

CineMates20

Trinchillo Giovanni N86003133

Libutti Simone N86002928

Gennaio 2021



# Contents

<b>1 Introduzione</b>	<b>4</b>
1.1 Funzionalità commissionate . . . . .	4
1.2 Proposta implementativa . . . . .	4
<b>2 Documento dei Requisiti Software</b>	<b>7</b>
2.1 Modello Funzionale . . . . .	7
2.1.1 Modellazione dei casi d'uso. . . . .	8
2.1.2 Tabelle di Cockburn per ogni caso d'uso . . . . .	15
2.1.3 Mock-up dell'interfaccia utente . . . . .	34
2.1.4 Glossario . . . . .	62
2.2 Modelli di Dominio . . . . .	62
2.2.1 Classi diagram di analisi . . . . .	62
2.2.2 Diagrammi di sequenza di analisi . . . . .	77
2.2.3 Activity Diagrams . . . . .	93
<b>3 Documento di Design del sistema</b>	<b>112</b>
3.1 Analisi dell'architettura del sistema . . . . .	112
3.1.1 Tecnologie Adoperate . . . . .	112
3.1.2 Criteri di design . . . . .	114
3.2 Diagramma delle classi di design . . . . .	119

3.3	CRC Card per le classi non di interfaccia utente . . . . .	121
3.4	Diagrammi di sequenza di design per gli scenari non banali dei casi d'uso assegnati . . . . .	133
<b>4</b>	<b>Testing</b>	<b>140</b>
4.1	Codice xUnit per unit testing di 4 funzionalità non banali . . . . .	140
4.1.1	Applicazione Mobile - Log In . . . . .	141
4.1.2	Registrazione . . . . .	145
4.1.3	Aggiornamento delle statistiche sul client . . . . .	151
4.1.4	Verifica delle credenziali admin da file . . . . .	152

# 1 Introduzione

In seguito alla richiesta di sviluppo del sistema CineMates20 della società SoftEngUniNA, si presenta di seguito il documento di progettazione relativo alle varie fasi della produzione del software.

## 1.1 Funzionalità commissionate

Oggetto della richiesta di implementazione sono le seguenti funzionalità. Alcune di queste sono state aggiunte in un secondo momento rispetto alle specifiche iniziali in modo tale da rendere l'applicazione il più integra possibile. La lista che segue elenca in maniera generale le caratteristiche che il prodotto finale deve presentare, ma è separata dalla formalizzazione dei casi d'uso della prossima sezione.

1. Registrazione e autenticazione degli utenti, con possibilità di autenticazione tramite un servizio esterno.
2. Possibilità di un utente di modificare le proprie informazioni personali.
3. Possibilità di un utente di terminare la propria sessione.
4. Effettuare ricerche di film, utilizzando API offerte da servizi esterni.
5. Aggiungere o rimuovere film dalla propria lista di preferiti, o di film salvati.
6. Inviare e ricevere richieste di collegamento a e dagli altri utenti, che possono essere accettate e rifiutate.
7. Visualizzare, in un feed, le azioni recenti effettuate dai propri collegamenti, in ordine cronologico.
8. Autenticazione degli amministratori mediante un'infrastruttura gestita da un Superadmin, che può generare nuove credenziali di accesso alla piattaforma admin.
9. Consentire agli eventuali amministratori di visualizzare statistiche sul sistema tramite la piattaforma admin.

## 1.2 Proposta implementativa

In questa sottosezione si da una descrizione generale di come il prodotto finale si presenterà, facendo riferimento alla precedente lista di funzionalità ed esplicitando il modo in cui ciascuna di queste ultime è stata implementata,

precedendo l'analisi formale dei requisiti e le descrizioni funzionali.

La richiesta di progettazione prevede due moduli software distinti collegati da un back-end unico: un'applicazione mobile e una piattaforma desktop. Sulla base di questo vincolo, si è deciso che il modo più naturale di procedere sarebbe stato quello di dedicare l'applicazione mobile alle funzionalità fruibili dagli utenti, mentre il client desktop rappresenterà la piattaforma di amministrazione per le operazioni di visualizzazione delle statistiche del sistema.

## 1. Applicazione mobile

L'applicazione CineMates20 è stata prodotta tenendo a mente i principi di base dell'usabilità e dell'immediatezza, facendo sì che ciascuna funzionalità risulti essere il più intuitiva possibile per gli utenti.

All'avvio dell'applicazione, gli utenti vengono accolti da una schermata di accesso, che presenta la possibilità di effettuare il log in tramite la propria combinazione email-password. In accordo con la **funzionalità 1**, questa schermata presenta anche l'opzione di effettuare il log in al sistema tramite Facebook. Qualora l'utente non dovesse disporre di un account, un apposito pulsante di registrazione lo condurrà ad una schermata dove sarà possibile crearne uno. Se l'utente ha già effettuato l'accesso, chiudere e riaprire l'applicazione non fa perdere la propria sessione attiva, e l'utente si ritroverà già autenticato al momento della riapertura.

All'accesso, l'applicazione presenta una barra di navigazione bassa, utilizzabile per accedere alle varie sezioni. La schermata iniziale è quella definita Home, che mette a disposizione la **funzionalità 7**. Come azioni visualizzabili sul feed da parte di un utente vi sono l'aggiunta e la rimozione di film nelle liste dei suoi collegamenti, nonché i collegamenti effettuati a loro volta da questi ultimi.

La seconda voce della barra di navigazione porta alla sezione Social. Le richieste di collegamento tra utenti da sviluppare come da **funzionalità 6** sono state implementate secondo una politica follow/follower. Se un utente segue un altro utente, vedrà le sue azioni recenti all'interno del suo feed, ma l'utente seguito non avrà relazioni con esso a meno che anch'egli non decida di seguirlo. Questo approccio è stato scelto in modo da evitare che accettare una richiesta di follow implichì l'essere forzati a ricevere notizie indesiderate.

Nella sezione Social gli utenti possono visualizzare la propria lista di seguiti e cercare un utente per inviargli una richiesta. Una richiesta inviata si può ritirare prima che il destinatario l'accetti. Si può inoltre consultare la lista di richieste in arrivo, e per ciascuna decidere se accettarla o respingerla.

La terza sezione consiste nella schermata di ricerca di film richiesta dalla **funzionalità 4**, ovvero il fine ultimo

dell'applicazione. Gli utenti possono cercare film per titolo, e i risultati saranno ordinati per popolarità. A una ricerca corrisponderà una lista di risultati, e ciascun risultato, se cliccato, condurrà ad una schermata più dettagliata del film in questione. In quest'ultima schermata è possibile aggiungere o rimuovere un film dalla propria lista di film preferiti o di film salvati, come da [funzionalità 5](#).

La quarta voce porta alle proprie liste di film. Premere su di un film presente in una delle due liste conduce alla stessa schermata a cui si giunge cliccando un film dai risultati della ricerca.

La quinta e ultima sezione dell'app è il proprio profilo, che l'utente può modificare in accordo con la [funzionalità 2](#). Questa schermata implementa anche la [funzionalità 3](#), consentendo all'utente di effettuare il log out, terminando la propria sessione.

## 2. Client desktop

La piattaforma di amministrazione tiene conto dell'implementazione delle [funzionalità 8 e 9](#). Siccome questo lato del software sarà utilizzato esclusivamente da una serie di amministratori interni all'azienda, le componenti estetiche sono state messe in secondo piano rispetto alla praticità dell'utilizzo.

Si suppone che il primo avvio del client in assoluto sia compiuto da una figura di Superadmin, che avrà controllo sugli admin futuri: sarà l'unico ad avere la possibilità di generare nuove credenziali. Egli inserirà delle credenziali iniziali prefissate che saranno consegnate ai committenti insieme al client, e il sistema riconoscerà il primo accesso e chiederà al Superadmin di modificare le sue credenziali personali. Una volta modificate, egli avrà accesso alle statistiche del sistema. Le statistiche che verranno visualizzate saranno:

- Numero di utenti
- Numero di accessi attivi al momento
- Numero di collegamenti tra gli utenti
- Numero di ricerche di film effettuate
- Numero di film d'avventura presenti nelle liste degli utenti
- Numero di film d'azione presenti nelle liste degli utenti
- Numero di commedie presenti nelle liste degli utenti
- Numero di thriller presenti nelle liste degli utenti

- Numero di horror presenti nelle liste degli utenti.

Sulla finestra, oltre alle statistiche, saranno presenti due buttoni: uno per aggiornare le statistiche, in modo tale da poterle disporre sempre in tempo reale, e uno per permettere la creazione di nuove credenziali admin. Le credenziali saranno una combinazione nome-password dove sia il nome che la password consisteranno in una stringa alfanumerica casuale. Con la creazione di nuove credenziali, si darà accesso alle statistiche ad un ulteriore profilo di amministrazione. E' chiaro che il bottone per la creazione di nuove credenziali sarà disponibile se e solo se si è effettuato il login come Superadmin.

## 2 Documento dei Requisiti Software

### 2.1 Modello Funzionale

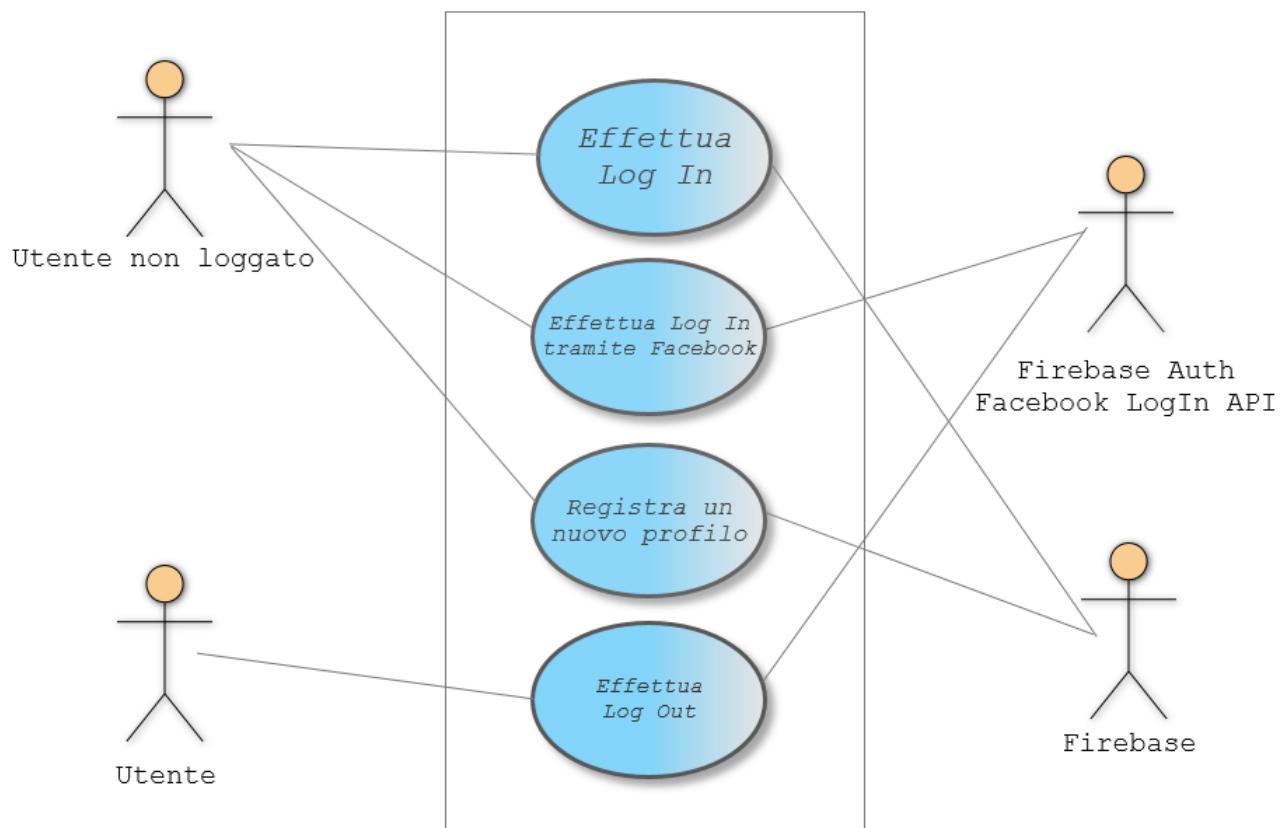
Come concordato con i committenti durante il colloquio di specifica dei requisiti, si esplicitano i casi d'uso oggetto di implementazione per CineMates20:

1. Sistema di autenticazione degli utenti
  - 1.1. Log In utente
  - 1.2. Log In utente tramite Facebook
  - 1.3. Registrazione di un nuovo profilo
  - 1.4. Log out
2. Sistema di autenticazione degli amministratori
  - 2.1. Log In amministratore
  - 2.2. Generazione di nuove credenziali da parte di un Superadmin
3. Ricerca di film tramite i servizi offerti da una API esterna

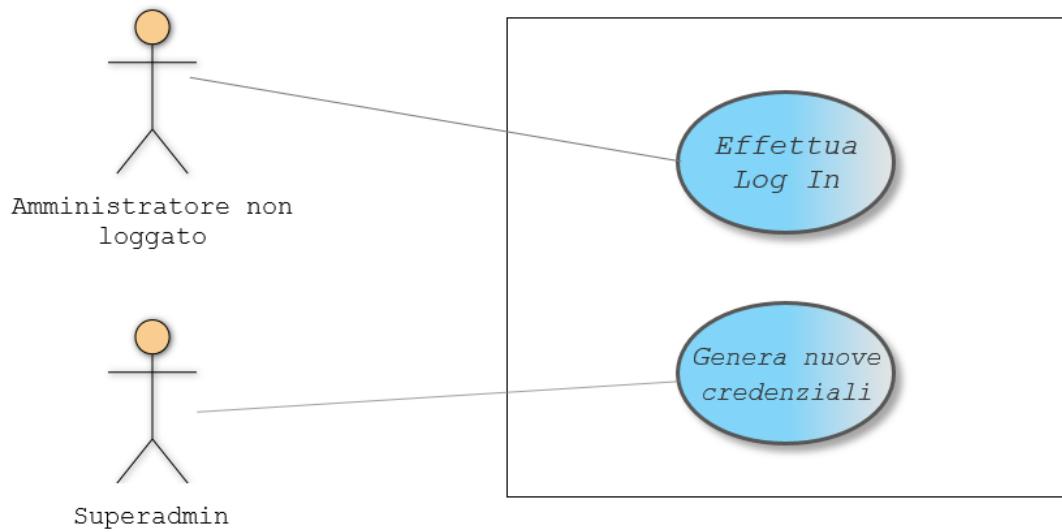
4. Presenza di liste di film preferiti e di film salvati
  - 4.1. Aggiunta di un film ai preferiti
  - 4.2. Rimozione di un film dai preferiti
  - 4.3. Aggiunta di un film ai salvati
  - 4.4. Rimozione di un film dai salvati
5. Sistema di collegamenti tra utenti
  - 5.1. Invio richiesta di follow
  - 5.2. Annullamento richiesta di follow
  - 5.3. Accettazione/Rifiuto di una richiesta di follow
6. Visualizzazione di un feed con le azioni recenti degli utenti
7. Gestione del proprio profilo
  - 7.1. Invio richiesta di follow
  - 7.2. Annullamento richiesta di follow
  - 7.3. Accettazione/Rifiuto di una richiesta di follow
8. Visualizzazione, da parte degli admin, delle statistiche del sistema

### **2.1.1 Modellazione dei casi d'uso.**

Ciascuno dei casi d'uso esplicitati ad inizio sezione è stato rappresentato graficamente con notazione UML tramite Use Case Diagram, presentati di seguito.



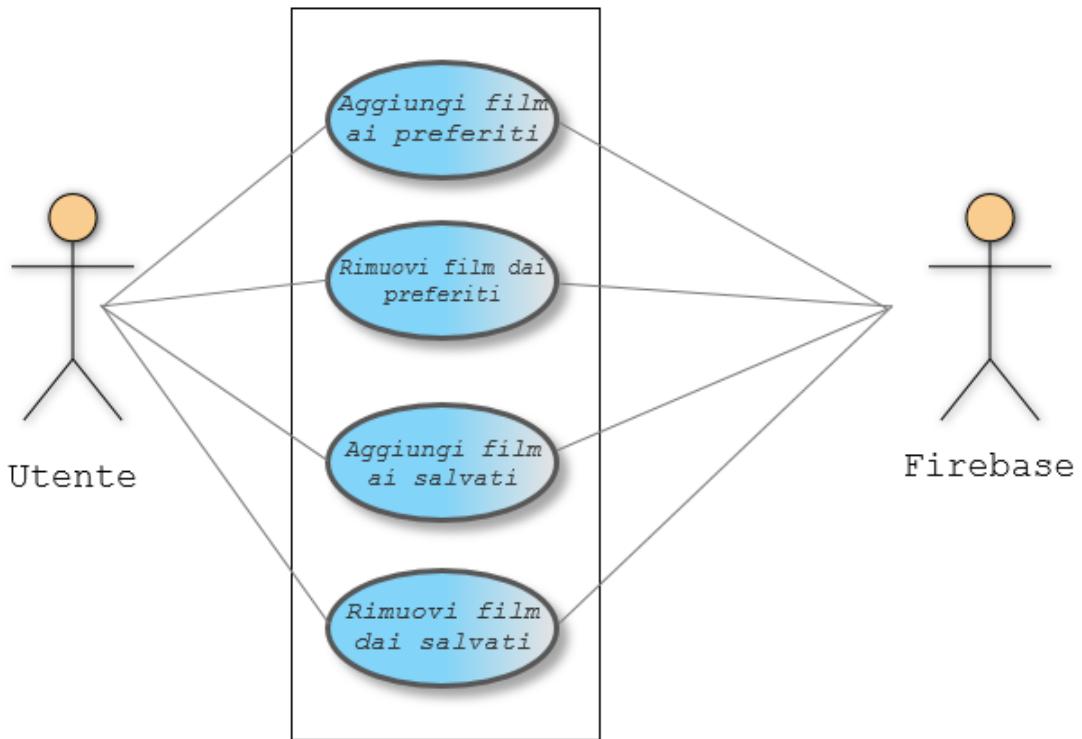
Use Case 1



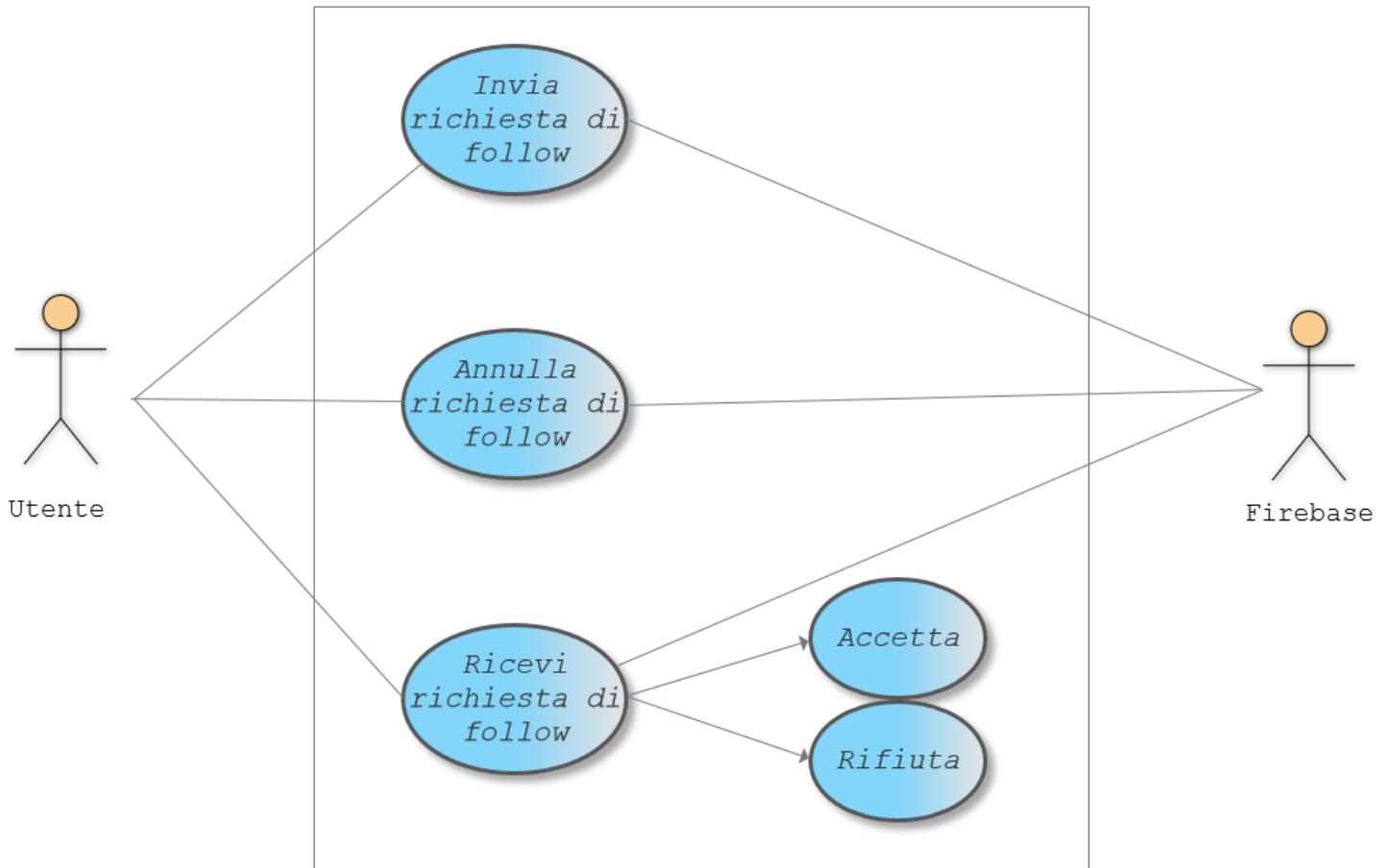
Use Case 2



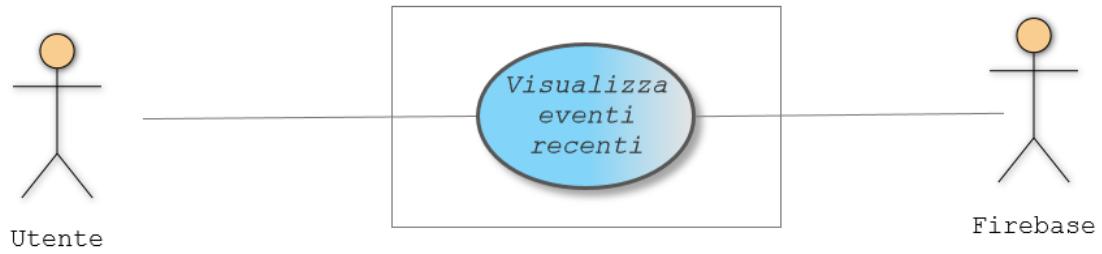
Use Case 3



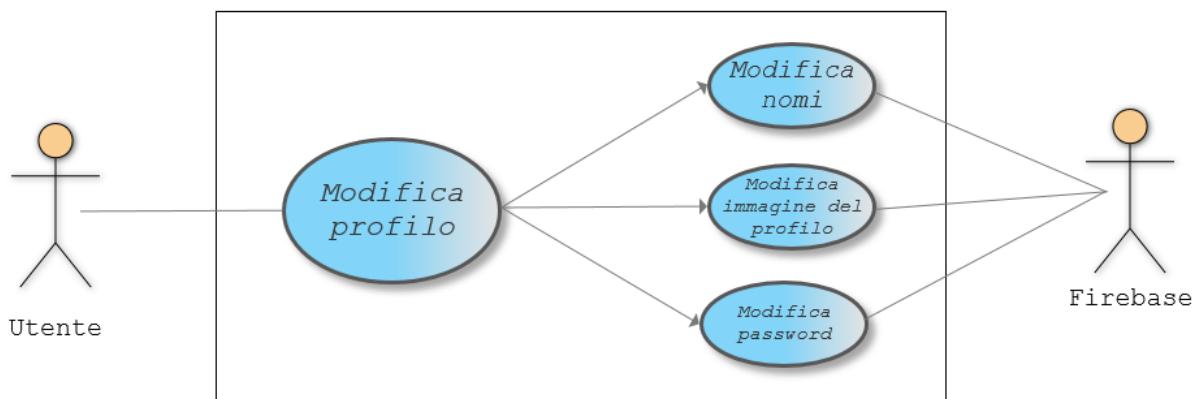
Use Case 4



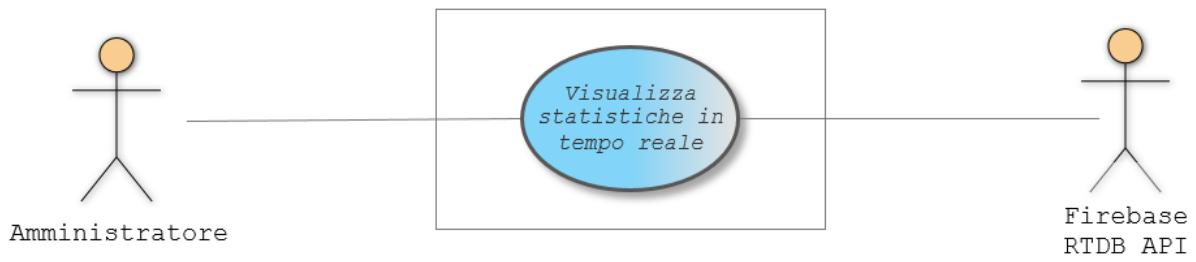
Use Case 5



Use Case 6



Use Case 7



Use Case 8

### **2.1.2 Tabelle di Cockburn per ogni caso d'uso**

Per ciascun sotto-caso d'uso è stata prodotta una tabella secondo il formalismo di Cockburn.

La politica adottata per lo sviluppo della tabella prevede che, per ciascuna di queste, le "Precondizioni" consistano in una situazione che deve essere in atto al momento in cui l'utente (o l'amministratore) si accinge a mettere in atto la funzionalità, mentre il "Trigger" consiste nella prima azione che l'utente (o l'amministratore) compie per avviare la stessa. Ne consegue che il primo *step* di ciascuna tabella rappresenterà l'evento immediatamente successivo al trigger, ovviamente qualora esso dovesse essere presente.

Per indicare situazioni estranee al normale svolgimento della funzionalità, si è usata la clausola di estensione. L'assenza di quest'ultima indica che non esistono motivi noti per i quali la funzionalità non può giungere correttamente a conclusione.

Si noti inoltre che per quanto riguarda le funzionalità di amministrazione, non sono stati inclusi i servizi web Amazon (si veda sezione 3.1.1) nel ruolo di attore esterno, sebbene essi forniscano la macchina virtuale su cui alloggia il front-end desktop. Ciò è dovuto al fatto che nessuna delle funzioni necessita di questi per essere eseguita, in quanto essi sono esclusivamente un supporto temporaneo per agevolare l'accesso al client.

<b>USE CASE 1</b>	<b>Log In utente</b>			
Obiettivo	<i>Un utente vuole effettuare l'accesso all'applicazione</i>			
Precondizioni	-			
Condizione di successo	<i>L'utente ha effettuato correttamente l'accesso all'applicazione</i>			
Condizione di fallimento	<i>L'utente non ha effettuato correttamente l'accesso all'applicazione per uno o più motivi</i>			
Attore primario	<i>Utente non loggato</i>			
Trigger	<i>Avvio dell'applicazione su piattaforma mobile Android</i>			
DESCRIZIONE	Step	Attore Utente non loggato	Sistema	Attore esterno Firebase
	1		Mostra MU1	
	2	Inserimento di Email e Password in MU1		
	3	Pressione di LOG IN su MU1		
	4		Verifica la correttezza sintattica delle credenziali	
	5			Verifica la validità delle credenziali
	6		Mostra MU3	
<u>ESTENSIONE 1:</u> Una o più credenziali non sono sintatticamente corrette	5 (1)		Verifica la presenza di un errore nelle credenziali	
	6 (1)		Notifica l'utente specificando il tipo di errore	
<u>ESTENSIONE 2:</u> Lo username è inesistente, la password è errata o l'utente non ha una password	5 (2)			Verifica l'entità dell'errore
	6 (2)		Notifica l'utente dell'errore	

Tabella 1.1

<b>USE CASE 1</b>	<b>Log In utente tramite Facebook</b>			
Obiettivo	<i>Un utente vuole effettuare l'accesso all'applicazione tramite Facebook</i>			
Precondizioni	-			
Condizione di successo	<i>L'utente ha effettuato correttamente l'accesso all'applicazione</i>			
Condizione di fallimento	<i>L'utente non ha effettuato correttamente l'accesso all'applicazione per uno o più motivi</i>			
Attore primario	<i>Utente non loggato</i>			
Trigger	<i>Avvio dell'applicazione su piattaforma mobile Android</i>			
DESCRIZIONE	<b>Step</b>	<b>Attore Utente non loggato</b>	<b>Sistema</b>	<b>Attore esterno Firebase Auth Facebook Log In API</b>
	<b>1</b>		<b>Mostra MU1</b>	
	<b>2</b>	Pressione di <i>Continua con Facebook</i> su <b>MU1</b>		
	<b>3</b>			Avvia procedura di Log In tramite Facebook
	<b>4</b>			Termina procedura di Log In tramite Facebook
	<b>5</b>		<b>Mostra MU3</b>	
<b>ESTENSIONE 1:</b> La procedura di Log In tramite Facebook non va a buon fine	<b>5 (1)</b>		<b>Mostra MU1</b>	

Tabella 1.2

<b>USE CASE 1</b>	<b>Registrazione nuovo profilo</b>		
Obiettivo	<i>Un utente vuole registrarsi all'applicazione</i>		
Precondizioni	-		
Condizione di successo	<i>L'utente si è registrato correttamente all'applicazione</i>		
Condizione di fallimento	<i>L'utente non si è registrato correttamente all'applicazione per uno o più motivi</i>		
Attore primario	<i>Utente non loggato</i>		
Trigger	<i>Pressione di REGISTRATI su MU1</i>		
DESCRIZIONE	Step	Attore Utente non loggato	Sistema
	1		Mostra MU2
	2	Inserimento di nome, cognome, Email e password su MU2	
	3	Pressione di REGISTRATI su MU2	
	4		Verifica la correttezza sintattica delle credenziali
	5		Verifica la validità delle credenziali
	6		Crea un nuovo utente
	7		Mostra MU3
<u>ESTENSIONE 1:</u> Una o più credenziali non sono sintatticamente corrette	5 (1)		Verifica la presenza di un errore nelle credenziali
	6 (1)		Notifica l'utente specificando il tipo di errore
<u>ESTENSIONE 2:</u> Lo username è già esistente o la Email non è valida	5 (2)		Verifica l'entità dell'errore
	6 (2)		Notifica l'utente dell'errore

Tabella 1.3

<b>USE CASE 1</b>	<b>Log out</b>			
Obiettivo	<i>Un utente vuole effettuare il Log out dall'applicazione</i>			
Precondizioni	-			
Condizione di successo	<i>L'utente ha effettuato correttamente il logout dall'applicazione</i>			
Condizione di fallimento	<i>L'utente non ha effettuato correttamente il logout dall'applicazione per uno o più motivi</i>			
Attore primario	<i>Utente</i>			
Trigger	<i>Pressione di Profilo sulla barra di navigazione di MU3</i>			
DESCRIZIONE	<b>Step</b>	<b>Attore Utente</b>	<b>Sistema</b>	<b>Attore esterno <i>Firebase</i></b>
	1		Mostra <b>MU11</b>	
	2	Pressione su <i>Menu</i> in alto a destra su <b>MU11</b>		
	3	Pressione su <i>Log out</i>		
	4			Disautentica l'utente
	5		Mostra <b>MU1</b>	
<u>ESTENSIONE 1:</u> La procedura di Log Out non va a buon fine	4 (1)		Mostra <b>MU11</b>	

Tabella 1.4

USE CASE 2	Log In amministratore		
Obiettivo	<i>Un amministratore vuole effettuare l'accesso al client desktop</i>		
Precondizioni	-		
Condizione di successo	<i>L'amministratore ha effettuato correttamente l'accesso al client</i>		
Condizione di fallimento	<i>L'amministratore non ha effettuato correttamente l'accesso al client per uno o più motivi</i>		
Attore primario	<i>Amministratore non loggato</i>		
Trigger	<i>Avvio del client desktop</i>		
DESCRIZIONE	Step	Attore Amministratore non loggato	Sistema
	1		Mostra MU12
	2	Inserimento delle credenziali admin in <b>MU12</b>	
	3	Pressione di <i>Accedi</i> su <b>MU12</b>	
	4		Verifica la validità delle credenziali
	5		Mostra MU14.1
<u>ESTENSIONE 1:</u> Il Log In corrente è il primo Log In	5 (1)		Mostra MU13
	6 (1)	Inserisce le nuove credenziali in <b>MU13</b>	
	7 (1)	Pressione di <i>Conferma</i> su <b>MU13</b>	
	8 (1)		Aggiorna le credenziali del Superadmin
	9 (1)		Mostra MU14
<u>ESTENSIONE 2:</u> L'amministratore che sta effettuando il Log In è il Superadmin	5 (2)		Mostra MU14
<u>ESTENSIONE 3:</u> Le credenziali non sono valide	4 (3)		Mostra un messaggio di errore

Tabella 2.1

<b>USE CASE 2</b>	<b>Genera nuove credenziali</b>		
Obiettivo	<i>Il Superadmin vuole generare delle nuove credenziali</i>		
Precondizioni	<i>Il Superadmin ha effettuato correttamente l'accesso al client</i>		
Condizione di successo	<i>Il Superadmin ha generato delle nuove credenziali per il client</i>		
Condizione di fallimento	<i>Il Superadmin non è riuscito a creare delle credenziali di accesso al client per uno o più motivi</i>		
Attore primario	<i>Superadmin</i>		
Trigger	-		
DESCRIZIONE	Step	Attore Superadmin	Sistema
	1	Pressione di <i>Genera nuove credenziali</i> su <b>MU14</b>	
	2		Genera nuove credenziali e mostra messaggio di conferma
<b>ESTENSIONE 1:</b> La creazione di nuove credenziali non è andata a buon fine	2 (1)		Mostra messaggio di errore

Tabella 2.2

<b>USE CASE 3</b>	<b>Effettua ricerca film</b>			
Obiettivo	<i>Un utente vuole effettuare una ricerca di film</i>			
Precondizioni	<i>L'utente ha effettuato correttamente l'accesso all'applicazione</i>			
Condizione di successo	<i>L'utente ha ricevuto dei risultati per la sua ricerca di Film</i>			
Condizione di fallimento	<i>L'utente non ha ricevuto risultati per la sua ricerca di Film</i>			
Attore primario	<i>Utente</i>			
Trigger	<i>Pressione del pulsante Ricerca nella barra di navigazione di MU3</i>			
DESCRIZIONE	Step	Attore Utente	Sistema	Attore TMDB API
	1	Inserimento di una voce di ricerca nella barra di ricerca di <b>MU7.1</b>		
	2		Invia richiesta (HTTP) all'API di TMDB	
	3			Produzione risposta (HTTP)
	4		Ricezione ed elaborazione dei dati	
	5		Mostra risultati della ricerca come indicato in <b>MU7</b>	
<b>ESTENSIONE 1:</b> La ricerca non ha prodotto risultati	5 (1)		Mostra <b>MU7.2</b>	
<b>ESTENSIONE 2:</b> La ricerca è fallita per assenza di connessione	3 (2)		Mostra messaggio di errore	

Tabella 3.1

<b>USE CASE 4</b>	<b>Aggiungi film ai preferiti</b>			
Obiettivo	<i>Un utente vuole aggiungere un film alla sua lista dei preferiti</i>			
Precondizioni	<i>L'utente ha effettuato una ricerca di film che ha prodotto risultati</i>			
Condizione di successo	<i>L'utente ha aggiunto il film ai preferiti</i>			
Condizione di fallimento	<i>L'utente non è riuscito ad aggiungere il film ai preferiti</i>			
Attore primario	<i>Utente</i>			
Trigger	<i>Pressione su di una voce del risultato della ricerca in MU7</i>			
DESCRIZIONE	<b>Step</b>	<b>Attore Utente</b>	<b>Sistema</b>	<b>Attore esterno <i>Firebase</i></b>
	1		Mostra <b>MU8</b>	
	2	Pressione sul pulsante <i>Cuore</i> di <b>MU8</b>		
	3			Aggiunta del film ai preferiti dell'utente
	4		Il pulsante <i>Cuore</i> di <b>MU8</b> si riempie di rosso	
<b>ESTENSIONE 1:</b> Il film è già tra i preferiti dell'utente	3 (1)			Vedasi <b>USE CASE 4 – Rimuovi film dai preferiti</b>

Tabella 4.1

<b>USE CASE 4</b>	<b>Rimuovi film dai preferiti</b>			
Obiettivo	<i>Un utente vuole rimuovere un film dalla sua lista dei preferiti</i>			
Precondizioni	<i>L'utente ha il film interessato nei suoi preferiti</i>			
Condizione di successo	<i>L'utente ha aggiunto il film ai preferiti</i>			
Condizione di fallimento	<i>L'utente non è riuscito ad aggiungere il film ai preferiti</i>			
Attore primario	<i>Utente</i>			
Trigger	<i>Pressione del tasto Liste della barra di navigazione di MU3</i>			
DESCRIZIONE	Step	Attore Utente	Sistema	Attore esterno <i>Firebase</i>
	1		Mostra MU9	
	2	Pressione sul film interessato		
	3		Mostra MU8	
	4	Pressione del pulsante Cuore di MU8		
	5			Rimozione del film dai preferiti dell'utente
	6		Il pulsante Cuore di MU8 si svuota	

Tabella 4.2

<b>USE CASE 4</b>	<b>Aggiungi film ai salvati</b>			
Obiettivo	<i>Un utente vuole aggiungere un film alla sua lista dei salvati</i>			
Precondizioni	<i>L'utente ha effettuato una ricerca di film che ha prodotto risultati</i>			
Condizione di successo	<i>L'utente ha aggiunto il film ai salvati</i>			
Condizione di fallimento	<i>L'utente non è riuscito ad aggiungere il film ai salvati</i>			
Attore primario	<i>Utente</i>			
Trigger	<i>Pressione su di una voce del risultato della ricerca in MU7</i>			
DESCRIZIONE	<b>Step</b>	<b>Attore Utente</b>	<b>Sistema</b>	<b>Attore esterno <i>Firebase</i></b>
	1		Mostra <b>MU8</b>	
	2	Pressione sul pulsante <i>Segnalibro</i> di <b>MU8</b>		
	3			Aggiunta del film ai salvati dell'utente
	4		Il pulsante <i>Segnalibro</i> di <b>MU8</b> si riempie di nero	
<b>ESTENSIONE 1:</b> Il film è già tra i preferiti dell'utente	3 (1)			Vedasi <b>USE CASE 4 – Rimuovi film dai salvati</b>

Tabella 4.3

<b>USE CASE 4</b>	<b>Rimuovi film dai salvati</b>			
Obiettivo	<i>Un utente vuole rimuovere un film dalla sua lista dei salvati</i>			
Precondizioni	<i>L'utente ha il film interessato nei suoi salvati</i>			
Condizione di successo	<i>L'utente ha aggiunto il film ai salvati</i>			
Condizione di fallimento	<i>L'utente non è riuscito ad aggiungere il film ai salvati</i>			
Attore primario	<i>Utente</i>			
Trigger	<i>Pressione del tasto Liste della barra di navigazione di MU3</i>			
DESCRIZIONE	Step	Attore Utente	Sistema	Attore esterno <i>Firebase</i>
	1		Mostra MU9	
	2	Pressione della tab <i>Salvati</i> di MU9		
	3		Mostra MU10	
	4	Pressione sul film interessato		
	5		Mostra MU8	
	6	Pressione sul pulsante <i>Segnalibro</i> di MU8		
	7			Rimozione del film dai salvati dell'utente
	8		Il pulsante <i>Segnalibro</i> di <b>MU8</b> si svuota	

Tabella 4.4

<b>USE CASE 5</b>	<b>Invia richiesta di follow</b>			
Obiettivo	<i>Un utente vuole inviare una richiesta di follow ad un altro utente</i>			
Precondizioni	<i>L'utente non è l'utente target, non ha già inviato una richiesta all'utente target e ha effettuato una ricerca di utenti che ha prodotto dei risultati</i>			
Condizione di successo	<i>L'utente ha inviato con successo la richiesta</i>			
Condizione di fallimento	<i>L'utente non è riuscito ad inviare la richiesta</i>			
Attore primario	<i>Utente</i>			
Trigger	-			
DESCRIZIONE	Step	Attore Utente	Sistema	Attore esterno <i>Firebase</i>
	5	Preme il pulsante + su di un utente in <b>MU5</b>		
	6			Inserisce la richiesta tra le richieste dell'utente target
	7		Mostra messaggio di conferma di invio della richiesta	

Tabella 5.1

<b>USE CASE 5</b>	<b>Annulla richiesta di follow</b>			
Obiettivo	<i>Un utente vuole annullare una richiesta di follow inviata ad un altro utente</i>			
Precondizioni	<i>L'utente non è l'utente target, ha già inviato una richiesta all'utente target e ha effettuato una ricerca di utenti che ha prodotto dei risultati</i>			
Condizione di successo	<i>L'utente ha annullato con successo la richiesta</i>			
Condizione di fallimento	<i>L'utente non è riuscito ad annullare la richiesta</i>			
Attore primario	<i>Utente</i>			
Trigger	<i>-</i>			
DESCRIZIONE	Step	Attore Utente	Sistema	Attore Firebase
	6	Preme il pulsante X su di un utente in MUS		
	7			Rimuove la richiesta dalle richieste dell'utente target
	8		Mostra messaggio di conferma di annullamento della richiesta	

Tabella 5.2

<b>USE CASE 6</b>	<b>Visualizza feed</b>			
Obiettivo	<i>Un utente vuole visualizzare il suo feed delle azioni recenti dei suoi seguiti</i>			
Precondizioni	<i>L'utente risulta correttamente loggato</i>			
Condizione di successo	<i>L'utente ha visualizzato correttamente il suo feed</i>			
Condizione di fallimento	<i>Si è verificato un errore nella visualizzazione del feed</i>			
Attore primario	<i>Utente</i>			
Trigger	-			
DESCRIZIONE	Step	Attore Utente	Sistema	Attore Firebase
	1			Preleva le informazioni relative al feed dell'utente
	2		Mostra <b>MU3</b>	
<u>ESTENSIONE 1:</u> Non vi sono azioni recenti da mostrare	2 (1)		Mostra <b>MU3.1</b>	

Tabella 6.1

<b>USE CASE 7</b>	<b>Modifica nomi</b>			
Obiettivo	<i>Un utente vuole modificare il suo nome [risp. cognome]</i>			
Precondizioni	<i>L'utente risulta correttamente loggato</i>			
Condizione di successo	<i>L'utente ha modificato il suo nome [risp. cognome]</i>			
Condizione di fallimento	<i>L'utente non è riuscito a modificare il suo nome [risp. cognome]</i>			
Attore primario	<i>Utente</i>			
Trigger	<i>Pressione del tasto Profilo della barra di navigazione in MU3</i>			
DESCRIZIONE	Step	Attore Utente	Sistema	Attore esterno <i>Firebase</i>
	1		Mostra MU11	
	2	Pressione su <i>Modifica di MU11</i>		
	3		Mostra MU11.1	
	4	Pressione di <i>Modifica nome</i> [risp. <i>Modifica cognome</i> ] su <b>MU11.1</b>		
	5		Mostra MU11.2 [risp. MU11.3]	
	6	Inserimento del nuovo nome [risp. cognome]		
	7		Verifica la validità del nome [risp. cognome]	
	8			Aggiorna il database
	9		Aggiorna MU11	
<u>ESTENSIONE 1:</u> Il nome [risp. cognome] non è valido	8 (1)		Mostra MU11	
	9 (1)		Mostra messaggio di errore all'utente	

Tabella 7.1

<b>USE CASE 7</b>	<b>Modifica password</b>			
Obiettivo	<i>Un utente vuole modificare la sua password</i>			
Precondizioni	<i>L'utente risulta correttamente loggato</i>			
Condizione di successo	<i>L'utente ha modificato la sua password</i>			
Condizione di fallimento	<i>L'utente non è riuscito a modificare la sua password</i>			
Attore primario	<i>Utente</i>			
Trigger	<i>Pressione del tasto Profilo della barra di navigazione in MU3</i>			
DESCRIZIONE	Step	Attore Utente	Sistema	Attore esterno <i>Firebase</i>
	1		Mostra MU11	
	2	Pressione su <i>Modifica di MU11</i>		
	3		Mostra MU11.1	
	4	Pressione di <i>Modifica password</i> su MU11.1		
	5		Mostra MU11.4	
	6	Inserimento della nuova password		
	7		Verifica la validità della password	
	8			Verifica che non sia passato troppo tempo dal Log In
	9			Aggiorna il database
ESTENSIONE 1: La password non è valida	8 (1)		Mostra MU11	
	9 (1)		Mostra messaggio di errore all'utente	
ESTENSIONE 2: E' passato troppo tempo dall'ultimo Log In	9 (2)		Mostra MU11	
	10 (2)		Mostra messaggio di errore all'utente	

Tabella 7.2

<b>USE CASE 7</b>	<b>Modifica foto profilo</b>			
Obiettivo	<i>Un utente vuole modificare la sua foto profilo</i>			
Precondizioni	<i>L'utente ha dato l'autorizzazione di accesso alla fotocamera e all'archivio</i>			
Condizione di successo	<i>L'utente ha modificato la sua foto profilo</i>			
Condizione di fallimento	<i>L'utente non è riuscito a modificare la sua foto profilo</i>			
Attore primario	<i>Utente</i>			
Trigger	<i>Pressione del tasto Profilo della barra di navigazione in MU3</i>			
DESCRIZIONE	Step	Attore Utente	Sistema	Attore esterno <i>Firebase</i>
	1		Mostra MU11	
	2	Pressione su <i>Modifica</i> di MU11		
	3		Mostra MU11.1	
	4	Pressione di <i>Modifica foto profilo</i> su MU11.1		
	5		Avvia procedura di sceglimento	
	6		Termina procedura di sceglimento	
	7			Aggiorna lo Storage e il database
	8		Aggiorna MU11	
ESTENSIONE 1: La procedura di sceglimento non è andata a buon fine	7 (1)		Mostra MU11	
	8 (1)		Mostra messaggio di errore all'utente	

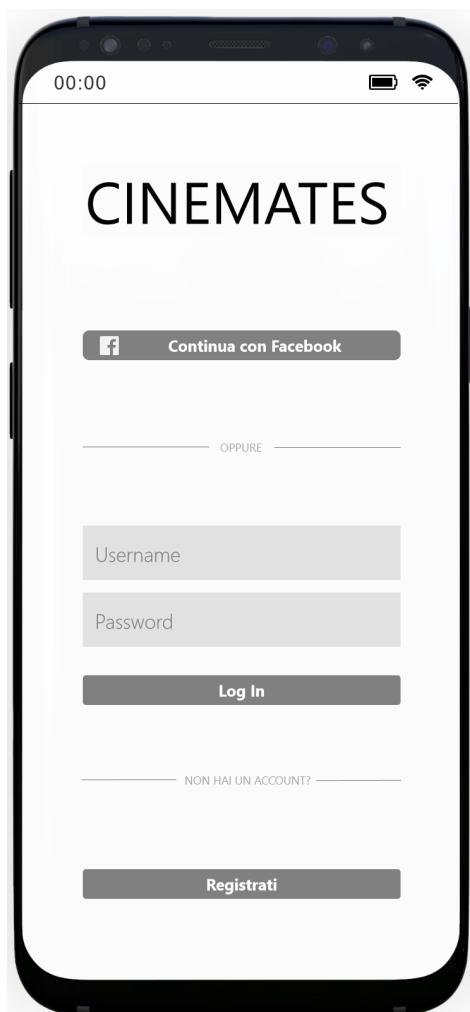
Tabella 7.3

<b>USE CASE 8</b>	<b>Visualizza statistiche in tempo reale</b>			
Obiettivo	<i>Un amministratore vuole visualizzare le statistiche sul sistema tramite il client</i>			
Precondizioni	<i>L'amministratore risulta loggato correttamente</i>			
Condizione di successo	<i>L'amministratore ha accesso alle statistiche aggiornate</i>			
Condizione di fallimento	<i>L'amministratore non è riuscito a visionare i dati</i>			
Attore primario	<b>Amministratore</b>			
Trigger	<i>Accesso al client desktop/pressione del tasto Aggiorna Statistiche su MU14/MU14.1</i>			
DESCRIZIONE	Step	Attore Amministratore	Sistema	Attore esterno <i>Firebase RTDB API</i>
	1		Invia richiesta (HTTP) all' API	
	2			Producি risposta (HTTP) contenente i dati
	3		Ricezione ed elaborazione dei dati	
	4		Mostra <b>MU14.1</b>	
<b>ESTENSIONE 1:</b> Si tratta del primo Log In	1 (1)	Effettua l'update delle credenziali come indicato in <b>USE CASE 1 – Log in Amministratore – Estensione 1</b>		
	2 (1)		Invia richiesta (HTTP) all' API	
	3 (1)			Producি risposta (HTTP) contenente i dati
	4 (1)		Ricezione ed elaborazione dei dati	
	5 (1)		Mostra <b>MU14.1</b>	
<b>ESTENSIONE 2:</b> L'amministratore è il Superadmin	4 (2)		Mostra <b>MU14</b>	
<b>ESTENSIONE 3:</b> Le statistiche non possono essere correttamente visualizzate a causa di errori di connessione	2 (3)		Mostra un messaggio di errore	
	3 (3)		Mostra <b>MU14/MU14.1</b>	

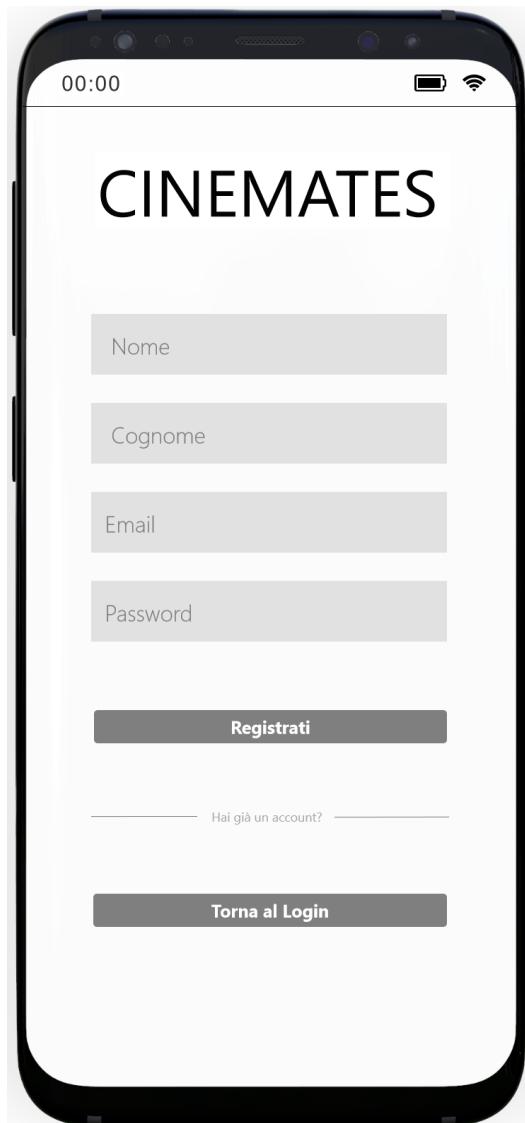
Tabella 8.1

### **2.1.3 Mock-up dell'interfaccia utente**

Di seguito sono riportati i Mock Up che rappresentano le principali schermate dell'applicazione mobile e del client desktop di amministrazione, alle cui componenti si fa riferimento nelle tabelle di Cockburn della sottosezione precedente (2.1.2).



1. Log In



2. Registrazione



3. Pagina principale



3.1 Pagina principale - vuota



4. Seguiti



4.1 Seguiti - vuota



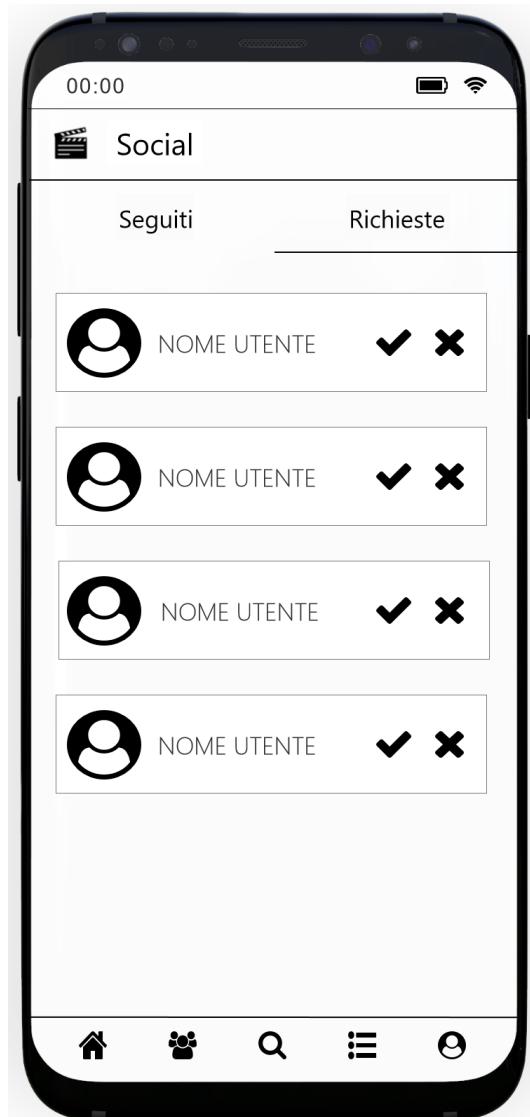
5. Ricerca utenti



5.1 Ricerca utenti - vuota



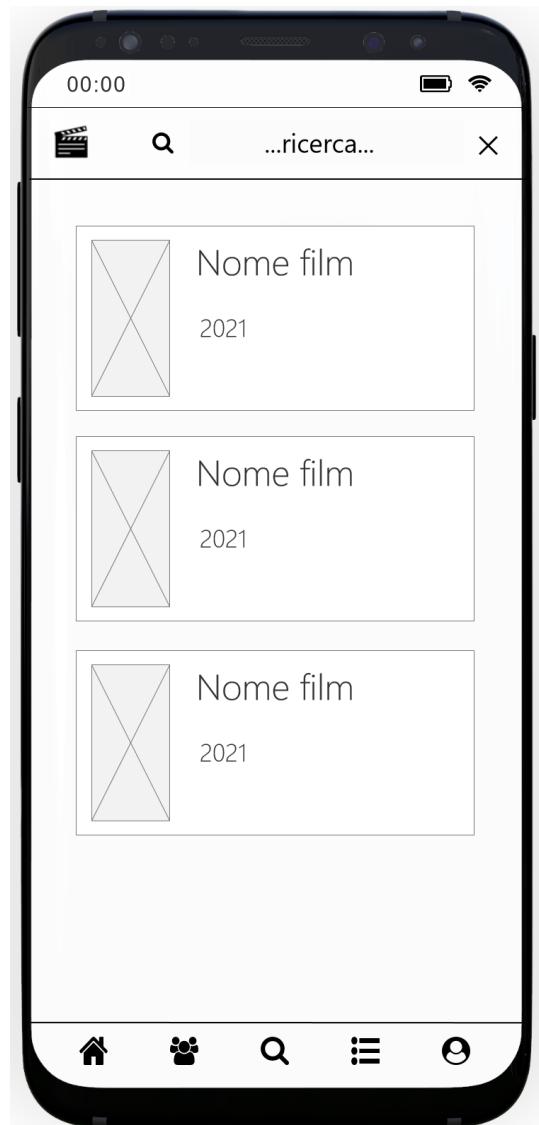
5.2 Ricerca utenti - nessun risultato



6. Richieste



6.1 Richieste - vuota



7. Ricerca film



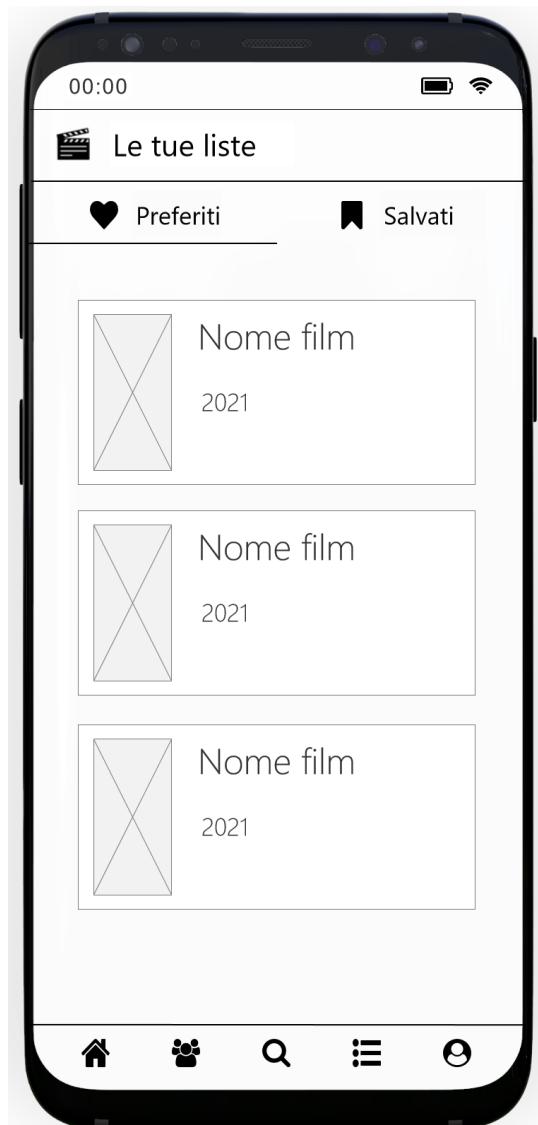
7.1 Ricerca film - vuota



7.2 Ricerca film - nessun risultato



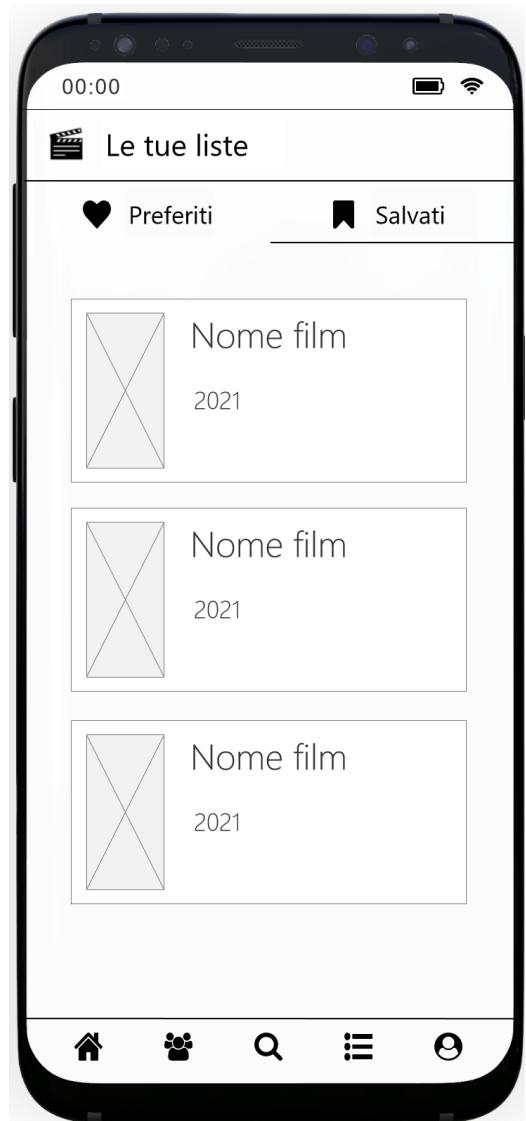
8. Scheda film



9. Preferiti



9.1 Preferiti - vuota



10. Salvati



10.1 Salvati - vuota



11. Profilo



11.1 Modifica dati



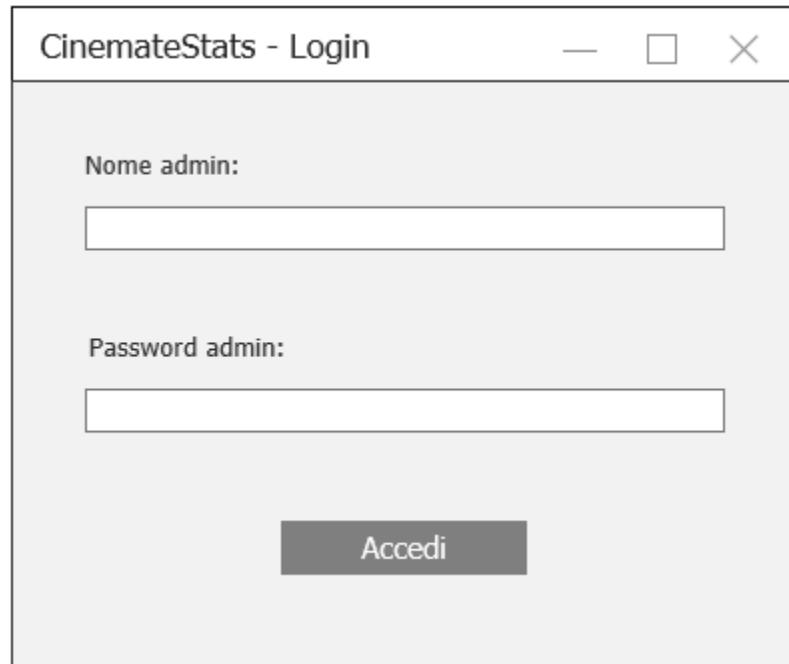
11.2 Modifica dati - nome



11.3 Modifica dati - cognome



11.4 Modifica dati - password



12 Log In Admin

CinemateStats - Creds

— □ ×

Nuovo nome:

Nuova password:

Conferma

13 Modifica credenziali Superadmin

CinamateStats

Utenti:	<input type="text"/>	Film di avventura nelle liste:	<input type="text"/>
Utenti connessi:	<input type="text"/>	Film di azione nelle liste:	<input type="text"/>
		Commedie nelle liste:	<input type="text"/>
Collegamenti tra utenti:	<input type="text"/>	Thriller nelle liste:	<input type="text"/>
Ricerche di film effettuate:	<input type="text"/>	Horror nelle liste:	<input type="text"/>

[Aggiorna Statistiche](#) [Genera nuove credenziali](#)

14 Visualizzazione statistiche Superadmin

CinamateStats

Utenti:	<input type="text"/>	Film di avventura nelle liste:	<input type="text"/>
Utenti connessi:	<input type="text"/>	Film di azione nelle liste:	<input type="text"/>
		Commedie nelle liste:	<input type="text"/>
Collegamenti tra utenti:	<input type="text"/>	Thriller nelle liste:	<input type="text"/>
Ricerche di film effettuate:	<input type="text"/>	Horror nelle liste:	<input type="text"/>

**Aggiorna Statistiche**

#### 14.1 Visualizzazione statistiche

#### **2.1.4 Glossario**

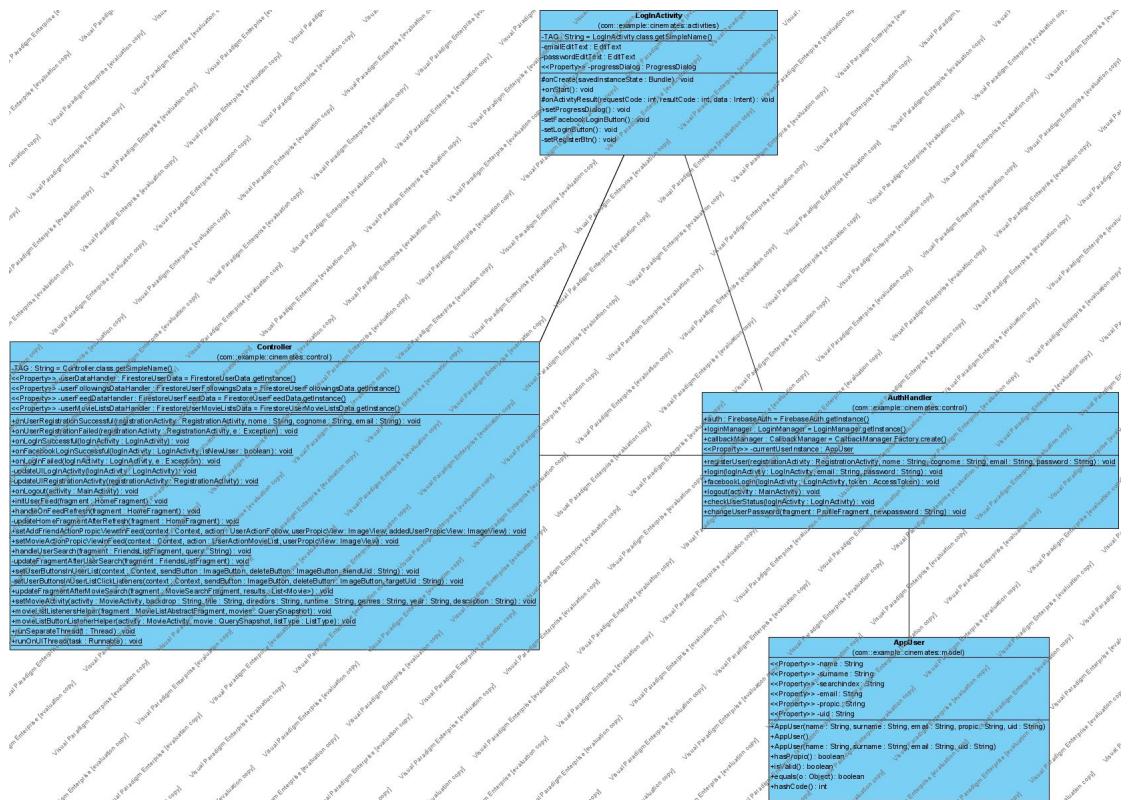
## **2.2 Modelli di Dominio**

### **2.2.1 Classi diagram di analisi**

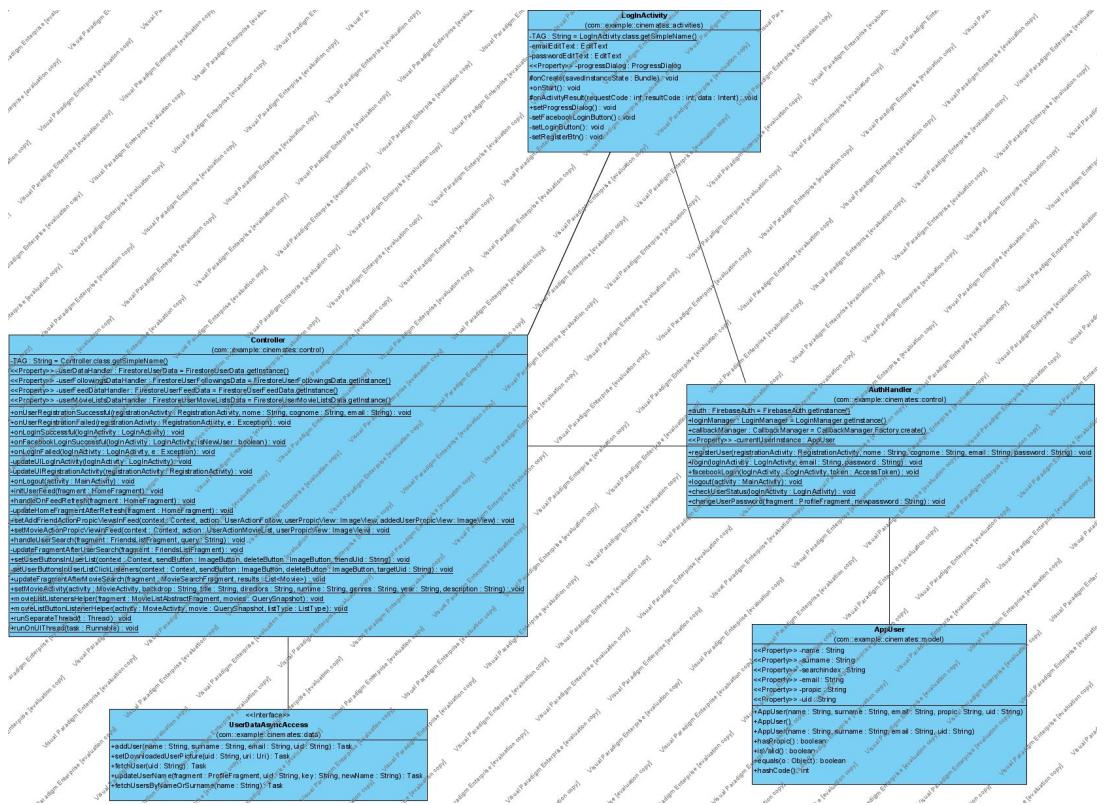
Seguono, per ciascuna funzionalità, le classi e le relazioni tra di esse che ne costituiscono la struttura portante.

Risulta chiaro che, nel contesto dell'analisi, i diagrammi sono stati costruiti in maniera tale da esplicitare le principali entità coinvolte nello svolgimento di ciascuna funzionalità. In ciascuno di essi viene evidenziata l'architettura a tre strati del sistema, di cui si discuterà in dettaglio nella sezione 3.1. Anche per questo motivo si è preferito omettere le classi di modello quando non indispensabili, in quanto esse rappresentano oggetti di supporto e non hanno un ruolo di rilievo nell'architettura stratificata del sistema.

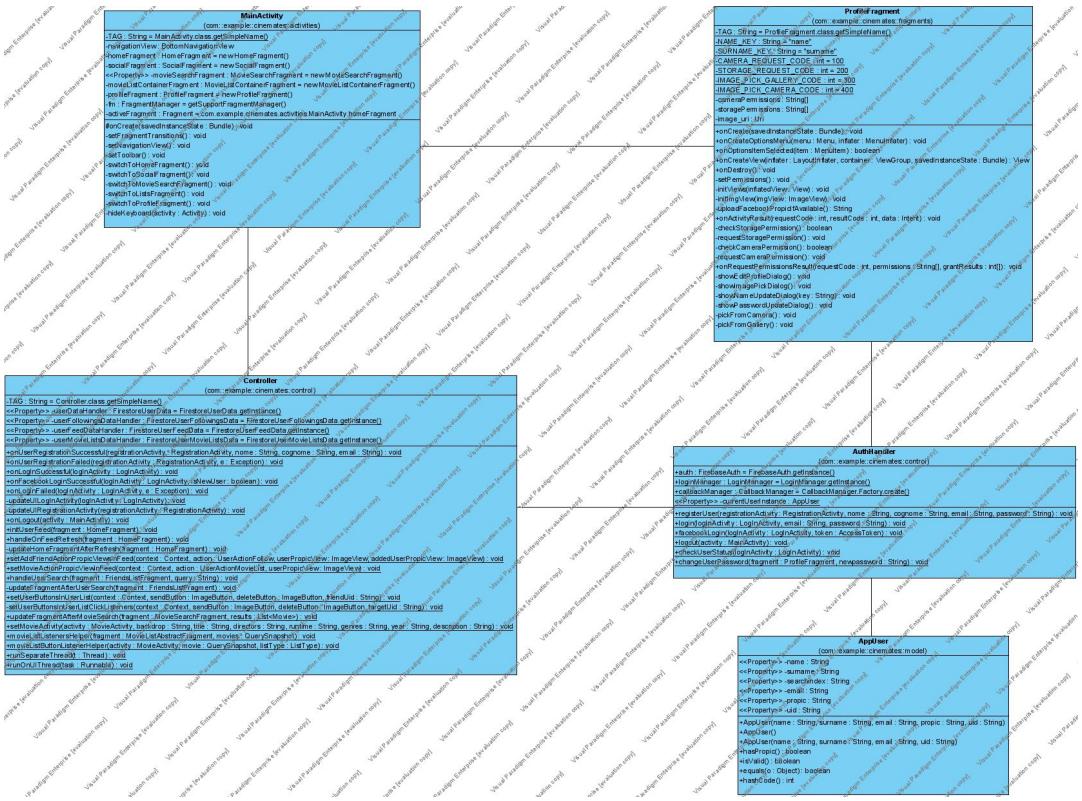
Oltre a ciò si precisa che siccome alcune funzionalità danno origine alla stessa struttura per quanto riguarda le interazioni tra le entità che le costituiscono, non vi è una corrispondenza diretta tra ciascun caso d'uso e i relativi diagrammi. Le descrizioni di questi ultimi saranno atte a disambiguare.



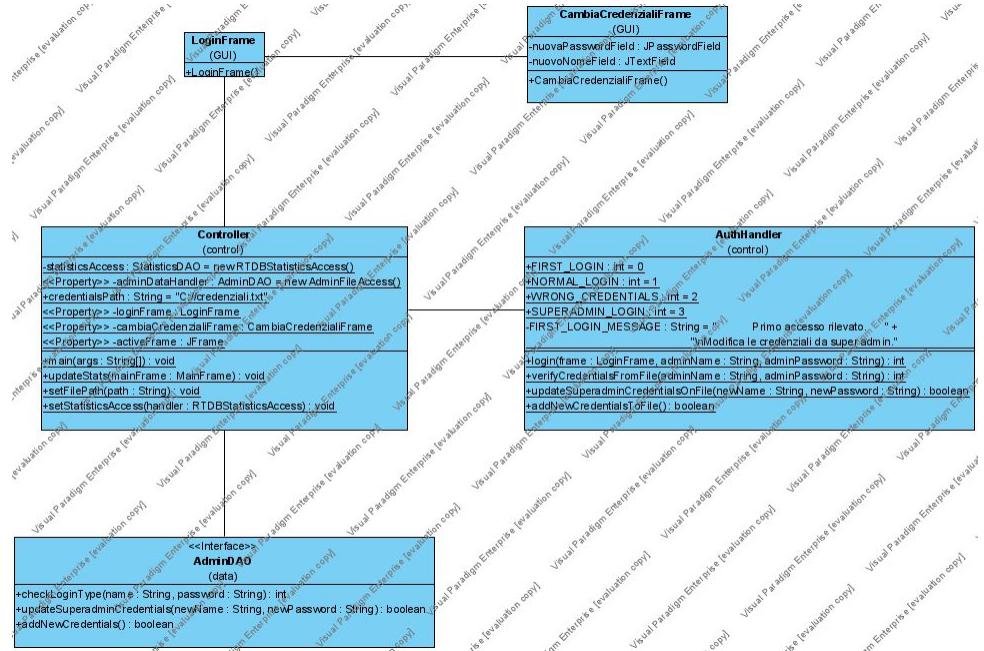
### 1.1/1.2 Log in e Log in tramite Facebook



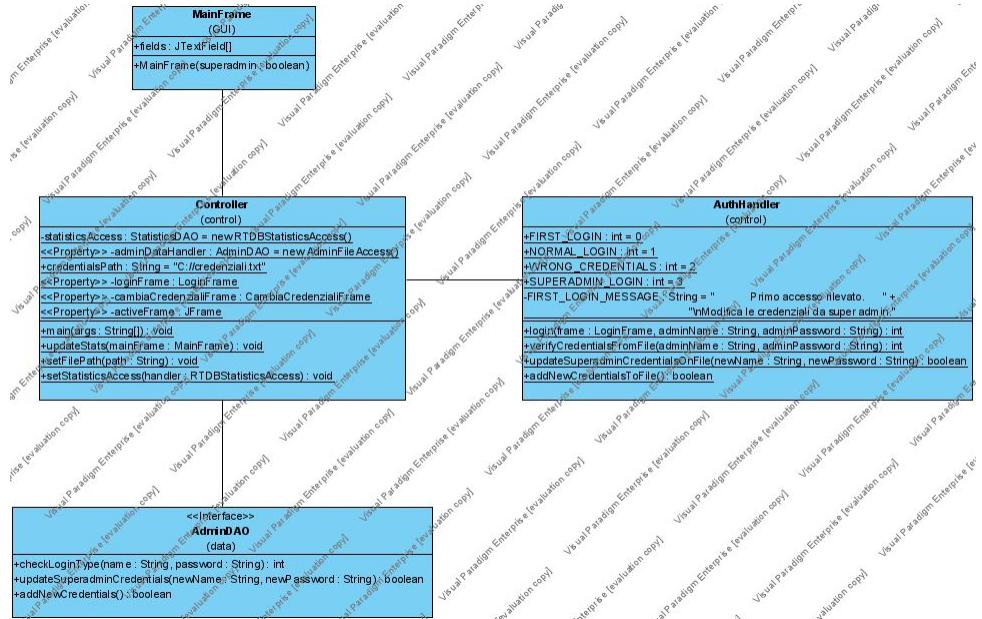
### 1.3 Registrazione di un nuovo profilo



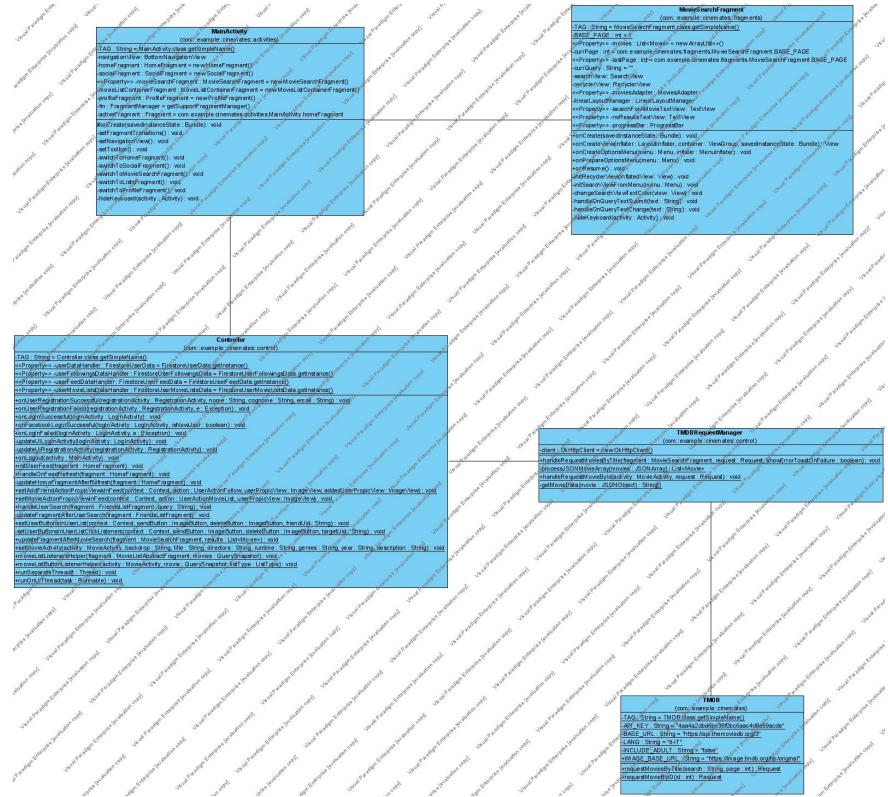
## 1.4 Log out



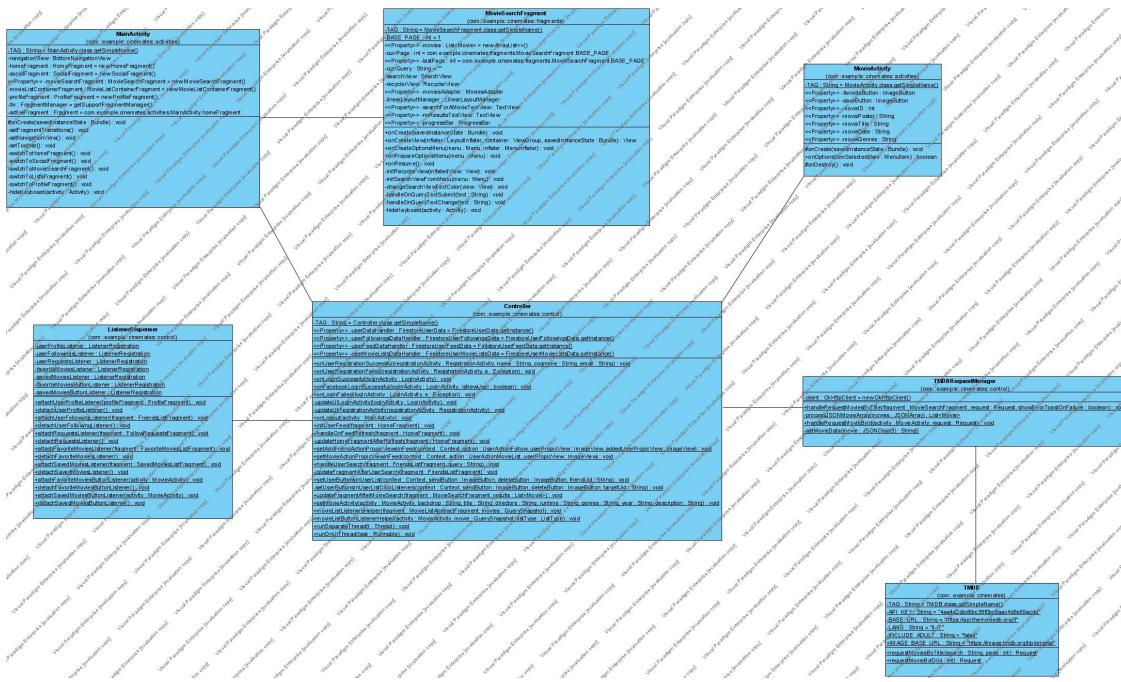
## 2.1 Log in amministratore



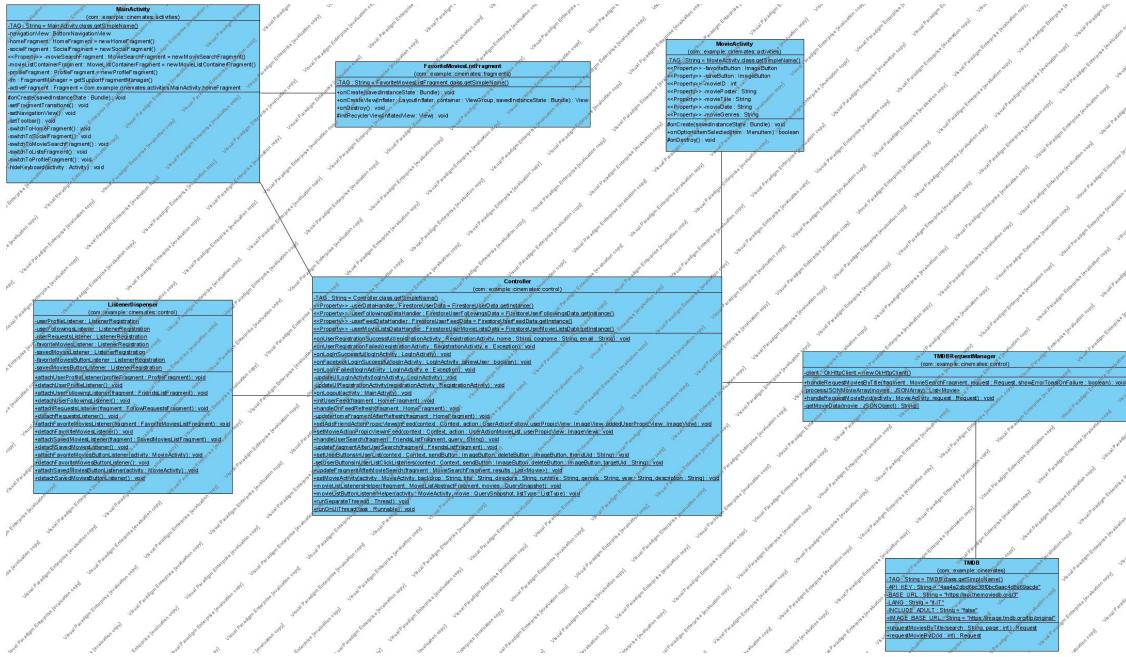
## 2.2 Generazione di nuove credenziali



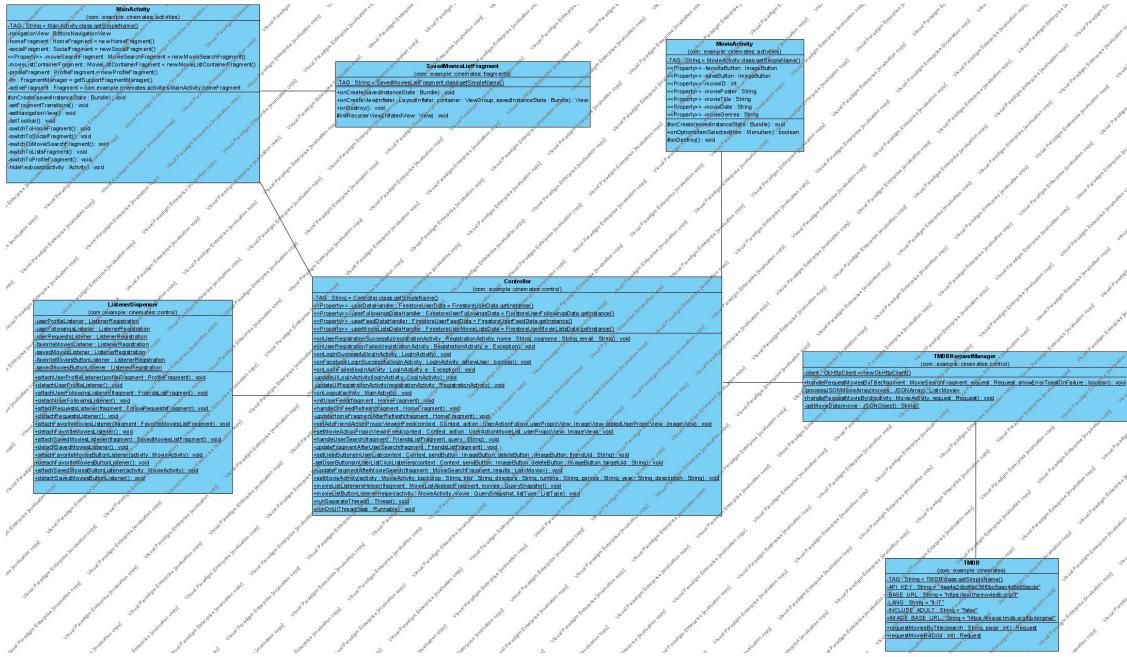
### 3.1 Ricerca di film



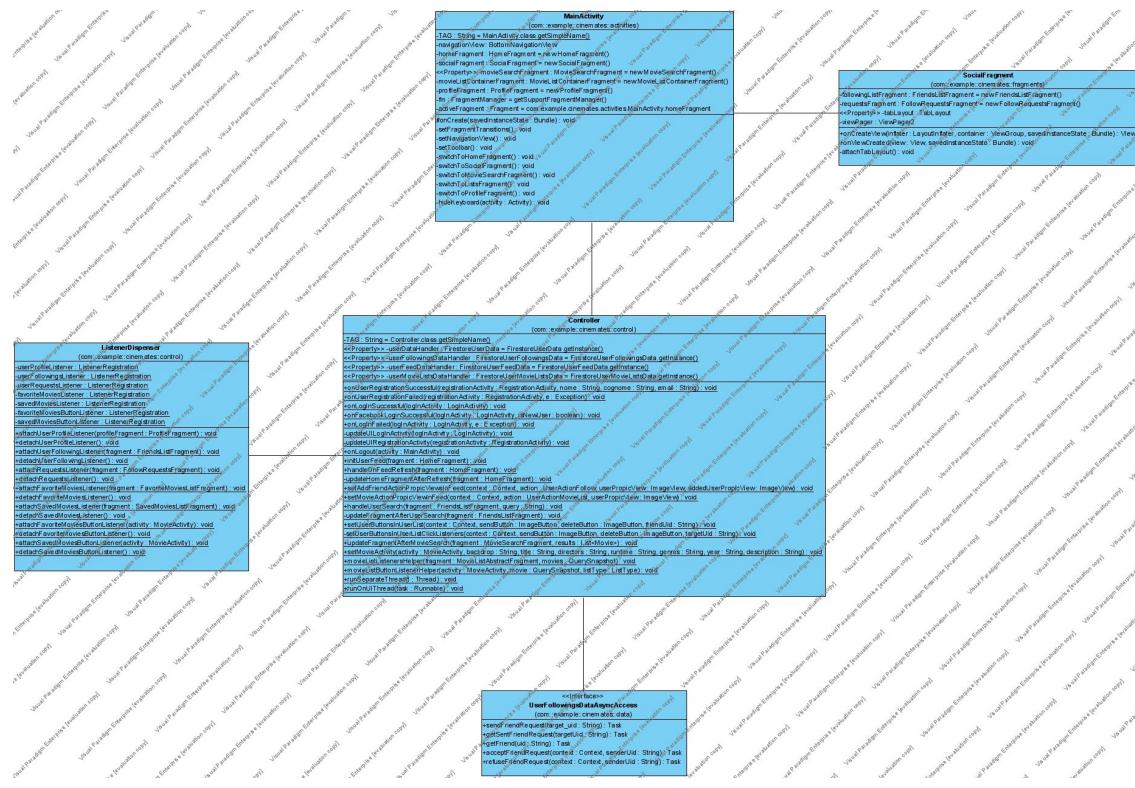
#### 4.1/4.3 Aggiunta di un film ai preferiti/salvati



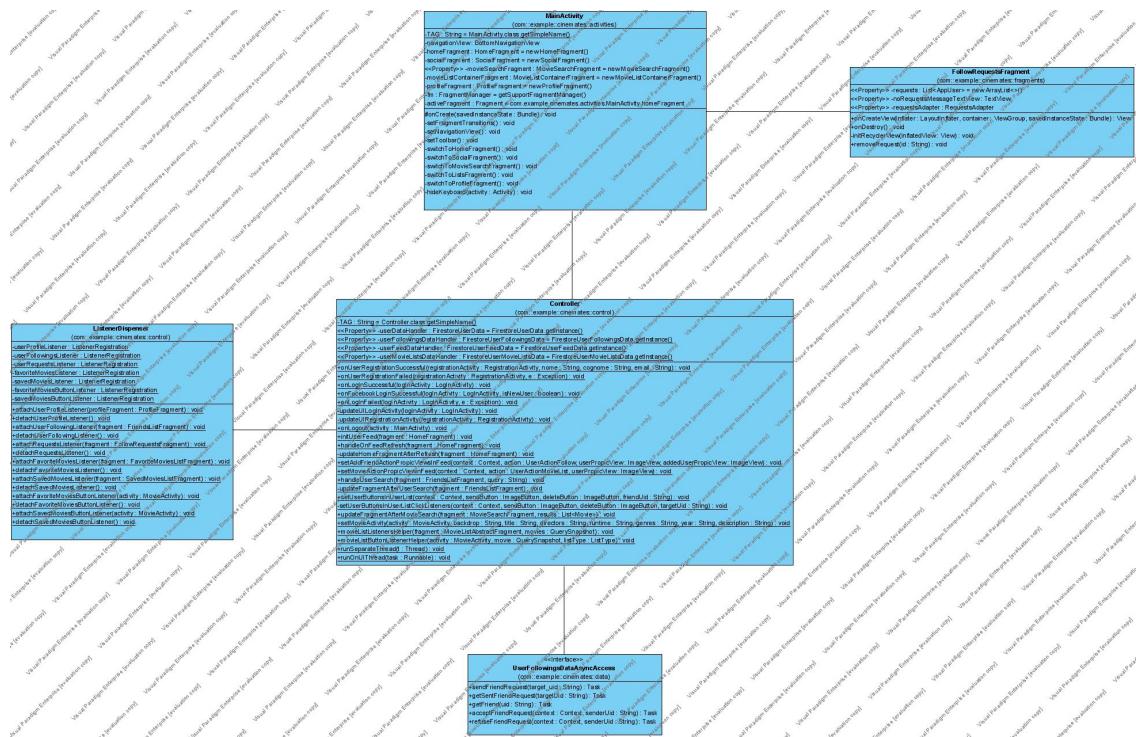
#### 4.2 Rimozione di un film dai preferiti



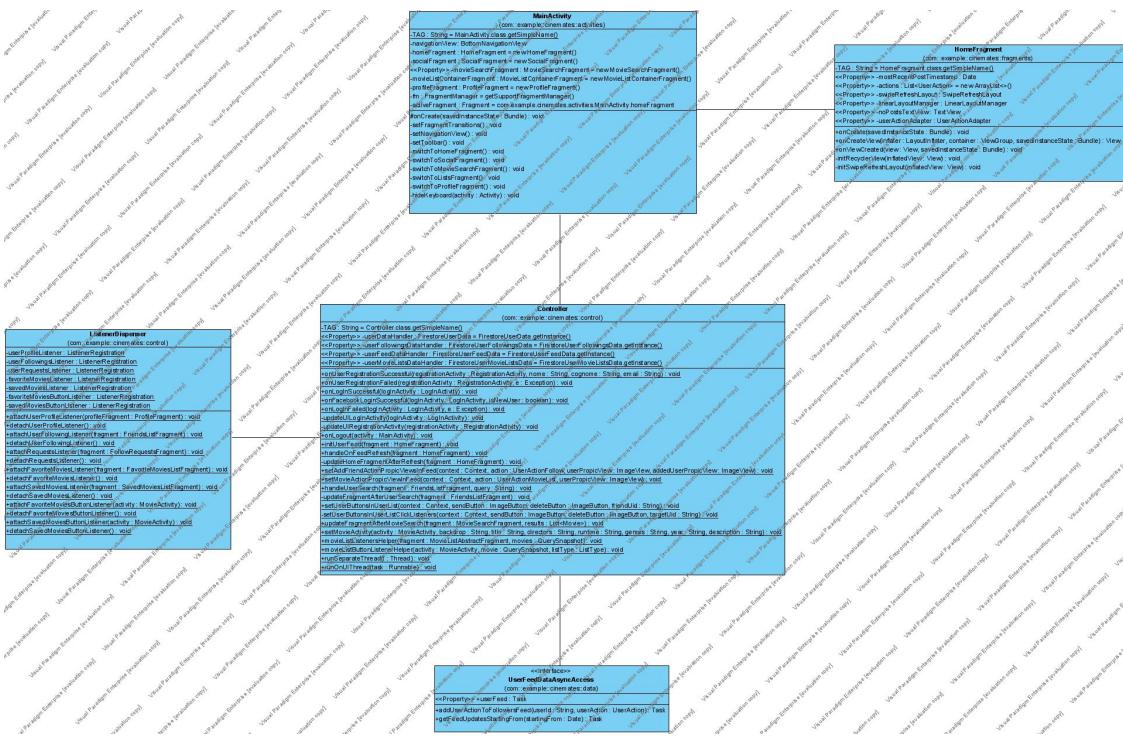
#### 4.4 Rimozione di un film dai salvati



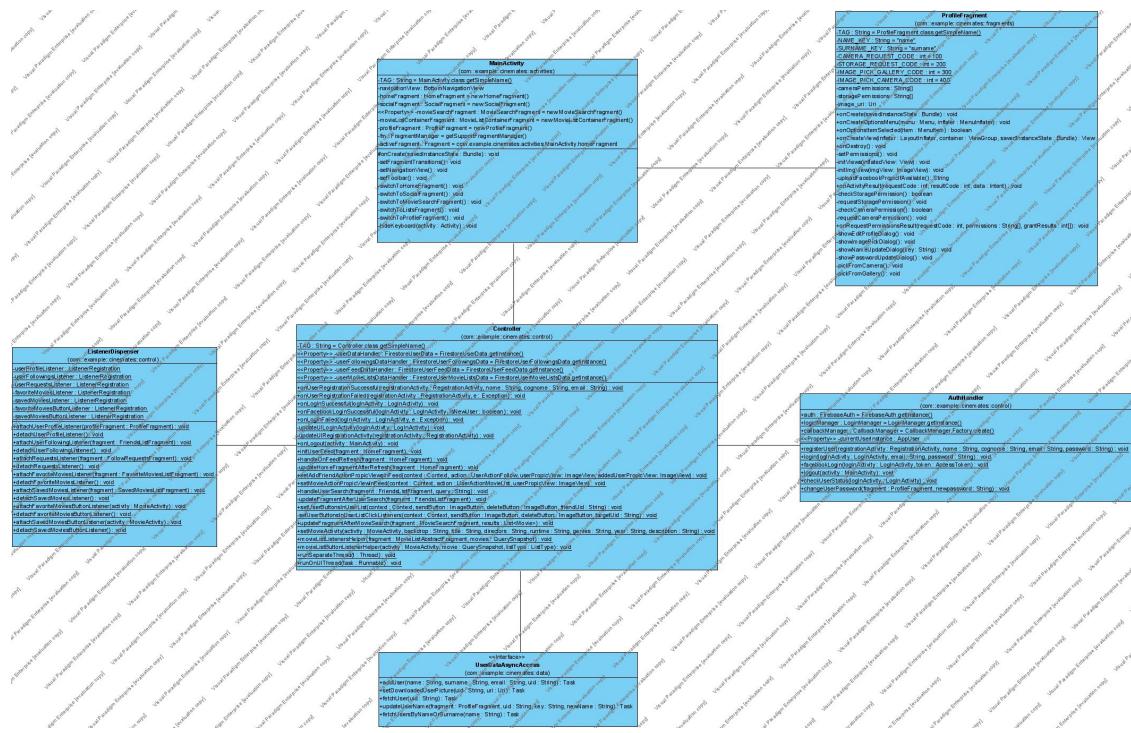
### 5.1/5.2 Invio/Annullamento di una richiesta di follow



### 5.3 Accettazione/Rifiuto di una richiesta di follow



### 6.1 Visualizzazione del feed



## 7.1/7.2/7.3 Aggiornamento dei dati del profilo

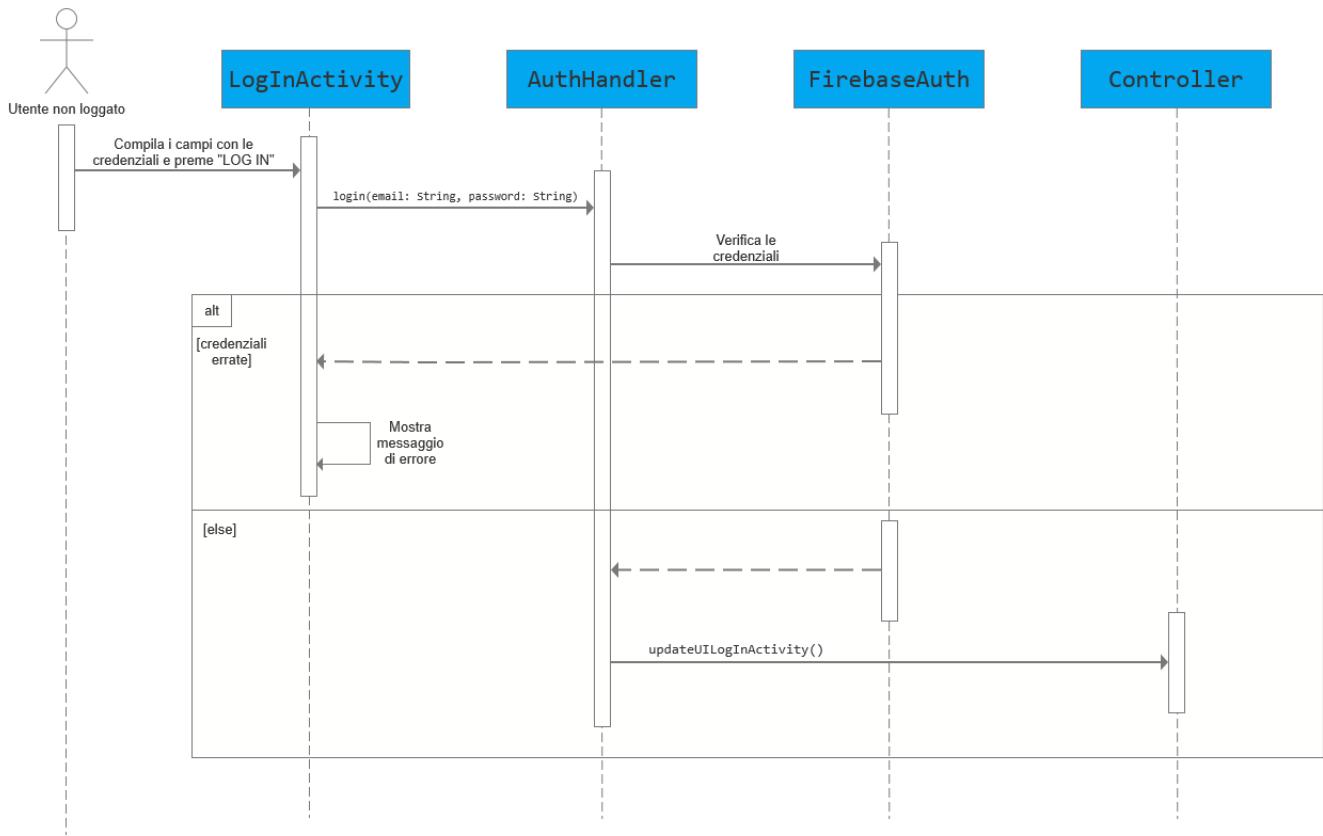


## 8.1 Visualizzazione delle statistiche del sistema

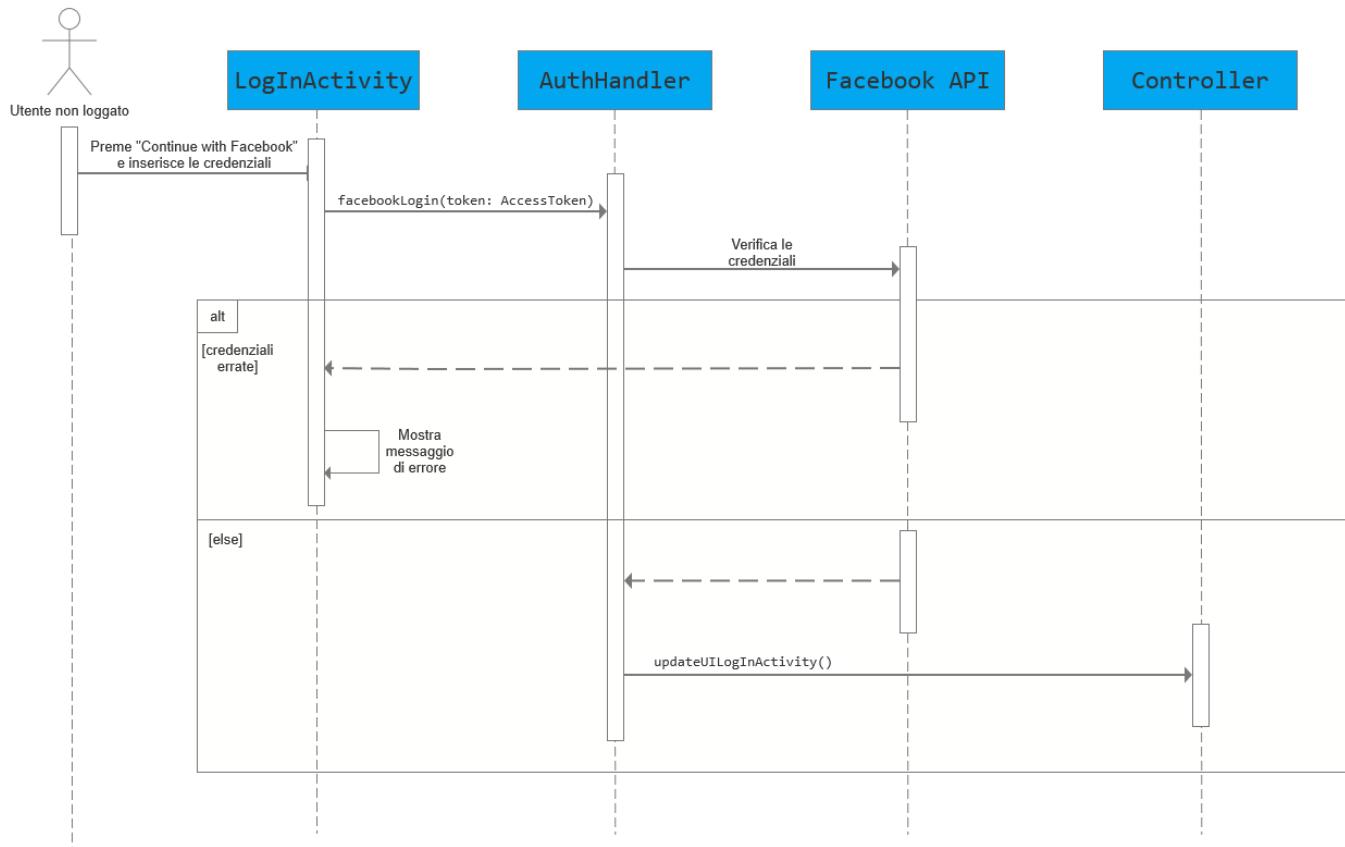
### **2.2.2 Diagrammi di sequenza di analisi**

I sequence diagram di analisi qui riportati fanno da ulteriore supporto alle rappresentazioni grafiche dei casi d'uso tramite notazione UML. Essi presentano maggiore vicinanza alla sezione implementativa dei moduli del software commissionato, sebbene si sia fatto in modo tale da mantenere un determinato grado di astrazione rispetto al prodotto finito, eliminando interazioni superflue e rendendo ciascun diagramma quanto più sintetico possibile.

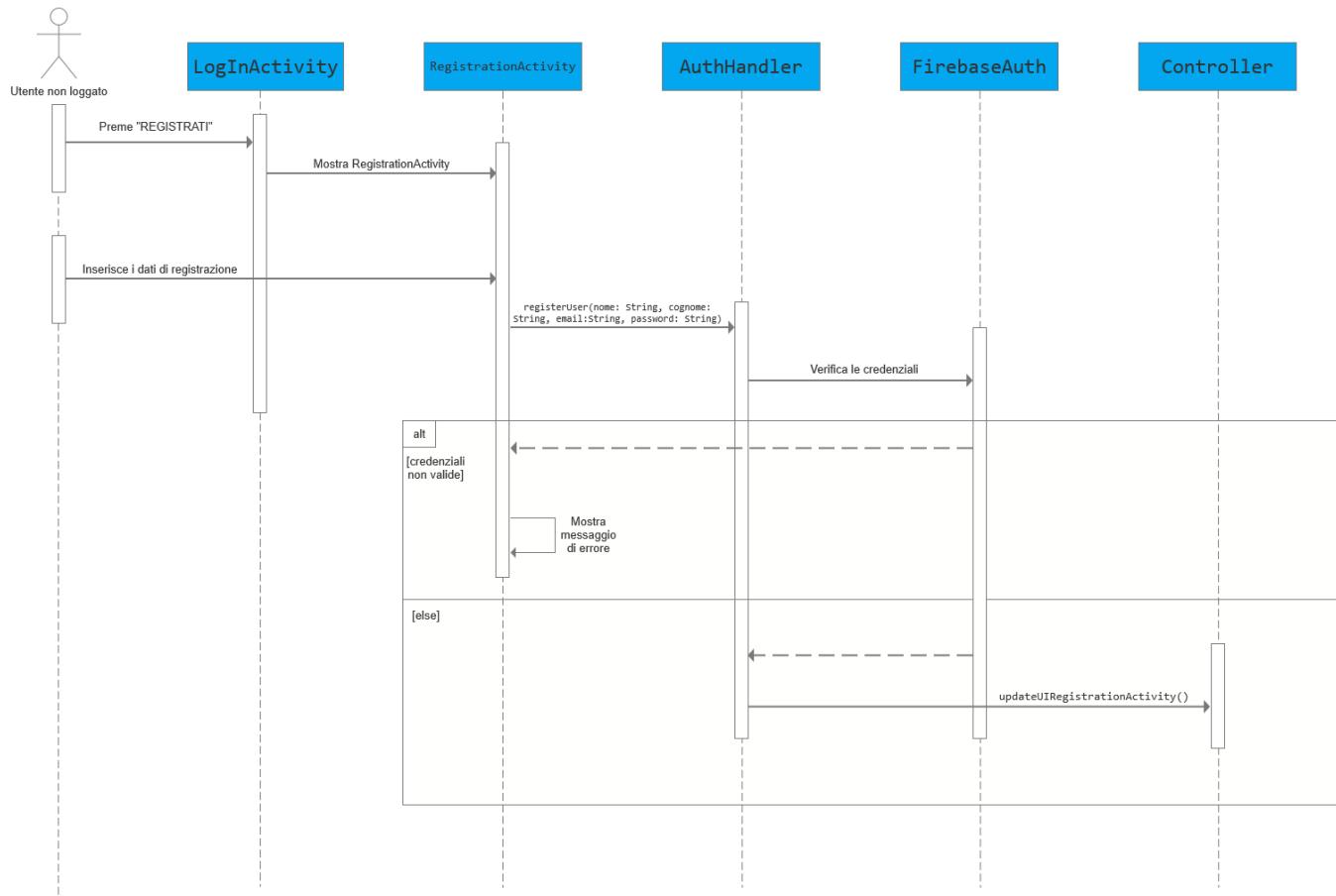
Questi diagrammi sono stati creati per analizzare l'andamento temporale della funzionalità che essi vanno a rappresentare. A differenza delle tabelle di Cockburn presentate in precedenza (sottosezione 2.1.2) e dei diagrammi di attività della prossima sottosezione (2.2.3), dove il caso d'uso è stato considerato introducendo anche in minima parte gli eventi che conducono ad esso, qui viene presa in esame la funzionalità con contesto quasi nullo.



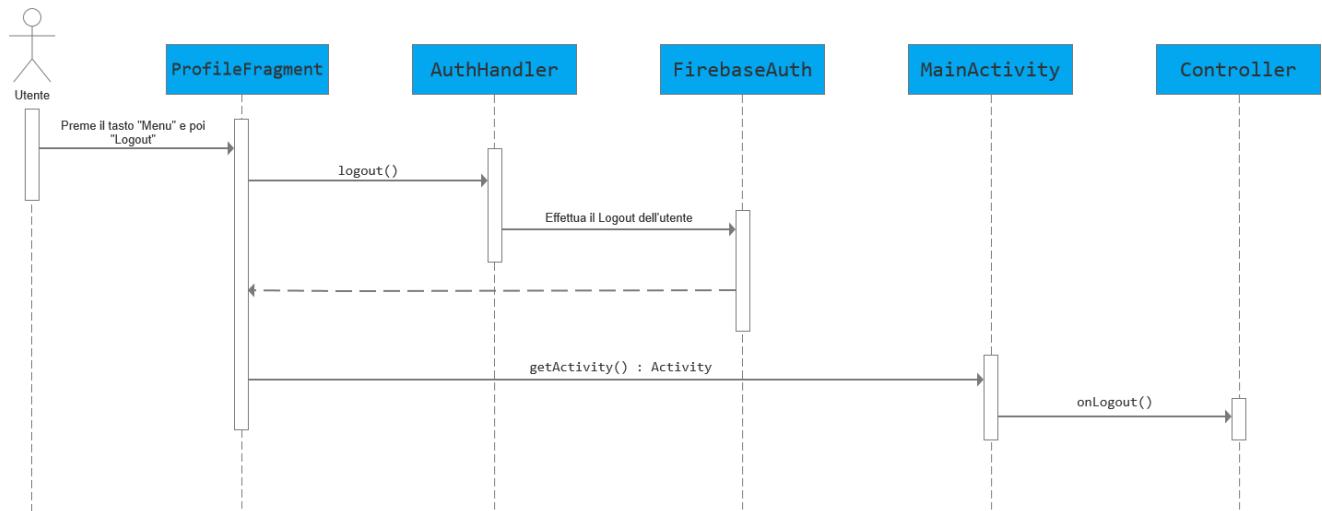
### 1.1 Log in



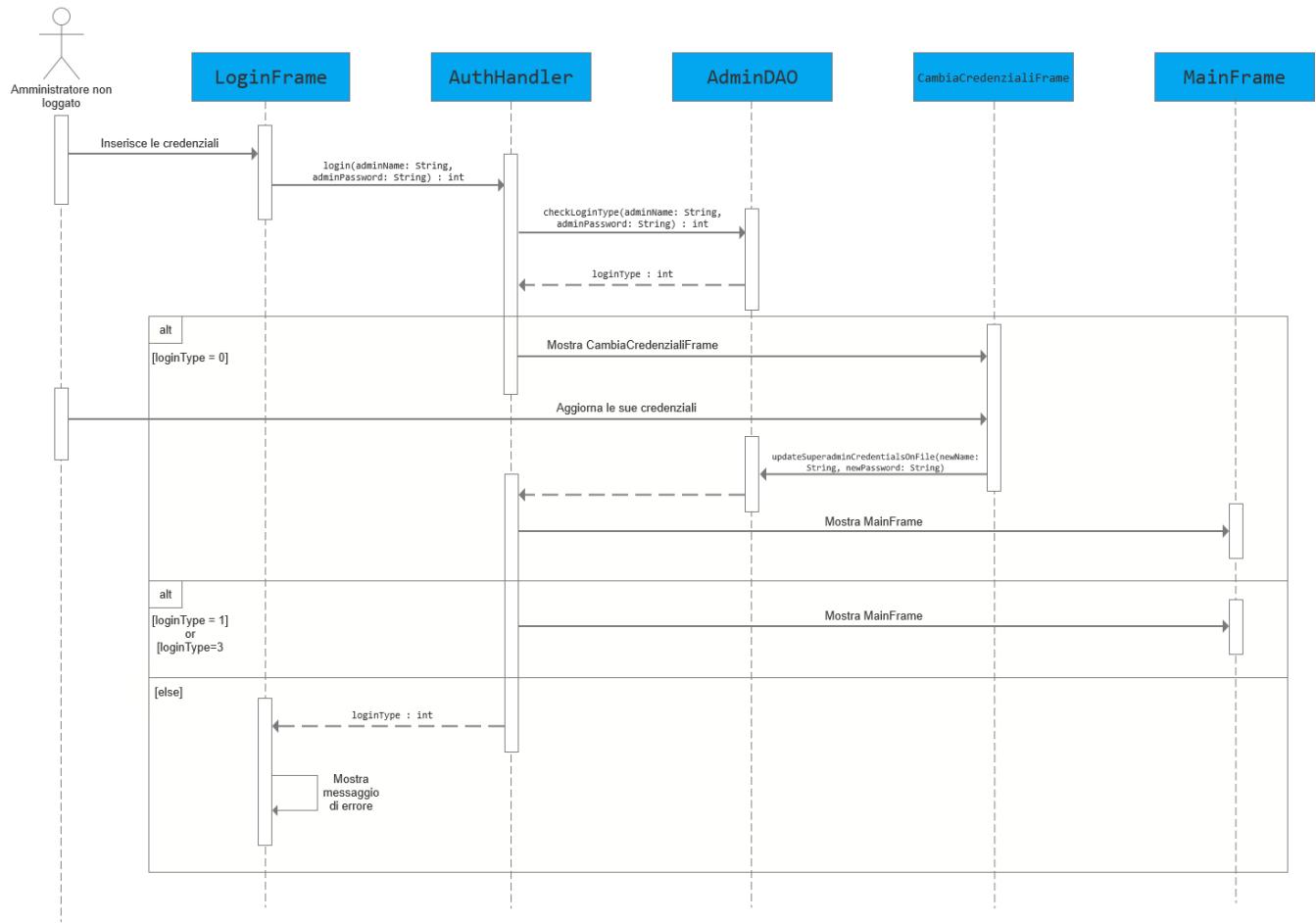
1.2 Log in tramite Facebook



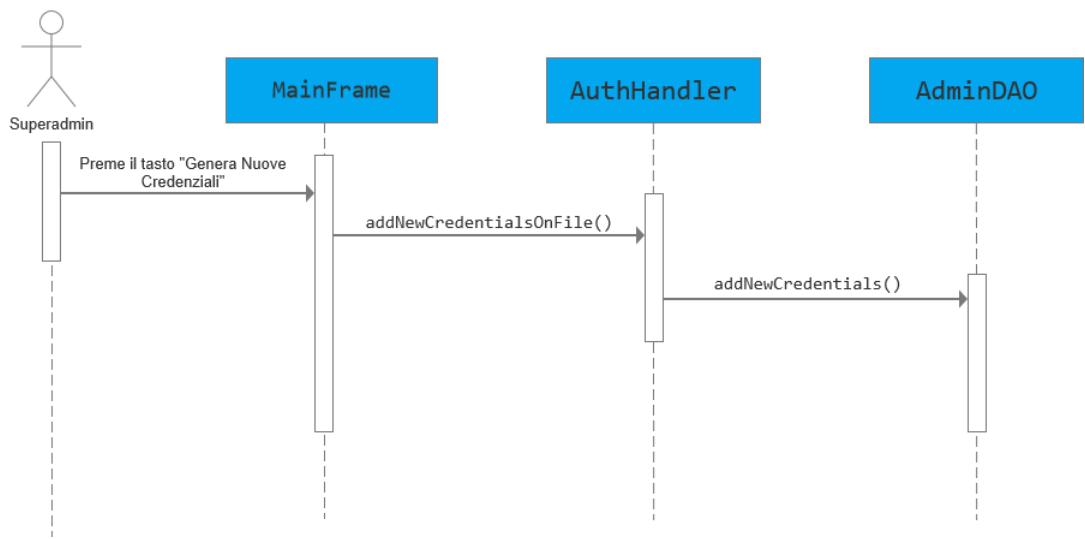
### 1.3 Registrazione di un nuovo profilo



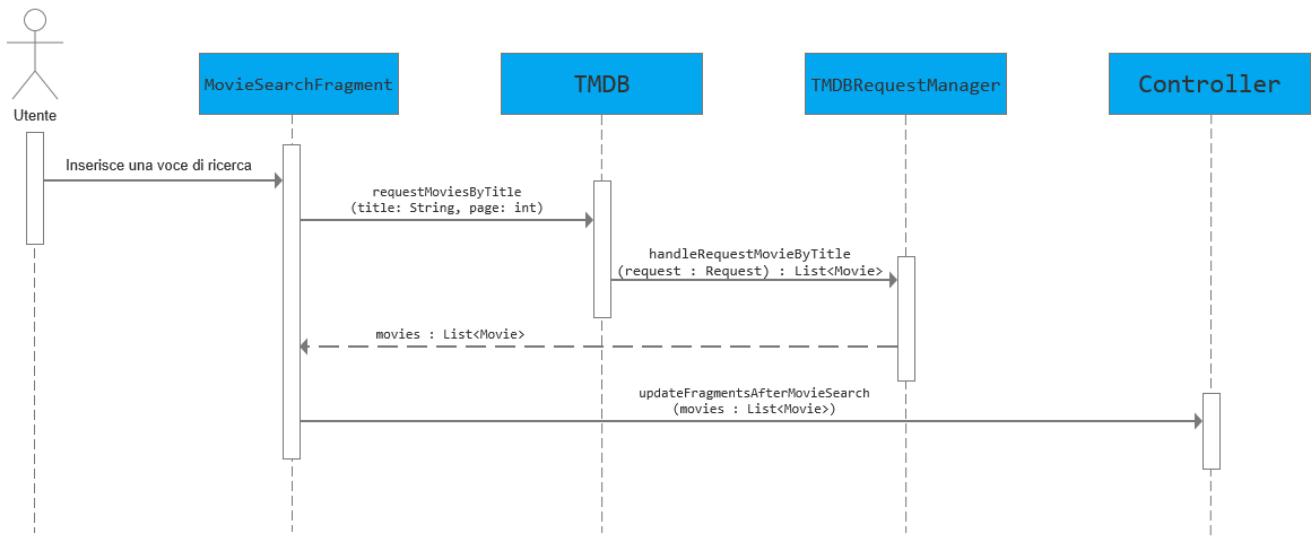
#### 1.4 Log out



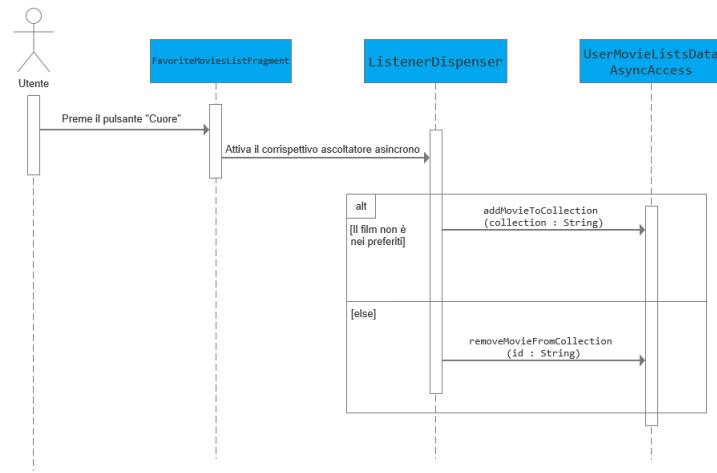
## 2.1 Log In amministratore



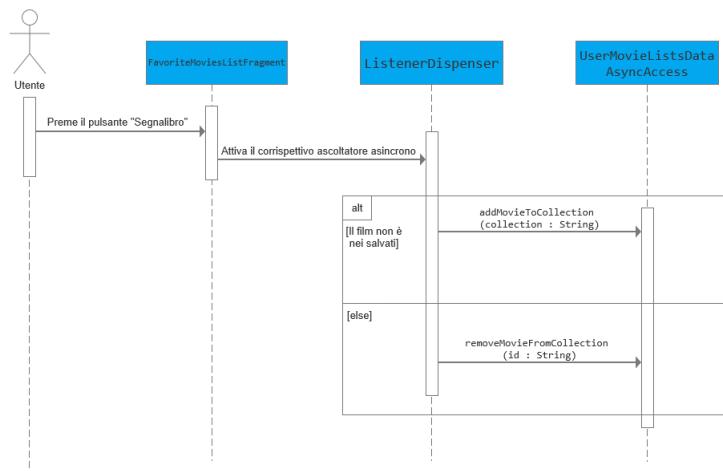
2.2 Generazione di nuove credenziali



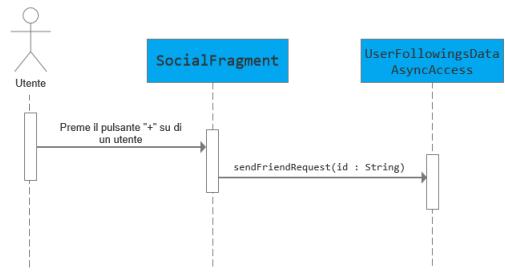
### 3.1 Ricerca di film



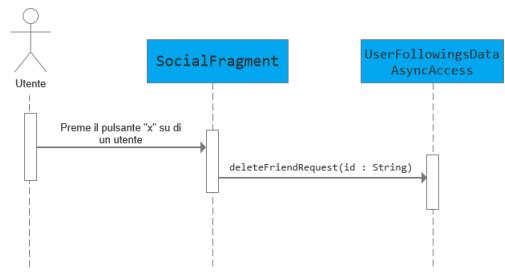
4.1/4.2 Aggiunta/Rimozione di un film dai preferiti



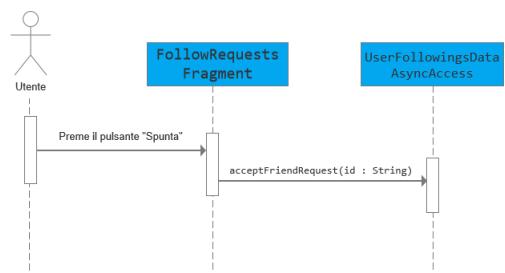
4.3/4.4 Aggiunta/Rimozione di un film dai salvati



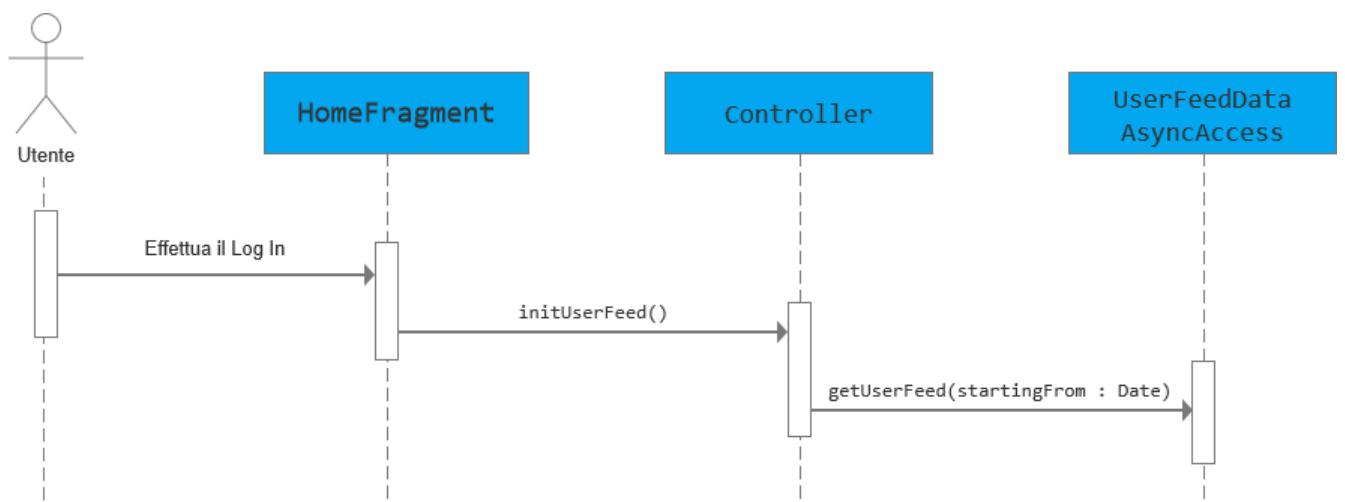
5.1 Invio di una richiesta di follow



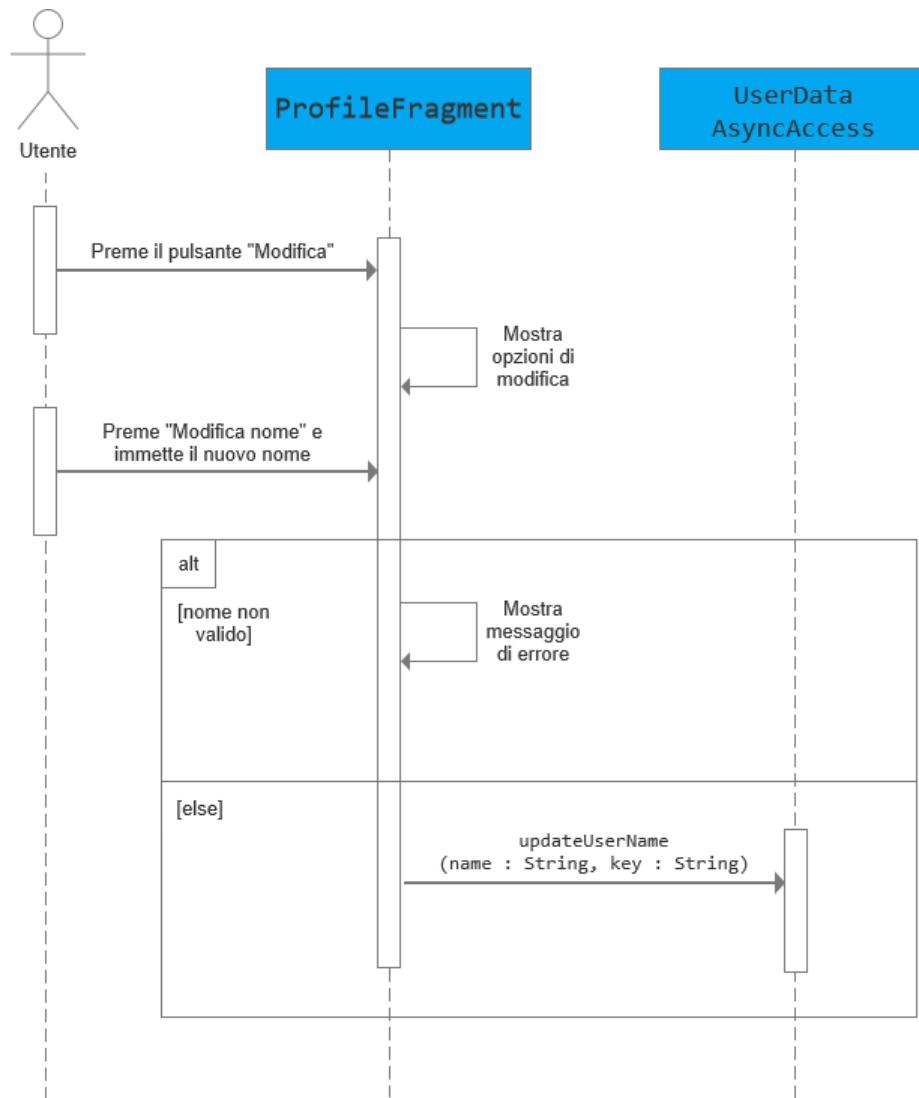
5.2 Annullamento di una richiesta di follow



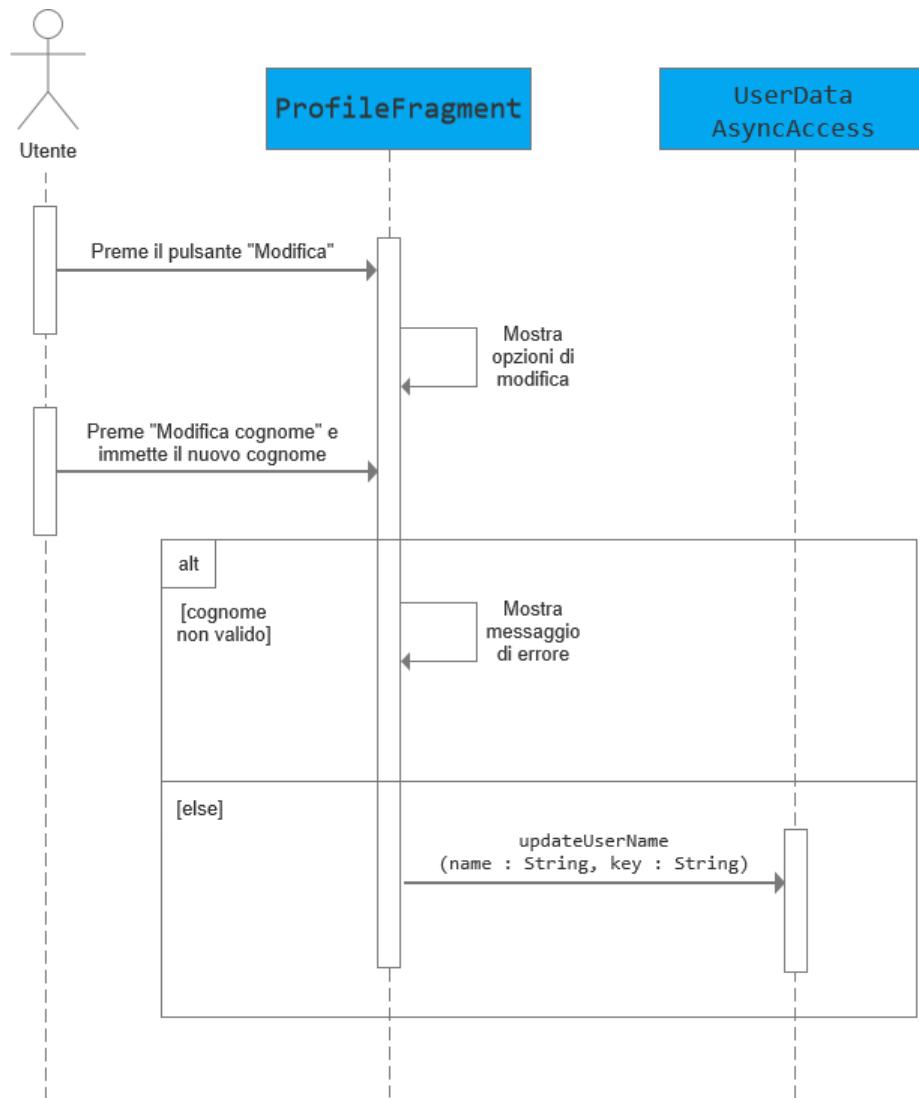
5.3 Accettazione/Rifiuto di una richiesta di follow



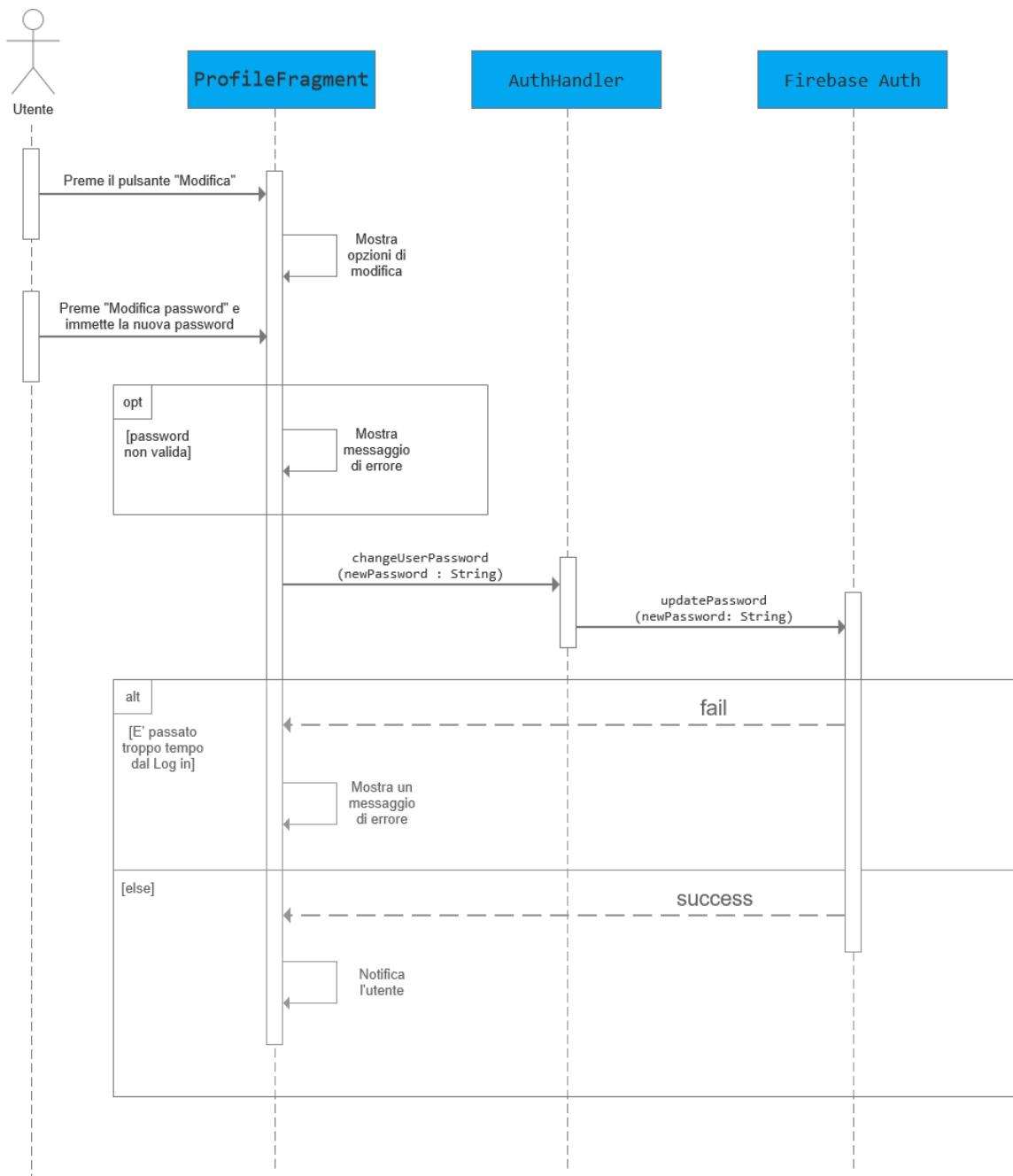
6.1 Visualizzazione del feed



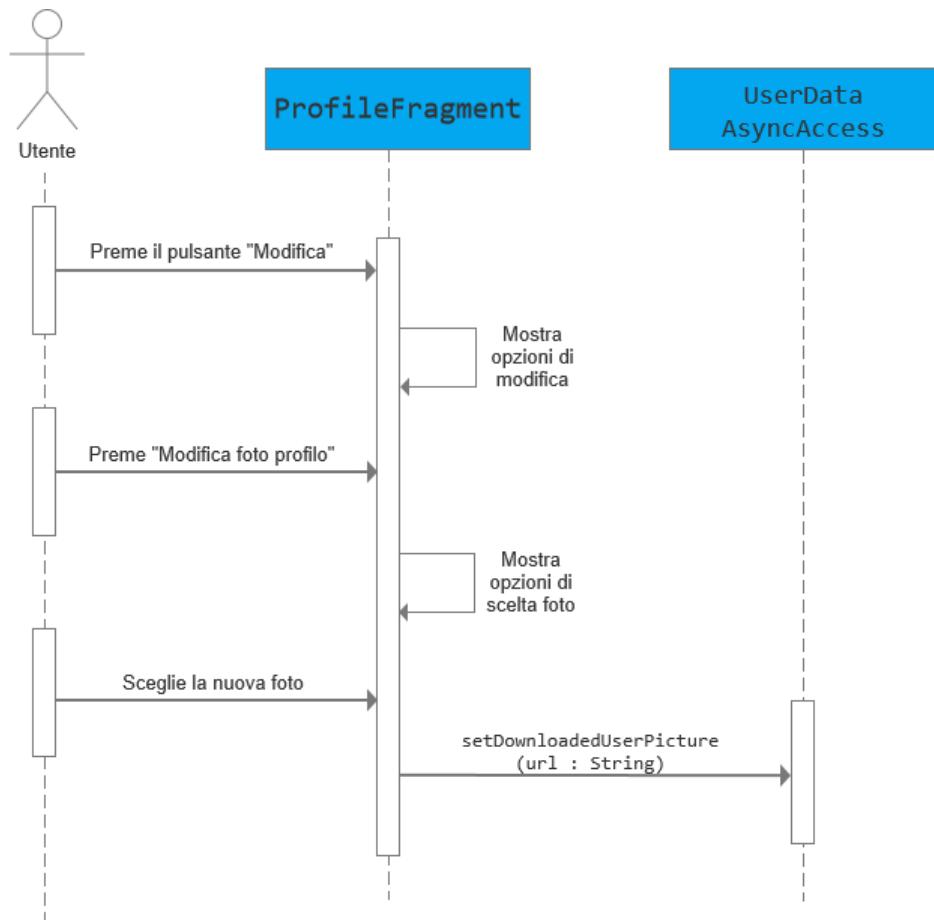
#### 7.1.1 Modifica del nome



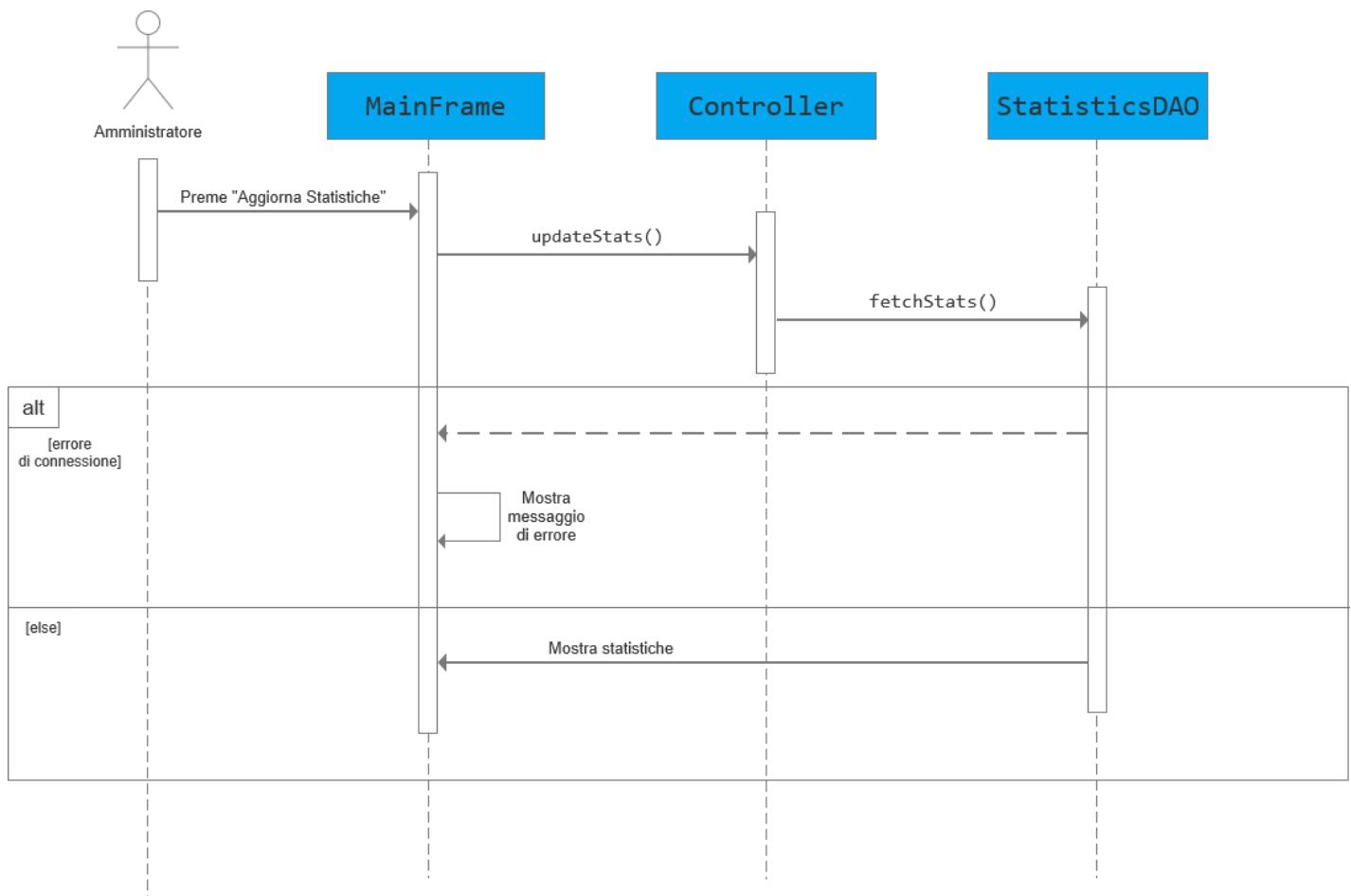
#### 7.1.1 Modifica del cognome



## 7.2 Modifica della password



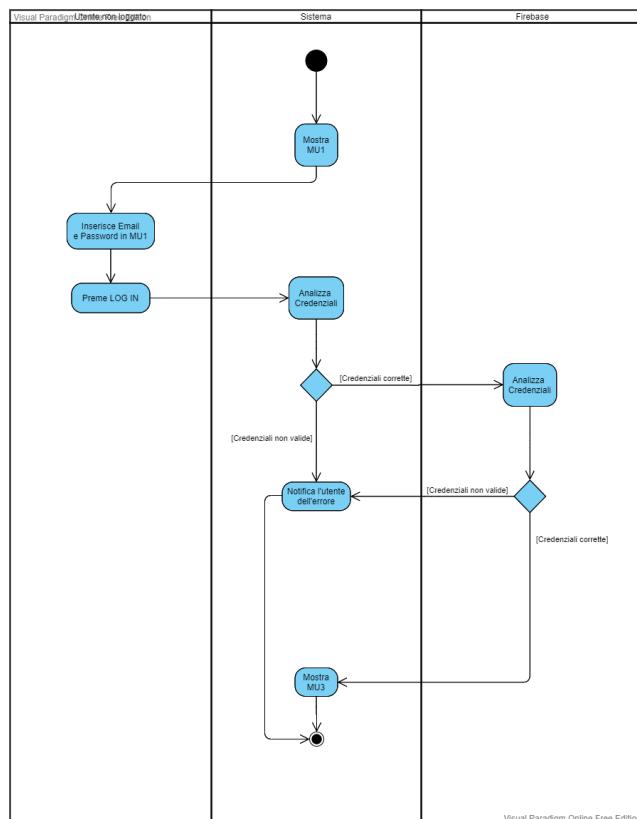
7.3 Modifica della foto profilo



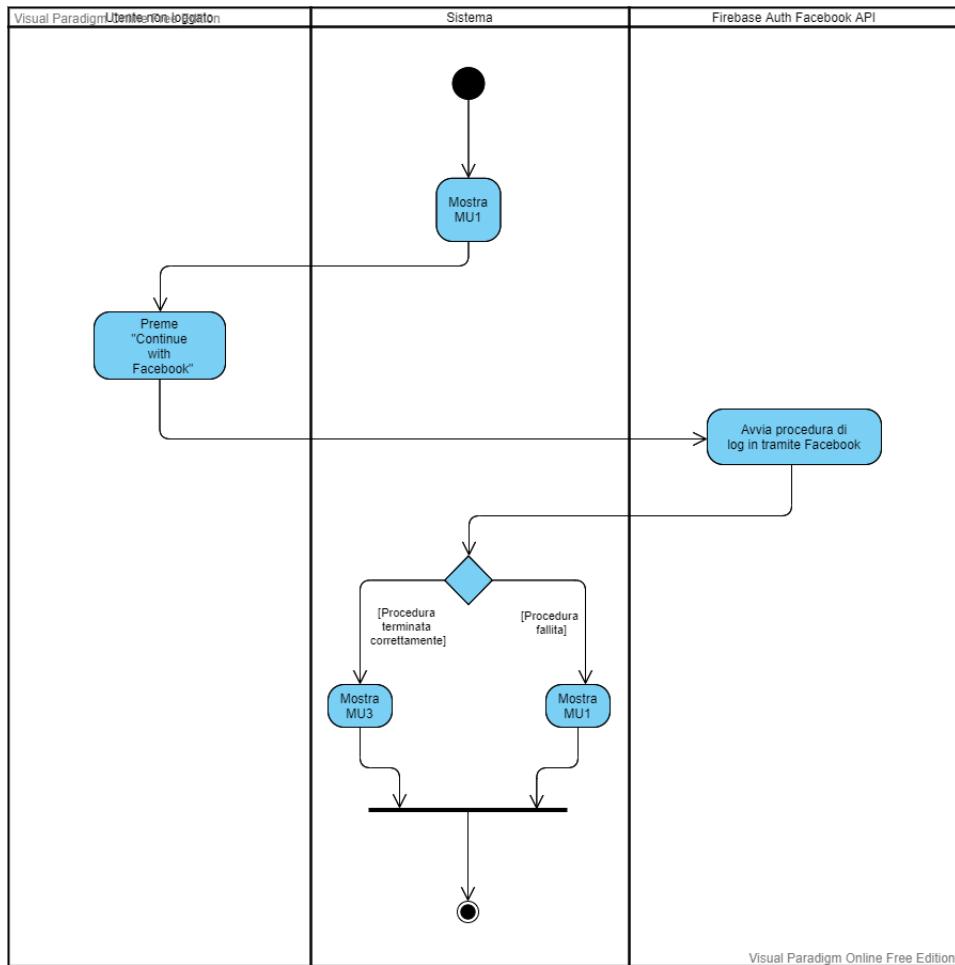
8.1 Visualizzazione delle statistiche del sistema

### 2.2.3 Activity Diagrams

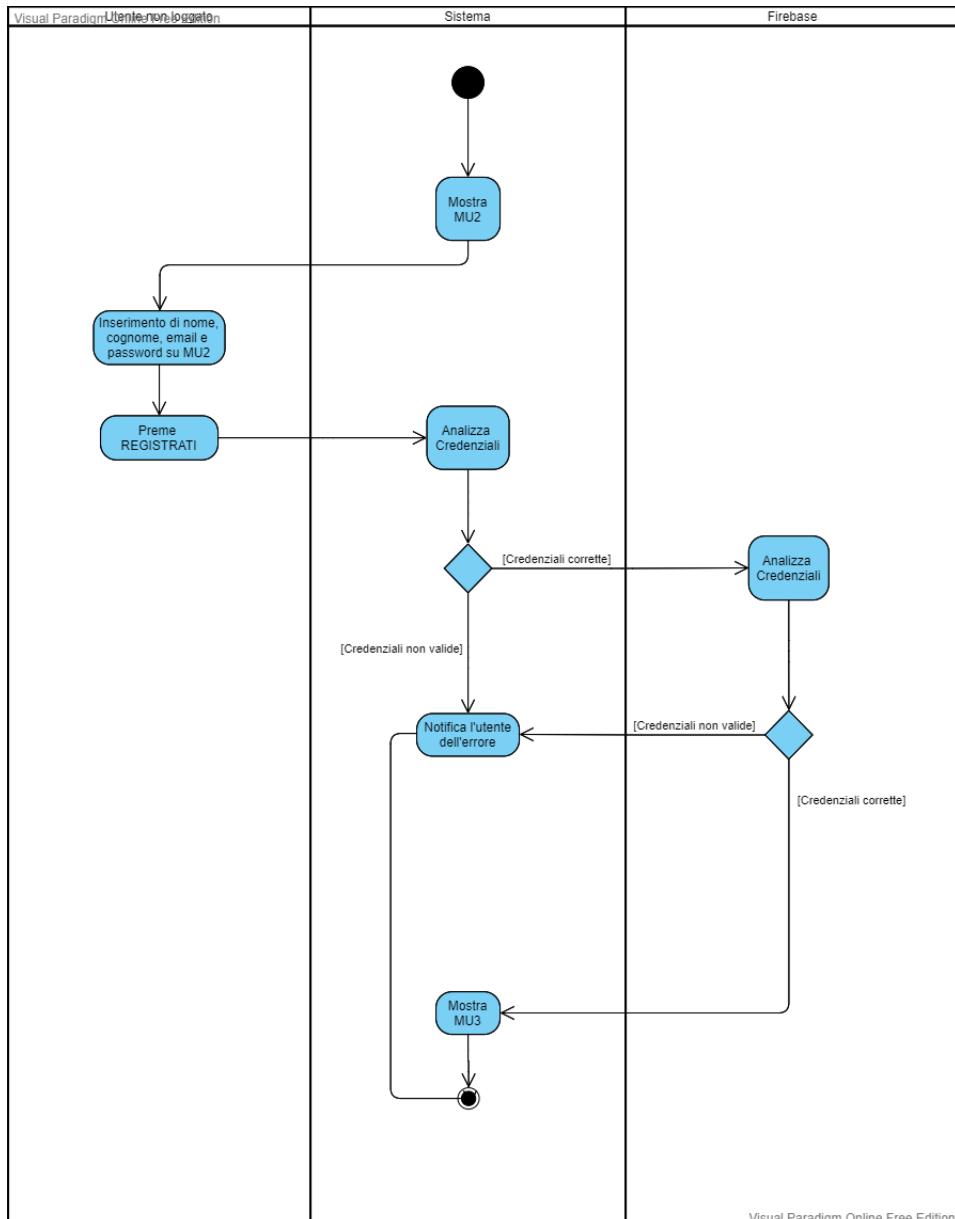
L'ultimo dei documenti UML di analisi dei requisiti è costituito da diagrammi di attività, atti a scandire con precisione il flusso di esecuzione di ciascuna funzionalità, come supporto alle tabelle di Cockburn.



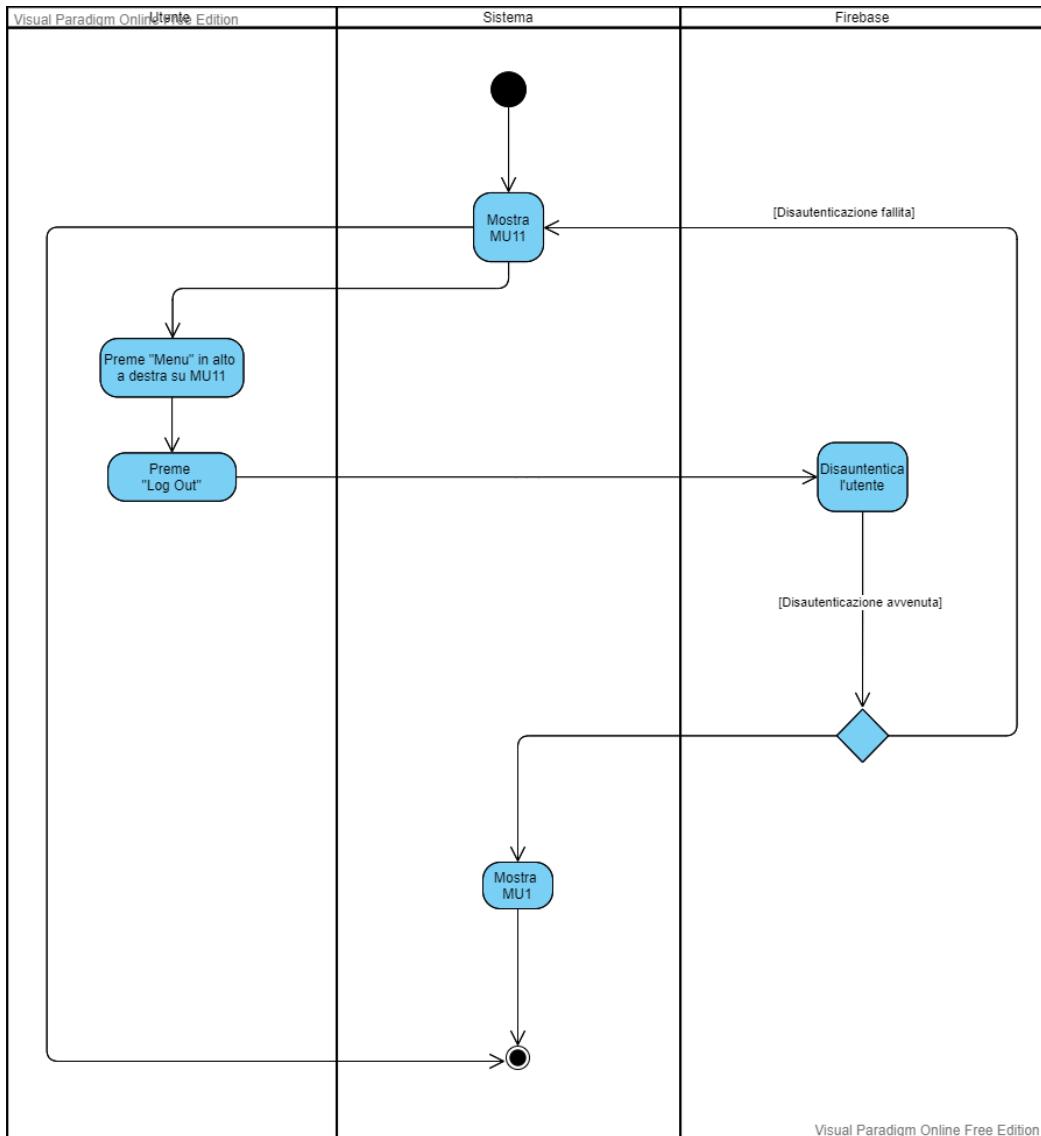
1.1 Log in



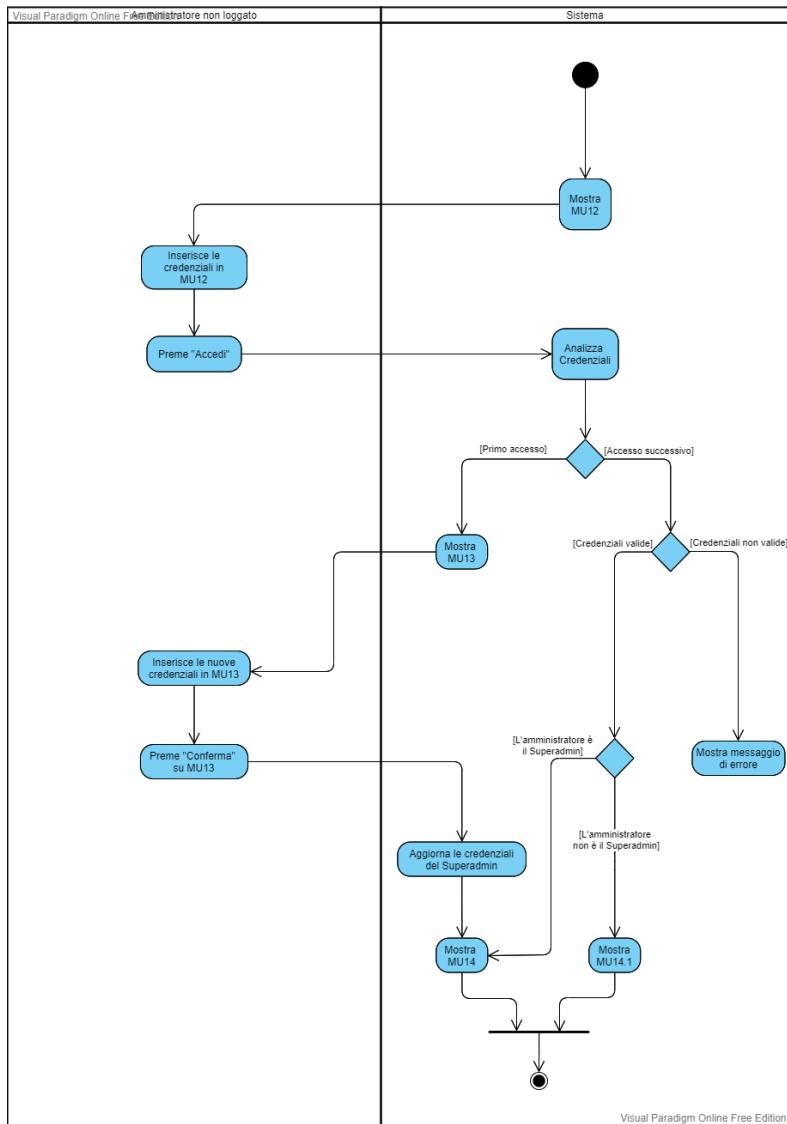
## 1.2 Log in tramite Facebook



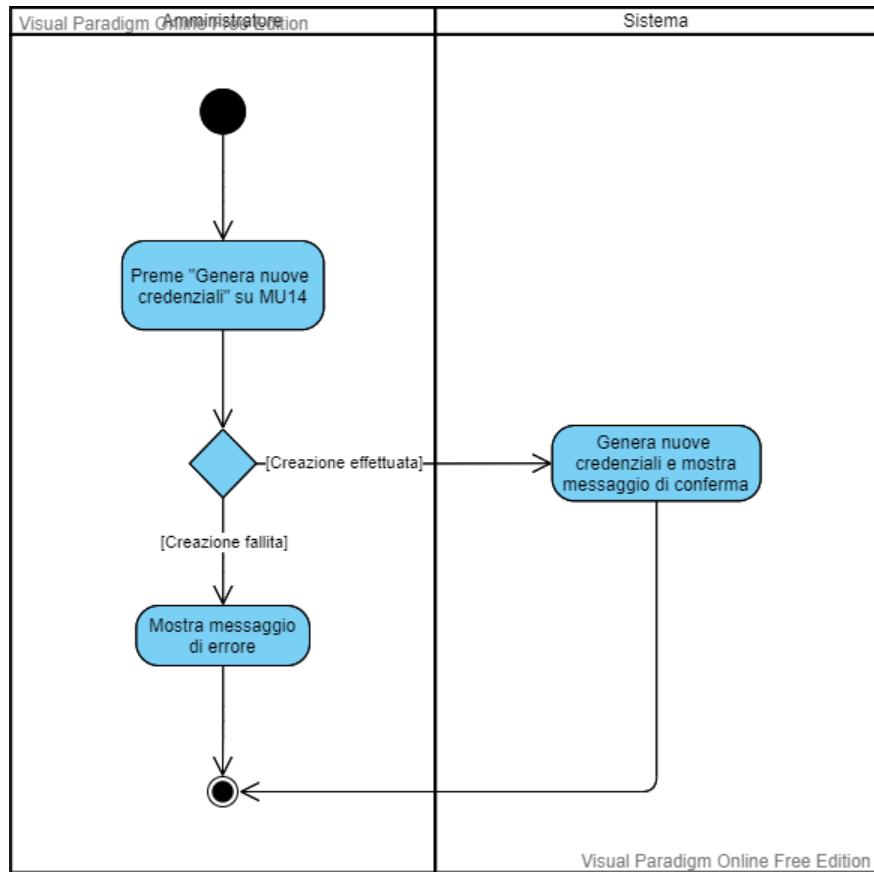
### 1.3 Registrazione di un nuovo profilo



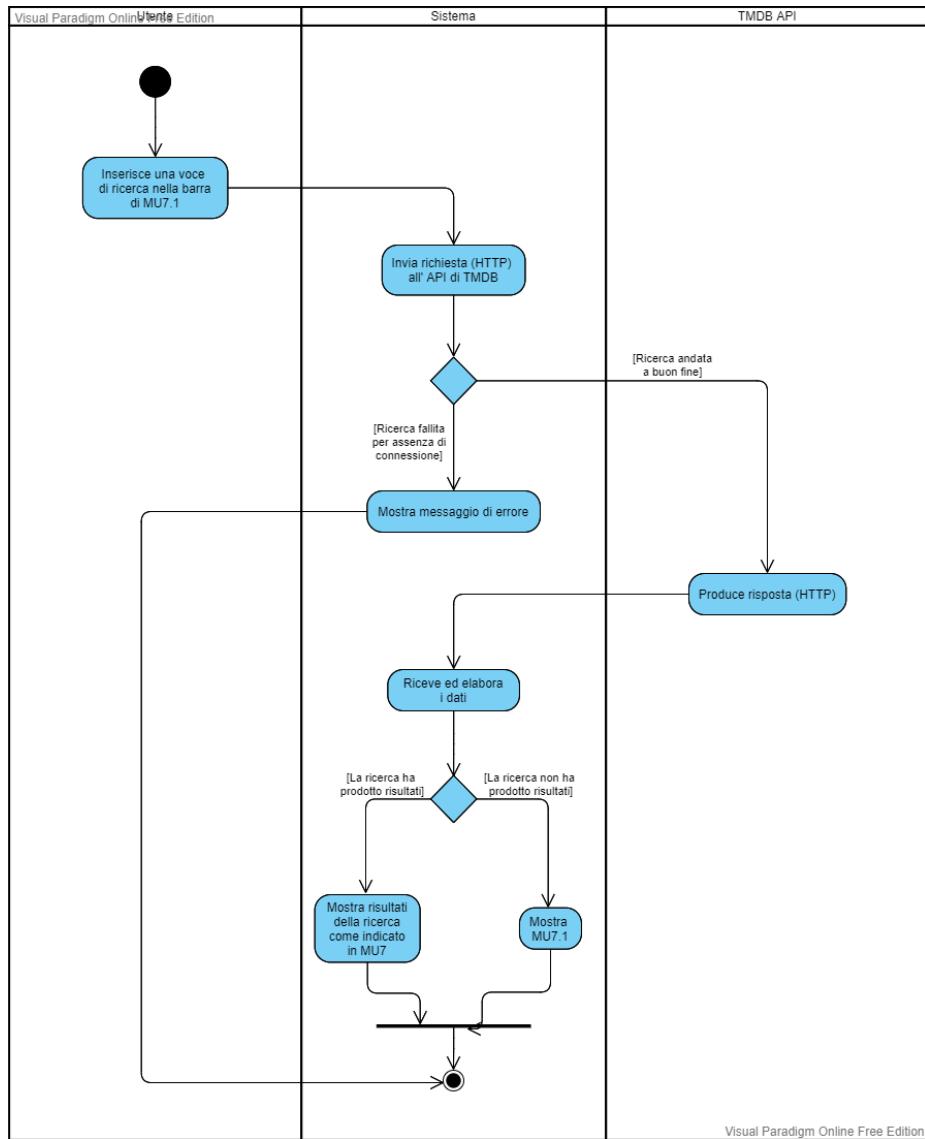
#### 1.4 Log out



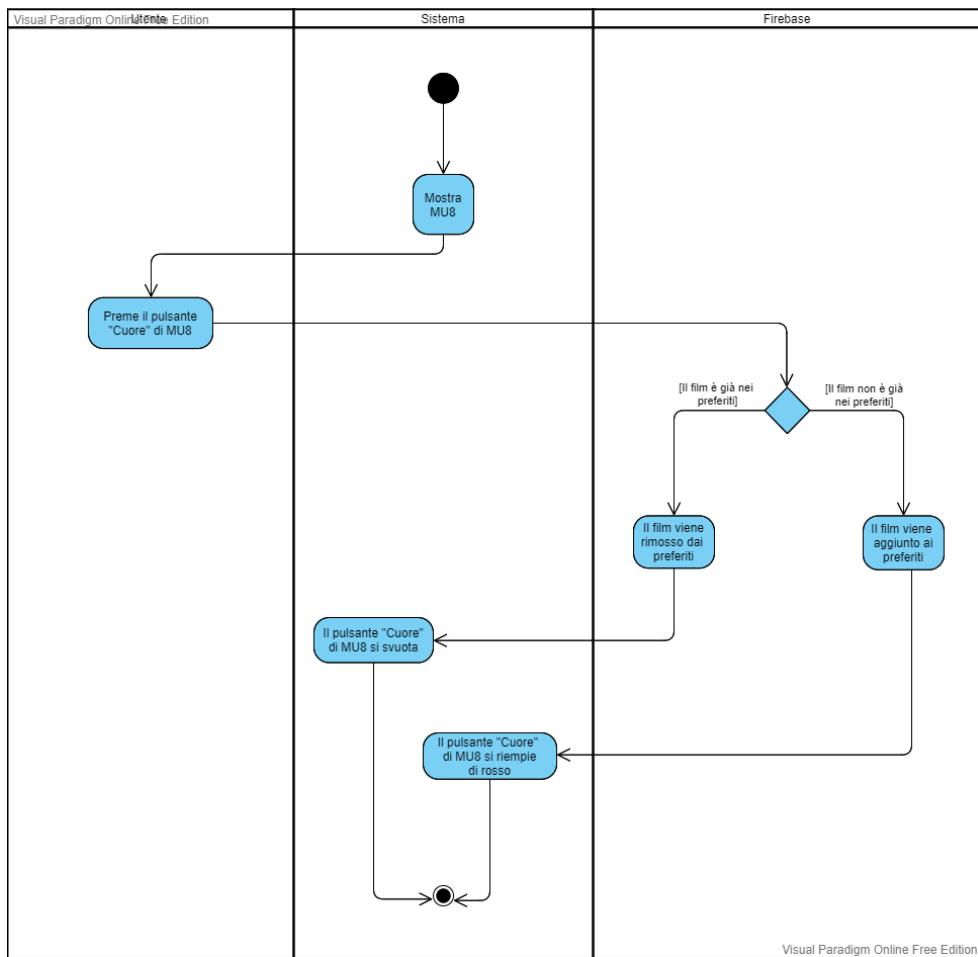
## 2.1 Log in amministratore



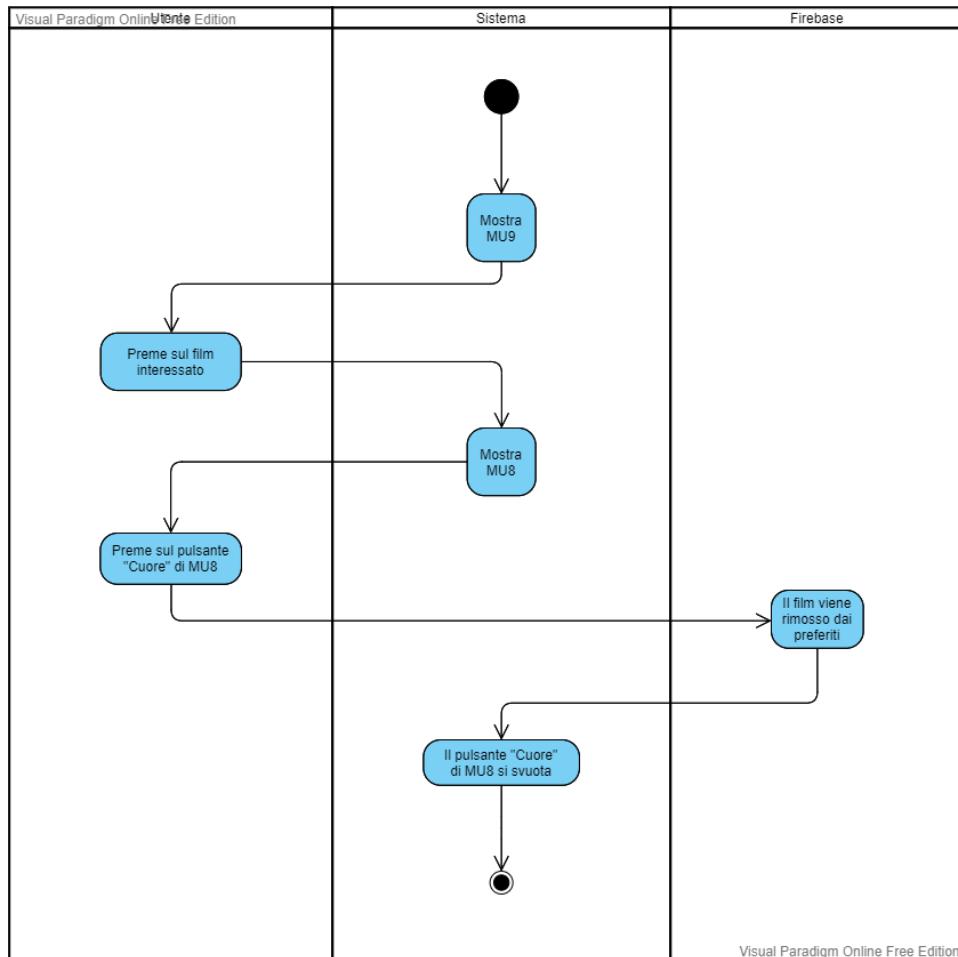
2.2 Generazione di nuove credenziali



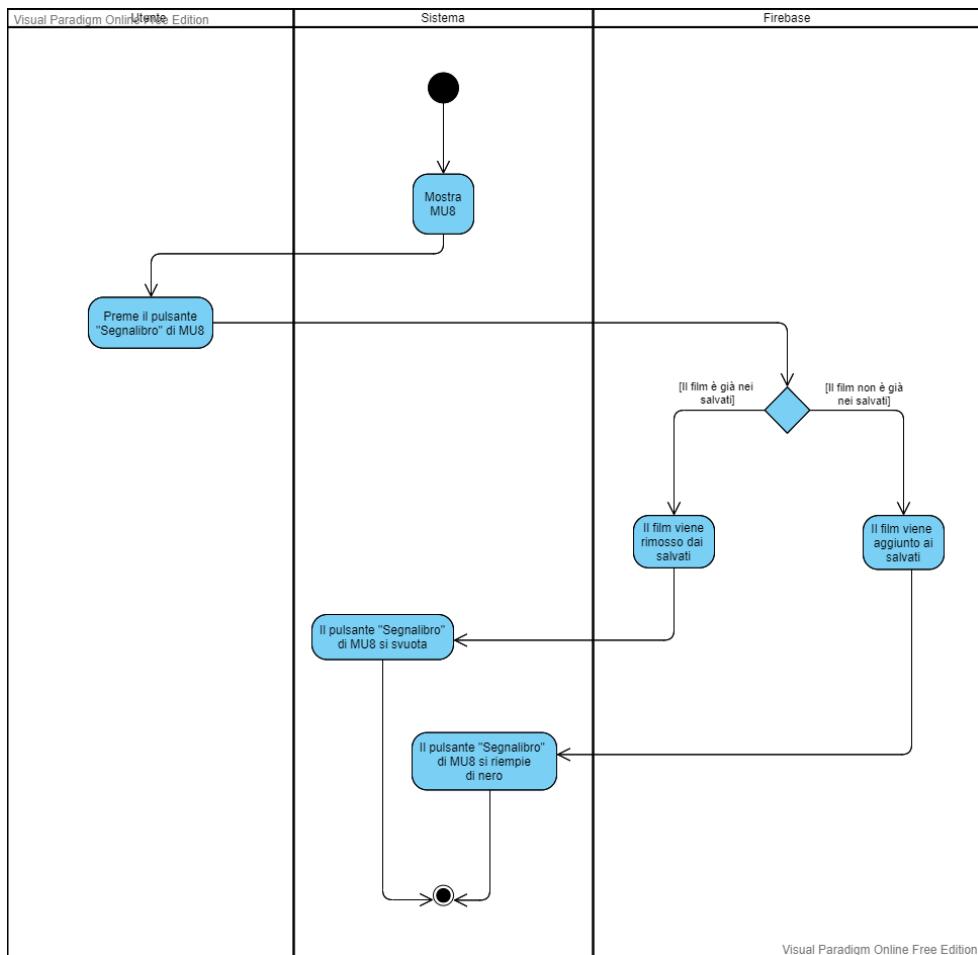
### 3.1 Ricerca di film



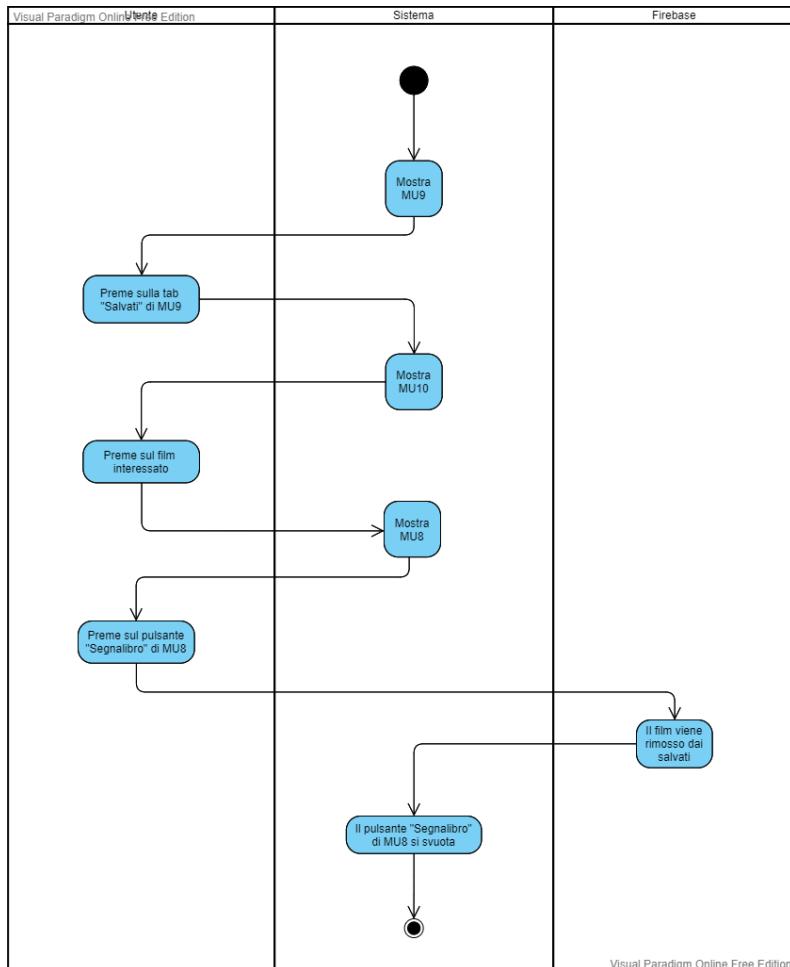
#### 4.1 Aggiungimento di un film ai preferiti



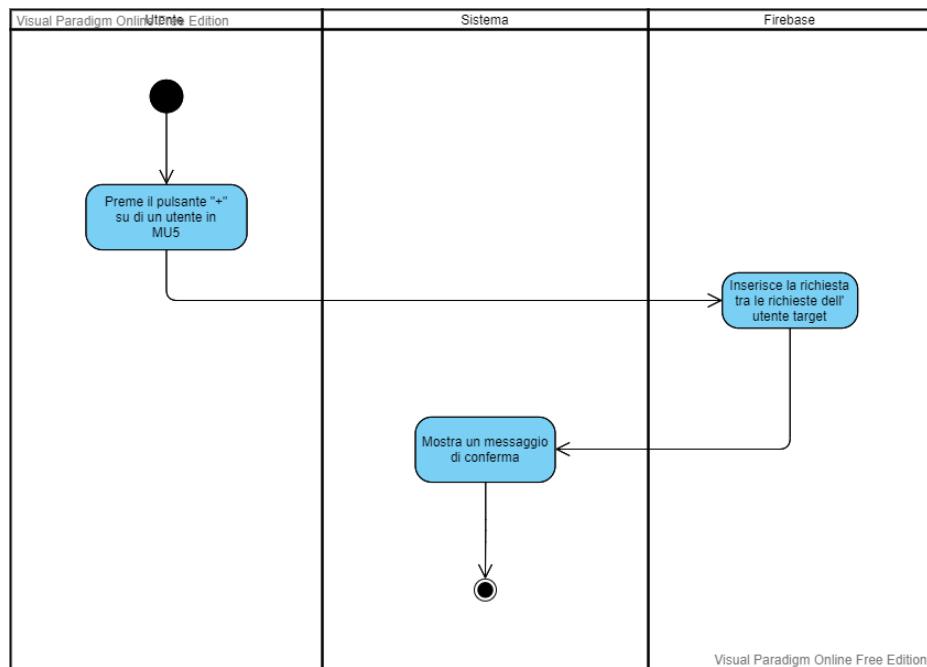
4.2 Rimozione di un film dai preferiti



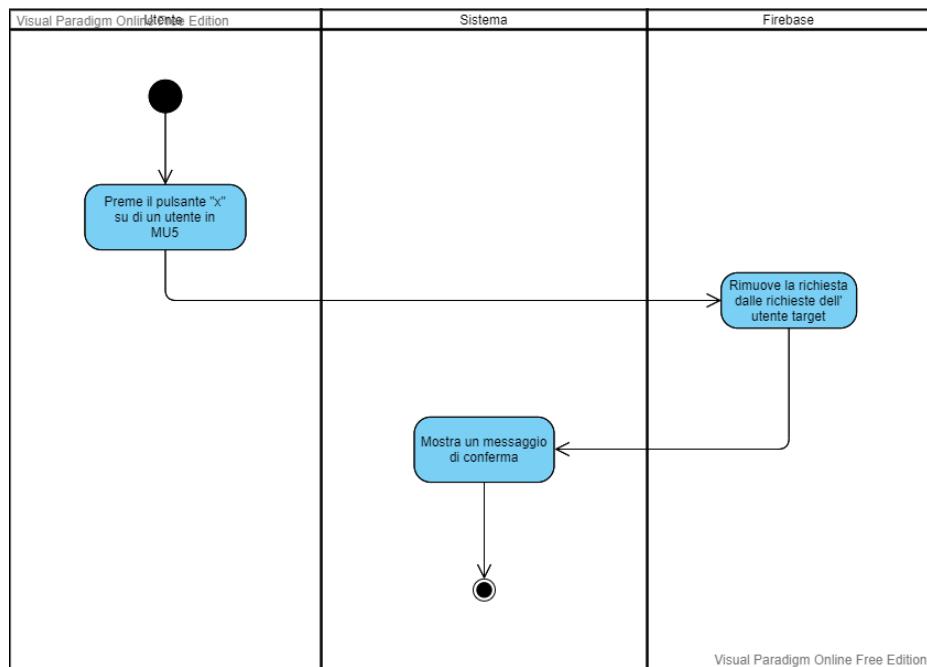
#### 4.3 Aggiungimento di un film ai salvati



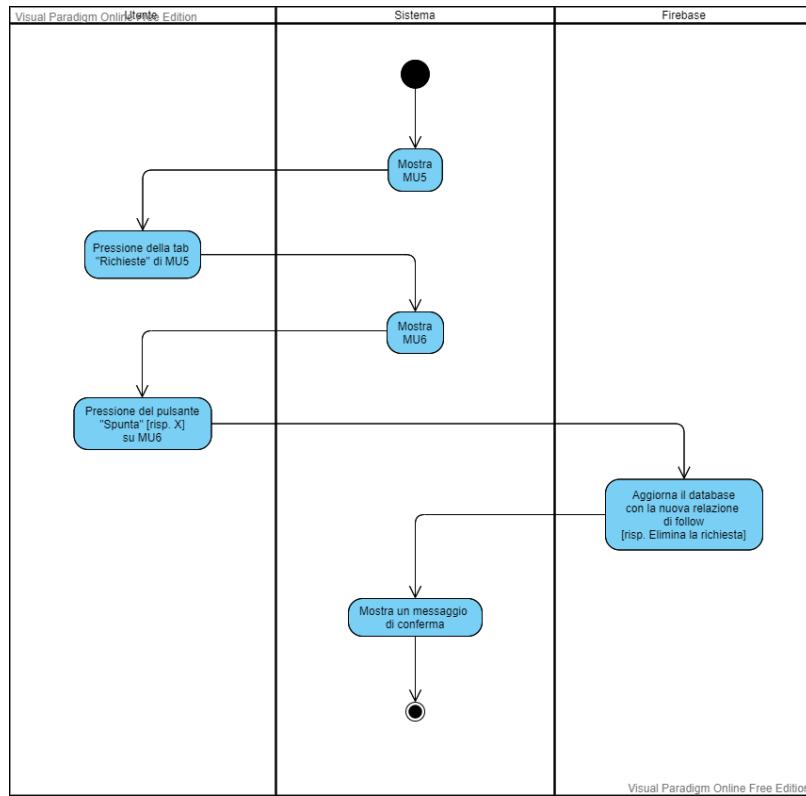
4.4 Rimozione di un film dai salvati



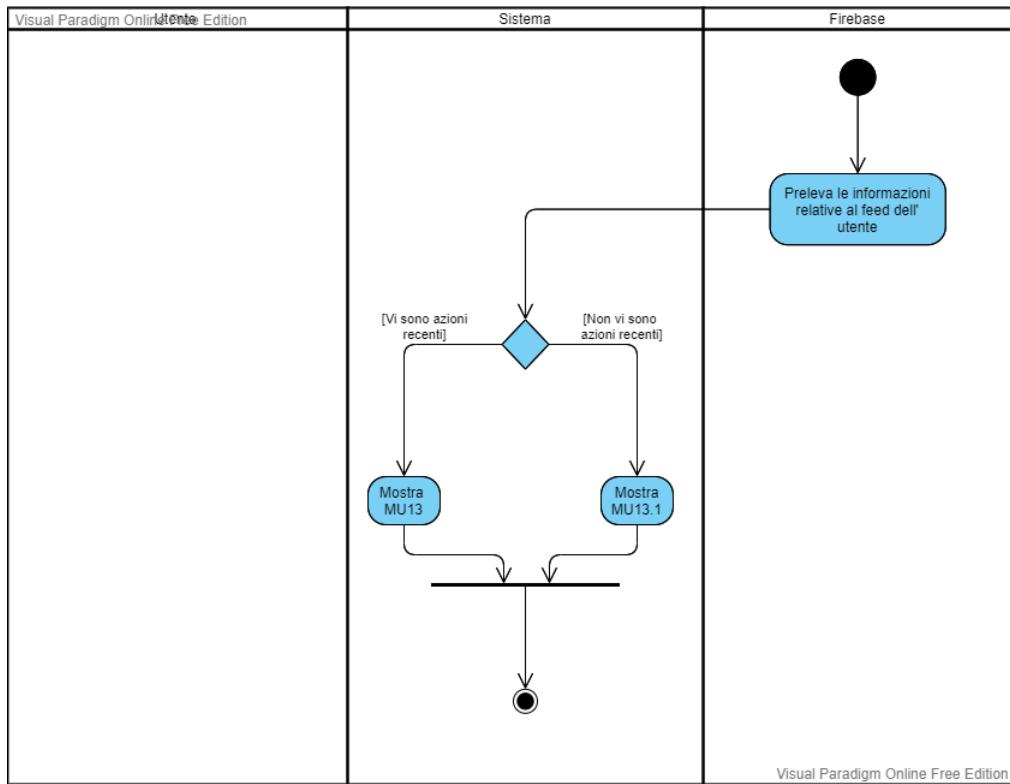
### 5.1 Invio di una richiesta di follow



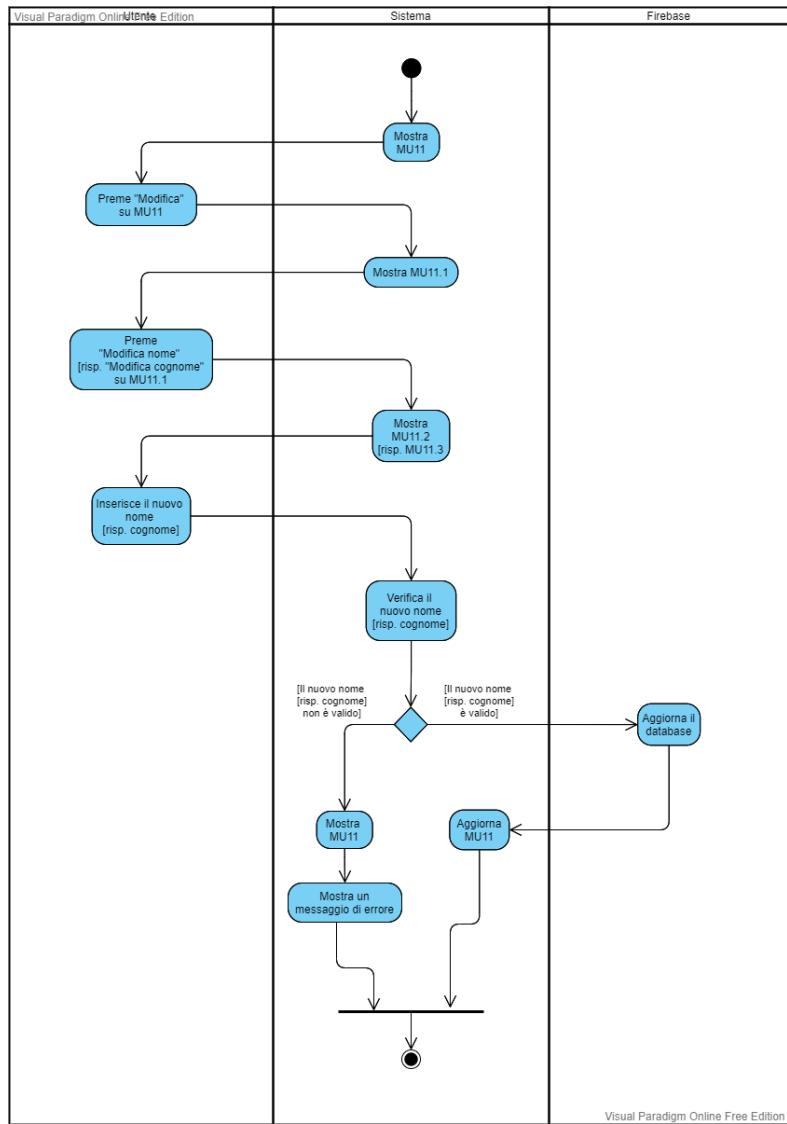
## 5.2 Annullamento di una richiesta di follow



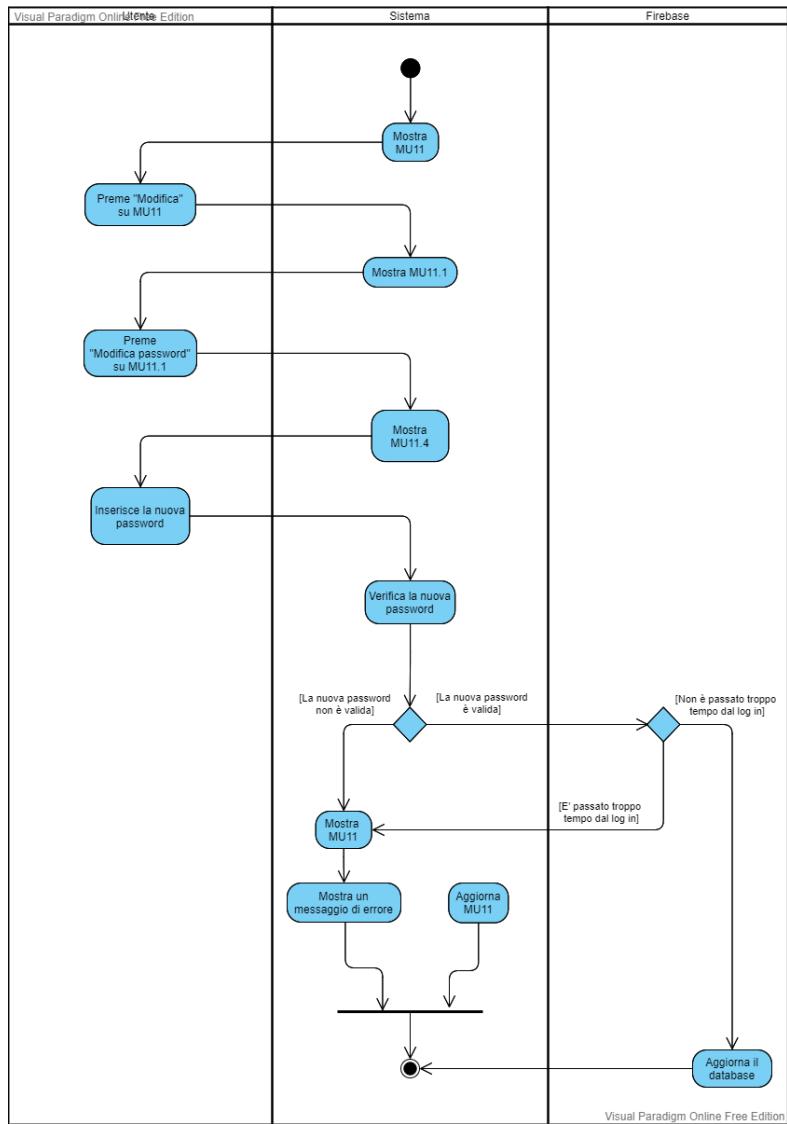
### 5.3 Accettazione/Rifiuto di una richiesta di follow



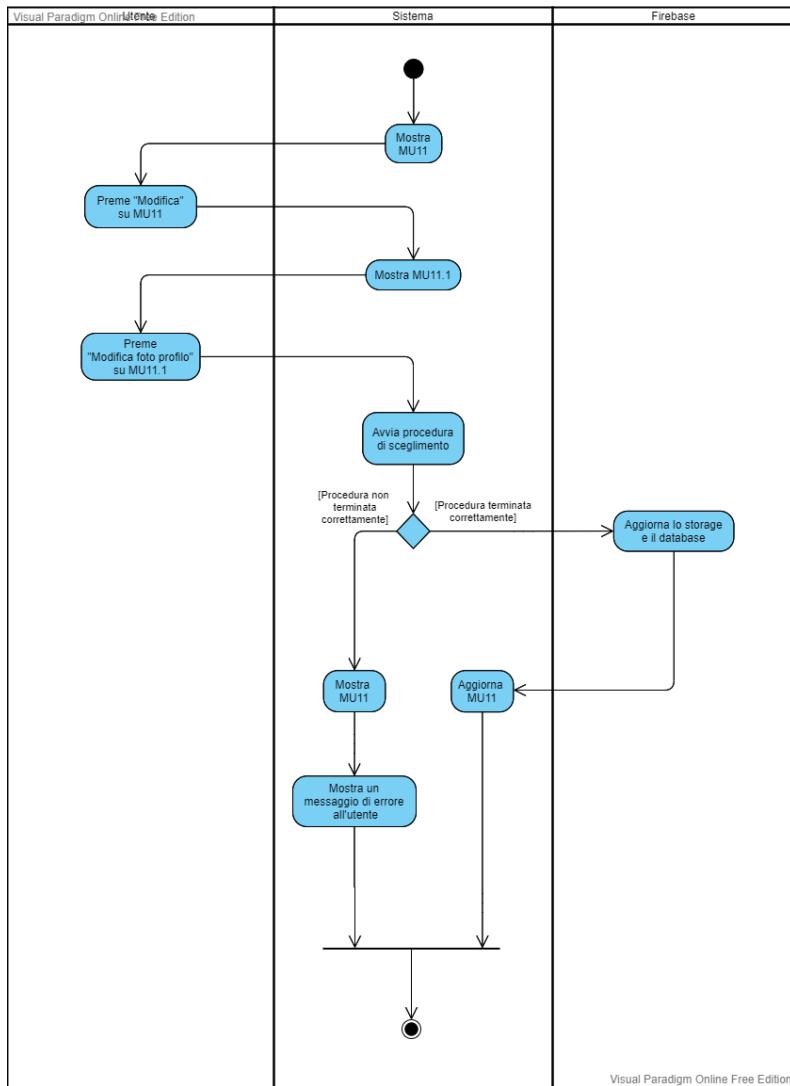
### 6.1 Visualizzazione del feed



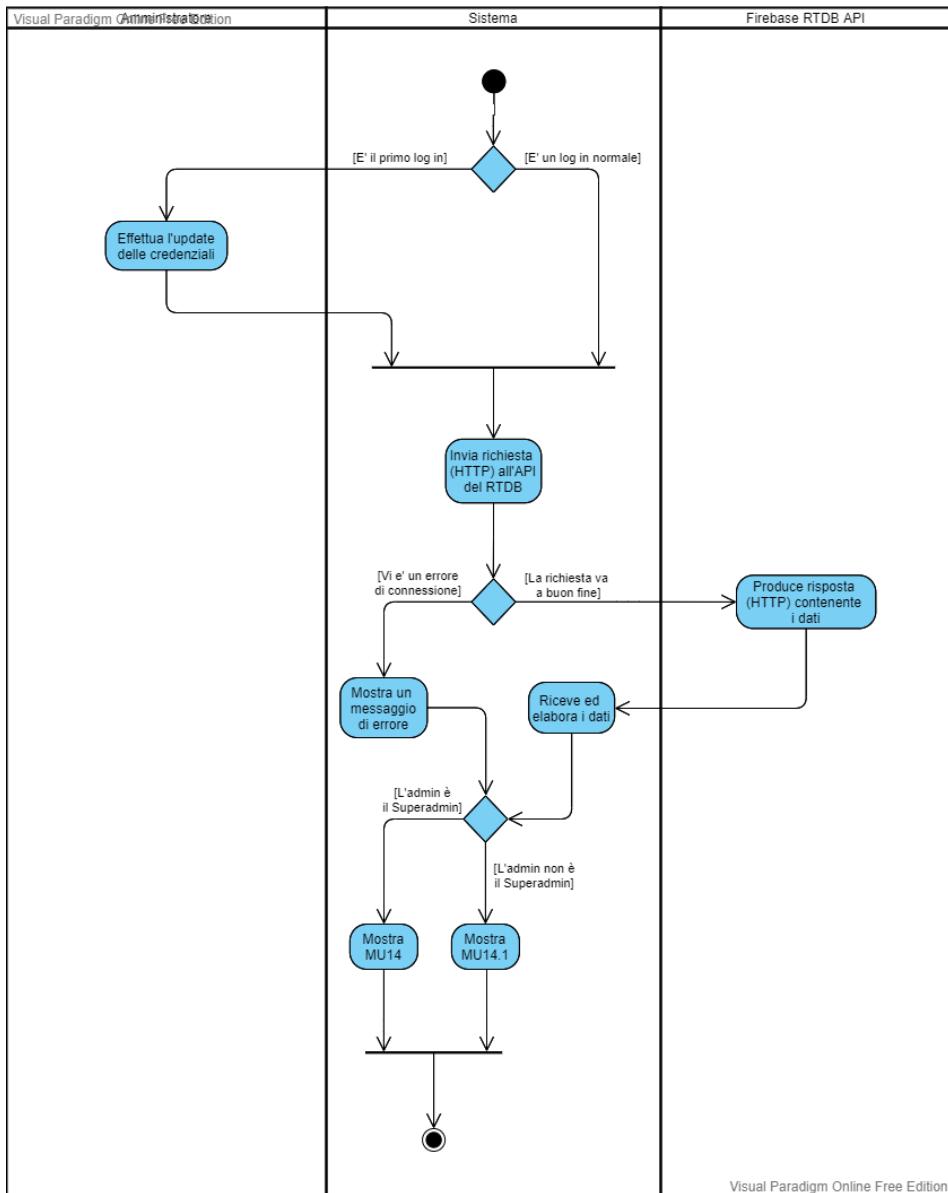
## 7.1 Modifica del nome/cognome



## 7.2 Modifica della password



### 7.3 Modifica della foto profilo



## 8.1 Visualizzazione delle statistiche del sistema

## 3 Documento di Design del sistema

### 3.1 Analisi dell'architettura del sistema

#### 3.1.1 Tecnologie Adoperate

In merito ai linguaggi di programmazione adoperati per lo sviluppo di entrambi i moduli richiesti dai committenti per il prodotto finale, la scelta è ricaduta su Java 8, linguaggio object oriented per eccellenza che ha offerto gli strumenti di progettazione più rapidi ed efficienti possibili, in quanto ottimizzato per la produzione di applicazioni Android e di client desktop semplici ed intuitivi, grazie alla libreria Swing che esso mette a disposizione.

Gli ambienti di sviluppo utilizzati sono stati Android Studio per la creazione dell'applicazione Android, e IntelliJ IDEA per la sezione desktop, entrambi prodotti JetBrains.

L'architettura del sistema vede come software di back end uniforme la piattaforma cloud Firebase offerta da Google.

Per quanto riguarda il client desktop, si è fatto uso degli Amazon Web Services per attivare una macchina virtuale sul quale posizionarlo, in modo da rendere l'accesso per gli amministratori il piu' versatile possibile.

Di seguito si esplicita, per ciascuna macrofunzionalità, lo specifico modulo software che si è utilizzato per implementarla.

#### 1. Autenticazione

##### (a) *Autenticazione degli utenti*

Per la gestione dell'autenticazione degli utenti di CineMates20, si è fatto uso di Firebase Auth, il modulo di Firebase adibito appunto all'autenticazione. Vincoli di unicità dei profili utenti e di validità delle loro credenziali sono quasi interamente gestiti tramite questo software, e grazie ad esso risulta di immediata implementazione un metodo per tenere attiva la sessione di un utente fino a che quest'ultimo non decide esplicitamente di terminarla.

(b) *Autenticazione degli amministratori*

Gli amministratori possono accedere al client desktop di visualizzazione delle statistiche del sistema sulla macchina virtuale AWS tramite un nome ed una password. Delle combinazioni nome-password valide si tiene traccia sulla macchina che ospita il client.

## 2. **Dati degli utenti**

Dei dati degli utenti si tiene traccia in maniera persistente tramite Firebase Firestore, il database No-SQL in tempo reale messo a disposizione dal cloud.

L'utilizzo di Firestore rende immediata la possibilità di adattarsi ad un progressivo aumento del numero di utenti: grazie alla scalabilità automatica, infatti, le risorse sufficienti vengono prese e cedute a seconda della necessità. Punto critico della questione della scalabilità è il fatto che quest'ultima avviene in orizzontale: Firestore aumenta dinamicamente il numero di macchine che supportano il back-end dell'applicazione quando necessario, contrariamente ad una politica di scalabilità verticale, che consisterebbe nell'aggiungere potenza di calcolo ad una singola macchina nel momento in cui è richiesto maggiore sforzo hardware.

L'adattamento orizzontale presenta come vantaggio principale il fatto che il back-end risulta omogeneamente distribuito a livello geografico, garantendo agli utenti di tutto il mondo un'esperienza con latenza minimizzata.

E' inoltre importante specificare che, per quanto riguarda il lato economico, Firebase Firestore adotta una politica "pay as you go", a differenza di una spesa mensile (o annuale) fissa. Il costo è direttamente proporzionale alle letture e le scritture effettuate, e fa sì che i costi si mantengano contenuti anche per migliaia di utenti, garantendo a chi lo usa di non spendere per servizi di cui non sta usufruendo.

## 3. **Ricerca di Film**

Per questo modulo ci si è appoggiati alla REST API di The Movie DataBase (TMDB). I vantaggi scaturiti dall'utilizzo di un servizio esterno per l'implementazione di questa macro-funzionalità sono molteplici. Il vincolo di tenere traccia dell'enorme quantità di film che l'utente ha a disposizione per la fruizione viene completamente eliminato e il piano gratuito messo a disposizione da TMDB consente un'implementazione libera da spese aggiuntive.

4. **Statistiche** Per poter visionare le statistiche del sistema (inteso come applicazione mobile) si è sfruttato il Firebase RealTime Database. Quest'ultimo si distingue da Firestore in quanto, mentre quest'ultimo è basato su documenti e collezioni di documenti, esso conserva i dati in formato JSON. Chiaramente, anche per il RTDB valgono i medesimi vantaggi evidenziati per Firestore.

Il motivo per cui si è optato per questo modulo per assolvere questo compito è che l'intero documento JSON è fruibile tramite REST API: in questo modo, il client desktop non deve far altro che inviare una richiesta HTTP alla suddetta API per ricevere tutti i dati da visualizzare in tempo reale, a differenza di un approccio convenzionale basato su un sistema di Analytics, che non agevola il trasferimento dei dati in tempo reale fra piattaforme.

### 3.1.2 Criteri di design

#### 1. Applicazione mobile

L'applicazione mobile ha una struttura interamente asincrona, in accordo con le modalità di accesso al database che Firebase mette a disposizione per il modulo Firestore.

E' proprio l'asincronia il punto forte del design dell'applicazione: le funzioni di accesso ai dati terminano immediatamente, senza avere nessun impatto sul thread principale dell'applicazione, che ne rimane isolato, per un notevole guadagno di performance.

In linea con questa politica di accesso asincrona, l'applicazione si basa su una variante della tipica architettura 3-tier. Entriamo dunque nel dettaglio di ciascun layer:

##### I Data

Elemento portante del data layer è l'oggetto Task. L'accesso ai dati degli utenti è mediato da una serie di interfacce che contengono esclusivamente metodi che restituiscono Task, una rappresentazione sottoforma di oggetto Java dell'operazione che si sta andando ad effettuare.

Risulta chiaro che i metodi in questione non potrebbero restituire direttamente il dato richiesto: poichè essi terminano immediatamente, come specificato, per garantire performance ottimali a livello di interfaccia utente, al momento della loro terminazione il dato non e' ancora stato recuperato. E' necessario dunque

un tipo di ritorno intermedio che rappresenti l'operazione.

Queste interfacce vengono implementate da classi (per ciascuna delle quali si è fatto uso del design pattern Singleton) finalizzate al recuperare i dati da Firestore, analoghe dei Data Access Object in un DAO pattern convenzionale. I tipi di ritorno di ciascuna funzione nelle classi che implementano le interfacce di accesso sono ora parametrizzati in base al tipo di dato che si sta andando a recuperare.

L'oggetto Task è dunque il mezzo tramite il quale l'applicazione Android, scritta interamente in Java, può comunicare in maniera asincrona con Firebase Firestore.

E' importante notare che l'accesso alle statistiche è considerato come un'operazione indipendente dal data layer: esse sono semplicemente aggiornate da una classe di servizio contenente esclusivamente metodi statici, in quanto sono state volutamente separate dai dati degli utenti.

Questa decisione non va chiaramente ad intaccare la modularità del codice, in quanto il passaggio ad un sistema di raccoglimento delle statistiche diverso non comporta modifiche a nessun' altra sezione.

Analogamente all'accesso alle statistiche, anche il reperimento dei dati dei film segue lo stesso ragionamento: come si menzionerà anche nella seguente analisi dell'Application Layer, i suddetti dati risultano separati dal Data Layer, che fa riferimento, come specificato, a Firebase Firestore, poichè queste informazioni provengono interamente da fornitori esterni.

## II *Application*

L'application layer svolge il ruolo fondamentale di intermediario tra i dati e l' interfaccia utente.

Vista la necessità dei metodi del data layer di restituire oggetti di tipo Task, le classi di controllo avranno il compito di reagire nel momento in cui le operazioni da essi rappresentate vengono effettivamente complete, facendo uso intensivo a tale proposito di OnSuccessListener opportunamente parametrizzati.

Le suddette classi hanno dunque l'onere di evitare la presenza di qualsivoglia manipolazione di Task all'interno delle classi di interfaccia, gestendo qualunque aggiornamento di quest'ultima derivante da un qualche cambiamento nei dati.

Quest'ultimo ruolo è svolto in primo luogo dalle classi `Controller` e `ListenerDispenser`. In particolare, la seconda ha il compito di attaccare e staccare i Listener di Firebase per garantire aggiornamenti in tempo reale dell'interfaccia utente, dove necessari.

Siccome l'applicazione si appoggia all'API esterna di The Movie DataBase (TMDB) per recuperare le informazioni concernenti i film, è risultato opportuno creare una classe di controllo ad hoc per manipolarle. Sarebbe stato infatti ambiguo trattare questi tipi di dati allo stesso modo di quelli provenienti da un proprio database, come già specificato, in quanto completamente distinti.

Sempre parte del layer di controllo è la classe di gestione dell'autenticazione degli utenti, che si ricorda essere implementata tramite Firebase Auth. Risulta naturale anche per essa, infatti, la possibilità di aggiornare l'interfaccia utente dinamicamente in base all'attuale stato di autenticazione dell'utente, manipolando le Task relative ad esso. Anche qui, come nel caso dell'accesso alle statistiche e delle informazioni sui film, si manifesta la distinzione tra gli utenti stessi, conservati all'interno di Firebase Firestore con tutte le informazioni ad essi relative, e i loro profili, cioè le combinazioni email-password utilizzate per l'accesso.

Tutte le classi di controllo sono state pensate e progettate come classi di servizio contenti solo metodi statici. Il motivo di questa decisione è semplice: poichè nessuna di queste classi rappresenta un oggetto, essendo il loro unico scopo quello di veicolare la comunicazione tra dati ed interfaccia utente, l'istanziazione di una qualunque di esse risulterebbe concettualmente errata.

Si precisa inoltre che sebbene sarebbe stato possibile condensare l'intera attività di controllo in un'unica classe di dimensione relativamente contenuta, viste le dimensioni iniziali dell'applicativo, si è preferito scindere il tutto in quattro classi per tenere fede il più possibile al Single Responsibility Principle.

### III *Presentation*

Il layer di presentazione dei dati, che corrisponde, com'è ovvio, all'interfaccia utente, si basa esclusivamente su Fragment e Activity, i fondamenti della UX in Android. La struttura principale dell'interfaccia, in particolare, consta di una barra di navigazione inferiore che l'utente utilizza per muoversi all'interno dell'applicativo, rendendo l'utilizzo intuitivo e immediato.

Di appoggio a questi 3 layer, una serie di classi di modello rappresentano nel codice sorgente gli oggetti principali del dominio, come gli i film e gli utenti, ma non partecipano attivamente alla definizione della sua architettura.

## 2. Client desktop di amministrazione

Per il lato client del prodotto, le scelte implementative sono state pressocchè standard. L'elevata semplicità del sistema, che deve mettere a disposizione di un ipotetico amministratore delle funzionalità di accesso e di visualizzazione delle statistiche dell'applicazione mobile, ha consentito di mettere a punto un convenzionale pattern 3-tier, che possiamo analizzare allo stesso modo della variante utilizzata per l'applicazione mobile:

### I Data

Qui, i nostri dati consistono semplicemente nelle combinazioni nome-password degli amministratori. Il client è stato pensato in maniera tale da non far sussistere alcuna differenza tra i profili degli amministratori, tranne per quanto riguarda il Superadmin, che può generare nuove credenziali di accesso. E' per questo motivo che non vi e' necessita' di classi di modello rappresentati gli amministratori, sebbene nello sviluppo del software non si limita in alcun modo questa possibilità per sviluppi futuri.

Il data layer è dunque rappresentato da due DAO, una per quanto riguarda i dati degli admin, provenienti in questa implementazione da un file sulla macchina ospite, e una per le statistiche, alle quali si accede tramite richieste HTTP all'API del Firebase RealTime Database, come specificato in precedenza.

Entrambe le DAO sono interfacce, implementate da due classi che si occupano dell'effettivo recupero dei dati.

### II Application

Il layer di controllo è in questo caso molto semplice: esso consiste in un gestore dell'autenticazione degli admin, che comunica direttamente con il relativo punto di accesso ai dati ed effettua le operazioni legate alle credenziali degli amministratori, e da un Controller che si occupa degli aggiornamenti dell'interfaccia grafica derivanti dai dati, primo tra tutti l'aggiornamento delle statistiche.

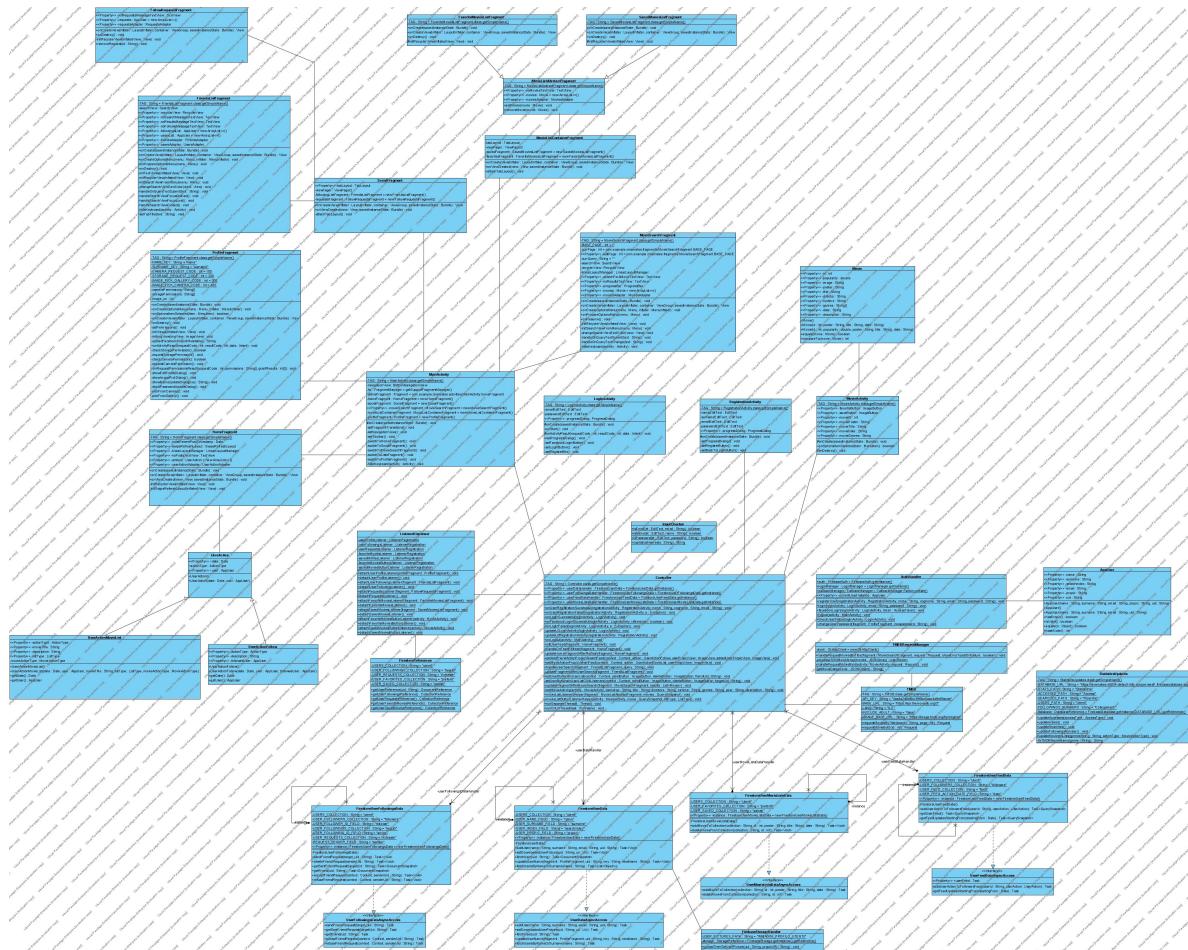
### III Presentation

Per quanto riguarda il presentation layer, si è fatto uso di semplici JFrame per le schermate di Log In, di modifica delle credenziali per il primo accesso e di visualizzazione delle statistiche, queste ultime aggiornate dalle operazioni del Controller che comunica con il Data Access Object incaricato del loro recupero.

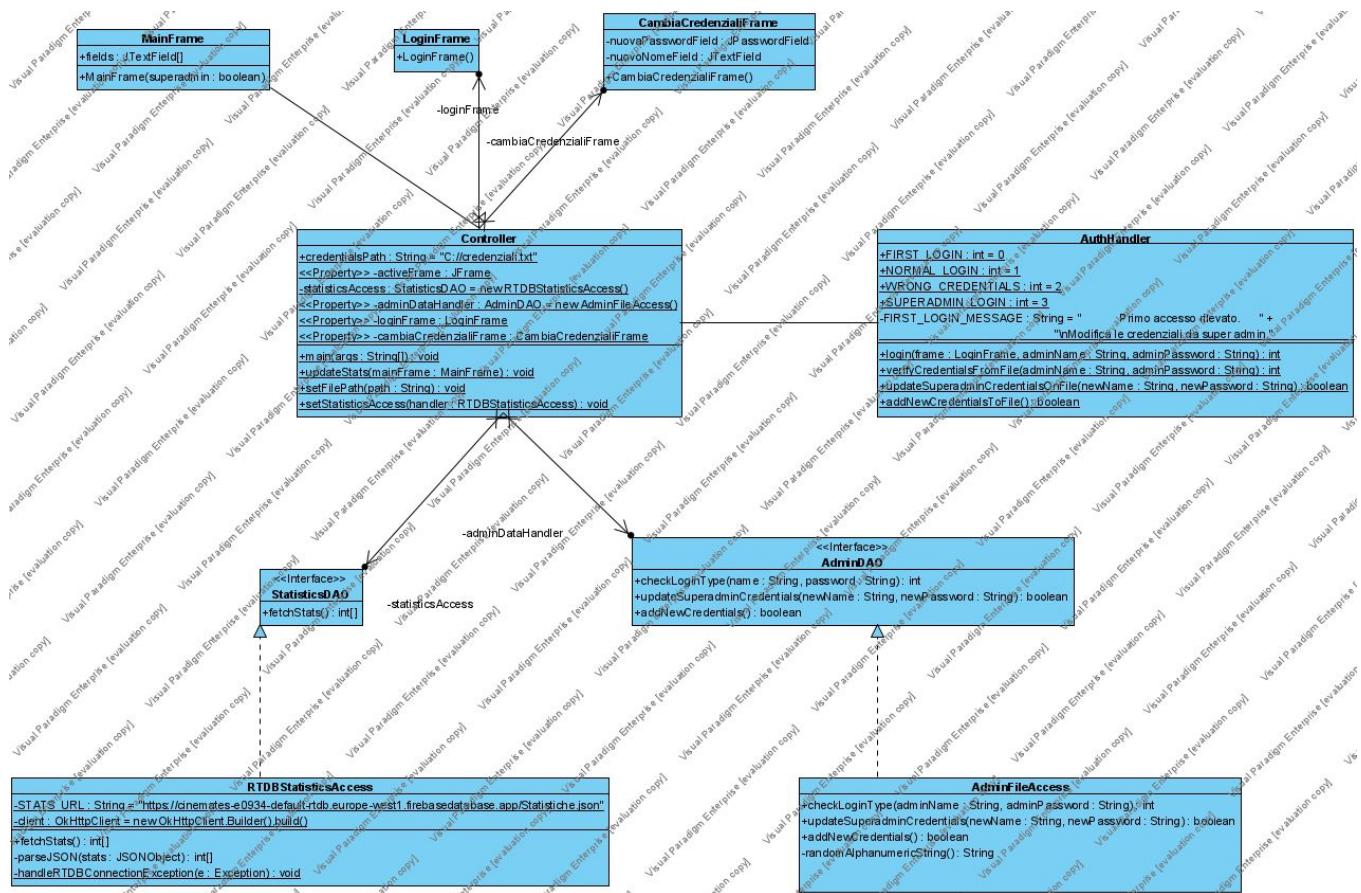


### 3.2 Diagramma delle classi di design

Seguono i class diagram integrali per l'applicazione mobile ed il client desktop.



## Applicazione Mobile



## Client Desktop

### 3.3 CRC Card per le classi non di interfaccia utente

Per ciascuna classe, al di fuori di quelle costituenti le interfacce grafiche dei due moduli del software, è stata compilata una Class-Responsibility-Collaborator card.

Ogni card presenta nella sezione Collaborator la classe (o le classi) con le quali interagisce, senza tenere conto, tuttavia, di eventuali legami dettati dalla struttura stratificata del progetto. Classi incaricate dell'accesso asincrono ai dati nell'applicazione mobile, ad esempio, non presentano collaboratori.

#### Applicazione Mobile - Application Layer

Controller	
Superclassi	-
Sottoclassi	-
Implementa	-
Responsabilità	Collaboratori
<ul style="list-style-type: none"><li>• Gestisce, insieme a ListenerDispenser e TMDBRequestManager, gli aggiornamenti dell'interfaccia utente derivanti dai dati</li><li>• Contiene metodi necessari alla gestione di thread</li></ul>	<ul style="list-style-type: none"><li>• FirestoreUserData</li><li>• FirestoreUserFollowingsData</li><li>• FirestoreUserFeedData</li><li>• FirestoreUserMovieListsData</li></ul>

ListenerDispenser	
Superclassi	-
Sottoclassi	-
Implementa	-
Responsabilità	Collaboratori
<ul style="list-style-type: none"> <li>• Attacca e stacca i Data Listener di Firebase Firestore</li> <li>• Svolge, insieme al Controller e a TMDBRequestManager, aggiornamenti di interfaccia grafica minori</li> </ul>	<ul style="list-style-type: none"> <li>• FirestoreReferences</li> </ul>

AuthHandler	
Superclassi	-
Sottoclassi	-
Implementa	-
Responsabilità	Collaboratori
<ul style="list-style-type: none"> <li>• Gestisce le operazioni relative all'autenticazione dell'utente</li> </ul>	<ul style="list-style-type: none"> <li>• AppUser</li> </ul>

TMDBRequestManager	
Superclassi	-
Sottoclassi	-
Implementa	-
Responsabilità	Collaboratori
<ul style="list-style-type: none"> <li>• Elabora le risposte HTTP dell'API di TMDB</li> </ul>	<ul style="list-style-type: none"> <li>• TMDB</li> </ul>

## Applicazione Mobile - Data Layer

<b>&lt;&lt;interface&gt;&gt;</b> <b>UserDataAsyncAccess</b>	
Superinterfacce	-
Sottointerfacce	-
<b>Responsabilità</b>	<b>Collaboratori</b>
• Gestisce l'accesso asincrono ai dati primari dell'utente	-

<b>&lt;&lt;interface&gt;&gt;</b> <b>UserFollowingsDataAsyncAccess</b>	
Superinterfacce	-
Sottointerfacce	-
<b>Responsabilità</b>	<b>Collaboratori</b>
• Gestisce l'accesso asincrono ai dati dell'utente relativi alle sue relazioni follow/follower con altri utenti	-

<b>&lt;&lt;interface&gt;&gt;</b> <b>UserFeedDataAsyncAccess</b>	
Superinterfacce	-
Sottointerfacce	-
<b>Responsabilità</b>	<b>Collaboratori</b>
• Gestisce l'accesso asincrono ai dati dell'utente relativi al feed delle azioni recenti	-

<<interface>> <b>UserMovieListsDataAsyncAccess</b>	
Superinterfacce	-
Sottointerfacce	-
Responsabilità	Collaboratori
• Gestisce l'accesso asincrono ai dati dell'utente relativi alle sue liste di film	-

<b>FirestoreUserData</b>	
Superclassi	-
Sottoclassi	-
Implementa	<b>UserDataAsyncAccess</b>
Responsabilità	Collaboratori
• Accede ai dati primari dell'utente tramite Firebase Firestore	-

<b>FirestoreUserFollowingsData</b>	
Superclassi	-
Sottoclassi	-
Implementa	<b>UserFollowingsDataAsyncAccess</b>
Responsabilità	Collaboratori
• Accede ai dati dell'utente relativi alle sue relazioni follow/follower con altri utenti tramite Firebase Firestore	-

FirestoreUserFeedData	
Superclassi	-
Sottoclassi	-
Implementa	UserFeedDataAsyncAccess
Responsabilità	Collaboratori
• Accede ai dati dell'utente relativi al feed delle azioni recenti tramite Firebase Firestore	-

FirestoreUserMovieListsData	
Superclassi	-
Sottoclassi	-
Implementa	UserMovieListsDataAsyncAccess
Responsabilità	Collaboratori
• Accede ai dati dell'utente relativi alle sue liste di film tramite Firebase Firestore	-

FirestoreReferences	
Superclassi	-
Sottoclassi	-
Implementa	-
Responsabilità	Collaboratori
• Gestisce l'accesso ai riferimenti alle collezioni e ai documenti di Firebase Firestore	• ListenerDispenser

FirebaseStorageHandler	
Superclassi	-
Sottoclassi	-
Implementa	-
Responsabilità	Collaboratori
• Gestisce l'accesso al Firebase Storage	-

## Applicazione Mobile - Model

AppUser	
Superclassi	-
Sottoclassi	-
Implementa	-
	Responsabilità
• Modella un utente dell'applicazione	Collaboratori -

Movie	
Superclassi	-
Sottoclassi	-
Implementa	Comparable<Movie>
	Responsabilità
• Modella un film con i dovuti dati	Collaboratori -

UserAction	
Superclassi	-
Sottoclassi	UserActionFollow, UserActionMovieList
Implementa	-
	Responsabilità
• Modella un'azione di un utente. Per azione si intende l'aggiungimento o la rimozione di un film da una lista o il follow di un altro utente.	Collaboratori • AppUser

UserActionFollow	
Superclassi	<i>UserAction</i>
Sottoclassi	-
Implementa	-
	Responsabilità
• Modella l'azione di follow di un utente	Collaboratori
	• AppUser

UserActionMovieList	
Superclassi	<i>UserAction</i>
Sottoclassi	-
Implementa	-
	Responsabilità
• Modella l'azione di aggiungimento o rimozione di un film da una lista	Collaboratori
	• AppUser

## Applicazione Mobile - Helper

TMDB	
Superclassi	-
Sottoclassi	-
Implementa	-
Responsabilità	Collaboratori
• Prepara le richieste HTTP da inoltrare all'API di TMDB	• <b>TMDBRequestManager</b>

StatisticsUpdates	
Superclassi	-
Sottoclassi	-
Implementa	-
Responsabilità	Collaboratori
• Aggiorna le statistiche del sistema	-

InputChecker	
Superclassi	-
Sottoclassi	-
Implementa	-
Responsabilità	Collaboratori
• Verifica gli input dell'utente sui dati e fornisce un'operazione di formattazione dei nomi	• <b>AppUser</b>

## Client Desktop - Application Layer

Controller	
Superclassi	-
Sottoclassi	-
Implementa	-
Responsabilità	Collaboratori
• Gestisce l'interazione tra i dati e l'interfaccia grafica	• AdminDAO

Controller	
Superclassi	-
Sottoclassi	-
Implementa	-
Responsabilità	Collaboratori
• Gestisce l'interazione tra i dati e l'interfaccia grafica	• AdminDAO

## Client Desktop - Data Layer

<b>&lt;&lt;interface&gt;&gt;</b> AdminDAO	
Superinterfacce	-
Sottointerfacce	-
Responsabilità	Collaboratori
• Gestisce l'accesso ai dati degli admin	-

<b>&lt;&lt;interface&gt;&gt;</b> StatisticsDAO	
Superinterfacce	-
Sottointerfacce	-
Responsabilità	Collaboratori
• Gestisce l'accesso alle statistiche della sezione Mobile	-

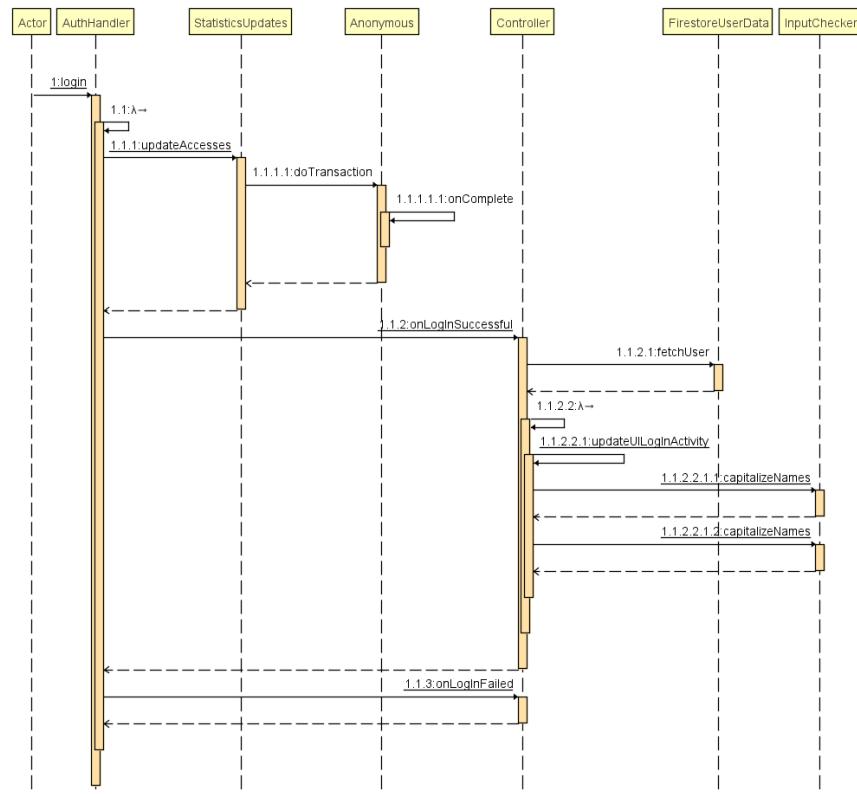
<b>AdminFileAccess</b>	
Superclassi	-
Sottoclassi	-
Implementa	AdminDAO
Responsabilità	Collaboratori
• Accede ai dati admin presenti sul file delle credenziali	-

RTDBStatisticsAccess	
Superclassi	-
Sottoclassi	-
Implementa	StatisticsDAO
Responsabilità	Collaboratori
• Accede alle statistiche della sezione Mobile tramite l'API del Firebase Real Time Database	-

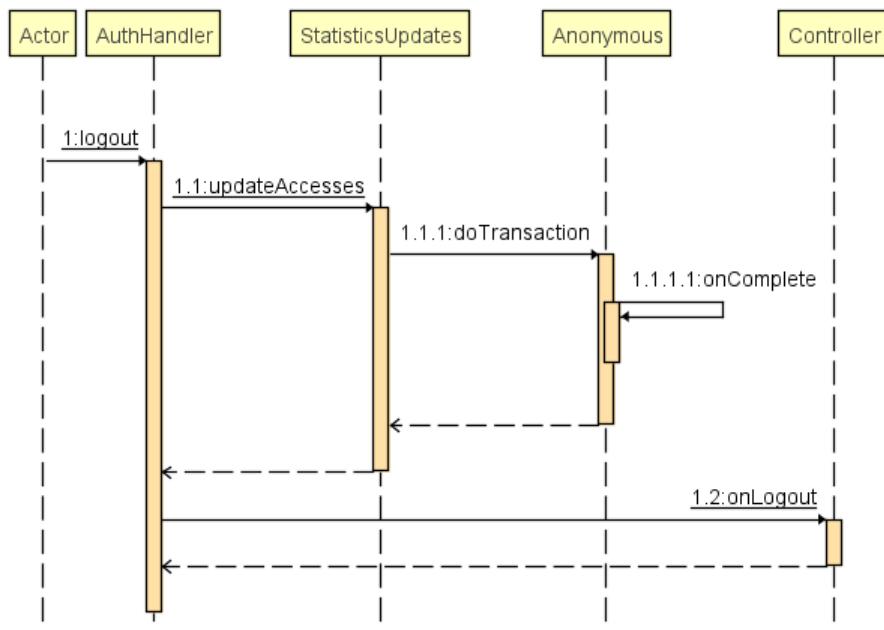
### 3.4 Diagrammi di sequenza di design per gli scenari non banali dei casi d'uso assegnati

Di seguito si presentano i Sequence Diagram per i metodi principali messi a punto per implementare le funzionalità descritte nelle precedenti sezioni, nei punti in cui è risultato effettivamente possibile evidenziare un significativo dinamismo.

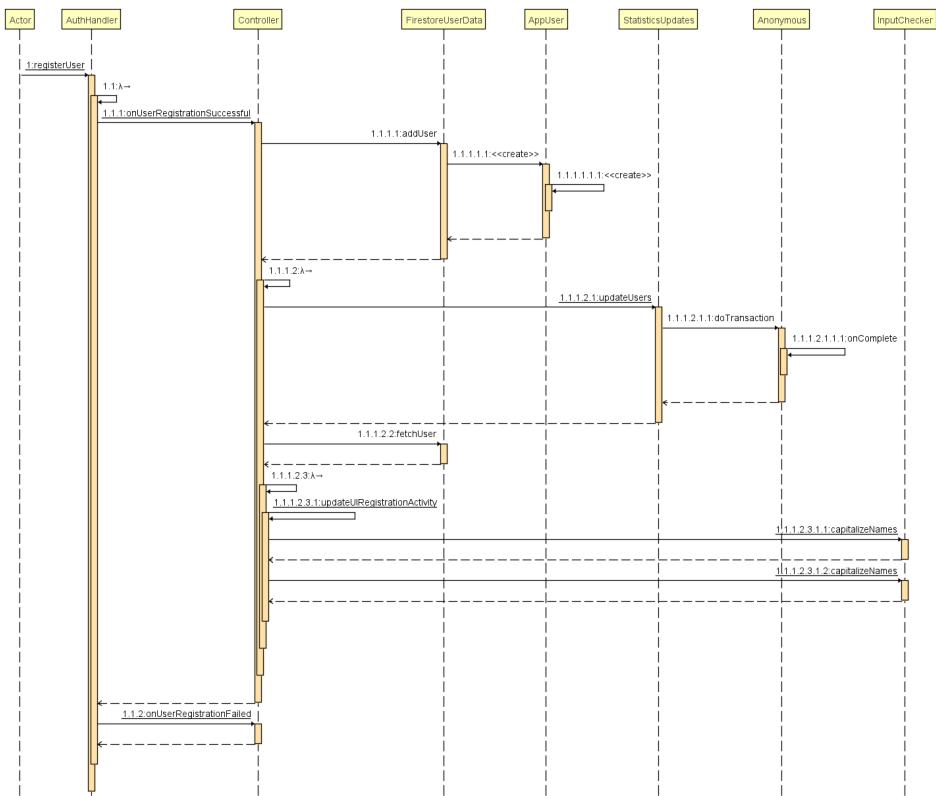
#### Applicazione Mobile



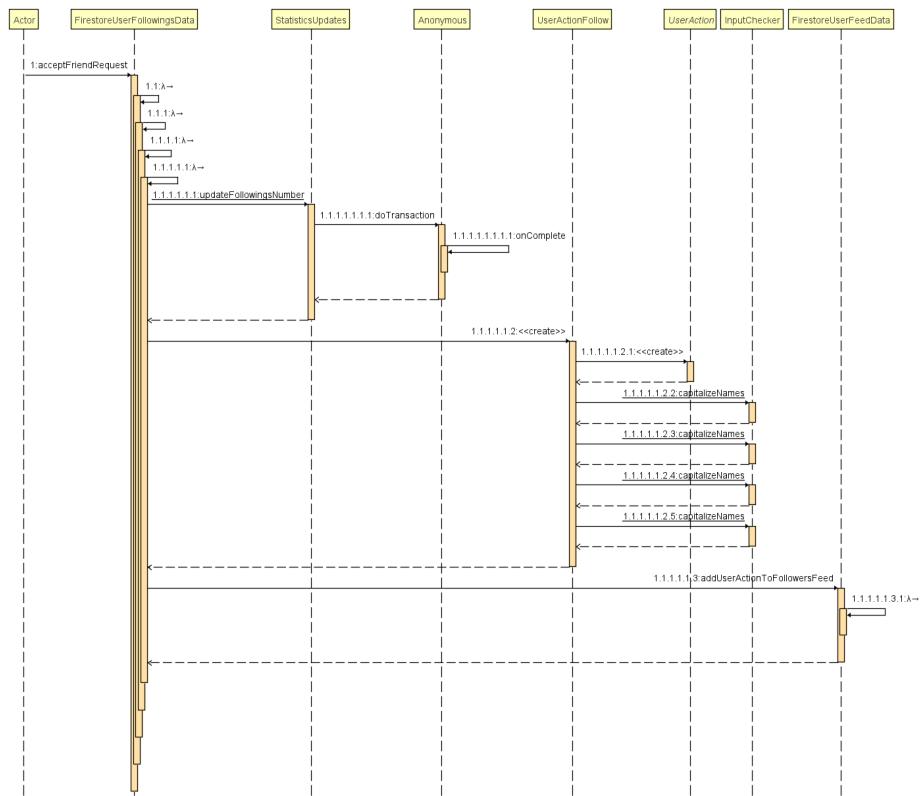
Sequence Diagram 1 - login



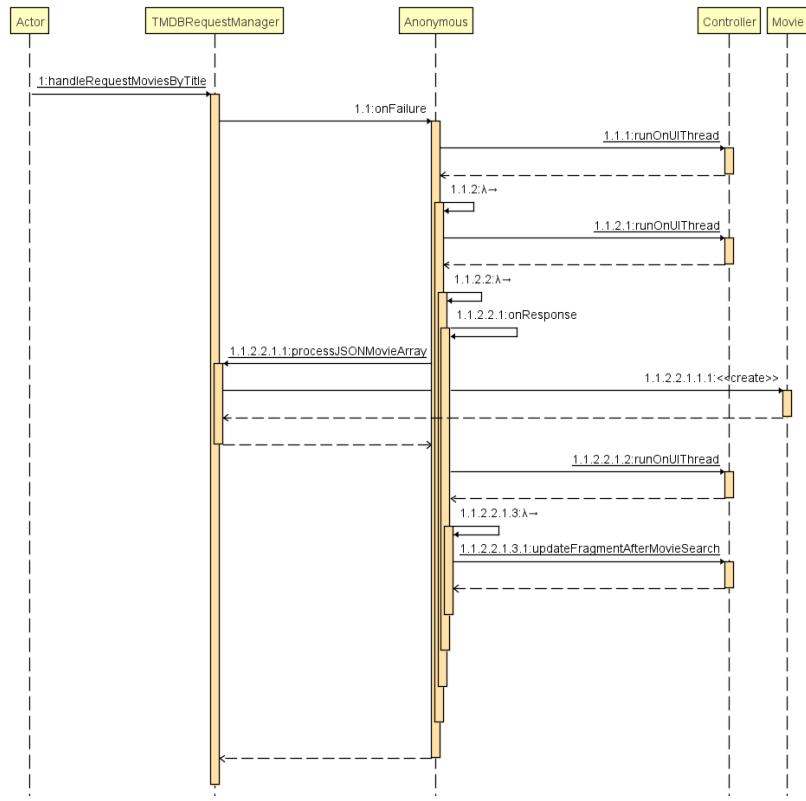
Sequence Diagram 2 - logout



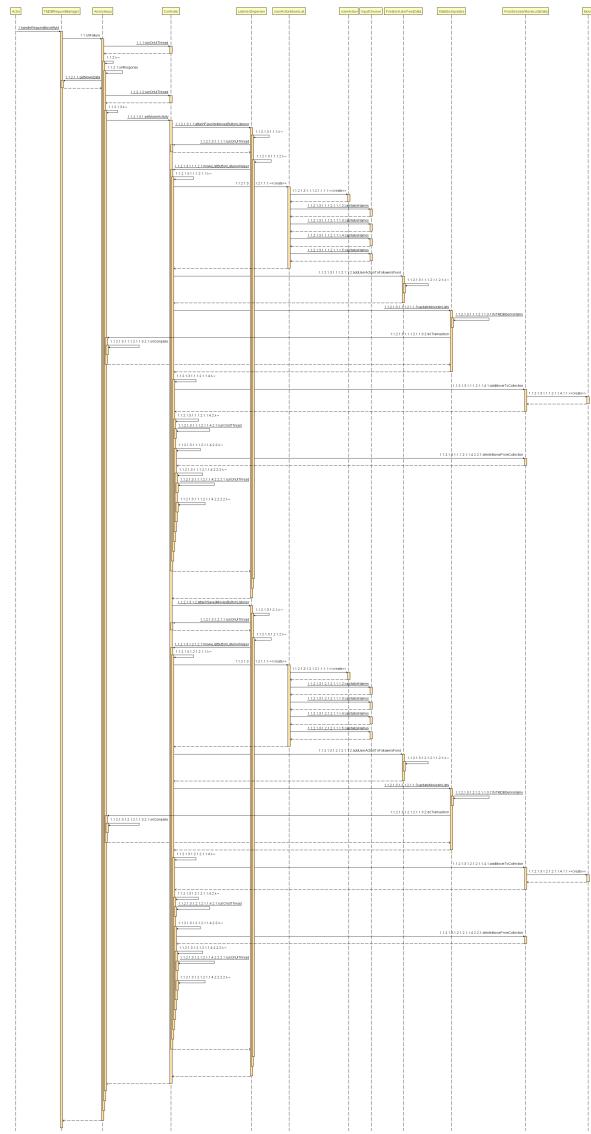
Sequence Diagram 3 - registerUser



## Sequence Diagram 4 - acceptFriendRequest

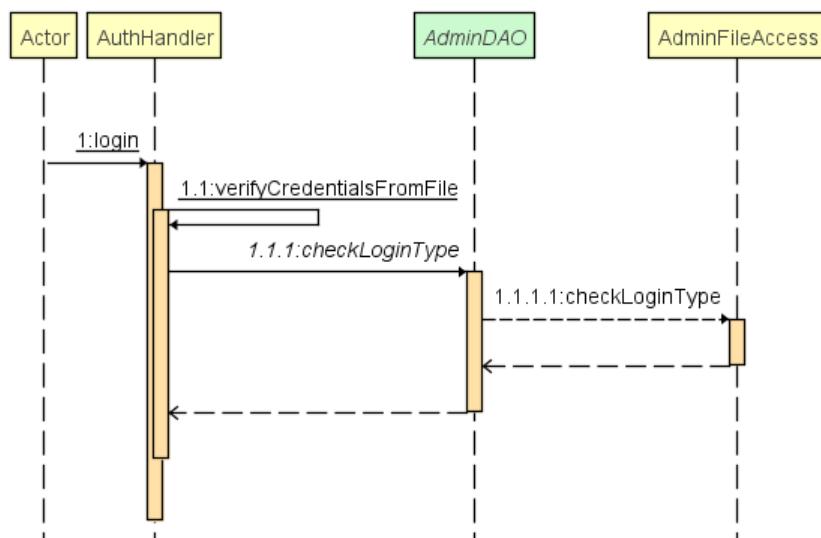


Sequence Diagram 5 - handleRequestMovieByTitle



Sequence Diagram 6 - handleRequestMovieById

## Client Desktop



Sequence Diagram - login

## 4 Testing

### 4.1 Codice xUnit per unit testing di 4 funzionalità non banali

In fase di testing del software, la scelta delle funzionalità da testare è ricaduta su due unità appartenenti all'applicazione mobile e due del client desktop di amministrazione, che sono state ritenute essere le più esemplificative.

Per quanto riguarda l'applicazione, si è optato per una verifica delle meccaniche di Log In e di Registrazione al sistema. Come supporto al testing mobile si è fatto uso del framework automatizzato Android Espresso, che appoggiandosi a JUnit agevola notevolmente il processo di verifica delle specifiche dell'interfaccia grafica.

Sul client desktop di amministrazione si sono scelti come casi di test il metodo di aggiornamento delle statistiche sulla finestra principale e quello di verifica delle credenziali di un admin da file. Per il primo di questi metodi si è fatto uso di un'operazione di mocking rendendo il test indipendente dal reperimento dei dati tramite l'API del Firebase Real Time Database.

Per ciascun caso di test è stata messa a punto una tabella che riporta un identificativo univoco del test, il suo nome all'interno del codice e una descrizione generale della sua utilità. Seguono a queste informazioni le sequenze di input che danno origine all'operazione di testing. Ciascuna tabella è accompagnata da opportuni screenshot del codice.

A causa di limitazioni dovute all'assenza di metodi particolarmente elaborati e alle problematiche di interazione tra Firebase, The Movie DataBase e i framework di testing disponibili, il modo migliore di procedere è stato quello di un approccio white-box diretto, che tuttavia non compromette in alcun modo la significatività del test. Per la visualizzazione completa dell'infrastruttura si rimanda agli opportuni package del codice sorgente dove essa è contenuta.

#### 4.1.1 Applicazione Mobile - Log In

Test ID	1A
Test name	A_testLogInWithNoEmail
Test description	Un utente sta cercando di effettuare il login senza aver inserito la sua Email.
<b>Input</b>	<b>Risultato atteso</b>
Viene inserita nel campo <i>Password</i> la stringa di prova “test”.	-
Viene premuto il tasto <i>LOG IN</i>	Viene mostrato un messaggio di errore nel campo <i>Email</i> .

```
@Test
//1A
public void A_testLogInWithNoEmail() {

    onView(withId(id_passwordInput))
        .perform(typeText(RANDOM_PASSWORD), closeSoftKeyboard());

    onView(withId(id_loginButton))
        .perform(click());

    onView(withId(id_emailInput)).check(matches(isDisplayed()));
}
```

Test ID	1B
Test name	B_testLogInWithInvalidEmailFormat
Test description	Un utente sta cercando di effettuare il login con una Email non valida
<b>Input</b>	<b>Risultato atteso</b>
Viene inserita nel campo <i>Email</i> la stringa di prova “test” come email non valida.	-
Viene inserita nel campo <i>Password</i> la stringa di prova “test”.	-
Viene premuto il tasto <i>LOG IN</i>	Viene mostrato un messaggio di errore nel campo <i>Email</i> .

```

    @Test
    //1B
    public void B_testLogInWithInvalidEmailFormat() {

        onView(withId(id_emailInput))
            .perform(typeText(INVALID_EMAIL), closeSoftKeyboard());
        onView(withId(id_passwordInput))
            .perform(typeText(RANDOM_PASSWORD), closeSoftKeyboard());

        onView(withId(id_LoginButton))
            .perform(click());

        onView(withId(id_emailInput)).check(matches(isDisplayed()));
    }
}

```

<b>Test ID</b>	1C
<b>Test name</b>	C_testLogInWithWrongPassword
<b>Test description</b>	Un utente sta cercando di effettuare il login con una password errata
<b>Input</b>	<b>Risultato atteso</b>
Viene inserita nel campo <i>Email</i> la stringa "prova@prova.it" corrispondente alla Email di un utente di test.	-
Viene inserita nel campo <i>Password</i> la stringa di prova "test".	-
Viene premuto il tasto <i>LOG IN</i>	I dati vengono processati, e viene mostrato un messaggio di errore all'utente

```

@Test
//1C
public void C_testLogInWithWrongPassword() {

    onView(withId(id_emailInput))
        .perform(typeText(CORRECT_EMAIL), closeSoftKeyboard());
    onView(withId(id_passwordInput))
        .perform(typeText(RANDOM_PASSWORD), closeSoftKeyboard());

    onView(withId(id_LoginButton))
        .perform(click());

    waitForTask();

    onView(withId(id_emailInput)).check(matches(isDisplayed()));
}

}

```

<b>Test ID</b>	1D
<b>Test name</b>	A_testLogInWithCorrectCredentials
<b>Test description</b>	Un utente sta cercando di effettuare il login con le credenziali corrette.
<b>Input</b>	<b>Risultato atteso</b>
Viene inserita nel campo <i>Email</i> la stringa "prova@prova.it" corrispondente alla Email di un utente di test.	-
Viene inserita nel campo <i>Password</i> la stringa "123123" corrispondente alla password dell'utente di prova.	-
Viene premuto il tasto <i>LOG IN</i>	I dati vengono processati e l'utente viene indirizzato alla Home.

```

@Test
//1D
public void D_testLogInWithCorrectCredentials() {

    onView(withId(id_emailInput)).check(matches(isDisplayed()))
        .perform(typeText(CORRECT_EMAIL), closeSoftKeyboard());
    onView(withId(id_passwordInput))
        .perform(typeText(CORRECT_PASSWORD), closeSoftKeyboard());

    onView(withId(id_LoginButton))
        .perform(click());

    waitForTask();

    onView(withId(id_mainActivity)).check(matches(isDisplayed()));
}

```

#### 4.1.2 Registrazione

Test ID	2A
Test name	A_testRegistrationWithInvalidName
Test description	Un utente sta cercando di registrarsi con un nome non valido
Input	Risultato atteso
Vengono inseriti rispettivamente nei campi <i>Cognome</i> , <i>Email</i> , <i>Password</i> le stringhe di prova valide “user”, “test@test.it”, “password”	-
Iterativamente, vengono inserite come nomi non validi le stringhe “”, “prova1”, “p” all’interno del campo <i>Nome</i> , e per ciascuna si preme il tasto <i>REGISTRATI</i>	Per ciascuna iterazione viene mostrato un messaggio di errore nel campo <i>Nome</i>

```

@Test
//2A
public void A_testRegistrationWithInvalidName() {

    onView(withId(id_surnameInput))
        .perform(typeText(VALID_SURNAME), closeSoftKeyboard());
    onView(withId(id_emailInput))
        .perform(typeText(VALID_EMAIL), closeSoftKeyboard());
    onView(withId(id_passwordInput))
        .perform(typeText(VALID_PASSWORD), closeSoftKeyboard());

    for (String s : INVALID_NAMES) {
        onView(withId(id_nameInput))
            .perform(clearText(), typeText(s), closeSoftKeyboard());
        onView(withId(id_registerButton))
            .perform(click());
        onView(withId(id_emailInput)).check(matches(isDisplayed()));
    }
}

```

Test ID	2B
Test name	B_testRegistrationWithInvalidSurname
Test description	Un utente sta cercando di registrarsi con un cognome non valido
<b>Input</b>	<b>Risultato atteso</b>
Vengono inseriti rispettivamente nei campi <i>Nome</i> , <i>Email</i> , <i>Password</i> le stringhe di prova valide “test”, “test@test.it”, “password”	-
Iterativamente, vengono inserite come cognomi non validi le stringhe “”, “prova1”, “p” all’interno del campo <i>Cognome</i> , e per ciascuna si preme il tasto <i>REGISTRATI</i>	Per ciascuna iterazione viene mostrato un messaggio di errore nel campo <i>Cognome</i>

```

@Test
//2B
public void B_testRegistrationWithInvalidSurname() {

    onView(withId(id_nameInput))
        .perform(typeText(VALID_NAME), closeSoftKeyboard());
    onView(withId(id_emailInput))
        .perform(typeText(VALID_EMAIL), closeSoftKeyboard());
    onView(withId(id_passwordInput))
        .perform(typeText(VALID_PASSWORD), closeSoftKeyboard());

    for (String s : INVALID_SURNAMES) {
        onView(withId(id_surnameInput))
            .perform(clearText(), typeText(s), closeSoftKeyboard());
        onView(withId(id_registerButton))
            .perform(click());
        onView(withId(id_emailInput)).check(matches(isDisplayed()));
    }
}

```

Test ID	2C
Test name	C_testRegistrationWithInvalidEmailFormat
Test description	Un utente sta cercando di registrarsi con una email non valida
Input	Risultato atteso
Vengono inseriti rispettivamente nei campi <i>Nome</i> , <i>Cognome</i> , <i>Password</i> le stringhe di prova valide "user", "test", "password"	-
Iterativamente, vengono inserite come Email non valide le stringhe "abc..def@mail.com", ".abc@mail.com", "abc#def@mail.com", "abc.def@mail.c", "abc.def@mail#archive.com", "abc.def@mail", "abc.def@mail..com" all'interno del campo <i>Email</i> , e per ciascuna si preme il tasto <i>REGISTRATI</i>	Per ciascuna iterazione viene mostrato un messaggio di errore nel campo <i>Email</i> o, dopo aver processato i dati, la Email viene rifiutata dal sistema

```

@Test
//2C
public void C_testRegistrationWithInvalidEmailFormat() {

    onView(withId(id_nameInput))
        .perform(typeText(VALID_NAME), closeSoftKeyboard());
    onView(withId(id_surnameInput))
        .perform(typeText(VALID_SURNAME), closeSoftKeyboard());
    onView(withId(id_passwordInput))
        .perform(typeText(VALID_PASSWORD), closeSoftKeyboard());

    for (String s : INVALID_EMAILS) {
        onView(withId(id_emailInput))
            .perform(clearText(), typeText(s), closeSoftKeyboard());
        onView(withId(id_registerButton))
            .perform(click());
        waitForTask();
        onView(withId(id_emailInput)).check(matches(isDisplayed()));
    }
}

```

Test ID	2D
Test name	D_testRegistrationWithInvalidPassword
Test description	Un utente sta cercando di registrarsi con una password non valida
<b>Input</b>	<b>Risultato atteso</b>
Vengono inserite rispettivamente nei campi <i>Nome</i> , <i>Cognome</i> , <i>Email</i> le stringhe di prova valide "user", "test", "test@test.it"	-
Iterativamente, vengono inserite come password non valide le stringhe "", "password", "passw", "password" all'interno del campo <i>Password</i> , e per ciascuna si preme il tasto <i>REGISTRATI</i>	Per ciascuna iterazione viene mostrato un messaggio di errore nel campo <i>Password</i> .

```

@Test
//2D
public void D_testRegistrationWithInvalidPassword() {

    onView(withId(id_nameInput))
        .perform(typeText(VALID_NAME), closeSoftKeyboard());
    onView(withId(id_surnameInput))
        .perform(typeText(VALID_SURNAME), closeSoftKeyboard());
    onView(withId(id_emailInput))
        .perform(typeText(VALID_EMAIL), closeSoftKeyboard());

    for (String s : INVALID_PASSWORDS) {
        onView(withId(id_passwordInput))
            .perform(clearText(), typeText(s), closeSoftKeyboard());
        onView(withId(id_registerButton))
            .perform(click());
        onView(withId(id_emailInput)).check(matches(isDisplayed()));
    }
}

```

Test ID	2E
Test name	E_testRegistrationWithExistingEmail
Test description	Un utente sta cercando di registrarsi con una email già in uso
<b>Input</b>	<b>Risultato atteso</b>
Vengono inserite rispettivamente nei campi <i>Nome</i> , <i>Cognome</i> , <i>Password</i> le stringhe di prova valide “user”, “test”, “password”	-
Viene inserita nel campo <i>Email</i> l'email “prova@prova.it”, appartenente all'utente di test	-
Si preme il tasto <i>REGISTRATI</i>	Le informazioni vengono processate dal sistema e la mail rifiutata, con un messaggio di errore

```

@Test
//2E
public void E_testRegistrationWithExistingEmail() {

    onView(withId(id_nameInput))
        .perform(typeText(VALID_NAME), closeSoftKeyboard());
    onView(withId(id_surnameInput))
        .perform(typeText(VALID_SURNAME), closeSoftKeyboard());
    onView(withId(id_passwordInput))
        .perform(typeText(VALID_PASSWORD), closeSoftKeyboard());

    onView(withId(id_emailInput))
        .perform(clearText(), typeText(EXISTING_EMAIL), closeSoftKeyboard());
    onView(withId(id_registerButton))
        .perform(click());
    waitForTask();
    onView(withId(id_emailInput)).check(matches(isDisplayed()));
}

}

```

Test ID	2F
Test name	F_testRegistrationWithValidData
Test description	Un utente sta effettuando la registrazione con dati validi
<b>Input</b>	<b>Risultato atteso</b>
Vengono inserite rispettivamente nei campi <i>Nome</i> , <i>Cognome</i> , <i>Email</i> , <i>Password</i> le stringhe di prova valide "user", "test", "test@test.it", "password"	-
Si preme il tasto <i>REGISTRATI</i>	Le informazioni vengono processate dal sistema, viene creato un nuovo utente con i dati inseriti e viene mostrata la schermata Home all'utente

```

@Test
//2F
public void F_testRegistrationWithValidData() {

    onView(withId(id_nameInput))
        .perform(typeText(VALID_NAME), closeSoftKeyboard());
    onView(withId(id_surnameInput))
        .perform(typeText(VALID_SURNAME), closeSoftKeyboard());
    onView(withId(id_emailInput))
        .perform(typeText(VALID_EMAIL), closeSoftKeyboard());
    onView(withId(id_passwordInput))
        .perform(typeText(VALID_PASSWORD), closeSoftKeyboard());

    onView(withId(id_registerButton))
        .perform(click());
    waitForTask();
    onView(withId(id_mainActivity)).check(matches(isDisplayed()));

}

```

#### 4.1.3 Aggiornamento delle statistiche sul client

Test ID	3A
Test name	testUpdateStats
Test description	Aggiornamento dei valori delle statistiche del sistema sul MainFrame del client
<b>Input</b>	<b>Risultato atteso</b>
Il metodo viene invocato con uno Stub che rimpiazza l'effettiva classe di connessione all'API	I campi del MainFrame vengono aggiornati con i valori raccolti dal metodo

```
@Test
//3A
void testUpdateStats() {

    MainFrame frame = new MainFrame(true);

    Controller.setStatisticsAccess(new RTDBStatisticsAccess_Stub());

    Controller.updateStats(frame);

    for (int i = 0; i < frame.fields.length; i++) {
        Assertions.assertEquals(i, Integer.valueOf(frame.fields[i].getText().trim()));
    }
}
```

#### 4.1.4 Verifica delle credenziali admin da file

Test ID	4A
Test name	testVerifyCredentialsFromFileFirstLogin
Test description	Primo tentativo di accesso al client desktop
<b>Input</b>	<b>Risultato atteso</b>
Si scrive sul file delle credenziali di accesso la strniga “admin password 0”.	-
Si inserisce nel campo <i>Nome Admin</i> il valore corretto “admin”	-
Si inserisce nel campo <i>Password Admin</i> il valore corretto “password”	-
Si preme il pulsante <i>Accedi</i>	Viene restituito il valore 0

```

@Test
//4A
void testVerifyCredentialsFromFileFirstLogin () {

    try (FileWriter writer = new FileWriter(TEST_FILE_PATH, false)) {
        writer.write(FIRST_LOGIN_CREDENTIALS);
    } catch (IOException e) {
        e.printStackTrace();
    }

    Controller.setFilePath(TEST_FILE_PATH);
    int result = AuthHandler.verifyCredentialsFromFile(FIRST_LOGIN_CREDENTIALS_INPUT[0],
                                                       FIRST_LOGIN_CREDENTIALS_INPUT[1]);
    assertEquals(AuthHandler.FIRST_LOGIN, result);

}

```

Test ID	4B
Test name	<b>testVerifyCredentialsFromFileSuperadminLogin</b>
Test description	Tentativo di accesso al client desktop da parte di un superadmin
<b>Input</b>	<b>Risultato atteso</b>
Si scrive sul file delle credenziali di accesso la stringa “newadmin newpassword 1”.	-
Si inserisce nel campo <i>Nome Admin</i> il valore corretto “newadmin”	-
Si inserisce nel campo <i>Password Admin</i> il valore corretto “newpassword”	-
Si preme il pulsante <i>Accedi</i>	Viene restituito il valore 1

```

@Test
//4B
void testVerifyCredentialsFromFileSuperadminLogin () {

    try (FileWriter writer = new FileWriter(TEST_FILE_PATH)) {
        writer.write(SUPERADMIN_LOGIN_CREDENTIALS);
    } catch (IOException e) {
        e.printStackTrace();
    }

    Controller.setFilePath(TEST_FILE_PATH);
    int result = AuthHandler.verifyCredentialsFromFile(SUPERADMIN_LOGIN_CREDENTIALS_INPUT[0],
                                                       SUPERADMIN_LOGIN_CREDENTIALS_INPUT[1]);
    assertEquals(AuthHandler.SUPERADMIN_LOGIN, result);

}

```

Test ID	4C
Test name	<b>testVerifyCredentialsFromFileNormalLogin</b>
Test description	Tentativo di accesso al client desktop da parte di un admin normale
<b>Input</b>	<b>Risultato atteso</b>
Si scrive sul file delle credenziali di accesso la stringa “name password”.	-
Si inserisce nel campo <i>Nome Admin</i> il valore corretto “name”	-
Si inserisce nel campo <i>Password Admin</i> il valore corretto “password”	-
Si preme il pulsante <i>Accedi</i>	Viene restituito il valore 2

```

@Test
//4C
void testVerifyCredentialsFromFileNormalLogin () {

    try (FileWriter writer = new FileWriter(TEST_FILE_PATH)) {
        writer.write(NORMAL_LOGIN_CREDENTIALS);
    } catch (IOException e) {
        e.printStackTrace();
    }

    Controller.setFilePath(TEST_FILE_PATH);
    int result = AuthHandler.verifyCredentialsFromFile(NORMAL_LOGIN_CREDENTIALS_INPUT[0],
                                                       NORMAL_LOGIN_CREDENTIALS_INPUT[1]);
    assertEquals(AuthHandler.NORMAL_LOGIN, result);

}

```

Test ID	4D
Test name	<b>testVerifyCredentialsFromFileFailedLogin</b>
Test description	Tentativo di accesso al client desktop da parte di un qualsiasi utente con credenziali errate
<b>Input</b>	<b>Risultato atteso</b>
Si scrive sul file delle credenziali di accesso la stringa "name password".	-
Si inserisce nel campo <i>Nome Admin</i> il valore "name"	-
Si inserisce nel campo <i>Password Admin</i> il valore non corretto "password1"	-
Si preme il pulsante <i>Accedi</i>	Viene restituito il valore 3
Si inserisce nel campo <i>Nome Admin</i> il valore non corretto "name1"	-
Si inserisce nel campo <i>Password Admin</i> il valore "password"	-
Si preme il pulsante <i>Accedi</i>	Viene restituito il valore 3

```

@Test
//4D
void testVerifyCredentialsFromFileFailedLogin () {

    try (FileWriter writer = new FileWriter(TEST_FILE_PATH)) {
        writer.write(FAILED_LOGIN_CREDENTIALS);
    } catch (IOException e) {
        e.printStackTrace();
    }

    Controller.setFilePath(TEST_FILE_PATH);
    int result;

    result = AuthHandler.verifyCredentialsFromFile(FAILED_LOGIN_CREDENTIALS_INPUT[0],
                                                   FAILED_LOGIN_CREDENTIALS_INPUT[1]);
    assertEquals(AuthHandler.WRONG_CREDENTIALS, result);

    result = AuthHandler.verifyCredentialsFromFile(FAILED_LOGIN_CREDENTIALS_INPUT[2],
                                                   FAILED_LOGIN_CREDENTIALS_INPUT[3]);
    assertEquals(AuthHandler.WRONG_CREDENTIALS, result);

}

```