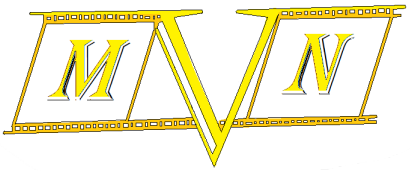
UNIVERSITÀ
DI TRENTO



Dipartimento di Ingegneria e
Scienza dell'Informazione

Progetto:

MoViewerNet

Titolo del documento:

Documento di Architettura

Gruppo T47, AA 2022/2023

Matteo Javid Battista, Aurora Ottaviani, Alessandro Nitti

"MoViewerNet"

Indice:

Scopo del documento:

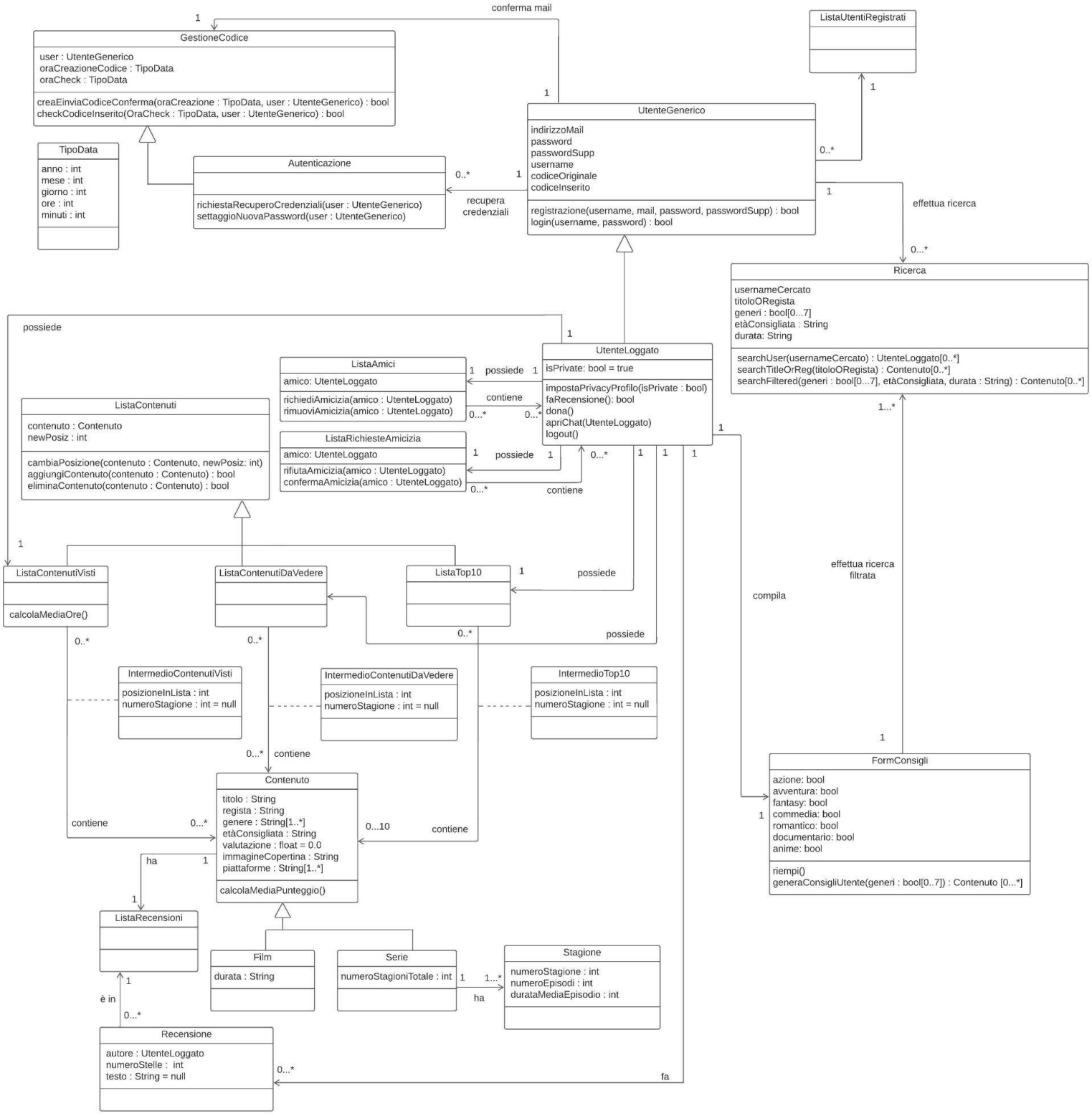
Nel documento precedente, si specificavano i requisiti funzionali e non funzionali tramite tabelle strutturate e use case in Unified Modeling Language UML e, inoltre, sono state fatte analisi del contesto e dei componenti tramite relativi diagrammi.

Nel presente documento, si descrive l'architettura del progetto MVN, a partire dalla precedente progettazione, con particolare riferimento al diagramma dei componenti del documento di specifica dei requisiti.

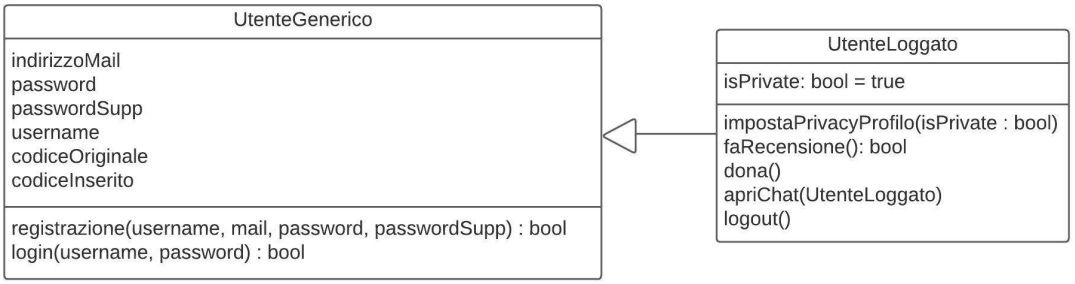
Si utilizza: il linguaggio naturale, il diagramma delle classi in linguaggio UML e codice in Object Constraint Language (OCL) per dettagliare ciò che non è specificabile nel diagramma delle classi ma, al contempo, è necessario per la logica di implementazione del sito web.

Diagramma delle classi.

Si riporta il diagramma delle classi in UML completo. Il diagramma è composto da classi, ognuna delle quali ha un nome, una lista di attributi che la caratterizza e/o necessaria alle funzioni della classe stessa.
Le classi possono essere associate da relazioni con frecce semplici, per facilitare la leggibilità del diagramma o relazioni di generalize (freccia vuota) per indicare le “isa”.
Si è cercato di massimizzare la coesione e minimizzare l’accoppiamento tra classi, al fine di efficientare il sistema.
Successivamente si illustreranno nel dettaglio le varie classi.



Per i due attori Utente generico e Utente loggato sono state introdotte due classi collegate da una relazione di is-a, poichè, rispetto al diagramma dei componenti, si è reso necessario raggruppare le funzione svolte dalle varie tipologie di utente.



UtenteGenerico:
La classe UtenteGenerico rappresenta un qualsiasi utente che utilizza il sito MVN.

Possiede i seguenti attributi:

- indirizzoMail
  Conterrà l'indirizzo e-mail inserito, a seconda dei casi, per registrarsi o autenticarsi.
- password
  Conterrà la prima digitazione della password (nel caso della registrazione) o la password per l'autenticazione o la prima digitazione di una nuova password (nel caso in cui l'utente abbia richiesto di reimpostare una nuova password).
- passwordSupp
  Conterrà la seconda digitazione della password (nel caso della registrazione) o la seconda digitazione di una nuova password (nel caso in cui l'utente abbia richiesto di reimpostare una nuova password).
- username
  Conterrà l'username scelto dall'utente (in caso di registrazione e/o autenticazione).
- codiceOriginale
  Conterrà il codice generato, inviato via mail, per confermare la validità della e-mail dell'utente (in caso di registrazione) o il codice generato, inviato via mail, per recuperare le credenziali.
- codiceInserito
  Conterrà il codice inserito nel sito dall'utente per confermare la validità della sua e-mail (in caso di registrazione) o il codice inserito nel sito dall'utente per recuperare le proprie credenziali.

Possiede i seguenti metodi:

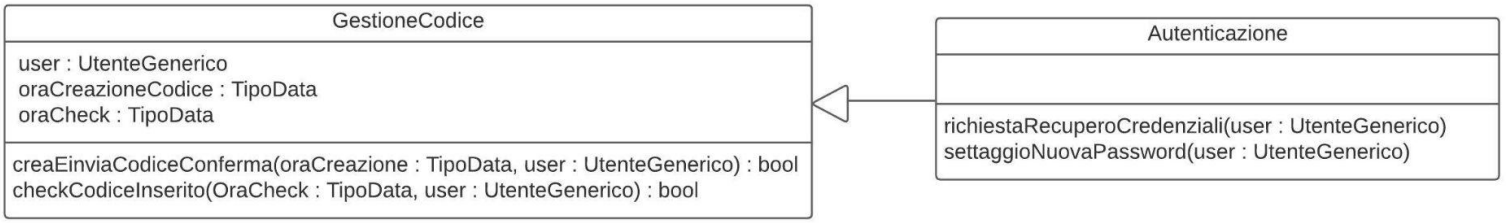
- registrazione(username, mail, password, passwordSupp) : bool
  Gli attributi passati come parametri sono inseriti dall'utente nel form di registrazione. Quando l'utente clicca su invio, vengono chiamati i metodi della classe GestioneCodice e l'utente viene reindirizzato ad una pagina nel quale inserire il codice ricevuto via mail, dunque CodiceOriginale e Codice inserito saranno poi confrontati, se uguali, verrà inizializzata un'istanza della classe UtenteGenerico. Se dovessero sussistere problemi in corso di registrazione, verrà mostrato un messaggio di errore all'utente.

É stata inoltre creata una classe “fittizia” ListaUtentiRegistrati per segnalare la necessità di una collection con all’interno tutti gli utenti registrati nel sito. Questo è indispensabile sia per il metodo “registrazione”, dove viene controllato che l’username e la mail inseriti non siano già presenti nella ListaUtentiRegistrati, sia per il metodo “login”, dove viene controllato che ci sia una corrispondenza tra i dati inseriti e la ListaUtentiRegistrati, sia per il metodo “searchUser”, che cercherà una corrispondenza tre la stringa inserita dall’utente e la ListaUtentiRegistrati.

Per il componente “Registrazione” del diagramma dei componenti non è stata creata una classe, per aumentare la coesione tra classi, poichè si è deciso di creare un’unica classe GestioneCodice sia per confermare la mail inserita dall’utente in fase di registrazione, sia per il recupero credenziali, per non creare duplicazione di codice superflua.

- login(username, password) : bool
  Gli attributi passati come parametri sono inseriti dall’utente nel form di autenticazione. Il parametro username può essere sia l’username dell’utente specifico, sia la sua mail.
  Quando l’utente clicca su invio, i parametri vengono passati alla classe ListaUtentiRegistrati che verifica i dati e, se corretti, viene inizializzata un’istanza della classe UtenteLoggato con quei dati.
  Se le credenziali inserite dovessero essere sbagliate o inesistenti, verrà mostrato un messaggio di errore all’utente.

  Nel qual caso, l’utente può decidere di recuperare le credenziali. Questa funzione, prima gestita solamente dal componente “Autenticazione” nel diagramma dei componenti, viene suddivisa tra le classi Autenticazione e GestioneCodice.



GestioneCodice:
La classe GestioneCodice viene utilizzata da un UtenteGenerico per confermare la mail in fase di registrazione. I suoi metodi, inoltre, sono utilizzati anche in Autenticazione per confermare la propria identità, in caso di richiesta recupero credenziali.

Possiede i seguenti attributi:

- user : UtenteGenerico
  Contiene un elemento di tipo UtenteGenerico, relativo alla mail o username inseriti.
- oraCreazioneCodice : TipoData
  Contiene l’ora in cui è stato creato “codiceConferma”

- oraCheck : TipoData
  Conterrà l'ora in cui verrà confrontato il "codiceConferma" con il "codiceInserito".

Possiede i seguenti metodi:

- creaEinviaCodiceConferma(oraCreazione : TipoData, user : UtenteGenerico) : bool
  Il sistema genera un codice di conferma e lo invia alla mail dello user passato come parametro alla funzione. La funzione ritorna true, a conferma dell'operazione andata a buon fine.

  Per il componente "Gestione Mail" non è stata creata una classe poichè la mail viene inviata direttamente grazie al metodo "creaEinviaCodiceConferma".

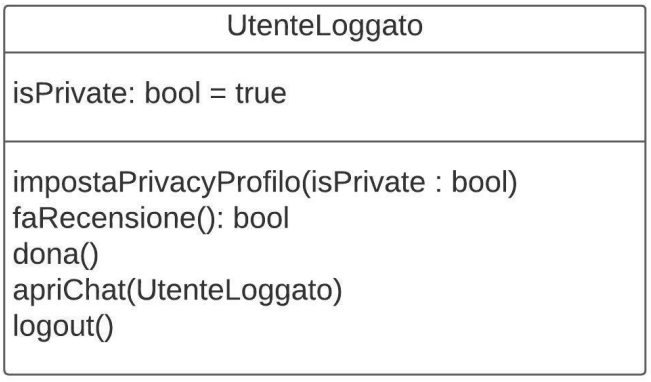
- checkCodiceInserito(oraCheck : TipoData, user : UtenteGenerico) : bool
  Se gli attributi "CodiceOriginale" e "CodiceInserito" risultano uguali e "CodiceOriginale" è ancora valido, la funzione ritorna true.

Autenticazione:
La classe autenticazione viene utilizzata da un UtenteGenerico per recuperare le proprie credenziali e settare una nuova password.

Possiede i seguenti metodi:

- richiestaRecuperoCredenziali(user)
  É il metodo che avvia la richiesta di recupero credenziali, relativa ad uno specifico user. All'interno verranno chiamati i metodi della classe GestioneCodice. Se entrambe le funzioni di GestioneCodice ritornano true viene inizializzata un'istanza della classe UtenteGenerico.
- settaggioNuovaPassword(user)
  È l'ultimo metodo utilizzato per la richiesta di recupero credenziali, relativa ad uno specifico "user", per reimpostare la propria password. Dopo aver inserito il codice ricevuto via mail, l'utente infatti viene reindirizzato ad una nuova pagina in cui inserire la nuova password.
  Se la password inserita è uguale alla vecchia password, il sistema riporterà un messaggio di errore all'utente.

UtenteLoggato:
La classe UtenteLoggato rappresenta un utente loggato che utilizza il sito MVN.



Possiede i seguenti attributi:

- isPrivate : bool = true
  L'attributo è indicativo delle impostazioni di visibilità del profilo utente, di default è settato a true.

Possiede i seguenti metodi:

- impostaPrivacyProfilo(isPrivate : bool)
  Metodo che modifica l'attributo isPrivate della classe, a seconda delle impostazioni profilo.

- faRecensione() : bool
  Crea un' istanza della classe Recensione e ritorna true se l'operazione è andata a buon fine.

- dona()
  Funzione che permette ad un UtenteLoggato di effettuare una donazione ai programmatori del sito. Reindirizza direttamente l'utente a Paypal.

  Il componente "Gestione Donazioni" trova il suo corrispondente nel metodo "dona" e non in una classe per favorire la semplicità del diagramma delle classi.

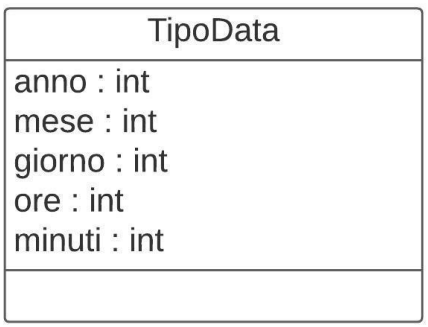
- apriChat(UtenteLoggato)
  Reindirizza l'UtenteLoggato al servizio chat, per metterlo in comunicazione con un secondo UtenteLoggato passato come parametro alla funzione.

  Il componente "Gestione Chat" trova il suo corrispondente nel metodo "apriChat" e non in una classe per favorire la semplicità del diagramma delle classi.

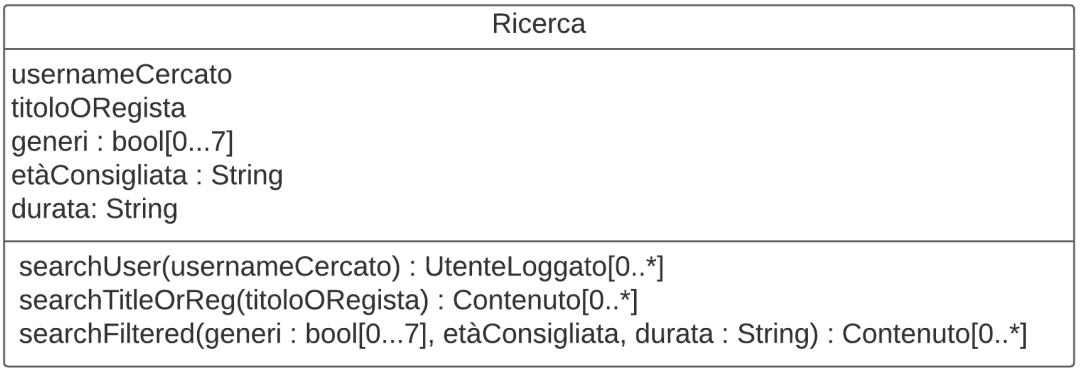
- logout()
  Cancella l'istanza dell'UtenteLoggato che torna ad essere un UtenteGenerico.

  Nel diagramma dei componenti la funzione di logout passava per il modulo "Autenticazione", ma nel diagramma delle classi si è deciso di includerla nelle funzioni della classe UtenteLoggato, poichè aumenta la semplicità del sistema. Similmente si è ragionato per la funzione "impostaPrivacyProfilo" che prima passava solo per il componente "Gestione Database" del diagramma dei componenti.

TipoData:
Il TipoData è una classe di supporto creata per gestire la distruzione dei codici della classe GestioneCodice.



Ricerca:
Il componente “Ricerca” del diagramma dei componenti è stato tradotto nella classe Ricerca e questa viene utilizzata sia dall’UtenteGenerico che dall’UtenteLoggato per effettuare un qualsiasi tipo di ricerca sul sito MVN.



Possiede i seguenti attributi:

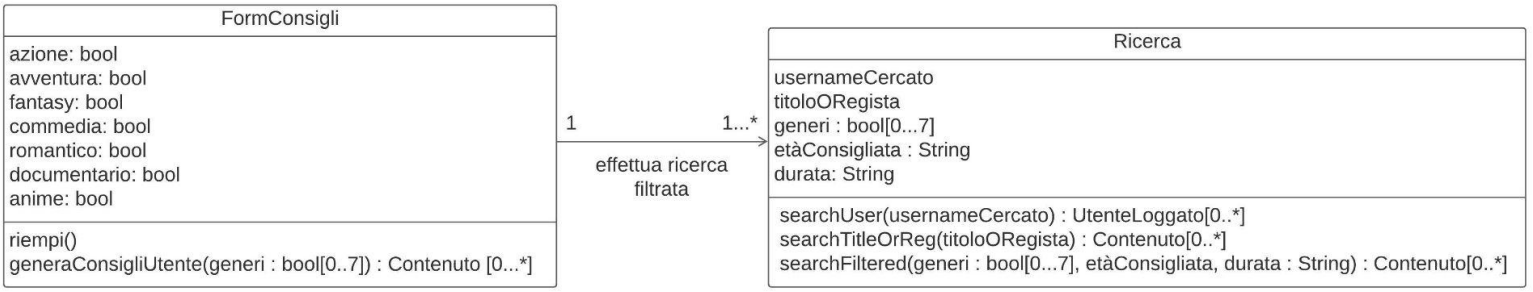
- usernameCercato
  Contiene l’username dell’ utente cercato (username inserito).
- titoloORegista
  Contiene il titolo o il regista del contenuto cercato.
- generi : bool[0...7]
  Se selezionati, contiene i filtri in base al genere di contenuto desiderato.
- etàConsigliata : String
  Se selezionato, contiene i filtri in base all’età consigliata selezionati per i contenuti da cercare.
- durata : String
  Se selezionato, contiene il filtro che determina la durata massima del contenuto che si vuole vedere

Possiede i seguenti metodi:

- searchUser(usernameCercato) : UtenteLoggato[0..*]
  Il metodo prende come parametro di ricerca la stringa usernameCercato inserita dall’utente e controlla se presente nella listaUtentiRegistrati. Se l’usernameCercato non è presente restituisce un errore all’utente, viceversa restituisce la pagina profilo dell’utente cercato.
- searchTitleOrReg(titoloORegista) : Contenuto[0..*]
  Il metodo prende come parametro di ricerca la stringa titoloORegista inserita dall’utente e restituisce una lista di contenuti (se vengono trovate più stringhe corrispondenti a titoloORegista o la pagina del contenuto cercato (se la corrispondenza è unica).
  Se non ci sono corrispondenze tra la stringa cercata e i contenuti presenti nel database, viene riportato un messaggio di errore all’utente.
- searchFiltered(generi, etàConsigliata, durata) : Contenuto[0..*]
  Il metodo prende come parametro di ricerca i filtri eventualmente impostati dall’utente e restituisce la pagina del contenuto cercato, se presente nel database.

Parte delle funzionalità del componente “Gestione Liste” del diagramma dei componenti sono quelle di organizzare i risultati delle ricerche effettuate da un utente (sia per i contenuti che per gli username). Questa particolare funzionalità non è stata esplicitata nel diagramma delle classi poiché viene direttamente svolta nelle funzioni della classe Ricerca.

FormConsigli:
Il componente “Form Consigli” del diagramma dei componenti è stato tradotto nella classe FormConsigli e viene utilizzata esclusivamente dall’UtenteLoggato per impostare le sue personali preferenze sul genere dei contenuti che desidera gli vengano suggeriti.



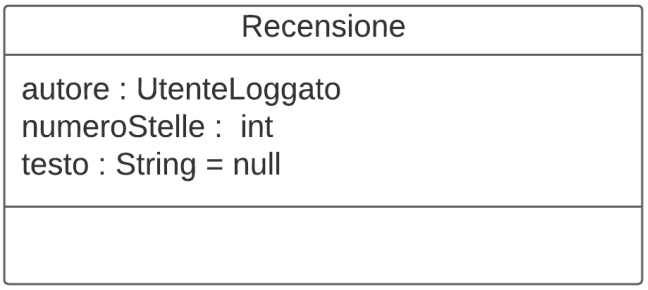
Possiede i seguenti attributi:

- azione : bool
- avventura : bool
- fantasy : bool
- commedia : bool
- romantico : bool
- documentario : bool
- anime : bool

Possiede i seguenti metodi:

- riempi()
  Il metodo reindirizza l’UtenteLoggato ad una nuova pagina dove imposta i valori dei generi che preferisce a true, impostando così i criteri di ricerca per la generazione dei suggerimenti personalizzati tramite il metodo sottostante.
- generaConsigliUtente(generi) : Contenuto [0...*]
  Il metodo crea una serie di contenuti suggeriti e li visualizza sulla pagina profilo dell’UtenteLoggato, effettuando una ricerca filtrata tramite i generi selezionati nel metodo riempi() dall’UtenteLoggato.

Recensione:
Il componente “Recensioni” del diagramma dei componenti è stato tradotto nella classe Recensione e rappresenta il tipo istanziato dal metodo “faRecensione” per valutare un titolo ed, eventualmente, recensirlo.



Possiede i seguenti attributi:

- autore : UtenteLoggato
  Questo attributo contiene un’istanza di tipo UtenteLoggato che esplicita l’autore della recensione.
- numeroStelle : int
  Indica la valutazione che un UtenteLoggato vuole assegnare ad un titolo, selezionando le stelle desiderate.
- testo : String = NULL
  L’attributo, eventualmente, contiene una recensione scritta da un UtenteLoggato.

È stata inoltre creata una classe “fittizia” ListaRecensioni per segnalare la necessità di una collection con all’interno tutte le recensioni di un singolo Contenuto. Questo è indispensabile per il metodo “calcolaMediaPunteggio”.

Contenuto:
La classe Contenuto non proviene da un componente del diagramma dei componenti, ma si è resa necessaria per raggruppare le funzionalità e le caratteristiche degli elementi principali del sito, ossia film e serie tv.

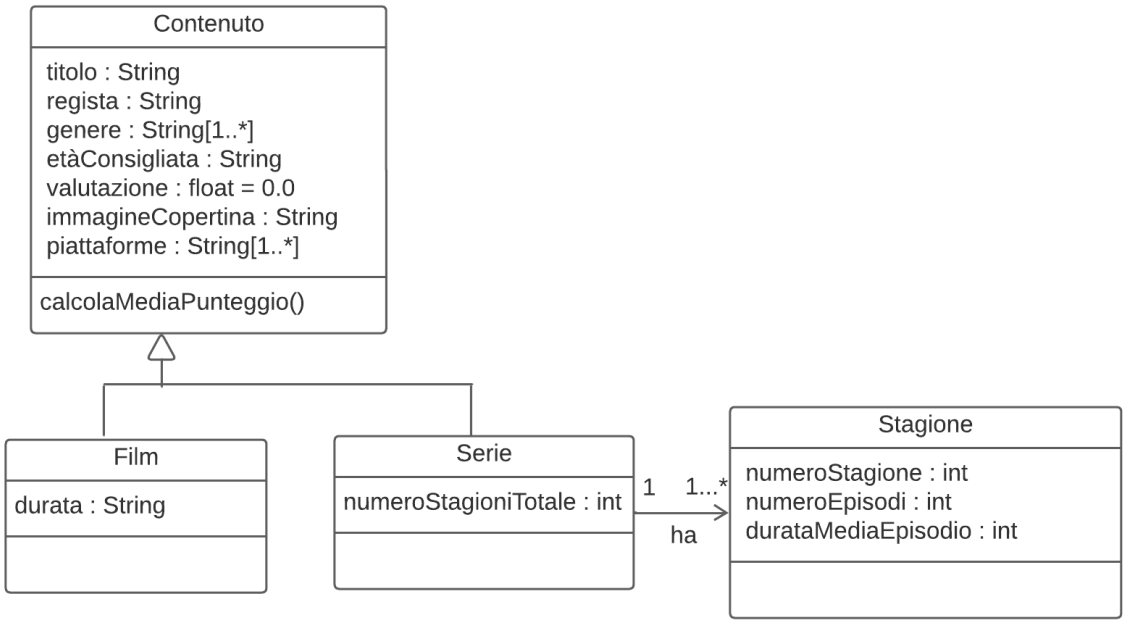
Possiede i seguenti attributi:

- titolo : String
  Contiene il titolo del Contenuto.
- regista : String
  Contiene il nome e cognome del regista del Contenuto.
- genere : String[1..*]
  Contiene un array di String con elencati i vari generi corrispondenti al Contenuto.
- etàConsigliata : String
  Contiene l'età consigliata per il singolo Contenuto.
- valutazione : float = 0.0
  Contiene la valutazione data dalla media del punteggio assegnata dagli utenti al Contenuto.
  Di default il valore è settato a zero, poichè, se non esistono recensioni, l'attributo "valutazione" non può essere di un valore differente.
- immagineCopertina : String
  Contiene il link all'immagine di copertina del Contenuto.
- piattaforme : String[1..*]
  Contiene i nomi delle piattaforme su cui il Contenuto è fruibile.

Possiede i seguenti metodi:

- calcolaMediaPunteggio()
  La funzione calcola l'attributo "valutazione" del Contenuto.

Nel diagramma delle classi troviamo due classi in relazione is-a con la classe Contenuto, poiché un contenuto può essere di tipo Film o di tipo Serie, entrambe sono nuove rispetto al diagramma dei componenti e specificano le differenti caratteristiche di un Contenuto.



Film:
Possiede l'attributo:

- durata : String
  Contiene la durata complessiva del film.

Serie:
Possiede l'attributo:

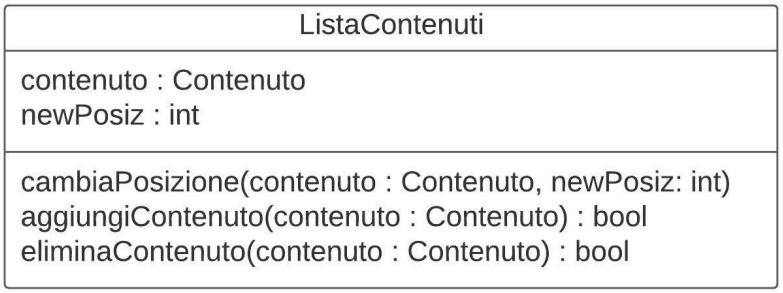
- numeroStagioniTotale : int
  Contiene il numero totale delle stagioni uscite di una determinata serie televisiva.

Stagione:
La classe stagione, non presente nel diagramma del componenti, è indispensabile per differenziare le caratteristiche delle diverse stagioni di una stessa serie.
Possiede i seguenti attributi:

- numeroStagione : int
  Contiene il numero della stagione della serie.
- numeroEpisodi : int
  Contiene il numero totale degli episodi della stagione in questione
- durataMediaEpisodio : int
  Contiene la durata media di un singolo episodio per la stagione in questione, indispensabile per il conteggio delle ore di visione accumulate dal singolo UtenteLoggato.

Nota: quando il Contenuto che viene visualizzato dall'utente è di tipo Serie, nella pagina del Contenuto in questione, la durata che viene visualizzata è sempre il valore corrispondente all'attributo "durataMediaEpisodio" della prima stagione. Invece, per il metodo "calcolaMediaOre" (nella classe ListaContenutiVisti) la media delle ore di visione viene calcolata utilizzando i dati particolari corrispondenti alle varie stagioni.

ListaContenuti:
La classe ListaContenuti raggruppa le principali funzionalità del componente "Gestione Liste" del diagramma dei componenti.



Possiede i seguenti attributi:
- contenuto : Contenuto
  Contiene un elemento di tipo Contenuto su cui si vogliono compiere determinate operazioni.
- newPosiz : int
  Contiene un intero che indica la nuova posizione in cui si desidera spostare il Contenuto.

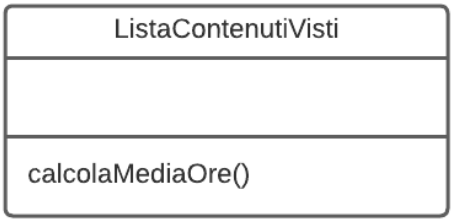
Possiede i seguenti metodi:
- cambiaPosizione(contenuto : Contenuto, newPosiz : int)
  La funzione prende come parametro un "contenuto" e modifica il suo attributo "posizioneInLista" in base al parametro "newPosiz"(che contiene la posizione in cui si desidera spostare il contenuto selezionato).
- aggiungiContenuto(contenuto : Contenuto) : bool
  La funzione aggiunge alla lista il Contenuto passato come parametro.
- eliminaContenuto(contenuto : Contenuto) : bool
  La funzione elimina dalla lista il Contenuto passato come parametro.

Nel diagramma delle classi troviamo tre classi in relazione is-a con la classe ListaContenuti, poiché una lista può essere di tipo ListaContenutiVisti o di tipo ListaContenutiDaVedere di tipo ListaTop10. Le ultime due classi sono vuote, poichè si è deciso di suddivedere in tre tipi di liste differenti le liste di contenuti, dal momento che:
- ListaContenutiVisti ha un metodo che ListaContenutiDaVedere non possiede, benchè lavorino similmente;
- ListaTop10 si distingue dalle due tipologie precedenti poichè la ricerca per aggiungere un contenuto a questa lista avviene automaticamente come ricerca filtra e la relazione con la classe Contenuto ha una cardinalità differente rispetto agli altri tipi di lista.

Inoltre la relazione tra ogni tipo di ListaContenuti e la classe Contenuto viene ulteriormente specificata da alcune association class chiamate, rispettivamente, IntermedioContenutiVisti, IntermedioContenutiDaVedere e IntermedioTop10.
Tutte queste association class possiedono i seguenti attributi:

- posizioneInLista : int
  Per specificare che posizione occupa il singolo contenuto all'interno della lista in cui si trova.
- numeroStagione : int = NULL
  Per specificare la stagione desiderata del singolo contenuto, solo se questo è una serie, all'interno della lista in cui si trova.
  Per i contenuti di tipo Film, questo attributo è settato a NULL.

ListaContenutiVisti:

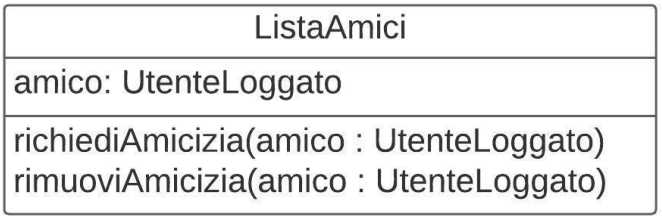


Possiede il metodo:

- calcolaMediaOre()
  La funzione calcola le ore totali di visione di un singolo UtenteLoggato.

Il componente “Gestione Amicizie” del diagramma dei componenti viene suddiviso, al fine di evidenziare le due differenti collection, in due classi: ListaAmici e ListaRichiesteAmicizia.

ListaAmici:
La classe rappresenta la collection degli amici di un singolo UtenteLoggato.



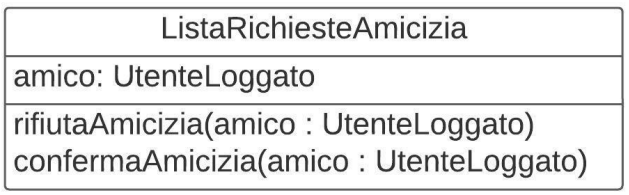
Possiede l’attributo:

- amico : UtenteLoggato
  Contiene un elemento di tipo UtenteLoggato.

Possiede i seguenti metodi:

- richiediAmicizia(amico : UtenteLoggato)
  La funzione invia una richiesta di amicizia all’UtenteLoggato corrispondente al parametro “amico”.
- rimuoviAmicizia(amico : UtenteLoggato)
  La funzione rimuove dalla ListaAmici l’UtenteLoggato corrispondente al parametro “amico”.

ListaRichesteAmicizia:
La classe rappresenta la collection delle richieste di amicizia di un singolo UtenteLoggato.



Possiede l’attributo:

- amico : UtenteLoggato
  Contiene un elemento di tipo UtenteLoggato.

Possiede i seguenti metodi:

- rifiutaAmicizia(amico : UtenteLoggato)
  La funzione rimuove dalla ListaRichiestaAmicizia l’UtenteLoggato corrispondente al parametro “amico”.
- confermaAmicizia(amico : UtenteLoggato)
  La funzione rimuove dalla ListaRichiestaAmicizia l’UtenteLoggato corrispondente al parametro “amico” e lo aggiunge alla ListaAmici.

Si noti che il componente “Gestione db” del diagramma dei componenti non trova una classe corrispondente nel diagramma delle classi. Questo per massimizzare la coesione del sistema e minimizzare l’accoppiamento tra le classi, infatti, le query rivolte al database vengono svolte direttamente all’interno delle funzioni che necessitano di interrogare il database.

Codice in Object Constraint Language:

Di seguito, vengono descritti formalmente operazioni tra classi e/o caratteristiche di determinate istanze in Object Constraint Language (OCL).

Per la classe **UtenteGenerico**:

- Prima della registrazione, la mail e l'username inseriti non devono essere già presenti nella ListaUtentiRegistrati.

  context UtenteGenerico::registrazione(username, mail, password, passwordSupp)
  pre : ListaUtentiRegistrati -> excludes(username)
  pre : ListaUtentiRegistrati -> excludes(mail)

- L'attributo "password", contenente la password creata, e l'attributo "passwordSupp", contenente la password di conferma, devono coincidere.

  context UtenteGenerico::registrazione(username, mail, password, passwordSupp)
  pre: self.password == self.passwordSupp

- Dopo la registrazione, la mail e l'username inseriti devono comparire in ListaUtentiRegistrati.

  context UtenteGenerico::registrazione(username, mail, password, passwordSupp)
  post : ListaUtentiRegistrati -> includes(username)
  post : ListaUtentiRegistrati -> includes(mail)

- Dopo l'avvenuta registrazione, gli attributi "passwordSupp", "CodiceOriginale" e "CodiceInserito" vengono risettati a NULL.

  context UtenteGenerico::registrazione(username, mail, password, passwordSupp)
  post : self.passwordSupp == NULL
  post : self.CodiceOriginale == NULL
  post : self.CodiceInserito == NULL

Per la classe **GestioneCodici**:

- Gli attributi "CodiceOriginale" e "CodiceInserito" devono corrispondere.

  context GestioneCodici::checkCodiceInserito(user : UtenteGenerico) : bool
  pre : self.user.CodiceOriginale == self.user.CodiceInserito

- Affinchè l'attributo CodiceInserito sia valido, non devono essere passati più di 10 minuti da quando è stato creato il codice.

  context GestioneCodici::checkCodiceInserito(user : UtenteGenerico) : bool
  pre : (self.oraCheck-self.oraCreazione) <= 10 min

- Gli attributi “CodiceOriginale” e “CodiceInserito” vengono resettati a NULL.

  context GestioneCodici::checkCodiceInserito(user : UtenteGenerico) : bool
  post : user.CodiceOriginale == NULL
  post : user.CodiceInserito == NULL

Per la classe **Autenticazione**:

- Per il recupero credenziali, la nuova password inserita non può essere uguale alla vecchia.

  context Autenticazione::settaggioNuovaPassword(user : UtenteGenerico)
  pre: self.user.password != self.user.passwordSupp

- Dopo il recupero credenziali avvenuto con successo, il contenuto dell’attributo “passwordSupp” va a sovrascrivere il contenuto dell’attributo “password” e l’attributo “passwordSupp” viene resettato a NULL.

  context Autenticazione::settaggioNuovaPassword(user : UtenteGenerico)
  post : self.user.passwordSupp == NULL

Per la classe **UtenteLoggato**:

- Un utente può effettuare solo una recensione per ogni contenuto.

  context UtenteLoggato::faRecensione()
  pre : ListaRecensioni -> excludes(autore)

- Per aprire la chat con un utente, è necessario che questo utente “amico” sia presente nella ListaAmici

  context UtenteLoggato::apriChat(UtenteLoggato)
  pre : ListaAmici -> includes (amico)

Per la classe **Recensione**:

- Si possono assegnare da 1 a 5 stelle nella valutazione.

  context Recensione inv : self.numeroStelle >= 1 && self.numeroStelle <=5

- Un utente non può fare una recensione senza valutare il contenuto in numero di stelle.

  context UtenteLoggato::faRecensione()
  post : Recensione.numeroStelle != NULL

Per la classe **Contenuto**:

- La valutazione di un contenuto è data dalla media dei punteggi (assegnati in numeroStelle) dei singoli utenti nelle recensioni, dunque può variare da 1 a 5 stelle.

  context Contenuto inv : self.valutazione >= 0 && self.valutazione <=5

Per la classe **ListaTop10**:

- Per aggiungere un contenuto alla collection ListaTop10, l'attributo "contenuto" deve essere presente nella collection ListaContenutiVisti.

  context ListaTop10::aggiungiContenuto(contenuto : Contenuto)
  pre : ListaContenutiVisti -> includes(contenuto)

Per la classe **ListaContenutiDaVedere**:

- Dopo aver aggiunto un contenuto alla collection ListaContenutiVisti, l'attributo "contenuto" non deve essere presente nella ListaContenutiDaVedere.

  context ListaContenutiVisti::aggiungiContenuto(contenuto : Contenuto)
  post : ListaContenutiDaVedere -> excludes(contenuto)

Per la classe **ListaContenutiDaVedere**:

- Per aggiungere un contenuto alla collection ListaContenutiDaVedere, l'attributo "contenuto" non deve essere presente nella ListaContenutiVisti.

  context ListaContenutiDaVedere::aggiungiContenuto(contenuto : Contenuto)
  pre : ListaContenutiVisti -> excludes(contenuto)

Per la classe **Stagione**:

- L'attributo "numeroStagione" dev'essere minore o uguale all'attributo "numeroStagioniTotale" della classe Serie, per un determinato Contenuto.

  context Stagione inv : self.numeroStagione <= Serie.numeroStagioniTotale

Per la classe **ListaRichiesteAmicizia**:

- Per confermare un' amicizia, prima dell'operazione, l'utente contenuto nell'attributo "amico" non deve essere già presente nella ListaAmici, viceversa, deve essere presente nella ListaRichiesteAmicizia., viceversa, dopo la conferma dell'amicizia, l'utente "amico" deve essere incluso nella ListaAmici.

  context ListaRichiesteAmicizia::confermaAmicizia(amico : UtenteLoggato)
  pre : ListaAmici -> excludes (amico)
  pre : ListaRichiestaAmicizia -> includes (amico)

- Per confermare un' amicizia, dopo l'operazione, l'utente contenuto nell'attributo "amico" non deve più essere nella ListaRichiesteAmicizia, viceversa, deve essere presente nella ListaAmici.

  context ListaRichiesteAmicizia::confermaAmicizia(amico : UtenteLoggato)
  post : ListaRichiestaAmicizia -> excludes (amico)
  post : ListaAmici -> includes (amico)

Per la classe **ListaAmici**:

- Per effettuare una richiesta di amicizia, due utenti non possono essere già amici.

  context ListaAmici::richiediAmicizia(amico : UtenteLoggato)
  pre : ListaAmici -> excludes (amico)

- Per rimuovere un utente dalla ListaAmici, l'utente "amico" deve essere presente nella ListaAmici, viceversa, dopo la rimozione l'utente "amico" non deve essere presente nella ListaAmici.

  context ListaAmici::rimuoviAmicizia(amico : UtenteLoggato)
  pre : ListaAmici -> includes (amico)
  post : ListaAmici -> excludes (amico)