# Documento Di Design

Gruppo 27

## Contents


**1 Introduzione** **2**

**2 Tipo di Scomposizione** **3**
2.1 Scomposizione Architetturale . . . . . 3
2.2 Scomposizione di Dettaglio . . . . . 3

**3 Diagramma Dei Package** **4**

**4 Diagramma Delle Classi** **4**
4.1 Analisi delle Classi . . . . . 4

**5 Diagrammi di Sequenza** **7**
5.1 1. Diagramma di Sequenza: Aggiunta di un Nuovo Contatto . . . . 7
5.2 2. Diagramma di Sequenza: Modifica di un Contatto . . . . . 7
5.3 3. Diagramma di Sequenza: Eliminazione di un Contatto . . . . . 8
5.4 4. Diagramma di Sequenza:Salvataggio della rubrica su un file VCard 9
5.5 4. Diagramma di Sequenza:Caricamento della rubrica da file VCard 9

# 1 Introduzione

Con la stesura di questo documento ci poniamo l'obiettivo di chiarire , utilizzando i concetti cardini 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 migliorate 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 contenuti con Contatto. L'accoppiamento cosi' elevato è dovuto al fatto che rubrica dipende dal contenuto di contatto per le operazioni di ricerca , salvataggio ecc... |
| ContactEmail | Funzionale, perchè contribuisce ad una sola funzione , quella di gestire un indirizzo email | Accoppiamento per dati con Contatto poiché ContactEmail fornisce informazioni strettamente necessarie (email) per la corretta descrizione di un Contatto |

| | | |
|---|---|---|
| ContactNumero | Funzionale, perchè contribuisce ad una sola funzione , cioè quella di gestire un numero di telefono | Accoppiamento per dati con Contatto che fornisce informazioni e metodi strettamente necessari per descrivere un Contatto , ovvero il numero di telefono |
| Contatto | Funzionale, perchè il modulo include funzionalità che lavorano insieme per realizzare un singolo Contatto | Accoppiamento per contenuti con ContactNumero e ContactEmail,in quanto forniscono metodi per la validazione di Numeri e Email di un contatto. Accoppiamento per Dati con ManagerContatti in quanto gli vengono fornite 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 Contatto, in quanto richede solo la lista Email e Numeri Telefonici e il nuovo numero o mail |
| 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 Contenuti con ContactFormController , infatti BindingController ha un riferimento ad un oggetto ContactFormController |

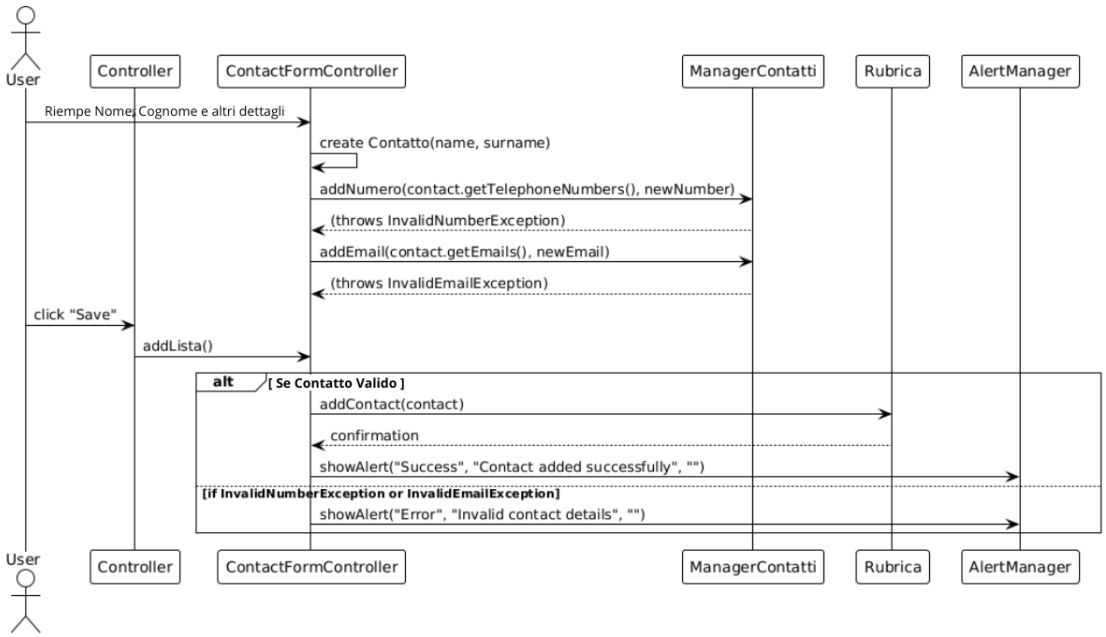
| | | |
|---|---|---|
| TableViewController | Funzionale, infatti la responsabilità di questa classe è quella di gestire la tabella e i dati ad essa associati | Accoppiamento per Contenuti con InterfaceRubrica , in quanto accede direttamente alla struttura dati di InterfaceRubrica in TableViewController . 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 Contenuti con InterfaceRubrica , in quanto accede direttamente alla struttura dati di InterfaceRubrica . Accoppiamento per controllo con AlertManager |
| Controller | Funzionale, poiché tutte le operazioni sono orientate al flusso del controllo dell'applicazione | Accoppiamento per Contenuti con i controller secondari , in quanto accede direttamente alla struttura dati di tutti. |
| ContactFormController | Funzionale, in quanto tutte le operazioni sono legate alla gestione dei contatti nella rubrica | Accoppiamento per contenuti con InterfaceRubrica , in quanto accede direttamente alla struttura dati di InterfaceRubrica . Accoppiamento per controllo con AlertManager |
| SearchController | Funzionale, in quanto serve esclusivamente a cercare e filtrare i Contatti | Accoppiamento per contenuti con InterfaceRubrica , in quanto accede direttamente alla struttura dati di InterfaceRubrica |

# 5 Diagrammi di Sequenza

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

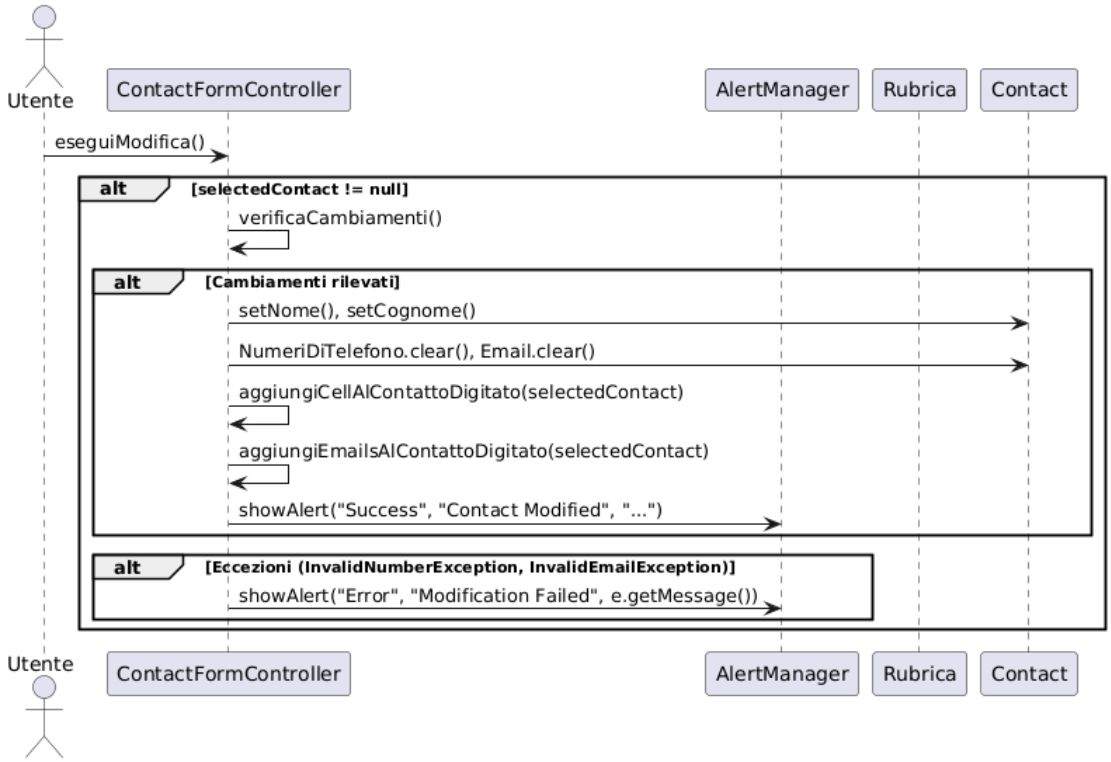
## 5.1 1. Diagramma di Sequenza: Aggiunta di un Nuovo Contatto

Il diagramma di sequenza per l'aggiunta di un nuovo contatto mostra come i vari oggetti collaborano per aggiungere un contatto alla rubrica.



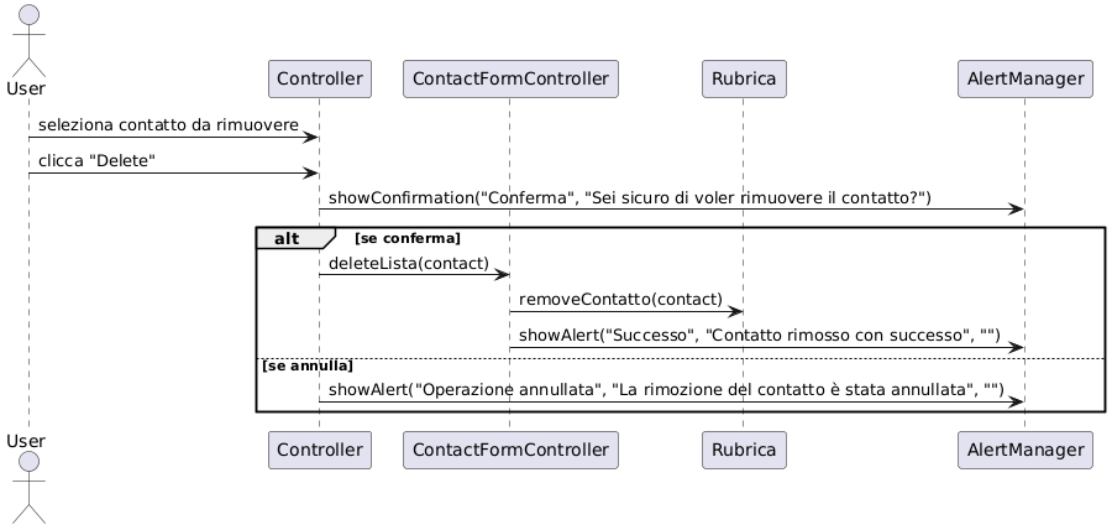

## 5.2 2. Diagramma di Sequenza: Modifica di un Contatto

Il diagramma di sequenza per la modifica di un contatto descrive il processo in cui un contatto esistente viene aggiornato con nuove informazioni.



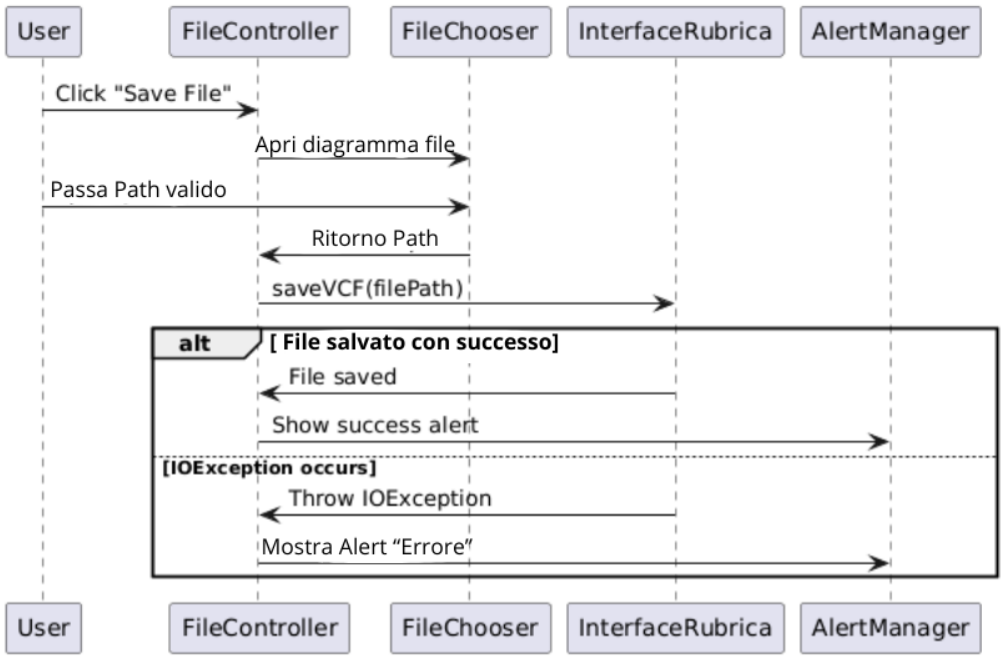

## 5.3 3. Diagramma di Sequenza: Eliminazione di un Contatto

Il diagramma di sequenza per l'eliminazione di un contatto mostra il flusso di messaggi quando un contatto viene rimosso dalla rubrica.



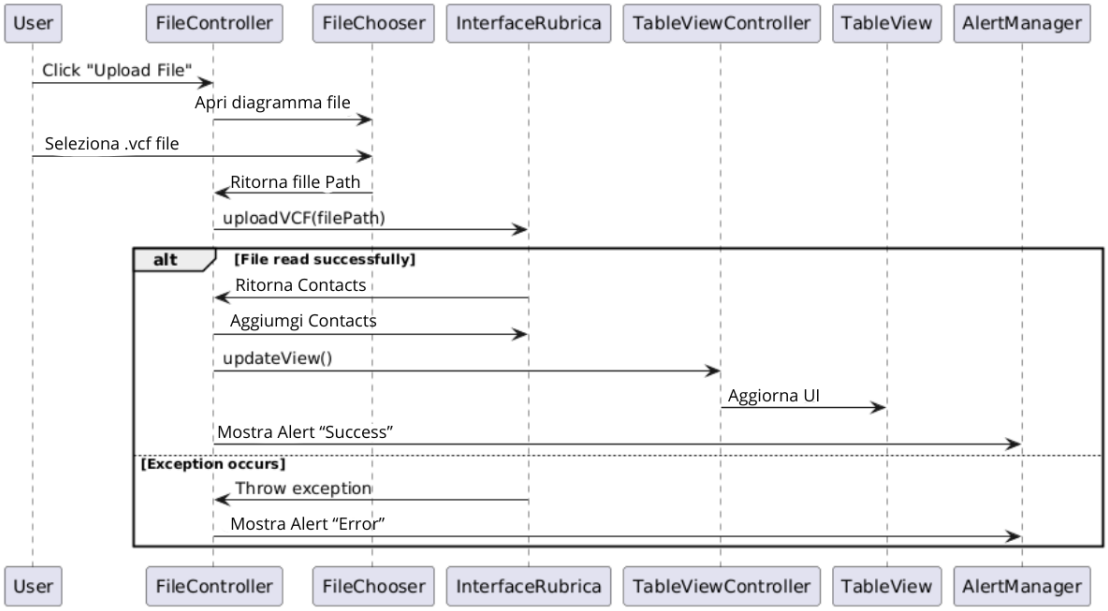
## 5.4 4. Diagramma di Sequenza:Salvataggio della rubrica su un file VCard

Questo diagramma rappresenta il flusso di interazioni per visualizzare i contatti preferiti della rubrica.




## 5.5 4. Diagramma di Sequenza:Caricamento della rubrica da file VCard

Questo diagramma rappresenta il flusso di interazioni per visualizzare i contatti preferiti della rubrica.


User
FileController
FileChooser
InterfaceRubrica
TableViewController
TableView
AlertManager
Click "Upload File"
Apri diagramma file
Seleziona .vcf file
Ritorna fille Path
uploadVCF(filePath)
alt
[File read successfully]
Ritorna Contacts
Aggiumgi Contacts
updateView()
Aggiorna UI
Mostra Alert “Success”
[Exception occurs]
Throw exception
Mostra Alert “Error”
User
FileController
FileChooser
InterfaceRubrica
TableViewController
TableView
AlertManager