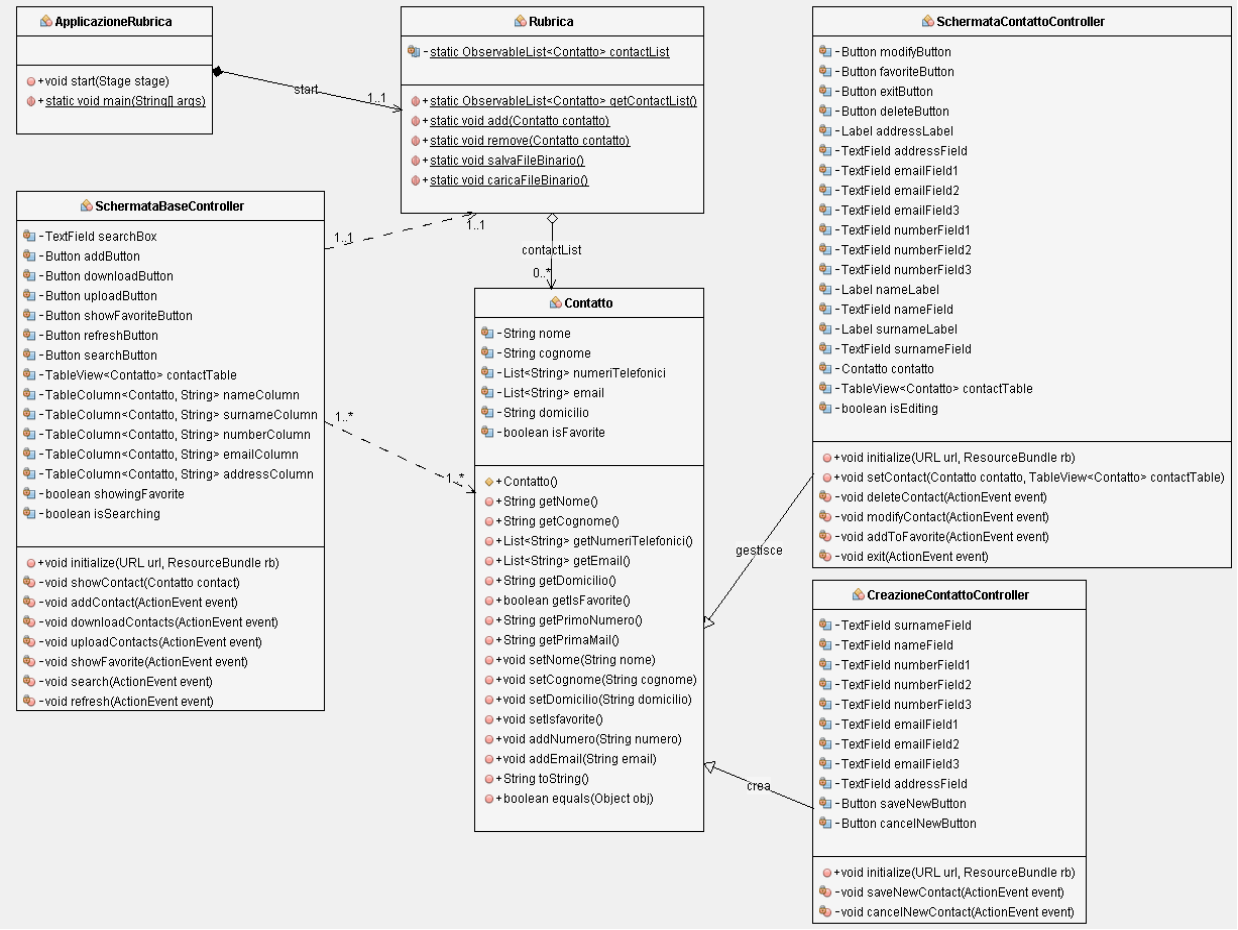
Documento Design

Descrizione Diagramma delle Classi



Il diagramma delle classi rappresenta un'applicazione per la gestione di una rubrica contatti, composta da varie classi che gestiscono la logica dell'applicazione e l'interfaccia grafica.

Il diagramma delle classi è stato realizzato cercando di mantenere un livello di coesione più alto possibile ed un livello di accoppiamento il più basso possibile, fatta eccezione per la dipendenza diretta dai componenti di tipo grafico, il tutto in modo semplice e facile da interpretare.

Per quanto riguarda i compiti di ciascuna classe, sono stati adottati i seguenti ragionamenti affiché ci sia un alto livello di coesione e un basso accopiamento :

- La classe **Contatto** è molto coesa in quanto si occupa esclusivamente della gestione delle informazioni relative ad un contatto (nome, cognome, numeri di telefono, email, domicilio e l'opzione "favorito"). Le operazioni fornite (come `setNome`, `addNumero`) sono strettamente legate alla gestione di questi dati.
- La classe **Rubrica** anche questa classe è coesa, dedicandosi alla gestione di una lista di contatti (`contactList`) e fornendo metodi statici per l'aggiunta, rimozione e salvataggio/caricamento dei dati. **Rubrica** utilizza

`ObservableList<Contatto>` per rappresentare la lista di contatti, il che riduce l'accoppiamento diretto con la classe `Contatto`, migliorando la flessibilità.

- Le classi **Controller (SchermataBaseController, SchermataContattoController, CreazioneContattoController)** Ogni controller è responsabile di una specifica parte dell'interfaccia utente, aumentando la coesione. Tuttavia, è importante mantenere il controller il più "snello" possibile, spostando la logica di business verso la classe modello (`Rubrica` o `Contatto`). I controller interagiscono con `Contatto` e `Rubrica`, creando una relazione di dipendenza necessaria.
- La classe **CreazioneContattoController** gestisce esclusivamente la logica di creazione di un nuovo contatto attraverso i text field ed i button. Le sue funzioni inoltre si limitano unicamente a salvare o annullare la creazione del contatto che in quel momento si sta andando a descrivere.
- La classe **SchermataContattoController** si occupa unicamente della visualizzazione e dell'eventuale modifica dei dettagli dei contatti.
- La classe **ApplicazioneRubrica** definisce solo il punto di partenza dell'applicazione e la gestione della finestra principale della rubrica.

Ad eccezion fatta per l'accoppiamento implicito tra i controller ed i corrispettivi componenti grafici, la separazione tra i vari controller è gestita affinché non vi siano dipendenze dirette che vincolano il comportamento di una classe al funzionamento dell'altra.

Stando ai Principi di Buona Progettazione:

- Il **Principio di Singola Responsabilità** è soddisfatto in quanto ogni classi si occupa di una funziona specifica:
  - **Contatto** gestisce esclusivamente i dati relativi ad un contatto della rubrica.
  - **Rubrica** gestisce una lista di contatti.
  - **SchermataBaseController** si occupa della gestione della rubrica.
  - **CreazioneContattoController** si occupa della creazione di nuovi contatti.
  - **SchermataContattoController** si occupa della visualizzazione e modifica di un singolo contatto.
- Le classi sono estensibili senza modifiche dirette al codice esistente. Ad esempio, si potrebbe aggiungere una nuova funzionalità per classificare i contatti senza alterare la struttura attuale della classe **Contatto**. Questo ragionamento fa riferimento al **Principio "Aperto/Chiuso"**.
- Non essendoci sottoclassi, come ad esempio delle specializzazioni della classe **Contatto**, il **Principio di Sostituzione di Liskov** è implicitamente rispettato. In caso contrario sarebbe necessario garantire che le nuove eventuali classi mantengano il comportamento della classe base.

- Per quanto riguarda il **Principio di Segregazione dell'Interfaccia**, le classi controller interagiscono unicamente con i componenti di cui hanno bisogno, mantenendo interfacce specifiche.
- In conclusione, per quanto riguarda il **Principio di Inversione della Dipendenza**, non si è ritenuta necessaria la realizzazione di classi astratte per cui questo principio è stato posto in secondo piano.

Relazioni tra le classi (SI PUO TOGLIERE è SOLO INDICATIVO)

Rubrica ↔ Contatto

- **Relazione**: Aggregazione
- **Cardinalità**: 1 ↔ 0..*
- Una rubrica è composta da una lista di contatti, ma i contatti possono esistere anche senza una rubrica (ad esempio, durante l'editing). Questa relazione è modellata usando `ObservableList<Contatto>`.

SchermataBaseController ↔ Contatto

- **Relazione**: Dipendenza
- **Cardinalità**: 1 ↔ 0..*
- Il controller utilizza i dati di `Contatto` per popolare la tabella nella vista. La relazione è temporanea e si limita all'interazione con l'interfaccia utente.

SchermataBaseController ↔ Rubrica

- **Relazione**: Dipendenza
- **Cardinalità**: 1 ↔ 1
- Il controller interagisce con la lista di contatti gestita dalla classe `Rubrica` per eseguire operazioni come ricerca, aggiunta o modifica di contatti.

CreazioneContattoController ↔ Contatto

- **Relazione**: Associazione
- **Cardinalità**: 1 ↔ 1
- Questa classe crea un nuovo oggetto `Contatto` utilizzando i dati immessi dall'utente.

SchermataContattoController ↔ Contatto

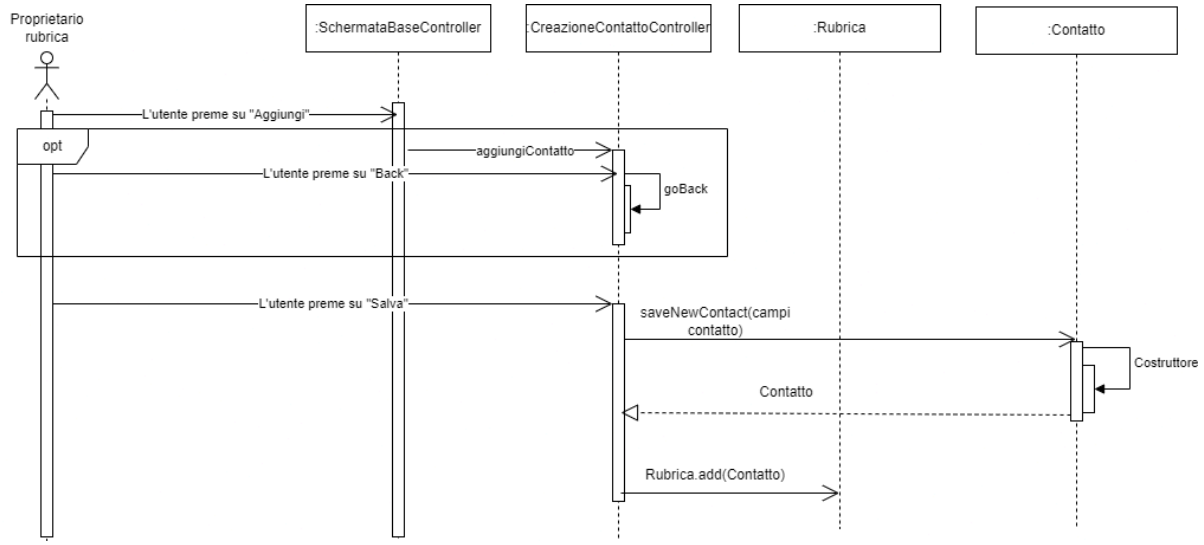
- **Relazione**: Associazione
- **Cardinalità**: 1 ↔ 1
- Questo controller modifica o visualizza un singolo oggetto `Contatto`. La relazione è diretta perché il controller ha bisogno di accedere ai dettagli del contatto.

ApplicazioneRubrica ↔ Rubrica

- **Relazione**: Composizione
- **Cardinalità**: 1 ↔ 1
- L'applicazione crea e gestisce l'istanza principale della rubrica, che vive finché l'applicazione è attiva.

Descrizione Diagramma di Sequenza

Caso d’uso: Creazione di un nuovo contatto

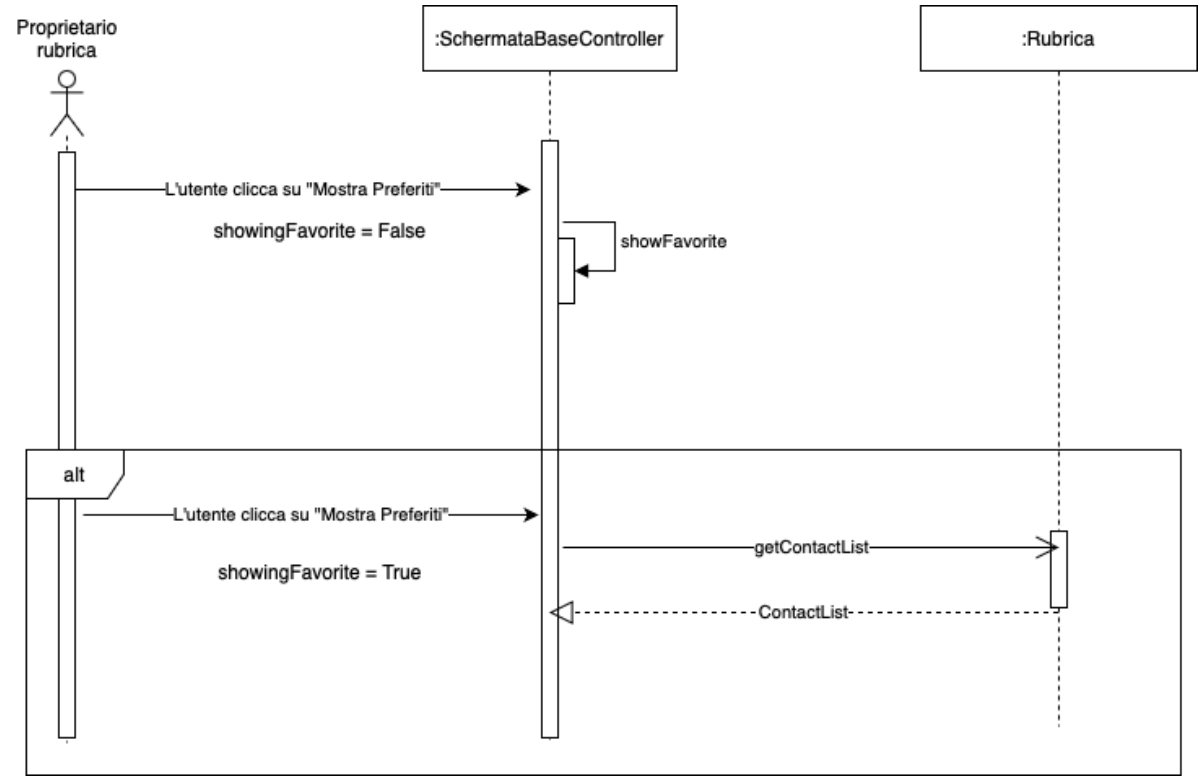


Questo diagramma di sequenza rappresenta il caso d'uso in cui un utente avvia il processo di creazione di un contatto, lo annulla inizialmente (percorso alternativo) e poi completa l'operazione in un secondo momento.
Il flusso è descritto come segue:

- L'utente avvia la procedura di creazione premendo il pulsante *"Aggiungi"* nella schermata principale (**SchermataBaseController**). Ciò richiama l'interazione tra **SchermataBaseController** e **CreazioneContattoController**.
- Durante la fase di input, l'utente sceglie di annullare l'operazione selezionando *"Back"*. L'interazione si conclude senza che un contatto venga creato.
- In un successivo tentativo, l'utente completa il processo di creazione premendo *"Salva"*. Questo porta all'istanziazione di un nuovo oggetto **Contatto**, che viene aggiunto alla lista dei contatti attraverso il metodo *Rubrica.add()*.

SchermataBaseController si occupa esclusivamente della gestione delle interazioni nella schermata principale e delega la creazione del contatto al **CreazioneContattoController**, rispettando il **Principio di Singola Responsabilità**. Questo assicura che ogni controller abbia un ruolo ben definito ed indipendente.

Caso d’uso: Visualizzazione contatti preferiti



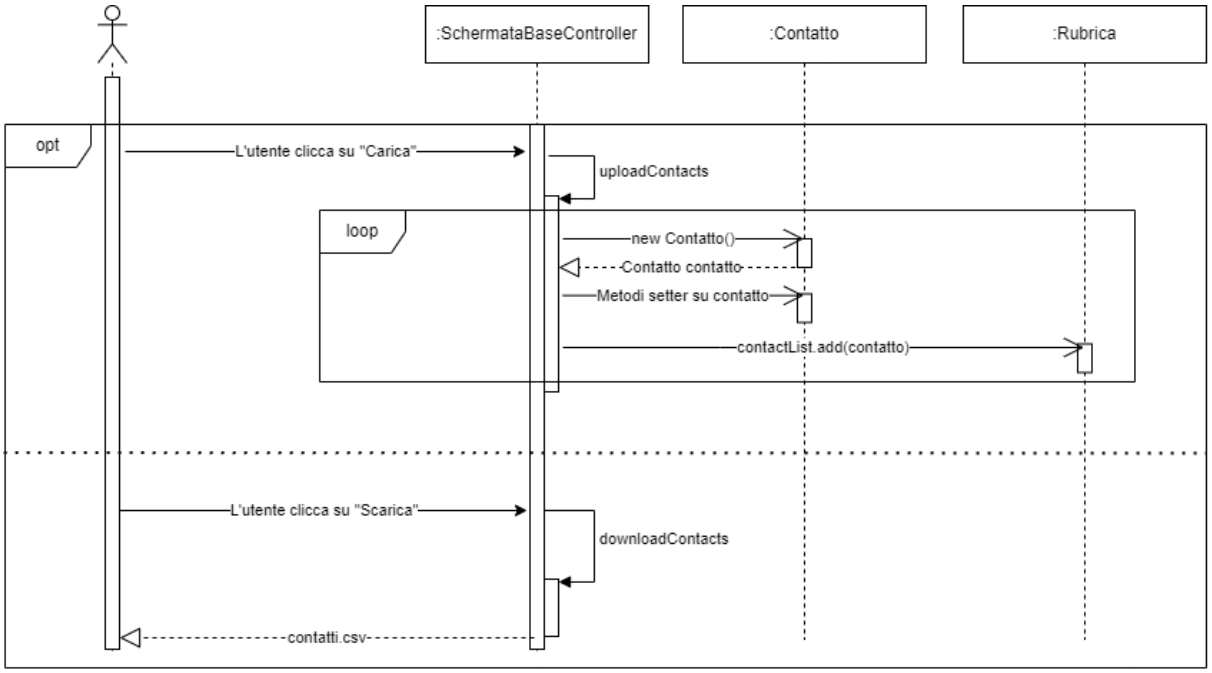
Questo diagramma di sequenza rappresenta il caso d'uso in cui un utente chiede attraverso il button *“Mostra preferiti”* di visualizzare in rubrica solo i contatti con l’attributo *isFavorite* = True.
Il flusso è descritto come segue:

- L’utente clicca su *“Mostra Preferiti”*
- *showingFavorite* passa da False (impostazione di default) a True.
- L’oggetto **SchermataBaseController** esegue il metodo *showFavorite* mostrando nell’elenco della rubrica solo i contatti aventi l’attributo *isFavorite* = True.

Il secondo caso, indicato come percorso alternativo, mostra l’utente che preme nuovamente il button *“Mostra preferiti”*, in questo caso il flag *showingFavorite* è già True, ovvero l’interfaccia sta già mostrando i contatti preferiti.
Il flusso è descritto come segue:

- L’utente clicca su *“Mostra Preferiti”*
- *showingFavorite* passa da True a False
- L’oggetto **SchermataBaseController** chiama il metodo *Rubrica.getContactList* per riottenere la lista di contatti completa della rubrica e mostrarla a schermo.

Caso d’uso: Upload e Download della Rubrica



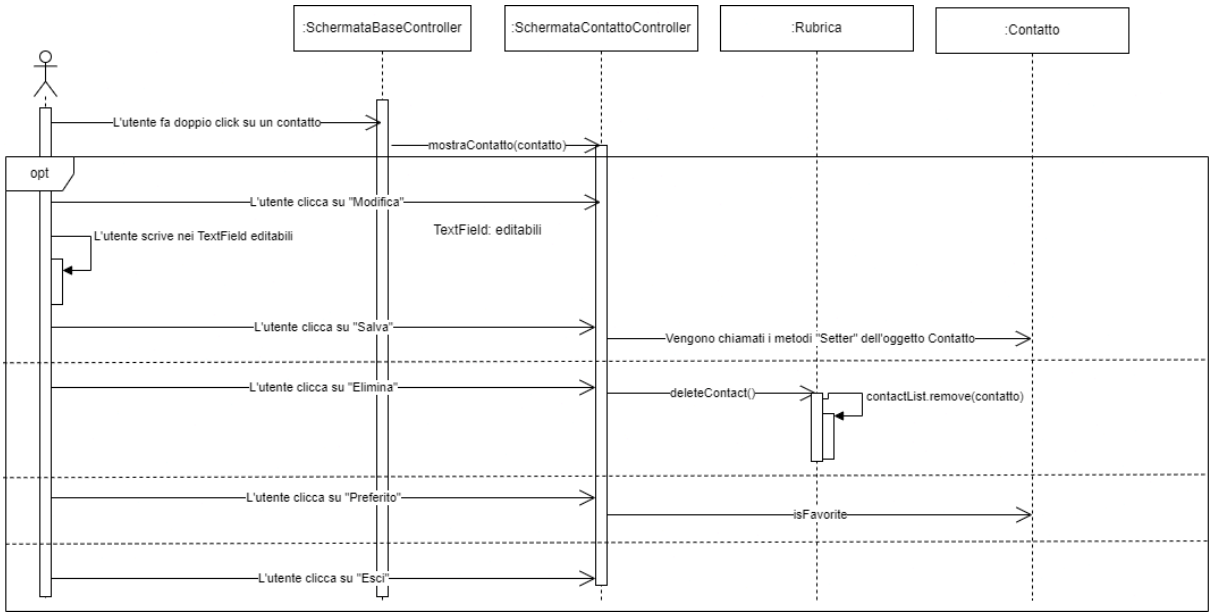
Questo diagramma di sequenza rappresenta il caso d'uso in cui l’utente sceglie di caricare e successivamente scaricare la lista dei contatti.
Il flusso è descritto come segue:
Primo percorso opzionale:

- L’utente clicca su *“Carica”*
- Tramite il metodo *uploadContacts* viene caricato un file .csv con al suo interno i contatti da inserire nella rubrica.
- Per ogni riga del file .csv viene istanziato un oggetto di classe **Contatto** all’interno del quale vengono automaticamente inizializzati gli attributi.
- L’oggetto contatto viene aggiunto alla **Rubrica**.

Secondo percorso opzionale:

- L’utente clicca su *“Scarica”*
- L’oggetto **SchermataBaseController** chiama il metodo *downloadContacts*
- L’utente riceve in ritorno un file .csv con l’elenco dei contatti presenti in quel momento all’interno della rubrica.

Caso d’uso: Modifica Contatto



Questo diagramma di sequenza rappresenta il caso d'uso in cui l’utente sceglie di modificare un contatto presente in rubrica.
Il flusso è descritto come segue:

- L’utente fa doppio click su un contatto;
- Attraverso il metodo *mostraContatto(contatto)* viene mostrata la finestra con al suo interno tutti le informazioni del contatto in questione

Primo percorso opzionale:

- L’utente clicca su *“Modifica”*;
- I TextField della finestra del contatto diventano editabili
- L’utente modifica i TextField tramite input di tastiera
- L’utente clicca su *“Salva”*;
- Attraverso i metodi setter vengono sovrascritti i nuovi attributi dell’oggetto contatto.

Secondo percorso opzionale:

- L’utente clicca su *“Elimina”*;
- Viene chiamato il metodo *deleteContact()*;
- Viene chiamato dalla **Rubrica** il metodo *contactList.remove(contatto)* per eliminare il contatto dalla **Rubrica**.

Terzo percorso opzionale:

- L’utente clicca su *“Preferito”*;
- Viene chiamato il metodo *isFavorite* che esegue una not dell’attributo *Favorite* dell’oggetto contatto in questione.

Quarto percorso opzionale:

- L’utente clicca su *“Esci”*;
- La finestra del contatto si chiude.

Storico delle versioni

Versione	Data	Descrizione	Autori
1	27/11/2024	Creazione del documento; Primi schizzi del diagramma delle classi;	De Rosa, Zito
2	29/11/2024	Realizzazione diagramma delle classi; Abbozzo del mockup;	De Rosa, Loffredo, Zito
3	02/12/2024	Aggiornamento diagramma delle classi; Creazione Doxyfile;	De Rosa, Loffredo, Zito
4	03/12/2024	Realizzazione del sequence diagram;	De Rosa, Loffredo, Zito
5	04/12/2024	Revisione delle classi nel codice; Correzione diagramma di Sequenza;	De Rosa, Loffredo, Zito
6	05/12/2024	Scrittura della descrizione del diagramma delle classi e del diagramma delle sequenze;	Loffredo
7	06/12/2024	Scrittura della descrizione dei diagrammi delle sequenze;	De Rosa, Loffredo, Zito
8	07/12/2024	Scrittura del diagramma delle sequenze;	De Rosa, Loffredo, Zito
9	13/12/2024	Rettificazione diagramma delle classi con annessa descrizione in base alle modifiche apportate nell’implementazione;	De Rosa