MyTalk

Sofware di comunicazione tra utenti senza requisiti di installazione

clockworkTeam7@gmail.com

Definizione di Prodotto

v 1.0

Informazioni sul documento

Nome documento	Definizione di Prodotto
Versione documento	v 1.0
Data creazione	2013/02/20
Data ultima modifica	2013/05/20
Uso documento	Esterno
Redazione	*Periodo1:* Ceseracciu Marco Gavagnin Jessica Palmisano Maria Antonietta*Periodo2:* Ceseracciu Marco Gavagnin Jessica
Verifica	Furlan Valentino La Bruna Agostino Zohouri Haghian Pardis
Approvazione	Bain Giacomo
Lista distribuzione	gruppo *Clockwork* Zucchetti SPA Prof. Tullio Vardanega

Sommario

Il presente documento intende descrivere in modo dettagliato tutte le componenti del Progetto **MyTalk** e i criteri con le quali interagiscono fra loro. Per ogni componente sono illustrati gli attributi con il loro significato e i metodi con il loro comportamento.

Diario delle modifiche

Autore	Modifica	Data	Versione
Bain Giacomo	Approvazione documento	2013/05/20	v 1.0
Furlan Valentino	Verifica generale del documento	2013/05/19	v 0.16
La Bruna Agostino	Verifica capitoli 4 e 5	2013/05/17	v 0.15
Zohouri Haghian Pardis	Verifica dal capitolo 1 al capitolo 3	2013/05/16 v 0.14	
Gavagnin Jessica	Modifiche apportate al capitolo 3	2013/05/14	v 0.13
Ceseracciu Marco	Modifiche apportate al capitolo 4	2013/05/013	v 0.12
Gavagnin Jessica	Stesura capitolo 5	2013/04/06	v 0.11
Ceseracciu Marco	Stesura finale del capitolo 3	2013/04/06	v 0.10
Palmisano Maria Antonietta	Stesura finale del capitolo 4	2013/04/06	v 0.9
Ceseracciu Marco	Stesura del capitolo 4	2013/04/05	v 0.8
Gavagnin Jessica	Stesura del capitolo 3, package usermanager	2013/03/29	v 0.7
Gavagnin Jessica	Stesura del capitolo 3, package functionmanager	2013/03/27	v 0.6
Ceseracciu Marco	Stesura del capitolo 3, package view	2013/03/26	v 0.5
Gavagnin Jessica	Stesura del capitolo 3, package transfer	2013/03/21	v 0.4
Gavagnin Jessica	Stesura del capitolo 3, package usermanager	2013/03/20	v 0.3
Gavagnin Jessica	Creazione e stesura del capitolo 3, package shared e dao	2013/03/18	v 0.2
Palmisano Maria Antonietta	Creazione documento, stesura sezione Introduzione	2013/02/20	v 0.1

Indice

Elenco delle figure

Elenco delle tabelle

1 Introduzione

1.1 Scopo del documento

Il presente documento descrive tutte le componenti del sistema e il modo in cui esse collaborano. Per ogni componente vengono illustrati gli attributi con il loro significato e i medoti con il loro comportamento. Lo scopo principale del documento è quello di fornire ai programmatori una solida e precisa guida alla codifica, in modo che essi possano lavorare in modo autonomo attenendosi alle scelte progettuali ed evitando soluzioni personali ed improvvisate.

1.2 Scopo del prodotto

Il prodotto denominato **MyTalk** si propone di fornire un software per un sistema di comunicazione audio e video tra utenti. Lo scopo del progetto è poter comunicare con altri utenti tramite il browser, utilizzando solo componenti standard, senza dover installare plugin o programmi esterni. L'utilizzatore dovrà poter chiamare un altro utente, iniziare la comunicazione sia audio che video, svolgere la chiamata e terminare la chiamata ottenendo delle statistiche sull'attività.

1.3 Glossario

Per evitare ambiguità i termini tecnici o di uso non comune, vengono evidenziati, alla loro prima occorrenza nel documento, tramite sottolineatura. Le definizioni di questi termini sono riportate nel documento in allegato `Glossario_v2.0.pdf`.

1.4 Riferimenti

1.4.1 Normativi

- Capitolato d'Appalto: **MyTalk**, rilasciato da Zucchetti SPA, reperibile all'indirizzo: `http://www.math.unipd.it/~tullio/IS-1/2012/Progetto/C1.pdf`
- Norme di Progetto (allegato `Norme_di_Progetto_v3.0.pdf`)

1.4.2 Informativi

- Analisi dei Requisiti (allegato `Analisi_dei_Requisiti_v3.0.pdf`)
- Specifica Tecnica (allegato `Specifica_Tecnica_v2.0.pdf`)

2 Standard di progetto

2.1 Standard di progettazione architetturale

Per gli standard di progettazione architetturale si veda il documento Specifica Tecnica (allegato `Specifica_Tecnica_v2.0.pdf`) e il documento Norme di Progetto (allegato `Norme_di_Progetto_v3.0.pdf`).

2.2 Standard di documentazione del codice

Per gli standard di documentazione del codice utilizzati si faccia riferimento al documento relativo alle Norme di Progetto (allegato `Norme_di_Progetto_v3.0.pdf`).

2.3 Standard di denominazione di entità e relazioni

Tutti gli elementi definiti, siano essi package, classi, metodi o attributi, devono avere denominazioni chiare ed autoesplicative, utilizzando nomi quanto più brevi possibili. Si faccia riferimento alle Norme di Progetto per ulteriori informazioni (allegato `Norme_di_Progetto_v3.0.pdf`).

2.4 Standard di programmazione

Gli standard di programmazione sono stati definiti nel documento Norme di Progetto. Si rimanda ad esso per l'elenco delle regole da seguire in fase di scrittura del codice (allegato `Norme_di_Progetto_v3.0.pdf`).

2.5 Strumenti di lavoro

Gli strumenti utilizzabili per per la realizzazione dei prodotti sono tutti e soli quelli delineati nelle Norme di Progetto (allegato `Norme_di_Progetto_v3.0.pdf`).

3 Specifica Server

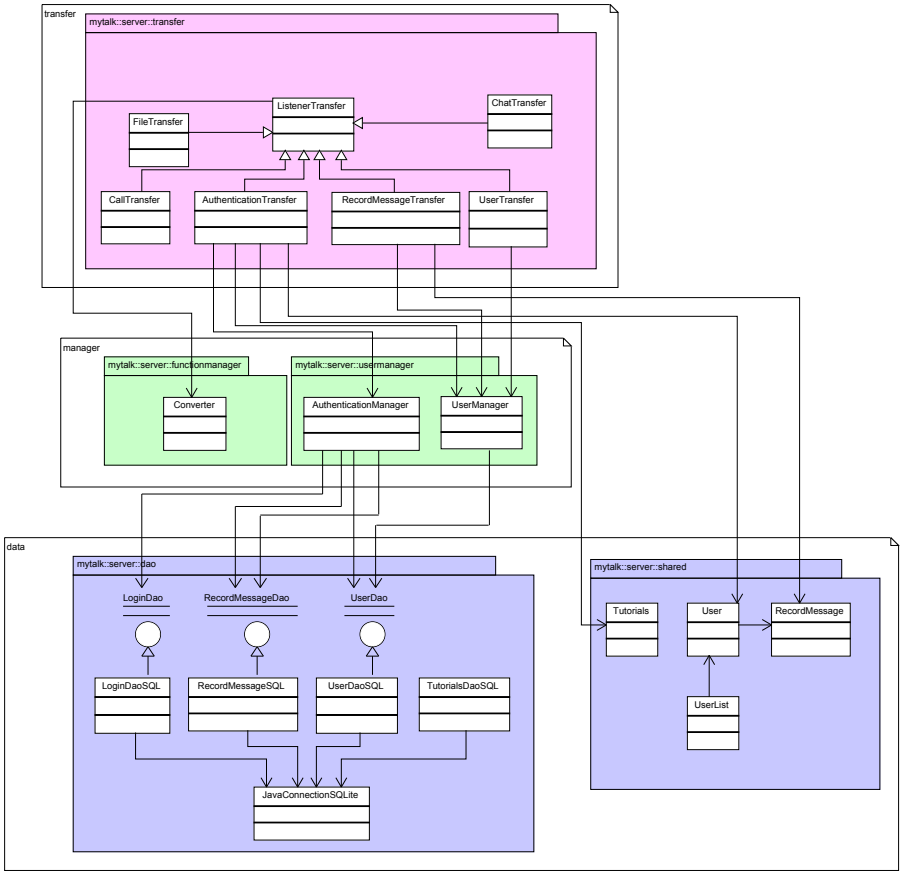

Figura 1: Architettura del server

Il server contiene i seguenti package:

- `mytalk.server.dao` (vedasi sezione 3.1)
- `mytalk.server.shared` (vedasi sezione 3.2)
- `mytalk.server.usermanager` (vedasi sezione 3.3)
- `mytalk.server.functionmanager` (vedasi sezione 3.4)
- `mytalk.server.transfer` (vedasi sezione 3.5)

3.1 Package dao

Il package `mytalk.server.dao` contiene tutte le classi che si occupano di eseguire le operazioni sul database.

Relazioni d'uso con altri moduli

- **Utilizza:**
  - `mytalk.server.shared`
- **Viene utilizzato da:**
  - `mytalk.server.usermanager`

Costituito da:

- `mytalk.server.dao.JavaConnectionSQLite` (vedasi sezione3.1.1)
- `mytalk.server.dao.LoginDao` (vedasi sezione3.1.2)
- `mytalk.server.dao.LoginDaoSQL` (vedasi sezione3.1.3)
- `mytalk.server.dao.RecordMessageDao` (vedasi sezione3.1.4)
- `mytalk.server.dao.RecordMessageDaoSQL` (vedasi sezione3.1.5)
- `mytalk.server.dao.TutorialsDaoSQL` (vedasi sezione 3.1.6)
- `mytalk.server.dao.UserDao` (vedasi sezione3.1.7)
- `mytalk.server.dao.UserDaoSQL` (vedasi sezione3.1.8)

3.1.1 `mytalk.server.dao.JavaConnectionSQLite`

La classe JavaConnectionSQLite contiene tutti i metodi per l'esecuzione di query all'interno del database

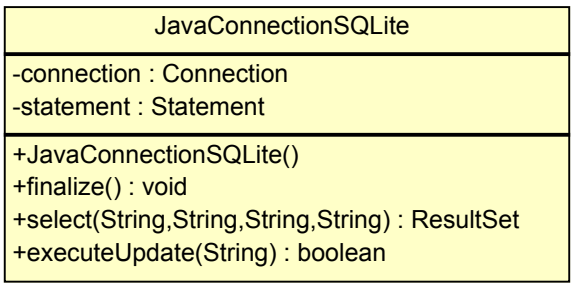

Figura 2: Classe JavaConnectionSQLite

Relazioni d'uso con altri moduli

- **Viene utilizzata da:**
  - `mytalk.server.dao.LoginDaoSQL`
  - `mytalk.server.dao.RecordMessageDaoSQL`
  - `mytalk.server.dao.TutorialsDaoSQL`
  - `mytalk.server.dao.UserDaoSQL`

Attributi privati

- **Connection connection:** conterrà il riferimento alla connessione al database
- **Statement statement:** conterrà il riferimento alla risorsa impiegata per eseguire query sul database

Metodi pubblici

- **JavaConnectionSQLite():** costruttore della classe che si occuperà di creare la connessione al database. In caso di fallimento della connessione, verrà lanciata un eccezione che stamperà a video l'errore riscontrato
- **void finalize():** metodo che fungerà da distruttore della classe JavaConnectionSQLite. Prima di distruggere una istanza, la connessione al database dovrà essere chiusa.
- **ResultSet select(String, String, String, String):** metodo che ritornerà una tupla di valori, della tabella indicata, che rispettano i parametri di controllo indicati. Quindi eseguirà la query sul database utilizzando i parametri passati:
  - *table:* rappresenterà il nome della tabella su cui eseguire la query
  - *column:* rappresenterà le colonne che saranno selezionate
  - *condition:* sarà la condizione principale applicata per effettuare la selezione
  - *extra:* rappresenterà le condizioni extra, se necessarie

  Se la funzione di esecuzione della query non lancerà alcuna eccezione, il metodo ritornerà un *ResultSet*
- **boolean executeUpdate(String):** metodo che si occuperà di eseguire la query passata nel database, aggiornandolo. Se la query verrà conclusa con successo il metodo ritornerà *true*, altrimenti ritornerà *false*

3.1.2 `mytalk.server.dao.LoginDao`

L'interfaccia LoginDao gestisce tutti i dati che riguardano i tentativi di connessione e disconnessione al sistema. Fornisce l'interfaccia minima necessaria a tutte le classi derivate che dovranno offrire tale servizio

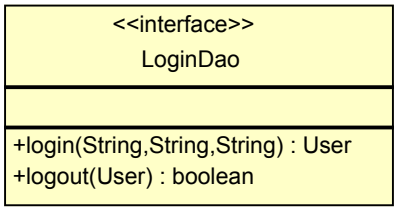

Figura 3: Interfaccia LoginDao

Relazioni d'uso con altri moduli

- **Utilizza:**
  - `mytalk.server.shared.User`
- **Viene implementata da:**
  - `mytalk.server.dao.LoginDaoSQL`
- **Viene utilizzata da:**
  - `mytalk.server.usermanager.AuthenticationManager`

Metodi pubblici

- **User login(String, String, String):** metodo che controllerà la correttezza dei dati inseriti da un utente per connettersi al sistema, e in caso affermativo reimposterà l'IP
- **boolean logout(User):** metodo che effettuerà la disconnessione dal sistema

3.1.3 `mytalk.server.dao.LoginDaoSQL`

La classe LoginDaoSQL si occupa di gestire tutti i dati riguardanti i tentativi di autenticazione all'interno del sistema.

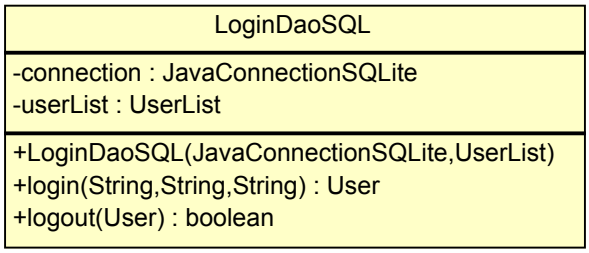

Figura 4: Classe LoginDaoSQL

Relazioni d'uso con altri moduli

- **Utilizza:**
  - `mytalk.server.dao.JavaConnectionSQLite`
  - `mytalk.server.shared.UserList`
  - `mytalk.server.shared.User`
- **Implementa:**
  - `mytalk.server.dao.LoginDao`
- **Viene utilizzata da:**
  - `mytalk.server.usermanager.AuthenticationManager`

Attributi privati

- **JavaConnectionSQLite connection:** conterrà il riferimento all'istanza della classe JavaConnectionSQLite
- **UserList userList:** conterrà il riferimento alla lista degli utenti registrati nel database

Metodi pubblici

- **LoginDaoSQL(JavaConnectionSQLite, UserList):** costruttore della classe che dovrà impostare il valore degli attributi secondo il valore dei parametri passati
- **User login(Stringe, String, String):** metodo che gestirà l'operazione di *login* dell'utente. Dovrà controllare la correttezza dello username e della password inseriti. Crearà una query per verificare se esiste un utente con lo *username* ricevuto e che la password associata a tale utente sia uguale a quella ricevuta. Se non dovesse esistere tale parametro restituirà *null*, in caso contrario aggiornerà l'indirizzo IP dell'utente che vuole effettuare la connessione

- **boolean logout(User):** metodo che gestirà l'operazione di *logout* dell'utente aggiornando l'indirizzo IP a *0*, sia nell'istanza dell'utente contenuta nel package `mytalk.server.shared`, sia in quello presente nel database

3.1.4 `mytalk.server.dao.RecordMessageDao`

L'interfaccia RecordMessageDao gestisce tutti i dati che riguardano i messaggi differiti inviati ai vari utenti. Fornisce l'interfaccia minima necessaria a tutte le classi derivate che dovranno offrire tale servizio

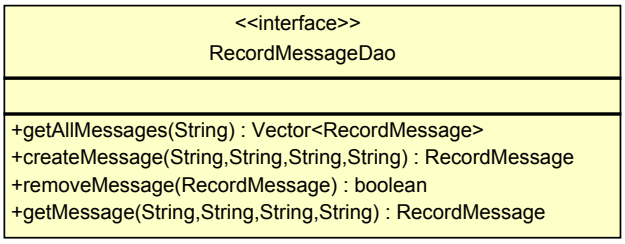

Figura 5: Interfaccia RecordMessageDao

Relazione d'uso con altri moduli

- **Utilizza:**
  - `mytalk.server.shared.RecordMessage`
- **Viene implementata da:**
  - `mytalk.server.dao.RecordMessageDaoSQL`
- **Viene utilizzata da:**
  - `mytalk.server.usermanager.UserManager`

Metodi pubblici

- **Vector<RecordMessage> getAllMessages(String):** metodo che ritornerà il vettore dei messaggi differiti che sono stati mandati ad un utente
- **RecordMessage createMessage(String, String, String, String):** metodo che aggiungerà un messaggio differito nel database
- **RecordMessage getMessage(String, String, String, String):** metodo che restituirà un dato messaggio differito presente nel database
- **boolean removeMessage(RecordMessage):** metodo che eliminerà un messaggio differito dal database

3.1.5 `mytalk.server.dao.RecordMessageDaoSQL`

La classe RecordMessageDaoSQL si occupa di gestire tutti i dati riguardanti i messaggi differiti.

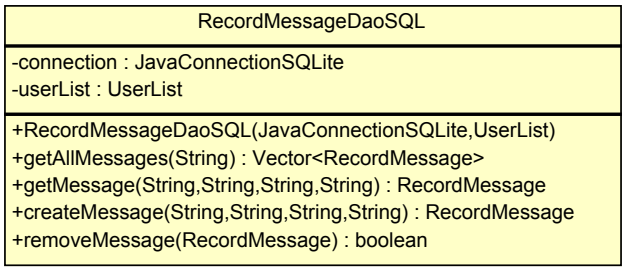

Figura 6: Classe RecordMessageDaoSQLite

Relazione d'uso con altri moduli

- **Utilizza:**
  - `mytalk.server.dao.JavaConnectionSQLite`
  - `mytalk.server.shared.RecordMessage`
  - `mytalk.server.shared.User`
  - `mytalk.server.shared.UserList`
- **Implementata:**
  - `mytalk.server.dao.RecordMessageDao`
- **Viene utilizzata da:**
  - `mytalk.server.usermanager.UserManager`

Attributi privati

- **JavaConnectionSQLite connection:** conterrà il riferimento all'istanza della classe JavaConnectionSQLite contenente i riferimenti al server
- **UserList userList:** conterrà il riferimento alla lista degli utenti registrati nel database

Metodi pubblici

- **RecordMessageDaoSQL(JavaConnectionSQLite, UserList):** costruttore della classe che dovrà impostare il valore degli attributi secondo il valore dei parametri passati
- **Vector<RecordMessage> getAllMessages(String):** metodo che ritornerà il vettore dei messaggi differiti che sono stati mandati ad un utente. Il metodo dovrà verificare che esista realmente un utente associato allo *username* ricevuto, se esisterà creerà la query da effettuare per trovare tutti i messaggi che come destinatario avranno tale username. Se la query restituirà almeno un messaggio differito, lo inserirà nel vettore dell'utente.
- **RecordMessage getMessage(String sender, String, String, String):** metodo che ritornerà il messaggio differito avente le caratteristiche richieste. Il metodo inizialmente troverà tutti i messaggi del destinatario richiesto, invocando il metodo gatAllMessages(String), successivamente ricercherà tra quei messaggi quello corrispondente alle richieste e lo restituirà. Nel caso che il messaggio non esista verrà restituito *null*
- **RecordMessage createMessage(String, String, String, String):** metodo che creerà un messaggio differito nel database. Dovrà verificare che l'utente destinatario esista nel database. Successivamente dovrà creare la query per aggiungere nel database il messaggio. Se l'aggiunta avverrà con successo, verrà aggiornata anche l'istanza dell'utente nel package `mytalk.server.shared.User.` Il metodo ritornerà il messaggio creato, che sarà *null* nel caso in cui la creazione non sia andata a buon fine
- **boolean removeMessage(RecordMessage):** metodo che eliminerà un messaggio differito dal database. Il metodo dovrà creare la query per la cancellazione del messaggio differito nel database. Successivamente verrà aggiornata anche l'istanza dell'utente nel package `mytalk.server.shared.User.` Nel caso l'eliminazione avvenga con successo il metodo ritornerà *true*, in caso contrario ritornerà *false*

3.1.6 `mytalk.server.dao.TutorialsDaoSQL`

La classe TutorialsDaoSQL si occupa di gestire i tutorial presenti nel database

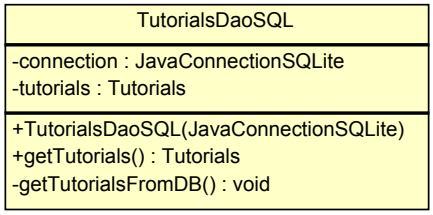

Figura 7: Classe TutorialsDaoSQLite

Relazione d'uso con altri moduli

- **Utilizza:**
  - `mytalk.server.dao.JavaConnectionSQLite`
  - `mytalk.server.shared.Tutorials`

Attributi privati

- **JavaConnectionSQLite connection:** conterrà il riferimento all'istanza della classe JavaConnectionSQLite contenente i riferimenti al server
- **Tutorials tutorials:** conterrà il riferimento alla lista dei tutorials presenti nel server

Metodi pubblici

- **TutorialsDaoSQL(JavaConnectionSQLite):** costruttore della classe che dovrà impostare il valore degli attributi secondo il valore dei parametri passati
- **Tutorials getTutorials():** metodo che restituirà la lista di tutorials presenti in `mytalk.server.shared.Tutorials`

Metodi privati

- **void getTutorialsFromDB():** metodo che inserirà in `mytalk.server.shared.Tutorials` i tutorials presenti nel database

3.1.7 `mytalk.server.dao.UserDao`

L'interfaccia UserDao gestisce tutti i dati che riguardano gli utenti. Fornisce l'interfaccia minima necessaria a tutte le classi derivate che dovranno offrire tale servizio

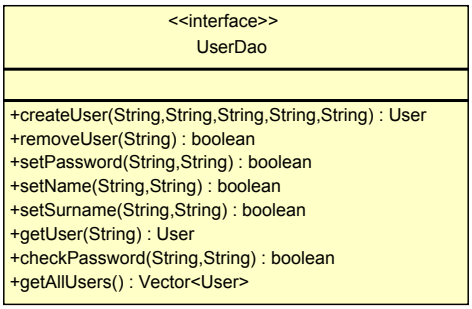

Figura 8: Interfaccia UserDao

Relazione d'uso con altri moduli

- **La classe utilizza:**
  - `mytalk.server.shared.User`
- **Viene implementata da:**
  - `mytalk.server.dao.UserDaoSQL`
- **Viene utilizzata da:**
  - `mytalk.server.usermanager.UserManager`
  - `mytalk.server.usermanager.AuthenticationManager`

Metodi pubblici

- **User createUser(String, String, String, String, String):** metodo che creerà un utente nel database
- **boolean removeUser(String):** metodo che eliminerà un utente dal database
- **boolean setPassword(String, String):** metodo che imposterà la password dell'utente corrispondente ad *username* con il nuovo valore
- **boolean setName(String, String):** metodo che imposterà il nome dell'utente corrispondente ad *username* con il nuovo valore
- **boolean setSurname(String, String):** metodo che imposterà il cognome dell'utente corrispondente ad *username* con il nuovo valore
- **User getUser(String)**: metodo che restituirà l'utente avente username corrispondente a quello dato

- **boolean checkPassword(String, String):** metodo che controllerà la correttezza dei parametri dati
- **Vector<User> getAllUsers():** metodo che restituirà la lista di tutti gli utenti presenti nel database

3.1.8 `mytalk.server.dao.UserDaoSQL`

La classe UserDaoSQL si occupa di gestire tutti i dati che riguardano gli utenti.

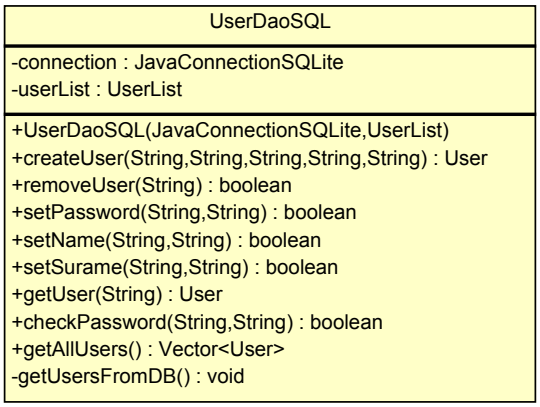

Figura 9: Classe UserDaoSQL

Relazioni d'uso con altri moduli

- **Utilizza:**
  - `mytalk.server.dao.JavaConnectionSQLite`
  - `mytalk.server.shared.UserList`
  - `mytalk.server.shared.User`
- **Implementa**
  - `mytalk.server.dao.UserDao`
- **Viene utilizzata da:**
  - `mytalk.server.usermanager.UserManager`
  - `mytalk.server.usermanager.AuthenticationManager`

Attributi privati

- **JavaConnectionSQLite connection:** conterrà il riferimento all'istanza della classe JavaConnectionSQLite contente i riferimenti al server
- **UserList userList:** conterrà il riferimento alla lista degli utenti registrati nel database

Metodi pubblici

- **UserDaoSQL(JavaConnectionSQLite, UserList):** costruttore della classe che dovrà impostare il valore degli attributi secondo il valore dei parametri passati
- **User createUser(String, String, String, String, String):** metodo che creerà un utente con i parametri passati[1]
- **boolean removeUser(String):** metodo che eliminerà un utente dal database. Se l'eliminazione sarà andata a buon fine ritornerà *true*, altrimenti ritornerà *false*
- **boolean setPassword(String, String):** metodo che imposterà la password dell'utente con il nuovo valore passato. Se il cambiamento sarà andato a buon fine, il metodo ritornerà *true*, altrimenti ritornerà *false*
- **boolean setName(String, String):** metodo che imposterà il nome dell'utente con il nuovo valore passato. Se il cambiamento sarà andato a buon fine, si aggiornerà anche l'istanza dell'utente presente in `mytalk.server.shared.User` ed il metodo ritornerà *true*, altrimenti ritornerà *false*
- **boolean setSurname(String, String):** metodo che imposterà il cognome dell'utente con il nuovo valore. Se il cambiamento sarà andato a buon fine, si aggiornerà anche l'istanza dell'utente presente in `mytalk.server.shared.User` ed il metodo ritornerà *true*, altrimenti ritornerà *false*
- **User getUser(String)**: metodo che restituirà l'utente, avente username corrispondente a quello dato, presente in `mytalk.server.shared.UserList`
- **boolean checkPassword(String, String):** metodo che controllerà nel database la corrispondenza dello username passato con la password passata
- **Vector<User> getAllUsers():** metodo che restituirà gli utenti presenti in `mytalk.server.shared.User`

Metodi privati

- **void getUsersFromDB():** metodo che inserirà in `mytalk.server.shared.userList` tutti gli utenti presenti nel database

[1] L'esistenza di un utente con lo username uguale a quello passato come parametro, verrà accertata nella classe `mytalk.server.dao.shared.UserList`.

3.2 Package shared

Il package shared contiene le classi contenenti le informazioni condivise fra i vari strati del server

Relazioni d'uso con altri moduli

- **Il package è utilizzato da:**
  - `mytalk.server.dao`
  - `mytalk.server.usermanager`
  - `mytalk.server.functionmanager`
  - `mytalk.server.transfer`

Costituito dalle classi:

- `mytalk.server.shared.RecordMessage` (vedasi sezione 3.2.1)
- `mytalk.server.shared.User` (vedasi sezione 3.2.3)
- `mytalk.server.shared.UserList` (vedasi sezione 3.2.4)
- `mytalk.server.shared.Tutorials` (vedasi sezione 3.2.2)

3.2.1 mytalk.server.shared.RecordMessage

La classe RecordMessage farà da contenitore per i dati riguardanti i messaggi differiti inseriti all'interno del database

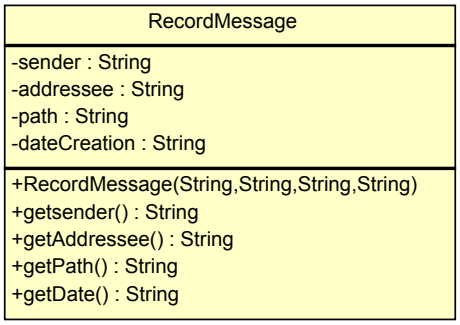

Figura 10: Classe RecordMessage

Relazioni d'uso con altri moduli

- **Viene utilizzata da:**
  - `mytalk.server.shared.User`
  - `mytalk.server.dao.RecordMessageDao`
  - `mytalk.server.dao.RecordMessageDaoSQL`
  - `mytalk.server.functionmanager.Converter`
  - `mytalk.server.usermanager.UserManager`
  - `mytalk.server.transfer.RecordMessageTransfer`

Attributi privati:

- **String sender:** rappresenta il mittente, colui che crea il messaggio
- **String addressee:** rappresenta il destinatario, colui che riceve il messaggio
- **String path:** rappresenta l'indirizzo dove si trova il messaggio differito all'interno del server
- **String dateCreation:** rappresenta la data di creazione del messaggio

Metodi pubblici:

- **RecordMessage(String sender, String addressee, String path, String dateCreation):** costruttore che imposta i valori degli attributi secondo i valori dei parametri passati
- **String getSender():** ritorna il mittente del messaggio
- **String getAddressee():** ritorna il destinatario del messaggio
- **String getPath():** ritorna l'indirizzo dove si trova il messaggio differito all'interno del server
- **String getDate():** ritorna la data di creazione del messaggio

3.2.2 mytalk.server.shared.Tutorials

La classe Tutorials rappresenterà la lista dei tutorials presenti nel server

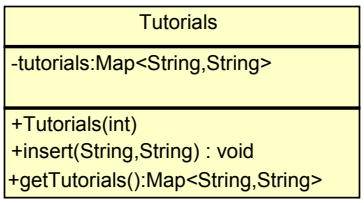

Figura 11: Classe Tutorials

Relazioni d'uso con altri moduli

- **Viene utilizzata da:**
  - `mytalk.server.dao.TutorialsDao`
  - `mytalk.server.transfer.AuthenticationTransfer`

Attributi privati

- **Map<String, String> tutorials:** rappresenta la lista dei tutorials, come chiave di identificazione si userà il titolo del tutorial

Metodi pubblici

- **Tutorials(int num):** costruttore che inizializza la lista di tutorials dellla dimensione passatagli dal parametro
- **void insert(String title, String url):** metodo che inserisce un nuovo tutorial nella lista
- **Map<String, String> getTutorials():** metodo che restituisce la lista di tutorial

3.2.3 mytalk.server.shared.User

La classe User farà da contenitore per i dati riguardanti gli utenti presenti nel database

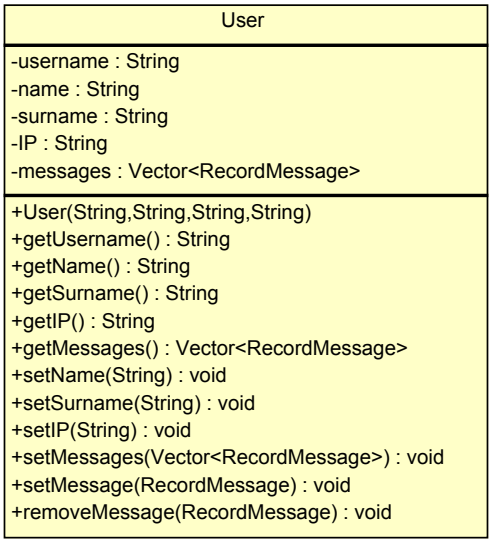

Figura 12: Classe User

Relazioni d'uso con altri moduli

- **Viene utilizzata da:**
  - `mytalk.server.shared.UserList`
  - `mytalk.server.dao.LoginDao`
  - `mytalk.server.dao.LoginDaoSQL`
  - `mytalk.server.dao.RecordMessageDaoSQL`
  - `mytalk.server.dao.UserDao`
  - `mytalk.server.dao.UserDaoSQL`
  - `mytalk.server.functionmanager.Converter`
  - `mytalk.server.usermanager.AuthenticationManager`
  - `mytalk.server.usermanager.UserManager`
  - `mytalk.server.transfer.AuthenticationTransfer`
  - `mytalk.server.transfer.UserTransfer`
- **Utilizza**
  - `mytalk.server.shared.RecordMessage`

Attributi privati

- **String username:** rappresenta lo username dell'utente
- **String name:** rappresenta il nome dell'utente
- **String surname:** rappresenta il cognome dell'utente
- **String IP:** rappresenta l'indirizzo IP dell'utente[2]
- **Vector<RecordMessage> messages:** rappresenta tutti i RecordMessage inviati all'utente in questione mentre questo non è in linea

Metodi pubblici

- **User(String username, String name, String surname, String IP):** costruttore che imposta i valori degli attributi secondo il valore dei parametri passati
- **String getUsername()**: metodo che ritorna lo username dell'utente
- **String getName()**: metodo che ritorna il nome dell'utente
- **String getSurname()**: metodo che ritorna il cognome dell'utente

[2]Se la stringa è uguale a 0 significa che l'utente in questione non è in linea, se è diverso l'utente è in linea.

- **String getIP()**: metodo che ritorna l'indirizzo IP dell'utente
- **void setName(String name)**: metodo che imposta il nome dell'utente con il nuovo valore
- **void setSurname(String surname)**: metodo che imposta il cognome dell'utente con il nuovo valore
- **void setIP(String IP)**: metodo che imposta l'indirizzo IP dell'utente con il nuovo valore
- **Vector<RecordMessage> getMessages()**: metodo che ritorna il vettore contenente i messaggi differiti
- **void setMessages(Vector<RecordMessage> messagesNew)**: metodo che cambia la lista dei messaggi differiti inviati all'utente in questione
- **void setMessage(RecordMessage message)**: metodo che aggiunge un messaggio differito nel vettore messages
- **void removeMessage(RecordMessage message)**: metodo che rimuove un messaggio differito dal vettore dei messaggi

3.2.4 mytalk.server.shared.UserList

La classe UserList rappresenterà la lista degli utenti registrati al server

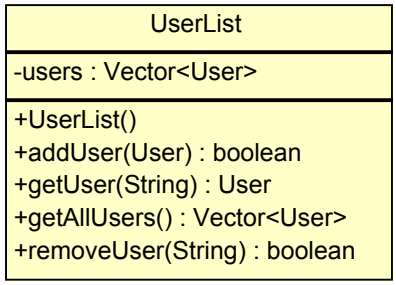

Figura 13: Classe UserList

Relazioni d'uso con altri moduli

- **La classe è utilizzata da:**
  - `mytalk.server.dao.LoginDaoSQL`
  - `mytalk.server.dao.RecordMessageDaoSQL`
  - `mytalk.server.dao.UserDaoSQL`

- **Utilizza**
  - `mytalk.server.shared.User`

Attributi privati

- **Vector<User> users:** rappresenta la lista degli utenti

Metodi pubblici

- **UserList():** costruttore della classe che inizializza il vettore
- **boolean addUser(User user):** metodo che inserisce un nuovo utente nella lista degli utenti. L'utente viene inserito nel vettore solo nel caso in cui non esistano altri utenti che hanno tale username
- **User getUser(String username):** metodo che restituisce l'utente corrispondente ad un dato username
- **Vector<User> getAllUsers():** metodo che restituisce tutta la lista di utenti
- **boolean removeUser(String username):** metodo che elimina un utente dal vettore di utenti

3.3 Package usermanager

Lo strato manager del server si occupa delle funzionalità logiche del server

Relazioni d'uso con altri moduli

- **Utilizza:**
  - `mytalk.server.shared`
  - `mytalk.server.dao`
- **Viene utilizzato da:**
  - `mytalk.server.transfer`

Costituito dalle classi:

- `mytalk.server.usermanager.AuthenticationManager` (vedasi sezione3.3.1)
- `mytalk.server.usermanager.UserManager` (vedasi sezione3.3.2)

3.3.1 `mytalk.server.usermanager.AuthenticationManager`

La classe AuthenticationManager si occupa di passare allo strato dao i dati necessari per la verifica dell'autenticazione

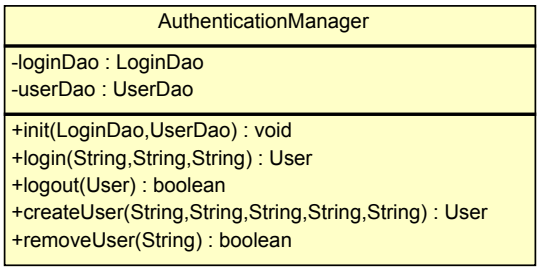

Figura 14: Classe AuthenticationManager

Relazioni d'uso con altri moduli

- **Utilizza:**
  - `mytalk.server.dao.LoginDao`
  - `mytalk.server.dao.LoginDaoSQL`
  - `mytalk.server.dao.UserDao`
  - `mytalk.server.dao.UserDaoSQL`
- **Viene utilizzata da:**
  - `mytalk.server.transfer.AuthenticationTransfer`

Attributi privati

- **LoginDao loginDao:** conterrà un riferimento alla istanza della classe LoginDaoSQL contenente i dati per effettuare l'autenticazione
- **UserDao userDao:** conterrà un riferimento alla istanza della classe UserDaoSQL contenente i dati per effettuare le operazioni di creazione e cancellazione dell'utente

Metodi pubblici

- **void init(LoginDao, UserDao):** metodo che fungerà da costruttore della classe, dovrà impostare il valore degli attributi secondo il valore dei parametri passati

- **User login(String, String, String):** metodo che invocherà il metodo *login(String, String, String)* della classe LoginDaoSQL per effettuare l'operazione di *login*. In caso di successo tale metodo ritornerà l'oggetto User corrispondente all'username
- **boolean logout(User):** metodo che effettuerà il *logout* dell'utente
- **User createUser(String, String, String, String, String):** metodo che si occuperà di creare un nuovo utente nella base di dati, purché lo username non sia già utilizzato
- **boolean removeUser(String):** metodo che eliminerà un utente registrato dal database

3.3.2 `mytalk.server.usermanager.UserManager`

La classe UserManager si occupa di gestire la modifica dei dati degli utenti e l'invio ai client della lista di utenti registrati al server

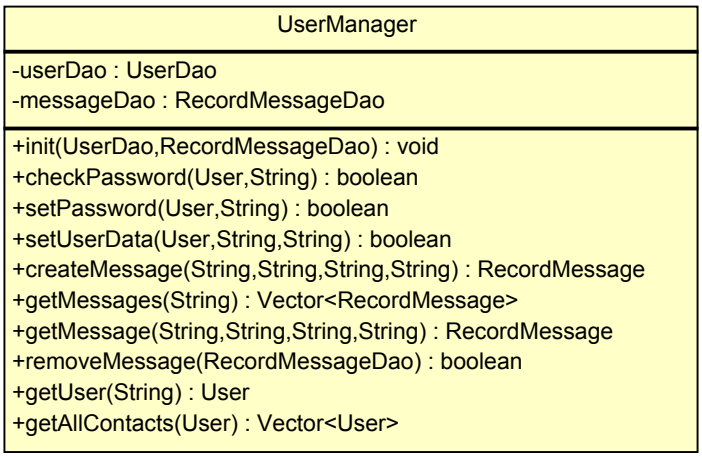

Figura 15: classe UserManager

Relazioni d'uso con altri moduli

- **Utilizza:**
  - `mytalk.server.dao.UserDao`
  - `mytalk.server.dao.UserDaoSQL`
  - `mytalk.server.dao.RecordMessageDao`
  - `mytalk.server.dao.RecordMessageDaoSQL`

- **Viene utilizzata da:**
  - `mytalk.server.transfer.AuthenticationTransfer`
  - `mytalk.server.transfer.RecordMessageTransfer`
  - `mytalk.server.transfer.UserTransfer`

Attributi privati

- **UserDao userDao:** conterrà un riferimento alla istanza della classe UserDaoSQL, per eseguire le operazioni sull'utente e per recuperare la lista di tutti gli utenti registrati nel server
- **RecordMessageDao messageDao:** conterrà un riferimento alla istanza della classe RecordMessageDaoSQL per permettere le operazioni sui messaggi registrati

Metodi pubblici

- **void init(UserDao, RecordMessageDao):** metodo che fungerà da costruttore della classe
- **boolean checkPassword(User, String):** metodo che controllerà la corrispondenza dello username passato con la password passata
- **boolean setPassword(User, String):** metodo che modificherà la password esistente con quella passata
- **boolean setUserData(User, String, String):** metodo che si occuperà di modificare il nome e il cognome esistenti con quelli passati
- **RecordMessage createMessage(String, String, String, String):** metodo che aggiungerà un messaggio registrato destinato all'utente
- **Vector<RecordMessage> getMessages(String):** metodo che ritornerà il vettore dei messaggi differiti che sono stati mandati ad un utente.
- **RecordMessage getMessage(String, String, String, String):** metodo che ritornerà il messaggio differito avente le caratteristiche richieste.
- **boolean removeMessage(RecordMessage):** metodo che rimuoverà un messaggio registrato
- **User getUser(String)**: metodo che restituirà l'utente corrispondente al dato username
- **Vector<User> getAllContacts(User):** metodo che restituirà tutti gli utenti presenti nel server, ad eccezione di quello dato

3.4 Package functionmanager

Il package functionmanager contiene tutte le classi che si occupano di convetire le strutture dati più complesse in stringhe per lo strato transfer

Relazioni d'uso con altri moduli

- **Utilizza:**
  - `mytalk.server.shared`
- **Viene utilizzato da:**
  - `mytalk.server.transfer`

Costituito dalla classe:

- `mytalk.server.functionmanager.Converter` (vedasi sezione3.4.1)

3.4.1 `mytalk.server.functionmanager.Converter`

La classe Converter restituisce in forma di stringa il vettore di utenti passato per argomento

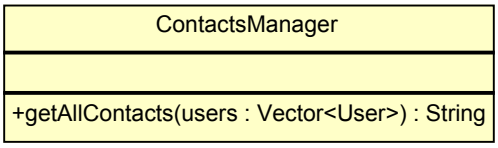

Figura 16: Classe Converter

Relazioni d'uso con altri moduli

- **Viene utilizzato da:**
  - `mytalk.server.transfer.AuthenticationTransfer`
  - `mytalk.server.transfer.RecordMessageTransfer`
  - `mytalk.server.transfer.UserTransfer`

Metodi pubblici

- **String convertUsers(Vector<User>, String):** metodo che si occuperà di convertire in forma di stringa il vettore di utenti passato per argomento, inserendo inizialmente la stringa type e il numero di utenti presenti nel vettore passato

- **String convertMessages(Vector<RecordMessage>, String):** metodo che si occuperà di convertire in forma di stringa il vettore di messaggi passato per argomento, inserendo inizialmente la stringa type e il numero di messaggi presenti nel vettore
- **String convertTutorials(Map<String, String>, String):** metodo che si occuperà di convertire in forma di stringa i tutorials passati per argomento, inserendo inizialmente la stringa type e il numero di tutorials presenti

3.5 Package transfer

Il package `mytalk.server.transfer` si occupa della comunicazione con il client.

Relazioni d'uso con altri moduli

- **Utilizza:**
  - `mytalk.server.usermanager`
  - `mytalk.server.functionmanager`
  - `mytalk.server.shared`
- **Comunica con:**
  - `mytalk.client.communication`

Costituito dalle classi:

- `mytalk.server.transfer.ListenerTransfer` (vedasi sezione3.5.1)
- `mytalk.server.transfer.AuthenticationTransfer` (vedasi sezione3.5.2)
- `mytalk.server.transfer.CallTransfer` (vedasi sezione3.5.3)
- `mytalk.server.transfer.ChatTransfer` (vedasi sezione3.5.4)
- `mytalk.server.transfer.FileTransfer` (vedasi sezione3.5.5)
- `mytalk.server.transfer.RecordMessageTransfer`(vedasi sezione3.5.6)
- `mytalk.server.transfer.UserTransfer` (vedasi sezione3.5.7)

3.5.1 `mytalk.server.transfer.ListenerTransfer`

La classe astratta ListenerTransfer implementa WebSocketServerTokenListener e viene estesa da tutte le altre classi dello strato.

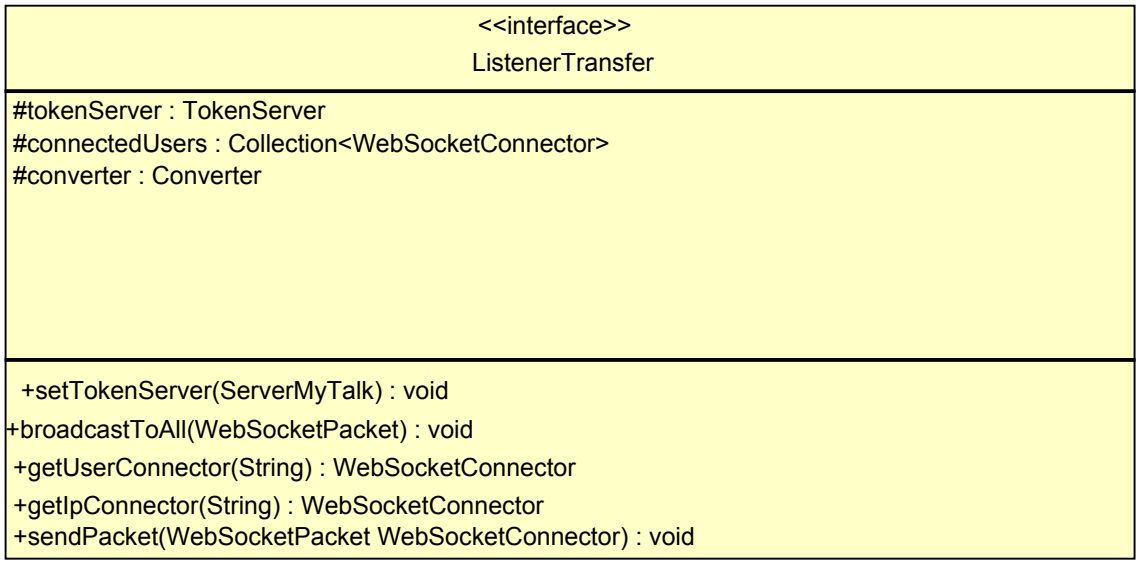

Figura 17: Classe ListenerTransfer

Relazioni d'uso con altri moduli

- **Utilizza:**
  - `mytalk.server.functionmanager.Converter`
- **Implementa:**
  - `org.jwebsocket.listener.WebSocketServerTokenListener`
- **Viene estesa da:**
  - `mytalk.server.transfer.AuthenticationTransfer`
  - `mytalk.server.transfer.CallTransfer`
  - `mytalk.server.transfer.ChatTransfer`
  - `mytalk.server.transfer.FileTransfer`
  - `mytalk.server.transfer.RecordMessageTransfer`
  - `mytalk.server.transfer.UserTransfer`

Attributi protetti

- **[TokenServer] tokenServer:** conterrà un riferimento al token server di JWebSocket
- **[Collection<WebSocketConnector>] connectedUsers**: conterrà la lista dei connettori collegati al server
- **[Converter] converter**: conterrà un riferimento alla classe Converter

Metodi pubblici

- **[void] setTokenServer(ServerMyTalk):** metodo per inizializzare *tokenServer*
- **[void] broadcastToAll(WebSocketPacket):** metodo per inviare il pacchetto a tutti gli utenti presenti in *connectedUsers*
- **[WebSocketConnector] getUserConnector(String):** metodo per trovare il connettore dell'utente connesso, con il dato username, al sistema
- **[WebSocketConnector] getIpConnector(String):** metodo per trovare il connettore, avente il dato indirizzo IP, connesso al sistema
- **[void] sendPacket(WebSocketPacket, WebSocketConnector):** metodo per inviare il dato pacchetto al dato connettore

3.5.2 `mytalk.server.transfer.AuthenticationTransfer`

La classe AuthenticationTransfer si occupa della registrazione e autenticazione degli utenti.

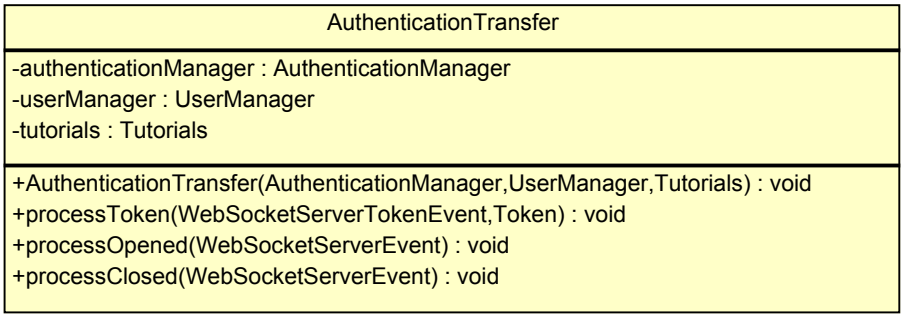

Figura 18: Classe AuthenticationTransfer

Relazioni d'uso con altri moduli

- **Utilizza:**
  - `mytalk.server.usermanager.AuthenticationManager`
  - `mytalk.server.usermanager.UserManager`
  - `mytalk.server.shared.Tutorials`
  - `mytalk.server.shared.User`
- **Estende:**
  - `mytalk.server.transfer.ListenerTransfer`
- **Comunica con:**
  - `mytalk.client.AuthenticationCommunication`
  - `mytalk.client.communication.ContactsCommunication`
  - `mytalk.client.communication.TutorialCommunication`

Attributi privati

- **[AuthenticationManager] authenticationManager:** conterrà il riferimento alla singola istanza della classe AuthenticationManager presente nel server
- **[UserManager] userManager:** conterrà il riferimento alla singola istanza della classe UserManager presente nel server
- **[Tutorials] tutorials:** conterrà il riferimento ai tutorials presenti nel package shared

Metodi pubblici

- **AuthenticationTransfer(AuthenticationManager, UserManager, Tutorials):** costruttore della classe, dovrà inizializzaere gli attributi della classe con i parametri ricevuti
- **[void] processToken(WebSocketServerTokenEvent, Token):** metodo per la gestione dei token in arrivo, in particolare si occuperà di quelli aventi come tipo:
  - *login*: invocherà il metodo *login(String, String, String)* della classe AuthenticationManager, usando come parametri lo username e la password, presenti nel token, e l'indirizzo IP del connettore da cui riceve il pacchetto. Se l'operazione non andrà a buon termine, invierà una risposta negativa al client, al contrario, invierà una risposta positiva al client e il nuovo indirizzo IP del contatto a tutti gli utenti connessi, utilizzando il metodo *broadcastToAll(WebSocketPacket),* infine assegnerà al connettore lo username dell'utente autenticato

  - *signUp*: invocherà il metodo *createUser(String, String, String, String, String)* della classe AuthenticationManager, usando come parametri i dati presenti nel token, e l'indirizzo IP del connettore da cui riceve il pacchetto. Se l'operazione non andrà a buon termine, invierà una risposta negativa al client, al contrario, invierà una risposta positiva al client ed invierà il nuovo indirizzo IP del contatto a tutti gli utenti connessi, utilizzando il metodo *broadcastToAll(WebSocketPacket)*
  - *getContacts*: invocherà il metodo *getAllContacts(User)* della classe UserManager, per ottenere il vettore degli utenti presenti nel server, e il metodo *convertUsers(Vector<User>,String)* della classe Converter per convertire tale lista in stringa, infine restituirà al client il pacchetto contenente tale stringa
  - *logout*: invocherà il metodo *logout(User)* della classe AuthenticationManager, se l'operazione non andrà a buon termine, invierà una risposta negativa al client, al contrario, invierà una risposta positiva al client ed invierà il nuovo indirizzo IP del contatto a tutti gli utenti connessi, utilizzando il metodo *broadcastToAll(WebSocketPacket),* infine rimuoverà dal connettore lo username

- **[void] processOpened(WebSocketServerEvent):** metodo che si occuperà della creazione di un processo. Aggiungerà il nuovo connettore ad *connectedUsers,* invocherà il metodo *convertTutorials(Map<String,String>, String)* della classe Converter e invierà la stringa così ottenuta al nuovo connettore

- **[void] processClosed(WebSocketServerEvent):** metodo che si occuperà della chiusura di un processo. Se non si è fatta l'operazione di *logout* dal client, invocherà il metodo *logout(User)* della classe AuthenticationManager ed invierà il nuovo indirizzo IP del contatto a tutti gli utenti connessi, utilizzando il metodo *broadcastToAll(WebSocketPacket).* Infine rimuoverà il connettore da *connectedUsers*

3.5.3 `mytalk.server.transfer.CallTransfer`

La classe CalTransfer si occupa di gestire il traffico di pacchetti per effettuare l'inizializzazione di una comunicazione audio/video

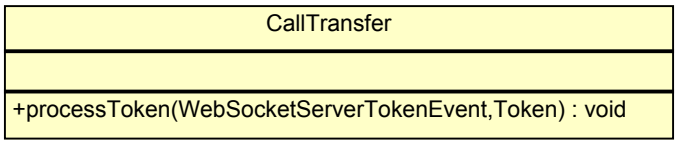

Figura 19: Classe CallTransfer

Relazioni d'uso con altri moduli

- **Estende:**
  - `mytalk.server.transfer.ListenerTransfer`
- **Comunica con:**
  - `mytalk.client.communication.CallCommunication`
  - `mytalk.client.communication.NotificationCommunication`

Metodi pubblici

- **[void] processToken(WebSocketServerTokenEvent, Token):**metodo per la gestione dei token in arrivo. Per ogni token che gestirà, invocherà il metodo *getUserConnector(String)*, usando come parametro lo username presente nel token, per trovare il connettore corrispondente. Se non esiste invierà al connettore che ha effettuato la richiesta un pacchetto con tipo *answeredCall* avente come risposta *error*. Se il connettore esiste risponderà con un pacchetto di tipo diverso a seconda della richiesta:
  - richiesta di tipo *call,* invierà un pacchetto di tipo *call,* contenente il tipo di chiamata, passatogli dal client, e lo username dell'utente che ha effettuato la richiesta
  - richiesta di tipo *answeredCall,* invierà un pacchetto di tipo *answeredCall*, avente come risposta *true*
  - richiesta di tipo *refuseCall,* invierà un pacchetto di tipo *answeredCall*, avente come risposta *false*
  - richiesta di tipo *busy,* invierà un pacchetto di tipo *answeredCall*, avente come risposta *busy*
  - richiesta di tipo *offer,* invierà un pacchetto contentent l'attributo *description* passatogli dal client che ha effettuato la richiesta
  - richiesta di tipo *candidate,* invierà un pacchetto contentent l'attributo *cand* passatogli dal client che ha effettuato la richiesta
  - richiesta di tipo *endCall,* invierà un pacchetto di tipo *endCall*
  - richiesta di tipo *candidateReady,* invierà un pacchetto di tipo *candidateReady*

3.5.4 `mytalk.server.transfer.ChatTransfer`

La classe ChatTransfer si occupa della registrazione dei messaggi di chat

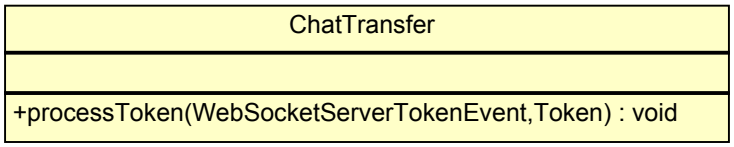

Figura 20: Classe ChatTransfer

Relazioni d'uso con altri moduli

- **Estende:**
  - `mytalk.server.transfer.ListenerTransfer`
- **Comunica con:**
  - `mytalk.client.communication.ChatCommunication`

Metodi pubblici

- **[void] processToken(WebSocketServerTokenEvent, Token):**metodo per la gestione dei token in arrivo. Gestirà i token aventi come tipo *sendText*, invocherà il metodo *getUserConnector(String)*, usando come parametro lo username presente nel token, per trovare il connettore corrispondente. Se il connettore non è presente, invierà al connettore che ha effettuato la richiesta un pacchetto con tipo *notDelivered,* altrimenti manderà al connettore trovato un pacchetto di tipo *sendText*, avente come attributi il messaggio ricevuto in arrivo e lo username del mittente

3.5.5 `mytalk.server.transfer.FileTransfer`

La classe FileTransfer si occupa della gestione dei file inviati

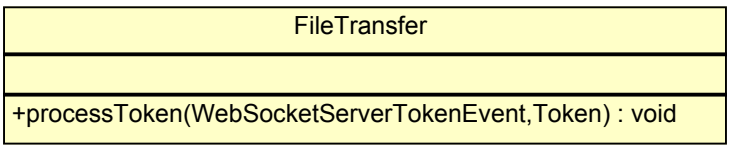

Figura 21: Classe FileTransfer

Relazioni d'uso con altri moduli

- **Estende:**
  - `mytalk.server.transfer.ListenerTransfer`

- **Comunica con:**
  - `mytalk.client.communication.FileCommunication`
  - `mytalk.client.communication.NotificationCommunication`

Metodi pubblici

- **[void] processToken(WebSocketServerTokenEvent, Token):**metodo per la gestione dei token in arrivo. Gestirà i token aventi come tipo *file*, invocherà il metodo *getUserConnector(String)*, usando come parametro lo username presente nel token, per trovare il connettore corrispondente. Infine Manderà al connettore trovato un pacchetto con lo stesso, avente come attributi il file ricevuto in arrivo e lo username del mittente

3.5.6 `mytalk.server.transfer.RecordMessageTransfer`

La classe RecordMessageTransfer si occupa della registrazione dei messaggi

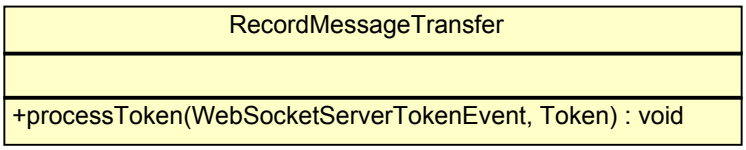

Figura 22: Classe RecordMessageTransfer

Relazioni d'uso con altri moduli

- **Utilizza:**
  - `mytalk.server.usermanager.UserManager`
  - `mytalk.server.shared.RecordMessage`
- **Estende:**
  - `mytalk.server.transfer.ListenerTransfer`
- **Comunica con:**
  - `mytalk.client.communication.RecordMessageCommunication`
  - `mytalk.client.communication.NotificationCommunication`

Metodi pubblici

- **[void] processToken(WebSocketServerTokenEvent, Token):** metodo per la gestione dei token in arrivo. in particolare si occuperà di quelli aventi come tipo:
  - *sendRecord*: invocherà il metodo *createMessage(String, String, String, String)* per salvare il messaggio nel server, usando come parametri quelli ricevuti. Successivamente invocherà il metodo *getUserConnector(String)*, usando come parametro lo username presente nel token, per trovare il connettore corrispondente. Se è presente, gli invierà un pacchetto di tipo *record*, contente le informazioni ricevute
  - *removerRecord:* invocherà i metodi *getMessage(String, String, String, String)* e *removeMessage(RecordMessage)* della classe UserManager per trovare ed eliminare il messaggio dal server

3.5.7 mytalk.server.transfer.UserTransfer

La classe UserTransfer si occupa della gestione delle modifiche dei dati dell'utente

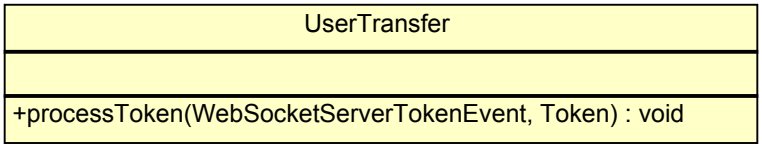

Figura 23: Classe UserTransfer

Relazioni d'uso con altri moduli

- **Utilizza:**
  - `mytalk.server.usermanager.UserManager`
- **Estende:**
  - `mytalk.server.transfer.ListenerTransfer`
- **Comunica con:**
  - `mytalk.client.communication.ChatCommunication`

Metodi pubblici

- **[void] processToken(WebSocketServerTokenEvent, Token):** metodo per la gestione dei token in arrivo. in particolare si occuperà di quelli aventi come tipo:
  - *checkCredentials*: invocherà il metodo *checkPassword(User, String)*, passandogli come parametro lo user corrispondente all'utente che ha effettuato la richiesta e la password presente nel pacchetto, e invierà al client la risposta ottenuta
  - *changeData:* invocherà i metodi *setUserData(User, String, String)* e *setPassword(UserString)* per modificare i dati dell'utente con quelli ricevuti nel pacchetto

4 Specifica Client

NOTA (chiave di lettura capitolo):

- Per quanto JavaScript sia un linguaggio debolmente tipizzato, al fine di descrivere meglio gli attributi si è scelto di specificare i tipi che dovranno assumere le variabili, una volta inizializzate. Questi verranno indicati fra parentesi quadre

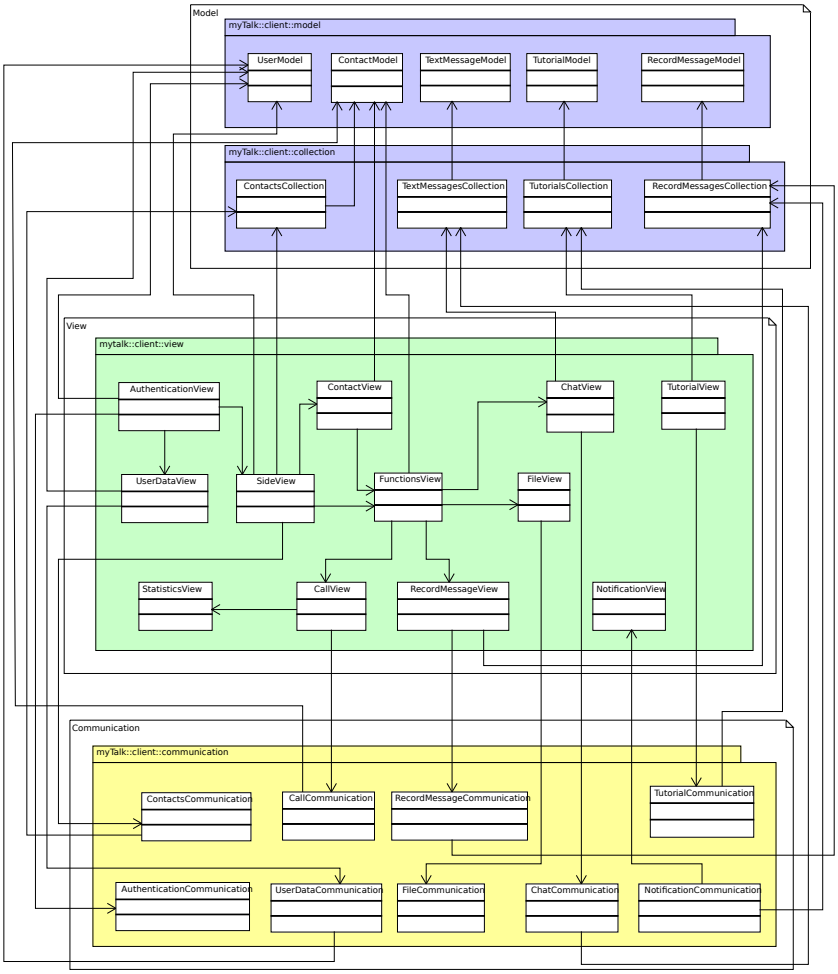

Figura 24: Architettura del client

Il client contiene i seguenti package:

- `mytalk.client.view` (vedasi sezione 4.1)
- `mytalk.client.communication` (vedasi sezione 4.2)
- `mytalk.client.collection` (vedasi sezione 4.3)
- `mytalk.client.model` (vedasi sezione 4.4)

4.1 Package view

Il package `mytalk.client.view` costituisce la parte del sistema che definisce ed implementa l'interfaccia web usufruibile dagli utenti mediante pagine web. Diversamente da uno schema di tipo MVC il package view contiene non solo le parti di interfaccia ma anche parti logiche, come previsto dal framework Backbone.js, che prevede uno schema di tipo MV*

Relazione d'uso con altri moduli

- **Il package utilizza:**
  - `mytalk.client.communication`
  - `mytalk.client.model`
  - `mytalk.client.collection`

Costituito dalle classi:

- `mytalk.client.view.AuthenticationView` (vedasi sezione 4.1.1)
- `mytalk.client.view.CallView` (vedasi sezione 4.1.2)
- `mytalk.client.view.ChatView` (vedasi sezione 4.1.3)
- `mytalk.client.view.ContactView` (vedasi sezione 4.1.4)
- `mytalk.client.view.FileView` (vedasi sezione 4.1.5)
- `mytalk.client.view.FunctionsView` (vedasi sezione 4.1.6)
- `mytalk.client.view.NotificationView` (vedasi sezione 4.1.7)
- `mytalk.client.view.RecordMessageView` (vedasi sezione 4.1.8)
- `mytalk.client.view.SideView` (vedasi sezione 4.1.9)
- `mytalk.client.view.StatisticsView` (vedasi sezione 4.1.10)
- `mytalk.client.view.TutorialView` (vedasi sezione 4.1.11)
- `mytalk.client.view.UserDataView` (vedasi sezione 4.1.12)

4.1.1 `mytalk.client.view.AuthenticationView`

La classe AuthenticationView deve definire la struttura, e la conseguente visualizzazione, della parte della pagina web che consente di effettuare la registrazione, l'accesso e la disconnessione dal sistema

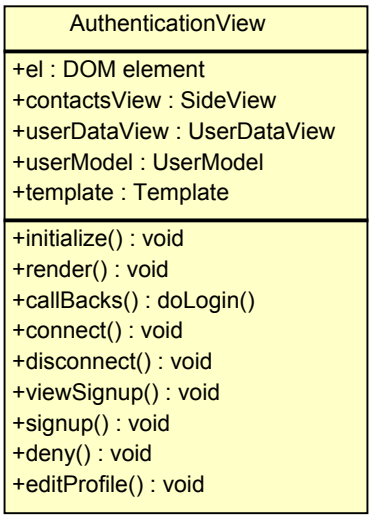

Figura 25: Classe AuthenticationView

Relazione d'uso con altri moduli

- **La classe utilizza:**
  - `mytalk.client.communication.AuthenticationCommunication`
  - `mytalk.client.model.UserModel`
  - `mytalk.client.view.UserDataView`
  - `mytalk.client.view.SideView`
  - `mytalk.client.template.AuthenticationTemplate`
- **La classe deriva da**:
  - `mytalk.client.Backbone.View`

Attributi pubblici:

- **[DOM element] el:** conterrà l'elemento DOM al quale è legata la classe
- [**SideView**] **contactsView:** conterrà il riferimento alla classe SideView
- **[UserDataView] userDataView:** conterrà il riferimento alla classe UserDataView
- **[UserModel] userModel:** conterrà il riferimento alla classe UserModel
- **[Template] template:** conterrà il riferimento al template utilizzato

Eventi:

- **click button#login:** evento che richiama la funzione *connect()*
- **click button#logout:** evento che richiama la funzione *disconnect()*
- **click button#signup:** evento che richiama la funzione *viewSignup()*
- **click button#sign:** evento che richiama la funzione *signup()*
- **click button#deny:** evento che richiama la funzione *deny()*
- **click button#edit:**evento che richiama la funzione *editProfile()*

Metodi pubblici:

- **[void] initialize():** metodo per inizializzare l'oggetto, si occuperà di inizializzare *contactsView* e richiamare il metodo *render()*
- **[void] render():** metodo per effettuare la scrittura nella pagina del template AuthenticationTemplate
- **[doLogin()] callBacks():** metodo che permetterà ad altre classi di usufruire del metodo *doLogin(String, String, String[], AuthenticationView)*
  - **[void] doLogin(String, String, String[], AuthenticationView)**: metodo che effettuerà l'operazione di login. Si occuperà di creare il model corrispondente all'utente, aggiornare il template e invocare il metodo *getContacts(AuthenticationView)* sull'oggetto *contactsView*
- **[void] connect():** metodo che avvierà l'operazione di login dell'utente richiamando il metodo *checkCredentials(String, String, AuthenticationView.callBacks(), AuthenticatioView)* della classe *AuthenticationCommunication* passandogli come parametri lo username e la password inseriti negli appositi campi
- **[void] disconnect():** metodo che si occuperà di chiudere la sessione con il server, dovrà invocare il metodo *logout(String)* della classe *AuthenticationCommunication*, aggiornare il template e deallocare *userDataView* e *contactsView*

- **[void] viewSignup():** metodo che permetterà la visualizzazione del form di registrazione
- **[void] signup():** metodo che avvierà l'operazione di registrazione dell'utente. Dovrà controllare che i campi obbligatori siano compilati e che la password e quella di conferma coincidano, in caso affermativo invocherà il metodo *signup(String, String, String, String, AuthenticationView.callBacks(), AuthenticationView)* della classe *AuthenticationCommunication* altrimenti mostrerà l'avviso di errore
- **[void] deny():** metodo per annullare la compilazione della registrazione e tornare allo stato precedente alla registrazione
- **[void] editProfile()**: metodo che avvierà l'operazione di modifica dati, inizializzando *userDataView*. Prima di effettuare l'operazione dovrà controllare che tutte le view presenti nel content siano state chiuse

4.1.2 `mytalk.client.view.CallView`

La classe CallView deve definire la struttura, e la conseguente visualizzazione, della pagina di chiamata

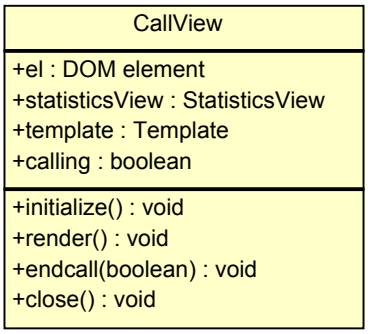

Figura 26: Classe CallView

Relazione d'uso con altri moduli

- **La classe utilizza:**
  - `mytalk.client.communication.CallCommunication`
  - `mytalk.client.view.StatisticsView`
  - `mytalk.client.template.CallTemplate`

- **La classe è utilizzata da:**
  - `mytalk.client.view.FunctionsView`
- **La classe deriva da**:
  - `mytalk.client.Backbone.View`

Attributi pubblici:

- **[DOM element] el:** conterrà l'elemento DOM al quale è legata la classe
- **[StatisticsView] statisticsView:** conterrà il riferimento alla classe StatisticsView
- **[Template] template:** conterrà il riferimento al template utilizzato
- **[boolean] calling:** indicherà se l'utente è occupato in una chiamata o no

Eventi:

- **click button#endcall:** evento che richiama la funzione *endcall()*

Metodi pubblici:

- **[void] initialize():** metodo per inizializzare l'oggetto, si occuperà di inizializzare *statisticsView* e richiamare il metodo *render()*
- **[void] render():** metodo per effettuare la scrittura nella pagina del template CallTemplate e che verificherà che sia in corso una chiamata e, in caso affermativo, invocherà il metodo *recoverCall()* di CallCommunication e il metodo *render()* di StatisticsView. Altrimenti, a seconda che l'utente sia il chiamato o il chiamante, invocherà i metodi *sendAnswer(String, ContactModel, CallView)* o *sendCall(String, ContactModel, CallView)* di CallCommunication e assegnerà a *calling* lo stato di chiamata
- **[void] endcall(boolean):** metodo che terminerà la chiamata in corso richiamando i metodi *closeViewCall()* della classe FunctionsView, *close()* della classe stessa e *close()* della classe StatisticsView. Inoltre nel caso il metodo venga invocato dall'utente che ha deciso di chiudere la chiamata, verrà invocato anche il metodo *endCall()* della classe CallCommunication.
- **[void] close():** metodo che chiuderà l'istanza della classe creata

4.1.3 `mytalk.client.view.ChatView`

La classe ChatView definisce la struttura, e la conseguente visualizzazione, della parte della pagina web che consente di effettuare comunicazioni testuali

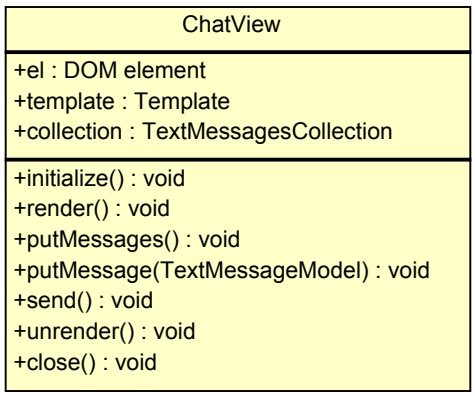

Figura 27: Classe ChatView

Relazione d'uso con altri moduli

- **La classe utilizza:**
  - `mytalk.client.communication.ChatCommunication`
  - `mytalk.client.collection.TextMessagesCollection`
  - `mytalk.client.template.ChatTemplate`
- **La classe è utilizzata da:**
  - `mytalk.client.view.FunctionsView`
- **La classe deriva da**:
  - `mytalk.client.Backbone.View`

Attributi pubblici:

- **[DOM element] el:** conterrà l'elemento DOM al quale è legata la classe
- **[Template] template:** conterrà il riferimento al template utilizzato
- **[TextMessagesCollection] collection:** conterrà il riferimento alla classe TextMessagesCollection

Eventi:

- **click button#send:** evento che richiama la funzione *send()*

Metodi pubblici:

- **[void] initialize():** metodo per inizializzare l'oggetto, collegherà il metodo *render()* della classe alla collection in modo da richiamarlo ogni qual volta avvenga un cambiamento di quest'ultima
- **[void] render():** metodo per effettuare la scrittura nella pagina del template ChatTemplate inoltre, invocherà il metodo *putMessages()* per visualizzare i messaggi scambiati fino ad ora
- **[void] putMessages()**: metodo che si occuperà di prelevare dalla *collection* tutti i messaggi di testo della conversazione scelta, e invocherà la funzione *putMessage(TextMessageModel)* per permetterne la visualizzazione all'utente
- **[void] putMessage(TextMessageModel)**: metodo che si occuperà della scrittura di un messaggio di testo, presente nella *collection,* all'interno della pagina
- **[void] send():** metodo per inviare un messaggio al contatto, per fare ciò invocherà il metodo *send(String, string)* della classe ChatCommunication, con parametri lo username del contatto e il testo del messaggio. Provvederà anche ad aggiungere il messaggio a *collection*
- **[void] unrender():** metodo per chiudere l'istanza, per fare ciò rimuoverà da *collection* la conversazione con l'utente e invocherà il metodo *close()*
- **[void] close():** metodo che chiuderà l'istanza della classe creata

4.1.4 `mytalk.client.view.ContactView`

La classe ContactView consente la visualizzazione all'interno della pagina web di un singolo utente registrato

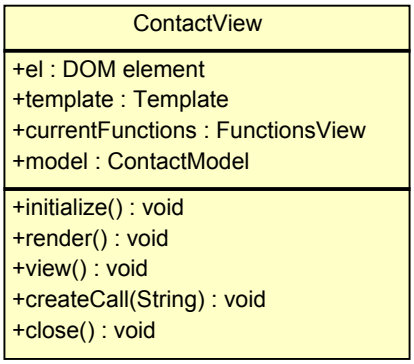

Figura 28: Classe ContactView

Relazione d'uso con altri moduli

- **La classe utilizza:**
  - `mytalk.client.view.FunctionsView`
  - `mytalk.client.model.ContactModel`
  - `mytalk.client.template.ContactTemplate`
- **La classe è utilizzata da:**
  - `mytalk.client.view.SideView`
- **La classe deriva da:**
  - `mytalk.client.Backbone.View`

Attributi pubblici:

- **[DOM element] el:** conterrà l'elemento DOM al quale è legata la classe
- **[Template] template:** conterrà il riferimento al template utilizzato
- **[FunctionsView] currentFunctions:** conterrà il riferimento alla classe FunctionsView
- **[ContactModel] model:** conterrà il model del contatto visualizzato

Eventi:

- **click span.contact:** evento che richiama la funzione *view()*

Metodi pubblici:

- **[void] initialize():** metodo per inizializzare l'oggetto, collegherà il metodo *render()* della classe al model in modo da richiamarlo ogni qual volta avvenga un cambiamento di quest'ultimo
- **[void] render():** metodo per effettuare la scrittura nella pagina del template ContactTemplate
- **[void] view():** metodo che si occuperà di avviare la FunctionsView relativa al contatto. Prima di fare ciò invocherà il metodo *closeOtherContacts(Strings)* della classe SideView per controllare che tutte le view presenti nel content siano state chiuse
- **[void] createCall(String):** metodo che verrà chiamato quando si accetterà una chiamata in arrivo; invocherà il metodo *closeOtherContacts(String)* della classe SideView, per controllare che tutte le view presenti nel content siano state chiuse, e uno tra i metodi *videocall(boolean)* o *audiocall(boolean)*, della classe FunctionsView, a seconda del tipo di chiamata passata dal parametro
- **[void] close():** metodo che chiuderà l'istanza della classe creata invocando il metodo *unrender()* della classe FunctionsView

4.1.5 `mytalk.client.view.FileView`

La classe FileView definisce la struttura, e la conseguente visualizzazione, della parte della pagina web che consente di selezionare un file salvato in locale per mandarlo all'utente autenticato selezionato

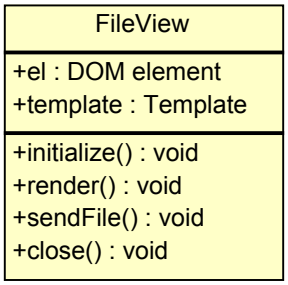

Figura 29: Classe FileView

Relazione d'uso con altri moduli

- **La classe utilizza:**
  - `mytalk.client.communication.FileCommunication`
  - `mytalk.client.template.FileTemplate`
- **La classe è utilizzata da:**
  - `mytalk.client.view.FunctionsView`
- **La classe deriva da**:
  - `mytalk.client.Backbone.View`

Attributi pubblici:

- **[DOM element] el:** conterrà l'elemento DOM al quale è legata la classe
- **[Template] template:** conterrà il riferimento al template utilizzato

Eventi:

- **click button#sendFile:** evento che richiama la funzione *sendFile()*

Metodi pubblici:

- **[void] initialize():** metodo per inizializzare l'oggetto, si occuperà di richiamare il metodo *render()*
- **[void] render():** metodo per effettuare la scrittura nella pagina del template FileTemplate
- **[void] sendFile():** metodo che invierà il file al server, attraverso l'utilizzo del metodo *sendFile(String, File)* di FileCommunication
- **[void] close():** metodo che chiuderà l'istanza della classe creata

4.1.6 `mytalk.client.view.FunctionsView`

La classe FunctionsView definisce la struttura, e la conseguente visualizzazione, delle funzionalità offerte dopo aver selezionato un contatto

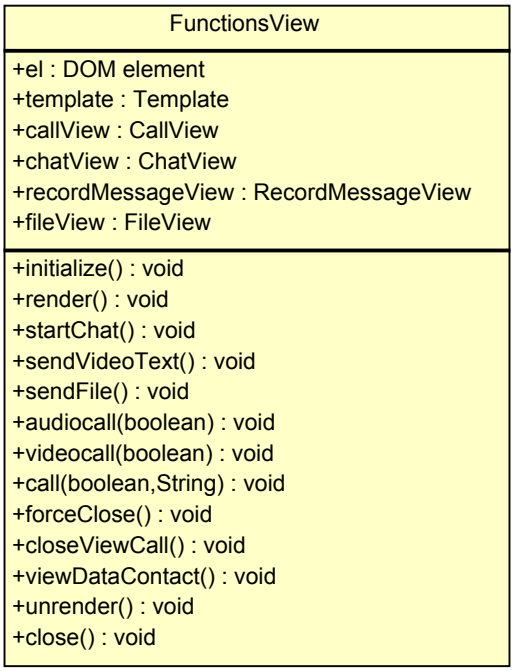

Figura 30: Classe FunctionsView

Relazione d'uso con altri moduli

- **La classe utilizza:**
  - `mytalk.client.view.CallView`
  - `mytalk.client.view.ChatView`
  - `mytalk.client.view.RecordMessageView`
  - `mytalk.client.view.FileView`
  - `mytalk.client.model.ContactModel`
- **La classe è utilizzata da:**
  - `mytalk.client.view.ContactView`
  - `mytalk.client.view.SideView`
- **La classe deriva da:**
  - `mytalk.client.Backbone.View`

Attributi pubblici:

- **[DOM element] el:** conterrà l'elemento DOM al quale è legata la classe
- **[Template] template:** conterrà il riferimento al template utilizzato
- **[CallView] callView:** conterrà il riferimento alla classe CallView
- **[ChatView] chatView:** conterrà il riferimento alla classe ChatView
- **[RecordMessageView] recordMessageView:** conterrà il riferimento alla classe RecordMessageView
- **[FileView] fileView:** conterrà il riferimento alla classe FileView

Eventi:

- **click button#dataContact:** evento che richiama la funzione *viewDataContact()*
- **click button#sendVideoText:** evento che richiama la funzione *sendVideoText()*
- **click button#sendFile:**evento che richiama la funzione *sendFile()*
- **click button#call:** evento che richiama la funzione *audiocall()*
- **click button#video:** evento che richiama la funzione *videocall()*

Metodi pubblici:

- **[void] initialize():** metodo per inizializzare l'oggetto, collegherà il metodo *render()* della classe al model in modo da richiamarlo ogni qual volta avvenga un cambiamento di quest'ultimo
- **[void] render():** metodo per effettuare la scrittura nella pagina del template FunctionsTemplate, per fare ciò dovrà controllare se l'istanza è stata creata da SideView o da ContactView e se esiste già una chiamata in corso, nel qual caso si mostrerà la chiamata.
- **[void] startChat():** metodo per avviare una comunicazione testuale con il contatto, per fare ciò inizializzerà *chatView*, se non è già esistente, e invocherà il metodo *render()* della classe ChatView
- **[void] sendVideoText():** metodo per inizializzare la vista per la registrazione di un messaggio, per fare ciò inizializzerà *recordMessageView*, se non è già esistente, e invocherà il metodo *render()* della classe RecordMessageView
- **[void] sendFile():** metodo per inizializzare la vista per l'invio di un file, per fare ciò inizializzerà *fileView*, se non è già esistente, e invocherà il metodo *render()* della classe fileView

- **[void] audiocall(boolean):** metodo per avviare una chiamata audio, per fare ciò invocherà il metodo *call(boolean, String),* passando come parametri quello di audiocall e la stringa 'audio'
- **[void] videocall(boolean):** metodo per avviare una chiamata audio, per fare ciò invocherà il metodo *call(boolean, String),* passando come parametri quello di videocall e la stringa 'video'
- **[void] call(boolean, String):** metodo per gestire l'inizializzazione di una chiamata, per fare ciò inizializzerà *callView* e invocherà il metodo *render(boolean, String, ContactModel, boolean)* specificando se si è il chiamante, il tipo di chiamata, il contatto con cui si sta comunicando e se si vuole registrare la chiamata. Inoltre si occuperà di mantenere la chat attiva, richiamando il metodo *startChat()*
- **[void] forceClose():** metodo che forzerà la chiusura di una chiamata invocando *endcall()* della classe CallView
- **[void] closeViewCall():** metodo che, a chiamata terminata, ripristinerà la vista
- **[void] viewDataContact():** metodo che mostrerà i dati dell'utente selezionato
- **[void] unrender():** metodo per chiudere definitivamente l'istanza, deve invocare il metodo *unrender()* della classe ChatView e il metodo *close()* della classe stessa
- **[void] close():** metodo che chiuderà l'istanza della classe creata e invoca il metodo *close()* della classe ChatView

4.1.7 `mytalk.client.view.NotificationView`

La classe NotificationView definisce la struttura, e la conseguente visualizzazione, della parte della pagina per la segnalazione di notifiche

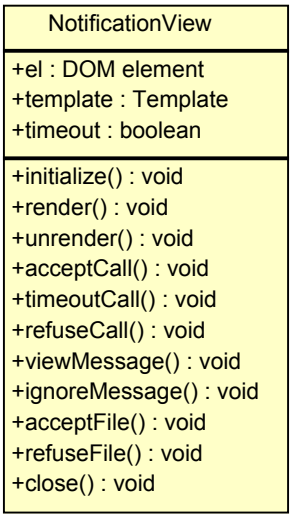

Figura 31: Classe NotificationView

Relazione d'uso con altri moduli

- **La classe utilizza:**
  - `mytalk.client.template.NotificationTemplate`
- **La classe è utilizzata da:**
  - `mytalk.client.communication.NotificationCommunication`
- **La classe deriva da**:
  - `mytalk.client.Backbone.View`

Attributi pubblici:

- **[DOM element] el:** conterrà l'elemento DOM al quale è legata la classe
- **[Template] template:** conterrà il riferimento al template utilizzato
- **[boolean] timeout:** variabile booleana che segnalerà quando non si risponderà in tempo ad una notifica

Eventi:

- **click button#acceptCall:** evento che richiama la funzione *acceptCall()*

- **click button#refuseCall:** evento che richiama la funzione *refuseCall()*
- **click button#viewMessage:** evento che richiama la funzione *viewRecordMessage*()
- **click button#ignoreMessage:** evento che richiama la funzione *ignoreMessage*()
- **click button#acceptFile**: evento che richiama la funzione *acceptFile*()
- **click button#refuseFile**: evento che richiama la funzione *refuseFile()*

Metodi pubblici:

- **[void] initialize():** metodo per inizializzare l'oggetto, si occuperà di inizializzare la variabile *timeout* a *true* e richiamare il metodo *render()*
- **[void] render():** metodo per effettuare la scrittura nella pagina del template NotificationTemplate
- **[void] unrender():** metodo per rimuovere la vista delle notifiche, invocherà il metodo *close()*
- **[void] acceptCall():** metodo per accettare una chiamata in arrivo, invocherà il metodo *unrender()* e assegnerà a *timeout* il valore *false,* infine invierà un evento di tipo *acceptCall* per segnalare che si sta iniziando una chiamata
- **[void] timeoutCall():** metodo per rifiutare una chiamata nel caso in cui non si risponda in tempo alla notifica, quindi se *timeout* risulterà *false* chiamerà il metodo *refuseCall()*
- **[void] refuseCall():** metodo per rifiutare una chiamata, per fare ciò invocherà il metodo *refuseCall(String)* della classe NotificationCommunication e il metodo *unrender()* della classe stessa
- **[void] viewMessage():** metodo per visualizzare il messaggio video/audio ricevuto, invocherà il metodo *unrender()* e invierà un evento di tipo *viewMessage* che sarà catturato da SideView
- **[void] ignoreMessage():** metodo per ignorare la visualizzazione del messaggio video/audio ricevuto, invocherà il metodo *unrender()*
- **[void] acceptFile():** metodo per accettare la ricezione di un file in arrivo, scaricherà il file e invocherà il metodo *unrender()*
- **[void] refuseFile():** metodo per rifiutare la ricezione di un file in arrivo, per fare ciò invocherà il metodo *refuseFile(String)* della classe NotificationCommunication e il metodo *unrender()* della classe stessa
- **[void] close():** metodo che chiuderà l'istanza della classe creata

4.1.8 `mytalk.client.view.RecordMessageView`

La classe RecordMessageView definisce la struttura, e la conseguente visualizzazione, della parte della pagina web per effettuare registrazioni audio o audio/video da inviare ad un utente registrato

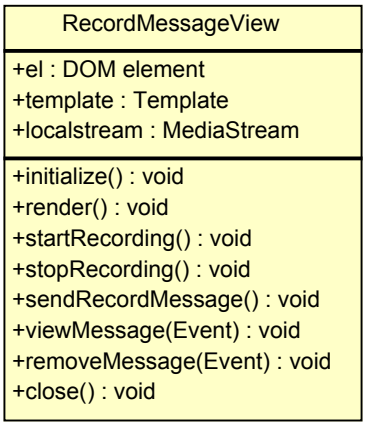

Figura 32: Classe RecordMessageView

Relazione d'uso con altri moduli

- **La classe utilizza:**
  - `mytalk.client.communication.RecordMessageCommunication`
  - `mytalk.client.template.RecordMessageTemplate`
  - `mytalk.client.collection.RecordMessagesCollection`
- **La classe è utilizzata da:**
  - `mytalk.client.view.FunctionsView`
- **La classe deriva da**:
  - `mytalk.client.Backbone.View`

Attributi pubblici:

- **[DOM element] el:** conterrà l'elemento DOM al quale è legata la classe
- **[Template] template:** conterrà il riferimento al template utilizzato
- **[MediaStream] localstream:** conterrà lo stream audio/video locale

Eventi:

- **click button#startRecord:** evento che richiama la funzione *startRecording()*
- **click button#stopRecord:** evento che richiama la funzione *stopRecording()*
- **click button#sendRecord:** evento che richiama la funzione *sendRecording()*
- **click li.message:** evento che richiama la funzione *viewMessage(Event)*
- **click button#removeRecord:** evento che richiama la funzione *removeMessage()*

Metodi pubblici:

- **[void] initialize():** metodo per inizializzare l'oggetto, si occuperà di inizializzare la variabile *localstream* e richiamare il metodo *render()*
- **[void] render():** metodo per effettuare la scrittura nella pagina del template RecordMessageTemplate
- **[void] startRecording():** metodo per avviare la registrazione del messaggio, inizializzerà la variabile *localStream* e inoltre sostituirà il pulsante per avviare la registrazione con il pulsante per fermarla
- **[void] stopRecording():** metodo per fermare la registrazione del messaggio, sostituirà il pulsante per terminare la registrazione con i pulsanti per riavviarla e per inviare il messaggio
- **[void] sendRecordMessage():** metodo per inviare il messaggio, farà ciò invocando il metodo *sendRecordMessage(String, Blob)* di RecordMessageCommunication e ripristinando lo stato iniziale della view
- **[void] viewMessage(Event):** metodo per visualizzare il video messaggio selezionato, dovrà anche aggiornare il template
- **[void] removeMessage(Event):** metodo per eliminare il video messaggio selezionato, invocherà il metodo *removeRecordMessage(RecordMessageModel)* della classe RecordMessageCommunication
- **[void] close():** metodo che chiuderà l'istanza della classe creata

4.1.9 `mytalk.client.view.SideView`

La classe SideView definisce la struttura, e la conseguente visualizzazione, della pagina web che consente la visualizzazione della lista utenti iscritti al server. La classe è composta da una lista di ContactView

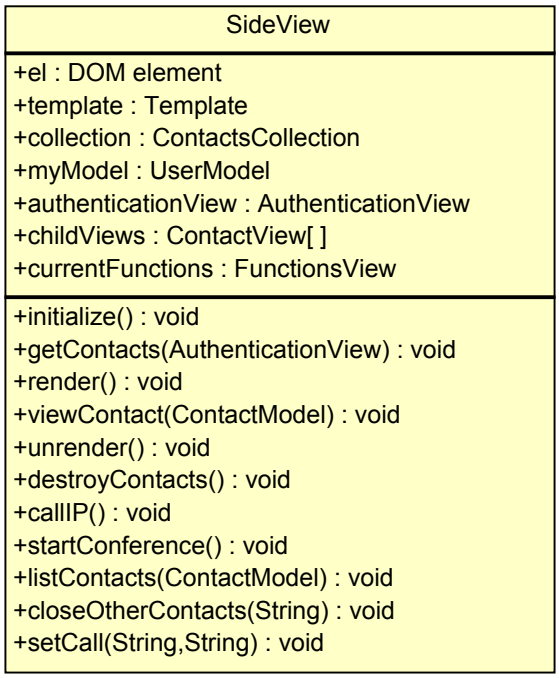

Figura 33: Classe SideView

Relazione d'uso con altri moduli

- **La classe utilizza:**
  - `mytalk.client.view.FunctionsView`
  - `mytalk.client.view.ContactView`
  - `mytalk.client.communication.ContactsCommunication`
  - `mytalk.client.template.SideTemplate`
  - `mytalk.client.collection.ContactsCollection`
  - `mytalk.client.model.UserModel`

- **La classe è utilizzata da:**
  - `mytalk.client.view.AuthenticationView`
- **La classe deriva da:**
  - `mytalk.client.Backbone.View`

Attributi pubblici:

- **[DOM element] el:** conterrà l'elemento DOM al quale è legata la classe
- **[Template] template:** conterrà il riferimento al template utilizzato
- **[ContactsCollection] collection:** conterrà il riferimento alla classe ContactsCollection
- **[UserModel] myModel:** conterrà il riferimento alla classe UserModel
- **[AuthenticationView] authenticationView:** conterrà il riferimento alla classe AuthenticationView
- **[ContactView[]] childViews:** conterrà i riferimenti alle istanze della classe ContactView
- **[FunctionsView] currentFunctions:** conterrà il riferimento alla classe FunctionsView

Eventi:

- **click button#callIP:** evento che richiama la funzione *callIP()*
- **click button#Conference:** evento che richiama la funzione *startConference()*

Metodi pubblici:

- **[void] initialize():** metodo per inizializzare l'oggetto, collegherà il metodo *render()* della classe alla *collection* in modo da richiamarlo ogni qual volta si aggiunga un elemento a quest'ultima. Predisporrà le funzioni *acceptCall(Event)* e *viewMessage(Event)* per gestire, rispettivamente, gli eventi *acceptCall* e *viewMessage* inviati da NotificationView. Infine effettuerà la scrittura, nella pagina, del template SideTemplate con lo stato dell'utente impostato come disconnesso
  - **[void] acceptCall(Event):** metodo che invocherà il metodo *setCall(String, String)* avente come parametri l'username del contatto che ha iniziato la chiamata e il tipo di chiamata

- **[void] getContacts(AuthenticationView):** metodo che aggiornerà la classe dopo l'autenticazione dell'utente, inizializzando *myModel* e *authenticationView* e aggiornando il template con lo stato dell'utente impostato a connesso. Inoltre invocherà il metodo *fetchContacts(String)* della classe ContactsCommunication per popolare la lista degli utenti registrati al server
- **[void] render():** metodo per visualizzare l'ultimo utente aggiunto nella collection, per fare ciò invocherà il metodo *viewContact(ContactModel)*
- **[void] viewContact(ContactModel):** metodo per visualizzare un contatto. Si occuperà di creare una ContactView corrispondente al contatto, di aggiungerla a *childViews* e di appenderla nella lista contenente tutti i contatti
- **[void] unrender():** metodo per chiudere la view, per fare ciò interromperà il collegamento tra la *collection* e il metodo *render()*, ripristinerà il template originale e invocherà il metodo *destroyContacts()*
- **[void] destroyContacts():** metodo per chiudere tutte le ContactView presenti in *childViews* e cancellare i contatti dalla *collection*
- **[void] callIP():** metodo per effettuare una comunicazione attraverso indirizzo IP, per fare ciò inizialmente invocherà il metodo *closeOtherContacts(Strings)*, per controllare che tutte le view presenti nel content siano state chiuse, e inizializzerà *currentFunctions* passandogli come parametro *From* il valore *IP*
- **[void] startConference():** metodo per effettuare una videoconferenza, , per fare ciò inizialmente invocherà il metodo *closeOtherContacts(Strings)*, per controllare che tutte le view presenti nel content siano state chiuse, e inizializzerà *currentFunctions* passandogli come parametro *From* il valore *Conference.* Inoltre invocherà il metodo *listContacts(ContactModel)* per visualizzare la lista degli utenti da selezionare per la conferenza
- **[void] listContacts(ContactModel):** metodo per gestire la lista degli utenti da selezionare per una conferenza, se l'utente risulta connesso si creerà un ContactView corrispondente all'utente e si appenderà nella lista degli utenti da selezionare
- **[void] closeOtherContacts(String):** metodo che si occuperà di chiudere le istanze di FunctionsView e UserDataView aperte, in modo da garantire che solo una view sia presente nel content. Se il metodo è invocato da un'istanza di ContactView, viene passato come parametro lo username relativo al contatto
- **[void] setCall(String, String):** metodo per impostare la ricezione di una chiamata, i parametri rappresentano lo username del chiamante e il tipo di chiamata. Invocherà la funzione *createCall(String)* dell'istanza, della classe ContactView, relativa al contatto

4.1.10 `mytalk.client.view.StatisticsView`

La classe StatisticsView definisce la struttura, e la conseguente visualizzazione, delle pagine web per la visualizzazione di statistiche riguardanti la chiamata

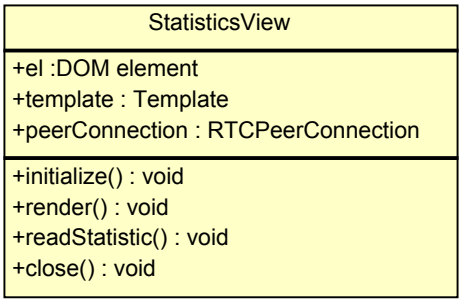

Figura 34: Classe StatisticsView

Relazione d'uso con altri moduli

- **La classe utilizza:**
  - `mytalk.client.template.StatisticsTemplate`
- **La classe è utilizzata da:**
  - `mytalk.client.view.CallView`
- **La classe deriva da**:
  - `mytalk.client.Backbone.View`

Attributi pubblici:

- **[DOM element] el:** conterrà l'elemento DOM al quale è legata la classe
- **[Template] template:** conterrà il riferimento al template utilizzato
- **[RTCPeerConnection] peerConnection:** conterrà il flusso dati della chiamata

Metodi pubblici:

- **[void] initialize():** metodo per inizializzare l'oggetto, inoltre richiamerà il metodo *render()*
- **[void] render():** metodo per effettuare la scrittura nella pagina del template StatisticsTemplate ed invocherà il metodo *readStatistic()* per visualizzare le statistiche a video
- **[void] readStatistic():** metodo per la gestione delle statistiche delle chiamate audio/video. Per fare ciò istanzierà un listener per gli eventi di tipo *setPeerConn,* collegandolo al metodo *showStatistics(Event)*
  - **[void] showStatistics(Event):** metodo che si occuperà di aggiornare, con la frequenza di un secondo, la durata, i byte di video e audio trasmessi, la latenza e la velocità di trasmissione della chiamata. Ricaverà i dati necessari dal metodo *getStats()* dell'oggetto *peerConnection* e utilizzerà il metodo *setInterval()* di JavaScript per aggiornarsi
- **[void] close():** metodo che chiuderà l'istanza della classe creata

4.1.11 `mytalk.client.view.TutorialView`

La classe TutorialView definisce la struttura, e la conseguente visualizzazione, della parte della pagina web che consente la visualizzazione dei video tutorial del prodotto **MyTalk**

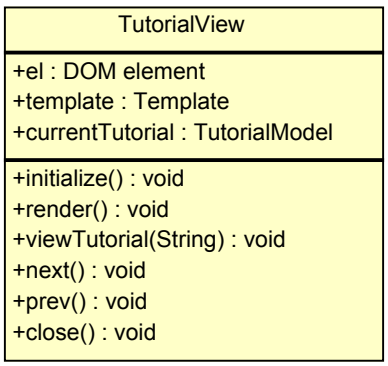

Figura 35: Classe TutorialView

Relazione d'uso con altri moduli

- **La classe utilizza:**
  - `mytalk.client.communication.TutorialCommunication`
  - `mytalk.client.template.TutorialTemplate`
  - `mytalk.client.collection.TutorialsCollection`
- **La classe deriva da**:
  - `mytalk.client.Backbone.View`

Attributi pubblici:

- **[DOM element] el:** conterrà l'elemento DOM al quale è legata la classe
- **[Template] template:** conterrà il riferimento al template utilizzato
- **[TutorialModel] currentTutorial:** utilizzato per tenere traccia del tutorial corrente

Eventi:

- **click li.videoTutorial:** evento che richiama la funzione *viewTutorial(String)*
- **click button#next:** evento che richiama la funzione *next()*
- **click button#previous:** evento che richiama la funzione *prev()*

Metodi pubblici:

- **[void] initialize():** metodo per inizializzare l'oggetto, richiamerà il metodo *getTutorials()* della classe TutorialCommunication e collegherà il metodo *render()* della classe alla *collection* in modo da richiamarlo ogni qual volta si aggiunga un elemento a quest'ultima
- **[void] render():** metodo per effettuare la scrittura nella pagina del template TutorialTemplate, al fine di visualizzare la lista dei tutorials disponibili
- **[void] viewTutorial(String):** metodo per visualizzare il tutorial selezionato, indicato attraverso il parametro, per fare ciò dovrà aggiornare il template. Inoltre assegnerà ad *currentTutorial* il tutorial selezionato
- **[void] next():** metodo per passare al video successivo, si occuperà anche di aggiornare *currentTutorial*
- **[void] prev():** metodo per passare al video precedente, si occuperà anche di aggiornare *currentTutorial*
- **[void] close():** metodo che chiuderà l'istanza della classe creata

4.1.12 `mytalk.client.view.UserDataView`

La classe UserDataView definisce la struttura, e la conseguente visualizzazione, della parte di pagina contenente i dati dell'utente autenticato e il collegamento ai metodi per la modifica di tali dati

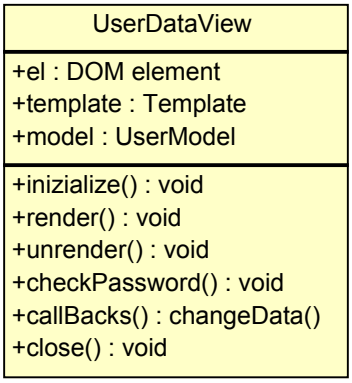

Figura 36: Classe UserDataView

Relazione d'uso con altri moduli

- **La classe utilizza:**
  - `mytalk.client.communication.UserDataCommunication`
  - `mytalk.client.template.UserDataTemplate`
  - `mytalk.client.model.UserModel`
- **La classe è utilizzata da:**
  - `mytalk.client.view.AuthenticationView`
- **La classe deriva da**:
  - `mytalk.client.Backbone.View`

Attributi pubblici:

- **[DOM element] el:** conterrà l'elemento DOM al quale è legata la classe
- **[Template] template:** conterrà il riferimento al template utilizzato
- **[UserModel] model:** conterrà il riferimento alla classe UserModel

Eventi:

- **click button#submitChange:** evento che richiama la funzione *checkPassword()*
- **click button#reset:** evento che richiama la funzione *render()*
- **click button#denyChange:** evento che richiama la funzione *unrender()*

Metodi pubblici:

- **[void] initialize():** metodo per inizializzare l'oggetto, si occuperà di richiamare il metodo *render()*
- **[void] render():** metodo per effettuare la scrittura nella pagina del template TutorialTemplate
- **[void] unrender():** metodo per rimuovere la vista dei dati dell'utente, invocherà il metodo *close()*
- **[void] checkPassword():** metodo che provvederà a controllare la correttezza della password, per fare ciò invocherà il metodo *checkPassword(UserModel, String, UserDataView)* della classe UserDataCommunication
- **[changeData()] callBacks():** metodo che permetterà ad altre classi di usufruire del metodo *changeData(UserModel, UserDataView)*
  - **[void] changeData(UserModel, UserDataView):** metodo per modificare i dati dell'utente. Per fare ciò, inizialmente, deve verificare che la nuova password, se esistente, coincida con quella di verifica, se supera il controllo invocherà il metodo *changeData(UserModel, String, String, String, UserDataView)* di UserDataCommunication, passandogli i dati aggiornati
- **[void] close():** metodo che chiuderà l'istanza della classe creata

4.2 Package Communication

Nel package `mytalk.client.communication` sono presenti le classi che effettuano la comunicazione con il server.

Relazione d'uso con altri moduli

- **Il package utilizzata:**
  - `mytalk.client.collection`
- **Il package è utilizzato da:**
  - `mytalk.client.view`

- **Il package comunica con:**
  - `mytalk.server.transfer`

Costituito dalle classi:

- `mytalk.client.communication.AuthenticationCommunication` (vedasi sezione 4.2.1)
- `mytalk.client.communication.CallCommunication` (vedasi sezione 4.2.2)
- `mytalk.client.communication.ChatCommunication` (vedasi sezione 4.2.3)
- `mytalk.client.communication.ContactsCommunication` (vedasi sezione 4.2.4)
- `mytalk.client.communication.FileCommunication` (vedasi sezione 4.2.5)
- `mytalk.client.communication.NotificationCommunication` (vedasi sezione 4.2.6)
- `mytalk.client.communication.RecordMessageCommunication` (vedasi sezione 4.2.7)
- `mytalk.client.communication.TutorialCommunication` (vedasi sezione 4.2.8)
- `mytalk.client.communication.UserDataCommunication` (vedasi sezione 4.2.9)

4.2.1 `mytalk.client.communication.AuthenticationCommunication`

La classe AuthenticationCommunication si occupa di inviare le credenziali inserite al server, e, inviare la risposta ricevuta a `mytalk.client.view.AuthenticationView`

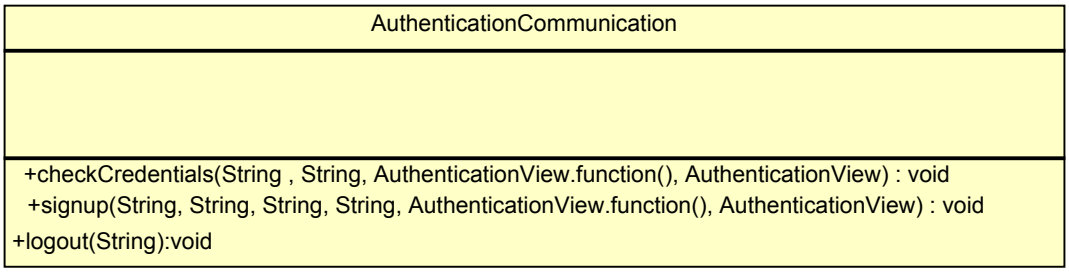

Figura 37: Classe AuthenticationCommunication

Relazione d'uso con altri moduli

- **La classe è utilizzata da:**
  - `mytalk.client.view.AuthenticationView`
- **La classe comunica con:**
  - `mytalk.server.transfer.AuthenticationTransfer`

Metodi pubblici:

- **[void] checkCredentials(String, String, AuthenticationView.function(), AuthenticationView):** metodo che si occuperà di controllare la correttezza dello username e della password inserite. Per fare ciò invierà al server un pacchetto, contenente username e password e avente come parametro aggiuntivo il tipo *login,* e attenderà il messaggio di risposta, anch'esso con tipo *login.* Se il messaggio sarà positivo invocherà il metodo *callBacks.doLogin* di AuthenticationView
- **[void] signup(String, String, String, String, AuthenticationView.function(), AuthenticationView):** metodo che si occuperà della registrazione dell'utente. Per fare ciò invierà al server un pacchetto contenente tutte le credenziali e avente tipo *signUp*, e attenderà il messaggio di risposta contrassegnato con il tipo *signUp.* Se il messaggio sarà positivo invocherà il metodo *callBacks.doLogin* di AuthenticationView
- **[void] logout(String):** metodo che gestisce la disconnessione dell'utente

4.2.2 `mytalk.client.communication.CallCommunication`

La classe CallCommunication si occuperà di avviare la comunicazione tra utenti

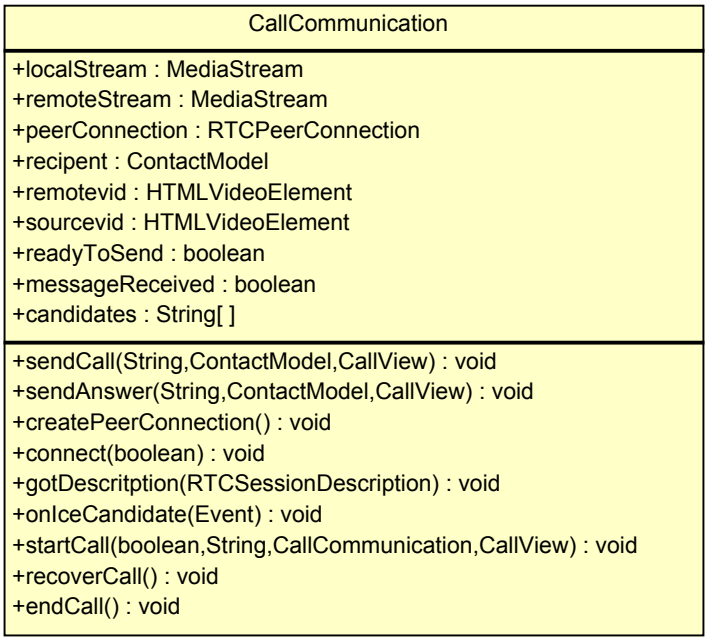

Figura 38: Classe CallCommunication

Relazione d'uso con altri moduli

- **La classe utilizza:**
  - `mytalk.client.model.ContactsModel`
- **La classe è utilizzata da:**
  - `mytalk.client.view.CallView`
- **La classe comunica con:**
  - `mytalk.server.transfer.CallTransfer`

Attributi pubblici:

- **[MediaStream] localStream:** conterrà lo stream audio/video dell'utente locale
- **[MediaStream] remoteStream:** conterrà lo stream audio/video dell'utente con cui si starà comunicando

- **[RTCPeerConnection] peerConnection:** conterrà il canale attraverso cui scorrerà il flusso della chiamata tra due utenti
- **[ContactModel] recipent:** conterrà il riferimento alla classe ContactModel riguardante l'utente con cui si starà comunicando
- **[HTMLVideoElement] remotevid:** conterrà l'elemento HTMLVideoElement dell'utente con cui si starà comunicando
- **[HTMLVideoElement] sourcevid:** conterrà l'elemento HTMLVideoElement dell'utente locale
- **[boolean] readyToSend:** variabile booleana che indicherà se è possibile o meno l'invio di *candidates*
- **[boolean] messageReceived:** variabile booleana che indicherà se l'utente con cui si vorrà comunicare sarà pronto ad accettare *candidates* o meno
- **[String[]] candidates:** array associativo contenente gli ICE candidates, *recipient* e il tipo di messaggio da inviare al server

Metodi pubblici:

- **[void] sendCall(String, ContactModel, CallView):** metodo per inizializzare la chiamata, dovrà avere come parametri il tipo d chiamata, l'istanza della classe ContactModel relativa al contatto e una riferimento a CallView. Lancerà l'evento *setOnCall,* che NotificationCommunication raccoglierà per impostare la variabile *onCalling* a *true*, e invierà al server un package di tipo *call* contenente il tipo di chiamata e il contatto che si vuole chiamare. Infine inizializzerà il listener che si occuperà di attendere la risposta del chiamato e invocare *onAnswer(Event)*
  - **[void] onAnswer(Event):** metodo che gestirà la risposta inviata dall'utente che si vuole chiamare, identificato dal tipo *answeredCall*; se la risposta sarà affermativa invocherà il metodo *startCall(boolean, String, CallCommunication, CallView)*, avente come parametri se l'utente è il chiamante, quindi in questo caso verrà inviato true, il tipo di chiamata e i riferimenti alle due classi, altrimenti lancerà l'evento *setOnCall,* che NotificationCommunication raccoglierà per impostare la variabile *onCalling* a *false*, e comunicherà all'utente il motivo della non inizializzazione della chiamata
- **[void] sendAnswer(String, ContactModel, CallView):** metodo per inviare all'utente che ha effettuato la chiamata una risposta positiva e per invocare il metodo *startCall(boolean,String, CallCommunication, CallView)*, avente come parametri se l'utente è il chiamante, quindi in questo caso verrà inviato false, il tipo di chiamata e i riferimenti alle due classi

- **[void] createPeerConnection():** metodo per inizializzare *peerConnection* e aggiungere gli ascoltatori per l'aggiunta e la rimozione dello stream remoto
  - **[void] onRemoteStreamAdded(Event):** metodo per impostare *remoteStream* allo stream remoto e mandare l'evento *setPeerConn* che verrà raccolto e gestito dalla classe StatisticsView
  - **[void] onRemoteStreamRemoved(Event):** metodo per rimuovere *localStream* da *peerConnection* (ancora non supportato da WebRTC)
- **[void] connect(boolean):** metodo per inizializzare una RTCPeerConnection invocando il metodo *createPeerConnection()*, aggiungendo a *peerConnection* il proprio *localStream* e invocando il metodo *createOffer(RTCSessionDescriptionCallback)* di RTCPeerConnection
- **[void] gotDescription(RTCSessionDescription):** metodo per impostare il proprio sdp tramite il metodo *setLocalDescription(RTCSessionDescription)* di RTCPeerConnection ed inviarlo all'utente con cui si vuole comunicare
- **[void] onIceCandidate(Event):** metodo per generare i candidati e memorizzarli in *candidates* e, una volta salvati tutti, inviarli all'utente remoto se sarà pronto a riceverli, altrimenti invierà al server un messaggio che indichi la disponibilità a ricevere candidati
- **[void] startCall(boolean, String, CallCommunication, CallView):** metodo per avviare la comunicazione tra due utenti. Imposterà *sourcevid* e *remotevid*, inizializzerà il listener *onMessage(Event)* e nel caso l'utente sia il chiamante imposterà lo stream audio o audio/video e invocherà il metodo *connect(boolean)*
  - **[void] onMessage(Event):** metodo per gestire messaggi dal server, in base al pacchetto ricevuto effettuerà una determinata serie di azioni:
    - se il pacchetto conterrà come tipo *offer*, e sarà inviato dal chiamante, imposterà *localStream*, inizializzerà *peerConnection* ed imposterà il RTCSessionDescription ricevuto e invierà il proprio RTCSessionDescription
    - se il pacchetto sarà di tipo *answer*, e sarà inviato dal chiamato, imposterà in *peerConnection* il RTCSessionDescription ricevuto
    - se il pacchetto conterrà il tipo *candidateready* imposterà a *true* *messageReceived* e nel caso in cui *readyToSend* sia *true* invierà *candidates*
    - se il pacchetto conterrà il tipo *candidate* preleverà il RTCIceCandidate e imposterà in *peerConnection*

    * se il pacchetto conterrà il tipo *endcall* interromperà *localStream*, chiuderà *peerConnection* e lancerà l'evento *setOnCall,* che NotificationCommunication raccoglierà per impostare la variabile *onCalling* a *false*, e invocherà il metodo *endCall(bool)* di CallView

- **[void] recoverCall():** metodo per ripristinare la visione dello stream nel caso in cui fosse stata interrotta selezionando un altro utente
- **[void] endCall():** metodo che interromperà *localStream*, chiuderà *peerConnection* e lancerà l'evento *setOnCall,* che NotificationCommunication raccoglierà per impostare la variabile *onCalling* a *false,* infine invierà al server un pacchetto, contrassegnato dal tipo *endcall* contenente lo username del contatto con cui si era in chiamata

4.2.3 `mytalk.client.communication.ChatCommunication`

La classe ChatCommunication si occuperà di comunicare con il server per quanto riguarda il trasferimento e la gestione di messaggi testuali

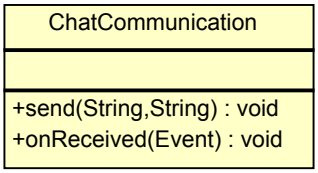

Figura 39: Classe ChatCommunication

Relazione d'uso con altri moduli

- **La classe utilizza**
  - `mytalk.client.collection.TextMessagesCollection`
- **La classe è utilizzata da:**
  - `mytalk.client.view.ChatView`
- **La classe comunica con:**
  - `mytalk.server.transfer.ChatTransfer`

Metodi pubblici:

- **[void] send(String, String):** metodo che si occuperà di inviare un pacchetto al server, contrassegnato con il tipo *sendText* contenente il contatto con cui si sta comunicando via chat e il messaggio che si vuole inviare
- **[void] onReceived(Event):** metodo di tipo listener per la ricezione e la gestione di segnali riguardanti i messaggi testuali; se il segnale significherà l'arrivo di un nuovo messaggio, sarà quindi caratterizzati dal tipo *sendText,* lo aggiungerà alla collezione di tipo TextMessagesCollection, mentre se indicherà errori nella consegna del testo, tipo *notDelivered*, avviserà l'utente e imposterà lo stato del messaggio, presente nella collection, come *notsent*

4.2.4 `mytalk.client.communication.ContactsCommunication`

La classe ContactsCommunication si occuperà di prelevare la lista degli utenti presenti nel server e di aggiornarla nel caso di cambiamenti

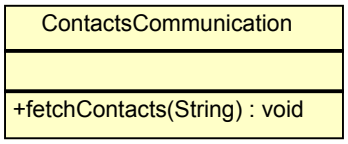

Figura 40: Classe ContactsCommunication

Relazione d'uso con altri moduli

- **La classe utilizza:**
  - `mytalk.client.collection.ContactsCollection`
- **La classe è utilizzata da:**
  - `mytalk.client.view.ChatView`
- **La classe comunica con:**
  - `mytalk.server.transfer.AuthenticationTransfer`

Metodi pubblici:

- **[void] fetchContacts(String):** metodo per prelevare dal server la lista dei contatti registrati, per fare ciò invierà al server un pacchetto di tipo *getContacts* contenente lo username dell'utente ed attenderà la risposta, che consisterà in un pacchetto, contrassegnato con lo stesso tipo, contenente il numero di contatti e la lista con i loro dati. Per ogni contatto controllerà che esso non sia già presente nella lista utenti, se affermativo aggiungerà il contatto nella collection, altrimenti aggiornerà l'attributo *IP* del *ContactUser* corrispondente

4.2.5 `mytalk.client.communication.FileCommunication`

La classe FileCommunication si occuperà di gestire l'invio di file ad un altro utente e la ricezione

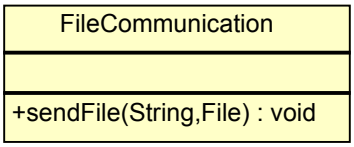

Figura 41: Classe FileCommunication

Relazione d'uso con altri moduli

- **La classe è utilizzata da:**
  - `mytalk.client.view.FileView`
- **La classe comunica con:**
  - `mytalk.server.transfer.FileTransfer`

Metodi pubblici:

- **[void] sendFile(String, File):** metodo per inviare un file salvato in locale, invierà al server un pacchetto, caratterizzato dal tipo *file,* contenente il file e lo username del destinatario

4.2.6 `mytalk.client.communication.NotificationCommunication`

La classe NotificationCommunication si occupa di comunicare con il server per quanto riguarda le notifiche

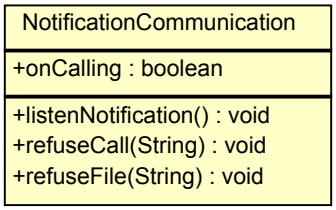

Figura 42: Classe NotificationCommunication

Relazione d'uso con altri moduli

- **La classe utilizza:**
  - `mytalk.client.collection.RecordMessagesCollection`
  - `mytalk.client.view.NotificationView`
- **La classe comunica con:**
  - `mytalk.server.transfer.CallTransfer`
  - `mytalk.server.transfer.FileTransfer`
  - `mytalk.server.transfer.RecordMessageTransfer`

Attributi pubblici:

- **[boolean] onCalling:** variabile che indicherà se l'utente avrà una chiamata in corso

Metodi pubblici:

- **[void] listenNotification():** metodo di tipo listener per gestire gli eventi di tipo *setOnCall* e *message,* in arrivo
  - **[void] SetOnCall[Event]** imposta *onCalling* a seconda del tipo racchiuso nel parametro passatogli
  - **[void] onNotification(String):** metodo per gestire le notifiche in ingresso, che potranno essere chiamate, messaggi video o file.

  - Se si tratta di una chiamata in arrivo, quindi il pacchetto risulterà contrassegnato dal tipo *call*, se l'utente non risulterà occupato in altre conversazioni verrà creata una istanza NotificationView e impostato il timeout, in modo da terminare la chiamata nel caso non si risponda dopo un determinato intervallo di tempo. Se l'utente risulterà invece già in chiamata, si manderà un messaggio al server, di tipo *busy* per avvisare il chiamante
  - Se si tratta della terminazione di una chiamata, quindi il pacchetto risulterà contrassegnato dal tipo *endcall*, si assegnerà *false* alla variabile onCalling
  - Se si tratta di un file in arrivo, pacchetto contrassegnato dal tipo *file*, verrà creata e avviata una istanza di NotificationView con i relativi dati
  - Se si tratta di un messaggio video in arrivo, pacchetto contrassegnato dal tipo *record,* verrà creato ed aggiunto il messaggio alla collection, successivamente verrà creata e avviata una istanza di NotificationView con i relativi dati

- **[void] refuseCall(String):** metodo per rispondere ad una richiesta di chiamata con un rifiuto, inviando un pacchetto con tipo *refuseCall*, contenente lo username del chiamante, allo strato transfer
- **[void] refuseFile(String):** metodo per rifiutare la ricezione di un file, invierà un pacchetto con tipo *refuse*File, contenente lo username del mittente, allo strato transfer

4.2.7 `mytalk.client.communication.RecordMessageCommunication`

La classe RecordMessageCommunication si occuperà di registrare, inviare e ricevere i messaggi registrati

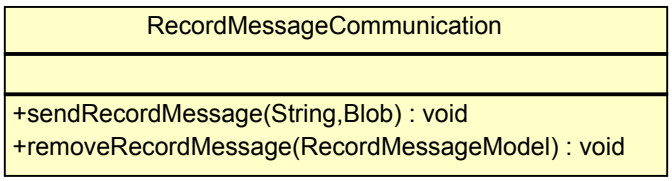

Figura 43: Classe RecordMessageCommunication

Relazione d'uso con altri moduli

- **La classe utilizza:**

  - `mytalk.client.collection.RecordMessageCollection`

- **La classe è utilizzata da:**
  - `mytalk.client.view.RecordMessageView`
- **La classe comunica con:**
  - `mytalk.server.transfer.RecordMessageTransfer`

Metodi pubblici:

- **[void] sendRecordMessage(String, Blob):** metodo per inviare un messaggio video/audio, per fare ciò invierà al server un pacchetto con tipo *sendRecord* e contenente il nome del destinatario e il file audio/video
- **[void] removeRecordMessage(RecordMessageModel):** metodo per eliminare un messaggio video/audio, invierà al server la richiesta, con tipo *removeRecord,* ed eliminerà il messaggio dalla collection

4.2.8 `mytalk.client.communication.TutorialCommunication`

La classe TutorialCommunication si occuperà di comunicare con il server per il prelievo dei tutorial video

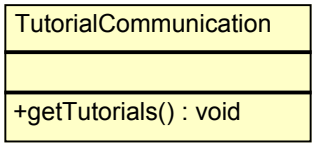

Figura 44: Classe TutorialCommunication

Relazione d'uso con altri moduli

- **La classe utilizza:**
  - `mytalk.client.collection.TutorialsCollection`
- **La classe è utilizzata da:**
  - `mytalk.client.view.TutorialView`
- **La classe comunica con:**
  - `mytalk.server.transfer.AuthenticationTransfer`

Metodi pubblici:

- **[void] getTutorials():** metodo per prelevare dal server la lista dei tutorial registrati, rappresenterà un listener che riceverà un pacchetto, contrassegnato dal tipo *tutorials*, contenente la lista dei tutorial, infine aggiungerà ciascuno di questi nella collection

4.2.9 `mytalk.client.communication.UserDataCommunication`

La classe UserDataCommunication si occuperà di comunicare con il server per la modifica dei dati dell'utente

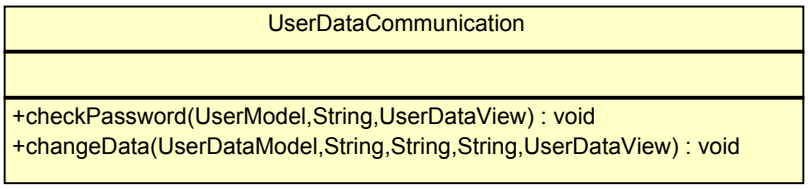

Figura 45: Classe UserDataCommunication

Relazione d'uso con altri moduli

- **La classe utilizza:**
  - `mytalk.client.model.UserDataModel`
- **La classe è utilizzata da:**
  - `mytalk.client.view.UserDataView`
- **La classe comunica con:**
  - `mytalk.server.transfer.UserDataTransfer`

Metodi pubblici:

- **[void] checkPassword(UserModel, String, UserDataView):** metodo per controllare che la password di verifica inserita dall'utente corrisponda a quella salvata nella base di dati. Per fare ciò invierà al server un pacchetto, contrassegnato dal tipo *checkCredentials* e contenente lo username dell'utente e la password inserita, se il pacchetto di risposta, contrassegnato dallo stesso tipo, sarà affermativo verrà invocato il metodo *view.callBacks.changeData(UserModel, UserDataView)*, altrimenti mostrerà un messaggio di errore

- **[void] changeData(UserModel, String, String, String, UserDataView):** metodo che invierà i dati aggiornati al server, racchiudendoli in un pacchetto di tipo *changeData* ed attenderà la risposta. Se il pacchetto di risposta, contrassegnato dallo stesso tipo, sarà affermativo si aggiorneranno i dati del model e si chiamerà il metodo *render()* di UserDataView, altrimenti verrà mostrato un messaggio di errore

4.3 Package Collection

Nel package `mytalk.client.collection` saranno presenti le classi che conterranno le liste di oggetti di tipo Model. Non sarà presente una classe del package Collection per ogni classe del package Model

Relazione d'uso con altri moduli

- **La classe utilizza:**
  - `mytalk.client.model`
- **La classe è utilizzata da:**
  - `mytalk.client.view`
  - `mytalk.client.communication`

Costituito dalle classi:

- `mytalk.client.collection.ContactsCollection` (vedasi sezione 4.3.1)
- `mytalk.client.collection.RecordMessagesCollection` (vedasi sezione 4.3.2)
- `mytalk.client.collection.TextMessagesCollection` (vedasi sezione 4.3.3)
- `mytalk.client.collection.TutorialsCollection` (vedasi sezione 4.3.4)

4.3.1 `mytalk.client.collection.ContactsCollection`

La classe ContactsCollection rappresenterà la collezione dei contatti registrati

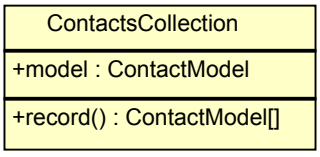

Figura 46: Classe ContactsCollection

Relazione d'uso con altri moduli

- **La classe utilizza:**
  - `mytalk.client.model.ContactModel`
- **La classe è utilizzata da:**
  - `mytalk.client.view.SideView`
  - `mytalk.client.communication.ContactsCommunication`
- **La classe deriva da:**
  - `mytalk.client.Backbone.Collection`

Attributi pubblici:

- **[ContactModel] model:** conterrà il riferimento alla classe ContactModel

Metodi pubblici:

- **[ContactModel[]] record():** metodo per restituire un array contenente tutti i ContactModel della collezione

4.3.2 `mytalk.client.collection.RecordMessagesCollection`

La classe RecordMessagesCollection rappresenterà la collezione dei messaggi audio e video ricevuti

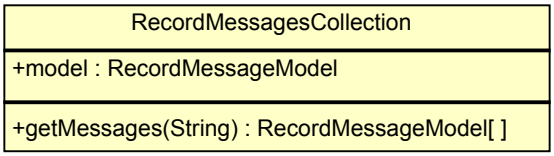

Figura 47: Classe RecordMessagesCollection

Relazione d'uso con altri moduli

- **La classe utilizza:**
  - `mytalk.client.model.RecordMessageModel`
- **La classe è utilizzata da:**
  - `mytalk.client.view.RecordMessageView`
  - `mytalk.client.communication.NotificationCommunication`
  - `mytalk.client.communication.RecordMessageCommunication`
- **La classe deriva da:**
  - `mytalk.client.Backbone.Collection`

Attributi pubblici:

- **[RecordMessageModel] model:** conterrà il riferimento alla classe RecordMessageModel

Metodi pubblici:

- **[RecordMessageModel[]] getMessages(String):** metodo per restituire un array contenente i RecordMessageModel della collezione, avente come mittente lo username passato per parametro

4.3.3 `mytalk.client.collection.TextMessagesCollection`

La classe TextMessagesCollection rappresenterà la collezione dei messaggi di testo di una singola conversazione

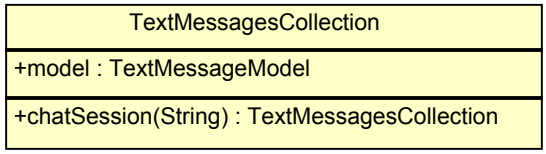

Figura 48: Classe TextMessagesCollection

Relazione d'uso con altri moduli

- **La classe utilizza:**
  - `mytalk.client.model.TextMessageModel`
- **La classe è utilizzata da:**
  - `mytalk.client.view.ChatView`
  - `mytalk.client.communication.ChatCommunication`
- **La classe deriva da:**
  - `mytalk.client.Backbone.Collection`

Attributi pubblici:

- **[TextMessageModel] model:** conterrà il riferimento alla classe TextMessageModel

Metodi pubblici:

- **[TextMessagesCollection] chatSession(String):** metodo per restituire l'intera conversazione con l'utente il cui username sarà parametro della funzione

4.3.4 `mytalk.client.collection.TutorialsCollection`

La classe TutorialsCollection rappresenterà la collezione dei tutorial video disponibili all'utente

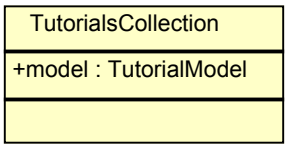

Figura 49: Classe TutorialsCollection

Relazione d'uso con altri moduli

- **La classe utilizza:**
  - `mytalk.client.model.TutorialModel`
- **La classe è utilizzata da:**
  - `mytalk.client.view.TutorialView`
  - `mytalk.client.communication.TutorialCommunication`
- **La classe deriva da:**
  - `mytalk.client.Backbone.Collection`

Attributi pubblici:

- **[TutorialModel] model:** conterrà il riferimento alla classe TutorialModel

4.4 Package Model

Il package `mytalk.client.model` costituirà la parte del sistema dedita alla permanenza ed alla gestione dei dati

Relazione d'uso con altri moduli

- **Il package viene utilizzato da:**
  - `mytalk.client.collection`
  - `mytalk.client.view`
  - `mytalk.client.communication`

Costituito dalle classi:

- `mytalk.client.model.ContactModel` (vedasi sezione 4.4.1)
- `mytalk.client.model.RecordMessageModel` (vedasi sezione 4.4.2)
- `mytalk.client.model.TextMessageModel` (vedasi sezione 4.4.3)
- `mytalk.client.model.TutorialModel` (vedasi sezione 4.4.4)
- `mytalk.client.model.UserModel` (vedasi sezione 4.4.5)

4.4.1 `mytalk.client.model.ContactModel`

La classe ContactModel rappresenterà un singolo contatto registrato

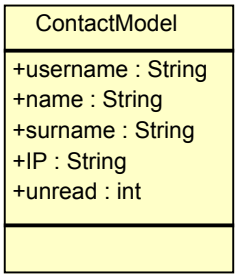

Figura 50: Classe ContactModel

Relazione d'uso con altri moduli

- **La classe è utilizzata da:**
  - `mytalk.client.view.ContactView`
  - `mytalk.client.view.FunctionsView`
  - `mytalk.client.communication.CallCommunication`
  - `mytalk.client.collection.ContactsCollection`
- **La classe deriva da:**
  - `mytalk.client.Backbone.Model`

Attributi pubblici:

- **[String] username**: username del contatto
- **[String] name**: nome del contatto
- **[String] surname**: cognome del contatto
- **[String] IP**: indirizzo IP del contatto
- **[int] unread:** numero di messaggi non ancora letti, inviati dal contatto

4.4.2 `mytalk.client.model.RecordMessageModel`

La classe RecordMessageModel rappresenterà un messaggio video/audio inviato all'utente

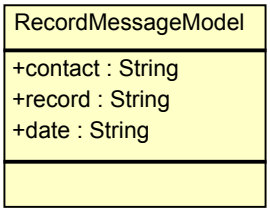

Figura 51: Classe RecordMessageModel

Relazione d'uso con altri moduli

- **La classe è utilizzata da:**
  - `mytalk.client.collection.RecordMessagesCollection`
- **La classe deriva da:**
  - `mytalk.client.Backbone.Model`

Attributi pubblici:

- **[String] contact:** mittente del messaggio
- **[String] record:** indirizzo url del server in cui è salvato il messaggio
- **[String] date:** data di creazione del messaggio

4.4.3 `mytalk.client.model.TextMessageModel`

La classe TextMessageModel rappresenterà un messaggio testuale inviato tra due utenti

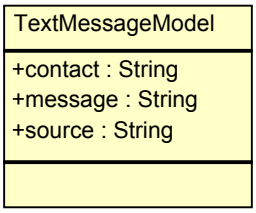

Figura 52: Classe TextMessageModel

Relazione d'uso con altri moduli

- **La classe è utilizzata da:**
  - `mytalk.client.collection.TextMessagesCollection`
- **La classe deriva da:**
  - `mytalk.client.Backbone.Model`

Attributi pubblici:

- **[String] contact:** mittente del messaggio testuale
- **[String] message:** contenuto del messaggio testuale
- **[String] source:** stato del messaggio
  - *sent* : se il messaggio è stato inviato al contatto (e *notsent*: se ci sono stati problemi nella consegna)
  - *received:* se il messaggio è stato inviato dal contatto

4.4.4 `mytalk.client.model.TutorialModel`

La classe TutorialModel rappresenterà il collegamento ad un video tutorial

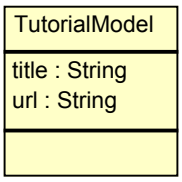

Figura 53: Classe TutorialModel

Relazione d'uso con altri moduli

- **La classe è utilizzata da:**
  - `mytalk.client.collection.TutorialsCollection`
- **La classe deriva da:**
  - `mytalk.client.Backbone.Model`

Attributi pubblici:

- **[String] title:** titolo del tutorial
- **[String] url:** indirizzo url del server in cui è salvato il tutorial

4.4.5 `mytalk.client.model.UserModel`

La classe UserModel rappresenterà i dati dell'utente attivo

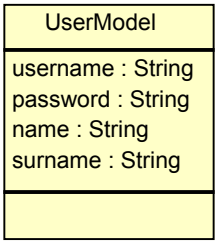

Figura 54: Classe UserModel

Relazione d'uso con altri moduli

- **La classe è utilizzata da:**
  - `mytalk.client.view.AuthenticationView`
  - `mytalk.client.view.SideView`
  - `mytalk.client.view.UserDataView`
  - `mytalk.client.communication.UserDataCommunication`
- **La classe deriva da:**
  - `mytalk.client.Backbone.Model`

Attributi pubblici:

- **[String] username**: username del utente
- **[String] password:** password dell'utente
- **[String] name**: nome del utente
- **[String] surname**: cognome del utente

4.5 template

`mytalk.client.template` costituisce la parte del sistema che implementa l'interfaccia web usufruibile dagli utenti mediante pagine web, visualizzata in base alle esigenze.

4.5.1 `mytalk.client.template.AuthenticationTemplate`

Questo template crea gli elementi dell'interfaccia grafica che l'utente visualizzerà necessari per effettuare le operazioni di *login* e registrazione e, una volta connessi, per accedere alla sezione di modifica dati ed effettuare il *logout*
Il template è utilizzato da: `mytalk.client.view.AuthenticationView`
Visualizzazione della pagina:

- **Utente non autenticato:**
  - **input#user:** input di tipo text, in cui l'utente dovrà inserire il proprio username per effettuare l'accesso
  - **input#password:** input di tipo password, in cui l'utente dovrà inserire la propria password per effettuare l'accesso
  - **button#login:** bottone per effettuare la *login*
  - **button#signup:** bottone per visualizzare il modulo di registrazione
- **Registrazione:**
  - **input#user:** input di tipo text, in cui l'utente dovrà inserire il proprio username per effettuare la registrazione
  - **input#password:** input di tipo password, in cui l'utente dovrà inserire la propria password per effettuare la registrazione
  - **input#password2:** input di tipo password, in cui l'utente dovrà inserire nuovamente la propria password per controllare che quella inserita precedentemente sia quella desiderata
  - **input#name:** input di tipo text, in cui l'utente potrà inserire il proprio nome
  - **input#surname:** input di tipo text, in cui l'utente potrà inserire il proprio cognome

  - **button#sign:** bottone per effettuare la registrazione
  - **button#deny:** bottone per annullare la registrazione, e chiudere il modulo di registrazione

- **Utente autenticato:**
  - **button#logout:** bottone per effettuare la *logout*
  - **button#edit:** bottone che farà accedere alla sezione per modificare i propri dati

4.5.2 `mytalk.client.template.CallTemplate`

Questo template crea gli elementi dell'interfaccia grafica che visualizzerà l'utente durante una chiamata in corso.

- **Il template è utilizzato da:** `myTalk.client.view.CallView`
- **Chiamata avviata:**
  - **video#remotevid:** video di tipologia autoplay, corrisponderà al video remoto
  - **video#sourcevid:** video di tipologia autoplay, corrisponderà al video locale
  - **button#endCall:** bottone che terminerà la chiamata

4.5.3 `mytalk.client.template.ChatTemplate`

Questo template crea gli elementi dell'interfaccia grafica che l'utente visualizzerà quando la chat è attiva.

- **Il template è utilizzato da:** `myTalk.client.view.ChatView`
- **Chat attivata:**
  - **textarea#compose:** area testuale per la scrittura dei messaggi testuali
  - **button#Send:** bottone che invierà il messaggio testuale

4.5.4 `mytalk.client.template.ContactTemplate`

Questo template crea gli elementi dell'interfaccia grafica dove l'utente visualizzerà la separazione degli utenti online da quelli offline nella lista degli utenti, e quando richiederà di avviare una video conferenza, mostrerà la possibilità di scegliere gli utenti partecipanti

- **Il template è utilizzato da:** `myTalk.client.view.ContactView`
- **Utente autenticato:**
  - verrà verificato l'indirizzo ip per ogni utente registrato al server e, se un utente è connesso verrà visto nella sidebar "online", altrimenti "offline"
- **Richiesta di video conferenza:**
  - **input#username:** input di tipo checkbox, dove l'utente decide i partecipanti alla video conferenza

4.5.5 `mytalk.client.template.FunctionsTemplate`

Questo template crea gli elementi dell'interfaccia grafica che l'utente visualizzerà necessari per effettuare le operazioni che si possono compiere dopo aver contattato un utente inserendo l'indirizzo IP, oppure selezionandolo dalla lista degli utenti.

- **Il template è utilizzato da:** `myTalk.client.view.FunctionsView`
- **Contatto tramite l'inserimento dell'indirizzo IP:**
  - **input#ip:** input di tipo text dove l'utente dovrà inserire l'indirizzo ip che vorrà contattare
  - **button#StartChat:** button che avvierà la chat
  - **button#call:** bottone che avvierà la chiamata
  - **button#video:** bottone che avvierà la video chiamata
  - **input#record:** input di tipo checkbox che registrerà la chiamata
- **Utente offline:**
  - **button#dataContact:** bottone che farà visualizzare i dettagli del contatto
  - **button#sendVideoText:** bottone che farà visualizzare le opzioni per registrare un video messaggio
- **Utente online:**
  - **button#call:** bottone che avvierà la chiamata
  - **button#video:** bottone che avvierà la video chiamata
  - **input#record:** bottone che registrerà la chiamata

4.5.6 `mytalk.client.template.NotificationTemplate`

Questo template crea gli elementi dell'interfaccia grafica che farà visualizzare all'utente destinatario necessari per accettare o rifiutare una chiamata dal mittente

- **Il template è utilizzato da:** `myTalk.client.view.NotificationView`
- **Notifica della ricezione di una chiamata:**
  - **button#acceptCall:** bottone che avvierà la chiamata
  - **button#refuseCall:** bottone che rifiuterà la chiamata

4.5.7 `mytalk.client.template.RecordMessageTemplate`

Questo template crea gli elementi dell'interfaccia grafica che l'utente visualizzerà quando vorrà registrare un video messaggio

- **Il template è utilizzato da:** `myTalk.client.view.RecordMessageView`
- **Registra un video messaggio:**
  - **video#live_video:** video di tipologia autoplay e controls, così l'utente in maniera tale da poter fermare la registrazione quando lo si desidera

4.5.8 `mytalk.client.template.SideTemplate`

Questo template creerà gli elementi dell'interfaccia grafica che l'utente visualizzerà nella sidebar

- **Il template è utilizzato da:** `myTalk.client.view.SideView`
- **Sidebar:**
  - **button#callIP:** bottone che permette di contattare una persona tramite l'indirizzo IP
  - **ul#contacts:** lista degli utenti registrati presso il server
  - **button#conference:** bottone che permetterà all'utente di selezionare i partecipanti alla video conferenza

4.5.9 `mytalk.client.template.StatisticsTemplate`

Questo template crea gli elementi dell'interfaccia grafica riguardanti le statistiche che l'utente visualizzerà durante una chiamata

- **Il template è utilizzato da:** `myTalk.client.view.StatisticsView`
- **Statistiche:**
  - Durata della chiamata: tempo di durata della chiamata dall'avvio della chiamata
  - Byte audio trasmessi: KB trasmessi dall'avvio della chiamata
  - Byte video trasmessi: KB trasmessi dall'avvio della chiamata
  - Latenza: tempo di latenza in ms
  - Velocità di trasmissione: KB/s trasmessi

4.5.10 `mytalk.client.template.UserDataTemplate`

Questo template crea gli elementi dell'interfaccia grafica che l'utente visualizzerà quando accederà alla sezione per modificare i propri dati

- **Il template è utilizzato da:** `myTalk.client.view.UserDataView`
- **Modifica dati:**
  - **input#name:** input di tipo text, dove l'utente potrà inserire il proprio nome
  - **input#surname:** input di tipo text, dove l'utente potrà inserire il poprio cognome
  - **input#password:** input di tipo password, dove l'utente potrà inserire la nuova password
  - **input#password2:** input di tipo password, dove l'utente dovrà inserire la conferma della nuova password
  - **input#oldPassword:** input di tipo password, dove l'utente dovrà inserire la password attuale
  - **button#submitChange:** bottone che confermerà le modifiche
  - **button#reset:** bottone che resetterà gli input del modulo di modifica dei dati
  - **button#denyChange:** bottone che annullerà le modifiche e chiuderà il modulo di modifica dei dati

5 Tracciamento requisiti-componenti-classi

Per quanto riguarda tale tracciamento si rimanda al documento `Specifica_Tecnica_v2.0.pdf`, sez. *Tracciamento componenti-requisiti* e *Tracciamento requisiti-componenti*.