

Documento Di Design

Gruppo 27

Contents

1	Introduzione	2
2	Tipo di Scomposizione	3
2.1	Scomposizione Architettuale	3
2.2	Scomposizione di Dettaglio	3
3	Diagramma Dei Package	4
4	Diagramma Delle Classi	4
4.1	Analisi delle Classi	4
5	Principi di buona progettazione applicati	7
5.1	Single Responsibility Principle (SRP)	7
5.2	Dependency Inversion Principle (DIP)	7
5.3	Keep It Simple, Stupid (KISS)	7
5.4	You Aren't Gonna Need It (YAGNI)	8
5.5	Interface Segregation Principle (ISP)	8
6	Diagrammi di Sequenza	9
6.1	Avvio dell'applicazione	9
6.2	Aggiunta di un Nuovo Contatto	10
6.3	Modifica di un Contatto	11
6.4	Eliminazione di un Contatto	12
6.5	Salvataggio della rubrica su un file VCard	13
6.6	Caricamento della rubrica da file VCard	14
6.7	Ricerca di un contatto nella rubrica	15
7	Diagramma delle attività	16
8	Versione aggiornata della Matrice di Tracciabilità	17

1 Introduzione

Con la stesura di questo documento ci poniamo l'obiettivo di chiarire , utilizzando i concetti cardine dell'ingegneria del software , quelle che sono state le nostre scelte progettuali per l'implementazione di una rubrica telefonica, le cui funzionalità sono riportate nell' SRS già stilato, che è possibile consultare all'interno della nostra repository

2 Tipo di Scomposizione

Il design del sistema è stato suddiviso in due livelli: scomposizione architetturale e scomposizione di dettaglio.

2.1 Scomposizione Architetturale

A livello architetturale, il sistema è organizzato in moduli(package) separati, ciascuno con responsabilità specifiche. A dettare questa scelta sono stati i vantaggi che se ne possono trarre in termini di modularità del sistema e manutenibilità dello stesso , garantendo anche una migliore lettura del progetto , facilitando la navigabilità nelle diverse directories. I principali pacchetti sono:

- **Models:** Contiene le classi principali per la rappresentazione dei dati , su cui i controller lavorano(pattern MVC) (contiene: **Rubrica**, **Contatto**).
- **Exceptions:** Include classi di eccezioni personalizzate per gestire errori specifici, facilitando la gestione di scenari di errori molto frequenti durante l'utilizzo dell'applicazione (es. **InvalidEmailException**, **InvalidNumberException**).
- **Interfaces:** Definisce contratti che garantiscono flessibilità e possibilità di estensione, garantendo manutenibilità del sistema (es. **InterfaceRubrica**).
- **Controllers:** Contiene i controllers , di cui ognuno è incaricato di gestire una parte specifica dell'interazione utente, coordinando le operazioni sui dati e la loro visualizzazione nelle interfacce grafiche.
- **Managers:** Include **AlertManager** e **ContactManager**, che gestiscono operazioni specifiche e ausiliarie. **AlertManager** si occupa della gestione degli avvisi nel sistema, mentre **ContactManager** gestisce operazioni ausiliarie per l'aggiunta di numero e mail associate ad un contatto stesso

Questa struttura architetturale consente una chiara separazione delle responsabilità tra moduli, riducendo l'accoppiamento tra i componenti principali.

2.2 Scomposizione di Dettaglio

A livello di dettaglio, il design adotta un approccio Object-Oriented per rappresentare le entità del sistema. Ogni elemento viene infatti suddiviso in entità più piccole , conservando le funzionalità.

- Ogni classe incapsula dati e comportamenti specifici, con responsabilità ben definite,rispettando un livello elevato di coesione (ad esempio, **Contatto** rappresenta un singolo contatto, mentre **Rubrica** modella un insieme di contatti).

- Le interfacce vengono utilizzate per definire contratti che separano le funzionalità dall' implementazione (es. `InterfaceRubrica`).
- I metodi sono progettati per garantire coesione elevata e accoppiamento basso, seguendo i principi SOLID.

3 Diagramma Dei Package

Il diagramma UML dei package può essere visualizzato al seguente link: [Diagramma Dei Package](#).

Nella figura viene presentata l'organizzazione in packages del progetto. Si è scelto di suddividere i pacchetti in base alla responsabilità di ognuno, per migliorare la modularità del sistema. Si veda la scomposizione architetturale [qui](#) per i dettagli sulla suddivisione

4 Diagramma Delle Classi

Il diagramma UML delle classi può essere visualizzato qui: [Diagramma Delle Classi](#)

4.1 Analisi delle Classi

Di seguito è riportata una tabella che descrive le caratteristiche di ciascuna classe del sistema, inclusi coesione, accoppiamento, relazione con altre classi e i principi SOLID applicati.

Classe	Coesione	Accoppiamento
Rubrica	Funzionale, comprende tutti metodi per la gestione di contatti , come aggiunta ,rimozione, oltre alle operazioni di salvataggio della stessa su file .vcf	Accoppiamento per timbro con Contatto , poichè Rubrica mantiene un set e una observable list di oggetti Contatto
ContactEmail	Funzionale, perchè contribuisce ad una sola funzione , quella di gestire un indirizzo email	Nessun accoppiamento
ContactNumero	Funzionale, perchè contribuisce ad una sola funzione , cioè quella di gestire un numero di telefono	Nessun accoppiamento

Contatto	Funzionale, perchè il modulo include funzionalità che lavorano insieme per realizzare un singolo Contatto	Accoppiamento per dati con ContactNumero e ContactEmail poiché Contatto fornisce a ContactNumero e ContactEmail le informazioni strettamente necessarie per la gestione dei numeri e delle emails di un contatto. Accoppiamento per dati con ManagerContatti in quanto Contatto fornisce a ManagerContatti le informazioni strettamente necessarie per aggiunta di Numeri ed Email a un contatto.
CheckerEmail	Funzionale, perchè la sua unica funzione è verificare la validità dell'Email	Nessun Accoppiamento
CheckerNumber	Funzionale, perchè la sua unica funzione è verificare la validità del Numero	Nessun Accoppiamento
ManagerContatti	Funzionale, perchè la sua unica funzionalità è l'aggiunta di Numeri e Email al contatto.	Accoppiamento per dati con ContactNumero e ContactEmail, in quanto servono per identificare gli oggetti all'interno delle liste che ManagerContatti utilizza come parametri dei metodi
AlertManager	Funzionale in quanto la sua unica funzione è quella di mostrare messaggi a schermo	Nessun Accoppiamento
BindingController	Funzionale, perchè gestisce esclusivamente i Bindings	Accoppiamento per timbro con ContactFormController e TableViewController , in quanto vengono passate anche informazioni non necessarie a BindingController

TableViewController	Funzionale, infatti la responsabilità di questa classe è quella di gestire la tabella e i dati ad essa associati	Accoppiamento per timbro con InterfaceRubrica , in quanto vengono passate anche informazioni non necessarie a TableViewController. Accoppiamento per controllo con AlertManager
FileController	Funzionale, in quanto si occupa unicamente delle operazioni di salvataggio e caricamento dei dati della rubrica in un file. Non gestisce altre funzionalità	Accoppiamento per timbro con InterfaceRubrica , in quanto vengono passate anche informazioni non necessarie a FileController . Accoppiamento per controllo con AlertManager
Controller	Funzionale, poiché tutte le operazioni sono orientate al flusso del controllo dell'applicazione	Accoppiamento per timbro con i controller secondari , in quanto vengono passate anche informazioni non necessarie a Controller
ContactFormController	Funzionale, in quanto tutte le operazioni sono legate alla gestione dei contatti nella rubrica	Accoppiamento per timbro con InterfaceRubrica , in quanto vengono passate anche informazioni non necessarie a ContactFormController . Accoppiamento per controllo con AlertManager
SearchController	Funzionale, in quanto serve esclusivamente a cercare e filtrare i Contatti	Accoppiamento per timbro con InterfaceRubrica , in quanto vengono passate anche informazioni non necessarie a SearchController

5 Principi di buona progettazione applicati

In questa sezione vengono descritti i principi di buona progettazione adottati nello sviluppo dell'applicazione. Tali principi sono stati seguiti per garantire modularità, manutenibilità e usabilità del software, in linea con le migliori pratiche di ingegneria del software.

5.1 Single Responsibility Principle (SRP)

Il principio di responsabilità singola è stato applicato garantendo che ciascuna classe abbia una sola responsabilità ben definita. Ad esempio :

- La classe **Contact** rappresenta esclusivamente un contatto con i relativi dati (nome, cognome, numero di telefono, ecc.).
- La classe **AlertManager** gestisce i soli messaggi di errore e messaggi di conferma delle eventuali operazioni eseguite dall'utente
- Ogni controller del package **Controllers** ha una propria responsabilità, separata dal compito di ogni altro controller. Il controller principale coordina le interazioni tra l'utente e i controller secondari, che gestiscono le singole funzionalità.

5.2 Dependency Inversion Principle (DIP)

Il principio di inversione delle dipendenze è stato applicato progettando il sistema in modo che i moduli di alto livello dipendano da astrazioni piuttosto che da dettagli di implementazione. In particolare:

- È stata introdotta l'interfaccia **InterfaceRubrica**, che astrae l'accesso ai dati, consentendo di modificare l'implementazione concreta senza dover apportare cambiamenti al codice che la utilizza.
- I controller interagiscono con **InterfaceRubrica** anziché con la classe concreta, riducendo l'accoppiamento e migliorando la manutenibilità del sistema.

5.3 Keep It Simple, Stupid (KISS)

Il principio KISS è stato adottato per garantire la semplicità design e l'implementazione del sistema:

- Ogni modulo è stato progettato per svolgere un compito specifico senza aggiungere funzionalità non necessarie.

- Le interfacce utente e la logica applicativa sono state mantenute chiare , riducendo al minimo il numero di passaggi richiesti per eseguire le operazioni principali,quando possibile.

5.4 You Aren't Gonna Need It (YAGNI)

Il principio YAGNI è stato seguito per evitare l'aggiunta di funzionalità non necessarie o premature, concentrandosi esclusivamente sui requisiti attuali del sistema. In particolare:

- Sono state implementate solo le funzionalità richieste per la gestione dei contatti, senza introdurre estensioni o opzioni inutili che avrebbero potuto complicare il codice.
- Questo approccio ha permesso di mantenere il sistema semplice, leggero e facilmente manutenibile, riducendo il rischio di introdurre complessità non necessarie.

5.5 Interface Segregation Principle (ISP)

Il principio di segregazione delle interfacce è stato applicato per garantire che ogni interfaccia esponga solo i metodi strettamente necessari al contesto in cui è utilizzata. In particolare:

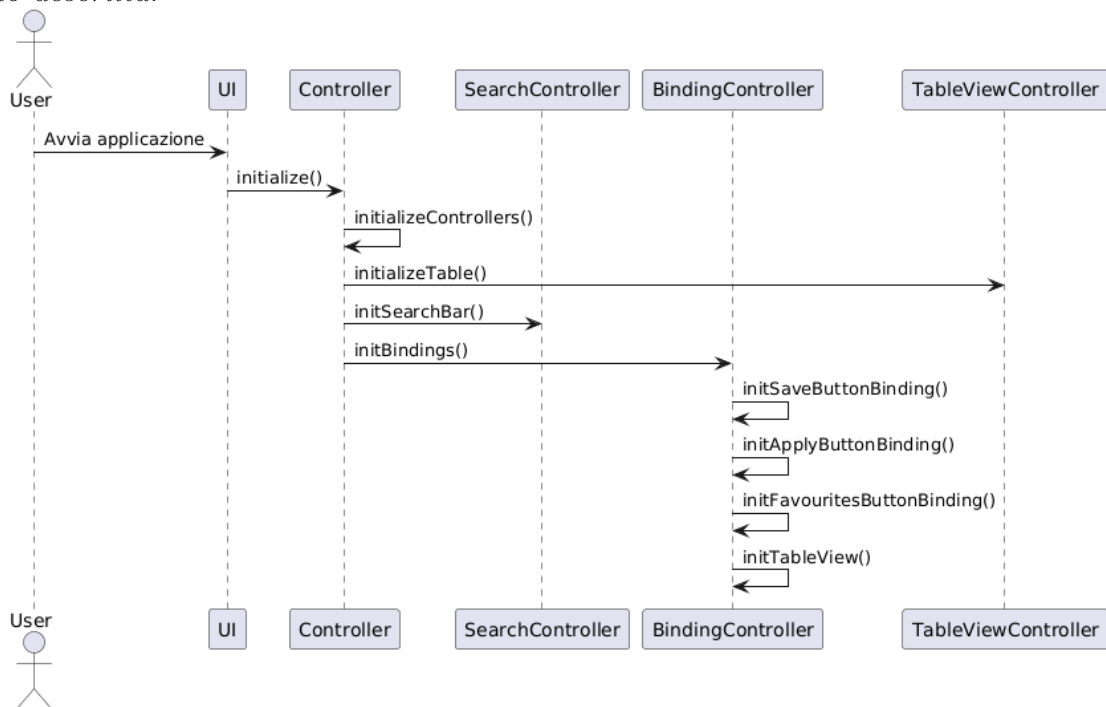
- Le interfacce `CheckerNumber` e `CheckerEmail` espongono solamente i metodi strettamente necessari e coerenti al contesto in cui vengono utilizzati. Di fatti espongono metodi per la validazione di numeri di telefono e email.
- Le classi che utilizzano queste interfacce dipendono solo dai metodi effettivamente necessari alle loro operazioni, migliorando la coesione e riducendo l'accoppiamento.
- Con questo approccio semplifichiamo la manutenibilità del codice

6 Diagrammi di Sequenza

Di seguito sono riportati diversi diagrammi di sequenza che rappresentano i principali scenari di utilizzo della rubrica.

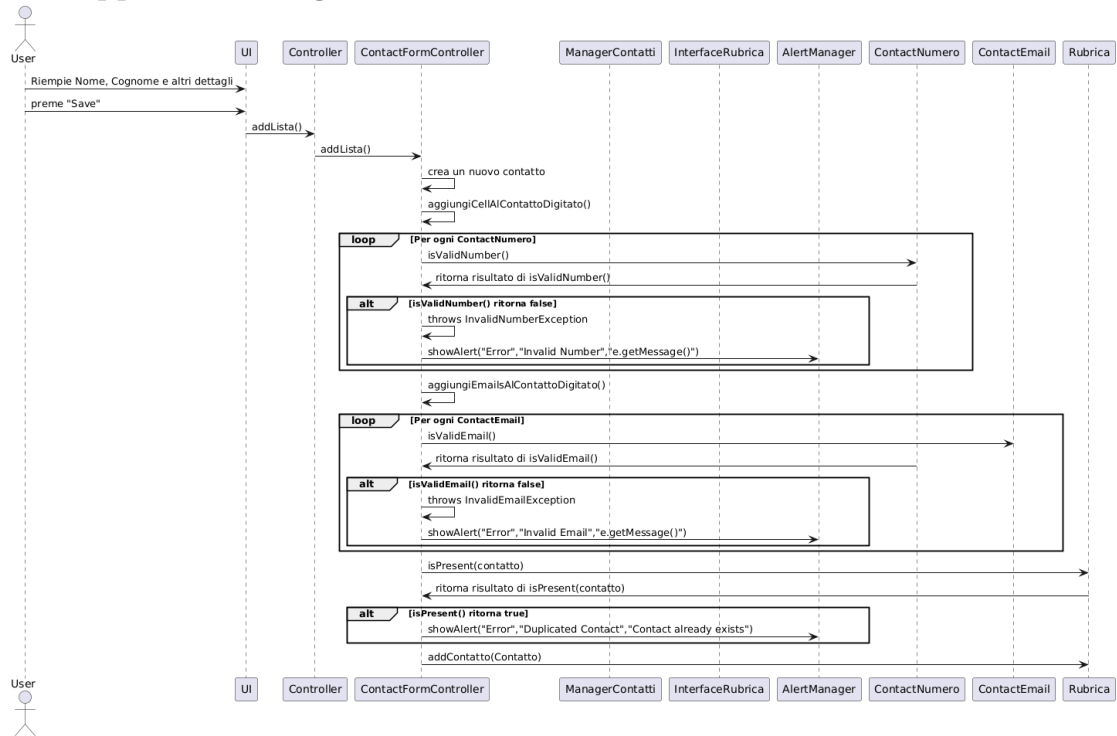
6.1 Avvio dell'applicazione

Il seguente sequence diagram rappresenta il flusso di eventi scatenati all'avvio dell'applicazione. L'attenzione è posta sulla configurazione iniziale dei componenti, in particolare sull'inizializzazione dei bindings tra model, view e controller e , gli event listener per la ricerca ed altre funzionalità. *Si è deciso di chiarire questo diagramma come primo , in quanto nei successivi , si darà per scontato che l'applicazione sia avviata , e che quindi sia avvenuta la sequenza di eventi qui di sotto descritta.*



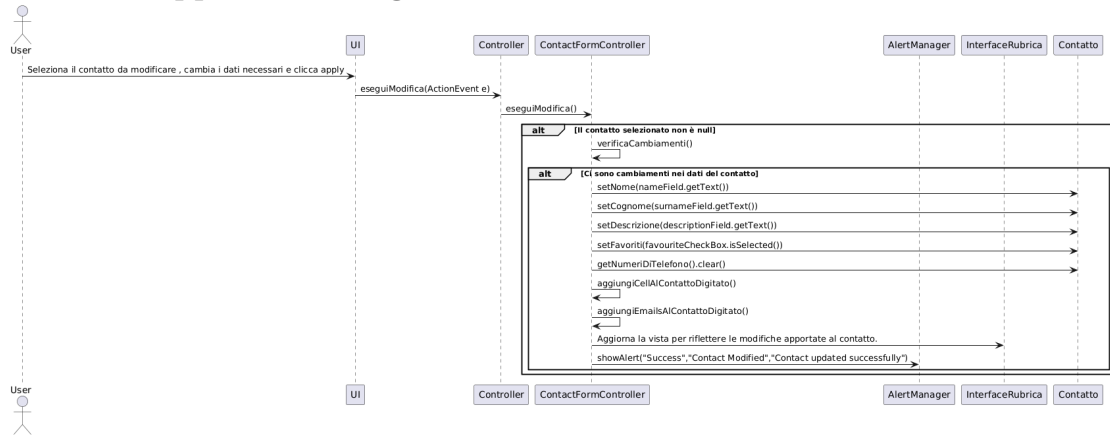
6.2 Aggiunta di un Nuovo Contatto

Il diagramma di sequenza per l'aggiunta di un nuovo contatto mostra come i vari oggetti collaborano per aggiungere un contatto alla rubrica. **Si considera il caso in cui l'applicazione è già avviata.**



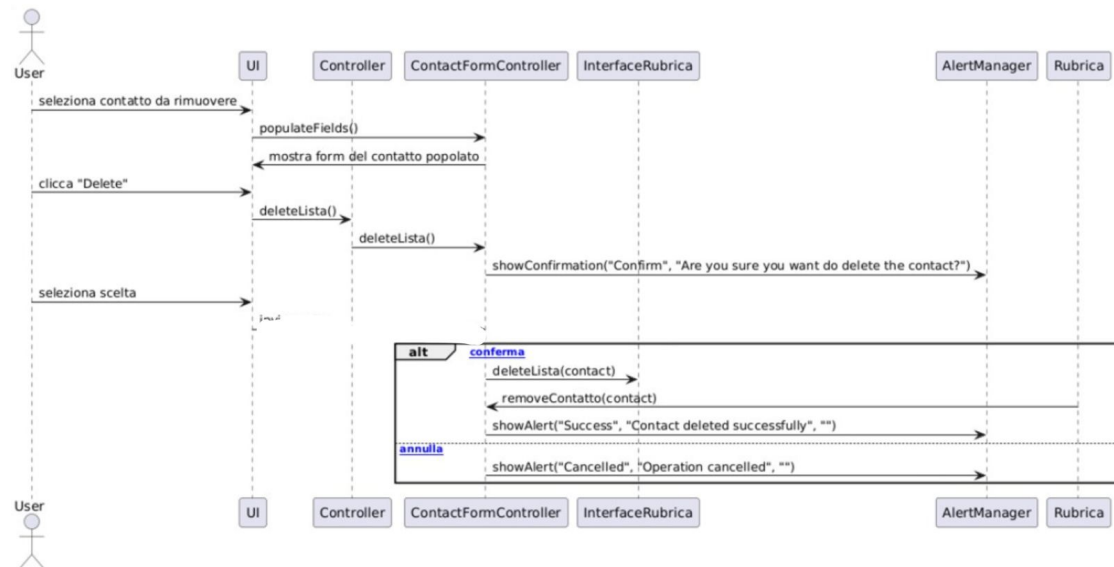
6.3 Modifica di un Contatto

Il diagramma di sequenza per la modifica di un contatto descrive il processo in cui un contatto esistente viene aggiornato con nuove informazioni. **Si considera il caso in cui l'applicazione è già avviata.**



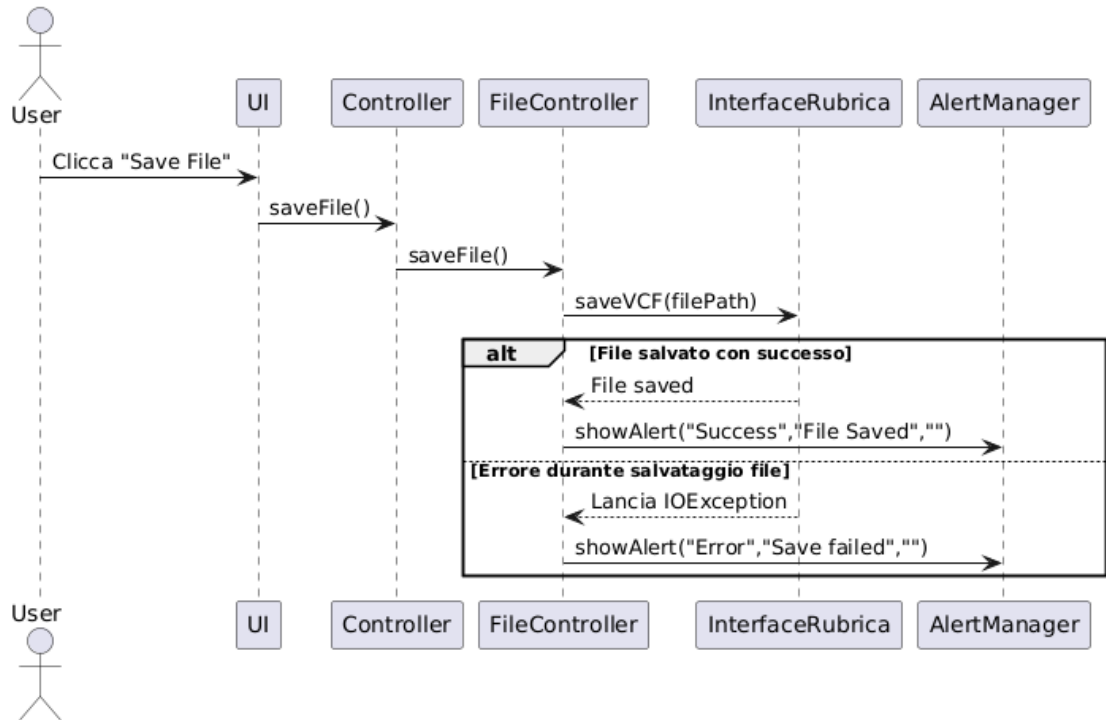
6.4 Eliminazione di un Contatto

Il diagramma di sequenza per l'eliminazione di un contatto mostra il flusso di messaggi quando un contatto viene rimosso dalla rubrica. **Si considera il caso in cui l'applicazione è già avviata.**



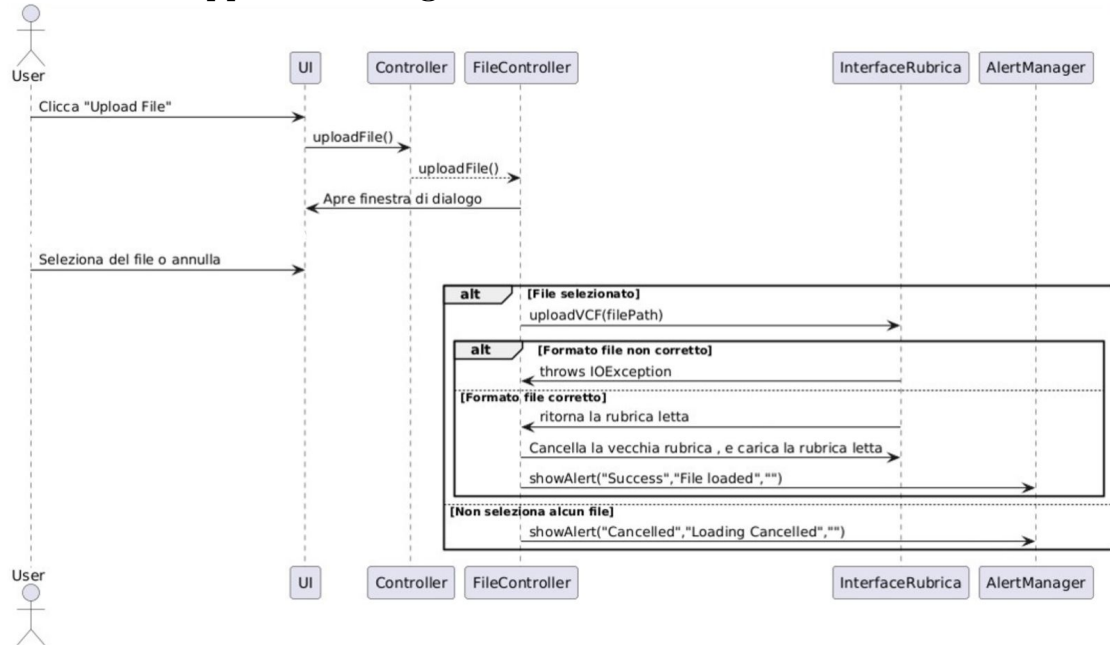
6.5 Salvataggio della rubrica su un file VCard

Flusso per il salvataggio della rubrica su un file VCard. Si considera il caso in cui l'applicazione è già avviata .



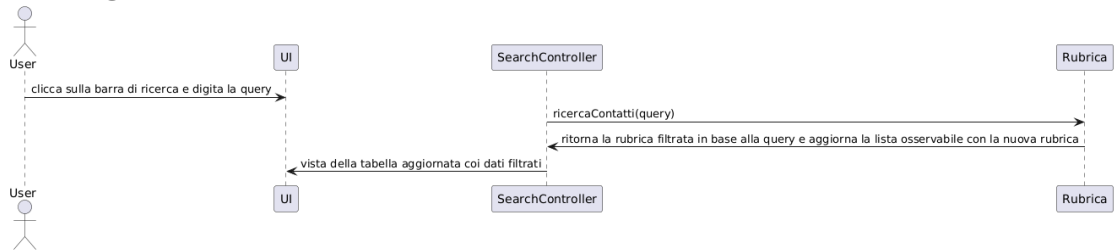
6.6 Caricamento della rubrica da file VCard

Flusso di interazione per il caricamento della rubrica da file VCard. **Si considera il caso in cui l'applicazione è già avviata.**



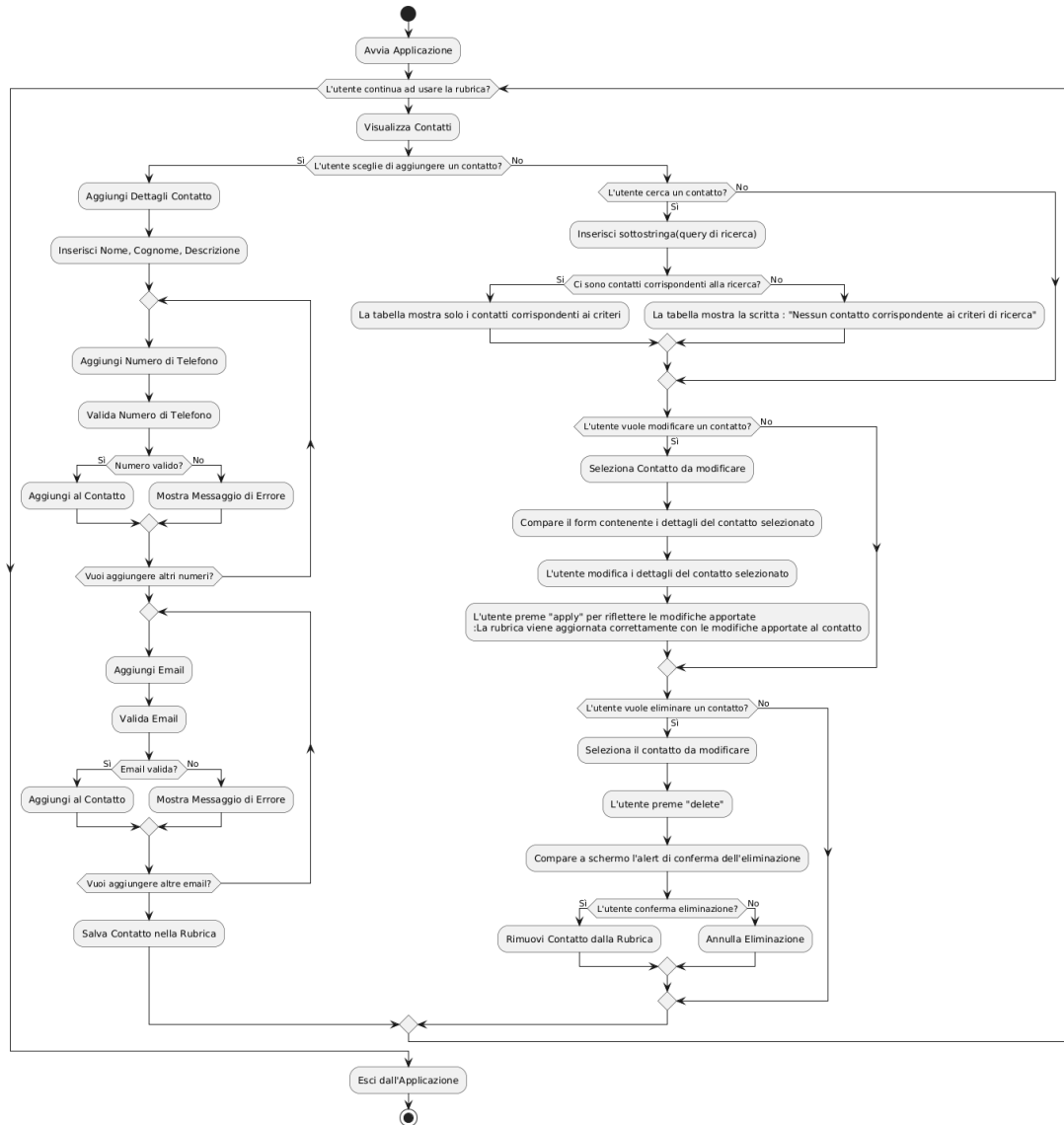
6.7 Ricerca di un contatto nella rubrica

Flusso di interazione per la ricerca di un contatto all'interno della rubrica. Si considera il caso in cui l'applicazione è già avviata, dunque la barra di ricerca è già stata inizializzata.



7 Diagramma delle attività

Con Il seguente diagramma di flusso ci poniamo l'obiettivo di illustrare il funzionamento dell'applicazione sviluppata, descrivendo i passi che l'utente compie durante il suo utilizzo



8 Versione aggiornata della Matrice di Tracciabilità

Requisiti	Design	Test	Status
IF1	Contatto,Rubrica, Controller, Contact-FormController		Started
IF2	Contatto,Rubrica, Controller, Contact-FormController		Started
IF3	Rubrica, ContactForm-Controller		Started
IF4	TableViewController		Started
IF5	TableViewController, ContactFormController		Started
IF6	Rubrica ,SearchBarController		Started
IF7	Rubrica ,FileController		Started
IF8	Rubrica ,FileController		Started
IF9	Rubrica ,Controller, TableViewController		Started
IF10	Rubrica ,Contatto , SearchBarController		Started
UI1	Controller, FileCon- troller, ContactForm- Controller, SearchBar- Controller, TableView- Controller, BindingCon- troller		Started
DF1	Rubrica		Started
DF2	Rubrica,Contatto		Started
DF3	ManagerContatti		Started

Table 2: Matrice di tracciabilità aggiornata