UNIVERSITÀ POLITECNICA DELLE MARCHE
FACOLTÀ DI INGEGNERIA
Dipartimento di Ingegneria dell'Informazione
Corso di Laurea Triennale in Ingegneria Informatica e dell'Automazione

TESINA PER IL CORSO DI INGEGNERIA DEL SOFTWARE
PROF. DOMENICO URSINO

Software gestionale per un ingegnere freelancer

Autori

Emanuele Sacco

ANNO ACCADEMICO 2025-2026

Indice

Parte I

Il Software Gestionale

CHAPTER 1

L'Applicazione

1.1 Introduzione

Il progetto qui presentato consiste nella progettazione e nello sviluppo di un software gestionale completo, destinato a supportare l'attività di un ingegnere freelance o di un libero professionista in ambito tecnico.

L'analisi preliminare del contesto operativo di questa figura professionale ha evidenziato una necessità critica: la centralizzazione della gestione. Attualmente, le operazioni quotidiane — dalla gestione dei clienti al tracciamento delle ore, dalla preventivazione alla fatturazione e al monitoraggio delle scadenze — sono spesso frammentate tra molteplici strumenti non integrati, come fogli di calcolo, editor di testo e calendari digitali. Questa frammentazione genera inefficienze, aumenta il rischio di errori manuali e rende complessa una visione d'insieme sull'andamento dell'attività.

La necessità emersa è stata quindi quella di disporre di un software di supporto unificato, capace di facilitare e automatizzare queste operazioni.

L'obiettivo di questo progetto è fornire un'unica applicazione desktop che integri tutte le funzionalità vitali per il professionista. Il sistema sviluppato centralizza l'anagrafica dei clienti e dei fornitori, gestisce il ciclo di vita completo dei progetti (dalla definizione delle fasi al tracciamento dettagliato delle ore lavorate) e automatizza il flusso finanziario. Quest'ultimo include la creazione di preventivi, la loro conversione in fatture, la registrazione dei pagamenti in una prima nota dedicata e la gestione di un inventario di magazzino.

Inoltre, il software agisce come un assistente proattivo, popolando automaticamente un calendario con le scadenze operative (derivanti da progetti e fatture) e fornendo un cruscotto analitico per il monitoraggio della produttività e delle finanze, con la possibilità di esportare report per analisi esterne o per la consultazione del commercialista.

Parte II

Progettazione

CHAPTER 2

Progettazione

2.1 Glossario dei termini

Il glossario di progetto è il dizionario che contiene i principali termini coinvolti nella logica di business e nell'architettura del software, comprensivi della loro descrizione e del contesto d'uso.

Tabella 2.1: Glossario dei termini di progetto

Termine	Significato	Tipo	Sinonimi
Backend	Il "motore" dell'applicazione. È il pacchetto software (`backend/`) che contiene tutta la logica di business, i calcoli, la validazione e la gestione dei dati. Opera in modo indipendente dall'interfaccia utente.	Architetturale	Model
Frontend	L'interfaccia grafica (GUI) con cui l'utente interagisce. È il pacchetto (`frontend/`) costruito con CustomTkinter. Ha il compito di mostrare i dati e raccogliere l'input, comunicando con il Backend.	Architetturale	GUI, View/Controller
Persistenza	Il meccanismo di salvataggio permanente dei dati. In questo progetto, è implementata tramite la serializzazione di oggetti Python su file (`.pkl`), gestita centralmente dal modulo `persistence.py` .	Architetturale	Salvataggio
`pickle`	La libreria Python standard utilizzata per la persistenza. Serializza (converte) oggetti Python complessi (come dizionari e liste) in un flusso di byte che può essere scritto su file.	Tecnico	Serializzazione

(continua...)

(Segue) Glossario dei termini di progetto

Termine	Significato	Tipo	Sinonimi
`Decimal`	Un tipo di dato numerico di Python (dalla libreria `decimal`) usato per tutti i calcoli finanziari. A differenza dei `float` (numeri in virgola mobile), garantisce la precisione assoluta ed elimina gli errori di arrotondamento.	Tecnico	Precisione finanziaria
Rubrica	Il modulo di business che gestisce l'anagrafica centralizzata.	Business	Anagrafica
Contatto	Una singola entità nella Rubrica. Viene tipizzata come "Cliente" (soggetto a cui si emettono fatture) o "Fornitore" (soggetto da cui si ricevono costi).	Business	Cliente, Fornitore
Progetto	L'entità centrale che rappresenta una commessa di lavoro. Aggrega un cliente, una tariffa oraria, e le liste di Fasi, Attività e File associati.	Business	Commessa
Attività	Una singola registrazione nel modulo di time-tracking, associata a un progetto. Contiene la data, le ore lavorate e un flag booleano per definire se è "fatturabile" o meno.	Business	Time-Tracking
Preventivo	Il documento commerciale (offerta) non fiscale generato dal modulo Documenti prima dell'inizio di un lavoro. Può essere convertito in fattura.	Business	Offerta, Quote
Fattura	Il documento fiscale (attivo) generato dal modulo Documenti per richiedere un pagamento a seguito di una prestazione.	Business	Documento fiscale
Prima Nota	Il registro contabile (gestito dal modulo `ledger.py`) di tutti i movimenti finanziari, sia in "Entrata" (incassi) che in "Uscita" (spese).	Business	Cassa, Ledger
Riconciliazione	Il processo (implementato nel backend) che collega un movimento di "Entrata" della Prima Nota a una "Fattura" emessa, aggiornando automaticamente lo stato di quest'ultima in "Pagato".	Business	Registrazione incasso
"Quote-to-Cash"	Termine di business che descrive l'intero flusso di lavoro automatizzato dal software: dalla creazione di un'offerta (Preventivo) fino all'incasso del pagamento (Cassa).	Business	Ciclo attivo

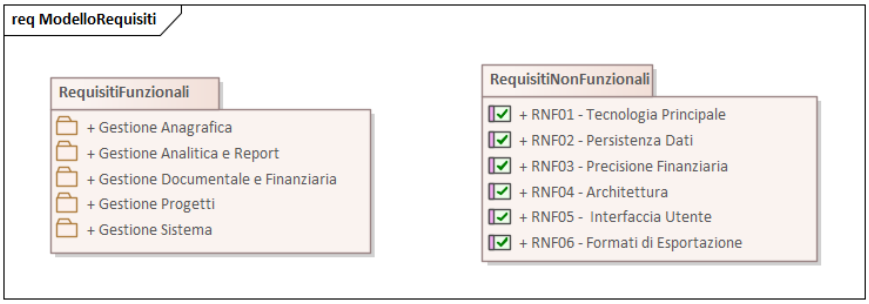
(continua...)

(Segue) Glossario dei termini di progetto

Termine	Significato	Tipo	Sinonimi
`pandas`	Libreria Python usata nei moduli di reportistica (`dashboard.py` , `time_reports.py`). Permette di caricare i dati in strutture (DataFrame) per eseguire aggregazioni e analisi complesse.	Tecnico	Analisi dati
`weasyprint`	Libreria Python usata per generare i file PDF. Converte un template HTML (popolato con i dati della fattura/preventivo) in un documento PDF finale.	Tecnico	Esportazione PDF
SMTP	(Simple Mail Transfer Protocol) Il protocollo standard per l'invio di email. Utilizzato dal modulo `calendar.py` (tramite `smtplib`) per inviare le notifiche delle scadenze.	Tecnico	Invio Email
KPI	(Key Performance Indicator) Un indicatore chiave di prestazione. Nel progetto, sono i valori aggregati (es. "Totale da Incassare", "Margine YTD") mostrati nel Dashboard.	Business	Metriche, Statistiche

2.2 Analisi dei Requisiti

Questo capitolo definisce formalmente le specifiche del software gestionale. La prima sezione elenca gli **Attori** che interagiscono con il sistema. La seconda sezione dettaglia i **Requisiti Funzionali** (ciò che il sistema deve fare) e i **Requisiti Non Funzionali** (i vincoli di qualità e implementazione). La terza sezione descrive i **Casi d'Uso** principali, illustrando le sequenze di eventi per le operazioni chiave.



2.2.1 Attori

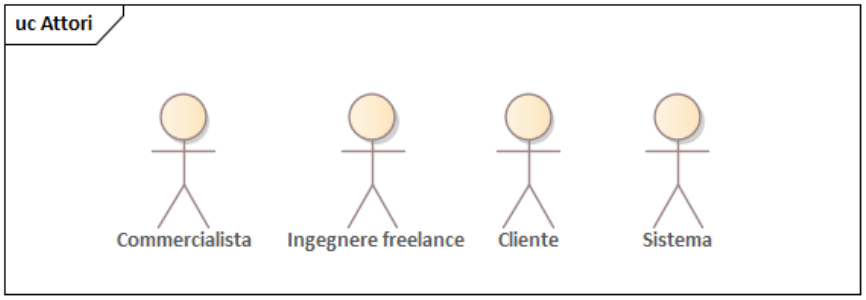
Gli attori rappresentano le entità, umane o sistemiche, che interagiscono con il software.

Ingegnere Freelance (Attore Primario) L'utente principale del sistema. È colui che esegue tutte le operazioni di gestione: inserisce contatti, crea progetti, traccia le ore, emette documenti fiscali, registra pagamenti e consulta i report.

Cliente (Attore Secondario) Un'entità esterna al sistema. Non interagisce direttamente con il software, ma è il destinatario finale di output generati dal sistema (es. Preventivi e Fatture in PDF, email).

Commercialista (Attore Secondario) Un'entità esterna che riceve gli export contabili (es. file Excel/CSV della prima nota) generati dal sistema per l'elaborazione fiscale.

Sistema (Attore Primario) Rappresenta il software stesso quando esegue operazioni automatiche o proattive, come la generazione di scadenze a calendario o l'invio di notifiche.



2.2.2 Package dei Requisiti

I requisiti sono raggruppati logicamente nei seguenti package funzionali:

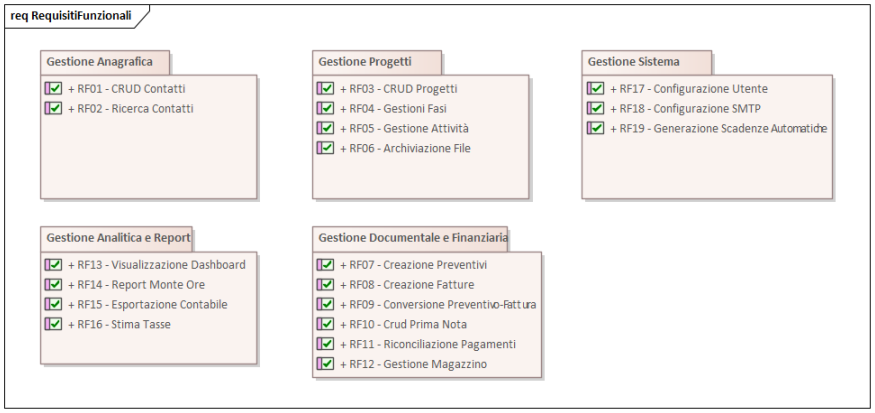
Gestione Anagrafica Include la gestione di clienti e fornitori.

Gestione Progetti Include la creazione di commesse, la gestione delle fasi e il tracciamento delle attività (time-tracking).

Gestione Documentale e Finanziaria Include l'intero ciclo di fatturazione (preventivi, fatture) e la gestione della cassa (prima nota).

Gestione Analitica e Report Include il cruscotto, i report grafici e le esportazioni.

Gestione Sistema Include la configurazione, le notifiche e la generazione automatica di eventi.

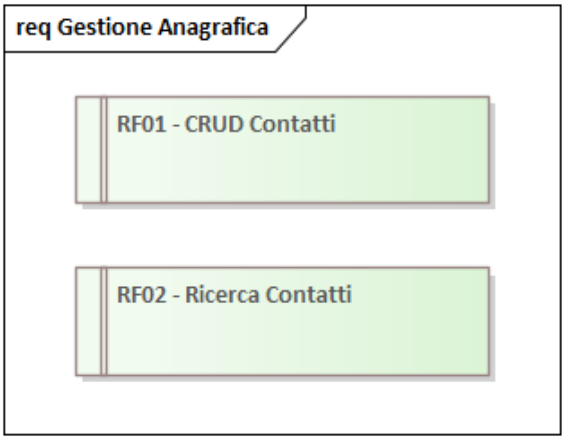


2.2.3 Requisiti Funzionali (RF)

Gestione Anagrafica

RF01 - CRUD Contatti Il sistema deve permettere la creazione, lettura, modifica ed eliminazione dei contatti (clienti e fornitori).

RF02 - Ricerca Contatti Il sistema deve fornire una funzione di ricerca per filtrare rapidamente la rubrica.



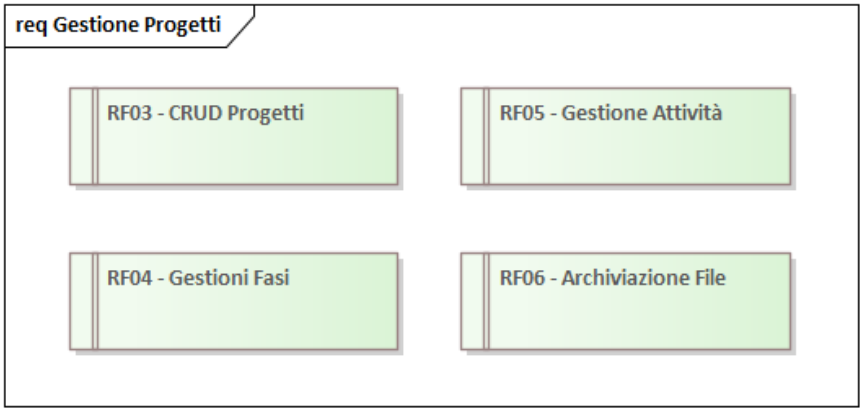
Gestione Progetti

RF03 - CRUD Progetti Il sistema deve permettere la creazione, lettura, modifica ed eliminazione di un progetto, associandolo a un cliente dalla rubrica.

RF04 - Gestione Fasi Il sistema deve permettere di aggiungere, modificare, eliminare e marcare come "completata" una fase del progetto.

RF05 - Gestione Attività (Time-Tracking) Il sistema deve permettere di registrare le attività lavorate (con data, ore, descrizione) per un progetto, specificando se l'attività è fatturabile o meno.

RF06 - Archiviazione File Il sistema deve consentire di allegare (copiandoli in una cartella dedicata) file e documenti a un progetto.



Gestione Documentale e Finanziaria

RF07 - Creazione Preventivi Il sistema deve permettere la creazione e l'esportazione in PDF di preventivi.

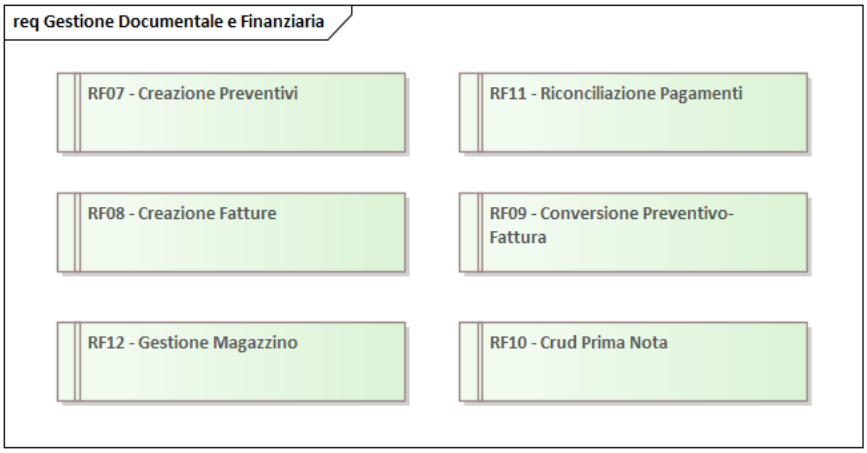
RF08 - Creazione Fatture Il sistema deve permettere la creazione manuale e l'esportazione in PDF di fatture, gestendo la numerazione progressiva e annuale.

RF09 - Conversione Preventivo-Fattura Il sistema deve permettere di convertire un preventivo in fattura, ereditando i dati del cliente e le voci.

RF10 - CRUD Prima Nota Il sistema deve permettere la registrazione manuale di entrate e uscite (costi) nel registro di cassa.

RF11 - Riconciliazione Pagamenti Il sistema deve permettere di registrare un incasso associandolo a una fattura specifica, aggiornandone lo stato in "Pagato".

RF12 - Gestione Magazzino Il sistema deve permettere una gestione base (CRUD e modifica stock) di un inventario articoli.



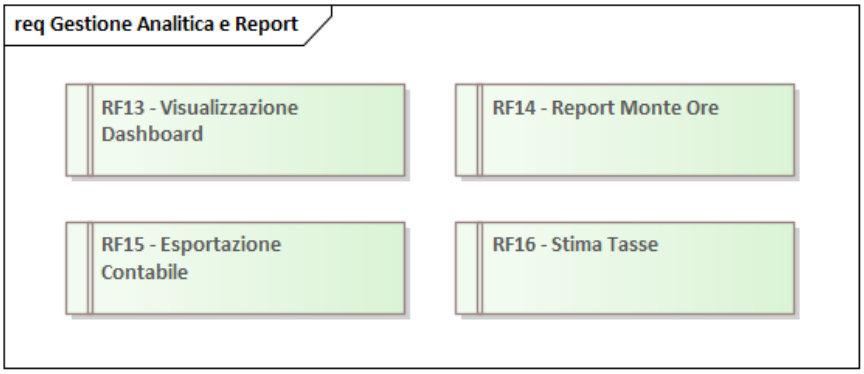
Gestione Analitica e Report

RF13 - Visualizzazione Dashboard Il sistema deve mostrare un cruscotto con i principali KPI (es. fatturato YTD, totale da incassare, progetti attivi).

RF14 - Report Monte Ore Il sistema deve generare un report (testuale e grafico) che mostri la distribuzione delle ore lavorate (fatturabili vs. non) per periodo.

RF15 - Esportazione Contabile Il sistema deve permettere di esportare il registro di prima nota in formato Excel (`.xlsx`) o CSV per il commercialista.

RF16 - Stima Tasse Il sistema deve fornire una stima (basata su aliquote configurabili) dell'IVA trimestrale e dei contributi YTD.

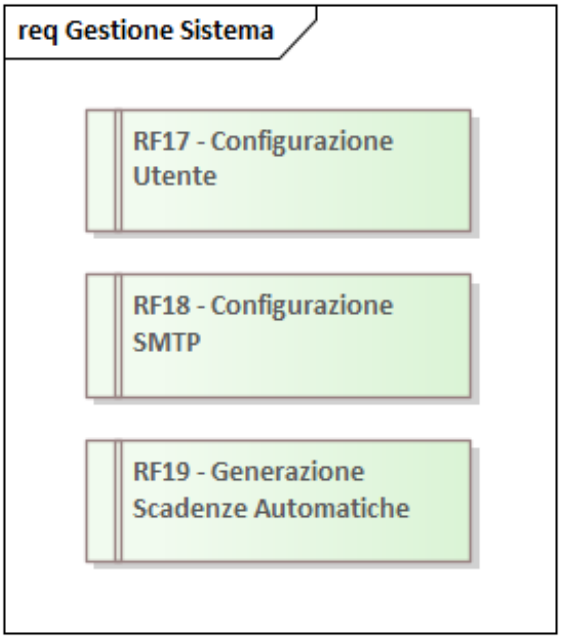


Gestione Sistema

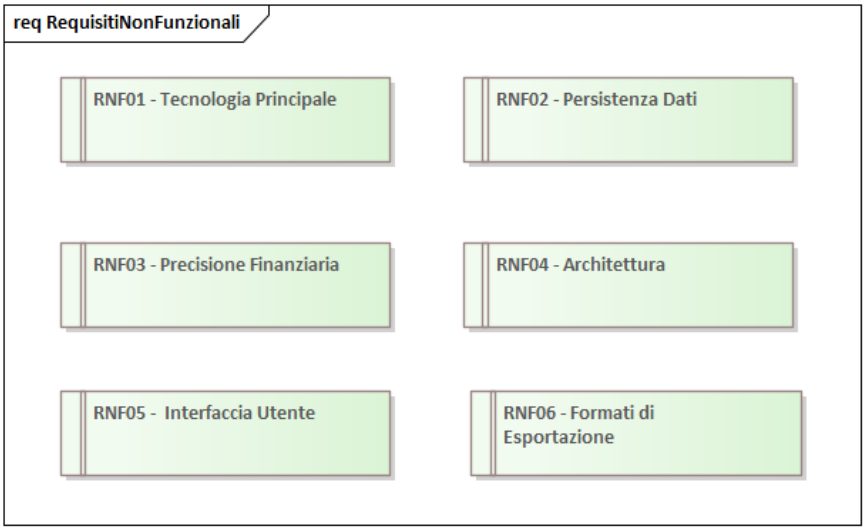
RF17 - Configurazione Utente Il sistema deve permettere all'utente di salvare le proprie aliquote fiscali stimate e i dettagli aziendali da inserire nei PDF.

RF18 - Configurazione SMTP Il sistema deve permettere di configurare i parametri di un server SMTP per l'invio email.

RF19 - Generazione Scadenze Automatiche Il sistema deve scansionare progetti e fatture per popolare automaticamente il calendario con le relative scadenze.



2.2.4 Requisiti Non Funzionali (RNF)



RNF01 - Tecnologia Principale L'intera applicazione (backend e frontend) deve essere sviluppata in linguaggio Python 3.

RNF02 - Persistenza Dati I dati devono essere salvati localmente su file tramite serializzazione `pickle`, per evitare dipendenze da database esterni e facilitare la portabilità.

RNF03 - Precisione Finanziaria Tutti i calcoli monetari devono utilizzare la libreria `Decimal` per garantire la precisione assoluta ed evitare errori di arrotondamento.

RNF04 - Architettura Il codice deve essere nettamente separato nei pacchetti `backend` (logica di business) e `frontend` (interfaccia grafica).

RNF05 - Interfaccia Utente La GUI deve essere moderna, reattiva e user-friendly, utilizzando la libreria `CustomTkinter`.

RNF06 - Formati di Esportazione Il sistema deve generare documenti in formati standard (PDF, XLSX, CSV, PNG).

2.3 Casi d'Uso

Di seguito sono descritti i flussi operativi (casi d'uso) più significativi che il sistema gestisce, coprendo i requisiti funzionali chiave.

Tabella 2.2: Caso d'Uso: Gestire un Contatto (RF01, RF02)

ID: CU 01	
Breve descrizione:	Gestione completa (CRUD e Ricerca) dei contatti in rubrica.
Attori primari:	Ingegnere Freelance
Attori secondari:	Nessuno
Precondizioni:	L'utente si trova nella sezione "Rubrica" dell'applicazione.
Postcondizioni:	I dati dei contatti (`rubrica.pkl`) sono creati, aggiornati o eliminati.
Sequenza eventi principale:	1. L'Ingegnere avvia "Nuovo Contatto" (il form si svuota) o seleziona un contatto (il form si popola). 2. L'Ingegnere compila o modifica i campi del form (nome, tipo, P.IVA, email, ecc.). 3. L'Ingegnere conferma premendo "Salva Modifiche". 4. Il Frontend raccoglie i dati, controlla se è in modalità "Modifica" (`id_contatto_selezionato` presente) o "Creazione" (`id` è `None`). 5. Se "Creazione", invoca `db_rubrica.create_contact(dati)` . Il Backend assegna un ID, aggiunge il contatto alla lista e salva. 6. Se "Modifica", invoca `db_rubrica.update_contact(id, dati)` . Il Backend cerca il contatto, aggiorna i dati e salva. 7. Il Frontend mostra un successo, svuota il form e ricarica la lista contatti.
Sequenze degli eventi alternative:	**(Ricerca)** a. L'Ingegnere digita nella barra di ricerca e preme "Cerca". b. Il Frontend invoca `db_rubrica.search_contacts(query)` . c. Il Backend filtra i contatti in memoria e restituisce la lista filtrata. d. Il Frontend aggiorna la lista visibile.

Tabella 2.3: Caso d'Uso: Creare un Nuovo Progetto (RF03)

ID: CU 02	
Breve descrizione:	Creazione di una nuova commessa di lavoro, associandola a un cliente.
Attori primari:	Ingegnere Freelance

Attori secondari:	Nessuno
Precondizioni:	Il Cliente per cui si crea il progetto deve esistere nella Rubrica (RF01).
Postcondizioni:	Un nuovo progetto viene creato e salvato (`progetti.pkl`).
Sequenza eventi principale:	1. L'Ingegnere avvia l'azione "Nuovo Progetto" dalla sezione Progetti. 2. Il Frontend apre un popup modale (`crea_nuovo_progetto_popup`). 3. Il Frontend interroga `db_rubrica.get_all_contacts(type='Cliente')` e popola una `CTkComboBox` con i clienti. 4. L'Ingegnere seleziona un Cliente, inserisce il nome del progetto e la tariffa oraria. 5. L'Ingegnere conferma l'operazione. 6. Il Frontend raccoglie i dati e invoca `db_progetti.create_project(dati)`. 7. Il Backend valida i dati, crea il dizionario del progetto (con liste vuote) e salva il database dei progetti. 8. Il Frontend mostra un successo, chiude il popup e aggiorna la lista dei progetti.
Sequenze degli eventi alternative:	**(Dati mancanti)** a. Se al punto 5 l'Ingegnere conferma senza selezionare un cliente o inserire un nome: b. Il Frontend mostra un avviso (`tkmb.showwarning`) e l'operazione si interrompe. Il popup rimane aperto.

Tabella 2.4: Caso d'Uso: Gestire un Progetto Esistente (RF04, RF05, RF06)

ID: CU 03	
Breve descrizione:	Aggiornamento di un progetto esistente con fasi, ore lavorate e file.
Attori primari:	Ingegnere Freelance
Attori secondari:	Nessuno
Precondizioni:	Un progetto è stato selezionato dalla lista.
Postcondizioni:	I dati del progetto selezionato (`progetti.pkl`) vengono aggiornati.

Sequenza eventi principale:	**Flusso 1: Registrare Attività (Time-Tracking)** 1. L'Ingegnere naviga nella scheda "Attività" e preme "Aggiungi Attività". 2. Il Frontend apre un popup modale (`apri_popup_attivita`). 3. L'Ingegnere inserisce data, ore (es. `2.5`), descrizione e seleziona "Fatturabile". 4. L'Ingegnere conferma. 5. Il Frontend invoca `db_progetti.add_attivita_to_project(project_id, dati_attivita)` . 6. Il Backend trova il progetto, aggiunge l'attività alla lista `attivita` e salva il database. 7. Il Frontend chiude il popup e aggiorna la lista delle attività. **Flusso 2: Gestire Fasi** 1. L'Ingegnere naviga nella scheda "Fasi". 2. Per aggiungere, apre un popup (`apri_popup_fase`), inserisce i dati e salva (invocando `db_progetti.add_fase_to_project`). 3. Per completare, clicca "Toggle Stato" (invocando `db_progetti.toggle_fase_status`). 4. Il Backend aggiorna la fase e salva; il Frontend aggiorna la lista. **Flusso 3: Allegare File** 1. L'Ingegnere naviga nella scheda "File" e preme "Aggiungi File". 2. Il Frontend apre una finestra di dialogo nativa (`filedialog.askopenfilename`). 3. Il Frontend invoca `db_progetti.add_file_to_project(project_id, path_file)` . 4. Il Backend copia il file nella cartella di progetto e aggiorna il database. 5. Il Frontend aggiorna la lista dei file.
Sequenze degli eventi alternative:	Nessuna.

Tabella 2.5: Caso d'Uso: Creare un Documento Manuale (RF07, RF08)

ID: CU 04	
Breve descrizione:	Creazione manuale di un Preventivo o di una Fattura.
Attori primari:	Ingegnere Freelance
Precondizioni:	Un Cliente deve esistere nella Rubrica per essere selezionato.
Postcondizioni:	Un nuovo Preventivo o Fattura viene creato e salvato (`documenti.pkl`).

Sequenza eventi principale:	1. L'Ingegnere preme "Crea Nuovo Preventivo" o "Crea Nuova Fattura". 2. Il Frontend apre il popup (`apri_popup_documento`) e popola la combobox dei clienti. 3. L'Ingegnere seleziona un cliente, inserisce le voci (descrizione, q.tà, prezzo) e i dati fiscali. 4. L'Ingegnere conferma. 5. Il Frontend raccoglie e valida i dati. 6. Se Fattura, invoca `db_docs.create_invoice(...)`. Se Preventivo, invoca `db_docs.create_quote(...)`. 7. Il Backend (se Fattura) richiede un numero progressivo a `persistence.get_next_document_number`. 8. Il Backend esegue i calcoli (usando `Decimal`), crea l'oggetto e salva il database. 9. Il Frontend mostra un successo, chiude il popup e aggiorna la lista.
Sequenze degli eventi alternative:	Nessuna.

Tabella 2.6: Caso d'Uso: Convertire Preventivo in Fattura (RF09)

ID: CU 05	
Breve descrizione:	Generazione rapida di una Fattura partendo da un Preventivo esistente.
Attori primari:	Ingegnere Freelance
Precondizioni:	Deve esistere almeno un Preventivo non ancora fatturato.
Postcondizioni:	Viene creata una nuova Fattura (`documenti.pkl`) e lo stato del Preventivo viene aggiornato in "Fatturato".
Sequenza eventi principale:	1. L'Ingegnere preme "Crea Fattura da Preventivo". 2. Il Frontend apre un popup (`apri_popup_conversione`) con la lista dei preventivi idonei. 3. L'Ingegnere seleziona il preventivo e preme "Avanti". 4. Il Frontend apre il popup principale (`apri_popup_documento`), pre-compilando i dati del preventivo (cliente e voci) come non modificabili. 5. L'Ingegnere inserisce i dati mancanti (data scadenza, ritenuta) e conferma. 6. Il Frontend invoca `db_docs.convert_quote_to_invoice(quote_id, ...)`. 7. Il Backend esegue la transazione: trova il preventivo, ottiene un nuovo numero fattura, crea la fattura, aggiorna lo stato del preventivo in "Fatturato" e salva il database. 8. Il Frontend mostra un successo e aggiorna entrambe le liste.

Sequenze degli eventi alternative:	Nessuna.

Tabella 2.7: Caso d'Uso: Registrare Incasso da Fattura (RF11)

ID: CU 06	
Breve descrizione:	Riconciliazione di un pagamento ricevuto con una fattura emessa.
Attori primari:	Ingegnere Freelance
Precondizioni:	Deve esistere almeno una Fattura con stato "In sospeso" o "Scaduto".
Postcondizioni:	Viene creato un movimento di "Entrata" (`primanota.pkl`) e lo stato della Fattura (`documenti.pkl`) viene aggiornato in "Pagato".
Sequenza eventi principale:	1. L'Ingegnere, dalla Contabilità, preme "Registra Incasso (da Fattura)". 2. Il Frontend apre un popup (`apri_popup_registra_incasso`) con la lista delle fatture non pagate (lette da `db_docs`). 3. L'Ingegnere seleziona la fattura e inserisce la data di incasso. 4. L'Ingegnere conferma. 5. Il Frontend invoca il workflow `db_ledger.create_movimento_from_invoice(invoice_id, data_incasso)` . 6. Il Backend (Modulo `ledger`) crea il movimento di Entrata e lo salva (`primanota.pkl`). 7. Il Backend (Modulo `ledger`) invoca `db_docs.update_document_status(invoice_id, 'Pagato')` . 8. Il Backend (Modulo `documents`) aggiorna lo stato della fattura e salva (`documenti.pkl`). 9. Il Frontend mostra un successo e aggiorna la lista dei movimenti.
Sequenze degli eventi alternative:	**(Nessuna fattura da incassare)** a. Se al punto 3 la lista delle fatture non pagate è vuota, il Frontend mostra un errore (`tkmb.showerror`) e chiude il popup.

Tabella 2.8: Caso d'Uso: Registrare Movimento di Cassa Manuale (RF10)

ID: CU 07	
Breve descrizione:	Registrazione di un costo (Uscita) o di un incasso non legato a fattura (Entrata).
Attori primari:	Ingegnere Freelance
Precondizioni:	L'utente si trova nella sezione "Contabilità".
Postcondizioni:	Un nuovo movimento (`primanota.pkl`) viene creato.

Sequenza eventi principale:	1. L'Ingegnere preme "Registra Uscita" o "Registra Entrata". 2. Il Frontend apre un popup (`apri_popup_movimento_manuale`). 3. L'Ingegnere compila i campi (data, descrizione, imponibile, IVA, ecc.). 4. L'Ingegnere conferma. 5. Il Frontend valida i dati (assicurandosi che siano `Decimal`) e invoca `db_ledger.create_movimento(dati)` . 6. Il Backend assegna un ID, salva il movimento e restituisce successo. 7. Il Frontend chiude il popup e aggiorna la lista dei movimenti.
Sequenze degli eventi alternative:	Nessuna.

Tabella 2.9: Caso d'Uso: Generare Scadenze Automatiche (RF19)

ID: CU 08	
Breve descrizione:	Popolamento automatico del calendario con scadenze operative e fiscali.
Attori primari:	Sistema (avviato dall'Ingegnere Freelance)
Attori secondari:	Nessuno
Precondizioni:	Nessuna.
Postcondizioni:	Il database del calendario (`calendario.pkl`) viene aggiornato.
Sequenza eventi principale:	1. L'Ingegnere avvia l'azione "Aggiorna Scadenze Automatiche" dalla sezione Calendario. 2. Il Frontend invoca `db_calendario.generate_scadenze_automatiche()` . 3. Il Backend (Modulo `calendar`): a. Carica il calendario e rimuove tutti gli eventi con `type == 'auto'` . b. Chiama `db_progetti.get_all_projects()` e crea eventi per le fasi non completate. c. Chiama `db_docs.get_all_documents()` e crea eventi per le fatture non pagate. d. Salva la lista eventi aggiornata. 4. Il Backend restituisce il numero di eventi generati. 5. Il Frontend mostra un successo e aggiorna la vista del calendario.
Sequenze degli eventi alternative:	Nessuna.

Tabella 2.10: Caso d'Uso: Generare un Report Analitico (RF14, RF15, RF16)

ID: CU 09	
Breve descrizione:	Creazione di output analitici (Excel, grafici, stime) per l'utente o il commercialista.
Attori primari:	Ingegnere Freelance
Attori secondari:	Commercialista (come destinatario dell'export)
Precondizioni:	Esistenza di dati nei moduli (progetti, cassa).
Postcondizioni:	Un file (PNG, XLSX) viene creato sul disco o un popup (stima tasse) viene mostrato.
Sequenza eventi principale:	**Flusso 1: Esportazione Contabile (RF15)** 1. L'Ingegnere preme "Esporta per Commercialista". 2. Il Frontend chiede l'anno e la destinazione (`filedialog.asksaveasfilename`). 3. Il Frontend invoca `db_ledger.export_per_commercialista(file_path, year, format)`. 4. Il Backend usa `pandas` per caricare, filtrare e salvare i movimenti in Excel/CSV. 5. Il Frontend mostra un successo e apre la cartella del file. **Flusso 2: Generazione Grafico Produttività (RF14)** 1. L'Ingegnere preme "Genera Grafico Annuale" (sez. Report Ore). 2. Il Frontend invoca `db_reporting.plot_produttivita_mensile(year, filename)`. 3. Il Backend usa `pandas` per aggregare le ore e `matplotlib` per salvare il grafico `.png`. 4. Il Frontend mostra un successo e apre il file immagine. **Flusso 3: Stima Tasse (RF16)** 1. L'Ingegnere preme un pulsante trimestre (sez. Contabilità). 2. Il Frontend invoca `db_tasse.get_stima_completa(year, quarter)`. 3. Il Backend esegue i calcoli e restituisce un dizionario con i risultati. 4. Il Frontend formatta i risultati e li mostra in un popup.
Sequenze degli eventi alternative:	Nessuna.

Tabella 2.11: Caso d'Uso: Configurare il Sistema (RF17, RF18)

ID: CU 10	
Breve descrizione:	Salvataggio delle impostazioni utente (dati aziendali, aliquote, SMTP).

Attori primari:	Ingegnere Freelance
Attori secondari:	Nessuno
Precondizioni:	Nessuna.
Postcondizioni:	Il file `settings.pkl` viene aggiornato.
Sequenza eventi principale:	1. L'Ingegnere apre un popup di configurazione (es. "Configura SMTP" o "Salva Aliquote"). 2. Il Frontend carica le impostazioni correnti (`db.load_settings()`) e popola i campi. 3. L'Ingegnere modifica i valori e preme "Salva". 4. Il Frontend raccoglie i dati, li valida e invoca `db.save_settings(dati)` . 5. Il Backend (Modulo `persistence`) salva il dizionario completo delle impostazioni nel file `settings.pkl` . 6. Il Frontend mostra un messaggio di successo e chiude il popup.
Sequenze degli eventi alternative:	Nessuna.

2.4 Matrice di Mapping

La matrice di mapping (o matrice di tracciabilità) è uno strumento fondamentale nell'ingegneria dei requisiti. Il suo scopo è garantire che il progetto sia completo e corretto, verificando che ogni Requisito Funzionale (RF) sia stato effettivamente coperto e implementato da almeno un Caso d'Uso (CU).

La tabella seguente pone i Casi d'Uso sulle righe e i Requisiti Funzionali sulle colonne. Un segno di spunta (✓) indica che il Caso d'Uso soddisfa, in tutto o in parte, il requisito specificato.

Tabella 2.12: Matrice di Mapping Requisiti-Casi d'Uso

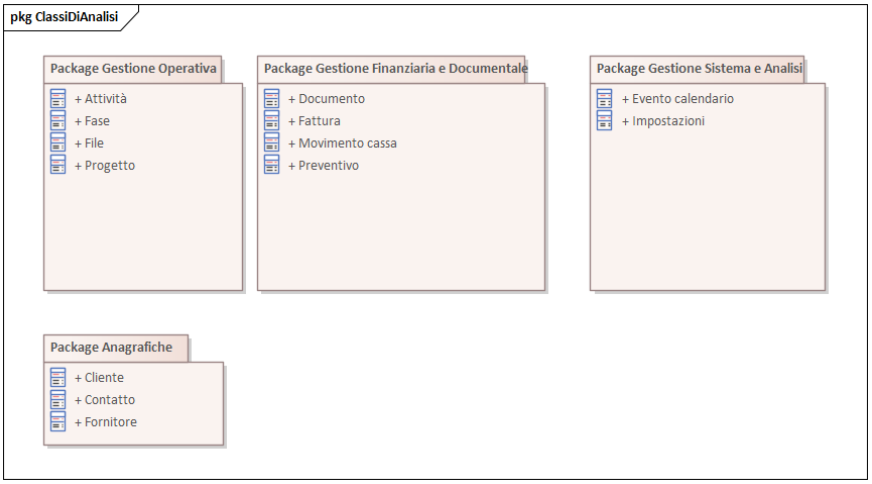
Caso d'Uso	RF01 (CRUD Contatti)	RF02 (Ricerca Contatti)	RF03 (CRUD Progetti)	RF04 (Gestione Fasi)	RF05 (Gestione Attività)	RF06 (Archiviazione File)	RF07 (Creazione Preventivi)	RF08 (Creazione Fatture)	RF09 (Conversione P->F)	RF10 (CRUD Prima Nota)	RF11 (Riconciliazione)	RF12 (Magazzino)	RF13 (Visual. Dashboard)	RF14 (Report Monte Ore)	RF15 (Export Contabile)	RF16 (Stima Tasse)	RF17 (Conf. Utente)	RF18 (Conf. SMTP)	RF19 (Gen. Scadenze)
CU 01: Gestire un Contatto	✓	✓																	
CU 02: Creare un Nuovo Progetto			✓																
CU 03: Gestire un Progetto Esistente			✓	✓	✓	✓													
CU 04: Creare un Documento Manuale							✓	✓											
CU 05: Convertire Preventivo in Fattura									✓										
CU 06: Registrare Incasso da Fattura										✓	✓								
CU 07: Registrare Movimento Cassa Manuale										✓									
CU 08: Generare Scadenze Automatiche																			✓
CU 09: Generare un Report Analitico													✓	✓	✓	✓			
CU 10: Configurare il Sistema																	✓	✓	

2.5 Diagrammi delle Classi

2.5.1 Classi di Analisi

Questa sezione offre un primo spunto di modellazione statica del sistema, definendo le principali entità di business e le loro relazioni, così come sono emerse dall'analisi dei requisiti. Il modello è "ad alto livello" (High-Level), con lo scopo di definire la struttura concettuale dei dati, ed è indipendente dalle scelte implementative (come `pickle` o `CustomTkinter`) che verranno discusse nella fase di progettazione.

Le entità del dominio sono state raggruppate nei seguenti package logici per facilitare l'analisi.



Package Anagrafiche

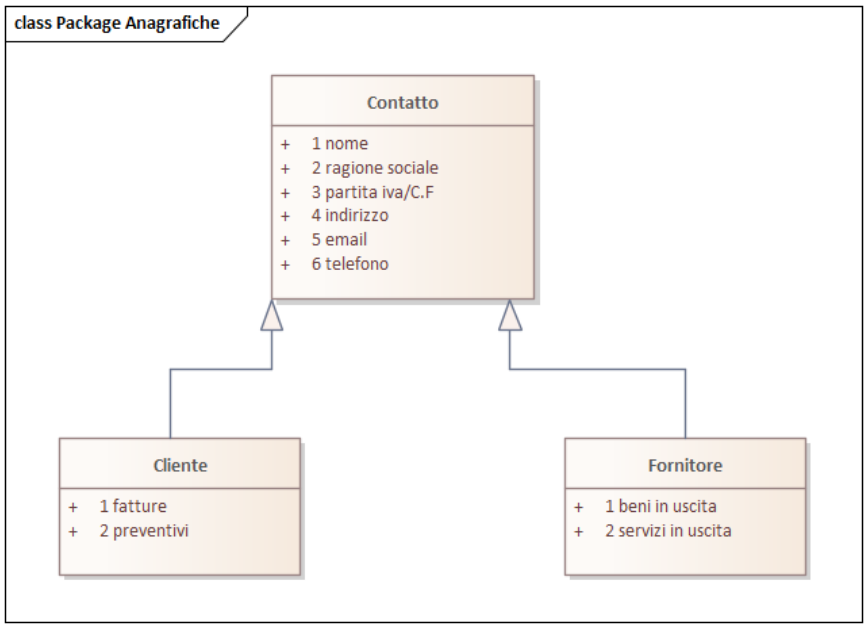
Questo package raggruppa le entità che rappresentano persone fisiche o giuridiche con cui il professionista interagisce.

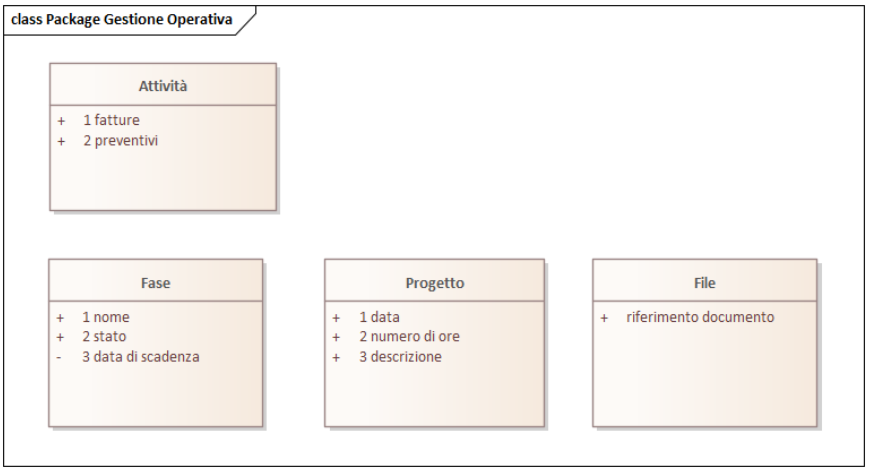
- **Contatto:** Rappresenta l'entità anagrafica di base. Serve come "superclasse" concettuale.
- **Cliente:** È una specializzazione del `Contatto`, destinataria di lavori e documenti attivi.
- **Fornitore:** È una specializzazione del `Contatto`, associata ai movimenti di `Uscita`.

Package Gestione Operativa

Questo package definisce le entità relative al lavoro quotidiano e alla gestione delle commesse.

- **Progetto:** È l'entità centrale che rappresenta una commessa. È associato a un `Cliente` e *contiene* (composizione) `Fasi`, `Attività` e `File`.
- **Fase:** Rappresenta una milestone di un `Progetto`.
- **Attività:** Rappresenta una singola registrazione del time-tracking.
- **File:** Un riferimento a un documento archiviato e associato a un `Progetto`.





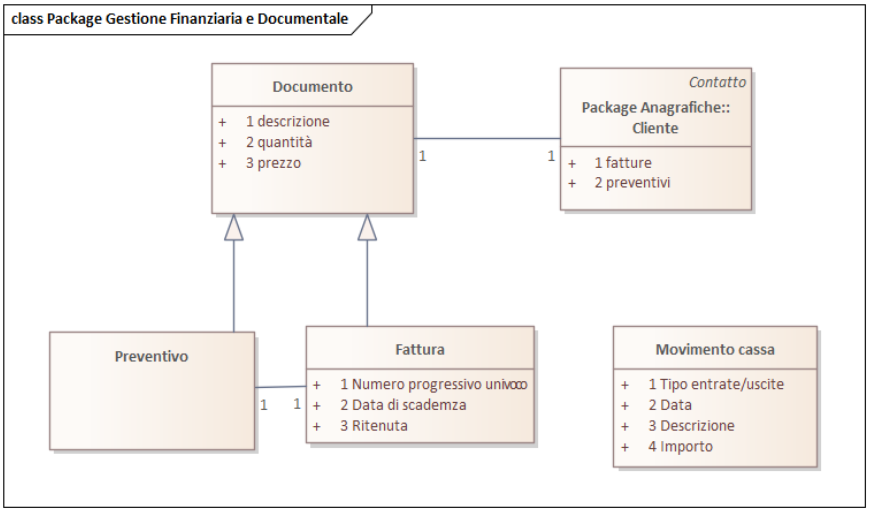
Package Gestione Finanziaria e Documentale

Questo package modella tutte le entità che hanno valore fiscale, contabile o commerciale.

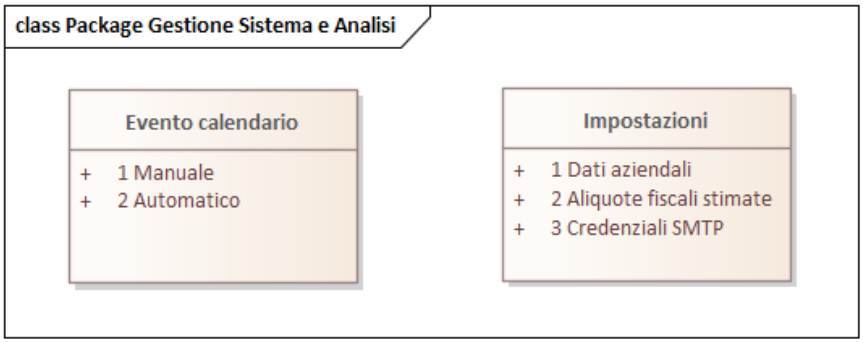
- **Documento:** Entità astratta base per `Preventivo` e `Fattura`.
- **Preventivo:** Specializzazione di `Documento` per le offerte.
- **Fattura:** Specializzazione di `Documento` con dati fiscali aggiuntivi (scadenza, ritenuta).
- **VoceDocumento:** Linea di dettaglio contenuta in un `Documento`.
- **MovimentoCassa:** Entità della prima nota (Entrata/Uscita).
- **Articolo:** Entità opzionale del magazzino.

Package Gestione Sistema e Analisi

Questo package include entità di supporto necessarie al funzionamento del software.



- **EventoCalendario:** Un appuntamento, generato manualmente o automaticamente da `Fasi` o `Fatture`.
- **Impostazioni:** Entità «`Singleton`» che detiene le configurazioni globali (dati aziendali, SMTP, aliquote).



2.5.2 Classi di Progettazione

Questo capitolo segna il passaggio dall'analisi (la definizione dei concetti di business) alla progettazione (la definizione della struttura software concreta). I diagrammi di classe di progettazione "raffinano" le entità concettuali , mappandole sull'architettura software scelta.

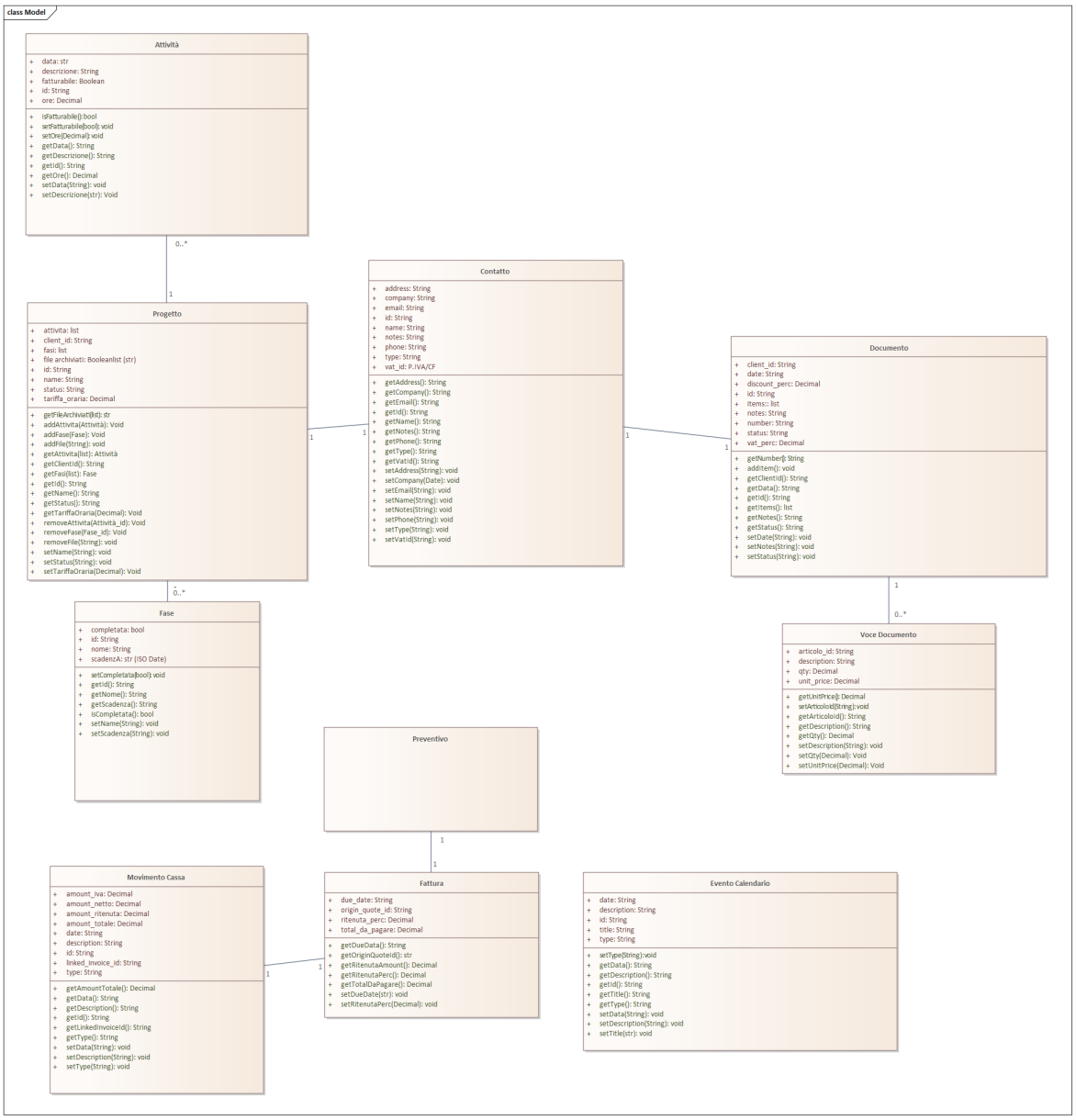
L'architettura del software segue un pattern che separa nettamente la logica di business dai dati su cui essa opera. Questa struttura, ispirata ai principi del Domain-Driven Design e visibile negli esempi forniti (come i diagrammi 'Model' e 'Controller'), è divisa in due package principali:

- **Package Model:** Contiene le classi "Entità" (come `Progetto` , `Fattura`). Queste classi sono "anemiche", ovvero contengono quasi esclusivamente gli attributi (dati) e i metodi di accesso (get/set). Rappresentano i "sostantivi" del sistema.
- **Package Controller/Service:** Contiene le classi "Gestore" (come `GestioneProgetti` , `GestioneFinanziaria`). Queste classi sono "stateless" (prive di stato) e contengono tutta la logica di business (i "verbi"). Ricevono i Model, eseguono operazioni su di essi e restituiscono i risultati.

I diagrammi seguenti illustrano in dettaglio la struttura interna di questi due package, mostrando le classi, i loro attributi e i metodi che definiscono il comportamento del sistema.

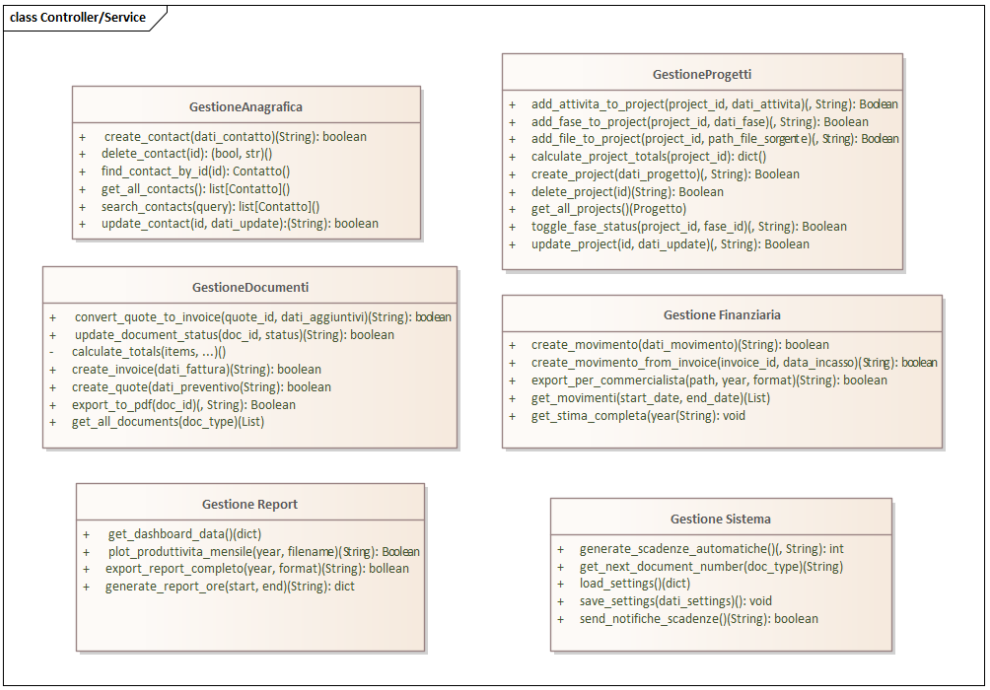
Package: Model (Entità Dati)

Il package Model contiene tutte le classi che rappresentano le strutture dati pure dell'applicazione. Queste classi incapsulano gli attributi e forniscono metodi di accesso (getter e setter) per la loro manipolazione, ma non contengono logica di business complessa.



Controller/Service (Gestori Logici)

Il package Controller (o Service) contiene i componenti "stateless" che implementano la logica di business. Queste classi ricevono le entità del Model, eseguono operazioni (calcoli, validazioni, workflow) e ne restituiscono i risultati.

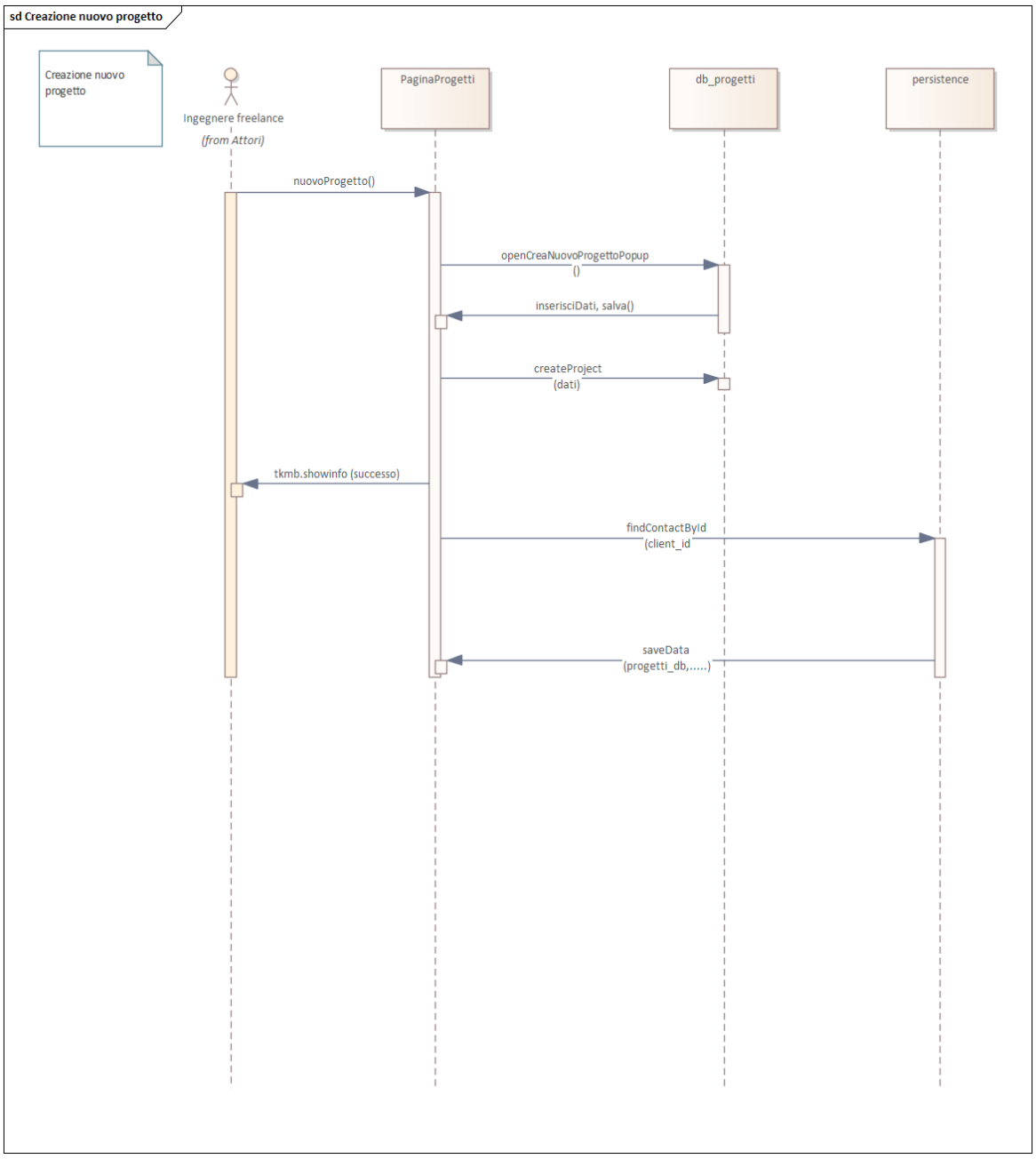


2.6 Diagrammi di Sequenza

I Diagrammi di Sequenza espongono in maniera chiara come le diverse parti del sistema comunicano tra di loro, scambiandosi messaggi (chiamate di funzione) con lo scopo di svolgere le operazioni definite nei casi d'uso. Questa vista dinamica è fondamentale per comprendere i flussi di controllo e di dati all'interno dell'applicazione.

SD 01: Creazione Nuovo Progetto

Questo diagramma illustra la sequenza di chiamate per la creazione di un nuovo progetto. L'interazione inizia dal `Frontend`, che prima interroga il gestore `db_rubrica` per ottenere l'elenco dei clienti da mostrare nel popup, e poi, alla conferma, invoca il gestore `db_progetti` per creare e persistere la nuova entità.



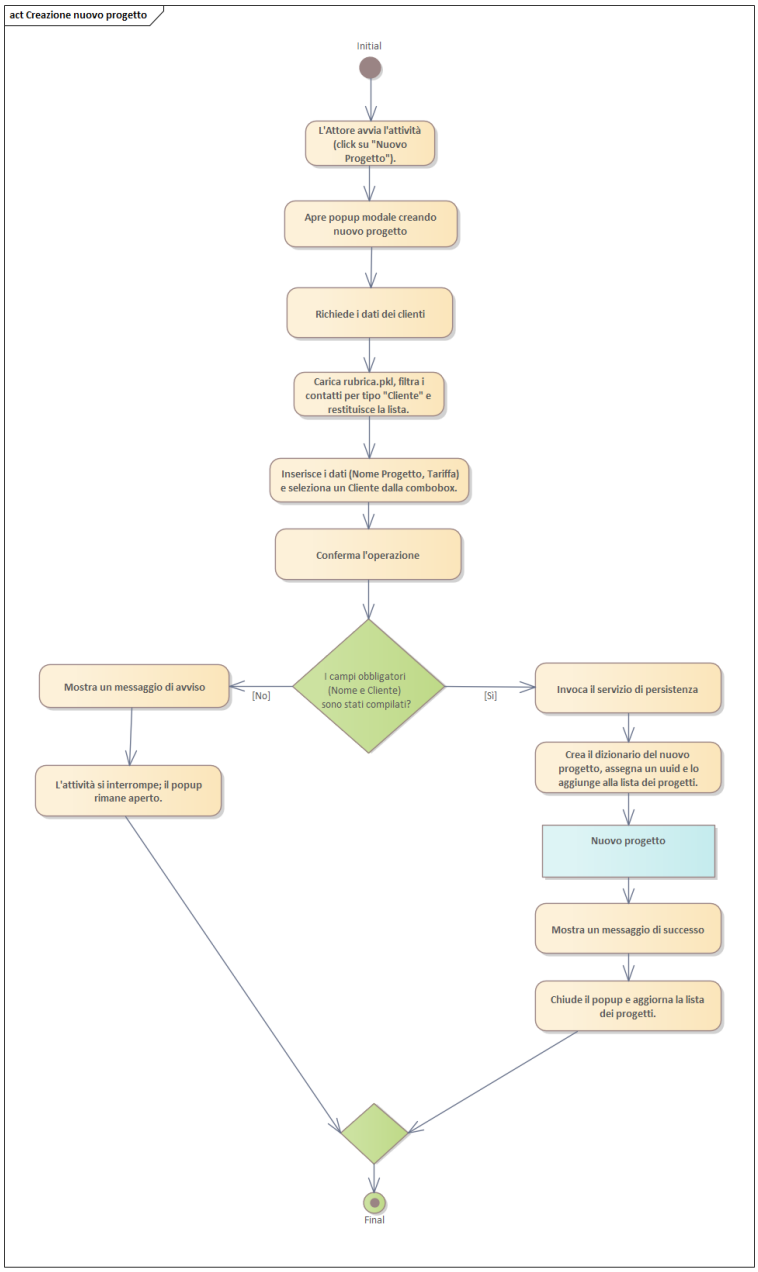
2.7 Diagrammi di Attività

I diagrammi di attività qui proposti ricalcano i comportamenti già descritti dai diagrammi di sequenza, ma utilizzando un approccio differente. L'obiettivo non è più mostrare i messaggi scambiati tra le componenti (le *lifeline*), bensì modellare il **flusso di controllo** algoritmico del processo.

Questi diagrammi scompongono un caso d'uso complesso in "azioni atomiche" (le attività), nodi decisionali (i rombi, che rappresentano gli `if...else`) e flussi alternativi. In questa fase di analisi, viene realizzato un diagramma di attività per ogni flusso operativo significativo, fornendo una visione comportamentale complementare a quella dei diagrammi di sequenza.

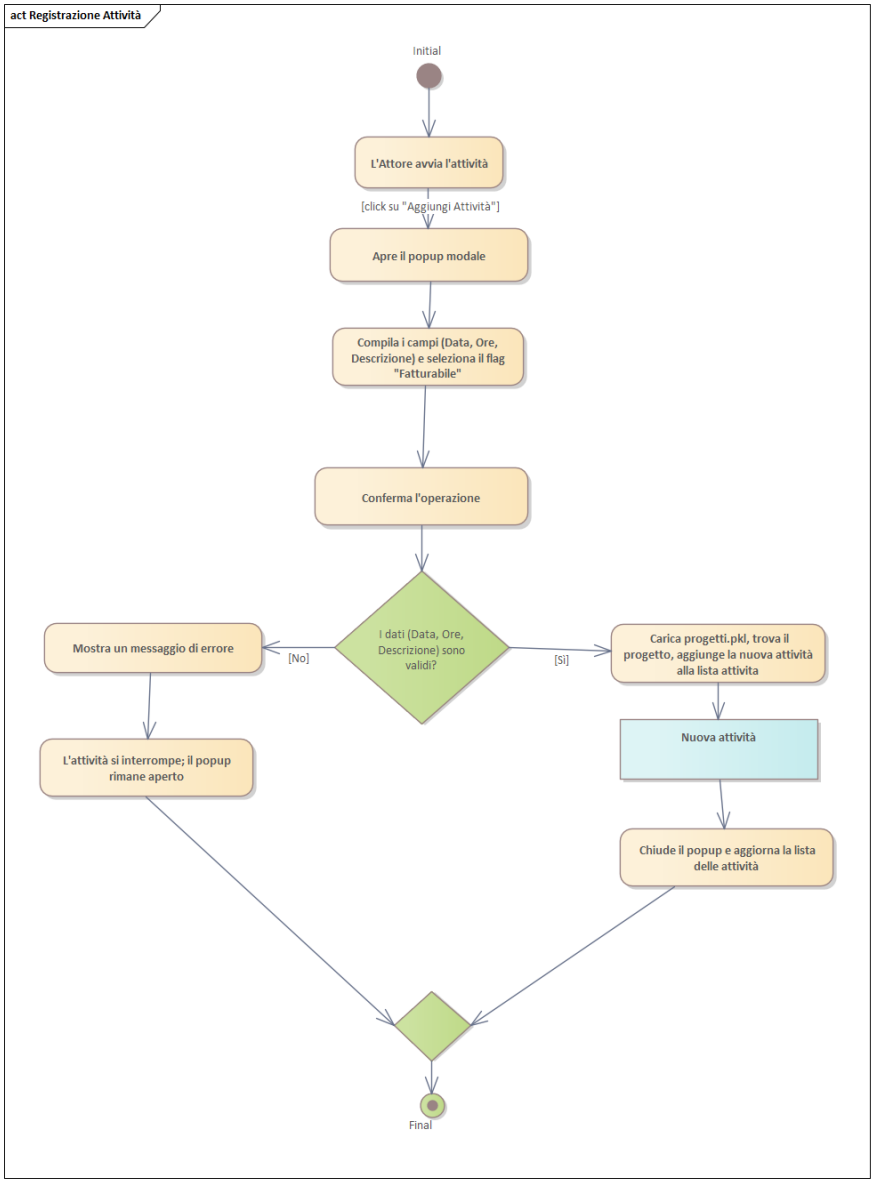
AD 01: Creazione Nuovo Progetto

Questo diagramma mostra il flusso per la creazione di un nuovo progetto. Inizia con l'azione dell'utente, prosegue con il caricamento dei dati dei clienti, la raccolta dell'input e un nodo decisionale che valida i campi obbligatori. Se la validazione fallisce, il flusso termina; se ha successo, il flusso prosegue con l'invocazione del `backend` e la persistenza dei dati.



AD 02: Registrazione Attività

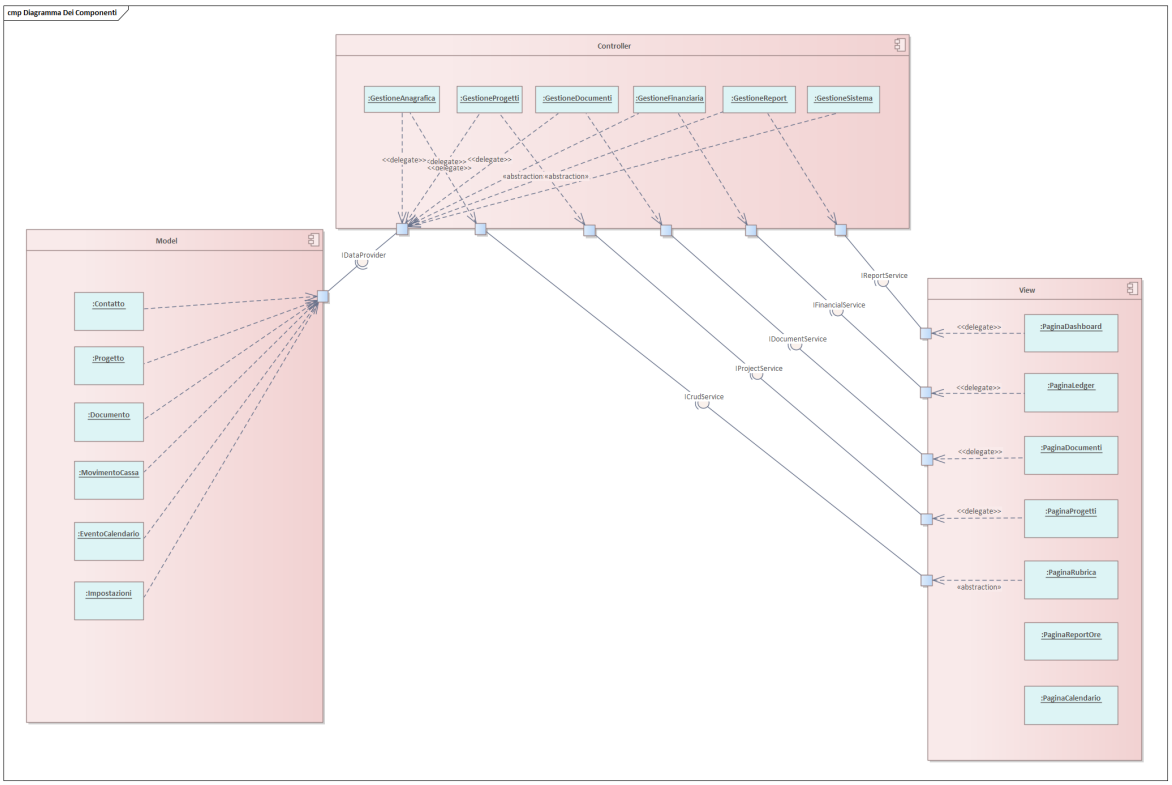
Questo flusso modella l'inserimento di una nuova attività. Include la validazione dei dati (es. verifica che il campo "ore" sia un numero valido). Il nodo decisionale indirizza il flusso verso un messaggio di errore o verso l'attività di salvataggio nel `backend` .



2.8 Diagramma dei Componenti

Il Diagramma dei Componenti offre una visione architetturale di "alto livello" del sistema, astraendo dalle singole classi (viste nel Capitolo 8) per concentrarsi sui moduli software e sulle loro interdipendenze. Un componente, secondo la definizione UML, è un blocco modulare e sostituibile del sistema, il cui comportamento è interamente descritto dalle interfacce che *fornisce* (provides) e dalle interfacce che *richiede* (requires).

L'architettura del software gestionale, come definito nel pattern MVC, è suddivisa in tre componenti principali: **Model**, **View**, e **Controller**.



2.9 Macchine a stati

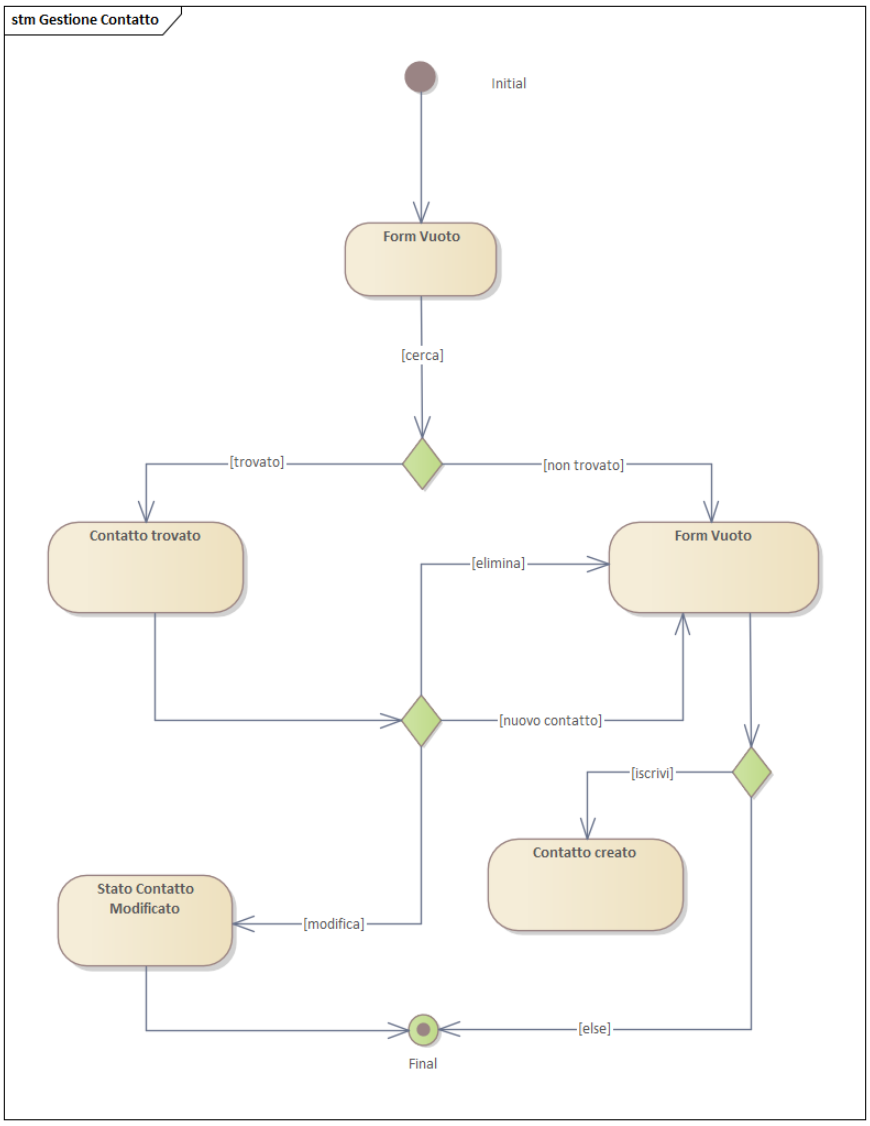
I Diagrammi delle Macchine a Stati (State Machine Diagrams) rappresentano il ciclo di vita di un oggetto (un'istanza di una classe di business), descrivendo in dettaglio gli stati attraverso i quali esso può transitare e gli eventi (azioni dell'utente o del sistema) che ne causano la transizione.

Questa modellazione è fondamentale per comprendere la logica temporale e le regole che governano i processi di business. Verranno presentati due tipi di diagrammi:

- **Diagramma Comportamentale (di Processo):** Modella un processo o un caso d'uso, mostrando il flusso di stati dell'interfaccia utente (simile all'esempio `stm Dipendente` fornito).
- **Diagramma del Ciclo di Vita (di Entità):** Modella gli stati interni di un'entità di business (simile all'esempio `stm Biglietto` del PDF) in risposta agli eventi.

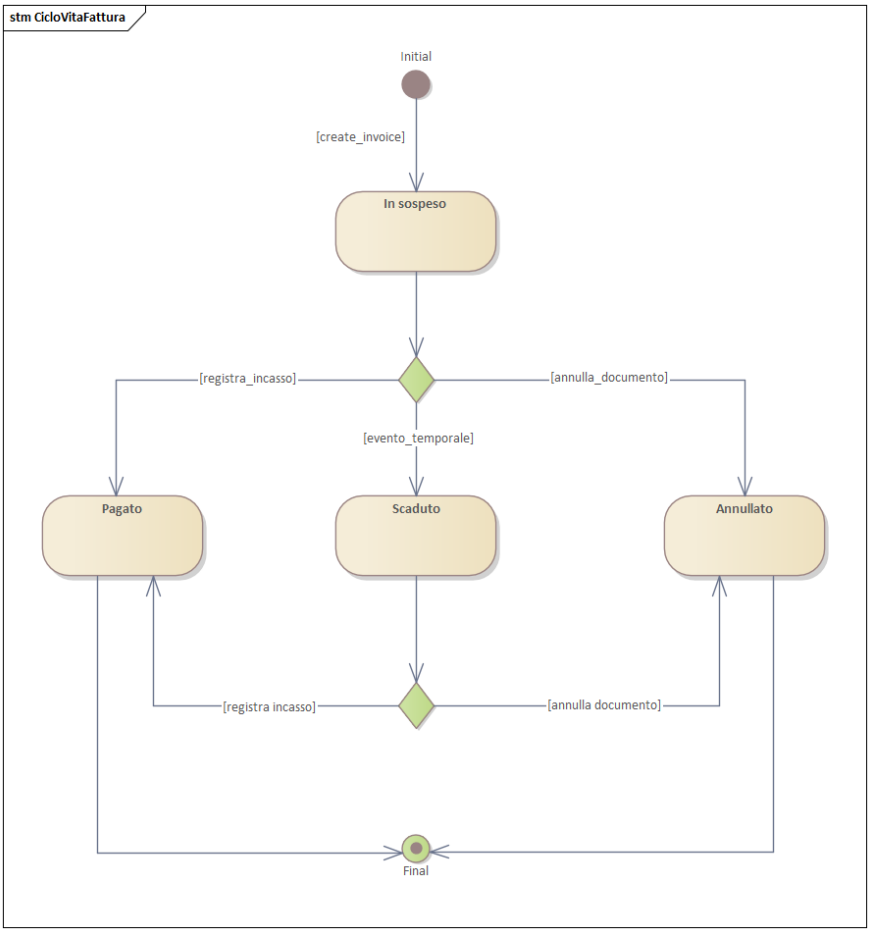
`stm GestioneContatto`

Questo diagramma modella il processo che l'utente (Ingegnere Freelance) segue quando interagisce con la pagina della Rubrica per cercare, visualizzare, modificare o creare un `Contatto` . Il diagramma mostra i cambi di stato dell'interfaccia utente (es. da "Form Vuoto" a "Contatto Trovato") in risposta agli eventi dell'utente (es. '[cerca]', '[modifica]').



`stm CicloVitaFattura`

Questo diagramma modella il ciclo di vita di una singola entità `Fattura` all'interno del sistema, mostrando come il suo attributo `status` cambia in base agli eventi. Gli stati principali sono `In Sospeso`, `Scaduto`, `Pagato` e `Annullato`. Le transizioni sono causate da eventi come `[registra_incasso]` o `[evento_temporale]`.



2.10 Diagramma di Deployment

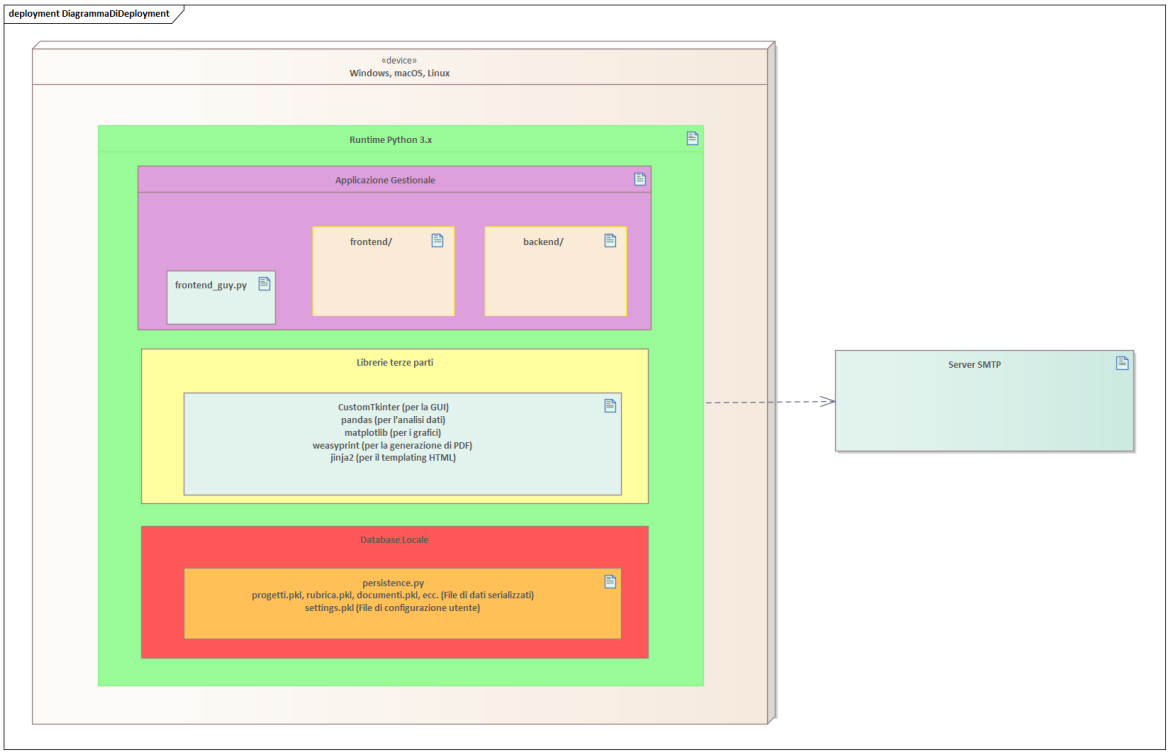
Il diagramma di deployment (distribuzione) rappresenta l'architettura fisica del sistema. Il diagramma illustra come gli artefatti software (i componenti eseguibili, le librerie e i file di dati) sono allocati sui nodi hardware che compongono l'ambiente operativo.

Come definito dai requisiti (RNF1, RNF2), il software è progettato come un'applicazione desktop *standalone* (autonoma). Non segue un'architettura client-server e non dipende da un server database esterno. L'intero sistema è quindi concepito per essere eseguito su un unico nodo fisico.

Il diagramma di deployment identifica i seguenti nodi e artefatti:

- **Nodo: `PC Locale del Professionista` (Host Windows/Linux/macOS)**
  - È il nodo computazionale primario (il computer dell'utente).
  - **Artefatto Ospitato: `Runtime Python 3.x`** (L'ambiente di esecuzione).
  - **Artefatto Ospitato: `Applicazione Gestionale`** (I pacchetti `backend/` e `frontend/`).
  - **Artefatto Ospitato: `Librerie di Terze Parti`** (Es. `CustomTkinter`, `pandas`, `weasyprint`).
  - **Artefatto Ospitato: `Database Locale`** (I file `*.pkl` e `settings.pkl` gestiti da `persistence.py`).
- **Nodo Esterno: `Server SMTP`**
  - È un nodo di rete esterno, non gestito dall'applicazione ma da essa utilizzato.
  - **Dipendenza:** Il componente `backend/email_utils.py` stabilisce una connessione di rete (protocollo SMTP) a questo nodo per inviare le email di notifica.

In sintesi, il deployment è volutamente semplice e riflette la natura autonoma dell'applicazione, dove tutti i componenti logici e i dati risiedono sulla stessa macchina.


deployment DiagrammaDiDeployment
«device»
Windows, macOS, Linux
Runtime Python 3.x
Applicazione Gestionale
frontend/
backend/
frontend_guy.py
Librerie terze parti
CustomTkinter (per la GUI)
pandas (per l'analisi dati)
matplotlib (per i grafici)
weasyprint (per la generazione di PDF)
jinja2 (per il templating HTML)
Database Locale
persistence.py
progetti.pkl, rubrica.pkl, documenti.pkl, ecc. (File di dati serializzati)
settings.pkl (File di configurazione utente)
Server SMTP

Parte III

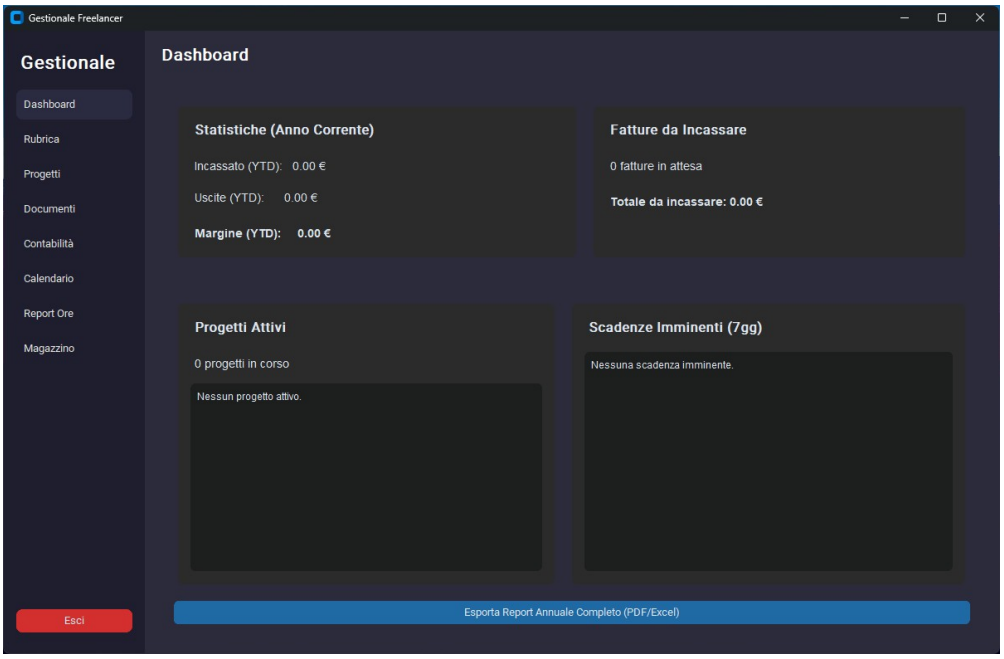
Interfacce Utente

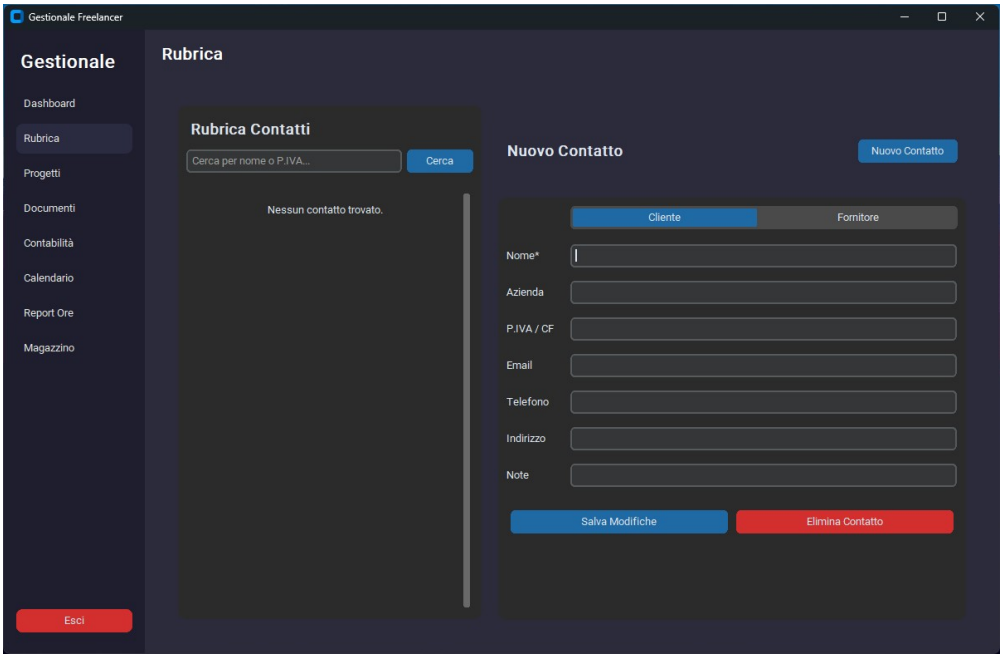
CHAPTER 3

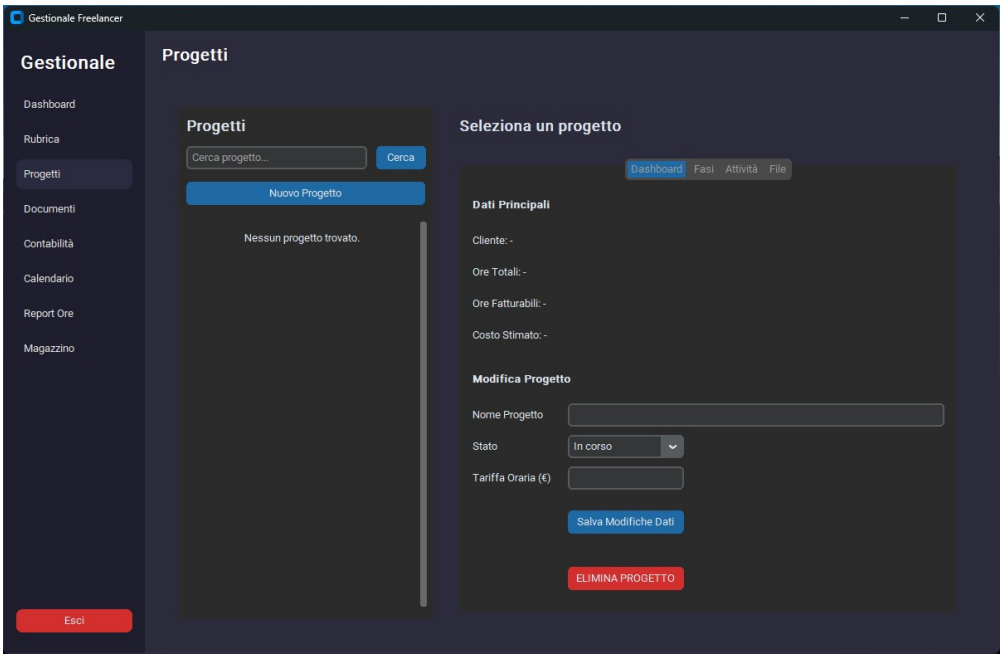
Immagini del Software

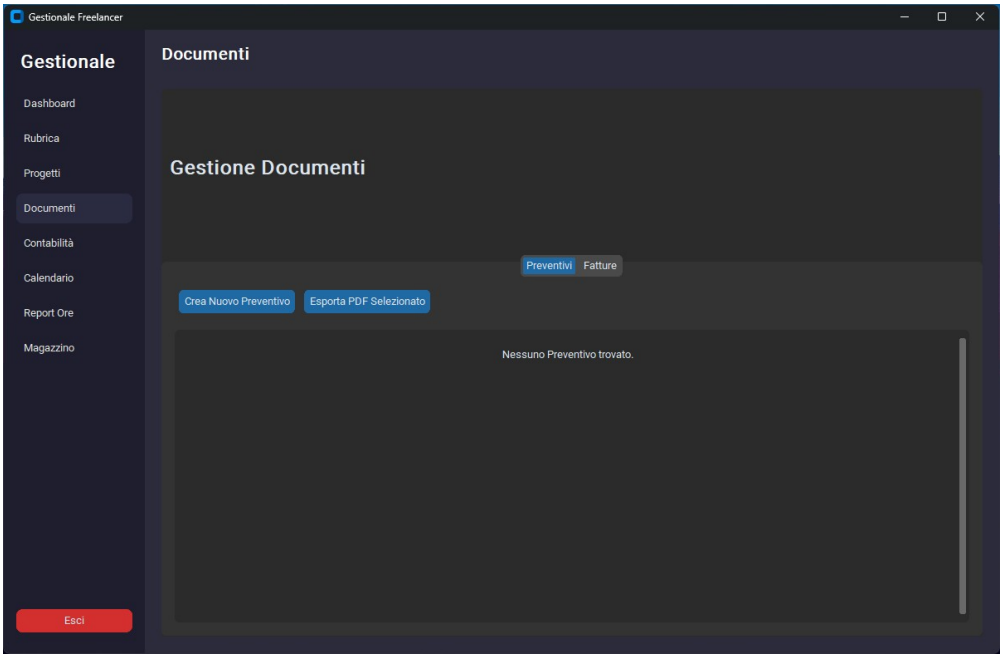
In seguito sono presenti degli screen effettuati al software durante l'uso.

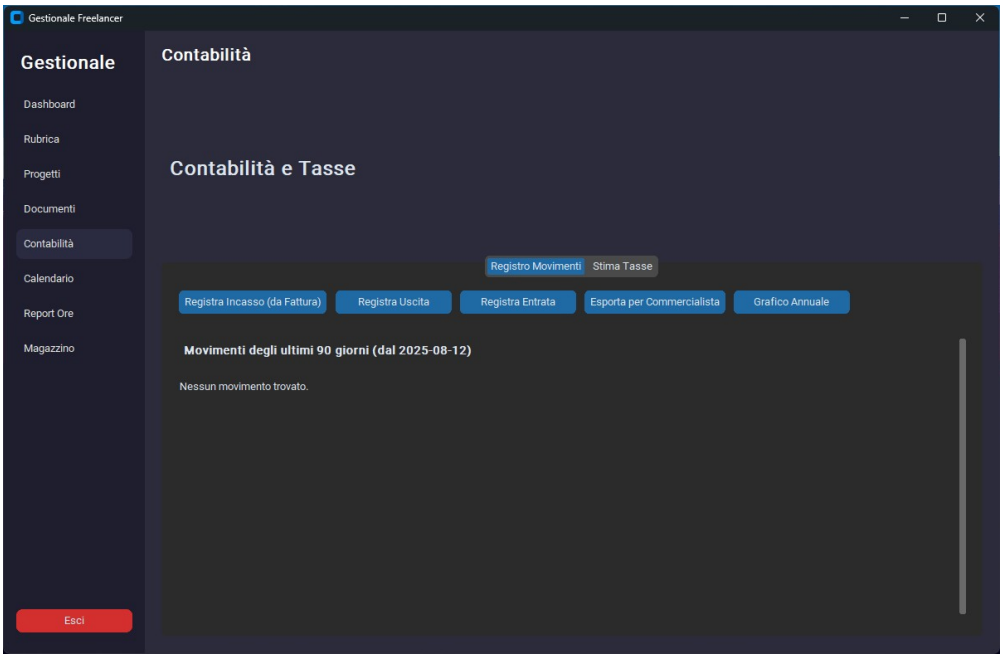
3.1 Il software

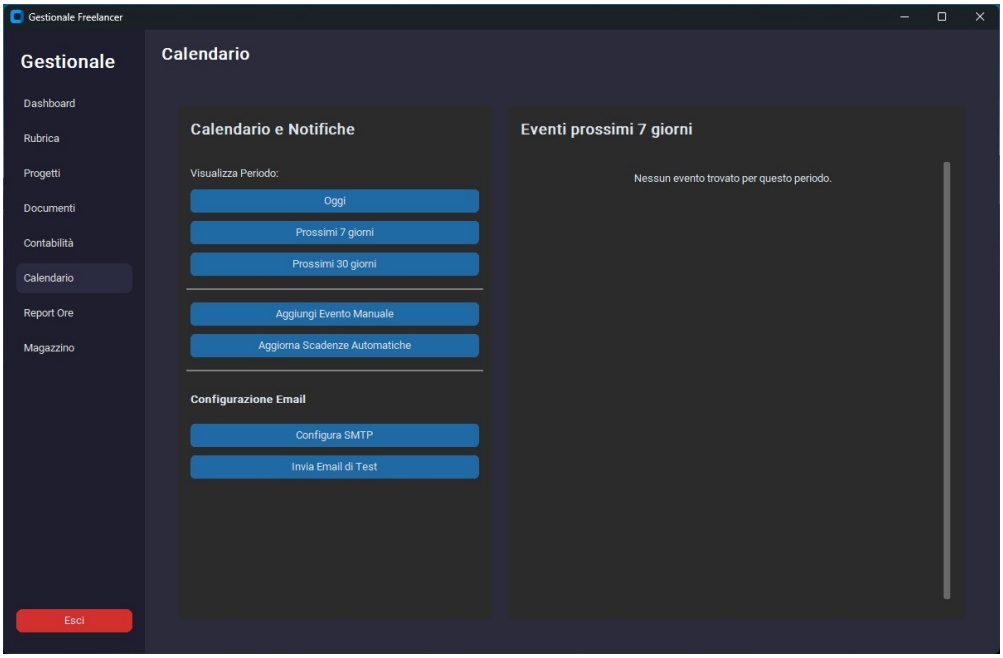


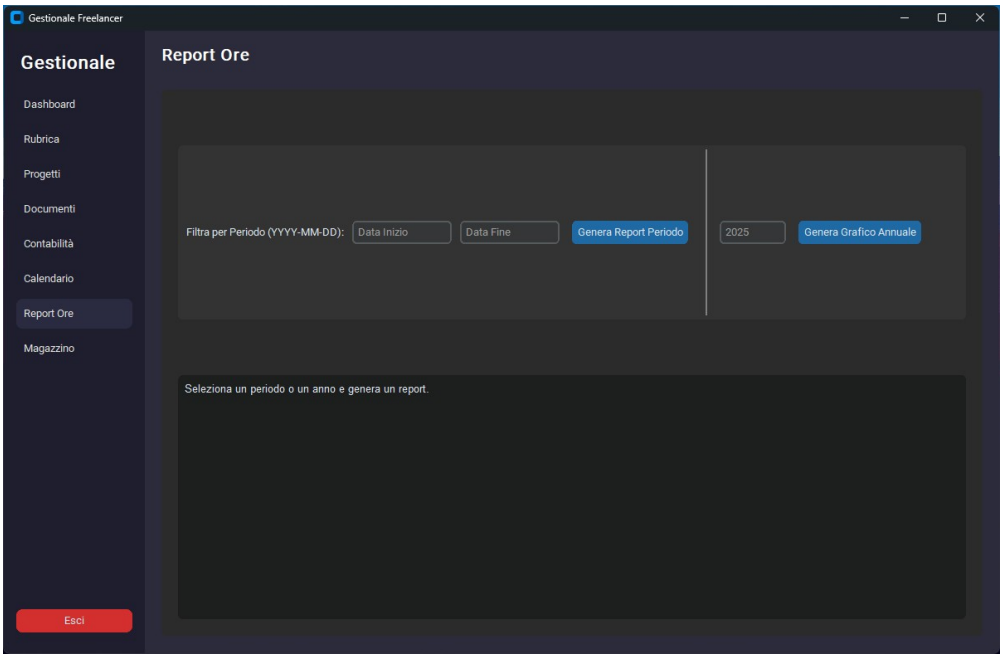

Gestionale Freelancer
Gestionale
Dashboard
Rubrica
Progetti
Documenti
Contabilità
Calendario
Report Ore
Magazzino
Esci
Rubrica
Rubrica Contatti
Cerca per nome o P.IVA...
Cerca
Nessun contatto trovato.
Nuovo Contatto
Nuovo Contatto
Cliente
Fornitore
Nome*
Azienda
P.IVA / CF
Email
Telefono
Indirizzo
Note
Salva Modifiche
Elimina Contatto

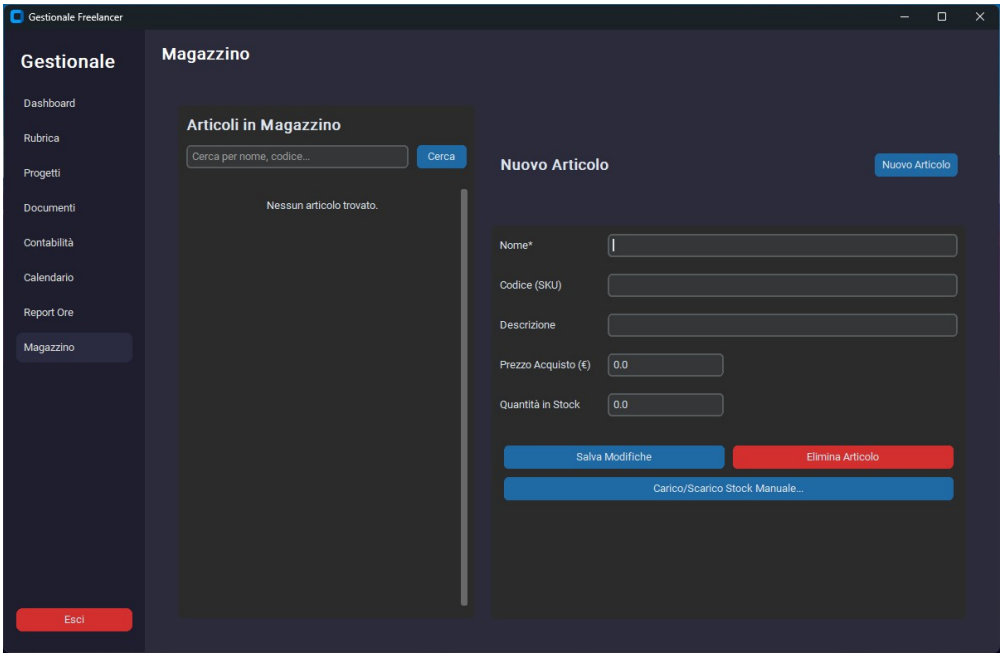

Gestionale Freelancer
Gestionale
Dashboard
Rubrica
Progetti
Documenti
Contabilità
Calendario
Report Ore
Magazzino
Esci
Progetti
Progetti
Cerca progetto...
Cerca
Nuovo Progetto
Nessun progetto trovato.
Seleziona un progetto
Dashboard
Fasi
Attività
File
Dati Principali
Cliente: -
Ore Totali: -
Ore Fatturabili: -
Costo Stimato: -
Modifica Progetto
Nome Progetto
Stato
In corso
Tariffa Oraria (€)
Salva Modifiche Dati
ELIMINA PROGETTO


Gestionale Freelancer
Gestionale
Dashboard
Rubrica
Progetti
Documenti
Contabilità
Calendario
Report Ore
Magazzino
Esci
Documenti
Gestione Documenti
Preventivi
Fatture
Crea Nuovo Preventivo
Esporta PDF Selezionato
Nessuno Preventivo trovato.


Gestionale Freelancer
Gestionale
Dashboard
Rubrica
Progetti
Documenti
Contabilità
Calendario
Report Ore
Magazzino
Esci
Contabilità
Contabilità e Tasse
Registro Movimenti
Stima Tasse
Registra Incasso (da Fattura)
Registra Uscita
Registra Entrata
Esporta per Commercialista
Grafico Annuale
Movimenti degli ultimi 90 giorni (dal 2025-08-12)
Nessun movimento trovato.


Gestionale Freelancer
Gestionale
Dashboard
Rubrica
Progetti
Documenti
Contabilità
Calendario
Report Ore
Magazzino
Esci
Calendario
Calendario e Notifiche
Visualizza Periodo:
Oggi
Prossimi 7 giorni
Prossimi 30 giorni
Aggiungi Evento Manuale
Aggiorna Scadenze Automatiche
Configurazione Email
Configura SMTP
Invia Email di Test
Eventi prossimi 7 giorni
Nessun evento trovato per questo periodo.


Gestionale Freelancer
Gestionale
Dashboard
Rubrica
Progetti
Documenti
Contabilità
Calendario
Report Ore
Magazzino
Esci
Report Ore
Filtra per Periodo (YYYY-MM-DD):
Data Inizio
Data Fine
Genera Report Periodo
2025
Genera Grafico Annuale
Seleziona un periodo o un anno e genera un report.


Gestionale Freelancer
Gestionale
Dashboard
Rubrica
Progetti
Documenti
Contabilità
Calendario
Report Ore
Magazzino
Esci
Magazzino
Articoli in Magazzino
Cerca per nome, codice...
Cerca
Nessun articolo trovato.
Nuovo Articolo
Nuovo Articolo
Nome*
Codice (SKU)
Descrizione
Prezzo Acquisto (€)
0.0
Quantità in Stock
0.0
Salva Modifiche
Elimina Articolo
Carico/Scarico Stock Manuale...

Parte IV

Tests

CHAPTER 4

Testing dell'applicazione

Test

La fase di test è cruciale per validare la correttezza, la robustezza e l'affidabilità del software. Per questo progetto, l'approccio al testing si è concentrato sul **backend**, verificando la logica di business e le interazioni tra i moduli in modo isolato dall'interfaccia utente (GUI).

Sono stati utilizzati il framework `unittest` di Python e la libreria `unittest.mock` per creare "stub" e "mock" dei componenti dipendenti (come il salvataggio su file o le chiamate ad altri moduli). Questo ha permesso di eseguire test unitari (*unit test*) e test di integrazione (*integration test*) mirati.

4.0.1 Test del Modulo `persistence`

L'obiettivo di questi test è validare la logica di business interna al livello di persistenza, in particolare la gestione critica della numerazione progressiva dei documenti.

Test 1: Incremento Semplice

Questo test verifica il caso d'uso standard di richiesta di un nuovo numero di fattura.

1. **Setup:** Il test simula (tramite *mock*) di essere nell'anno 2025 e che l'ultimo numero salvato (letto da `load_settings`) è il 41.
2. **Azione:** Viene invocata la funzione `get_next_document_number('invoice')` .
3. **Verifica:** Il test asserisce che il numero restituito sia "F2025/042". Inoltre, verifica che la funzione `save_settings` sia stata chiamata e che i dati salvati contengano il nuovo contatore aggiornato a 42.

Test 2: Rollover di Fine Anno

Questo test verifica lo scenario critico del cambio d'anno, che deve azzerare il contatore e aggiornare il prefisso.

1. **Setup:** Il test simula di essere nel nuovo anno (1 Gennaio 2026). I dati caricati da `load_settings` sono quelli dell'anno precedente (ultimo numero 99, prefisso "F2025/").

2. **Azione:** Viene invocata la funzione `get_next_document_number('invoice')`.
3. **Verifica:** Il test asserisce che il numero restituito sia "F2026/001". Verifica inoltre che i dati salvati su disco contengano il contatore azzerato a 1 e il nuovo prefisso "F2026/".

4.0.2 Test del Modulo `documents`

L'obiettivo è validare la correttezza dei calcoli finanziari (RNF03) e la corretta interazione con il modulo di inventario.

Test 3: Calcolo Finanziario Complesso

Questo è un puro test unitario sulla logica di calcolo finanziario.

1. **Setup:** Viene creato un set di voci (totale 200€) e applicati sconto (10%), IVA (22%) e ritenuta (20%).
2. **Azione:** Viene invocata la funzione privata `_calculate_totals(...)`.
3. **Verifica:** Il test asserisce, utilizzando il tipo `Decimal`, che ogni campo calcolato (imponibile, iva, totale da pagare, ecc.) sia *esattamente* uguale ai valori attesi, garantendo l'assenza di errori di arrotondamento.

Test 4: Creazione Fattura e Scarico Magazzino

Questo test di integrazione verifica che la creazione di una fattura attivi correttamente lo scarico del magazzino (RF12).

1. **Setup:** Si crea una voce di fattura collegata a un ID articolo (`articolo_id: 'item_123'`) con quantità 2. Si simula (mock) una risposta positiva da `db_magazzino.update_st`.
2. **Azione:** Viene invocata la funzione `create_invoice(...)`.
3. **Verifica:** Il test asserisce che il mock di `db_magazzino.update_stock` sia stato chiamato esattamente una volta, con i parametri corretti: `'item_123'` e `Decimal('-2')` (la quantità negativa da scaricare).

Test 5: Fallimento Creazione Fattura per Stock Insufficiente

Questo test verifica il "percorso triste" (sad path): la creazione della fattura deve fallire se il magazzino non ha scorte.

1. **Setup:** Si simula (mock) che la funzione `db_magazzino.update_stock` fallisca, restituendo `(False, "Stock insufficiente")`.
2. **Azione:** Si invoca `create_invoice(...)`.
3. **Verifica:** Il test asserisce che venga sollevata un'eccezione `ValueError` e, soprattutto, che la funzione di salvataggio (`db.save_data`) **non** sia stata chiamata, garantendo che la fattura non venga creata se lo scarico fallisce.

4.0.3 Test del Modulo `ledger`

L'obiettivo è validare il workflow di riconciliazione dei pagamenti (CU 06), che collega i moduli `ledger` e `documents`.

Test 6: Riconciliazione Pagamento Fattura

Questo test di integrazione verifica il Caso d'Uso CU 06.

1. **Setup:** Si simula il caricamento di una fattura ("inv123") con stato "In sospeso" e i relativi importi.
2. **Azione:** Viene invocata la funzione `ledger.create_movimento_from_invoice('inv123', ...)`.
3. **Verifica:** Il test esegue due asserzioni critiche:
   (a) Che la funzione `db_docs.update_document_status` sia stata chiamata con i parametri `('inv123', 'Pagato')`.
   (b) Che la funzione `db.save_data` del `ledger` sia stata chiamata con un nuovo movimento di "Entrata" contenente gli stessi dati finanziari (importi, numero fattura) copiati dalla fattura.

Test 7: Annullamento Riconciliazione

Questo test verifica la logica di "annullamento": se un pagamento collegato viene eliminato, lo stato della fattura deve tornare "In sospeso".

1. **Setup:** Si simula l'esistenza di un movimento di pagamento collegato all'ID `'inv123'`.
2. **Azione:** Viene invocata la funzione `ledger.delete_movimento(...)`.
3. **Verifica:** Il test asserisce che la funzione `db_docs.update_document_status` sia stata chiamata per ripristinare lo stato della fattura `'inv123'` a `'In sospeso'`.

Test 8: Eliminazione Movimento Manuale

Questo test verifica che l'eliminazione di un movimento non collegato (es. un costo manuale) non tenti erroneamente di aggiornare alcuno stato di fattura.

1. **Setup:** Si simula un movimento con `linked_invoice_id: None`.
2. **Azione:** Viene invocata la funzione `ledger.delete_movimento(...)`.
3. **Verifica:** Il test asserisce che la funzione `db_docs.update_document_status` **non** sia stata chiamata.

Parte V

Conclusioni

In conclusione, questo progetto ha dimostrato la fattibilità e l'efficacia di un software gestionale desktop integrato, progettato specificamente per le esigenze di un ingegnere freelance. Sono riuscito a implementare con successo le funzionalità chiave di gestione anagrafica, tracciamento dei progetti (incluse fasi e time-tracking), e l'intero ciclo finanziario "Quote-to-Cash", dall'emissione di preventivi alla riconciliazione dei pagamenti in prima nota.

Nonostante le sfide intrinseche legate alla gestione dei dati senza un database server (utilizzando `pickle`) e alla necessità di garantire la precisione assoluta dei calcoli finanziari, l'applicazione raggiunge gli obiettivi prefissati. Offre all'utente uno strumento coeso e intuitivo per acquisire una visione d'insieme completa della propria attività, sostituendo la frammentazione di molteplici strumenti. L'adozione di un'architettura **Model-View-Controller** (separando nettamente `backend` e `frontend`) e l'astrazione del livello di persistenza ('persistence.py') hanno contribuito significativamente alla manutenibilità, scalabilità e chiarezza del codice.

Le limitazioni riscontrate, legate principalmente alla scelta di `pickle` come motore di persistenza (in termini di scalabilità e assenza di concorrenza), sono state una scelta di design consapevole per favorire la semplicità e la portabilità, ma definiscono chiaramente i passi futuri. La decisione di utilizzare **CustomTkinter** per l'interfaccia si è rivelata vincente, fornendo un'estetica moderna e un'ottima esperienza utente con un minimo overhead.

Per il futuro, questo progetto rappresenta una solida base e offre ampie possibilità di espansione. Tra i potenziali miglioramenti si potrebbero includere:

- **Migrazione del Database (Priorità Alta):** Sostituzione del motore di persistenza `pickle` con un database **SQLite**, per migliorare drasticamente le prestazioni, la sicurezza dei dati e l'efficienza delle interrogazioni.
- **Sviluppo di un'API Web:** Esposizione della logica `backend` tramite un'API REST (con **Flask** o **FastAPI**) per disaccoppiare la logica dall'interfaccia e aprire alla possibilità di client futuri (es. un'app mobile).
- **Integrazione Contabile:** Aggiunta di moduli per l'esportazione dei dati in formati compatibili con software di contabilità esterni o per la pre-compilazione della fatturazione elettronica.
- **Miglioramenti UI/UX:** Ulteriori ottimizzazioni dell'interfaccia, come l'introduzione di grafici interattivi (invece di `.png` statici) nel dashboard e nei report.

In sintesi, questo software gestionale rappresenta una solida base per un'applicazione professionale, con un notevole potenziale di crescita, mirato a fornire al libero professionista uno strumento di lavoro potente e centralizzato.

Riferimenti

Cito le fonti da cui ho attinto per le informazioni e le librerie utilizzate (oltre a tutte le slide e le spiegazioni forniteci a lezione dal docente):

- Python (Linguaggio): `https://docs.python.org/3/`
- CustomTkinter (Frontend): `https://customtkinter.com/documentation`
- Pandas (Analisi Dati): `https://pandas.pydata.org/docs/`
- Matplotlib (Grafici): `https://matplotlib.org/stable/contents.html`

- WeasyPrint (Generazione PDF): `https://doc.weasyprint.org/en/stable/`
- Jinja2 (Templating): `https://jinja.palletsprojects.com/en/3.1.x/`