• GraphQL configurato ma schema minimale (solo in mygest/schema.py)

• REST API solo per comunicazioni (comunicazioni/api/)

• Mancano endpoint per: documenti, pratiche, fascicoli, protocollo

• Autenticazione API limitata a Session/Basic - mancano token JWT

• Nessuna throttling/rate limiting

• Documentazione API assente (considerare drf-spectacular)

Monitoring:

• Sentry o simile per error tracking

• Application Performance Monitoring (APM) - es. New Relic, DataDog

• Structured logging (JSON logs per parsing automatico)

• Health check endpoints (/health/, /ready/)

• Metrics/Prometheus per metriche applicative

• Uptime monitoring e alerting

■ Punti di Forza

Architettura modulare: 12 app Django ben separate: anagrafiche, documenti, pratiche, fascicoli, protocollo, scadenze, comunicazioni, whatsapp, archivio_fisico, stampe, titolario

Modelli Django standard: Uso corretto di ForeignKey, ManyToMany, scelte, validators personalizzati

Validatori robusti: Validazione CF/PIVA con checksum completo secondo normativa italiana

Documentazione presente: 4 guide in docs/: guida_messa_in_produzione, guida_principianti, guida_scadenze

Script di deploy: scripts/deploy.sh automatizza: pull, pip install, migrate, collectstatic, restart

Permessi gestiti: @login_required presente su tutte le view sensibili

Transazioni database: Uso corretto di @transaction.atomic per operazioni critiche

Query optimization: select_related/prefetch_related presenti in views e admin

Migrazioni strutturate: 148 file di migrazione, schema database ben evolvibile

Integrazione esterna: WhatsApp Cloud API, Google Calendar, IMAP/SMTP configurabile

Multi-DB test: Configurazione automatica SQLite per tests (settings.py linea 137)

Gestione protocollo: Sistema completo di protocollazione IN/OUT con numerazione automatica

■ Roadmap Evolutiva

Fase 1: Qualità e Monitoring (1-2 mesi) - PRIORITÀ ALTA

Area	Attività	Effort	Impatto
Settings	Separare settings per ambiente Template .env.example Audit logging	3gg	ALTO
Monitoring	Integrare Sentry Health check endpoints Log aggregation (ELK/Loki) Uptime monitoring	4gg	ALTO
Testing	Coverage base 70%+ Test per protocollo/documenti CI/CD pipeline (GitHub Actions) Test di integrazione	8gg	ALTO
Database	Review CASCADE deletes Aggiungere indici mancanti Verificare backup automatici Query optimization	4gg	ALTO

Fase 2: Performance (2-3 mesi) - PRIORITÀ MEDIA

Area	Attività	Effort	Impatto
Caching	Redis per sessioni e cache Cache query frequenti Template fragment caching	4gg	ALTO
Async Tasks	Setup Celery + Redis Queue per email e notifiche Background jobs import/export	6gg	ALTO
DB Optimization	Query analyzer e ottimizzazione Connection pooling (pgBouncer) Partitioning tabelle grandi	5gg	MEDIO
Frontend	Asset optimization (compress) Lazy loading immagini Webpack/Vite per bundle	4gg	MEDIO

Fase 3: Feature Enhancement (3-6 mesi) - PRIORITÀ BASSA

- **API Completa:** REST API per tutte le risorse, versioning, OpenAPI/Swagger docs, JWT auth
- **Frontend Moderno:** SPA con React/Vue, Progressive Web App, real-time updates (WebSockets)
- **Integrazioni Avanzate:** Firma digitale, PEC avanzata, calendar sync bidirezionale, export contabilità
- **Business Intelligence:** Dashboard analytics, report customizzabili, export strutturati
- **Multi-tenancy:** Supporto multi-cliente, permessi granulari RBAC, isolamento dati
- **Mobile App:** App nativa iOS/Android o PWA avanzata per gestione mobile

■ Checklist Immediata (Prossime 2 Settimane)

Prior.	Task	Effort	Status
■	Separare settings: base.py, dev.py, prod.py	2h	■
■	Creare .env.example (senza valori sensibili)	30min	■
■	Aggiungere Sentry per error tracking	1h	■
■	Implementare health check endpoint	1h	■
■	Verificare backup automatici DB su VPS	1h	■
■	Creare test per funzioni critiche (protocollo)	4h	■
■	Aggiungere test coverage measurement (coverage.py)	1h	■
■	Documentare API GraphQL/REST esistenti	2h	■
■	Review CASCADE deletes nei modelli	3h	■
■	Aggiungere indici database mancanti	2h	■
■	Implementare audit log per azioni critiche	3h	■
■	Setup Django Debug Toolbar in dev	30min	■
■	Configurare Celery + Redis per task async	4h	■
■	Aggiungere rate limiting (django-ratelimit)	2h	■
■	Implementare monitoring uptime (UptimeRobot)	30min	■

Legenda: ■ Critico | ■ Importante | ■ Consigliato

■ Metriche e KPI Raccomandati

Metriche da tracciare per monitorare la salute dell'applicazione:

Categoria	Metrica	Soglia Target	Tool Consigliato
Performance	Tempo risposta medio endpoint	< 200ms	APM / Django Debug Toolbar
Performance	Query database lente	< 100ms	django-silk / pgBadger
Performance	Tempo generazione PDF	< 5s	Custom logging
Affidabilità	Error rate	< 0.1%	Sentry
Affidabilità	Uptime	> 99.5%	UptimeRobot / Pingdom
Affidabilità	Success rate email	> 98%	Custom metric
Storage	Utilizzo disco NAS	< 80%	System monitoring
Storage	File temp non puliti	0	Cron job + log
Database	Dimensione database	Monitoraggio crescita	PostgreSQL stats
Database	Connection pool usage	< 80%	pgBouncer stats
Sicurezza	Failed login attempts	Alert > 10/min	Django signals + Sentry
Sicurezza	Accessi non autorizzati	0	Custom middleware
Business	Documenti protocollati/giorno	Trend	Custom analytics
Business	Pratiche aperte/chiuse	Ratio	Custom analytics
Business	Tempo medio chiusura pratica	Report mensile	Custom analytics

Conclusioni

Il progetto **MyGest** presenta un'architettura solida e ben strutturata, con moduli Django chiaramente separati e un buon utilizzo delle funzionalità del framework. La qualità del codice è generalmente buona, con validatori personalizzati, gestione transazioni e ottimizzazioni query già implementate.

Punti di attenzione per lo sviluppo continuo:

- Testing: aumentare coverage e aggiungere test di integrazione
- Monitoring: implementare Sentry e metriche applicative
- Performance: valutare caching e task asincroni per operazioni pesanti
- Manutenibilità: separare settings per ambiente e migliorare documentazione API

L'applicazione è già operativa in produzione con successo. Gli sforzi futuri dovrebbero concentrarsi su qualità del codice, testing, monitoring e performance optimization. L'implementazione della Fase 1 (Qualità e Monitoring) porterebbe significativi benefici in termini di manutenibilità e visibilità sullo stato dell'applicazione.

Raccomandazione finale: Concentrarsi sulla checklist immediata e sulla Fase 1 della roadmap, con particolare enfasi su testing, monitoring e separazione degli ambienti. Il ROI di queste attività è alto in termini di stabilità operativa e manutenibilità futura.
