**Progettazione**

Indice

[**1 Design architetturale 2**](#_Toc210641987)

[**2 Design dettagliato 2**](#_Toc210641988)

[**2.1 Class diagram 4**](#_Toc210641989)

[**2.1.1 Descrizione dei Class Diagram 5**](#_Toc210641991)

[**2.2 Sequence diagram 10**](#_Toc210641992)

[**3 Activity Diagram 18**](#_Toc210641993)

[**4 Revisione 20**](#_Toc210641994)

[**5 Tabella di Tracciabilità 21**](#_Toc210641996)

# 1 Design architetturale

Per strutturare il sistema ho optato per un approccio **Model View Controller** (MVC), un pattern architetturale in cui il sistema viene separato in tre componenti principali:

**Model**: Gestisce i dati e la logica di business.

**View:** Rappresenta l’interfaccia utente.

**Controller**: Gestisce l’input dell’utente e aggiorna il modello.

Questo approccio consente una chiara separazione delle responsabilità, facilitandone l’utilizzo del codice e rendendo il sistema più manutenibile e scalabile.

# 2 Design dettagliato

In questa fase ho decomposto il sistema finale. Se durante la fase di ingegnerizzazione dei requisiti abbiamo affrontato il “*Cosa*”, cioè cosa dovrebbe fare il sistema, in questa fase affrontiamo il “*Come*”, ovvero come funziona il sistema e i singoli componenti, e come essi si interfacciano e interagiscono tra di loro. Basandomi su un design orientato agli oggetti, si è prodotto:

* Un **Class Diagram** per rappresentare le entità funzionali del sistema.
* Dei **Sequence Diagram** per descrivere appropriatamente la sequenza e il flusso di messaggi che le diverse classi debbano scambiarsi per poter soddisfare gli Use-Case definiti nel documento di ingegnerizzazione dei requisiti.
* Degli **Activity Diagram** per descrivere l’interazione utente-interfaccia grafica.

# 2.1 Class diagram

# 

# 2.1.1 Descrizione dei Class Diagram

|  |  |
| --- | --- |
| **AppRunner** (*main class/ bootstrap class*): è la classe principale che si occupa di avviare l’applicazione. | |
| METODI | DESCRIZIONE |
| main(args: String[]): void | Metodo statico di avvio dell’applicazione. Inizializza i componenti principali e avvia la *MainView*. |
| Initialize(): void | Metodo per creare gli oggetti principali (*RubricaController* e *MainView*), collegandoli tra loro. |
| shutdownHook(): void | Metodo che registra un hook di terminazione (ad esempio per salvare la rubrica o rilasciare risorse prima della chiusura del programma) |

2.1.1.1 Package runner

|  |  |
| --- | --- |
| **Contatto** (*entity class/ core class*): rappresenta l’entità principale della rubrica. E’una classe del dominio (*MODEL*), che incapsula i dati di un singolo contatto | |
| ATTRIBUTI | DESCRIZIONE |
| nome: String | Nome del contatto. Può essere vuoto se è presente il cognome |
| cognome: String | Cognome del contatto. Può essere vuoto se è presente il nome. |
| telefoni: List<String> [0…3] | Lista di massimo 3 numeri telefonici. |
| email: List<String> [0…3] | Lista di massimo 3 indirizzi email. |
| METODI | DESCRIZIONE |
| Contatto(nome: String, cognome: String) | Costruttore che inizializza un nuovo contatto. |
| getNome(): String | Restituisce il nome. |
| getCognome(): String | Restituisce il cognome. |
| getTelefoni(): String | Restituisce i numeri telefonici del contatto. |
| getEmail(): String | Restituisce le email del contatto. |
| setNome(nome: String): void | Modifica il nome. |
| setCognome(cognome: String): void | Modifica il cognome. |
| aggiungiTelefono(numero: String): void | Aggiunge un numero alla lista (rispettando il limite di 3). |
| rimuoviTelefono(numero: String): void | Rimuove un numero della lista. |
| aggiungiEmail(indirizzo: String): void | Aggiunge un’email alla lista (rispettando il limite di 3). |
| rimuoviEmail(indirizzo: String): void | Rimuove un un’email della lista. |
| valida(): boolean | Controlla la correttezza base del contatto(nome/cognome non entrambi vuoti, max 3 numeri, max 3 email). |

2.1.1.2 Package classes

|  |  |
| --- | --- |
| **Rubrica** (*aggregate entity class*): gestisce l’insieme dei contatti. E’ la classe contenitore che fornisce operazioni di ricerca e gestione. | |
| ATTRIBUTI | DESCRIZIONE |
| contatti: List<Contatto> | Lista di tutti i contatti della rubrica. |
| METODI | DESCRIZIONE |
| aggiungiContatto(contatto: Contatto): void | Aggiunge un contatto. |
| modificaContatto(id: int, nuovoContatto: Contatto): void | Sostituisce un contatto esistente con uno nuovo. |
| eliminaContatto(id: int): void | Rimuove un contatto tramite un identificativo. |
| getContatto(id: int): Contatto | Restituisce il contatto corrispondente all’id. |
| getContattiOrdinati(): List<Contatto> | Restituisce i contatti ordinati (alfabeticamente per nome e cognome). |
| cercaContatti(sottostringa: String): List<Contatto> | Ricerca contatti che contengono la sottostringa. |
| isVuota(): boolean | Restituisce “*true*” se la rubrica è vuota. |

|  |  |
| --- | --- |
| **RubricaController** (*controller class*): gestisce la logica dell’applicazione, funge da intermediario tra interfaccia utente (*scinterface*) e modello (*Rubrica*) | |
| ATTRIBUTI | DESCRIZIONE |
| rubrica: Rubrica | Riferimento alla rubrica gestita. |
| validatore: ValidatoreDati | Usato per controllare la correttezza dei dati. |
| gestoreFile: GestoreFile | Usato per leggere/scrivere la rubrica da/su file. |
| METODI | DESCRIZIONE |
| creaContatto(nome, cognome, telefoni, email): void | Crea un nuovo contatto valido e lo aggiunge alla rubrica. |
| rimuoviContatto(indice: int): void | Elimina un contatto dalla rubrica. |
| salvaRubrica(filePath: String): void | Salva la rubrica su file. |
| caricaRubrica(filePath: String): void | Carica la rubrica da file. |
| getContatti(ordinati: boolean): List<Contatto> | Restituisce i contatti ordinati o meno. |
| ricercaContatti(sottostringa: String): List<Contatto> | Ricerca contatti per sottostringa. |

|  |  |
| --- | --- |
| **GestoreFile** (*persistence class*): è la classe principale che si occupa della persistenza su file. | |
| METODI | DESCRIZIONE |
| leggi(filePath: String): Rubrica | Legge i dati dal file e restituisce una rubrica. |
| scrivi(rubrica: Rubrica, filePath: String): void | Salva la rubrica su file. |

|  |  |
| --- | --- |
| **ValidatoreDati** (*checker class*): è la classe di supporto che si occupa della validazione dei dati (Requisto ER.1). | |
| METODI | DESCRIZIONE |
| validaNomeCognome(nome: String, cognome: String): void | Verifica che non siano entrambi vuoti. |
| validaNumeroTelefono(numero: String): void | Controlla che il numero sia solo cifre. |
| validaEmail(email: String): void | Controlla che l’indirizzo email sia nel formato corretto. |
| validaLista(lista: List, max: int): void | Verifica che la lista non superi il limite massimo consentito. |

2.1.1.3 Package exceptions

|  |  |
| --- | --- |
| **RubricaVuotaException** (*system state exception*): viene lanciata quando un’operazione che richiede la presenza di dati (come eliminare un contatto, cercare o salvare la rubrica su file) viene tentata su un oggetto *Rubrica* che non contiene alcun *Contatto.* Serve a prevenire azioni inappropriate su uno stato vuoto. | |
| METODI | DESCRIZIONE |
| RubricaVuotaException(messaggio: String) | Crea l’eccezione specificando un messaggio dettagliato. |
| RubricaVuotaException() | Costruttore di default. |

|  |  |
| --- | --- |
| **ContattoNonTrovatoException** (*query exception*): viene lanciata principalmente dal metodo di ricerca (*ricercaContatti*) nel *RubricaController* o *Rubrica* quando la query eseguita non produce alcun risultato. E’ un metodo specifico per comunicare all’interfaccia utente che la ricerca non ha avuto successo, distinto da un errore generico. | |
| METODI | DESCRIZIONE |
| ContattoNonTrovatoException(messaggio: String) | Crea l’eccezione fornendo il criterio di ricerca fallito |
| ContattoNonTrovatoException() | Costruttore di default. |

|  |  |
| --- | --- |
| **FileNonValidoException** (*persistance exception*): viene lanciata dalla classe *GestoreFile* (nel metodo *leggi*) quando il sistema fallisce nel caricare i dati da un file, tipicamente perché il file è corrotto, vuoto o non è nel formato atteso. | |
| METODI | DESCRIZIONE |
| FileNonValidoException(messaggio: String) | Crea l’eccezione indicando la causa dell’errore di lettura |
| ContattoNonTrovatoException() | Costruttore di default. |

|  |  |
| --- | --- |
| **SpazioInsufficienteException** (*persistence exception*): viene lanciata dalla classe *GestoreFile* (nel metodo *scrivi*) quando si verifica un problema di I/O a livello di sistema operativo, in particolare la mancanza di spazio su disco durante il salvataggio dei dati della rubrica. | |
| METODI | DESCRIZIONE |
| SpazioInsufficienteException(messaggio: String) | Crea l’eccezione per notificare la mancanza di risorse. |
| SpazioInsuffizienteException() | Costruttore di default. |

|  |  |
| --- | --- |
| **FormatoDatiException** (*data fortmat exception*): viene lanciata dal *ValidatoreDati* quando i dati inseriti dall’utente non rispettano le regole di formato (es. un numero di telefono contiene lettere, o un indirizzo email non ha la @). E’essenziale per far rispettare i requisiti di formato (ER.1.1, ER.1.2) | |
| METODI | DESCRIZIONE |
| FormatoDatiException(messaggio: String) | Crea l’eccezione specificando il dato che ha fallito la convalida. |
| FormatoDatiException() | Costruttore di default. |

|  |  |
| --- | --- |
| **CampoObbligatorioException** (*data constraint exception*): viene lanciata dai *ValidatoreDati* o dal metodo *valida(*) in *Contatto* quando un campo o una combinazione di campi essenziali (come nome e cognome) viene lasciato vuoto, violando un vincolo di intergrità (DF.1.1, ER.1.4) | |
| METODI | DESCRIZIONE |
| CampoObbligatorioException(messaggio: String) | Crea l’eccezione per indicare il campo mancante. |
| CampoObbligatorioException() | Costruttore di default. |

|  |  |
| --- | --- |
| **VincoloQuantitàException** (*data constraint exception*): viene lanciata dal *ValidatoreDati* (metodo *validaLista*) quando si tenta di superare il limite massimo di elementi consistenti per una lista, come tentare di aggiungere un quarto numero di telefono o email (ER.1.3). | |
| METODI | DESCRIZIONE |
| VincoloQuantiàException(messaggio: String) | Crea l’eccezione per specificare il limite raggiunto. |
| VincoloQuantitòException() | Costruttore di default. |

2.1.1.4 Package scienterface

|  |  |
| --- | --- |
| **MainView** (*main view class*): rappresenta la componente *View* dell’architettura MVC. Il suo ruolo è unicamente quello di presentare i dati e raccogliere l’input, delegando la logica al *RubricaController*. | |
| ATTRIBUTI | DESCRIZIONE |
| controller: RubricaController | Gestisce la logica applicativa. |
| panelStrumenti: PanelStrumenti | Pannello inferiore con i bottoni di gestione. |
| panelLista: PanelListaContatti | Pannelòp centrale con la lista di contatti. |
| METODI | DESCRIZIONE |
| MainView(controller:RubricaController) | Costruttore che inizializza la view. |
| mostraMessaggio(messaggio: String): void | Mostra i messaggi di sistema o errore. |

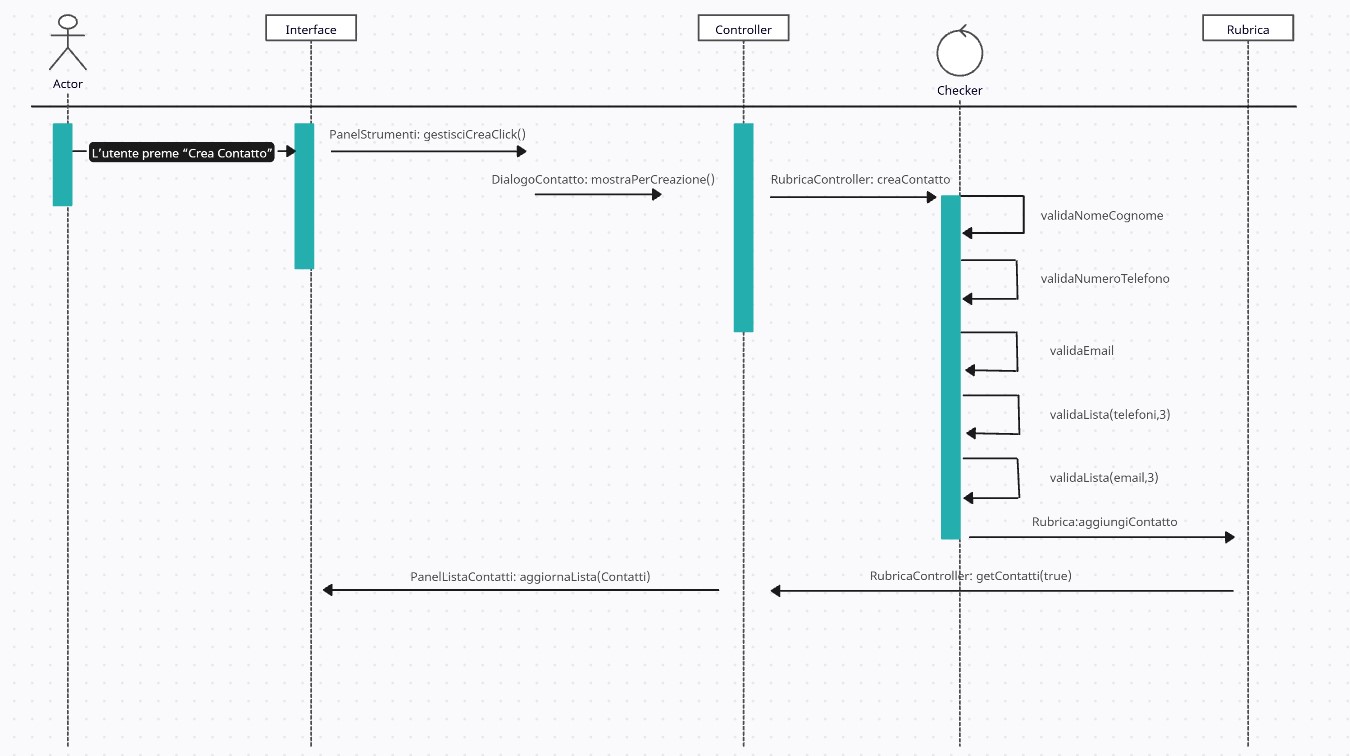
|  |  |
| --- | --- |
| **PanelStrumenti** (*UI subview*): è un pannello che gestisce la sezione dell’interfaccia per le azioni principali: File I/O e Creazione/Modifica. | |
| METODI | DESCRIZIONE |
| GestisciSalvaClick(): void | Gestisce l’azione di salvataggio della rubrica. |
| GestisciCaricaClick(): void | Gestisce l’azione di caricamento della rubrica. |
| GestisciCreaClick(): void | Gestisce la creazione di un nuovo contatto nella rubrica. |
| GestisciModificaClick(): void | Gestisce la modifica di un contatto selezionato. |

|  |  |
| --- | --- |
| **PanelListaContatti** (*UI subview*): gestisce la visualizzazione della lista, la ricerca, la selezione e le azioni sul contatto selezionato (Modifica, Elimina). | |
| METODI | DESCRIZIONE |
| aggiornaLista(contatti:List): void | Aggiorna la lista visualizzata. |
| GestiscRicerca(): void | Gestisce la ricerca testuale dei contatti. |
| GestisciSeelezionaContatto(): void | Gestisce la selezione di un contatto. |
| GestiscieliminaClick(): void | Gestisce la cancellazione di un contatto. |

|  |  |
| --- | --- |
| **DialogoContatto** (*UI dialog class*): è un form modale utilizzato per l’inserimento o la modifica dei dati di un singolo contatto. | |
| METODI | DESCRIZIONE |
| mostraPerCreazione(): Contatto | Mostra un form vuoto per la creazione di un nuovo contatto. |
| mostraPerMofidica(contatto:Contatto): Contatto | Mostra i dati di un contatto esistente per modificarli. |

# 2.2 Sequence diagram

2.2.1 Sequence Diagram “Creare Contatto” UC.1.1

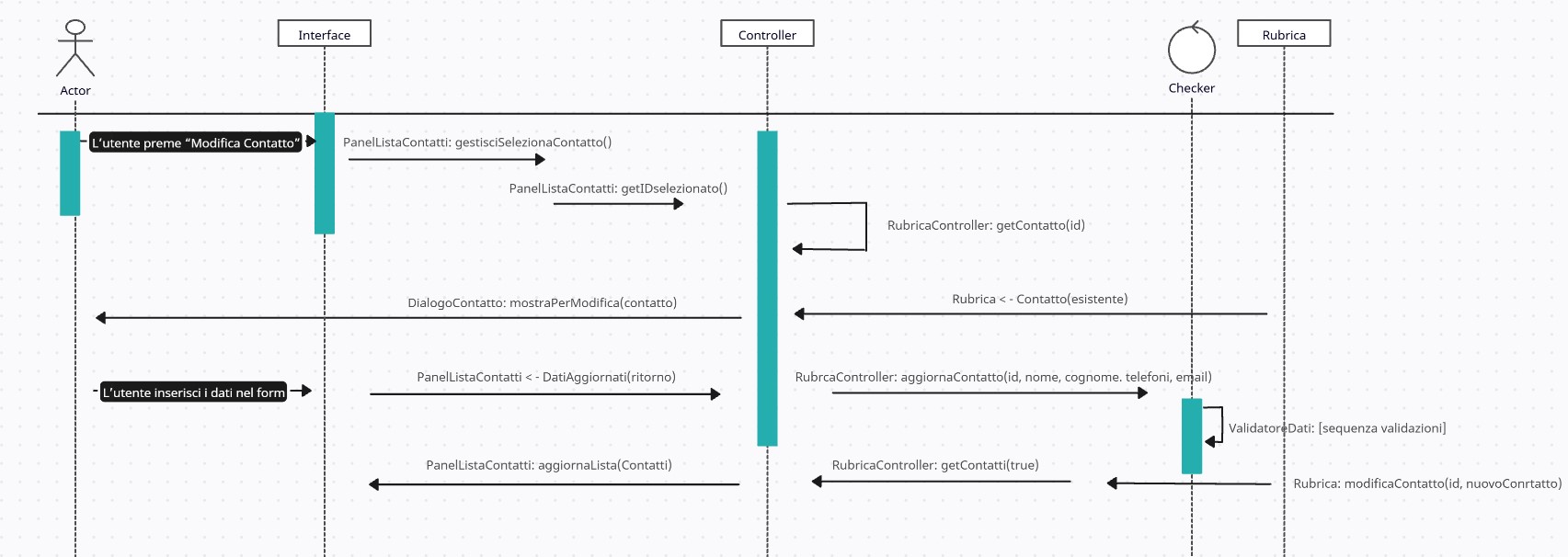


Descrizione Sequence Diagram UC.1.1

Il diagramma di sequenza rappresenta l’interazione che avviene tra l’attore Utente e la rubrica per eseguire l’operazione di creazione di un nuovo contatto.

La sequenza comincia con l’utente che preme il pulsante “*Crea Contatto*” (passo 1 di UC). Il panel, a sua volta, lancia il form modale per l’inserimento dei dati (dal passo 2 al 4 UC). Il form restituirà i dati raccolti quando l’Utente conferma (passo 5 UC) e il Panel delega la richiesta al Controller. Il Controller inizia la validazione (ER.1.4) di ciascun campo del form: valida ogni telefono (ER.1.1) da 0 a 3 volte; valida ogni indirizzo email (ER.1.2) da 0 a 3 volte; verifica i vincoli di quantità per i numeri di telefono e indirizzi email (ER.1.3). Se tutte le validazioni passano, il Controller istanzia un *Contatto* e lo aggiunge al modello (passo 6 UC). Infine il Controller richiederà la lista ordinata che verrà successivamente aggiornata e mostrata dalla User Interface (Postcondizione).

2.2.2 Sequence Diagram “Modifica Contatto” UC.1.2

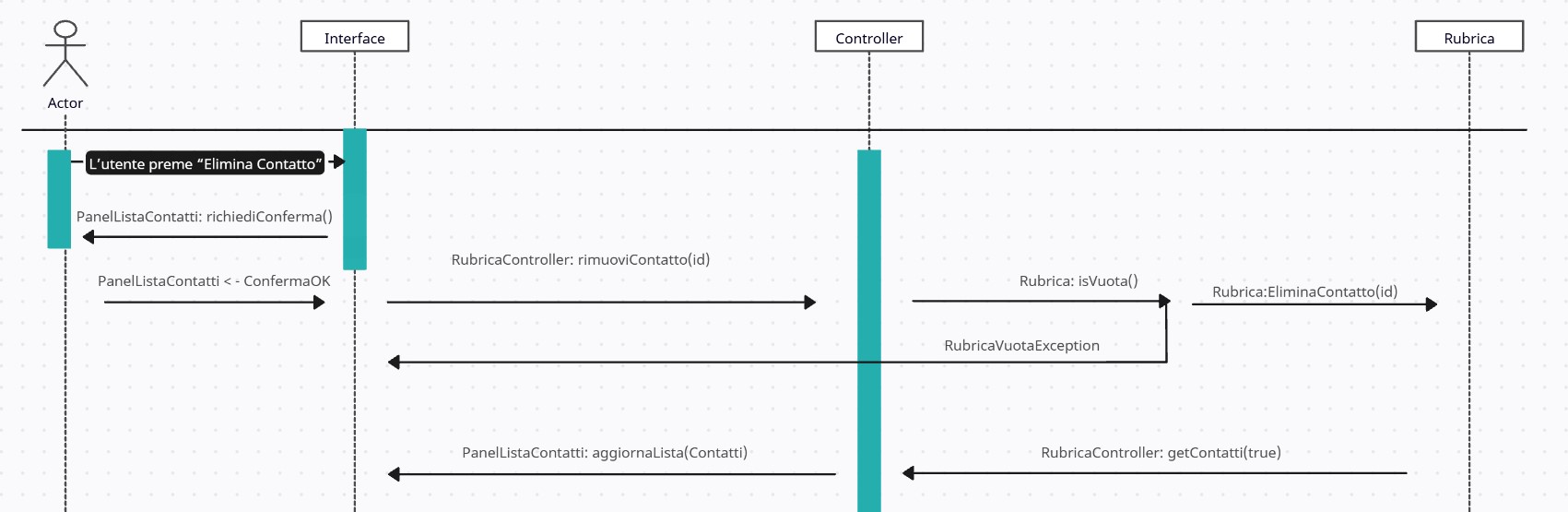


Descrizione Sequence Diagram UC.1.2

Il diagramma di sequenza rappresenta l’interazione che avviene tra l’attore Utente e la rubrica per eseguire l’operazione di “*Modifica Contatto*”.

La sequenza comincia quando l’utente seleziona il contatto (passo 1 e 2 UC), assumendo che l’operazione di modifica sia disponibile dopo la selezione. Il Panel ottiene l’identificatore del contatto selezionato e chiede al Controller di recuperare i dati correnti. Il controller recupera l’oggetto del modello e il Panel lancia il form precompilato con i dati esistenti (passo 3-4 UC). Il form restituisce i dati quando l’Utente conferma le modifiche apportare (passo 5 UC) e il Panel delega la modifica al Controller. Il Controller esegue l’intera sequenza di validazione (ER.1.1, ER.1.2, ER.1.3 e ER.1.4) come in UC1.1(vedere l’immagine precedente per ulteriori dettagli). Se le validazioni passano, il Controller aggiorna il contatto nel modello (passo 6 UC) e richiede la lista aggiornata e ordinata che tramite la User Interface viene mostrata all’utente (Postcondizione).

2.2.3 Sequence Diagram “Eliminare un Contatto” UC.1.3

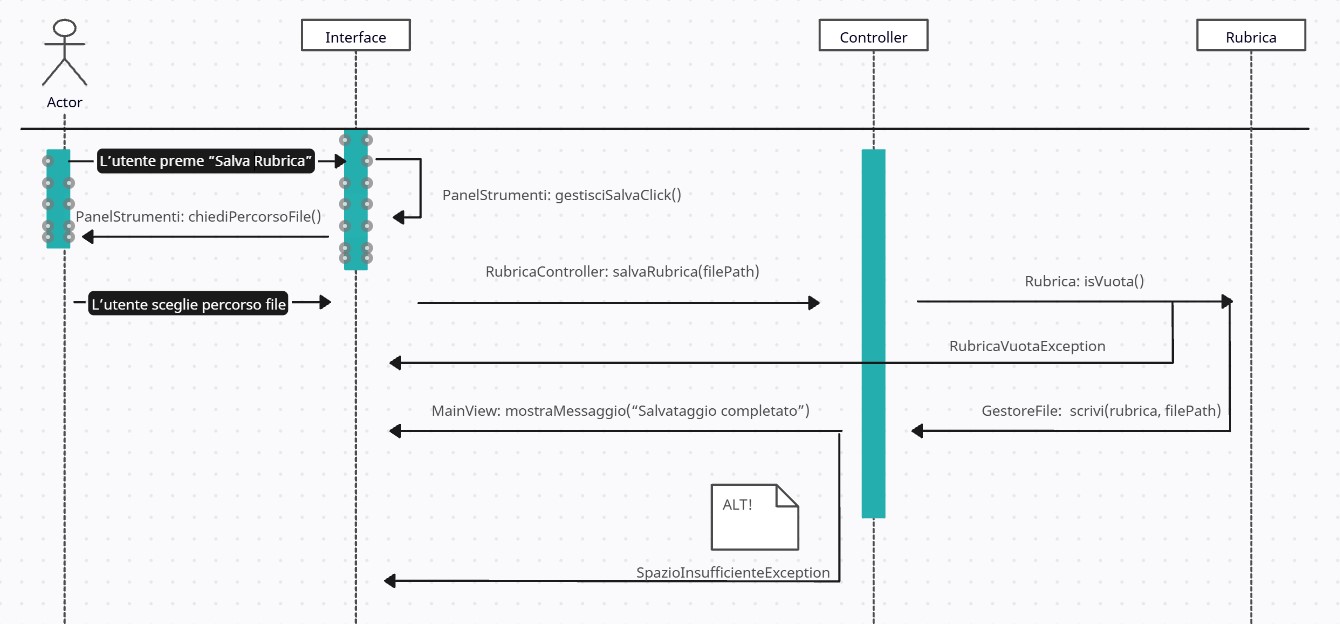


Descrizione Sequence Diagram UC.1.3

Il diagramma di sequenza rappresenta l’interazione che avviene tra l’attore Utente e la rubrica per eseguire l’operazione di “*Elimina Contatto*”.

La sequenza comincia quando l’Utente seleziona un contatto e preme “*Elimina Contatto*” (passo 1-2 UC) con il Panel che invia un messaggio per chiederne la conferma all’utente (passo 3 UC). L’Utente conferma l’eliminazione (passo 4 UC) e il Panel delega la richiesta al Controller, dove a sua volta verifica che la rubrica non sia vuota. Se la rubrica è vuota, non è possibile effettuare l’operazione richiesta quindi il sistema invia *RubricaVuotaException*. Se l’operazione è falsa, il Controller rimuove il contatto (passo 5 UC) e richiede la lista aggiornata che verrà visualizzata grazie all’User Interface (postcondizione).

2.2.4 Sequence Diagram “Salvare Rubrica” UC.3.1

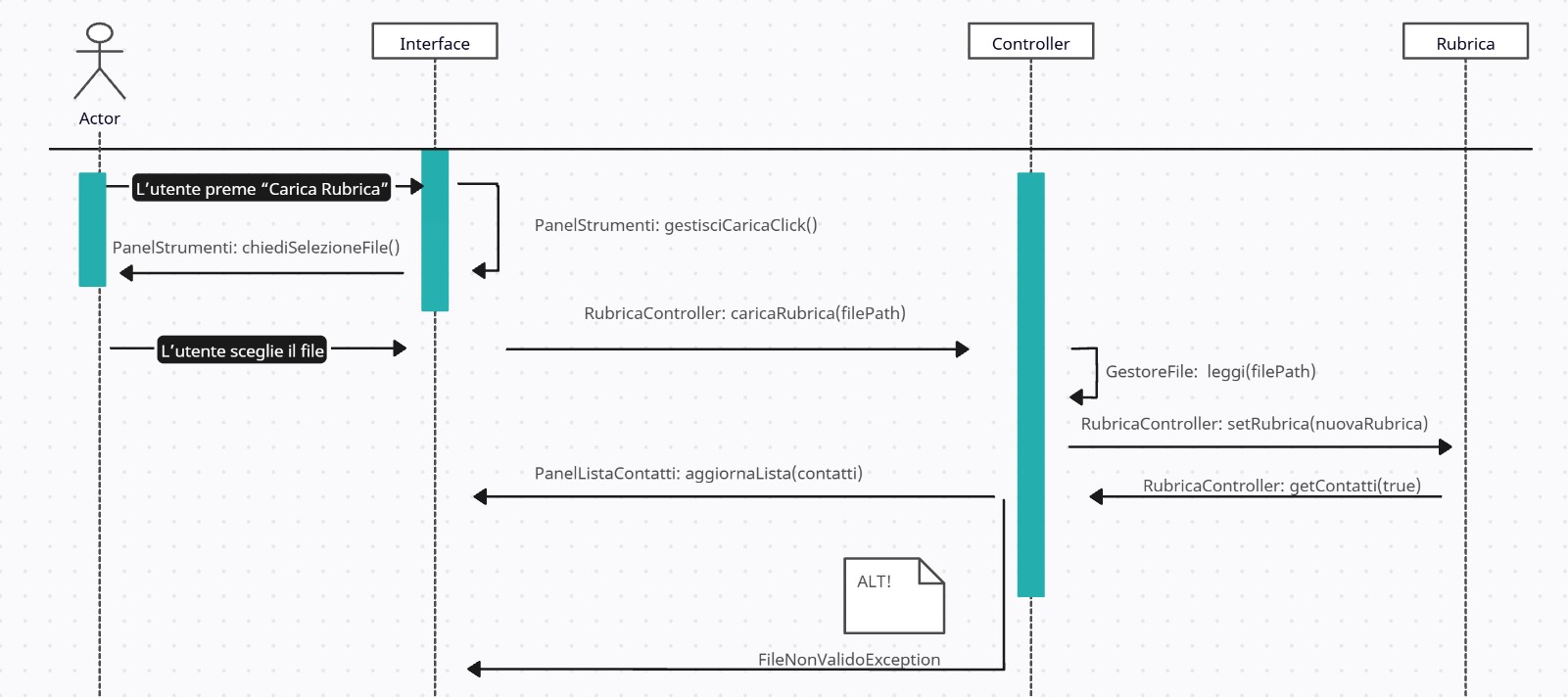


Descrizione Sequenze Diagram UC.3.1

Il diagramma di sequenza rappresenta l’interazione che avviene tra l’attore Utente e la rubrica per eseguire l’operazione di “*Salvare Rubrica*”.

La sequenza comincia quando l’Utente seleziona “*Salva Rubrica*” (passo 1 UC). Il Panel acquisisce il percorso/nome del file (passo 2 UC) e invia una conferma dell’operazione all’utente. L’utente conferma il percorso (passo 3 UC) e il Panel delega l’azione al Controller che verifica che la rubrica non sia vuota. Se la rubrica è vuota viene lanciata *RubricaVuotaException*, altrimenti il Controller delega la scrittura al Gestore File (passo 4 UC). Il Controller notifica la MainView del successo dell’operazione (Postcondizione), ma solamente dopo aver verificato che lo spazio sul disco sia sufficiente al salvataggio. In caso contrario viene inviato SpazioInsufficienteException per notificare il mancato salvataggio.

2.2.5 Sequence Diagram “Carica Rubrica” UC.3.2

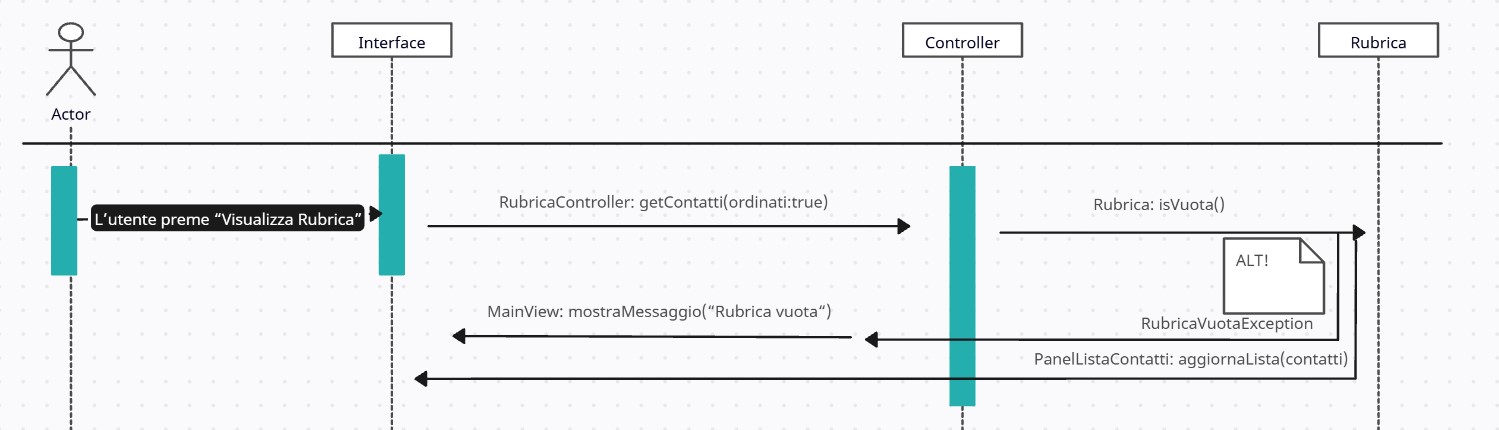


Descrizione Sequence Diagram UC.3.2

Il diagramma di sequenza rappresenta l’interazione che avviene tra l’attore Utente e la rubrica per eseguire l’operazione di “*Carica Rubrica*”.

La sequenza comincia con l’Utente che seleziona “*Carica Rubrica*” (passo 1 UC). Il Panel apre la finestra di dialogo file (passo 2 UC) e l’utente seleziona e conferma il file (passo 3 UC). Il panel delega l’azione al Controller che a sua volta delega la lettura al Gestore File (passo 4 UC). Il Controller riceve la nuova rubrica e la imposta come modello in memoria (passo 5 UC). Il Controller richiede la lista caricata e ordinata che verrà visualizzata grazie alla User Interface, ma questo solamente se il File selezionato dall’utente è valido, altrimenti invia FileNonValidoException e fallisce l’operazione.

2.2.6 Sequence Diagram “Visualizza Rubrica Ordinata” UC.2.1

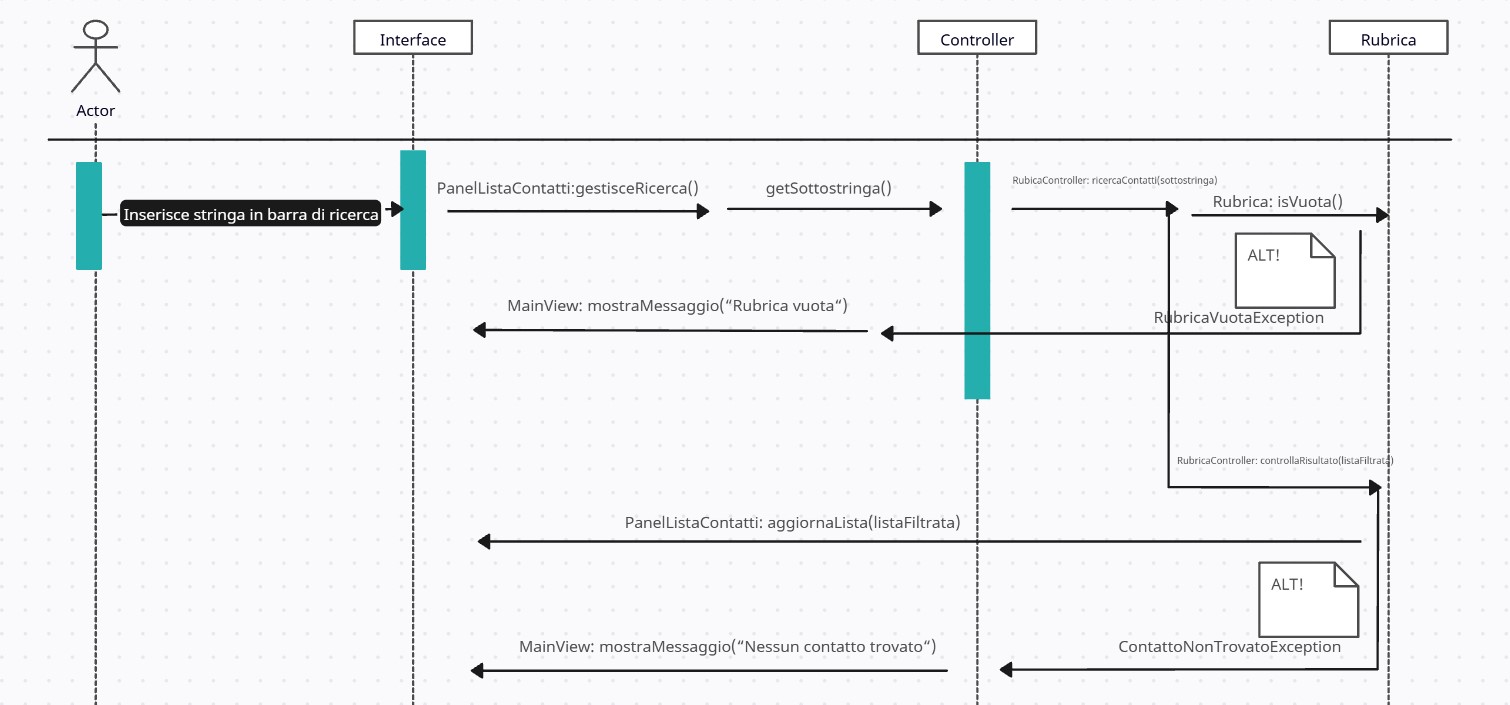


Descrizione Sequence Diagram UC.2.1

Il diagramma di sequenza rappresenta l’interazione che avviene tra l’attore Utente e la rubrica per eseguire l’operazione di “*Visualizza Rubrica Ordinata*”.

La sequenza comincia quando l’utente seleziona l’opzione di “Visualizza Rubrica Ordinata” (passo 1 UC). Il Panel delega al Controller la richiesta dei dati ordinati e successivamente vi è la verifica che la rubrica non sia vuota. In caso affermativo viene inviato RubricaVuotaException che impossibilita la visualizzazione della rubrica ordinata; altrimenti il Controller passa la lista al pannello per la visualizzazione tramite User Interface (passo 2 UC).

2.2.7 Sequence Diagram “Cercare Un Contatto” UC.2.2

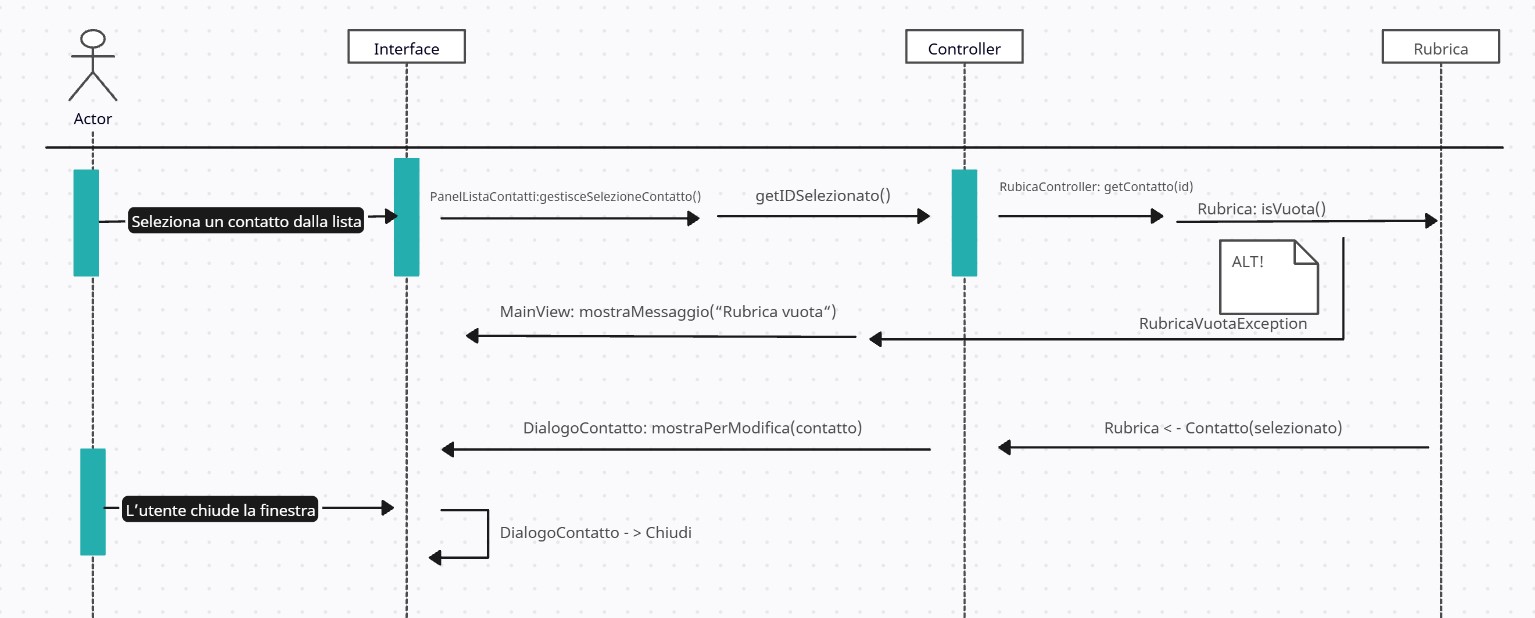


Descrizione Sequence Diagram UC.2.2

Il diagramma di sequenza rappresenta l’interazione che avviene tra l’attore Utente e la rubrica per eseguire l’operazione di “*Cercare un Contatto*”.

La sequenza comincia quando l’utente interagisce con la barra di ricerca e inserisce una sottostringa (passo 1-2 UC). Il panel avvia l’azione recuperando la sottostringa inserita dall’Utente. Il Pannel delega la richiesta di ricerca al Controller, il quale prima di ricercare i contatti nella rubrica verifica che non sia vuota; in caso positivo, viene inviato RubricaVuotaException correlato dal messaggio “Rubrica vuota”, non permettendo all’utente di ricercare alcun contatto. Se la rubrica non è vuota, il Controller chiede al modello di eseguire la logica di confronto (passo 3 UC). Il Controller verifica che la lista restituita dal modello non sia vuota, per poterla restituire al Panel, che grazie all’User Interface mostra i risultati della ricerca. Altrimenti, se la lista è vuota, il Controller lancia ContattoNonTrovatoException correlato dal messaggio “Nessun Contatto trovato”.

2.2.8 Sequence Diagram “Visualizzare Contatto Singolo” UC.2.3



Descrizione Sequence Diagram UC.2.3

Il diagramma di sequenza rappresenta l’interazione che avviene tra l’attore Utente e la rubrica per eseguire l’operazione di “*Visualizzare un contatto singolo*”.

La sequenza comincia quando l’Utente seleziona un contatto dalla lista (passo 1 UC). Il Panel ottiene l’identificatore del contatto e chiede al Controller di recuperare l’oggetto specifico. Questa operazione non è possibile in caso la lista della rubrica non ha nessun contatto e l’utente visualizza “*Rubrica vuota*”, dovuto a *RubricaVuotaException*. Il Controller riceve l’oggetto dal modello e il Panel apre il *DialogoContatto* in modalità solo lettura per mostrare i dettagli completi (passo 2 UC). L’Utente chiude la finestra dei dettagli.

# 3 Activity Diagram

In questo diagramma delle attività si vuole mostrare una panoramica generale delle funzionalità del sistema. Nota: alcune specifiche, relative ai limiti del sistema e al controllo degli errori, sono state omesse per una visualizzazione più semplice e intuitiva.

In particolare:

* **L’utente avvia l’applicazione**, che carica automaticamente la rubrica salvata in precedenza. Se è il primo accesso, la rubrica sarà vuota.
* L’utente può effettuare le seguenti azioni:

1. **Caricare una rubrica**:

* L’utente seleziona un file, compatibile con il sistema.
* Importa la rubrica selezionata

1. **Aggiungere un nuovo contatto**:

* L’utente inserisce i dettagli del contatto (nome, cognome, numero di telefono [0…3], email[0…3]), tenendo conto dei limiti posti dal sistema.
* Il contatto viene salvato e aggiunto alla rubrica.

1. **Modificare un contatto esistente**:

* L’utente seleziona un contatto nella rubrica.
* Effettua le modifiche desiderate (nome, cognome, numeri, email), sempre tenendo conto dei limiti posti dal sistema.
* Le modifiche vengono salvate.

1. **Eliminare un contatto esistente**:

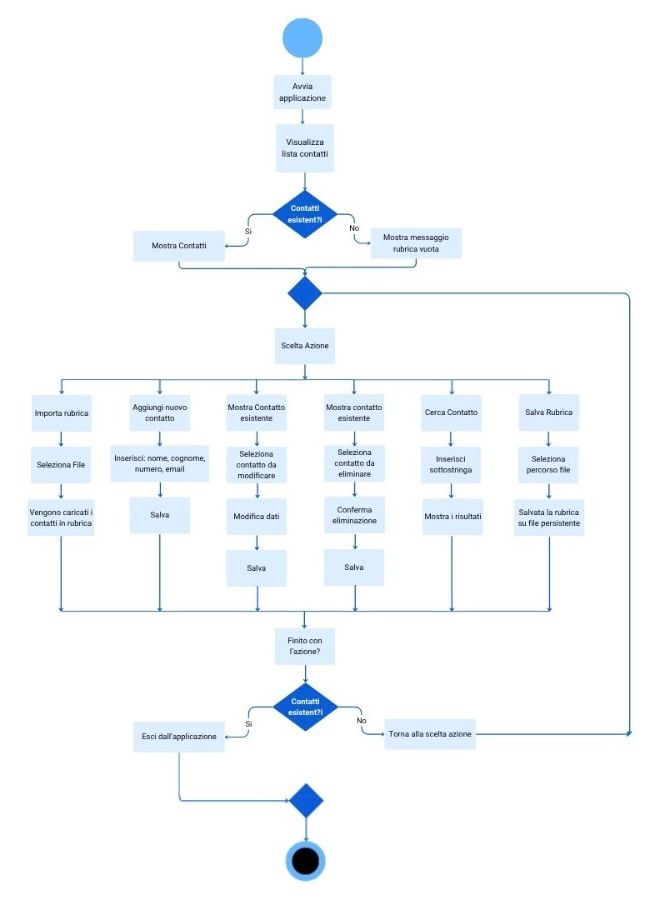
* L’utente seleziona un contatto da eliminare.
* Il sistema chiede una conferma prima di procedere
* Se confermato, il contatto viene eliminato e la rubrica aggiornata.

1. **Cercare un contatto**:

* L’utente inserisce la sottostringa.
* Il sistema mostra i risultati corrispondenti.

1. **Salvare la rubrica**:

* L’utente seleziona il percorso file.
* Se c’è spazio sul disco, l’utente salva la rubrica su file persistente.
* Conclusione dell’azione: Dopo aver completato un’operazione, l’utente può
* **Uscire dall’applicazione**
* **Effettuare un’altra operazione**



# 4 Revisione

# A seguito di una revisione completa della documentazione, ho apportato le seguenti modifiche:

* + - **Correzione Ortografica:** Sono stati eliminati tutti gli errori ortografici e di battitura.
    - **Aggiunta la classe AppRunner:** per separare la logica del sistema dal codice d’avvio del sistema.

# 5 Tabella di Tracciabilità

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ID Requisito | Progettazione | Implementazione | Testing | Requisiti correlati |
| IF.1.1 | RubricaController, Rubrica, SD\_UC.1.1 |  |  | DF.1.1, DF.1.2, DF.1.3, ER.1.1, ER.1.4, US.1.3 |
| IF.1.2 | RubricaController, Rubrica, SD\_UC.1.2 |  |  | DF.1.1, DF.1.2, DF.1.3, ER.1.1, ER.1.3, US.3.2 |
| IF.1.3 | RubricaController, Rubrica, SD\_UC.1.3 |  |  | ER.3.1 |
| DF.1.1 | Contatto, ValidatoreDati, SD\_UC.1.1, SD\_UC.1.2 |  |  | IF.1.1, IF.1.2, ER.1.4, IF.3.1, IF.3.2 |
| DF.1.2 | Contatto, ValidatoreDati, SD\_UC.1.1, SD\_UC.1.2 |  |  | IF.1.1, IF.1.2, ER.1.1, ER.1.3 |
| DF.1.3 | Contatto, ValidatoreDati, SD\_UC.1.1, SD\_UC.1.2 |  |  |  |
| IF.2.1 | RubricaController, GestoreFile, SD\_UC.3.1 |  |  | ER.2.2. ER.2.3, US.1.2 |
| IF.2.2 | RubricaController, GestoreFile, SD\_UC.3.2 |  |  | ER.2.1, US.1.1 |
| IF.3.1 | Rubrica, SD\_UC.2.1, SD\_UC.2.3 |  |  | IF.3.2, ER.3.1, US.2.1 |
| IF.3.2 | Rubrica, SD\_UC.2.2 |  |  | ER.3.1, ER.3.2, US.2.2 |
| US.1.1 | PanelStrumenti |  |  | IF.2.2 |
| US.1.2 | PanelStrumenti |  |  | IF2.1 |
| US.1.3 | PanelStrumenti, DialogoContatto |  |  | IF1.1 |
| US.2.1 | PanelListaContatti |  |  | IF3.1 |
| US.2.2 | PanelListaContatti |  |  | IF.3.2 |
| US.3.1 | RubricaController. PanelListaContatti |  |  | ER.3.1 |
| US.3.2 | PanelListaContatti, DialogoContatto |  |  | IF.1.2 |
| ER.1.1 | ValidatoreDati, MainView, SD\_UC.1.1, SD\_UC.1.2 |  |  | DF.1.2, IF.1.1, IF.1.2 |
| ER.1.2 | ValidatoreDati, MainView, SD\_UC.1.1, SD\_UC.1.2 |  |  | DF.1.3, IF.1.1, IF.1.2 |
| ER.1.3 | ValidatoreDati, MainView, SD\_UC.1.1, SD\_UC.1.2 |  |  | DF.1.2, DF.1.3, IF.1.1, IF.1.2 |
| ER.1.4 | ValidatoreDati, MainView, SD\_UC.1.1, SD\_UC.1.2 |  |  | DF.1.1, IF.1.1 |
| ER.2.1 | GestoreFile, MainView, SD\_UC.3.2 |  |  | IF.2.2 |
| ER.2.2 | RubricaController, MainView, SD\_UC.3.1, SD\_UC.2.1, SD\_UC.2.2, SD\_UC.2.3 |  |  | IF.2.1 |
| ER.2.3 | GestoreFile, MainView, SD\_UC.3.1 |  |  | IF.2.1 |
| ER.3.1 | RubricaController, MainView, SD\_UC.2.1 |  |  | IF.1.3, IF.3.1, IF.3.2, US.3.1 |
| ER.3.2 | RubricaController, MainView, SD\_UC.2.2 |  |  | IF.3.2 |