Documento di progettazione

MyContactManager

ALESSANDRO PETTI
ALESSANDRO SABIA
ALFONSO VILLANI
GIOVANNI LAMBERTI

Ingegneria del Software
2024/2025

Indice

1. Descrizione del progetto

Il progetto consiste nel realizzare una **rubrica** in grado di contenere dei **contatti** a cui vengono associate le seguenti informazioni:

- nome
- cognome
- numero di telefono
- indirizzo e-mail
- note
- una foto

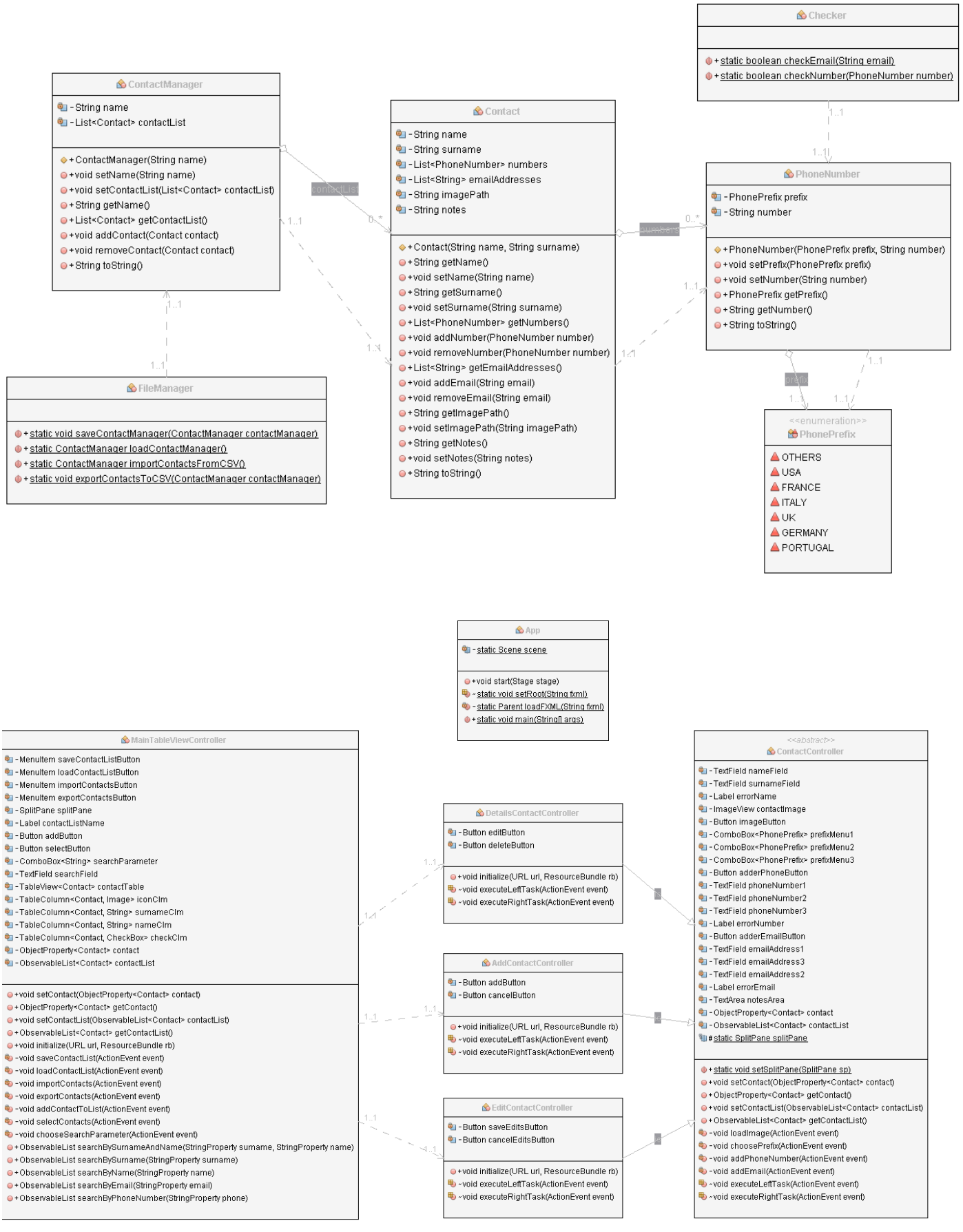
Il numero massimo di indirizzi e-mail e numeri di telefono che possono essere associati ad un contatto varia da zero a tre.
Affinché possa essere salvato in rubrica, è necessario che per ciascun contatto venga indicato almeno **un nome** o **un cognome**.
Durante l'inserimento del numero di telefono e dell'indirizzo email, viene **verificato** che il **formato** sia corretto tramite un metodo apposito, che consente l'inserimento dei dati solo dopo la validazione.
Il numero di telefono può essere preceduto da un **prefisso**, scelto tra quelli disponibili, oppure può non essere preceduto da alcun prefisso.

È possibile rimuovere più contatti tramite la funzione di **selezione multipla** che viene attivata cliccando il tasto apposito; da questo momento l'utente può selezionare i contatti desiderati tramite una colonna di checkbox che appare di fianco ai contatti. Sono offerte funzioni di salvataggio su file binario, caricamento da file binario, importazione su file CSV ed esportazione su file CSV.

La rubrica sarà visualizzabile per intero tramite una **TableView** realizzata su un file **fxml;** su di essa saranno consentite operazioni di ricerca mediante nome, cognome, numero o email al seconda del criterio selezionato dall'utente.
L'interfaccia complessiva sarà realizzata in molteplici fxml.
Tutti i relativi controller condividono uno stesso **SplitPane** caricato, necessario a mostrare un massimo di due pannelli affiancati. Sul pannello di sinistra sarà visualizzata sempre la TableView con la rubrica (MainTableView.fxml).
Sul pannello di destra si alternano i pannelli relativi alla funzionalità alla quale si accede: aggiungi contatto alla rubrica (AddContactView.fxml), visualizza contatto

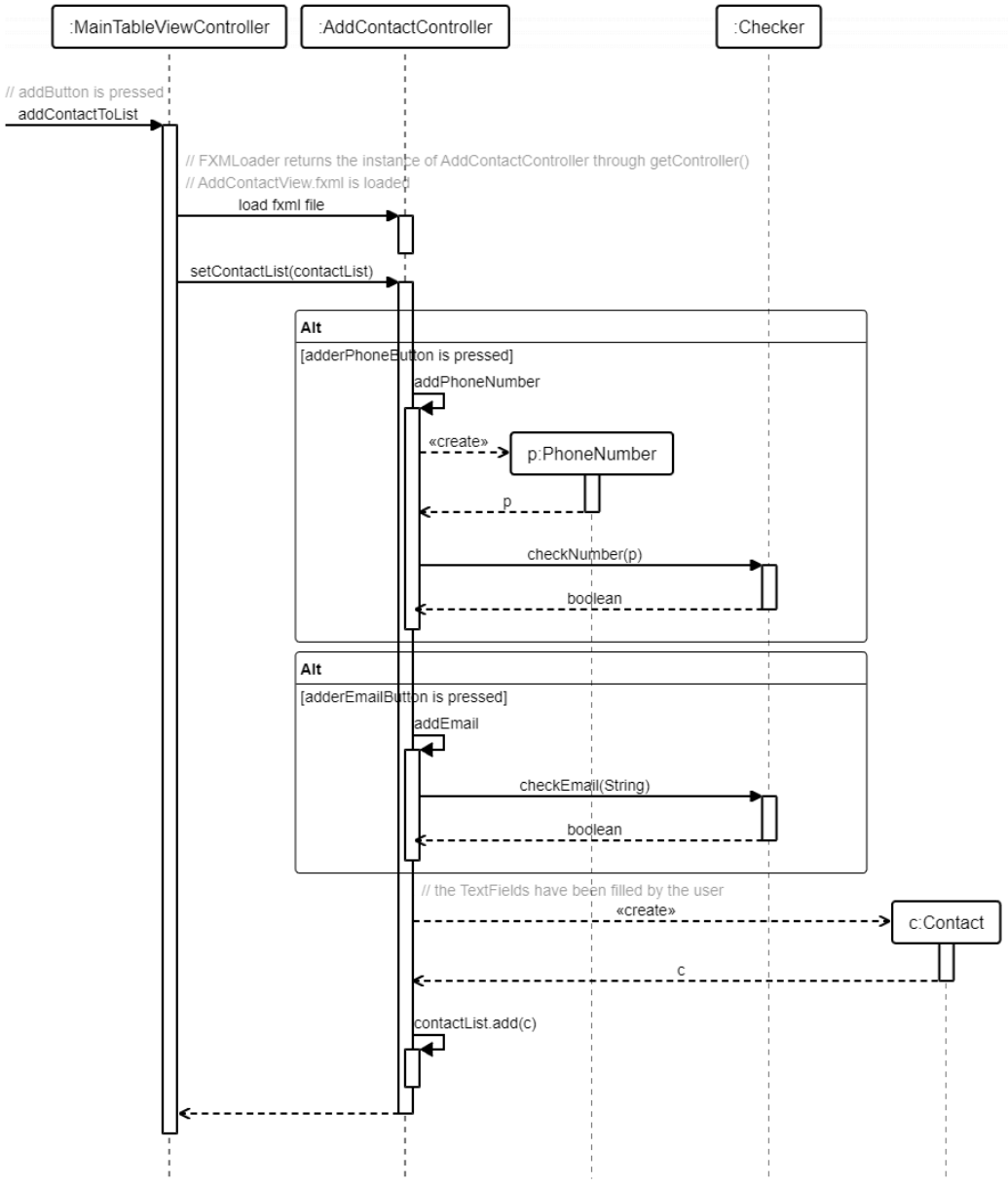
selezionato (DetailsContactView.fxml) e modifica contatto selezionato (EditContactView.fxml).

2. Diagramma delle classi

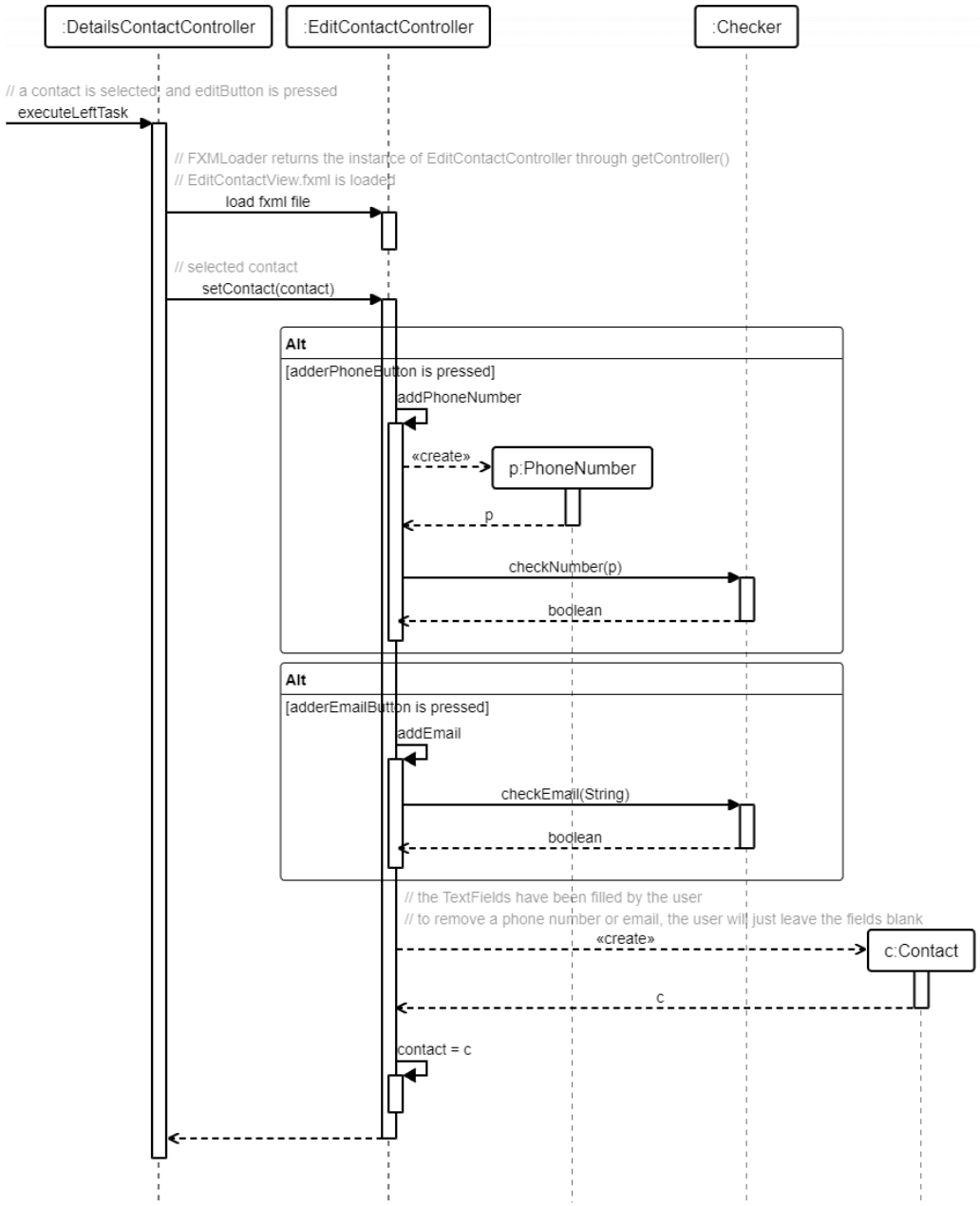


3. Diagrammi di sequenza

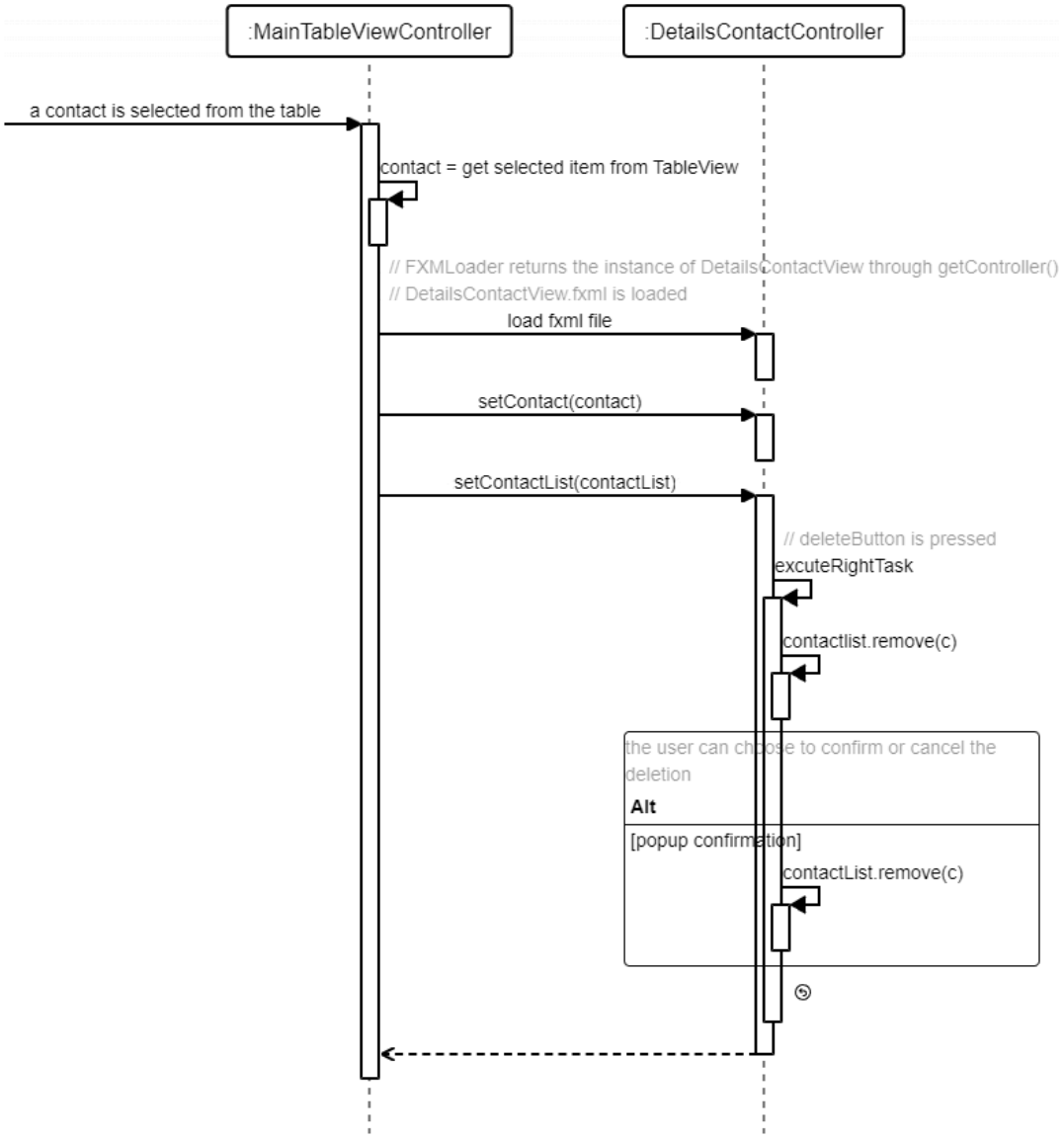
3.1 Aggiungi Contatto



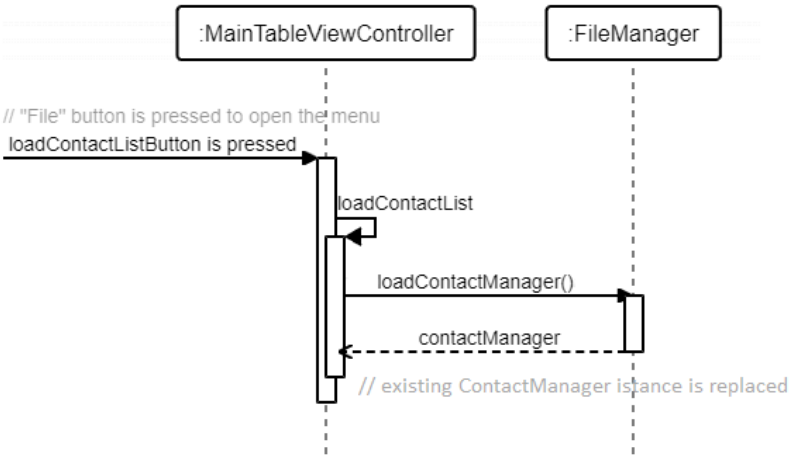
3.2 Modifica Contatto



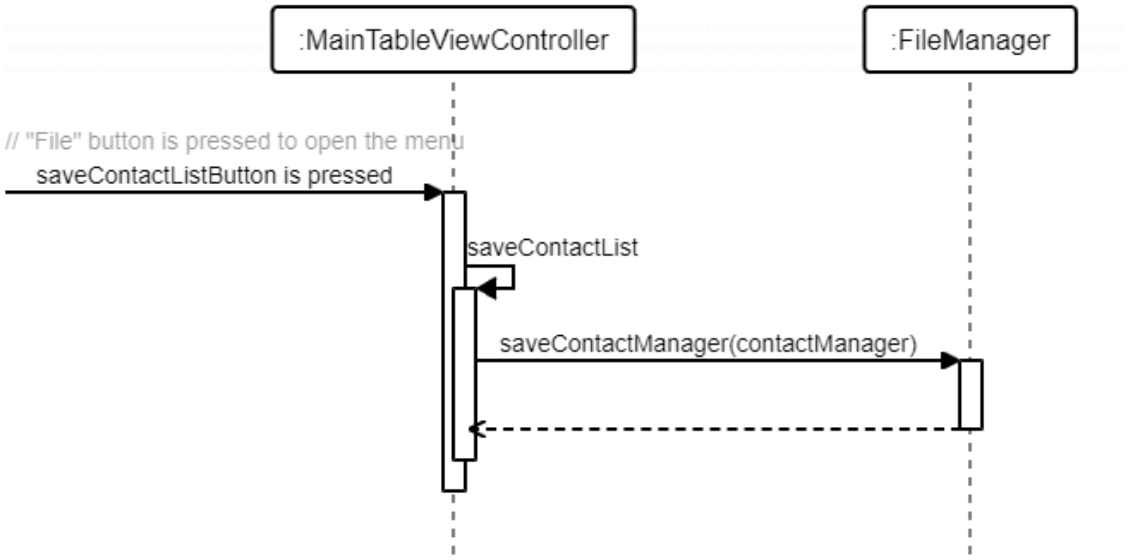
3.5 Rimuovi Contatto



3.1 Carica Rubrica



3.2 Salva Rubrica



4. Coesione

Classi	Livello di coesione	Descrizione
ContactManager	Funzionale	Il modulo offre funzionalità coordinate con l'obiettivo specifico di gestire l'elenco dei contatti.
PhoneNumber	Funzionale	Il modulo esegue tre operazioni: acquisisce due informazioni, le unisce e garantisce una corretta rappresentazione del numero di telefono.
PhonePrefix	Funzionale	Il modulo fornisce funzionalità mirate a svolgere un unico compito specifico: selezionare un prefisso tra quelli disponibili.
FileManager	Funzionale	Il modulo ha un alto livello di coesione poiché si dedica esclusivamente al trasferimento dei dati dei contatti, con tutti i metodi orientati a raggiungere questo unico obiettivo.
ContactController	Funzionale	La classe implementa le funzioni che determinano l'interazione tra l'utente e l'interfaccia grafica.
MainTableViewController	Comunicazionale	La classe si occupa di gestire l'interfaccia principale che presenta la tabella e di disporre le altre sezioni che devono essere aggiunte alla schermata principale su richiesta.
Contact	Funzionale	Tutti i campi sono strettamente correlati al concetto di contatto e i metodi implementati (getter, setter, aggiunta e rimozione di numeri e email) sono le operazioni elementari per la modifica e la lettura di un contatto.

Checker	Temporale	I due metodi booleani vengono chiamati durante la verifica dei dati, controllando indipendentemente la correttezza di email e numero di telefono, pur essendo entrambi legati al contatto.

5. Accoppiamento

Classi	Livello di Accoppiamento	Descrizione
PhonePrefix - PhoneNumber	Per dati	L'interazione diretta con le altre classi è minima dato il passaggio del solo prefisso telefonico alla classe PhoneNumber.
PhoneNumber - Contact	Per dati	La classe PhoneNumber passa alla classe Contact solo le informazioni necessarie al suo funzionamento; che sono rispettivamente il numero e il prefisso telefonico del contatto.
Contact - ContactManager	Per dati	Gestisce un elenco di Contact
ContactController- Checker	Per dati	Le istanze delle classi che estendono ContactController passano i dati di un PhoneNumber alla classe Checker
ContactController - ContactManager	Per dati	Queste classi usano un'istanza di ContactManager
MainTableViewController - ContactManager	Per dati	
ContactManager - FileManager	Per timbro	FileManager richiede un'istanza di ContactManager per scriverne su file i contenuti o l'oggetto stesso.
ContactController - MainTableViewController	Area Comune	Queste classi condividono un'istanza di ContactProperty sotto forma di proprietà osservabile, ma l'esigenza di uno SplitPane come attributo statico alza il livello di accoppiamento

6. Principi di buona progettazione

L'architettura e l'implementazione fornite mirano a garantire gli attributi di qualità, con particolare attenzione ai principi S.O.L.I.D. e ad altre linee guida note. Questo approccio contribuisce a rendere il codice più chiaro, flessibile ed estendibile.

6.1 Principio della Singola Responsabilità (SRP)

Partendo dal *principio di Singola Responsabilità* per modellare ciascuna entità in modo tale che il componente realizzato svolga un singolo compito ben definito.

Il sistema è organizzato in classi con compiti ben definiti. Ad esempio, *Contact* gestisce esclusivamente le informazioni di un singolo contatto, mentre *ContactManager* amministra l'insieme dei contatti. Inoltre, i vari controller (AddContactController, DetailsContactController, EditContactController e MainTableViewController) gestiscono soltanto la logica di presentazione, il che rende più semplice intervenire sul codice, assicurando manutenibilità e riutilizzo.

6.2 Principio Aperto/Chiuso (OCP)

Il *Principio Aperto-Chiuso* è stato cruciale sulle decisioni di implementazione delle operazioni di I/O della rubrica e non lascia che la presentazione di nuovi scenari ne richieda la riprogettazione, ma invece permette di adattarsi a possibili nuove funzionalità.

Infatti nella struttura delle classi è stata utilizzata la pratica di incapsulamento, rendendo inaccessibili dall'esterno i dettagli implementativi della maggior parte delle classi.

6.3 Principio di Sostituzione di Liskov (LSP)

L'assenza di gerarchie di classi complesse implica che il Principio di Sostituzione di Liskov (LSP) non viene particolarmente messo alla prova.

6.4 Principio di Segregazione delle Interfacce (ISP)

Nel nostro progetto non si è reso necessario utilizzare interfacce, in quanto la struttura adottata non ne prevede l'impiego. Di conseguenza, il Principio di Segregazione delle Interfacce non è verificabile. Tuttavia, abbiamo cercato di promuovere chiarezza e modularità, evitando di sovraccaricare le classi con responsabilità eccessive.

6.5 Altre Buone Pratiche Applicate

- **DRY (Don’t Repeat Yourself):**
  L’implementazione evita duplicazioni ridondanti nella logica di dominio. Concentrare le responsabilità in classi specifiche come *Contact* e *ContactManager* non ripetendole in altre classi e far sì che i controller ereditano comportamenti comuni da *ContactController* riducendo la ripetizione di codice.

- **Principio della minima Sorpresa:**
  L’uso coerente di convenzioni, come la notazione CamelCase e nomi intuitivi e per classi e metodi, rende il codice più leggibile e prevedibile, facilitando l’orientamento e la comprensione per chiunque vi metta mano. Il codice è stato scritto in lingua inglese per renderlo comprensibile da un più ampio numero di sviluppatori.

- **Separazione dei Compiti (Separation of Concerns):**
  La distinzione tra gestione dei contatti, logica di presentazione (controller e viste) e servizi di supporto (ad es. Checker e FileManager) aumenta la chiarezza del progetto. Ogni parte può essere modificata o estesa senza influenzare il resto del sistema, migliorando manutenibilità e testabilità.