Università degli Studi di Salerno

Dipartimento di Ingegneria dell'Informazione ed Elettrica e Matematica Applicata (DIEM)

Documentazione del Progetto

Sviluppo di un applicazione Java per la gestione di una rubrica

Corso di Ingegneria del Software - A.A. 2024/25

Studenti Gruppo 12:

Francesco Peluso
Matricola: 0612707469

Gerardo Selce
Matricola: 0612707692

Valerio Volzone
Matricola: 0612707693

Sharon Schiavano
Matricola: 0612708676

Docente:
Prof. Nicola Capuano

Last update: 15 dicembre 2024

1 Link utili

- Repository GitHub del progetto[https://github.com/francescopeluso/SWE-Project]
- Issues della repository su GitHub[https://github.com/francescopeluso/SWE-Project/issues]
- Roadmap delle scadenze su GitHub Projects[https://github.com/users/francescopeluso/projects/2]
- Documentazione Doxygen aggiornata online[https://francescopeluso.github.io/SWE-Project]

Indice

Abstract

Il software da realizzare é una rubrica telefonica, con una interfaccia utente intuitiva e facilmente accessibile, che permette l'inserimento, la modifica e la cancellazione dei contatti. Le funzionalità richieste includono la visualizzazione in ordine alfabetico della lista dei contatti, la ricerca di un contatto attraverso parte del suo nome e/o cognome.

2 Elicitazione dei requisiti

- Importare una rubrica già esistente.
- Esportare una rubrica
- Creare un contatto.
- Salvare un contatto.
- Modificare un contatto.
- Eliminare un contatto.
- Ricercare un contatti.
- Visualizzare dei contatti.
- Interfaccia utente (GUI) semplice e accessibile.
- Ogni contatto deve avere almeno il nome o il cognome.
- Associare da 0 a 3 numeri di telefono per contatto.
- Inserimento di una categoria per ognuno dei numeri di telefono salvati.
- Associare da 0 a 3 indirizzi email per contatto.
- Inserimento di una categoria per ognuna delle email salvate.
- Creare una lista di contatti preferiti.
- Il sistema software dovrà poter gestire un massimo di 5000 contatti.
- Il sistema software dovrà mantenere un tempo di risposta inferiore a 2 secondi in operazioni di ricerca e salvataggio.
- Il file contenente i contatti, e le informazioni associate, sarà criptografato.

3 Requisiti e vincoli di progetto

Una prima categorizzazione viene effettuata tra requisiti funzionali e non funzionali.

3.1 Requisiti Funzionali

1. Consentire la creazione di un contatto con i seguenti campi:
   - Obbligatori: nome o cognome.
   - Opzionali: fino a tre numeri di telefono (ognuno con una categoria opzionale) e fino a tre indirizzi email (ognuno con una categoria opzionale).
2. Salvare i contatti nel sistema.
3. Modificare un contatto esistente.
4. Eliminare un contatto.
5. Ricercare un contatto.
6. Visualizzare un elenco di contatti con opzioni di ordinamento.
7. Creare una lista di contatti preferiti.
8. Importare una rubrica già esistente.
9. Esportare una rubrica.

3.2 Requisiti Non Funzionali

1. Il sistema deve supportare un massimo di 5000 contatti, mantenendo un tempo di risposta inferiore a 2 secondi per operazioni di ricerca e salvataggio.
2. I dati devono essere criptografati utilizzando l'algoritmo AES-256.
3. L'interfaccia utente deve essere intuitiva e accessibile.
4. L'applicazione deve essere compatibile con sistemi operativi Windows, MacOS e Linux.

4 Categorizzazione di dettaglio

La categorizzazione di dettaglio può seguire le sei dimensioni dei requisiti, mostrati e descritti qui di seguito.

4.1 Funzionalità individuali

Vengono qui elencati i servizi individuali (ed essenziali) che il sistema dovrà offrire:

- **[IF-1.1] Gestione CRUD** dei contatti all'interno della rubrica.
  - **CRUD:** Create Read Update Delete;
- **[IF-1.2] Ricerca di un contatto** in base ad uno specifico attributo, quale può essere un numero di telefono, un indirizzo email, il nome con cui è stato salvato, una sottostringa, o altri attributi che il sistema memorizza per un contatto;
- **[IF-1.3] Ordinamento della vista dei contatti** in base a criteri come ordine alfabetico in base a nome, cognome o altri attributi;
- **[IF-1.4] Gestione dei contatti** tramite lista preferiti o associazione ad una categoria *(Es. Personale, Lavoro, Scuola, ecc...)*;
- **[IF-2] Gestione di una rubrica** che l'utente ha creato/importato;
- **[IF-3.1] Salvataggio automatico della rubrica** su un file salvato in una directory di sistema *(Es. "AppData" di Windows)*;
- **[IF-3.2] Import/Export** di un'intera rubrica o di un singolo contatto (quest'ultimo tramita standard vCard, utilizzato anche da Android, iOS, ...);

4.2 Business flow (Scenari)

Vengono qui elencatyi le interazioni tra uno o più utenti e il sistema, per la realizzazione di uno specifico processo:

- **[BF-1] Registrazione di un contatto**
  - L'utente vuole registrare un nuovo contatto all'interno della propria rubrica. Vengono richiesti dunque dal sistema, obbligatoriamente almeno il nome o il cognome del contatto. La registrazione non può avvenire se non vengono compilati i campi obbligatori, il resto è facoltativo. A registrazione avvenuta, l'utente potrà vedere che il nuovo contatto è stato registrato, poiché sarà visibile fin da subito nella lista di tutti i suoi contatti.
- **[BF-2] Modifica di un contatto**
  - L'utente vuole effettuare una modifica ad un contatto che è presente nella propria rubrica. Può effettuare qualunque modifica a qualunque campo presente, affinché vengano rispettati i requisiti precedentemente richiesti (ovvero la presenza obbligatoria di almeno il nome o il cognome).

- **[BF-3] Eliminazione di un contatto**
  - L'utente vuole eliminare un contatto presente nella propria rubrica. Se questa azione verrà effettuata nella lista generale, allora il contatto verrà eliminato da qualsiasi lista categorizzata presente all'interno della rubrica. Se l'azione verrà invece effettuata all'interno di una lista specifica (es. preferiti), allora l'eliminazione avrà effetto solo su quella lista, e non su tutte quelle presenti.
- **[BF-4] Salvataggio/Lettura di una rubrica su/da file**
  - Di default, il software effettuerà in automatico operazioni di salvataggio e lettura da un file che verrà salvato in una directory di sistema. Questo viene fatto per garantire che all'apertura del software, l'utente potrà subito interagire con gli elementi che ha inserito, e che alla chiusura, tutte le modifiche effettuate vengono salvate. Per prevenire la perdita di modifiche non ancora salvate (per cause esterne e non controllabili dal programma, come crash, blackout, ecc...) viene fatto anche un salvataggio periodico in background durante il runtime del programma.

4.3 Esigenze di dati e informazioni

Vengono qui elencati e descritti i dati che il sistema deve gestire:

- **[DF-1] Dati dei contatti**
  - Dati anagrafici generali dell'utente (nome, cognome) e informazioni di contatto (indirizzo email, numero di telefono).
- **[DF-2] Sistema operativo**
  - Famiglia di OS attualmente in uso dall'utente (Windows, Linux o macOS) per effettuare il salvataggio nell'apposita directory di sistema per le applications data.
- **[DF-3] Chiave di crittografia**
  - Chiave usata per codificare/decodificare il file di salvataggio automatico della rubrica dell'utente.

4.4 Interfaccia utente

Vengono qui elencati le caratteristiche e gli elementi che si concentrano sulla progettazione e usabilità dell'interfaccia utente da realizzare:

- **[UI-1.1] Dashboard**
  - Deve visualizzare tutti i contatti fino ad ora registrati, oltre ad una barra di menù per effettuare operazioni di massa sulla lista, delle tab per poter cambiare lista di visualizzazione (es. Generale, Preferiti, Lavoro, ecc...), il numero di contatti inseriti.

- **[UI-1.2] Scheda contatto**
  - Riepilogo di tutti i dati inseriti per il contatto che è stato selezionato. Devono essere messi a disposizioni anche dei bottoni per poter effettuare modifiche sul singolo contatto o per poterlo eliminare.
- **[UI-2] Design reattivo**
  - Il software deve essere accessibile, intuitivo e di facile utilizzo. Per questa applicazione è previsto l'uso esclusivamente da desktop o PC tablet, dunque la finestra e i layout interni all'applicazione devono essere reattivi e potersi adattare in base alla dimensione della finestra.
- **[UI-3] Messaggi di errore**
  - In caso di errori da parte dell'utente, l'applicazione deve indicare chiaramente dove e perchè si è verificato l'errore, e una possibile soluzione. Il tutto deve essere indicato in maniera chiara e trasparente, in modo da poter rendere utilizzabile l'applicazione anche agli utenti meno esperti.

4.5 Altre interfacce con sistemi/piattaforme esterne

Vengono qui elencati i modi in cui il sistema interagisce con altri sistemi esterni:

- *Il sistema attualmente non prevede l'utilizzo di interfacce con sistemi esterni.*

4.6 Ulteriori vincoli non funzionali

- **[FC-1] Prestazioni**
  - Il sistema deve supportare un massimo di 5000 contatti, mantenendo un tempo di risposta inferiore a 2 secondi per qualsiasi operazione che viene fatta sulla lista di contatti;
- **[FC-2] Sicurezza**
  - I dati che vengono salvati e letti dal file presente nella directory di sistema sono crittografati utilizzando l'algoritmo di cifratura AES-256. Opzionalmente, è possibile anche crittografare con una propria password l'export della rubrica;
- **[FC-3] Accessibilità**
  - L'interfaccia utente (GUI) deve avere una UI e una UX abbastanza curata, dunque di facile utilizzo, ma soprattutto accessibile.
- **[FC-4] Compatibilità**
  - L'applicazione deve essere compatibile con i sistemi operativi Windows, macOS e le varie distribuzioni Linux attualmente disponibili.

4.7 Schema di categorizzazione

Area dei requisiti	Prefisso	Numerazione delle dichiarazioni dei requisiti
Funzionalità individuali	IF	IF-1.1, IF-1.2, IF-1.3, IF-1.4, IF-2, IF-3.1, IF-3.2
Business flow	BF	BF-1, BF-2, BF-3, BF-4
Dati e formato dei dati	DF	DF-1, DF-2, DF-3
Interfaccia utente	UI	UI-1.1, UI-1.2, UI-3, UI-4
Interfaccia con i sistemi	IS	-
Ulteriori vincoli	FC	FC-1, FC-2, FC-3, FC-4

Tabella 1: La Tabella 1 mostra chiaramente ogni categoria ed i requisiti ad essa associata. E' esplicitato il prefisso con il quale riconoscere la categoria ed il numero del requisito, richiamando la descrizione fatta nei punti precedenti.

5 Valutazione dei requisiti

5.1 Priorità dei requisiti

Questo elenco aiuta a definire le priorità di ciascun requisito. Si è scelto di valutare i requisiti in base al criterio del business value - dunque ad ogni requisito verrà assegnato un livello di priorità tra i tre disponibili:

- **Alto (Must have)** - se il requisito non è soddisfatto, il sistema non è utilizzabile;
- **Medio (Should have)** - importante, ma non comporta un fallimento se omesso;
- **Basso (Nice to have)** - da soddisfare solo se non richiede grandi sforzi;

Livello di priorità	**Requisito**
Alto	• Gestione CRUD dei contatti; • Ricerca di un contatto; • Ordinamento dei contatti; • Salvataggio automatico della rubrica; • Interfaccia utente;
Medio	• Gestione della lista dei contatti preferiti; • Accessibilità dell'interfaccia; • Sicurezza tramite crittografia sui file di salvataggio; • Efficienza del sistema;
Basso	• Messaggi di errore; • Import/Export di una rubrica da/su file;

Tabella 2: La tabella mostra il livello di priorità associato ad ogni requisito, ottenuto valutando il grado di necessità di ognuno per il corretto funzionamento del sistema.

5.2 Rischio tecnico

Questo elenco aiuta a definire il rischio tecnico di ciascun requisito. Ad ogni requisito verrà assegnato un grado di rischio tra i tre disponibili:

- **Alto** - è difficile prevedere se si riuscirà a realizzare il requisito (in assoluto, o semplicemente nei tempi e nei costi previsti);
- **Medio** - si ritiene di poter realizzare il requisito, ma ci sono alcuni fattori di rischio che non possono essere controllati;
- **Basso** - si ha piena fiducia nella propria capacità di realizzarlo;

Grado di rischio	**Requisito**
Alto	• Accessibilità dell'interfaccia; • Sicurezza tramite crittografia sui file di salvataggio; • Efficienza del sistema;
Medio	• Salvataggio automatico della rubrica; • Ordinamento dei contatti; • Gestione della lista dei contatti preferiti; • Import/Export di una rubrica da/su file;
Basso	• Interfaccia utente; • Ricerca; • Gestione CRUD dei contatti; • Messaggi di errore;

Tabella 3: La tabella mostra il grado di rischio associato ad ogni requisito, ottenuto valutando la difficoltà di implementazione di ognuno nel rispetto delle scadenze concordate.

6 Casi d'uso

6.1 Diagramma dei casi d'uso

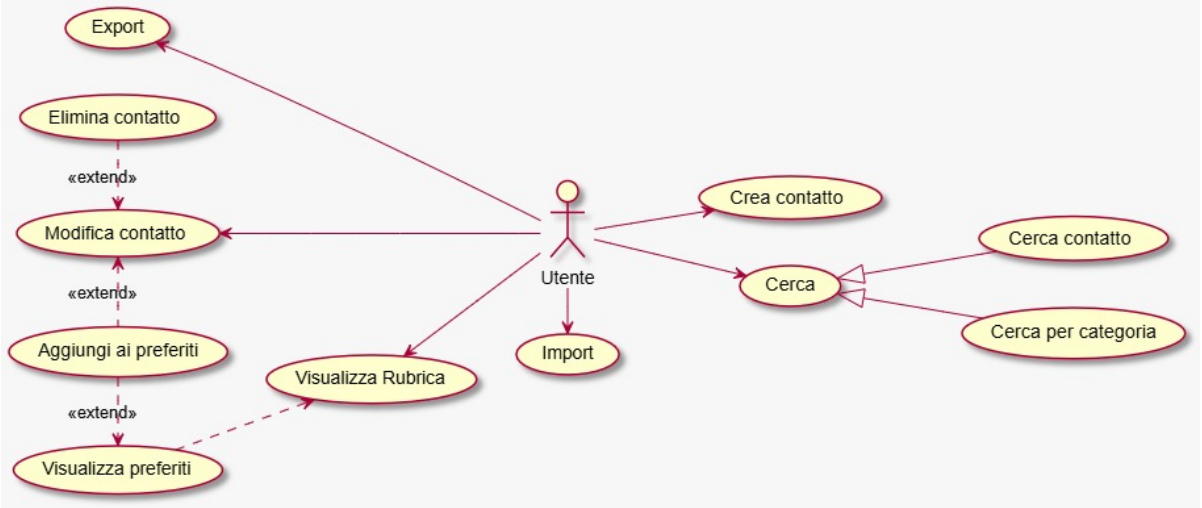

Figura 1: Il diagramma dei casi d'uso esplicita le varie funzionalità del sistema con gli attori che le attivano ed utilizzano. Delle operazioni si può anche capire il rapporto che queste hanno tra loro grazie al tipo di relazione che le lega.

6.2 Descrizione dei casi d'uso

Nome:	**Creazione di un contatto**
Attore partecipante:	Utente
Precondizioni:	Nessuna
Postcondizioni:	Salvataggio del contatto sulla rubrica.
Flusso di eventi:	1. L'utente visualizza una schermata con diversi campi 2. Il cliente compila i campi obbligatori (nome o cognome) ed eventualmente i campi opzionali (da 0 a 3 numeri di telefono e da 0 a 3 email con la possibilità di categorizzare ognuno di essi). 3. Il sistema salva il nuovo contatto.

Nome:	**Modifica/Eliminazione di un contatto**
Attore partecipante:	Utente
Precondizioni:	Esistenza in rubrica del contatto da modificare
Postondizioni:	Modifica richiesta apportata al contatto in rubrica
Flusso di eventi:	1. L'utente sceglie il contatto a cui apportare una modifica 2. L'utente sceglie l'opzione "Modifica contatto" 3. Si apre una schermata con le informazioni del contatto selezionato 4. L'utente modifica le informazioni 5. La modifica è apportata al contatto nell'elenco
Flusso alternativo:	a. Eliminazione del contatto 2a. L'utente sceglie l'opzione "Elimina contatto" 3a. Il contatto viene eliminato dalla rubrica b. Aggiungi ai preferiti 2b. L'utente sceglie l'opzione "Aggiungi ai preferiti" 3b. Il contatto è aggiunto alla lista dei preferiti

Nome:	**Ricerca di un contatto**
Attore partecipante:	Utente
Precondizioni:	Nessuna
Postcondizioni:	Risultati coerenti con la ricerca effettuata
Flusso di eventi:	1. L'utente seleziona la ricerca nell'elenco 2. L'utente ricerca per nome o per cognome 3. Il sistema reagisce dinamicamente durante l'inserimento dei dati, mostrando i risultati coerenti de presenti
Flusso alternativo:	a. Ricerca di una categoria 2a. L'utente seleziona la ricerca per Categoria 3a. Il sistema mostra le Categorie presenti 4a. L'utente seleziona la Categoria nella lista 5a. Il sistema mostra gli elementi che rispondono alla ricerca. b. Nessun risultato coerente 3b. Il sistema mostra una schermata vuota nel momento in cui i criteri non corrispondono a nessun contatto, cambiare i dati in qualsiasi momento aggiorna la ricerca dinamicamente

Nome:	**Visualizzazione contatti**
Attore partecipante:	Utente
Precondizioni:	Nessuna
Postcondizioni:	Elenco visualizzato su schermo
Flusso di eventi:	1. L'utente esegue l'applicativo 2. Il sistema mostra l'elenco dei contatti in ordine alfabetico per cognome
Flusso alternativo:	a. Visualizzazione lista dei preferiti 3a. L'utente seleziona "lista dei preferiti" 4a. Il sistema mostra l'elenco dei contatti preferiti ab. Ampliamento della lista dei preferiti 5ab. L'utente seleziona l'opzione di aggiunta dei contatti alla lista preferiti 6ab. Il sistema mostra l'elenco 7ab. L'utente seleziona nell'elenco i contatti da aggiungere alla lista dei preferiti 8ab. I contatti selezionati sono aggiunti alla lista dei preferiti
Nome:	**Salvataggio dei contatti in un file**
Attore partecipante:	Utente
Precondizioni:	Esistenza del file da cui prendere i contatti
Postcondizioni:	Prodotto un file contenente i contatti della rubrica
Flusso di eventi:	1a. Il cliente seleziona l'opzione "Esporta" 2a. Il sistema salva il contenuto della rubrica in un file
Nome:	**Importazione dei contatti da un file**
Attore partecipante:	Utente
Precondizioni:	Esistenza del file da cui prendere i contatti
Postcondizioni:	I contatti contenuti nel file sono presenti in rubrica
Flusso di eventi:	1. Il cliente seleziona l'opzione "Importa" 2. Il sistema inserisce i contatti presenti nel file all'interno della rubrica
Flusso alternativo:	a. Formato del file non coerente 3a. Il sistema mostra una schermata di errore, torna al punto 1

7 Design architetturale

Gli attributi di qualità che il nostro sistema, a parere del team, dovrebbe rispettare, sono i seguenti:

7.1 Attributi di qualità esterni

- **Disponibilità:** Il sistema deve essere sempre pronto all'uso.
- **Efficienza:** Il sistema deve rispettare i vincoli di risposta.
- **Security:** Il sistema deve proteggere i dati dell'utente.
- **Usabilità:** Il sistema deve fornire un'interfaccia chiara e un alto livello di user experience.
- **Installabilità:** Il sistema deve essere di facile installazione.
- **Adattabilità:** Il sistema deve fornire un'interfaccia grafica adattabile alle capacità grafiche del dispositivo utilizzato dall'utente.

7.2 Attributi di qualità interni

- **Manutenibilità:** Il sistema deve essere progettato con un alto livello di coesione e un basso livello di accoppiamento per facilitare la manutenibilità.
- **Modularità:** Il sistema deve distribuire le responsabilità su più moduli funzionali.
- **Riusabilità:** Il sistema deve essere composto da moduli funzionali riutilizzabili in altri progetti.
- **Portabilità:** Il sistema deve essere progettato per poter essere eseguito su dispositivi eterogenei.
- **Testabilità:** Il sistema deve essere composto da moduli funzionali facilmente testabili.

7.3 Diagramma dei package

La progettazione del diagramma dei package di un'applicazione per una rubrica telefonica in Java utilizzando Maven, JavaFX e SceneBuilder dovrebbe seguire i principi di modularità e separazione delle responsabilità.

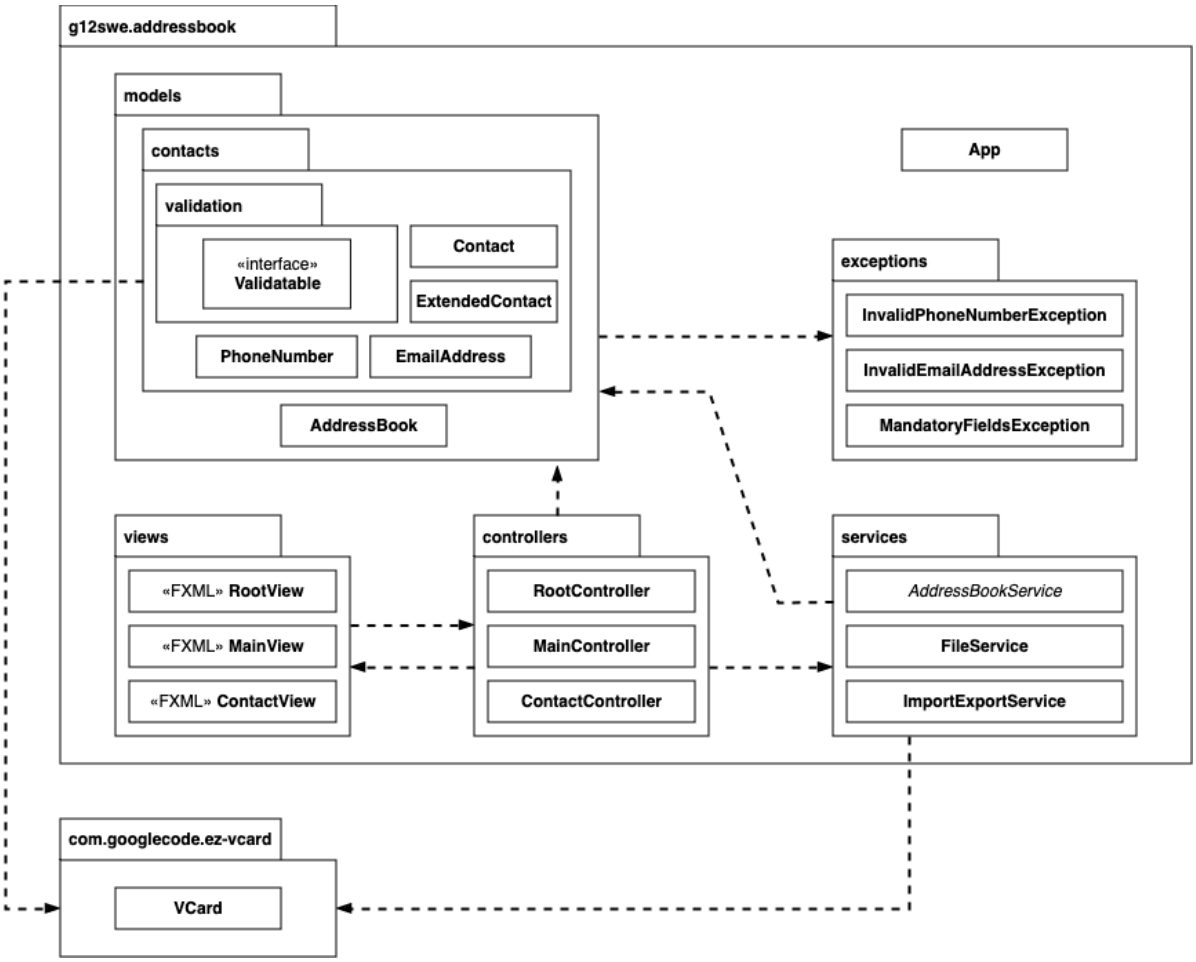

Figura 2: Diagramma dei package

Il diagramma dei package descrive la struttura del progetto attraverso le macrofunzioni implementate. L'intero progetto risiede nel package "addressbook", con diversi sotto package per le varie funzioni. Si è scelto di assegnare ad un package "services" la gestione delle operazioni su file, al package "models" la gestione della struttura dell'elenco ed ai package "view" e "controller" rispettivamente la vista e la gestione dell'interfaccia utente.

8 Design funzionale

Una documentazione più dettagliata relativa al design by contract è stata generata automaticamente da doxygen.

8.1 Diagramma delle classi

Il diagramma delle classi, generato automaticamente da EasyUML, descrive le varie classi utilizzate nel progetto. Mostra le informazioni relative alla visibilità degli attributi e dei metodi di ogni classe, esplicitando anche le relazioni tra classi.

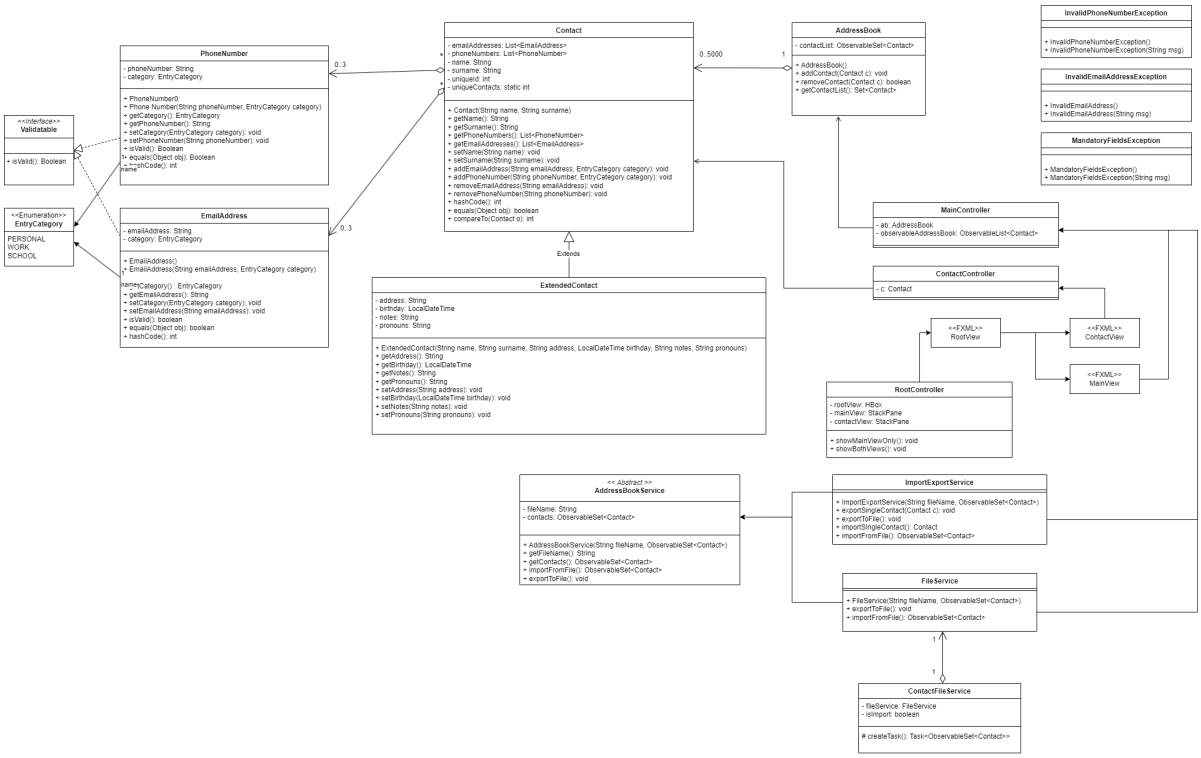

Figura 3:

8.2 Diagrammi di sequenza

I seguenti diagrammi di sequenza descrivono il flusso di eventi per le operazioni principali della nostra rubrica. Si è deciso di realizzare i diagrammi per le operazioni di creazione, modifica e cancellazione di un contatto, nonché di import della rubrica in un file ed export della stessa da file esterni. Si è cercato di affidare ad ogni componente un chiaro ambito di responsabilità, infatti:

- MainView è responsabile dell'interfaccia utente, mostrare moduli e messaggi di errore.
- MainController gestisce la logica dell'applicazione, orchestrando le interazioni tra l'interfaccia utente e i servizi sottostanti.
- AddressBookService si occupa esclusivamente delle operazioni sui dati della rubrica (recupero e aggiornamento).

- Validatable è responsabile della convalida dei dati, isolando così la logica di validazione.
- VCard si occupa della conversione dei contatti nel formato omonimo.

Questa separazione delle responsabilità, a parere del team, mantiene alta la coesione poiché ogni modulo si concentra su un singolo aspetto delle varie funzionalità. La struttura presenta un basso accoppiamento poiché MainController funge da intermediario tra MainView e la logica sottostante dell'applicazione riducendo il diretto accoppiamento tra l'interfaccia utente e servizi come quello fornito dall'interfaccia Validatable. Infatti tale interfaccia separa gli oneri di convalida dei contatti dal controller, rendendo la logica di validazione riutilizzabile in altri contesti. I diagrammi rispettano il Single Responsibility Principle (SRP) dato che ogni componente ha una responsabilità unica e ben definita. Inoltre viene rispettato l'Open-Closed Principle poiché la struttura si mostra aperta per estensioni (ad esempio, aggiungere nuove regole di validazione) ma chiusa per modifiche dirette, in particolare grazie all'interfaccia Validatable. Infine, il MainController è stato pensato per non interagire direttamente con i componenti sottostanti ma attraverso le loro interfacce/contratti, facilitando l'inversione delle dipendenze, rispettando così il Liskov Substitution Principle (LSP). Questo approccio alla progettazione aumenterà la flessibilità e la testabilità della successiva implementazione della nostra applicazione.

- **Aggiunta di un contatto**

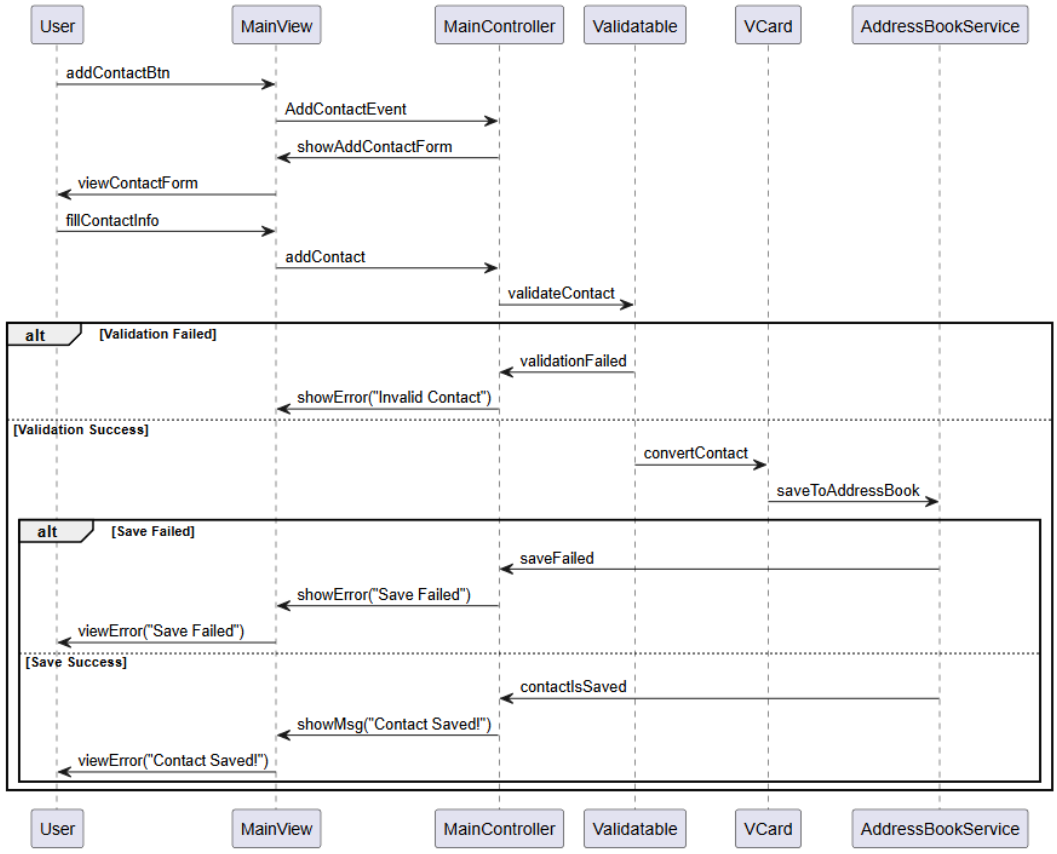

Figura 4: diagramma di sequenza per l'operazione di aggiunta di un contatto. L'utente interagisce direttamente con la MainView, la quale si occupa a sua volta di comunicare le scelte dell'utente al MainController. Ricevute le informazioni, il MainController si occupa di comunicare con le varie interfacce al fine di effettuare le operazioni più opportune relativamente alle scelte dell'utente. Il diagramma mostra anche i flussi degli eventi relativi a un eventuale fallimento nella validazione o nel salvataggio dei contatti. A seconda dei casi, l'interfaccia Validatable o l'AddressBookService genera un errore che il MainController si occupa di comunicare all'utente attraverso un messaggio visualizzato dalla MainView.

- **Eliminazione di un contatto**

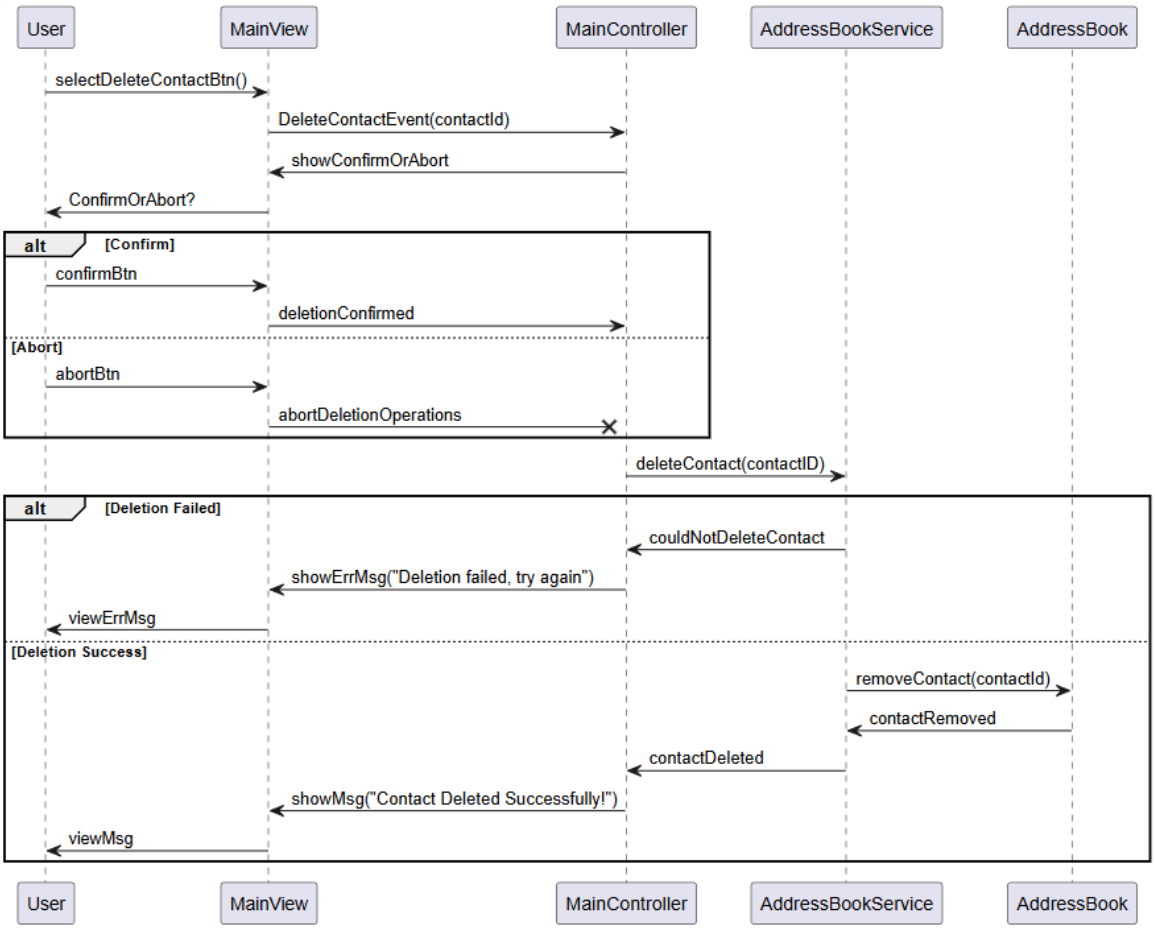

Figura 5: diagramma di sequenza per l'eliminazione di un contatto. L'utente seleziona l'opzione appropriata e visualizza a schermo una richiesta di conferma di cancellazione. L'utente può scegliere se confermare o annullare l'operazione: nel primo caso si prosegue con l'eliminazione, nel secondo caso il flusso termina. Il secondo flusso alternativo viene scatenato nell'eventualità in cui l'eliminazione non vada a buon termine, e in tal caso viene mostrato un messaggio di errore all'utente, al quale viene chiesto di ritentare.

- **Modifica di un contatto**

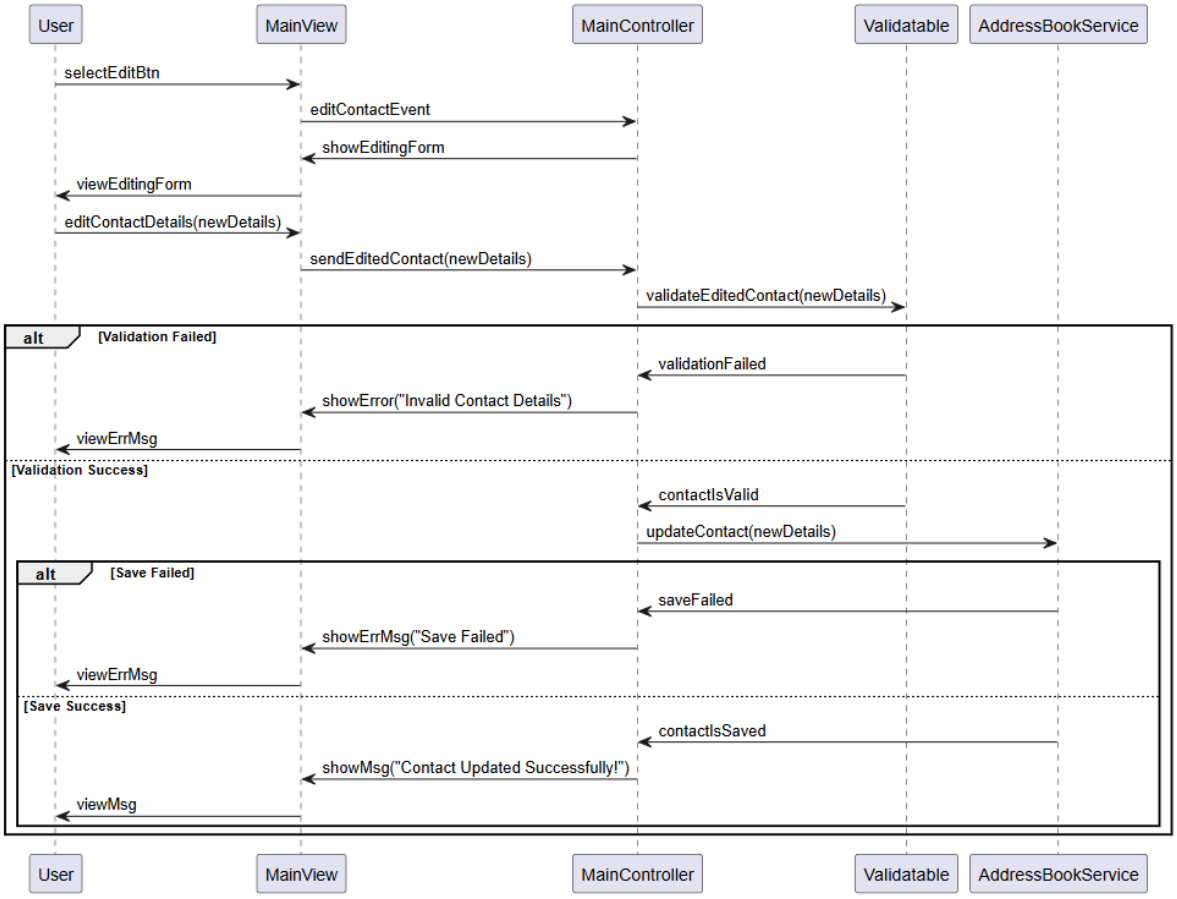

Figura 6: diagramma di sequenza per la modifica di un contatto. Il flusso degli eventi è molto simile a quello relativo all'aggiunta di un nuovo contatto. La MainView genera un evento (editContactEvent) e visualizza il modulo di modifica attraverso il metodo showEditingForm. Una volta compilato il modulo, i dettagli modificati vengono inviati al MainController tramite il metodo sendEditedContact. Il MainController, a sua volta, delega la validazione dei nuovi dettagli al componente Validatable, come nel caso dell'aggiunta di un nuovo contatto. Nel caso in cui la validazione fallisca, il sistema gestisce l'errore visualizzando un messaggio di errore all'utente tramite showError in MainView mentre in caso di validazione riuscita, il MainController invoca il metodo updateContact dell'AddressBookService. Se il salvataggio fallisce, il metodo viewErrMsg della MainView comunica il problema all'utente con il messaggio "Save Failed", altrimenti viene mostrato un messaggio di conferma.

- **Export dei contatti**

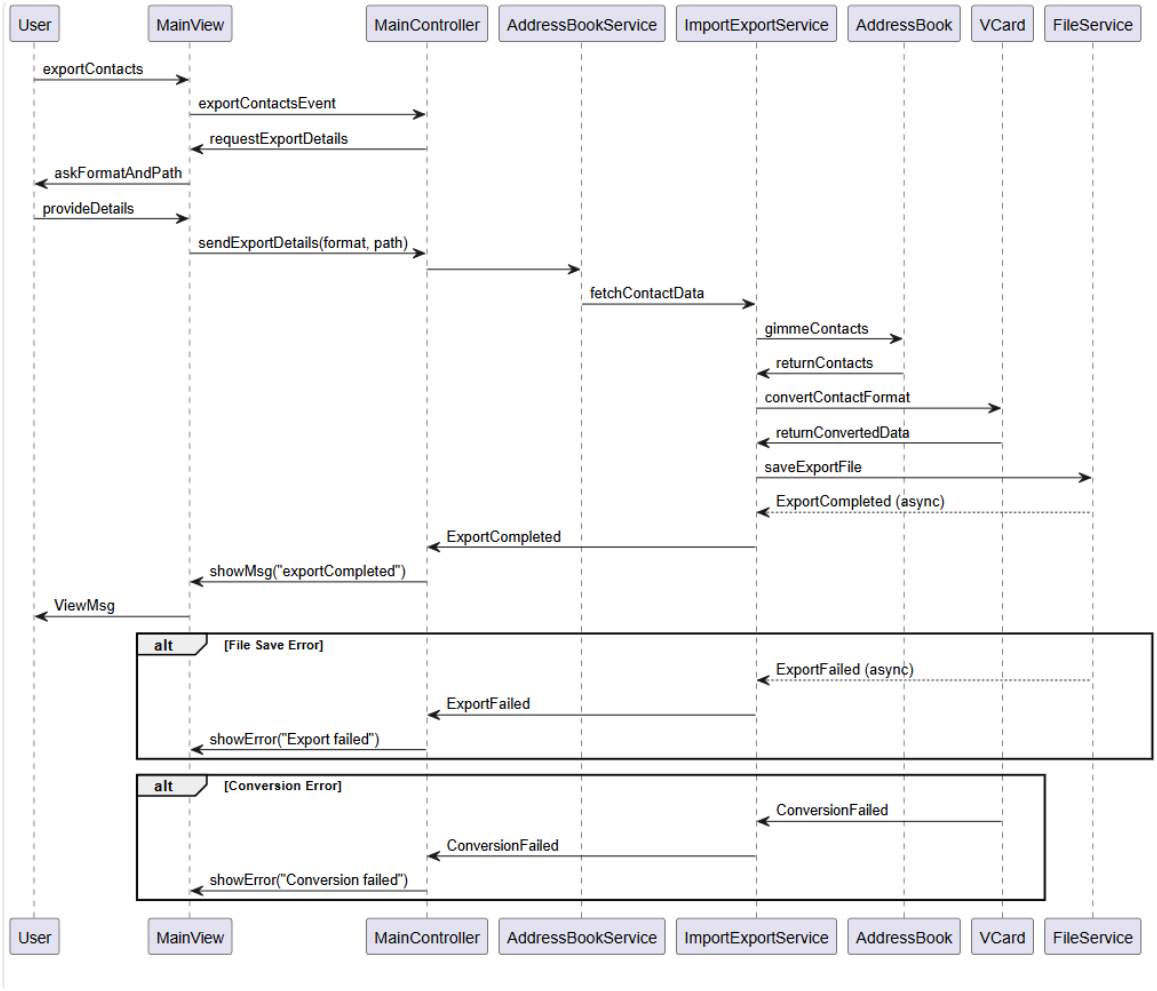

Figura 7: diagramma di sequenza per l'esportazione dei contatti. L'utente avvia il processo selezionando l'opzione exportContacts nella MainView, che genera un evento exportContactsEvent. Il MainController richiede i dettagli di esportazione tramite il metodo requestExportDetails. Dopo aver ricevuto i dettagli richiesti attraverso il metodo provideDetails, il MainController invia i dati di esportazione all' AddressBookService, il quale richiama getContactList per ottenere i dati dei contatti dall'AddressBook. I dati vengono quindi convertiti nel formato richiesto utilizzando convertContactFormat del componente VCard, che restituisce i dati convertiti tramite returnConvertedData. Successivamente, il FileService salva i dati esportati utilizzando saveToExportFile. Se l'operazione ha successo, viene notificata all'utente tramite un messaggio di conferma showMsg dalla MainView altrimenti, se si verifica un errore durante la conversione dei dati, il sistema genera un errore ConversionFailed e mostra un messaggio di errore all'utente tramite showError. In caso di fallimento durante il salvataggio del file, viene generato un errore ExportFailed, e l'utente viene informato con un messaggio di errore showErrMsg dalla MainView.

- **Import dei contatti**

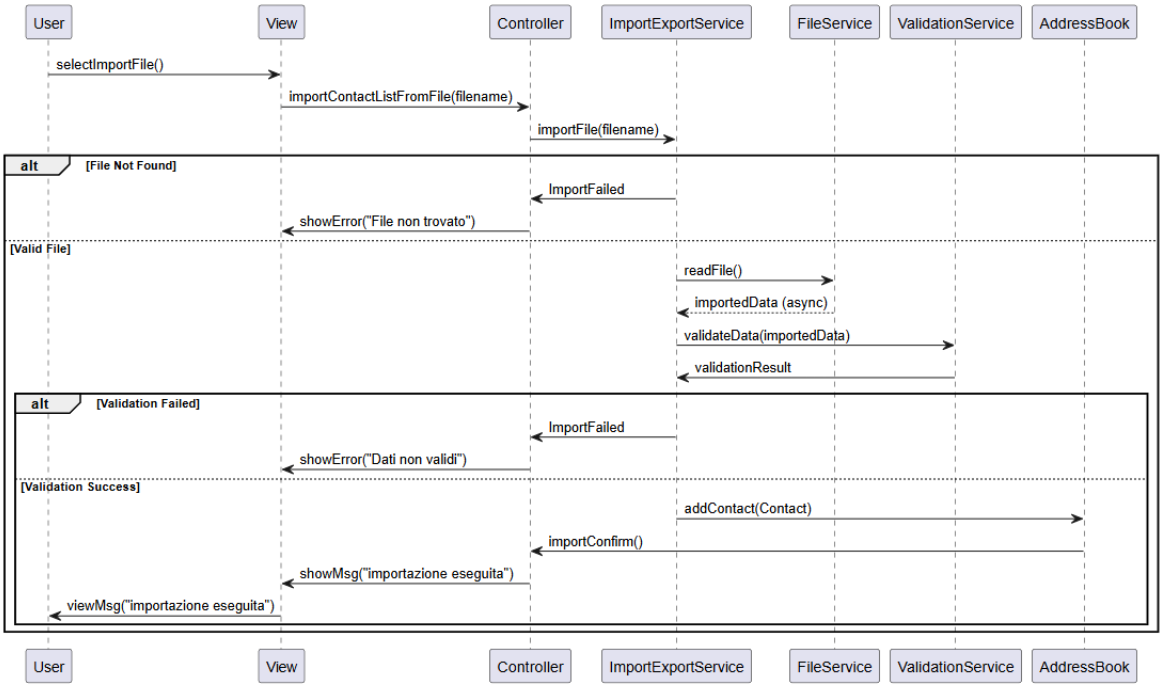

Figura 8: diagramma di sequenza per l'operazione di importazione di una lista di contatti. L'utente avvia il processo selezionando un file tramite il metodo selectImportFile nella MainView, che inoltra la richiesta al MainController attraverso il metodo importContactListFromFile. Il MainController delega l'importazione all' AddressBookService, che coinvolge l'ImportExportService per l'elaborazione del file tramite il metodo importFile. Se il file risulta non valido o non leggibile, viene generato un errore unreadableFile, e viene mostrato all'utente un messaggio di errore tramite la MainView. Se il file è valido, i dati vengono letti dal FileService tramite readFile e inviati al Validatable per la validazione e convertiti nel formato appropriato in caso di validazione riuscita. In caso di dati non validi, viene generato l'errore ImportFailed e la MainView notifica l'utente con un messaggio di errore, altrimenti il MainController procede all'importazione dei contatti nell'AddressBook tramite importContacts. L'operazione si conclude con un messaggio di conferma mostrato all'utente tramite il metodo showMsg della MainView, indicando che l'importazione è stata completata con successo.

9 Implementazione e testing

9.1 Suddivisione del lavoro e strumenti utilizzati

La suddivisione del lavoro nel progetto è stata organizzata utilizzando gli strumenti collaborativi offerti da **GitHub**, che ci hanno permesso di lavorare in modo efficace, anche in modalità asincrona. Di seguito viene descritto il nostro approccio.

Gestione delle Attività

Le attività sono state tracciate attraverso **GitHub Issues**, dove ogni membro del team ha avuto la possibilità di:

- Creare e assegnare attività specifiche, suddivise in funzionalità principali e sottocompiti.
- Monitorare lo stato di avanzamento e segnalare eventuali problematiche.
- Etichettare le attività con tag descrittivi (`bug`, `documentation`, `implementation`, etc.) per una migliore classificazione.

Pianificazione con GitHub Projects

Per coordinare le attività e rispettare le scadenze, abbiamo utilizzato **GitHub Projects**, organizzando i compiti in tre categorie principali:

- **To Do:** Attività ancora da svolgere.
- **In Progress:** Lavori in corso.
- **Done:** Attività completate.

L'utilizzo di questa bacheca ci ha permesso di visualizzare chiaramente lo stato del progetto e di aggiornarlo in tempo reale, facilitando la collaborazione.

Vantaggi dell'approccio utilizzato

L'adozione di questi strumenti ha garantito:

- **Collaborazione fluida:** Ogni membro del team ha avuto una visione chiara del lavoro da svolgere.
- **Flessibilità:** Possibilità di lavorare anche in modo asincrono, senza vincoli di orario.
- **Efficacia:** Monitoraggio costante dei progressi e rapida identificazione di eventuali problemi (bug, errori nella documentazione, ecc...)

9.2 Convenzioni utilizzate

Nel corso dello sviluppo del progetto, abbiamo seguito un insieme di convenzioni di codifica ben definite, con l'obiettivo di mantenere il codice leggibile, coerente e facilmente manutenibile. Di seguito sono descritte le principali linee guida adottate.

Documentazione del codice

Tutti i metodi, le classi e gli attributi sono stati documentati utilizzando lo standard **Doxygen**, uno strumento versatile per generare documentazione tecnica direttamente dal codice sorgente. La documentazione è stata redatta interamente in **inglese**, per favorire la comprensione anche in contesti internazionali e per allinearci alle buone pratiche di sviluppo software. Ogni funzione include una descrizione dettagliata del suo scopo, dei parametri in ingresso e del valore restituito, ove applicabile, e delle pre-condizioni, post-condizioni e invarianti che ogni metodo deve rispettare.

Naming convention

Per mantenere uniformità nel codice, sono state adottate le seguenti regole di naming convention:

- **Metodi e Attributi:** È stato utilizzato il formato *camelCase*, in cui il nome inizia con una lettera minuscola, mentre ogni parola successiva inizia con una lettera maiuscola (esempio: `addContact()`, `contactList`).
- **Classi:** I nomi delle classi e delle interfacce seguono lo stile *PascalCase*, in cui ogni parola inizia con una lettera maiuscola (esempio: `ContactManager`, `AddressBook`).
- **Costanti:** Le costanti sono definite utilizzando lettere maiuscole e underscore per separare le parole (esempio: `MAX_CONTACTS`).

Organizzazione del Codice

Il codice è stato organizzato seguendo una struttura modulare, con separazione delle responsabilità tra i diversi componenti:

- La logica di business è implementata in classi dedicate (contenute nel package `g12swe.addressbook.models`).
- L'interfaccia grafica è separata dalla logica applicativa grazie al pattern **MVC (Model-View-Controller)**.
- I test unitari sono collocati in una directory separata, seguendo le convenzioni del framework **JUnit**.

9.3 Tech stack utilizzato

Per lo sviluppo del progetto della Rubrica, è stato adottato un insieme di tecnologie e strumenti moderni, selezionati per garantire efficienza, modularità e facilità di manutenzione.

Linguaggio di Programmazione: Java

Il cuore del progetto è stato sviluppato in **Java**, un linguaggio di programmazione orientato agli oggetti, noto per la sua robustezza e portabilità. Nello specifico, si è scelto di utilizzare, per lo sviluppo, una versione della JDK (Java Development Kit) pari o superiore alla 17.

Framework per l'Interfaccia Grafica: JavaFX

L'interfaccia grafica del progetto è stata realizzata con **JavaFX**, un framework per lo sviluppo di interfacce utente ricche e interattive, dove ciascuna interfaccia è stata realizzata grazie a software come SceneBuilder, e personalizzata grazie all'utilizzo di linguaggi di markup come FXML (variante di XML per definire la struttura delle interfacce JavaFX) e fogli di stile in CSS.

Build e gestione dependencies: Maven

Per semplificare la gestione del ciclo di vita del progetto, è stato utilizzato **Apache Maven**, un tool fondamentale per l'automazione del build, del test e la gestione delle dipendenze. Grazie ad esso, è stato possibile integrare facilmente librerie e plug-in, mantenendo un alto livello di organizzazione del codice e dei package. Il suo utilizzo è stato fondamentale soprattutto per il download automatico delle librerie necessarie al funzionamento del programma in base all'architettura, o alla famiglia di OS, che i vari membri del team utilizzano.

Testing: JUnit

La qualità del software è stata garantita attraverso l'implementazione di test unitari utilizzando **JUnit**. Questo framework ha consentito di validare le funzionalità critiche, riducendo i rischi di bug e errori logici più o meno gravi che potevano influire sull'esperienza d'uso dell'utente.

Libreria per Gestione Contatti: ez-vCard

Per gestire l'importazione e l'esportazione di contatti, è stata integrata la libreria **ez-vCard**. Questa soluzione offre un supporto completo per il formato standard vCard, assicurando l'interoperabilità con altre applicazioni e sistemi operativi (es. Micorosft Outlook, Android, iOS, ecc...)

Documentazione del Codice: Doxygen

La documentazione del progetto è stata realizzata utilizzando **Doxygen**, uno strumento utile per generare documentazione tecnica direttamente dal codice sorgente, grazie a commenti standardizzati. L'intera documentazione prodotta viene successivamente distribuita tramite **GitHub Pages**.

Automazioni con GitHub Actions

Come elemento aggiuntivo nel nostro progetto, abbiam deciso di includere l'utilizzo di **GitHub Actions** per automatizzare alcune operazioni, ovvero:

- **Build e test automatici:** Ad ogni push nel branch `main` del repository, GitHub Actions esegue automaticamente la build del progetto e tutti i test unitari definiti.
- **Release automatica:** Se la build va a buon fine, e i test vengono superati con successo, allora viene generata una nuova release del nostro prodotto, con la pubblicazione del pacchetto eseguibile `.jar` prodotto da Maven.
- **Generazione della documentazione Doxygen:** La documentazione prodotta con Doxygen viene automaticamente pubblicata su **GitHub Pages** (in un branch a parte), garantendo un accesso sempre aggiornato e centralizzato.

9.4 Build e test automatizzati

L'obiettivo di questo documento è fornire una panoramica dettagliata delle attività di testing condotte durante lo sviluppo del software. Il testing è stato eseguito per verificare la funzionalità dei requisiti specificati. Abbiamo adottato una combinazione di test unitari, di integrazione, funzionali e non funzionali. Abbiamo utilizzato JUnit per automatizzare i test unitari. Un test è stato considerato superato quando:

- Il risultato effettivo coincide con il risultato atteso.
- Non sono emersi errori critici o comportamenti anomali.

ID Test	Descrizione	Tipo di Test	Casi di Test	Risultato atteso	Risultato ottenuto	Stato
T005	Verifica della validità di un indirizzo email	Unit Test	Creazione di una email con dati non validi	Email non valida	Email non valida	Passato
T007	Confronto tra due indirizzi email	Unit Test	Due email diverse	Confronto non riuscito	Confronto non riuscito	Passato
T008	Confronto tra due indirizzi email	Unit Test	Due email uguali	Confronto riuscito	Confronto riuscito	Passato

T014	Verifica della validità di un numero di telefono	Unit Test	Creazione di un numero di telefono con dati validi	Numero di telefono valido	Numero di telefono valido	Passato
T016	Confronto tra due numeri di telefono	Unit Test	Due numeri di telefono diversi	Confronto non riuscito	Confronto non riuscito	Passato
T017	Confronto tra due numeri di telefono	Unit Test	Due numeri di telefono uguali	Confronto riuscito	Confronto riuscito	Passato
T031	Aggiunta di un indirizzo email in un contatto	Unit Test	Email valida	Email aggiunta correttamente	Email aggiunta correttamente	Passato
T032	Aggiunta di un numero di telefono in un contatto	Unit Test	Numero di telefono valido	Numero di telefono aggiunto correttamente	Numero di telefono aggiunto correttamente	Passato
T033	Rimozione di un indirizzo email da un contatto	Unit Test	Email presente	Email rimossa correttamente	Email rimossa correttamente	Passato
T034	Rimozione di un indirizzo email da un contatto	Unit Test	Email non presente	Email non rimossa	Email non rimossa	Passato
T035	Rimozione di un numero di telefono da un contatto	Unit Test	Numero di telefono presente	Numero di telefono rimosso	Numero di telefono rimosso	Passato
T036	Rimozione di un numero di telefono da un contatto	Unit Test	Numero di telefono non presente	Numero di telefono non rimosso	Numero di telefono non rimosso	Passato
T039	Equals di due contatti	Unit Test	Uno dei due contatti è null	Confronto non riuscito	Confronto non riuscito	Passato
T040	Equals di due contatti	Unit Test	I due contatti hanno nomi diversi	Confronto non riuscito	Confronto non riuscito	Passato
T041	Equals di due contatti	Unit Test	I due contatti hanno nomi uguali	Confronto non riuscito	Confronto non riuscito	Passato

T042	Equals di due contatti	Unit Test	I due contatti sono lo stesso oggetto	Confronto riuscito	Confronto riuscito	Passato
T043	CompareTo di due contatti	Unit Test	I due contatti sono diversi	Risultato diverso da 0	Risultato diverso da 0	Passato
T046	CompareTo di due contatti	Unit Test	I due contatti sono lo stesso oggetto	Risultato uguale a 0	Risultato uguale a 0	Passato
T062	Import di una rubrica da file binario	Unit Test	Il file contiene una rubrica	Rubrica correttamente importata	Rubrica correttamente importata	Passato
T063	Export di una rubrica su un file binario	Unit Test	La rubrica da salvare è non vuota	Il file creato ha dimensione maggiore di 0 byte	Il file creato ha dimensione maggiore di 0 byte	Passato
T064	Import di una rubrica da file vcf	Unit Test	Il file contiene una rubrica in formato vcf	Rubrica correttamente importata	Rubrica correttamente importata	Passato
T065	Export di una rubrica su un file vcf	Unit Test	La rubrica da salvare è non vuota	Il file creato in formato vcf ha dimensione maggiore di 0 byte	Il file creato in formato vcf ha dimensione maggiore di 0 byte	Passato
T066	Import di un singolo contatto vcard	Unit Test	Il contatto da importare esiste ed è salvato su un file in formato vcf	Il contatto è correttamente importato	Il contatto è correttamente importato	Passato
T067	Export di un singolo contatto vcard	Unit Test	Il contatto esiste in rubrica	Il file in formato vcf contenente il contatto è correttamente creato	Il file in formato vcf contenente il contatto è correttamente creato	Passato

T070	Aggiunta di un contatto alla rubrica	Unit Test	Il contatto è diverso da null	Contatto aggiunto in rubrica	Contatto aggiunto in rubrica	Passato
T071	Rimozione di un contatto dalla rubrica	Unit Test	Il contatto esiste in rubrica	Contatto rimosso dalla rubrica	Contatto rimosso dalla rubrica	Passato
T068	Verifica della velocità di carimento in rubrica di 5000 contatti	Stress Test	Un file binario contiene 5000 contatti validi	Tempo di caricamento inferiore a $2s$	Tempo di caricamento inferiore a $2s$	Passato
T069	Verifica della velocità di ricerca di un contatto in una rubrica di 5000 contatti	Stress Test	Il contatto è presente nella rubrica	Tempo di ricerca inferiore a $2s$	Tempo di ricerca inferiore a $2s$	Passato

Sono stati riportati solo i test più rilevanti. Ulteriori test, relativi a metodi getter, setter e hashcode, sono stati omessi dalla tabella per ragioni di leggibilità. Test di integrazione sono stati eseguiti man mano che il software venisse sviluppato e gli unit test ritornassero un esito positivo.