Ingegneria del Software

Corso di Laurea in Ingegneria Informatica

Prof. Nicola Capuano

Sviluppo di un'applicazione software per la gestione di una rubrica, finalizzata all'organizzazione e alla conservazione dei contatti telefonici e delle e-mail.

Progettazione

Davide PERNA RUGGIERO
October 12, 2025

.DIEM

1 Introduzione

L'applicazione è stata progettata secondo il paradigma *Object-Oriented* e segue il pattern architetturale **Model-View-Controller** (**MVC**), che separa la logica applicativa dai dati e dalla presentazione. Questa scelta consente di realizzare un software ben strutturato, mantenibile ed estendibile.

1.1 Pattern architetturale

MVC
Il pattern architetturale usato prevede la suddivisione dell'applicazione in tre componenti principali:

- **Model**
  Che si occupa dei dati dell'applicazione;
- **View**
  Che è responsabile della presentazione dei dati (interfaccia);
- **Controller**
  Intermediario tra *Model* e *View*; riceve gli input e agisce di conseguenza.

La scelta di questo pattern architetturale è affine all'utilizzo di un linguaggio di programmazione orientato agli oggetti quale Java (per la cura dell'interfaccia - GUI - JavaFX).

1.2 Model

Come anticipato, al *Model* appartengono le classi relative ai dati dell'applicazione

- **Contact**: modella un singolo contatto della rubrica, con i relativi attributi identificativi e informativi.
- **AddressBook**: gestisce l'insieme dei contatti in memoria, fornendo le operazioni di ricerca, filtro, ordinamento e modifica.
- **FilePersistence**: si occupa della gestione dei dati su file locale, garantendo il salvataggio, l'aggiornamento e il caricamento persistente della rubrica.

1.3 View

Al componente View appartengono le classi relative all'interfaccia e alla presentazione dell'applicazione, nonché l'usabilità della stessa.

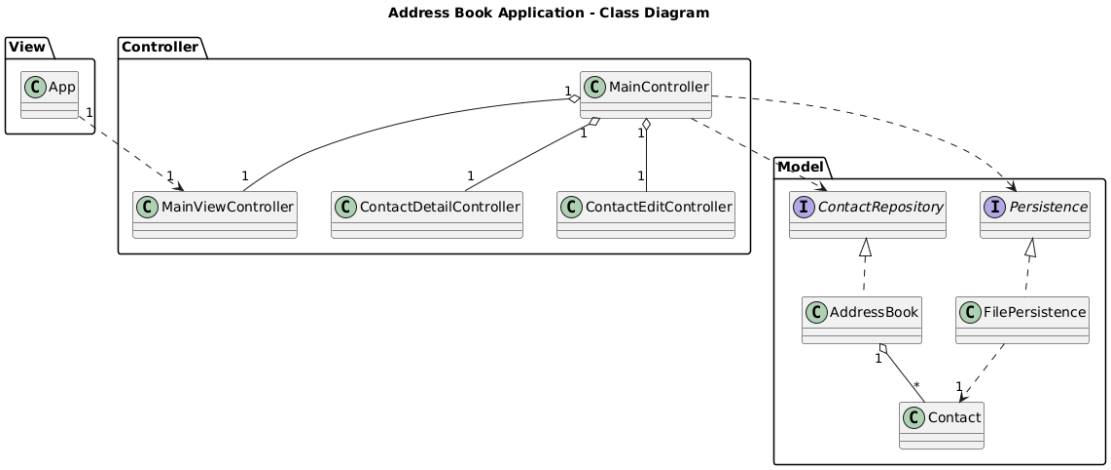
- **MainView**: mostra la rubrica completa, con la tabella dei contatti e i controlli di ricerca, filtro e ordinamento.
- **ContactDetailView**: visualizza i dettagli di un singolo contatto, con opzioni per modificarlo o eliminarlo.
- **ContactEditView**: fornisce il form di creazione o modifica di un contatto, gestendo l'inserimento e la validazione dei dati.

1.4 Controller

Infine, al componente *Controller* appartengono le classi responsabile di far funzionare *il Model* e *la View* tra loro, reagendo a modifiche nel *Model* o nelle interazioni con l'utente.

- **MainController**: rappresenta il punto di coordinamento logico dell'applicazione, orchestrando le operazioni tra modello, persistenza e vista.
- **MainViewController**: controlla la finestra principale e inoltra al *MainController* le azioni generate dall'utente nell'interfaccia principale.
- **ContactDetailController**: gestisce la vista di dettaglio di un contatto, aggiornando i dati mostrati e coordinando le azioni di modifica o eliminazione.
- **ContactEditController**: governa la finestra di modifica o creazione di un contatto, raccogliendo i dati inseriti e validandoli prima del salvataggio.

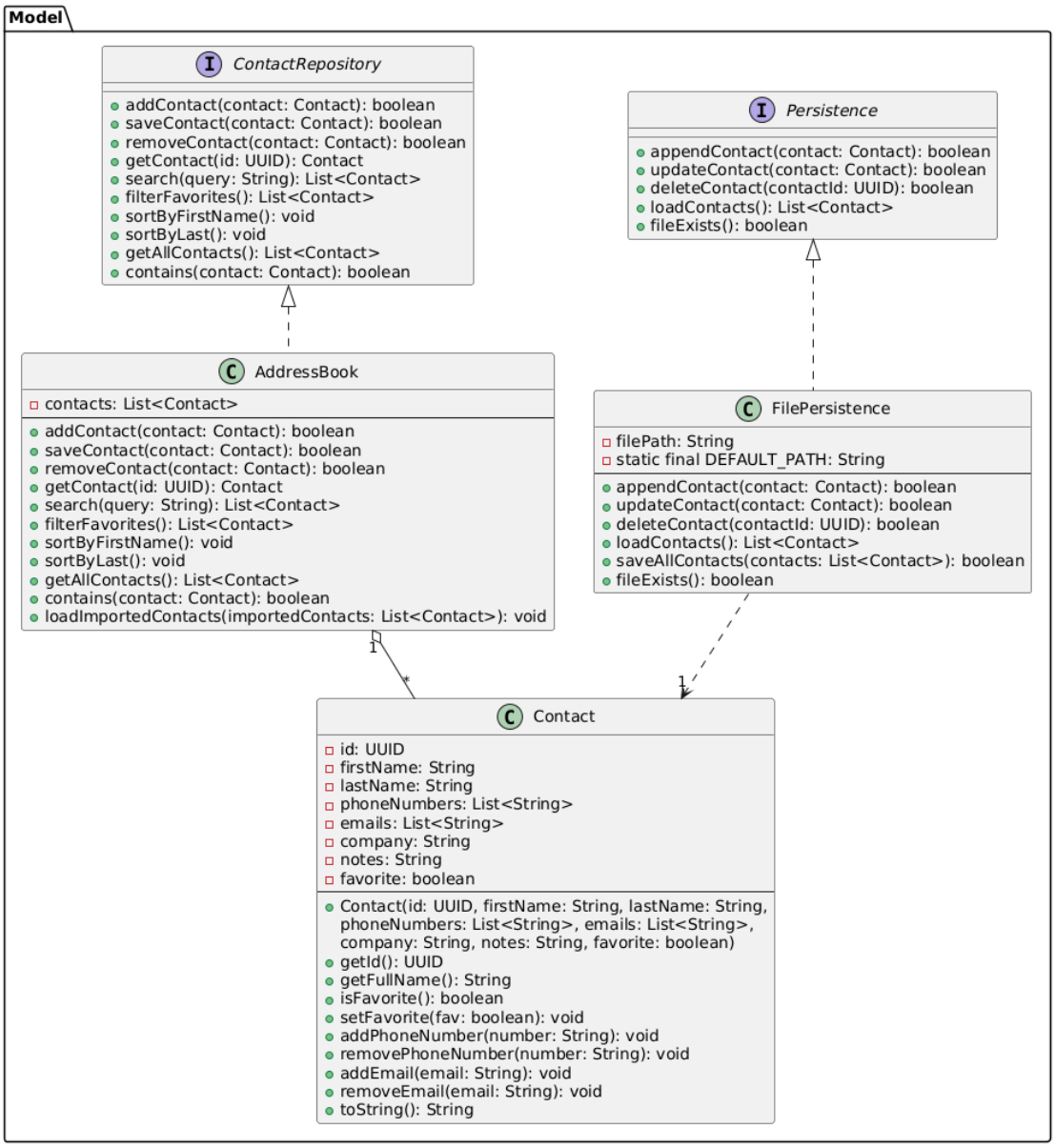
2 Class Diagram



Nota: questa illustrata è una versione ridotta del Class Diagram. Ha il solo scopo di illustrare in modo *compatto* il diagramma; seguiranno i diagrammi per ogni singolo componente e il diagramma completo.

2.1 Descrizioni delle classi

2.2 Model



2.2.1 Classe Contact

La classe *Contact* rappresenta un singolo contatto della rubrica. Contiene gli attributi principali, tra cui id, nome, cognome, numeri di telefono, indirizzi e-mail, azienda di appartenenza, note e stato di preferenza. I metodi offrono le

funzionalità di accesso e modifica (getter e setter), nonché la gestione dinamica delle liste di numeri e indirizzi. È inoltre presente il metodo *toString()*, che fornisce una rappresentazione testuale completa del contatto, utile per la visualizzazione e per le operazioni di debug.

2.2.2 Interfaccia ContactRepository

L'interfaccia *ContactRepository* è stata pensata per la gestione dei contatti. Stabilisce i metodi essenziali per l'inserimento, la modifica, l'eliminazione, la ricerca, l'applicazione del filtro e l'ordinamento dei contatti, oltre a quelli per l'accesso all'intera collezione. Fornendo un livello di astrazione tra la logica applicativa e la persistenza dei dati, favorisce l'estendibilità e la manutenibilità del sistema.

2.2.3 Classe AddressBook

La classe *AddressBook* implementa l'interfaccia *ContactRepository* e costituisce il nucleo logico del modello. Gestisce l'insieme dei contatti in memoria, incapsulando tutte le operazioni di aggiunta, modifica, eliminazione, ricerca, filtro e ordinamento. Garantisce la coerenza dei dati, fungendo da punto di accesso unificato alla rubrica. Include inoltre il metodo *loadImportedContacts()*, che consente di integrare in memoria nuovi contatti provenienti da file esterni, assegnando identificativi univoci (utili al riconoscimento di ogni singolo contatto, ominimi compresi, all'interno del file di salvataggio utilizzato dalla classe FilePersistence) pur mantenendo intatti i dati esistenti.

2.2.4 Interfaccia Persistence

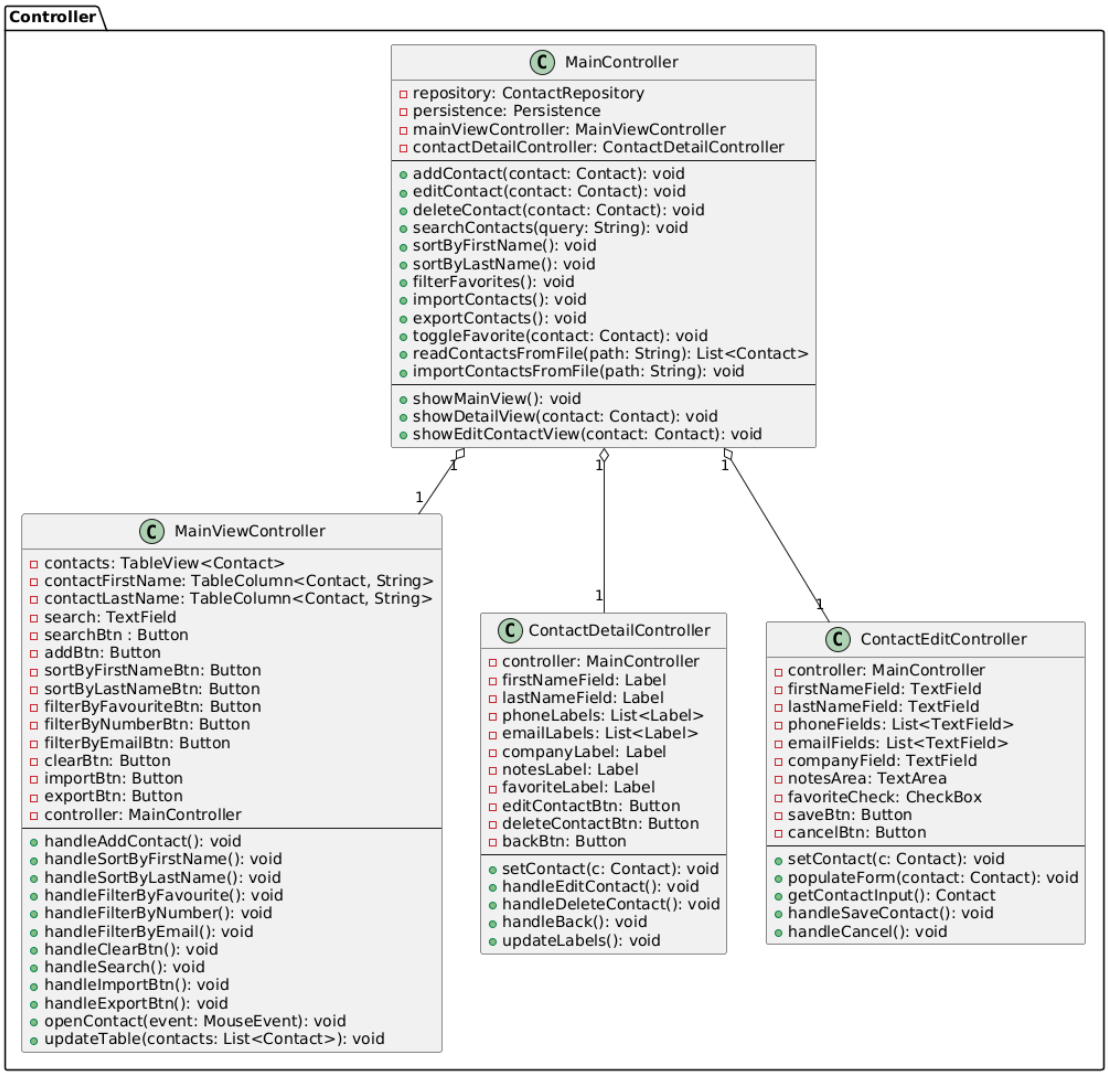
L'interfaccia *Persistence* è stata pensata per gestire la memorizzazione permanente dei dati. Specifica i metodi per l'inserimento, l'aggiornamento, la rimozione e il caricamento dei contatti dal file, oltre al controllo dell'esistenza del file stesso. Questa separazione di responsabilità consente di isolare la logica di business dalla modalità di archiviazione, rendendo il sistema indipendente dal formato fisico dei dati e facilitando l'adozione di diverse strategie di memorizzazione.

2.2.5 Classe FilePersistence

La classe *FilePersistence* implementa l'interfaccia *Persistence* e gestisce la memorizzazione effettiva dei contatti su file locale. Gli attributi principali includono il percorso del file (e un valore di default per l'inizializzazione). I metodi permettono di aggiungere, aggiornare o eliminare contatti, oltre a caricare e salvare

l'intera rubrica in modo atomico attraverso *saveAllContacts()*. La classe garantisce la disponibilità e la consistenza del file, assicurando un corretto salvataggio e recupero delle informazioni durante l'esecuzione dell'applicazione.

2.3 Controller



2.3.1 Classe MainController

La classe *MainController* rappresenta il centro logico dell'applicazione e coordina la comunicazione tra modello e vista. Mantiene i riferimenti ai moduli principali (repository dei contatti, componente di persistenza e i controller che si occupano

dell'interfaccia con l'utente) e gestisce il flusso delle operazioni richieste dall'utente. Attraverso i suoi metodi vengono eseguite le principali funzionalità del sistema: creazione, modifica, eliminazione, ricerca, filtraggio e ordinamento dei contatti, oltre alle operazioni di importazione ed esportazione da file. In questa classe risiedono anche le funzioni di gestione dei file esterni, come *readContactsFromFile()* e *importContactsFromFile()*, che permettono di integrare contatti provenienti da sorgenti esterne in modo sicuro e coerente. Il *MainController* funge dunque da mediatore centrale, garantendo la sincronizzazione tra i dati e la loro rappresentazione.

2.3.2 Classe MainViewController

La classe *MainViewController* controlla la finestra principale dell'applicazione, dedicata alla visualizzazione complessiva della rubrica. Gestisce la tabella dei contatti, le colonne e i controlli di ricerca, filtro e ordinamento. Gli *handler* dei pulsanti e dei campi di input inoltrano le azioni dell'utente al *MainController*, mentre i metodi di aggiornamento curano la sincronizzazione con lo stato del *Model*.

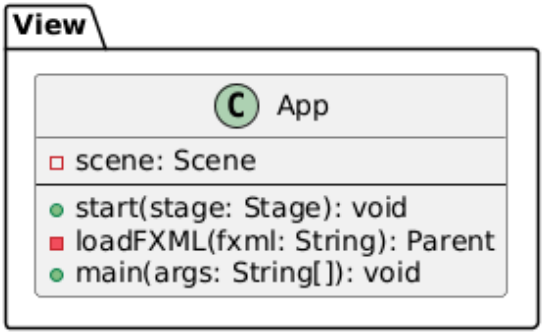
2.3.3 Classe ContactDetailController

La classe *ContactDetailController* gestisce la vista di dettaglio di un singolo contatto. Gli attributi corrispondono ai campi di sola lettura e ai pulsanti di navigazione (*Modifica*, *Elimina*, *Indietro*).

2.3.4 Classe ContactEditController

La classe *ContactEditController* gestisce la finestra dedicata alla creazione o modifica di un contatto. Contiene i riferimenti ai campi di input e ai pulsanti *Salva* e *Annulla*. Attraverso i suoi metodi vengono caricati i dati da modificare, raccolti quelli inseriti dall'utente e restituiti al *MainController* per l'aggiornamento del modello. La classe implementa inoltre controlli di validazione sui campi obbligatori, garantendo la correttezza e la consistenza dei dati memorizzati.

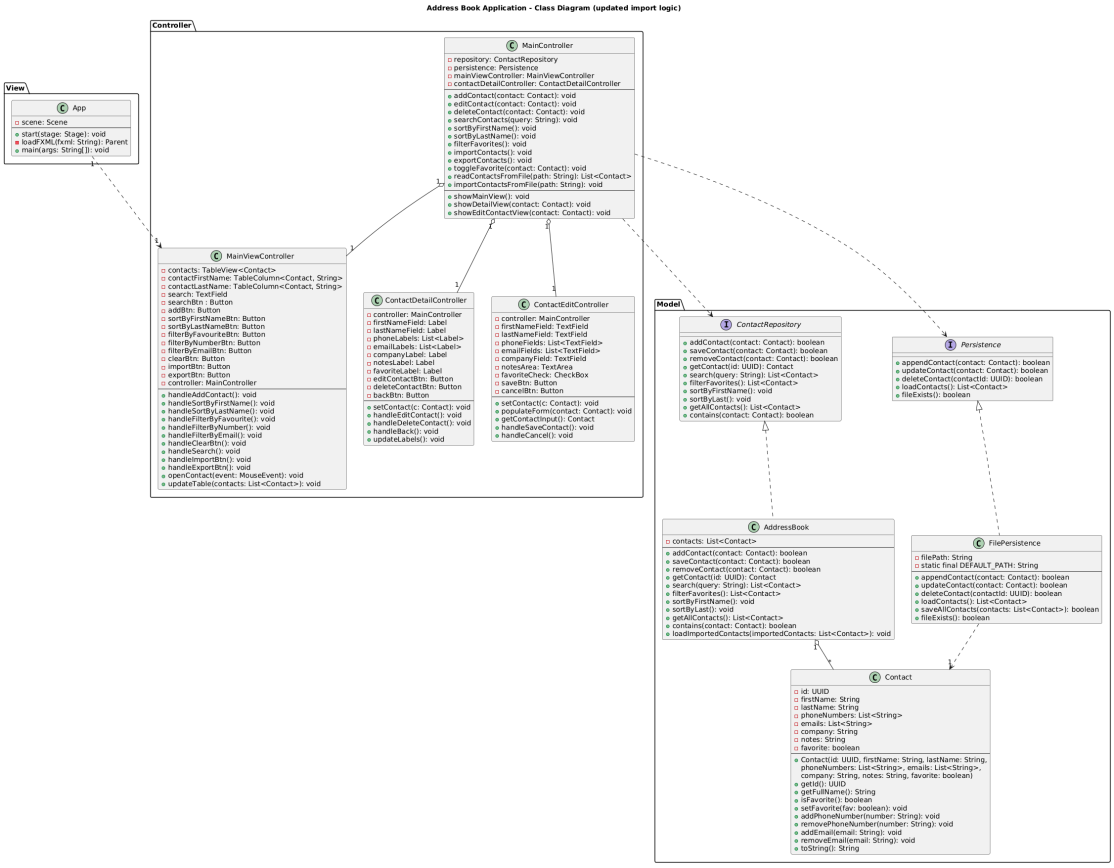
2.4 View



2.4.1 Classe App

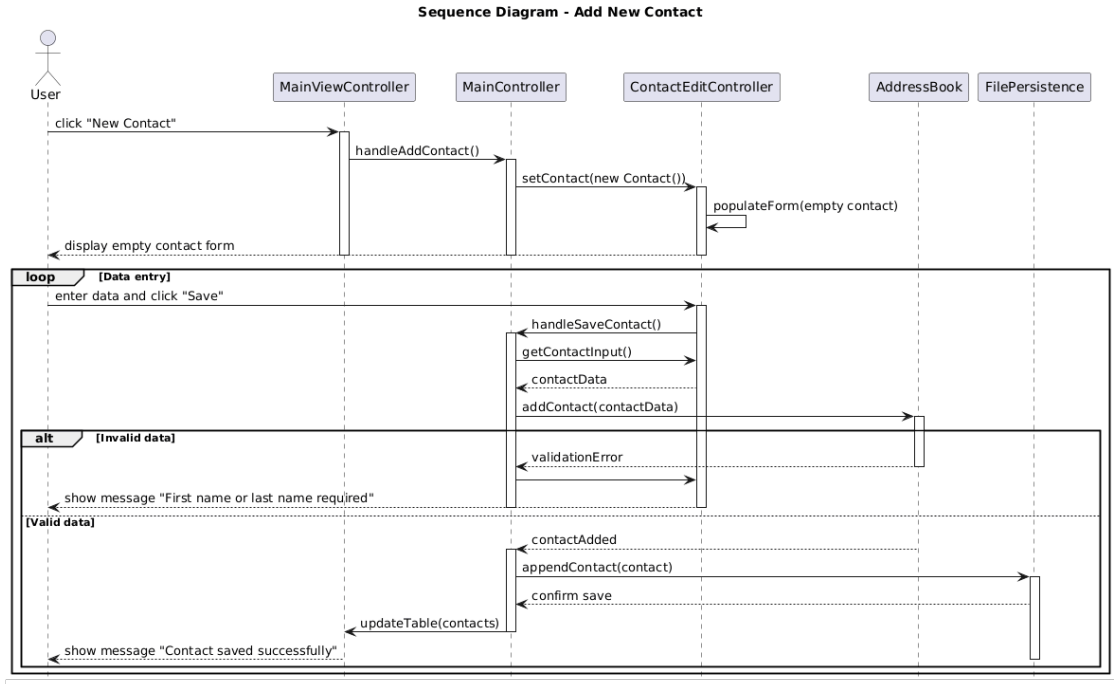
La classe *App* si occupa dell'inizializzazione della scena, del caricamento dei file FXML e dell'avvio della vista principale.

2.5 Class Diagram (completo)



3 Sequence Diagram

3.1 Sequence Diagram - Add New Contact



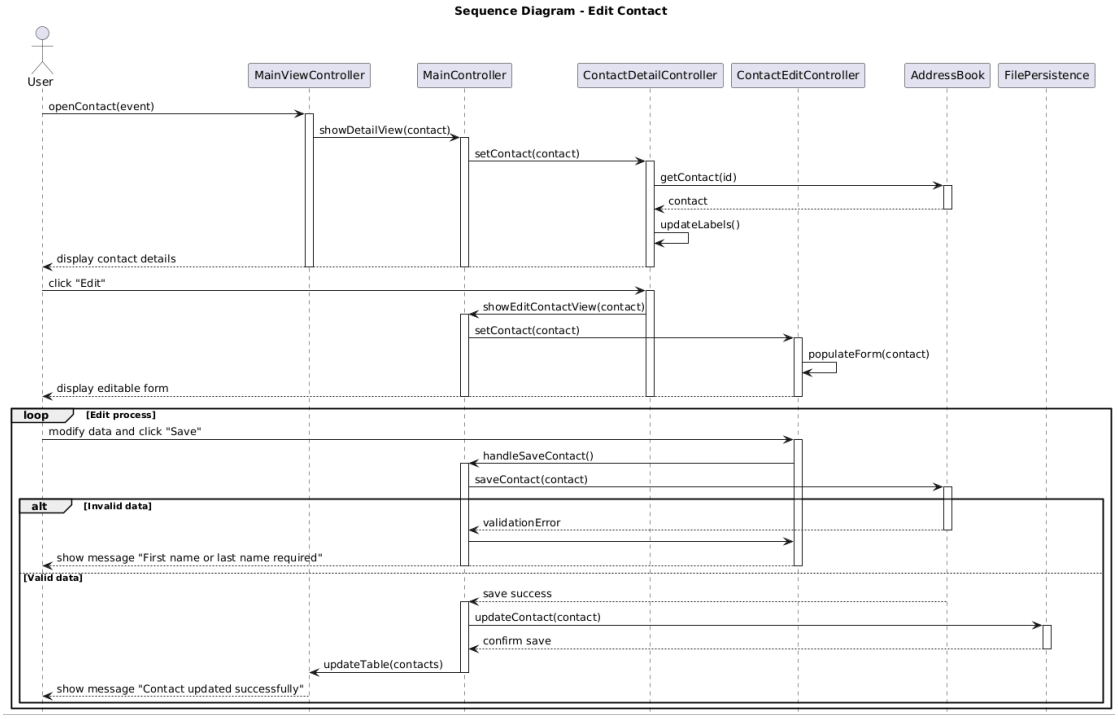
Descrive il processo di creazione di un nuovo contatto nella rubrica, dalla richiesta dell'utente fino al salvataggio nel file di persistenza.
Attore: *User*
Flusso principale:

- L'utente clicca sul pulsante *New Contact* nella finestra principale.
- Il *MainViewController* chiama *handleAddContact()* nel *MainController*.
- Il *MainController* crea un nuovo oggetto *Contact* e lo passa al *ContactEditController*, che inizializza e mostra il form vuoto di inserimento.
- L'utente compila i campi e clicca *Save*.
- Il *ContactEditController* richiama *handleSaveContact()* nel *MainController*, che ottiene i dati compilati tramite *getContactInput()*.

- Il *MainController* invia i dati al *AddressBook* per la validazione e l'aggiunta.
- In caso di dati non validi viene mostrato un messaggio di errore e l'utente può correggere l'input.
- Se i dati sono validi, il contatto viene aggiunto e salvato tramite *FilePersistence.appendContact()*.
- Il *MainController* aggiorna la vista principale con *updateTable()* e informa l'utente del successo.

Risultato atteso: Il nuovo contatto è aggiunto correttamente alla rubrica e salvato nel file locale, con aggiornamento immediato della vista.

3.2 Sequence Diagram - Edit Contact



Descrive il flusso di modifica di un contatto esistente e la successiva memorizzazione delle modifiche.

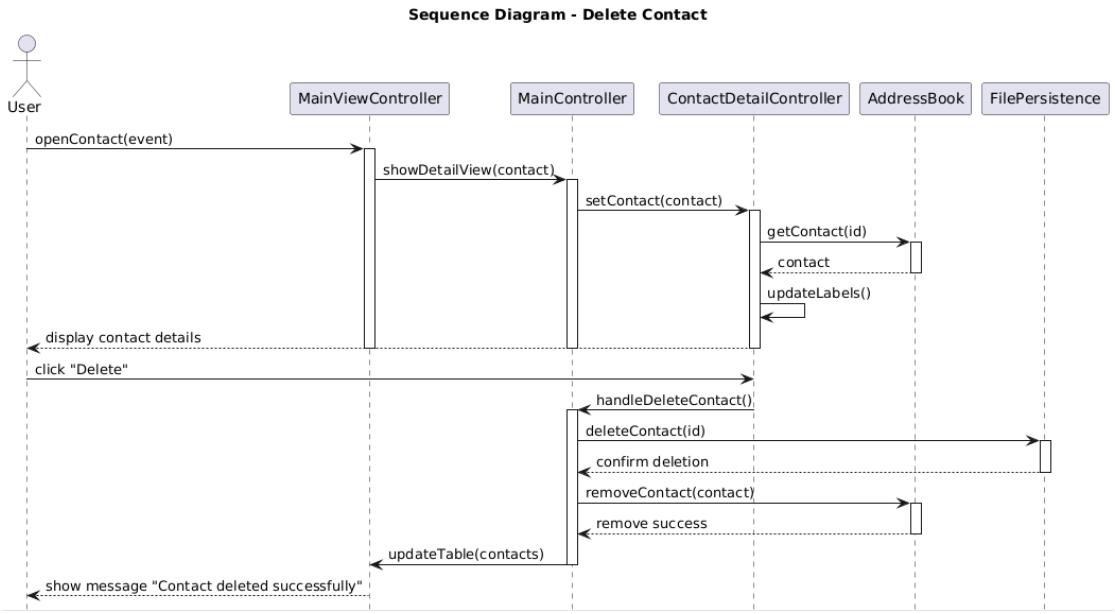
Attore: *User*

Flusso principale:

- L'utente apre i dettagli di un contatto dalla tabella principale.
- Il *MainViewController* chiama *showDetailView(contact)* nel *MainController*.
- Il *ContactDetailController* recupera i dati tramite *AddressBook.getContact(id)* e li mostra all'utente.
- L'utente clicca *Edit*, aprendo il form di modifica (*showEditContactView(contact)*).
- Dopo aver apportato modifiche, l'utente clicca *Save*.
- Il *ContactEditController* chiama *handleSaveContact()* $\rightarrow$ *MainController.saveContact(contact)*.
- Se i dati non sono validi viene mostrato un messaggio di errore.
- Se validi, *AddressBook.saveContact()* aggiorna i dati in memoria e *FilePersistence.updateContact()* li scrive su file.
- La vista principale viene aggiornata e mostra la conferma dell'avvenuto salvataggio.

Risultato atteso: Il contatto selezionato viene aggiornato correttamente sia nella rubrica che nel file persistente.

3.3 Sequence Diagram - Delete Contact

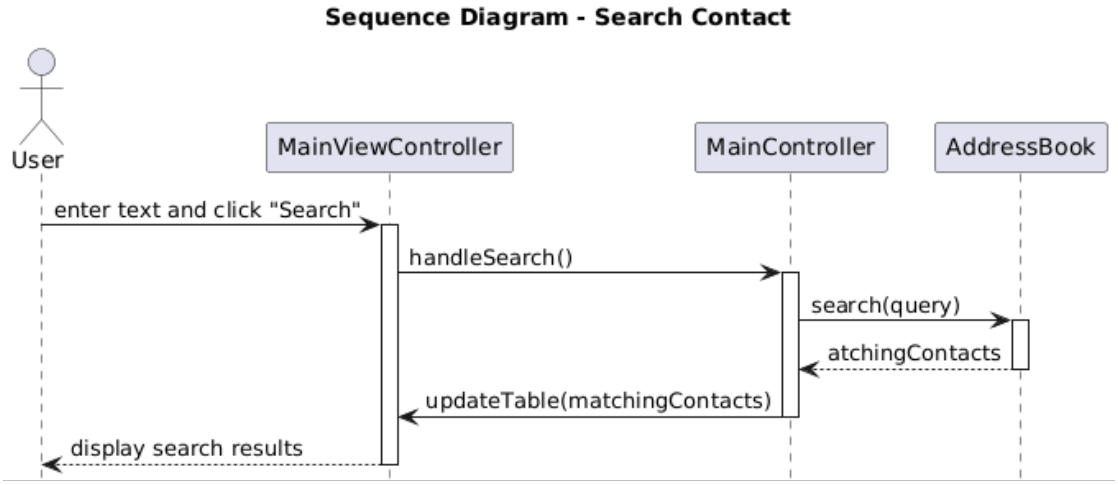


Descrive il processo di eliminazione di un contatto dalla rubrica e dal file di persistenza.
Attore: *User*
Flusso principale:

- L'utente apre i dettagli di un contatto e seleziona *Delete*.
- Il *ContactDetailController* invoca *handleDeleteContact()* nel *MainController*.
- Il *MainController* ordina la rimozione del contatto dal file tramite *FilePersistence.deleteContact(id)*.
- Dopo conferma dell'eliminazione, rimuove il contatto anche dal *AddressBook* con *removeContact(contact)*.
- La *MainView* viene aggiornata e all'utente viene notificata l'eliminazione.

Risultato atteso: Il contatto è rimosso sia dalla rubrica in memoria sia dal file di persistenza, con aggiornamento visivo della tabella.

3.4 Sequence Diagram - Search Contact

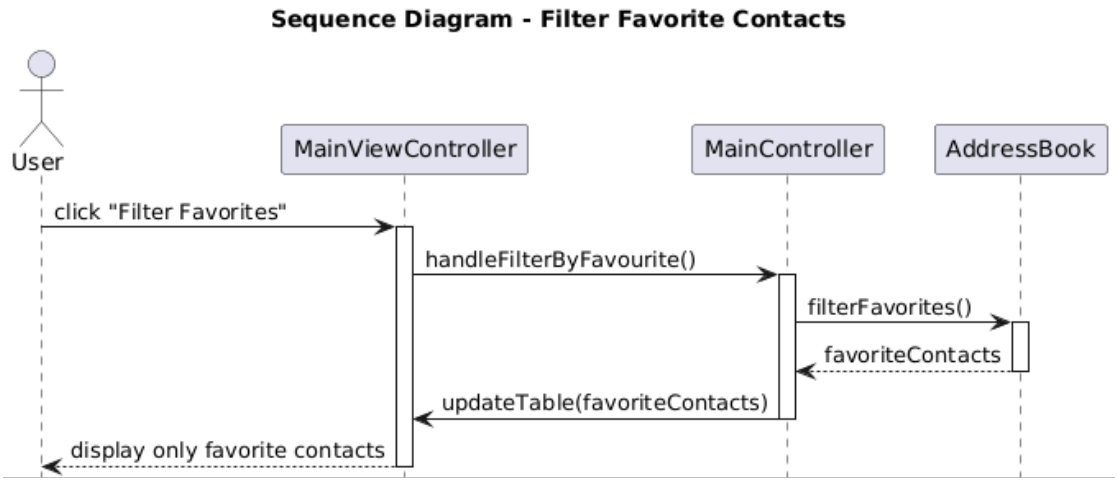


Descrive la ricerca di contatti nella rubrica tramite la sottostringa del nome o del cognome.
Attore: *User*
Flusso principale:

- L'utente inserisce un testo nel campo di ricerca e clicca *Search*.
- Il *MainViewController* invoca *handleSearch()* nel *MainController*.
- Il *MainController* chiama *AddressBook.search(query)*.
- L'*AddressBook* restituisce l'elenco dei contatti corrispondenti.
- Il *MainController* aggiorna la vista tramite *updateTable()*.

Risultato atteso: L'interfaccia mostra i contatti che corrispondono ai criteri di ricerca inseriti.

3.5 Sequence Diagram - Filter Favorite Contacts



Descrive il processo di *filtraggio* dei contatti contrassegnati come preferiti.

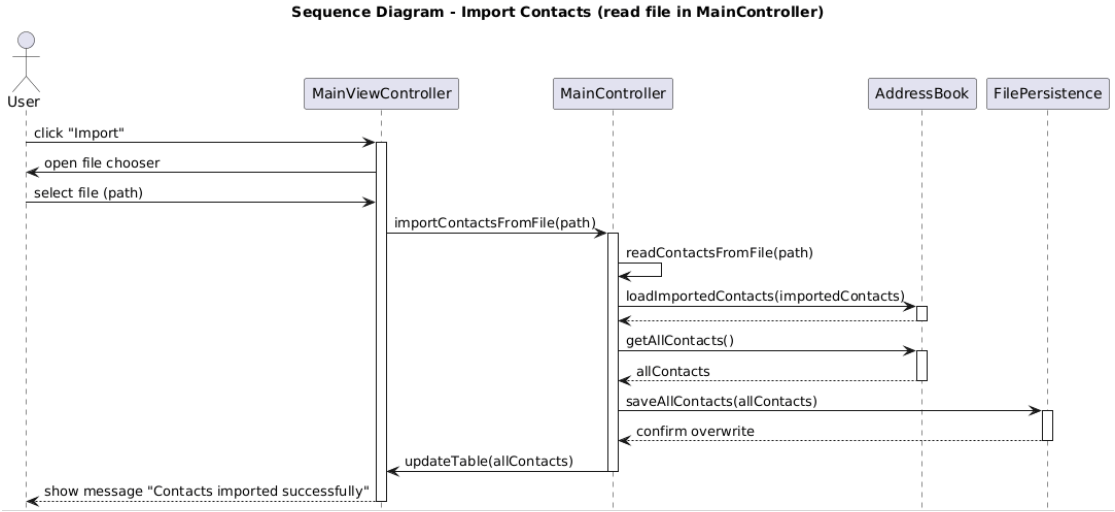
Attore: *User*

Flusso principale:

- L'utente clicca sul pulsante *Filter Favorites*.
- Il *MainViewController* invoca *handleFilterByFavourite()* nel *MainController*.
- Il *MainController* chiama *AddressBook.filterFavorites()*.
- Il model restituisce solo i contatti con flag *favorite = true*.
- La tabella viene aggiornata mostrando esclusivamente i contatti preferiti.

Risultato atteso: La vista mostra solo i contatti contrassegnati come preferiti.

3.6 Sequence Diagram - Import Contacts

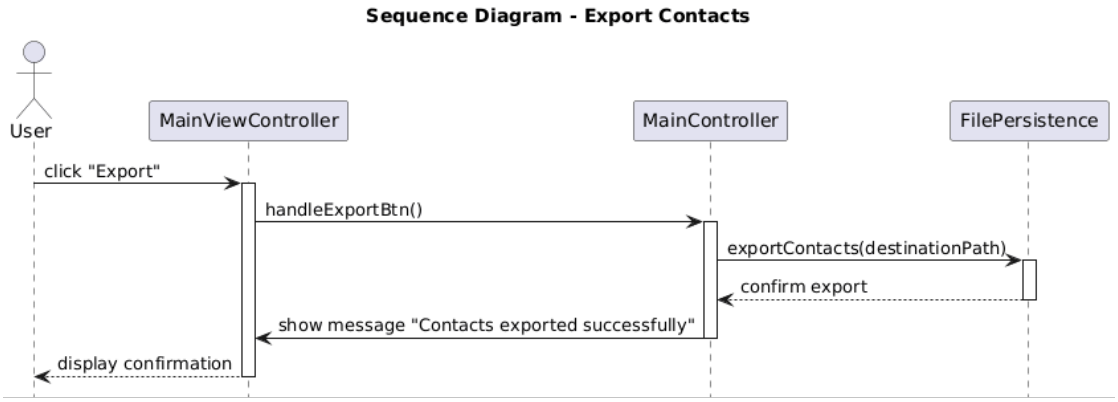


Descrive l'importazione di contatti da un file esterno selezionato dall'utente.
Attore: *User*
Flusso principale:

- L'utente clicca *Import*, il sistema apre una finestra di selezione file.
- Dopo la scelta del file, il *MainViewController* chiama *importContactsFromFile(path)* nel *MainController*.
- Il *MainController* legge i contatti direttamente dal file tramite *readContactsFromFile(path)*.
- I contatti vengono aggiunti in memoria con *AddressBook.loadImportedContacts(importedContac*
- Il *MainController* recupera tutti i contatti aggiornati tramite *getAllContacts()*.
- *FilePersistence.saveAllContacts(allContacts)* riscrive il file ufficiale di persistenza con i dati aggiornati.
- La vista principale viene aggiornata con la rubrica completa e viene mostrato un messaggio di conferma.

Risultato atteso: I contatti importati vengono aggiunti alla rubrica, salvati nel file principale e visualizzati nell'interfaccia.

3.7 Sequence Diagram - Export Contacts



Descrive il processo di esportazione della rubrica in un file scelto dall'utente.
Attore: *User*
Flusso principale:

- L'utente clicca *Export* nella finestra principale.
- Il *MainViewController* invoca *handleExportBtn()* nel *MainController*.
- Il *MainController* chiama *FilePersistence.exportContacts(destinationPath)*.
- Il sistema salva tutti i contatti della rubrica in un file di destinazione scelto.
- Alla conferma del salvataggio, la vista mostra un messaggio di completamento.

Risultato atteso: La rubrica viene esportata correttamente in un file esterno selezionato dall'utente.