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 18, 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.

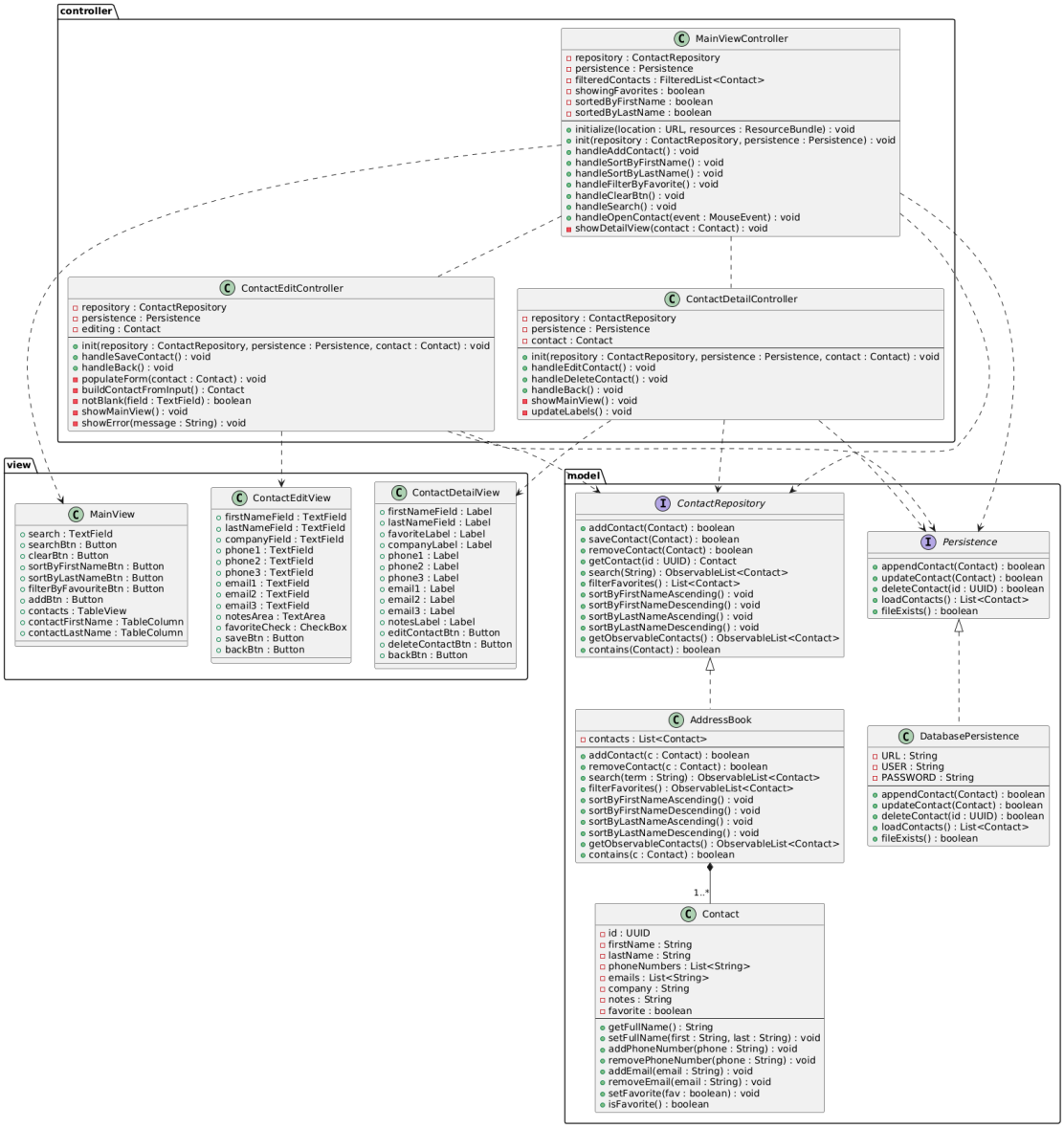
- **MainView**: mostra la rubrica completa, con la tabella dei contatti e i controlli di ricerca, filtro e ordinamento.
- **ContactDetailView**: visualizza i dettagli di un singolo contatto, con opzioni per modificarlo o eliminarlo.
- **ContactEditView**: fornisce il form di creazione o modifica di un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

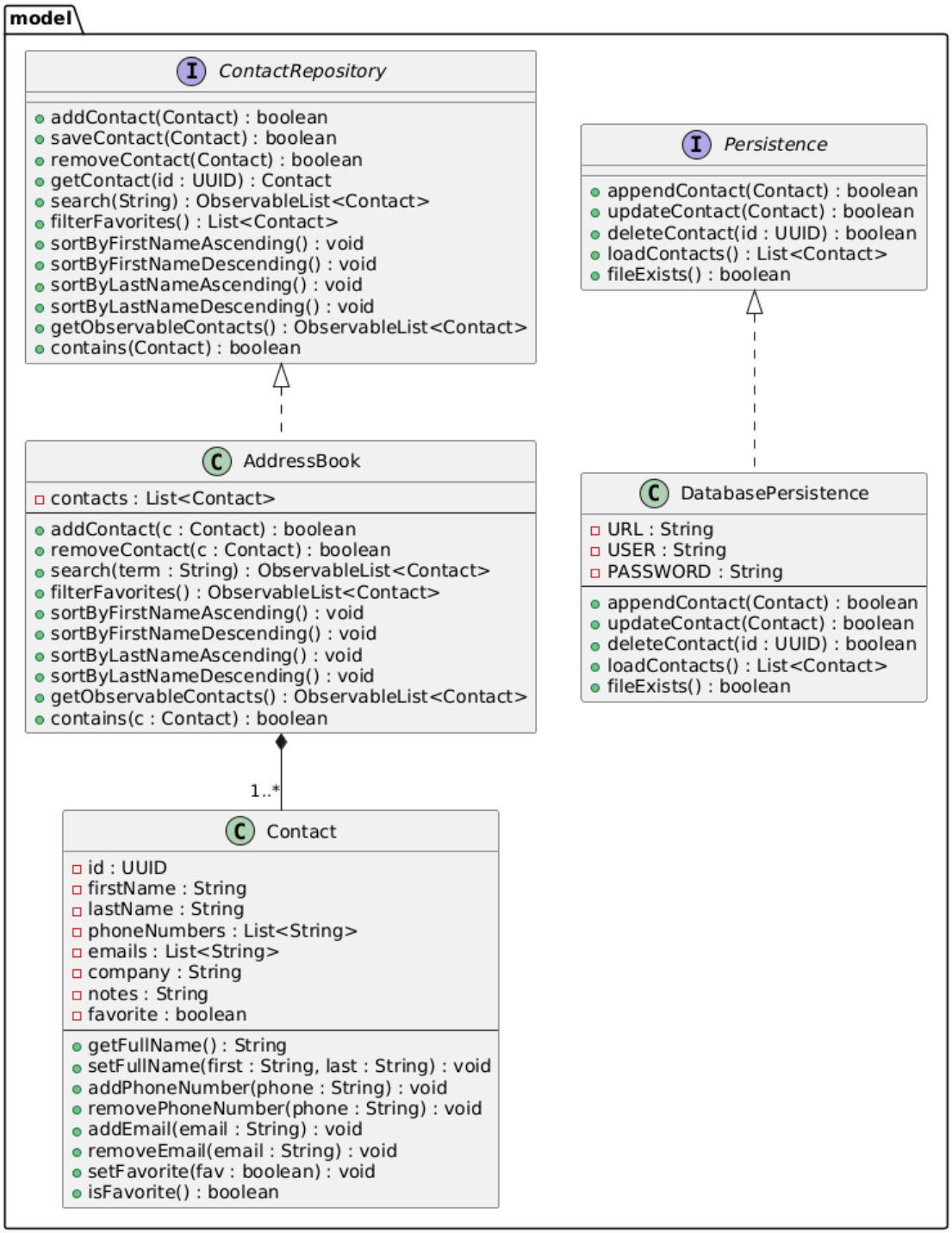
- **MainViewController**: rappresenta il punto di coordinamento logico dell'applicazione, orchestrando le operazioni tra modello, persistenza e vista; controlla direttamente la finestra principale.
- **ContactDetailController**: gestisce la vista di dettaglio di un contatto, aggiornando i dati mostrati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

2 Class Diagram



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.

2.2.2 Interfaccia ContactRepository

L'interfaccia *ContactRepository* è stata pensata per la corretta 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 model. 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.

2.2.4 Interfaccia Persistence

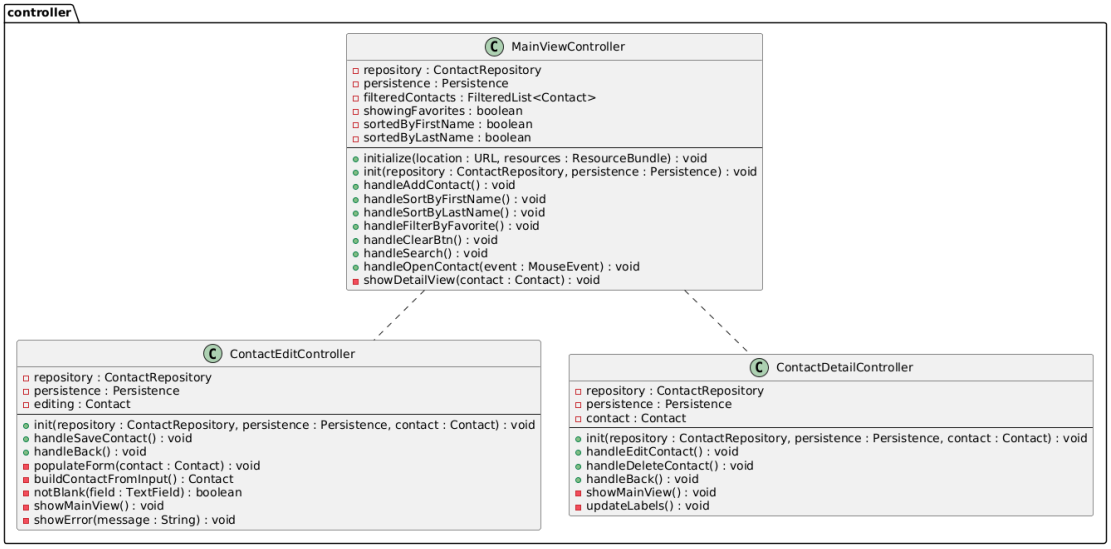
L'interfaccia *Persistence* è stata pensata per gestire la memorizzazione permanente dei dati. Specifica i metodi per l'inserimento, l'aggiornamento, la rimozione e il caricamento dei contatti, oltre al controllo dell'esistenza della sorgente dei dati (sia file, sia database). Questa separazione di responsabilità 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 DatabasePersistence

La classe *DatabasePersistence* implementa l'interfaccia *Persistence* e gestisce la memorizzazione permanente dei contatti all'interno di un database PostgreSQL. Durante l'inizializzazione verifica la presenza della tabella *contacts* e, se necessario, la crea automaticamente per garantire la disponibilità del supporto dati. I suoi metodi principali consentono di inserire nuovi contatti (*appendContact()*),

aggiornarne i dati esistenti (*updateContact()*), eliminarli tramite identificativo univoco (*deleteContact()*) e il caricamento dei dati salvati (*loadContacts()*). La classe assicura la consistenza dei dati e la corretta gestione delle connessioni al database, fungendo da interfaccia tra il modello applicativo e la base di dati sottostante.

2.3 Controller



2.3.1 Classe MainViewController

La classe *MainViewController* costituisce il punto di accesso principale dell'interfaccia grafica dell'applicazione. Gestisce la finestra contenente la tabella dei contatti e coordina le operazioni di visualizzazione, ricerca, filtro e ordinamento. Mantiene riferimenti al *ContactRepository* e al componente di *Persistence*, garantendo l'allineamento tra la rappresentazione grafica e i dati effettivi del modello. Attraverso i metodi associati ai pulsanti e ai campi di input, riceve le azioni dell'utente (come l'aggiunta, l'apertura o la rimozione di contatti) e gestisce il flusso verso le viste dedicate al dettaglio o alla modifica. In sintesi, funge da mediatore tra l'utente e il modello, assicurando la coerenza dell'interfaccia con lo stato corrente della rubrica.

2.3.2 Classe ContactDetailController

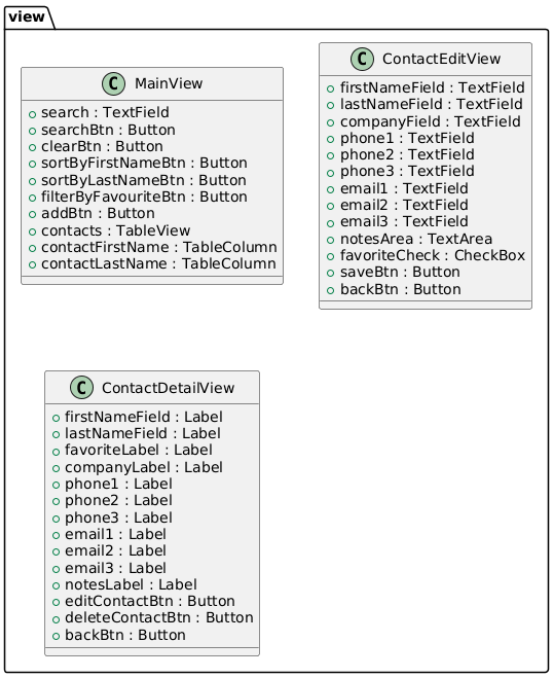
La classe *ContactDetailController* gestisce la vista di dettaglio relativa a un singolo contatto. Mostra tutte le informazioni registrate nel modello, in modalità di sola lettura, e fornisce le azioni principali: *Modifica*, *Elimina* e *Indietro*. Utilizza i riferimenti a *ContactRepository* e *Persistence* per applicare in modo immediato le

operazioni richieste dall'utente, come la cancellazione o l'apertura della finestra di modifica. La classe assicura la sincronizzazione tra l'interfaccia grafica e lo stato aggiornato del contatto visualizzato.

2.3.3 Classe ContactEditController

La classe *ContactEditController* gestisce la finestra dedicata alla creazione o alla modifica di un contatto. I suoi attributi rappresentano i campi di input (nome, cognome, numeri di telefono, e-mail, azienda, note e preferito) e i pulsanti di interazione (*Salva* e *Annulla*). Durante l'editing, la classe carica i dati del contatto selezionato (se presenti), consente la modifica dei valori e valida gli input per garantirne la correttezza formale. Al momento del salvataggio, comunica con il *ContactRepository* per aggiornare il modello e con il componente di *Persistence* per la scrittura sulla memoria permanente. In tal modo, *ContactEditController* assicura un flusso coerente tra interfaccia, logica applicativa e livello di persistenza.

2.4 View



2.4.1 MainView

La *MainView* rappresenta la finestra principale dell'applicazione. Contiene la tabella che mostra l'elenco dei contatti e gli elementi di controllo per la gestione della rubrica: ricerca, ordinamento, filtro dei preferiti e aggiunta di nuovi contatti. L'interfaccia consente all'utente di interagire in modo diretto con la rubrica, fornendo un accesso centralizzato a tutte le operazioni principali del sistema. Gli elementi grafici principali includono il campo di ricerca (*TextField*), i pulsanti di azione (*Button*) e la tabella dei contatti con relative colonne (*TableView*, *TableColumn*).

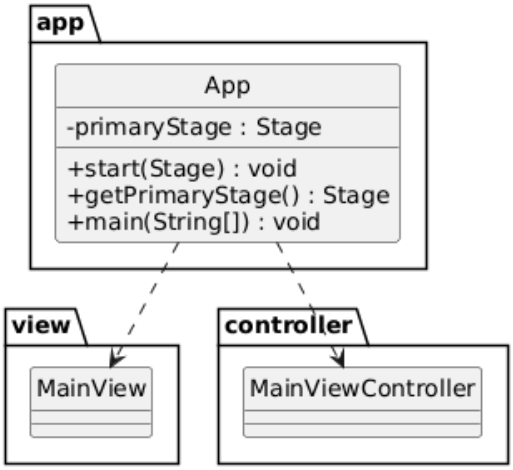
2.4.2 ContactEditView

La *ContactEditView* fornisce l'interfaccia dedicata alla creazione o modifica di un contatto. Include i campi di input per nome, cognome, azienda, numeri di telefono, indirizzi e-mail, note e flag di preferito. L'utente può inserire o aggiornare le informazioni e confermare tramite il pulsante *Save* oppure annullare con *Back*. La vista è progettata per garantire chiarezza e semplicità nella compilazione dei dati, integrando controlli di validazione sui campi obbligatori.

2.4.3 ContactDetailView

La *ContactDetailView* visualizza i dettagli completi di un singolo contatto in modalità di sola lettura. Mostra tutte le informazioni memorizzate, inclusi nomi, azienda, numeri di telefono, e-mail, note e stato di preferito. Offre inoltre pulsanti di navigazione per modificare (*Edit*), eliminare (*Delete*) o tornare alla schermata principale (*Back*).

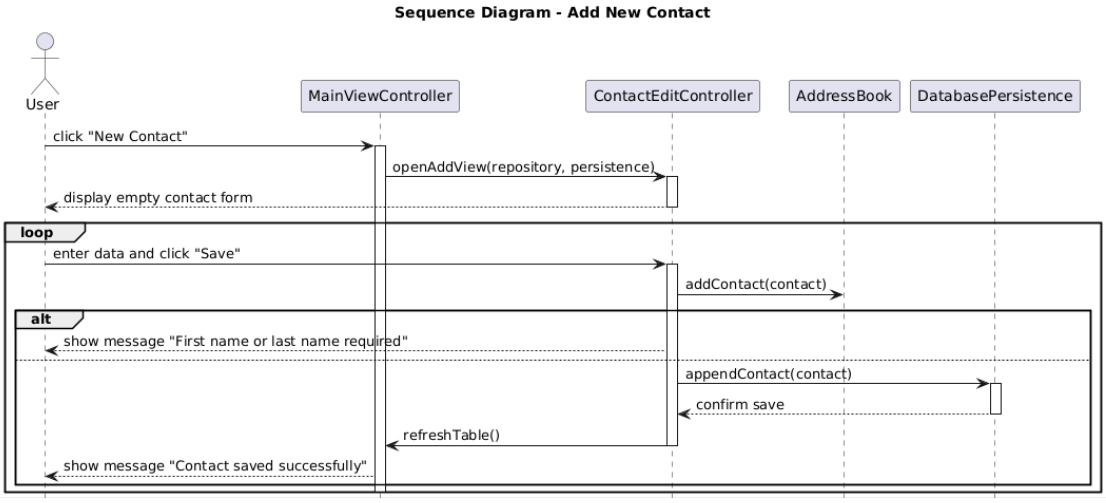
2.5 Classe App



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

3 Sequence Diagram

3.1 Sequence Diagram - Add New Contact

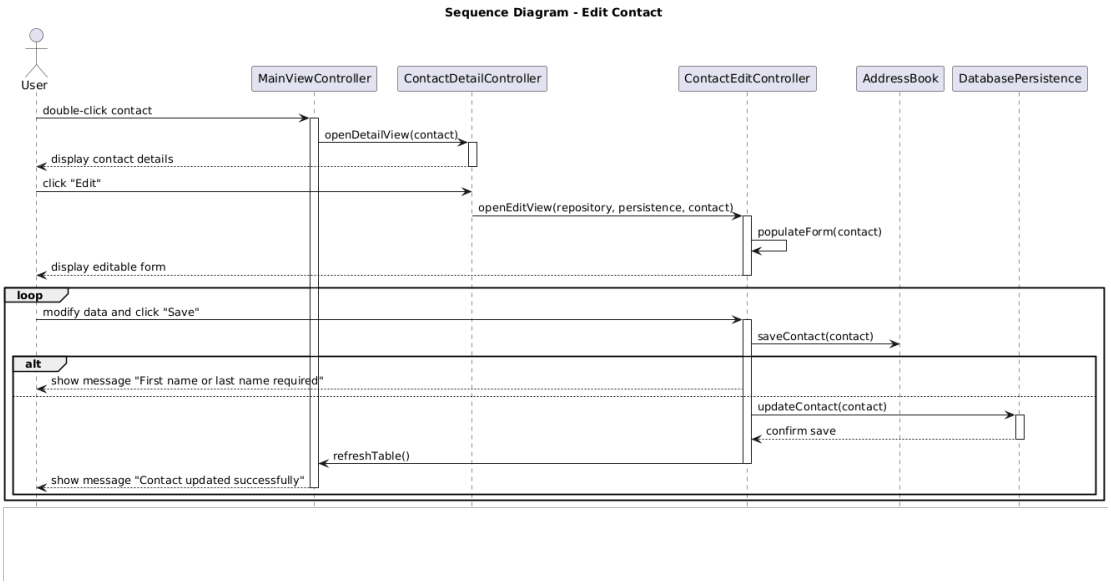


Descrive il processo di creazione di un nuovo contatto nella rubrica, dalla richiesta dell'utente fino al salvataggio nel database.
Attore: *User*
Flusso principale:

1. L'utente clicca sul pulsante *New Contact* nella finestra principale.
2. Il *MainViewController* apre la finestra di inserimento tramite il *ContactEditController*.
3. Il *ContactEditController* mostra un form vuoto e attende i dati dell'utente.
4. L'utente compila i campi e clicca *Save*.
5. Il *ContactEditController* valida i dati inseriti e, se validi, aggiunge il nuovo contatto al modello tramite *AddressBook.addContact(contact)*.
6. In seguito, il contatto viene salvato in modo permanente nel database tramite *DatabasePersistence.appendContact(contact)*.
7. Il *MainViewController* aggiorna la tabella dei contatti e notifica il successo dell'operazione.

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

3.2 Sequence Diagram - Edit Contact



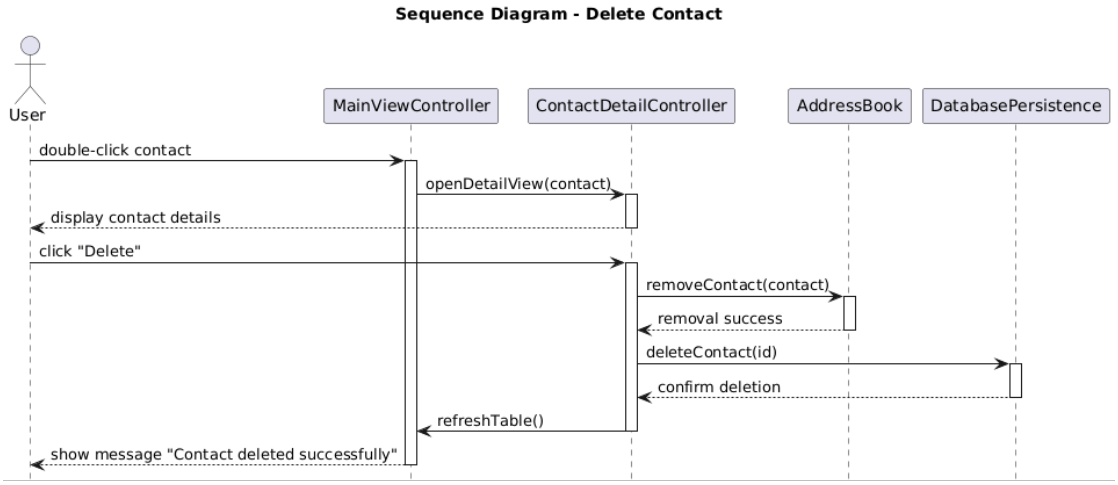
Descrive il flusso di modifica di un contatto esistente e la successiva memorizzazione delle modifiche.
Attore: *User*
Flusso principale:

1. L'utente apre i dettagli di un contatto dalla finestra principale.
2. Il *MainViewController* richiama il *ContactDetailController* per mostrare le informazioni del contatto selezionato.
3. L'utente clicca *Edit*, aprendo la finestra di modifica gestita dal *ContactEditController*.
4. Il *ContactEditController* precompila il form con i dati del contatto.
5. L'utente apporta modifiche e clicca *Save*.
6. I dati vengono validati e, se corretti, aggiornati nel modello tramite *AddressBook.saveContact(contact)*.
7. La persistenza viene gestita con *DatabasePersistence.updateContact(contact)*.

8. Il *MainViewController* aggiorna la tabella e visualizza un messaggio di conferma.

Risultato atteso: Il contatto selezionato viene aggiornato correttamente nella rubrica e nel database.

3.3 Sequence Diagram - Delete Contact

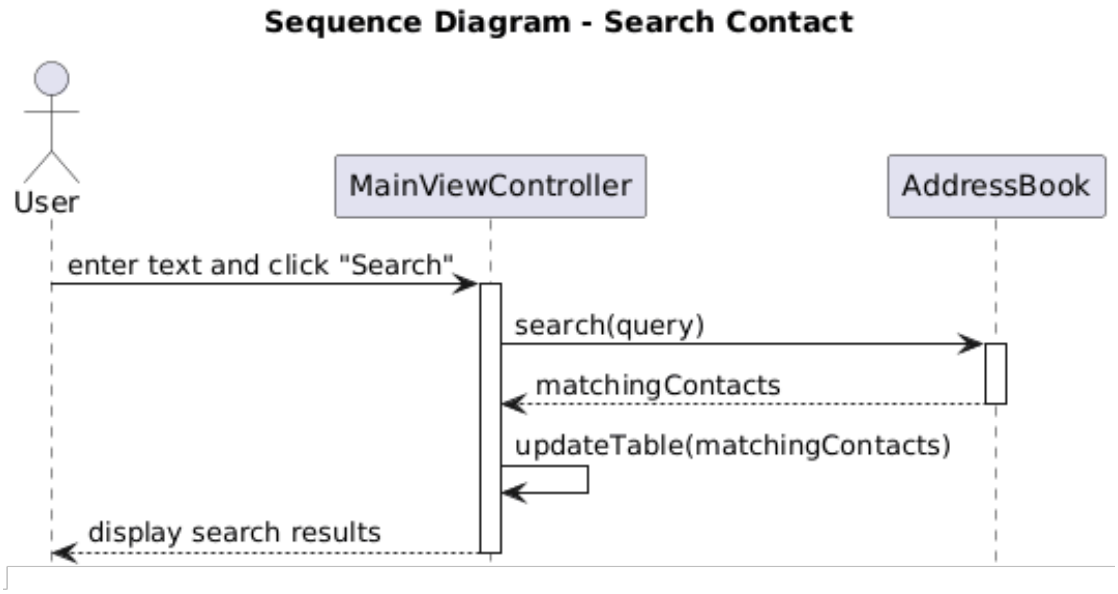


Descrive il processo di eliminazione di un contatto dalla rubrica e dal database.
Attore: *User*
Flusso principale:

1. L'utente apre la finestra di dettaglio di un contatto e seleziona *Delete*.
2. Il *ContactDetailController* mostra una finestra di conferma per l'eliminazione.
3. In caso di conferma, il contatto viene rimosso dal modello tramite *AddressBook.removeContact(contact)*.
4. Successivamente, il record viene cancellato dal database tramite *DatabasePersistence.deleteContact(id)*.
5. Il *MainViewController* aggiorna la tabella principale e mostra un messaggio di eliminazione riuscita.

Risultato atteso: Il contatto è rimosso sia dalla rubrica in memoria sia dal database, con aggiornamento visivo della vista principale.

3.4 Sequence Diagram - Search Contact

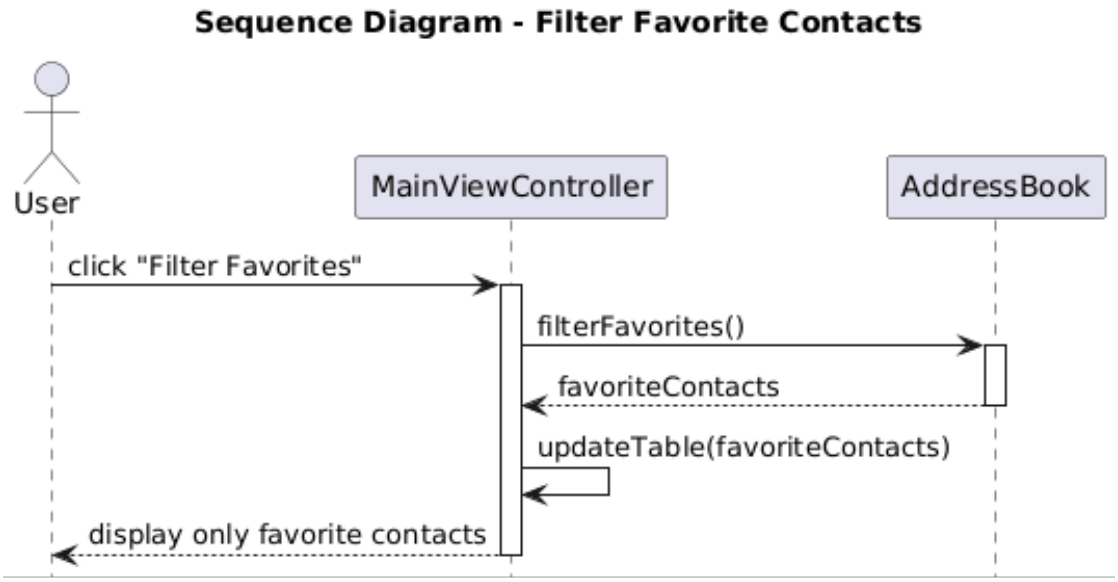


Descrive la ricerca dei contatti nella rubrica tramite la sottostringa del nome o del cognome.
Attore: *User*
Flusso principale:

1. L'utente inserisce un testo nel campo di ricerca e clicca *Search*.
2. Il *MainViewController* invoca *AddressBook.search(query)*.
3. L'*AddressBook* filtra i contatti che contengono la sottostringa nel nome o nel cognome.
4. I risultati vengono restituiti al *MainViewController*, che aggiorna la tabella visualizzata.

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

3.5 Sequence Diagram - Filter Favorite Contacts

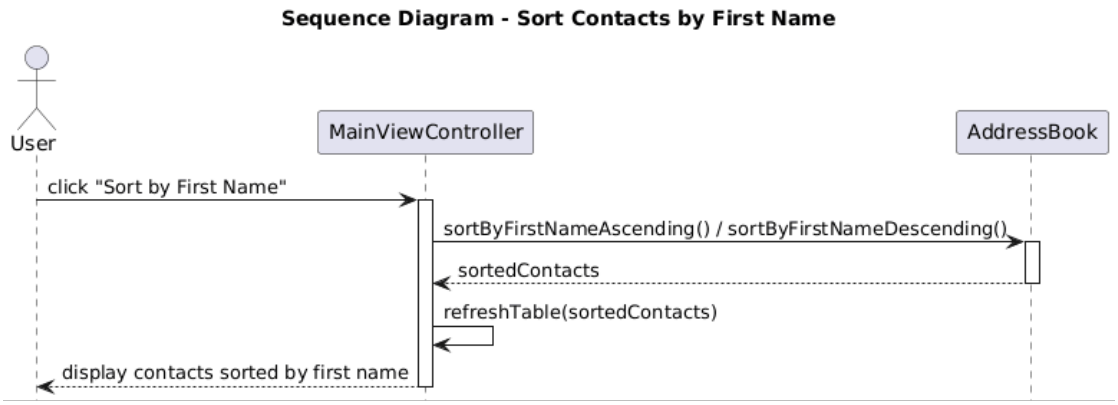


Descrive il processo di filtraggio dei contatti contrassegnati come preferiti.
Attore: *User*
Flusso principale:

1. L'utente clicca sul pulsante *Filter Favorites*.
2. Il *MainViewController* invoca *AddressBook.filterFavorites()*.
3. Il model restituisce solo i contatti con l'attributo *favorite = true*.
4. Il *MainViewController* aggiorna la tabella mostrando esclusivamente i contatti preferiti.

Risultato atteso: La vista principale mostra soltanto i contatti marcati come preferiti.

3.6 Sequence Diagram - Sort Contacts by First Name

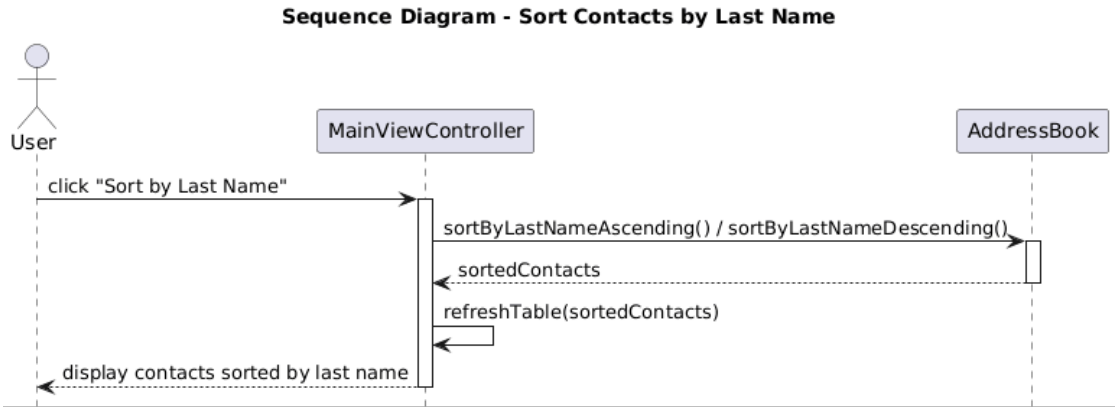


Descrive il processo di ordinamento dei contatti in base al nome.
Attore: *User*
Flusso principale:

1. L'utente clicca sul pulsante *Sort by First Name*.
2. Il *MainViewController* chiama *AddressBook.sortByFirstNameAscending()* oppure *sortByFirstNameDescending()*, alternando l'ordine.
3. L'*AddressBook* riordina la lista dei contatti.
4. Il *MainViewController* aggiorna la tabella per riflettere il nuovo ordinamento.

Risultato atteso: I contatti vengono visualizzati ordinati alfabeticamente per nome.

3.7 Sequence Diagram - Sort Contacts by Last Name



Descrive il processo di ordinamento dei contatti in base al cognome.
Attore: *User*
Flusso principale:

1. L'utente clicca sul pulsante *Sort by Last Name*.
2. Il *MainViewController* invoca *AddressBook.sortByLastNameAscending()* oppure *sortByLastNameDescending()*.
3. L'*AddressBook* riordina la lista in base al cognome.
4. Il *MainViewController* aggiorna la vista mostrando i contatti ordinati.

Risultato atteso: I contatti sono mostrati in ordine alfabetico per cognome.

4 Matrice di tracciabilità

ID	Requisito	Design	Codice	Test	Requisiti collegati
RF01	Gestione dei contatti	Progettato	Implementato	Da testare	
RF02	Persistenza dei dati	Progettato	Implementato	Da testare	
RF03	Ricerca	Progettato	Implementato	Da testare	RF01
RF04	Ordinamento	Progettato	Implementato	Da testare	RF05
RF05	Interfaccia	Progettato	Implementato	Da testare	RF01, RF03, RF04

5 Registro degli aggiornamenti

5.1 Versione 2.0 - Aggiornamento 17 Ottobre 2025

A seguito di alcuni errori riscontrati nel documento di progettazione "Software-Engineering-Project-Design-v1.0" e di un chiarimento relativo ad alcune specifiche che ha comportato una lieve revisione dei requisiti, si è ritenuto necessario fornire una versione aggiornata del documento.

- **Revisione del Class Diagram** Il diagramma delle classi è stato aggiornato per aderire pienamente alla struttura e alle specifiche effettive del software. Sono state corrette alcune imprecisioni e chiariti i requisiti relativi alle operazioni di *Salvataggio* e *Caricamento*, inizialmente interpretate come funzionalità di import/export distinte dalla persistenza dei dati (ovvero il salvataggio e il recupero dei contatti).
  Nota: il chiarimento è stato definito a seguito di un confronto con il committente.

- **Revisione dei Sequence Diagram** Revisione conseguente all'aggiornamento del *Class Diagram* e alla correzione di alcuni errori di impaginazione.

- **Matrice di tracciabilità** Aggiunta della matrice di tracciabilità al documento di progettazione.