responsabile@hourglabs.org

Definizione di Prodotto

Versione 2.0

Informazioni documento

Nome	*Definizione_di_Prodotto_v2.0.pdf*
Versione	2.0
Data creazione	2013-06-20
Data ultima modifica	2013-09-16
Stato del Documento	Formale ad uso esterno
Redazione	Thomas Rossetto Giovanni Morlin Paolo Bustreo Sasa Ilievski Riccardo Cesarotto Umberto Martinati
Verifica	Paolo Bustreo Giovanni Morlin
Approvazione	Sasa Ilievski
Distribuzione	hourglass Prof. Tullio Vardanega Prof. Riccardo Cardin

Registro delle modifiche

Data	Versione	Ruolo	Descrizione	Autore
2013-09-16	2.0	Responsabile	Approvazione versione finale documento	Thomas Rossetto
2013-09-14	1.4	Verificatore	Verifica documento	Giovanni Morlin
2013-09-11	1.3	Progettista	Correzione schemi UML errati segnalati in sede di revisione	Gioele Lorenzo Cresce
2013-09-01	1.2	Programmatore	Aggiunta sezione Introduzione e Package display	Paolo Bustreo
2013-08-22	1.1	Progettista	Aggiornamento codifica metodi	Riccardo Cesarotto
2013-05-28	1.0	Responsabile	Validazione documento	Sasa Ilievski
2013-05-28	0.9	Verificatore	Verifica documento	Giovanni Morlin, Paolo Bustreo
2013-05-26	0.8.1	Progettista	Aggiornamento relazioni d'uso e metodi classi	Thomas Rossetto
2013-05-23	0.8	Progettista	Stesura ContactList e Message (presenter e view)	Riccardo Cesarotto

Data	Versione	Ruolo	Descrizione	Autore
2013-05-20	0.7	Progettista	Inizio stesura componenti per requisiti opzionali : classe ContactProfile (presenter e view), stesura subpackage view.contactprofile	Umberto Martinati
2013-03-22	0.6.1	Responsabile	Validazione documento	Paolo Bustreo
2013-03-22	0.6	Verificatore	Verifica documento	Giovanni Morlin, Sasa Ilievski
2013-03-21	0.5	Progettista	Stesura componenti package view e restanti componenti	Paolo Bustreo
2013-03-19	0.4	Progettista	Stesura componenti server	Thomas Rossetto
2013-03-18	0.3	Progettista	Stesura componenti package presenter	Giovanni Morlin
2013-03-14	0.2	Progettista	Stesura componenti package communication, appcontroller ed event	Thomas Rossetto
2013-03-04	0.1	Progettista	Stesura scheletro documento	Thomas Rossetto

Indice

1 Introduzione

Si è deciso di strutturare il documento proponendo per ogni sottosistema i diagrammi delle componenti, diagrammi che rimanderanno di volta in volta alle definizioni delle singole classi che li compongono.

1.1 Scopo del documento

Il presente documento ha lo scopo di definire nel dettaglio la struttura del sistema MyTalk, approfondendo la struttura tecnica della piattaforma già presentata nel documento *Specifica Tecnica v2.0*. I programmatori dovranno seguire le specifiche definite in questo documento per la loro azione di codifica evitando di apportare alcuna modifica: per tale motivo il sistema viene descritto in questo documento approfonditamente e completamente.

1.2 Scopo del prodotto

Il progetto MyTalk consiste nello sviluppare un software che permetta la comunicazione tra utenti tramite il browser, utilizzando solo componenti standard senza dover installare programmi o plugin esterni. Gli utenti potranno chiamare altri utenti, iniziare la comunicazione sia audio che video, svolgere la chiamata e terminare la chiamata ottenendo delle statistiche sull'attività. Il prodotto sviluppato dovrà offrire agli utenti un'interfaccia grafica di semplice comprensione e di facile utilizzo.

1.3 Glossario

Per avere una maggiore comprensione dei termini tecnici e degli acronimi presenti in questo documento, (sottolineati alla loro prima occorrenza), si rimanda alla consultazione del glossario allegato, nominato `Glossario_v3.0.pdf`.

1.4 Riferimenti

1.4.1 Riferimenti normativi

- Norme di Progetto, v4.0 (allegato `Norme_di_Progetto_v4.0.pdf`)
- Specifica tecnica, v3.0 (allegato `Specifica_Tecnica_v3.0.pdf`)
- Analisi dei Requisiti, v4.0 (allegato `Analisi_dei_Requisiti_v4.0.pdf`)

1.5 Standard di progettazione

1.6 Diagrammi UML

Tutti i diagrammi UML presenti in questo documento sono rappresentati in conformità dello standard UML 2.0.

1.7 Notazione

Per facilitare la lettura e la comprensibilità del documento sono state adottate le seguenti convenzioni:

- Per ogni package sarà presentata una breve descrizione generica sulla funzione delle classi di tale package.
- Iniziare la descrizione di una nuova classe di un package (tranne la prima) in una pagina nuova.
- Per la descrizione dei metodi di una classe si utilizza la notazione UML 2.0 utilizzando come marcatori d'accesso di metodi e attributi i seguenti simboli:

  \+ per indicare il marcatore d'accesso **public**

  \# per indicare il marcatore d'accesso **protected**

  \- per indicare il marcatore d'accesso **private**
- Evidenziare con il colore blue i metodi di una classe.
- Evidenziare con il colore orange gli attributi di una classe.
- Evidenziare con il colore purple le classi e le interfacce che una classe utilizza.
- Per facilitare la lettura quando si riferisce ad un'altra classe non presente nel package della classe che la riferisce si omette il percorso `it.hourglass.myTalk.client`

2 Specifica

2.1 Introduzione all'architettura

L'architettura si può dividere in due sezioni: Client e Server.

`Client` Il Client è la sezione dell'architettura con cui l'utente può interagire senza richiedere dati dal database ed è costituito da 8 package principali:

- appcontroller
- communication
- event
- presenter
- view
- rpcservice
- shared
- wrapper

Il package appcontroller contiene la classe appcontroller la quale si occupa della creazione della connessione *WebSocket*, inizializzare la classe `Presenter.BaseViewPresenter` e a ricevere gli eventi (definiti nel package event) lanciati dalle classi del package presenter. Le classi del presenter si occupano di gestire la logica riguardante le operazioni che il sistema può eseguire (autenticazione (e de-autenticazione), chiamata, invio e lettura messaggi (e richieste di amicizia) e registrazione). Per ridurre la complessità e l'accoppiamento si è deciso di creare la classe `Presenter.BaseViewPresenter`. Questa classe funge da presenter base al sistema. Esso contiene 3 presenter: un presenter per il menù, il presenter per il body, un presenter per il footer e un presenter per la lista contatti.Il base presenter si occupa di cambiare il presenter in uso (tale richiesta viene fatta tramite un evento che l'appcontroller riceve e richiede lo *switch*. Ogni classe presenter contiene un istanza della view corrispettiva. Ogni view implementerà un'interfaccia denominata Display che espone i metodi che il presenter può richiedere alla view. Le classi in communication si occupano della comunicazione con il server *WebSocket* e per creare la comunicazione *WebRTC* per le chiamate. Infine il package rpcservice contiene le interfacce necessarie alla comunicazione e gestione dei dati dal database utilizzando le chiamate asincrone RPC.

`Server` Il server è la parte del sistema che si occupa di gestire il recupero dei dati dal database e la connessione con l'apposito server *WebSocket* necessario per lo scambio dei messaggi *JSON*. E' costituito da 2 package:

- model

- WSServer

Il package model contiene le classi che contengono la logica del sistema che si occupa del recupero (e richiesta di salvataggio) dei dati nel database. La classe `model.businessmodel.RPCServiceImpl` contiene la definizione dei metodi esposti dall'interfaccia in `client.rpcservice.RPCService`. Le classi nel package DAO si occupano del collegamento con il database. Il WSServer è il server che si occupa della ricezione e dell'invio dei messaggi JSON necessari per instaurare una chiamata e a ricevere le notifiche per la ricezione di un messaggio o di una richiesta di amicizia.

2.2 Package it.hourglass.myTalk.client

Questo package contiene le classi necessarie all'inizializzazione e controllo del programma myTalk. Vengono inoltre definiti i metodi per la gestione della chiamata e della connessione, e scambio di messaggi JSON con il server WebSocket.

2.2.1 client.MyTalk - Classe

Funzione

Questa classe è la prima che viene caricata all'avvio del programma e si occupa di inizializzare l'istanza unica di `HandlerManager` (classe fornita dal framework GWT) utilizzata per la gestione degli eventi dal client myTalk e l'unica istanza della classe `AppController` la quale si occupa di inizializzare il client e di gestire le azioni dello stesso.

Relazione d'uso con altri moduli

- La classe implementa:

  `com.google.gwt.core.client.EntryPoint`

Attributi

- `HandlerManager eventBus`
  Questo oggetto di tipo HandlerManager è l'unico HandlerManager presente nel sistema. Esso verrà passato ad `controller` attraverso il quale gestirà gli eventi.
- `AppController controller`
  Oggetto di tipo AppController che viene creato all'avvio del sistema. Esso avrà il compito di inizializzare il client `myTalk`.

Metodi

+ `void onModuleLoad()`
  Inizializza le variabili `eventBus` (istanza della classe `HandlerManager`) e `controller` (istanza della classe `AppController`).

2.3 Package it.hourglass.myTalk.client.communication

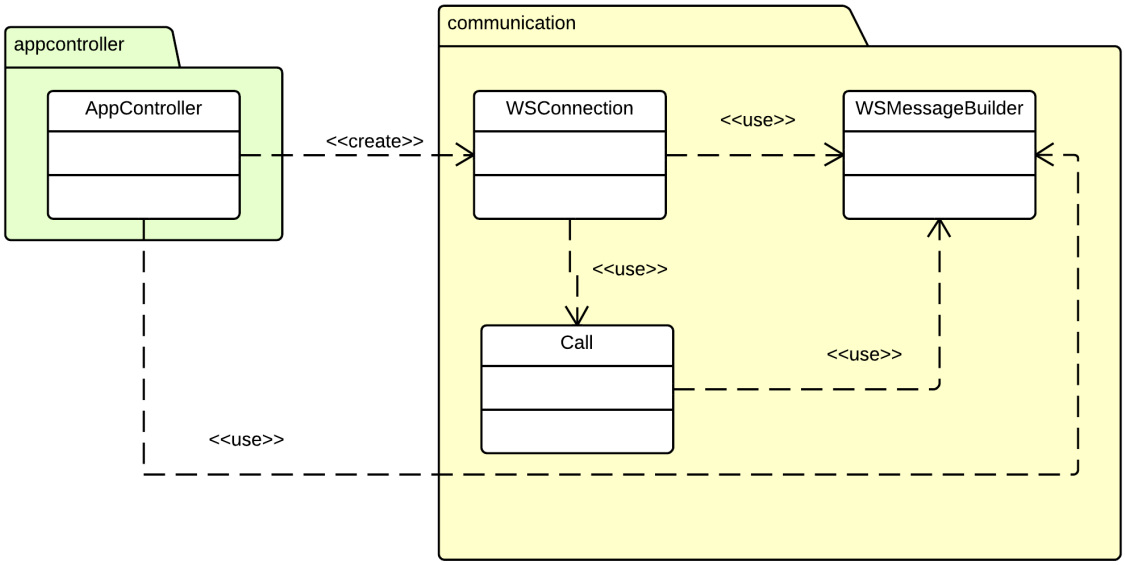

Figura 1: Package it.hourglass.myTalk.client.communication

2.3.1 client.communication.WSConnection - Classe

Funzione

Questa classe viene utilizzata per la creazione e la gestione della connessione al web-socket della servlet WSServer che si occupa dello smistamento dei messaggi JSON e degli utenti online. Questa classe si occupa inoltre della creazione e della gestione della chiamata.

Relazione d'uso con altri moduli

- La classe è utilizzata da:

  `it.hourglass.myTalk.client.appcontroller.AppController`

  `it.hourglass.myTalk.client.presenter.BaseViewPresenter`

  `it.hourglass.myTalk.client.presenter.CallPresenter`

  `it.hourglass.myTalk.client.presenter.ContactListPresenter`

  `it.hourglass.myTalk.client.presenter.LoginPresenter`

  `it.hourglass.myTalk.client.presenter.PasswordRecoverPresenter`

  `it.hourglass.myTalk.client.presenter.PopupMessagePresenter`

it.hourglass.myTalk.client.presenter.UserListPresenter

- La classe utilizza:

  it.hourglass.myTalk.client.communication.WSMessageBuilder

  it.hourglass.myTalk.client.communication.Call.CallStateListener

  it.hourglass.myTalk.client.communication.Call.Direction

  it.hourglass.myTalk.client.view.EndCallPopup

  it.hourglass.myTalk.client.view.PopupInformation

  it.hourglass.myTalk.client.wrappers.ConsoleLog

- La classe implementa

  Handler, necessaria per garantire la definizione dei metodi utilizzati per creare ed utilizzare un Web Socket.

Attributi

- String url
  Contiene l'URL del server websocket.
- Connection connection
  Variabile che rappresenta il Web Socket collegato al server grazie all'URL salvato nella variabile url.
- WSConnectionStateCallback callback
  Utilizzato per rappresentare i metodi utilizzati durante la connessione Web Socket dalla classe chiamante, ovvero `AppController`.
- CallMessageCallback callMessagesCallback
  Utilizzato per rappresentare i metodi utilizzati durante la connessione Web Socket dalla classe `Call`, istanziata in questa classe.
- String mynick
  Contiene il nick assegnato dopo aver effettuato la connessione al server.
- Call call
  Contiene un'istanza di `Call`.
- JSONObject incomingReq
  Contiene il JSON arrivato in ingresso per la richiesta di una nuova chiamata. Rimane salvato fintanto che l'utente non ha deciso di rifiutare la chiamata.
- boolean session
  Indica se l'utente è un utente autenticato o meno.
- HandlerManager eventBus
  Riferimento a una variabile che avrà la funzione di ricevitore di eventi.

Metodi

+ `WSConnection(String url, WSConnectionStateCallback cb, HandlerManager eb)`
  Costruttore della classe. Vengono assegnati alle variabili `url`, `callback` ed `eventbus` i rispettivi valori. Viene invocato il metodo `connect()` per creare l'oggetto Web Socket vero e proprio.

+ `void addCallMessageCallback()`
  Metodo utilizzato per assegnare alla variabile `callMessageCallback` la ridefinizione dei metodi da parte della classe `Call`.

+ `void removeCallMessageCallback()`
  Metodo utilizzato per eliminare il contenuto della variabile `callMessageCallback` dopo la chiusura di una chiamata.

+ `void connect()`
  Metodo utilizzato per la creazione di un nuovo Web Socket tramite la classe `WebSocketConnection`, per poi venir assegnato alla variabile `connection`. La classe utilizzata per la creazione del Web Socket, è un wrapper del Javascript per la creazione di quest'ultimo.

+ `synchronized void send(Object o)`
  Metodo utilizzato per spedire oggetti tramite il Web Socket. Nel nostro caso sono oggetti di tipo JSON.

+ `void onConnectionOpened(Connection connection)`
  Metodo che viene eseguito appena la comunicazione Web Socket inizia. Viene stampato in console che la comunicazione è iniziata e, se l'utente non è autenticato, viene avviata la registrazione anonima al server.

+ `void regOnServer()`
  Metodo utilizzato per richiedere la registrazione anonima al server. Se all'utente è già stato assegnato un nickname questo viene eliminato e comunicato al server di sconnettere tale nick, per poi farsi assegnare un nuovo nickname anonimo.

+ `void regOnServer(String s)`
  Funziona come il metodo `regOnServer()`, in questo caso però è specificato con quale nome l'utente vuole essere connesso al server.

+ `void onConnectionClosed(Connection connection)`
  Metodo che viene eseguito appena dopo che la comunicazione Web Socket è stata chiusa. Viene stampato per prima cosa in console che la comunicazione è finita e viene invocato il metodo `onConnectionClosed()` definito dalla classe chiamante `AppController`.

+ `void onMessageReceived(Connection connection, Object message)`
  Metodo che viene eseguito quando viene ricevuto un messaggio dal server WSServer. Per prima cosa avviene un parsing del messaggio e viene stampato a console come prova dell'avvenuta comunicazione. Solo se viene individuato che il JSON ha una struttura conosciuta allora si procede all'estrazione del contenuto: Il tipo viene salvato in una variabile type, il contenuto (che è un altro JSON) viene

salvato in jso. Viene successivamente invocato il metodo `handleMessage(String type, JSONObject jso)` per le successive elaborazioni.

`+ void handleMessage(String type, JSONObject jso)`
Questo metodo viene utilizzato per eseguire la corretta elaborazione del messaggio in base al tipo ricevuto. Vi sono vari tipi di messaggio che questo metodo riesce ad elaborare:

- registration: Viene estratto il nick ricevuto dal server e conservato nella variabile `mynick` per poi mandare i dati ricevuti alla classe `AppController` per successive elaborazioni.
- info: Viene scritta in console un'informazione generica per l'utente.
- endcall: Viene chiusa la chiamata attraverso la chiamata del metodo `endcall()`.
- incoming : Qui vengono gestite una parte dei messaggi utilizzati per l'inizio di una chiamata. Viene gestito il caso di richiesta, ovvero viene controllato che l'utente non sia già in comunicazione con un'altro utente e nel caso questo avvenga viene automaticamente rifiutata la chiamata altrimenti viene smistata alla classe `AppController`. L'altro caso che viene gestito è il rifiuto della chiamata. L'accettazione dell'invito è demandata alla classe `AppController`.
- notification : Non viene gestito da questa classe ma demandato all'AppController.

`+ void doCall(Call.Direction direction,String callnick, Video localVideo, Video remoteVideo, TextArea textachat,int s, TextBox calltLenght, TextBox callByteUp, TextBox callByteDown)`
Metodo utilizzato per la creazione di una nuova chiamata, ovvero un'istanza della classe `Call` che andrà salvata nella variabile `call`. In ingresso vengono inviati:

- `direction`: Indica in quale direzione la richiesta di chiamata sta arrivando, o esternamente o internamente.
- `callnick`: Indica il nickname del chiamante o del chiamato.
- `localVideo`: E' un riferimento ad un oggetto rappresentante il video locale, ovvero dove andrà proiettato lo stream catturato dalla nostra periferica.
- `remoteVideo`: E' un riferimento ad un oggetto rappresentante il video remoto, ovvero dove andrà proiettato lo stream catturato dalla periferica del nostro interlocutore.
- `textareachat`: E' un riferimento all'area di testo della chat.
- `s`: Un numero che rappresenta il tipo di chiamata e di conseguenza i vari stream da utilizzare.
- `calltLenght`: E' un riferimento all'area di testo dove andrà stampata la durata della conversazione.
- `callByteUp`: E' un riferimento all'area di testo dove andrà stampata la quantità di byte inviati.

- `callByteDown`: E' un riferimento all'area di testo dove andrà stampata la quantità di byte scaricati.

`+ void doCall(Video localStream,Video remoteVideo, TextArea textachat, TextBox calltLenght, TextBox callByte, TextBox latency, Label nickRec)`
Metodo utilizzato per la creazione di chiamate in entrata, utilizza il metodo `doCall` definito sopra e modifica il testo della view predisposta alla visione della chiamata inserendo il nome dell'interlocutore.

`+ void closeCall(Call.Direction d)`
Metodo utilizzato per la chiusura di una chiamata. Viene utilizzato il metodo `Call.hangup(d)` con d la direzione della richiesta di fine chiamata e viene poi creata un'istanza della classe `it.hourglass.myTalk.client.view.EndCallPopup` con la durata della conversazione.

2.3.2 client.communication.Call - Classe

Funzione

Questa classe rappresenta la chiamata vera e propria. Viene istanziata da WSConnection quando vi è la richiesta di una nuova chiamata.

Relazione d'uso con altri moduli

- La classe è utilizzata da:

  `it.hourglass.myTalk.client.communication.WSConnection`

  `it.hourglass.myTalk.client.presenter.CallPresenter`

- La classe utilizza:

  `it.hourglass.myTalk.client.communication.WSMessageBuilder`

  Tutte le classi contenute nel package `it.hourglass.myTalk.client.wrappers`

- La classe implementa

  `PeerConnectionCallbacks` Necessaria per garantire la definizione dei metodi utilizzati per gestire i vari eventi tra gli utenti.

  `CallMessageCallback` Necessaria per garantire la definizione dei metodi utilizzati per gestire i vari eventi di quando arriva un messaggio via WebSocket.

Attributi

- `PeerConnectionWrapper pc`
  Rappresenta un oggetto PeerConnection.
- `Video remoteVideoElement`
  Variabile che rappresenta dove lo stream dell'utente remoto andrà proiettato .
- `Video localVideoElement`
  Variabile che rappresenta dove lo stream dell'utente locale andrà proiettato .
- `WSConnection wsConnection`
  E' un riferimento alla classe chiamante `WSConnection`.
- `String callId`
  Contiene il nick dell'utente interlocutore.
- `Direction direction`
  Indica se la chiamata è stata inviata o ricevuta, facendo comportare la classe in modo diverso.
- `String mynick`
  Contiene il nick dell'utente interlocutore.
- `MediaStream localStream`
  Utilizzata per creare un URI per potersi collegare allo stream.

- `boolean audio,video,chat`
  Rappresentano i flussi attivi nella chiamata.
- `long callTime`
  Rappresenta la durata della chiamata.
- `TextBox callByteU`
  Rappresenta il riferimento alla textbox che rappresenta i byte in upload.
- `TextBox callByteD`
  Rappresenta il riferimento alla textbox che rappresenta i byte in download.
- `TextBoxcalltLenght`
  Rappresenta il riferimento alla textbox che rappresenta il tempo trascorso.
- `boolean open`
  Booleano che assume il valore false quando la chiamata in corso viene riattaccata.
- `SDPCreateOfferCallback impl`
  Contiene l'implementazione dei metodi utilizzati per generare la risposta quando arriva l'sdp dell'altro interlocutore.

Metodi

+ `Call(Direction direction,WSConnection wsConnection,Video localVideoElement, Video remoteVideoElement,CallStateListener listener,String callId, TextArea textachat, int n, TextBox calltLenght, TextBox callByteUp, TextBox callByteDw)`
  Unico costruttore di Call. Per prima cosa vengono inizializzate tutte le variabili necessarie. Si passa poi alla configurazione della PeerConnection, indicando il server STUN a cui richiedere le informazioni per la propria raggiungibilità, per poi creare un fine un'istanza di PeerConnection. Come ultima cosa, in base alla direzione della comunicazione, viene: o fatta partire la chiamata passando il JSON di incoming arrivato precedentemente oppure viene richiesto l'accesso alle proprio risorse audio/video.
+ `void setStream(int s)`
  In base al numero s, che può andare da 1 a 4, vi sono quattro combinazioni di stream:

  1 = audio

  2 = audio + video

  3 = audio + video + chat

  4 = chat
- `int getStream()`
  Ritorna un valore da 1 a 4 in base allo stream scelto.
+ `void onicecandidate(JavaScriptObject jso)`
  Metodo utilizzato per la creazione del messaggio ICECandidate e per l'invio di quest'ultimo al server. Fa l'Override del metodo `icecandidate()` della classe `PeerConnectionCallback`.

+ `void onaddstream(JavaScriptObject jso)`
  Metodo utilizzato per l'aggiunta di uno stream quando la conversazione è stata accettata da entrambe le parti. Viene catturato lo stream e viene generato un URL per connetterlo all'elemento video.

+ `void onremovestream(JavaScriptObject jso)`
  Metodo utilizzato quando lo stream viene rimosso. Viene mandato un messaggio all'interlocutore per avvisarlo che la chiamata è finita.

+ `void onchannelopen(JavaScriptObject jso)`
  Metodo utilizzato quando il datachannel passa allo stato open. Vengono effettuate due stampe nella console JavaScript del browser che riportano questo avvenimento.

+ `void onchannelmessage(JavaScriptObject jso)`
  Metodo utilizzato quando vi è un messaggio della chat in arrivo. Viene riportato l'evento nella console Javascript del browser e viene stampato nella TextArea riservata alla chat.

+ `void onchannelclose(JavaScriptObject jso)`
  Metodo utilizzato quando il datachannel viene chiuso. Viene riportato l'evento nella console Javascript del browser.

+ `void sendMessage(String str)`
  Metodo utilizzato per inviare messaggi nello stream della chat. Utilizza il metodo `sendMessage()` attraverso l'oggetto `pc` della classe PeerConnectionWrapper.

+ `void onMessageReceived(String type , JSONObject jso)`
  Metodo utilizzato per smistare i messaggi in arrivo che non vengono gestiti dalla classe `WSConnection`. Vengono gestiti i messaggi di offerta (offer), risposta (answer), ICECandidate (candidate) e di ricezione chiamata (incoming).

  - `void handle_offer(JSONObject jso)`
    Metodo che viene invocato per elaborare il messaggio di offerta arrivato dall'utente. Viene estratto l'sdp, impostata la remote description della Peer Connection tramite il metodo `setRemoteDescription()` e viene creata ed inviata una risposta all'utente.

  - `void handle_answer(JSONObject jso)`
    Metodo che viene invocato per elaborare il messaggio di risposta arrivato dall'utente. Viene estratto l'sdp ed impostata la remote description della Peer Connection tramite il metodo `setRemoteDescription()`.

  - `void handle_candidate(JSONObject jso)`
    Metodo che riceve un messaggio ICECandidate e lo aggiunge all'oggetto PeerConnection.

  - `void handle_incoming(JSONObject jso)`
    Metodo che riceve il messaggio di richiesta o di conferma di accettazione della chiamata. Per prima cosa viene avviato un timer per il calcolo del tempo trascorso, poi si controlla il tipo del messaggio: se è una richiesta di chiamata

viene controllato che lo stream sia diverso dal tipo 4 e viene avviato il metodo `getUserMedia()` altrimenti viene utilizzato il metodo `connecting()`. Se si tratta di una conferma, viene avviato il metodo `handle_b_party_confirmation()`.

`+ String hangup(Direction d)`

Metodo che chiude la chiamata in corso. Per prima cosa viene controllato se la richiesta arriva da fuori o da dentro. Nel secondo caso invia un messaggio all'interlocutore per avvisarlo della fine della chiamata. Viene poi chiuso il Peer Connection, impostato a null il remote video e ritornata una String che rappresenta i minuti ed i secondi trascorsi in chiamata.

`- void handle_b_party_confirmation()`

Metodo che viene invocato quando la controparte accetta la chiamata. Viene per prima cosa aggiunto lo stream locale all'oggetto Peer Connection, successivamente viene creata l'offerta da mandare all'interlocutore e viene avviato il timer.

`+ void getUserMedia()`

Metodo invocato quando si vuole ottenere lo stream audio/video locale. Viene richiesto all'utente il permesso di poter accedere allo stream tramite il browser. Ogni browser ha un suo modo per richiedere l'autorizzazione. Chrome fa apparire una barra di dialogo nella parte superiore della pagina, Firefox fa apparire un popup a fianco la barra degli indirizzi. Se tale accesso è autorizzato viene per prima cosa collegato lo stream locale con l'elemento video preposto a riceverlo e successivamente viene aggiunto un riferimento della chiamata al MessageCallback di WSConnection. Se la chiamata è in uscita verrà inviato un messaggio di incoming al richiedente, mentre nel caso fosse in entrata verrà inviato un messaggio di conferma all'offerente. Nel caso in cui non sia possibile accedere allo stream e la chiamata sia in uscita verrà stampato nella console del browser il motivo del fallimento, mentre se la chiamata è in entrata verrà mandato un messaggio di rifiuto della chiamata all'offerente.

`- void connecting()`

Metodo utilizzato se si avvia una chiamata testuale, ovvero solo con lo stream testuale. Inizializza la chiamata.

2.3.3 client.communication.WSMessageBuilder - Classe

Funzione

Questa classe viene utilizzata per la creazione e composizione dei JSON da inviare alla servlet per lo scambio di informazioni e per la richiesta di servizi.

Relazione d'uso con altri moduli

- La classe è utilizzata da:

  `it.hourglass.myTalk.client.communication.WSConnection`

  `it.hourglass.myTalk.client.communication.Call`

  `it.hourglass.myTalk.client.appcontroller.AppController`

  `it.hourglass.myTalk.client.shared.Profile`

- La classe utilizza:

  `it.hourglass.myTalk.client.wrappers.ConsoleLog`

Attributi

La classe non ha attributi.

Metodi

`+ static String registrationMsg(String nickName)`
Crea un JSON con la richiesta di registrazione al server con il nome contenuto in `nickName`. Se si vuole la registrazione anonima il valore è vuoto. Viene utilizzato come prima comunicazione dopo aver aperto la comunicazione con i Web Socket.

`+ static String callMsg(String callee, String mynick,int s)`
Si occupa della creazione del JSON di inizio chiamata. Questo viene solitamente creato all'inizio della comunicazione tra due utenti. Con `s` viene indicato quale tipo di comunicazione si vuole avere.

`+ static String confirmCallMsg(String callee, String mynick)`
Si occupa della creazione del JSON di conferma della chiamata. Questo tipo di messaggio viene creato quando l'utente ricevente acconsente all'inizio della chiamata.

`+ static String declineCallMsg(String callee, String mynick)`
Si occupa della creazione del JSON di negazione della chiamata. Questo tipo di messaggio viene creato quando l'utente ricevente rifiuta l'inizio della chiamata.

`+ static String endCallMsg(String callid,String info)`
Si occupa della creazione del JSON che indica la fine della chiamata. Questo tipo di messaggio viene creato quando la chiamata è stata chiusa in modo atteso.

`+ static String relayMsg(String callee, String mynick, JSONObject toRelay )`
Si occupa della creazione del JSON contenete le informazioni utili alle librerie

webRTC. Possono essere scambiati tre tipi di messaggi aventi la medesima struttura, ovvero l'offerta dell'sdp da parte di chi inizia la chiamata (offer), la risposta di quest'ultimo con il suo sdp (answer) e lo scambio delle informazioni sullo stream (candidate).

+ `static String byeMsg`
  Si occupa della creazione del JSON che indica la disconnessione dalla servlet WSServer. Questo tipo di messaggio viene creato quando il client sta per chiudere la connessione Web Socket .

+ `static String contactList(String mynick, String[] l)`
  Metodo utilizzato quando si vuole inviare la propria lista contatti alla servlet WSServer. Questo tipo di messaggio viene creato subito dopo la registrazione non anonima per associare la lista di contatti con il proprio nickname .

+ `static String sendNotificationFriendRequest(String callee,String mynick)`
  Si occupa della creazione del JSON che notifica a `callee` la presenza di una nuova richiesta di amicizia inviata da `mynick`. Questo tipo di messaggio viene creato quando `mynick` chiede l'amicizia a `callee`.

+ `static String sendNotificationAcceptFriendRequest(String callee, String mynick)`
  Si occupa della creazione del JSON che notifica a `callee` l'accettazione della richiesta di amicizia da parte di `mynick`. Questo tipo di messaggio viene creato quando `mynick` accetta l'amicizia di `callee`.

+ `static String sendNotificationMsgSend(String callee, String mynick)`
  Si occupa della creazione del JSON che notifica a `callee` la presenza di un nuovo messaggio inviato da `mynick`. Questo tipo di messaggio viene creato quando `mynick` manda un messaggio a `callee`.

+ `static String sendNotificationDeleteFriend(String callee, String mynick)`
  Si occupa della creazione del JSON che notifica a `callee` l'eliminazione di `mynick` dalla sua lista contatti. Questo tipo di messaggio viene creato quando `mynick` elimina `callee` dalla sua lista contatti .

- `static String sendNotificationMsg(String callee, String mynick, int i )`
  E' la classe che crea tutti e quattro i tipi di notifica elencati sopra.

+ `static String Ping()`
  E' la classe che crea il messaggio di ping al server per mantenere aperta la connessione con il Web Socket.

2.4 Package it.hourglass.myTalk.client.appcontroller

Questo package contiene la definizione della classe `AppController`.

2.4.1 client.appcontroller.AppController - Classe

Funzione

Questa classe si occupa dell'inizializzazione del programma MyTalk e del suo controllo. Il programma utilizza il sistema `History` fornito dal framework GWT per il riconoscimento delle richieste fatte dall'utente. Tali richieste sono intercettate dal sistema `HandlerManager` anch'esso fornito dal framework GWT. Per ogni evento sarà definito il comportamento che il programma dovrà avere.
L'appcontroller quando riceve un evento (lanciato dalle classi contenute nel package `presenter` si occupa di eseguire la logica in risposta ad esso. Se l'evento è la richiesta di *switch view* lo richiede alla classe `presenter.baseViewPresenter`. La classe si occupa di inizializzare la connessione con il server WebSocket necessaria per il corretto funzionamento del programma.

Relazione d'uso con altri moduli

- La classe è utilizzata da:

  `it.houglass.myTalk.client.MyTalk`

- La classe utilizza:

  `it.hourglass.myTalk.client.view.PopupInformation`

  `it.hourglass.myTalk.client.RPCService.RPCService`

  `it.hourglass.myTalk.client.RPCService.RPCServiceAsync`

  Tutte le classi contenute nel package `it.hourglass.myTalk.client.event`

  `it.hourglass.myTalk.client.presenter.BaseViewPresenter`

  `it.hourglass.myTalk.client.presenter.CallPopupPresenter`

  `it.hourglass.myTalk.client.profile.Profile`

  `it.hourglass.myTalk.client.shared.Message`

  `it.hourglass.myTalk.client.shared.SignIn`

  `it.hourglass.myTalk.client.shared.User`

  `it.hourglass.myTalk.client.view.CallPopup`

  `it.hourglass.myTalk.client.wrappers.ConsoleLog`

- La classe implementa

  `ValueChangeHandler<String>` Necessaria per gestire gli eventi tramite l'HandlerManager.

Attributi

- `final HandlerManager eventBus`
  Riferimento a una variabile che avrà la funzione di ricevitore di eventi.
- `final WSConnection wsconnection`
  Variabile che serve per connettersi al WebSocket.
- `BaseViewPresenter view`
  Riferimento alla view di base.
- `Profile profile`
  Variabile che si riferisce all'oggetto che mantiene i dati relativi all'utente principale garantendone coerenza e accessibilità in tutte le componenti dell'applicativo.

Metodi

+ `AppController(final HandlerManager eventBus)`
  Costruttore della classe. Assegna `eventBus` e inserisce in esso gli eventi grazie al metodo `bind()`). Viene successivamente creata la connessione con il server WebSocket tramite l'instanziazione di un oggetto di tipo `client.communication.WSConnection`. A quest'oggetto viene ridefinito il metodo `onConnectionOpened()` che riceve la risposta all'apertura della connessione con il server, il metodo `onConnectionClosed()` che riceve la risposta alla chiusura della connessione con il server e il metodo `onCallUnrelatedMessage()` che si occupa di gestire l'arrivo di un messaggio JSON dal server WebSocket.
- `void bind()`
  Metodo che aggiunge ad `eventBus` gli eventi e indica al sistema cosa fare al verificarsi degli stessi. Ad ogni evento aggiunto all'`eventBus` andrà ridefinito il metodo contenuto nella classe Handler dell'evento.
+ `void onValueChange(ValueChangeEvent<String> event)`
  Metodo invocato automaticamente dopo il cambiamento del History token. In base all'operazione specificata sul token si decide cosa fare modificando così il sistema.
- `static RPCServiceAsync getService()`
  Metodo che ritorna una variabile di tipo RPCServiceAsync per la chiamata di procedura remota.
+ `void init()`
  Metodo per l'inizializzazione della view e per la creazione del primo History token.
- `void doCallPageRequest()`
  Questo metodo si occupa di richiedere la pagina di Chiamata. Questo avverrà tramite la creazione di un nuovo history token call.
- `void doCallRequest()`
  Questo metodo si occupa di richiedere la pagina di Chiamata in seguito ad una richiesta di chiamata accettata. Questo avverrà tramite la creazione di un nuovo history token call_request.

- `void doChangeFooter()`
  Questo metodo si occupa di richiedere il refresh del footer alla baseView in seguito al cambio dell'identificativo corrente dell'utente.
- `void doContactManagementPageRequest()`
  Questo metodo si occupa di richiedere la pagina di gestione delle amicizie. Questo avverrà tramite la creazione di un nuovo history token contact_managment.
- `void doFriendContactProfilePageRequest()`
  Questo metodo si occupa di richiedere la pagina di visione del profilo di un altro utente. Questo avverrà tramite la creazione di un nuovo history token friend_profile.
- `void doHomePageRequest()`
  Questo metodo si occupa di richiedere la HomePage. Questo avverrà tramite la creazione di un nuovo history token home.
- `void doLoginPageRequest()`
  Questo metodo si occupa di richiedere la pagina di Login. Questo avverrà tramite la creazione di un nuovo history token login.
- `void doLoginRequest()`
  Questo metodo si occupa di richiedere il recupero del Profilo di un utente in seguito all'autenticazione di un utente.
- `void doLogoutRequest()`
  Questo metodo si occupa di gestire la de-autenticazione di un Utente. Si occupa di cancellare il profilo locale e di far tornare il sistema allo stato iniziale di utente non autenticato.
- `void doMessagePageRequest()`
  Questo metodo si occupa di richiedere la pagina di visualizzazione dei messaggi. Questo avverrà tramite la creazione di un nuovo history token message.
- `void doRecuperaPasswordRequest()`
  Questo metodo si occupa di richiedere la pagina di Recupero Password. Questo avverrà tramite la creazione di un nuovo history token password_recover.
- `void doRegisterPageRequest()`
  Questo metodo si occupa di richiedere la pagina di Registrazione. Questo avverrà tramite la creazione di un nuovo history token registration.
- `void refuseCall()`
  Questo metodo si occupa gestire il rifiuto di una chiamata. Questo avverrà richiedendo il metodo `refuseCall()` definito nella classe `client.communication.WSConnection`.
- `void signIn(String uniqueId)`
  Metodo invocato dopo che l'utente ha effettuato il login. Questo metodo si occupa di recuperare i dati del profilo dal database. In esso si definisce l'AsyncCallback<SignIn> necessaria all'invocazione `signIn()` della RPC definita nella

classe `client.rpcservice.RPCService`. Se la RPC ha successo si recupera il profilo il quale viene salvato in `client.shared.Profile`. A questo punto si richiederà lo switch view del sistema in seguito all'autenticazione tramite il metodo `switchLoggedIn()` definito nella classe `client.presenter.BaseViewPresenter` e lanciando l'evento `client.event.IdChangedEvent`. Infine si occuperà di controllare se ci sono richieste di amicizia a cui l'utente non ha ancora risposto mediante il metodo `friendRequestControl()`.

- `void doProfileManagementPageRequest()`
  Questo metodo si occupa di richiedere la pagina di gestione del profilo. Questo avverrà tramite la creazione di un nuovo history token profile_managment.
- `void doRemoveContact(String contact)`
  Questo metodo si occupa di gestire la rimozione dalla lista contatti di un utente.
- `void doNewFriendContactList(String contact,Boolean status)`
  Questo metodo si occupa di gestire l'aggiunta di un utente nella lista contatti.
- `static RPCServiceAsync getService()`
  Questo metodo si occupa di restituire il servizio necessario alla richiesta di una RPC.
- `void init()`
  Questo metodo si occupa di inizializzare la view del sistema e di richiedere il primo History token: home.
- `void friendRequestControl()`
  Questo metodo, invocato in seguito al recupero del profilo controlla se l'utente ha delle richieste di amiciza pendenti. Se ciò viene riscontrato il metodo si occupa di segnalarlo all'utente tramite un popup (utilizzando il popup `it.hourglass.myTalk.clien.view.PopupInformation`).

2.5 Package it.hourglass.myTalk.client.event

Questo package contiene la definizione di tutti gli eventi presenti nel sistema. Tali eventi verranno ricevuti da `eventBus` contenuto nella classe `client.appcontroller.AppController` (tale classe contiene anche la logica di gestione di ogni evento). Tutti gli eventi hanno una struttura simile: essi sono composti da un interfaccia Handler e una classe. L'interfaccia definisce la il metodo di risposta all'evento (il quale sarà poi ridefinito in `client.appcontroller.AppController` mediante il metodo `bind()`). La classe invece definisce il TYPE dell'evento il quale lo contraddistingue dagli altri eventi.

NOTA Per brevità si fornisce una descrizione unica di tutti i metodi comuni ad i vari eventi.
In dettaglio si possono verificare i seguenti eventi:

- CallPageRequest
- CallRequest
- CallRequestRefused
- ContactManagementPageRequest
- FriendContactProfilePageRequest
- HomePageRequest
- IdChanged
- LoginPageRequest
- LoginRequest
- LogoutRequest
- MessagePageRequest
- NewFriendContactList
- PasswordRecoverPageRequest
- ProfileManagementPageRequest
- RegistrationPageRequest
- RemoveFriendContact

2.5.1 client.event.***Event - Classe

Funzione

Per ogni evento viene definita una classe. Questa classe servirà per identificare l'evento una volta che esso viene lanciato. L'identificazione avviene grazie all'attributo TYPE il quale si occupa di marcare l'evento. TYPE è un tipo parametrico al quale viene associato come parametro l'interfaccia ***Handler corrispettiva all'evento. Ogni tipo di evento verrà poi registrato da `eventBus` che è un attributo della classe `client.appcontroller.AppController`. Per la lista degli eventi si faccia riferimento alla sezione metodi delle interfacce `***EventHandler`.

Relazione d'uso con altri moduli

- La classe estende:

  `com.google.gwt.event.shared.GwtEvent`
  Questa estensione è necessaria per far si che la classe venga percepita come evento da HandlerManager.

Attributi

`+ static Type<***EventHandler> TYPE`
Questo attributo definisce il Type dell'evento.

Metodi

`+ Type<AddContactEventHandler> getAssociatedType()`
Questo metodo restituisce il TYPE dell'evento

`# void dispatch(AddContactEventHandler handler)`
Questo metodo viene invocato automaticamente quando l'evento viene ricevuto. Per ogni evento si invocherà il metodo esposto dall'interfaccia ***Handler.

2.5.2 client.event.***EventHandler - Interfaccia

Funzione

Per ogni evento viene definita un'interfaccia Handler. Tale interfaccia espone il metodo che verrà invocato in seguito alla ricezione dello stesso e funge da marcatore per il TYPE di ogni evento.

Relazione d'uso con altri moduli

- L'interfaccia estende :

  `com.google.gwt.event.shared.EventHandler`
  Questa estensione è necessaria per marcare il fatto che l'interfaccia è un Handler di un evento.

Attributi

Nessuno.

Metodi

Ogni evento esporrà un evento diverso. Ogni metodo verrà ridefinito nella classe `client.appcontroller.AppController` all'interno del metodo `bind()`.

- L'evento `CallPageRequestEvent` espone il metodo :
  `void onCallPageRequest(CallPageRequestEvent event)`
  Questo metodo verrà utilizzato per richiedere la pagina di chiamata.
- L'evento `CallRequestEvent` espone il metodo :
  `void onCallRequest(CallRequestEvent callRequestEvent)`
  Questo metodo verrà utilizzato per avvertire l'utente di una chiamata in arrivo.
- L'evento `CallRequestRefusedEvent` espone il metodo :
  `void onCallRequestRefused(CallRequestRefusedEvent callRequestEvent)`
  Questo metodo verrà utilizzato per informare l'utente che la chiamata è stata rifiutata.
- L'evento `ContactManagementPageRequest` espone il metodo :
  `void onContactManagementPageRequest(ContactManagementPageRequestEvent contactManagementPageRequestEvent)`
  Questo metodo verrà utilizzato per richiedere la pagina di gestione delle amicizie.
- L'evento `FriendContactProfilePageRequest` espone il metodo :
  `void onFriendContactProfilePageRequest(FriendContactProfilePageRequest friendContactProfilePageRequest)`
  Questo metodo verrà utilizzato per richiedere la pagina di visualizzazione del profilo di un amico.
- L'evento `HomePageRequestEvent` espone il metodo :
  `void onHomePageRequestEvent(HomePageRequestEvent Request)`
  Questo metodo verrà utilizzato per richiedere l'homepage.
- L'evento `IdChangedEvent` espone il metodo :
  `void onIdChangedEvent(IdChangedEvent Request)`
  Questo metodo verrà utilizzato per avvertire il sistema che l'identificativo di un utente è cambiato.
- L'evento `LoginPageRequestEvent` espone il metodo :
  `void onLoginPageRequest(LoginPageRequestEvent Request)`
  Questo metodo verrà utilizzato per richiedere la pagina di login.
- L'evento `LoginRequestEvent` espone il metodo :
  `void onLoginRequest(LoginRequestEvent event)`
  Questo metodo verrà utilizzato per avvertire il programma che l'utente si è autenticato.
- L'evento `LogoutRequestEvent` espone il metodo :
  `void onLogoutRequest(LogoutRequestEvent event)`

Questo metodo verrà utilizzato per avvertire il programma che un utente ha richiesto di de-autenticarsi.

- L'evento `MessagePageRequestEvent` espone il metodo :
  `void onMessagePageRequestEvent(MessagePageRequestEvent event)`
  Questo metodo verrà utilizzato per richiedere la pagina di visione dei messaggi.
- L'evento `NewFriendContactListEvent` espone il metodo :
  `void onNewFriendContactListEvent(NewFriendContactListEvent event,String contact,Boolean status)`
  Questo metodo verrà utilizzato per avvertire il programma che un utente ha accettato la richiesta di amicizia. Si aggiungerà dunque un contatto `contact` con stato `status` alla lista utenti.
- L'evento `PasswordRecoverPageRequestEvent` espone il metodo :
  `onPasswordRecoverPageRequest(PasswordRecoverPageRequestEvent event)`
  Questo metodo verrà utilizzato per richiedere la pagina per il recupero della password.
- L'evento `ProfileManagementPageRequestEvent` espone il metodo :
  `onProfileManagementPageRequest(ProfileManagementPageRequestEvent event)`
  Questo metodo verrà utilizzato per richiedere la pagina di gestione del profilo.
- L'evento `RegistrationPageRequestEvent` espone il metodo :
  `void onRegistrationPageRequestEvent(RegistrationPageRequestEvent event)`
  Questo metodo verrà utilizzato per richiedere la pagina di registrazione.
- L'evento `RemoveFriendContactEvent` espone il metodo :
  `onRemoveFriendContactEvent(this,contact)`
  Questo metodo verrà utilizzato per rimuovere l'utente `contact` dalla lista amici.

Prima di presentare le classi e le interfacce del package `presenter` verranno presentate le interfacce `Display` contenute nel sottopackage `presenter.display`.

2.6 Package it.hourglass.myTalk.client.presenter.display

Questo package contiene la definizione di tutte le interfacce `Display` le quali vengono implementate dalle classi del package `view`. Tali interfacce espongono i metodi che le classi `Presenter` possono richiedere alla view che implementa tale interfaccia. *Nota: Per rendere il documento più leggibile ed evitare ridondanza la descrizione dei metodi delle interfacce verranno descritti dalla classe che implementa l'interfaccia*

2.6.1 client.presenter.display.BaseDisplay - Interfaccia

Questa interfaccia viene implementata dalla classe `view.BaseView`.
L'interfaccia espone i seguenti metodi:

```
+ void switchBody(AbsolutePanel pannello)
```

```
+ void switchFooter(AbsolutePanel pannello)
```

```
+ void switchMenu(AbsolutePanel pannello)
```

```
+ void addContactList(AbsolutePanel panel)
```

2.6.2 client.presenter.display.AnnoMenuDisplay - Interfaccia

Questa interfaccia viene implementata dalla classe `view.AnnoMenuView`.
L'interfaccia espone i seguenti metodi:

```
+ HasClickHandlers getHomeButton()
```

```
+ HasClickHandlers getCallButton()
```

```
+ HasClickHandlers getRegistrationButton()
```

```
+ HasClickHandlers getLoginButton()
```

```
+ HasClickHandlers getRecuperaPasswordButton()
```

```
+ AbsolutePanel getView()
```

2.6.3 client.presenter.display.SignedMenuDisplay - Interfaccia

Questa interfaccia viene implementata dalla classe `view.SignedMenuDisplay`.
L'interfaccia espone i seguenti metodi:

```
+ HasClickHandlers getHomeButton()
```

```
+ HasClickHandlers getCallButton()
```

+ HasClickHandlers getProfileManagementButton()

+ HasClickHandlers getLogoutButton()

+ AbsolutePanel getView()

2.6.4 client.presenter.display.CallDisplay - Interfaccia

Questa interfaccia viene implementata dalla classe `view.CallView`.
L'interfaccia espone i seguenti metodi:

- Label getCalee()

- HasClickHandlers getSendButton()

- HasClickHandlers getCallButton()

- HasClickHandlers getCloseButton()

- Video getLocalVideo()

- Video getGuestVideo()

- TextArea getChat()

- TextBox getByteD()

- TextBox getByteU()

- TextBox getLenght()

- TextBox getcallUserTextBox()

- TextBox getTextBoxInvia()

- String getStreamRequest()

- AbsolutePanel getView()

- void nascondi()

- void show()

2.6.5 client.presenter.display.CallPopupDisplay - Interfaccia

Questa interfaccia viene implementata dalla classe `view.CallPopupView`.
L'interfaccia espone i seguenti metodi:

+ void close()

+ HasClickHandlers getAcceptButton()

+ HasClickHandlers getRefuseButton()

+ String getName()

2.6.6 client.presenter.display.ContactListDisplay - Interfaccia

Questa interfaccia viene implementata dalla classe `view.ContactListView`.
L'interfaccia espone i seguenti metodi:

```
public void addUser(AbsolutePanel userView)

public void removeContactList()

public HasClickHandlers getContactManagmentPage()

public AbsolutePanel getView()
```

2.6.7 client.presenter.display.FooterDisplay - Interfaccia

Questa interfaccia viene implementata dalla classe `view.FooterView`.
L'interfaccia espone i seguenti metodi:

```
+ String getUsername()

+ void setUsername(String username)

+ void refresh()

+ public AbsolutePanel getView()
```

2.6.8 client.presenter.display.UserListDisplay - Interfaccia

Questa interfaccia viene implementata dalla classe `view.UserListView`.
L'interfaccia espone i seguenti metodi:

```
+ HasClickHandlers getCallButton()

+ HasClickHandlers getMessageButton()

+ HasClickHandlers getProfileButton()

+ void changeStatus(boolean status)

+ AbsolutePanel getView()

+ void remove()
```

2.6.9 client.presenter.display.ContactManagementDisplay - Interfaccia

Questa interfaccia viene implementata dalla classe `view.ContactManagementView`.
L'interfaccia espone i seguenti metodi:

```
+ HasValue<String> getUserRequest()

+ HasClickHandlers getFriendshipRequestButton()
```

+ AbsolutePanel getView()

+ void addFriendMessage(AbsolutePanel panel)

+ void addFriend(AbsolutePanel panel)

2.6.10 client.presenter.display.FriendMessageDisplay - Interfaccia

Questa interfaccia viene implementata dalla classe `view.FriendMessageView`.
L'interfaccia espone i seguenti metodi:

+ HasClickHandlers getAcceptButton()

+ HasClickHandlers getRefuseButton()

+ AbsolutePanel getView()

+ void selfDelete()

2.6.11 client.presenter.display.EditProfileDisplay - Interfaccia

Questa interfaccia viene implementata dalla classe `view.EditProfileView`.
L'interfaccia espone i seguenti metodi:

+ void addAvatarView(AbsolutePanel panel)

+ void addPersonalDataView(AbsolutePanel panel)

+ void addExtendedDataView(AbsolutePanel panel)

+ void addFooter()

+ HasClickHandlers getMessageButton()

+ HasClickHandlers getDeleteAccountButton()

+ AbsolutePanel getView()

2.6.12 client.presenter.display.FriendContactDisplay - Interfaccia

Questa interfaccia viene implementata dalla classe `view.FriendContactView`.
L'interfaccia espone i seguenti metodi:

+ HasClickHandlers getRemoveButton()

+ AbsolutePanel getView()

+ void selfDelete()

2.6.13 client.presenter.display.ContactProfileDisplay - Interfaccia

Questa interfaccia viene implementata dalla classe `view.ContactProfileView`.
L'interfaccia espone i seguenti metodi:

```
+ void init()
+ AbsolutePanel getView()
```

2.6.14 client.presenter.display.LoginDisplay - Interfaccia

Questa interfaccia viene implementata dalla classe `view.LoginView`.
L'interfaccia espone i seguenti metodi:

```
+ HasClickHandlers getLoginButton()
+ HasValue<String> getUsername()
+ HasValue<String> getPassword()
+ AbsolutePanel getView()
+ void showLoginError(String error)
```

2.6.15 client.presenter.display.MessageDisplay - Interfaccia

Questa interfaccia viene implementata dalla classe `view.MessageView`.
L'interfaccia espone i seguenti metodi:

```
public void addMessage(AbsolutePanel panel)
public AbsolutePanel getView()
```

2.6.16 client.presenter.display.TextMessageDisplay - Interfaccia

Questa interfaccia viene implementata dalla classe `view.TextMessageView`.
L'interfaccia espone i seguenti metodi:

```
+ void addUser(AbsolutePanel userView)
+ void removeContactList()
HasClickHandlers getContactManagmentPage()
+ AbsolutePanel getView()
```

2.6.17 client.presenter.display.PasswordRecoverDisplay - Interfaccia

Questa interfaccia viene implementata dalla classe `view.PasswordRecoverView`.
L'interfaccia espone i seguenti metodi:

```
+ HasClickHandlers getRequestButton()
+ HasClickHandlers getContineuButton()
+ HasClickHandlers getConfirmButton()
+ void switchFase2()
+ void switchFase3()
+ void showErrorFase1(String error)
+ void showErrorFase2(String error)
+ void showErrorFase3(String error)
+ TextBox getConfirmCode()
+ TextBox getEmail()
+ PasswordTextBox getPassword()
+ PasswordTextBox getConfirmPassword()
+ AbsolutePanel getView()
+ void showLoginError(String error)
```

2.6.18 client.presenter.display.PopupMessageDisplay - Interfaccia

Questa interfaccia viene implementata dalla classe `view.`.
L'interfaccia espone i seguenti metodi:

```
+ HasClickHandlers getSendButton()
+ HasClickHandlers getCancellButton()
+ HasValue<String> getText()
+ void error()
+ void remove()
```

2.6.19 client.presenter.display.RegistrationDisplay - Interfaccia

Questa interfaccia viene implementata dalla classe `view.RegistrationView`.
L'interfaccia espone i seguenti metodi:

```
+ HasClickHandlers getRegisterButton()
+ HasClickHandlers getShowConfirmCodeButton()
+ HasClickHandlers getConfirmCodeButton()
+ HasValue<String> getUsername()
+ HasValue<String> getPassword()
+ HasValue<String> getRePassword()
+ HasValue<String> getEmail()
+ HasValue<String> getName()
+ HasValue<String> getLastName()
+ HasValue<String> getAddress()
+ HasValue<String> getConfirmCode()
+ HasValue<String> getConfirmUniqueId()
+ String getGender()
+ String getBirthDay()
+ String getBirthMonth()
+ String getBirthYear()
+ void setWrongUsername(String err)
+ void setWrongPassword(String err)
+ void setWrongEmail(String err)
+ void setWrongName(String err)
+ void setWrongLastName(String err)
+ void setWrongDate(String err)
+ void setOutcome(String result)
+ void setConfirmationOutcome(String result)
+ void changeView()
+ AbsolutePanel getView()
```

2.6.20 client.presenter.display.AvatarProfileManagementDisplay - Interfaccia

Questa interfaccia viene implementata dalla classe `view.AvatarProfileManagementView`.
L'interfaccia espone i seguenti metodi:

```
+ HasClickHandlers getChangeAvatarButton()
+ AbsolutePanel getView()
```

2.6.21 client.presenter.display.NewAvatarPopupDisplay - Interfaccia

Questa interfaccia viene implementata dalla classe `view.NewAvatarPopupView`.
L'interfaccia espone i seguenti metodi:

```
+ HasValue<String> getUrl()
+ HasClickHandlers getConfirmButton()
+ HasClickHandlers getCancelButton()
+ void removePopup()
```

2.6.22 client.presenter.display.PersonalDataProfileManagementDisplay - Interfaccia

Questa interfaccia viene implementata dalla classe `view.PersonalDataProfileManagementView`.
L'interfaccia espone i seguenti metodi:

```
+ void change()
+ void visualize()
+ void showError(String error)
+ HasValue<String> getNameTextBox()
+ HasValue<String> getLastNameTextBox()
+ HasValue<String> getEmailTextBox()
+ String getSexListBox()
+ String getDayListBox()
+ String getMonthListkBox()
+ String getYearListBox()
+ HasValue<String> getpasswordTextBox()
```

```
+ HasValue<String> getConfirmPasswordTextBox()
+ HasClickHandlers getChangeButton()
+ HasClickHandlers getConfirmButton()
+ AbsolutePanel getView()
```

2.6.23 client.presenter.display.NewEmailConfirmPopupDisplay - Interfaccia

Questa interfaccia viene implementata dalla classe `view.NewEmailConfirmPopupView`. L'interfaccia espone i seguenti metodi:

```
+ Button getRemoveButton()
+ void selfDelete()
+ AbsolutePanel getView()
```

2.6.24 client.presenter.display.ExtendedDataProfileManagementDisplay - Interfaccia

Questa interfaccia viene implementata dalla classe `view.ExtendedDataProfileManagementView`. L'interfaccia espone i seguenti metodi:

```
+ HasValue<String> getSecretCode()
+ HasClickHandlers getConfirmButton()
+ HasClickHandlers getCancelButton()
+ void setError(String error)
+ void close()
```

2.7 Package it.hourglass.myTalk.client.presenter

Questo package contiene la definizione di tutti i presenter presenti nel programma. Ogni presenter ha la funzione di gestire la logica di ogni singola pagina del sistema. La struttura base di ogni presenter è la medesima: ogni presenter possiede un'istanza di una interfaccia `Display` contenuta nel package `presenter.display` la quale espone i metodi che il presenter può richiedere alla view che gestisce.

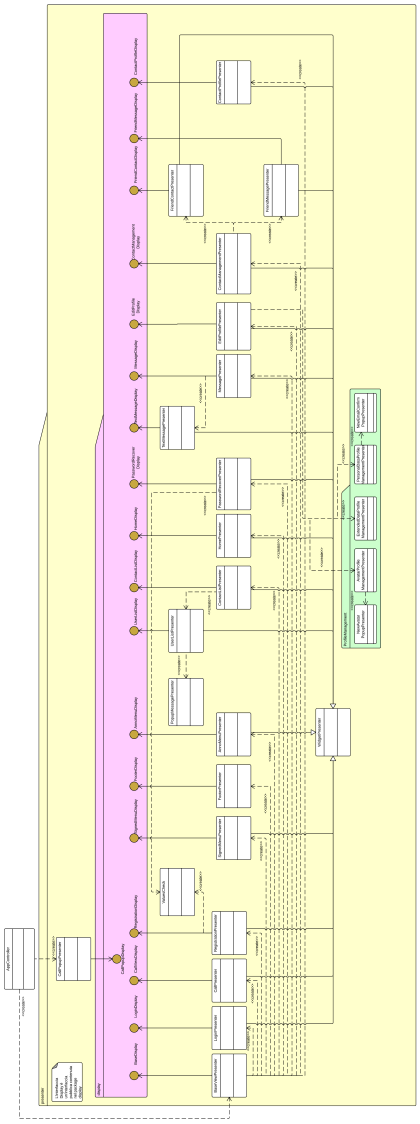

Figura 2: Package it.hourglass.myTalk.client.presenter

2.7.1 client.presenter.BaseViewPresenter - Classe

Funzione

Questa classe è il presenter della `view.BaseView`. Questa classe si occupa di eseguire lo switch view quando viene richiesto in seguito alla richiesta da parte della classe `AppController`.

Relazione d'uso con altri moduli

- La classe è utilizzata da:

  `it.hourglass.myTalk.client.appcontroller.AppController`

- La classe utilizza:

  `it.hourglass.myTalk.client.RPCService.RPCService`

  `it.hourglass.myTalk.client.RPCService.RPCServiceAsync`

  `it.hourglass.myTalk.client.event.HomePageRequestEvent`

  `it.hourglass.myTalk.client.event.LoginRequestEvent`

  `it.hourglass.myTalk.client.profile.Profile`

  `it.hourglass.myTalk.client.communication.WSConnection`

  `it.hourglass.myTalk.client.view.AnnoMenuView`

  `it.hourglass.myTalk.client.view.BaseView`

  `it.hourglass.myTalk.client.view.CallView`

  `it.hourglass.myTalk.client.view.ContactListView`

  `it.hourglass.myTalk.client.view.ContactManagementView`

  `it.hourglass.myTalk.client.view.ContactProfileView`

  `it.hourglass.myTalk.client.view.EditProfileView`

  `it.hourglass.myTalk.client.view.FooterView`

  `it.hourglass.myTalk.client.view.HomeView`

  `it.hourglass.myTalk.client.view.LoginView`

  `it.hourglass.myTalk.client.view.MessageView`

  `it.hourglass.myTalk.client.view.PasswordRecoverView`

  `it.hourglass.myTalk.client.view.RegistrationView`

  `it.hourglass.myTalk.client.view.SignedMenuView`

  `it.hourglass.myTalk.client.presenter.display.BaseDisplay`

  `com.google.gwt.event.shared.HandlerManager`

  `com.google.gwt.user.client.rpc.AsyncCallback`

  `com.google.gwt.user.client.ui.AbsolutePanel`

Attributi

- `boolean signedIn`
  Booleano che indica se l'utente è autenticato oppure no. E' utilizzato per controllare che un utente non autenticato non abbia accesso a pagine a cui non dovrebbe aver accesso.
- `WSConnection wsconnection`
  Riferimento alla WSConnection con il server WebSocket.
- `HandlerManager eventBus`
  Riferimento all'eventBus presenter in `client.appcontroller.AppController`. Esso sarà utilizzato per lanciare eventi.
- `BaseDisplay display`
  Questa variabile fa riferimento alla BaseView. Essa è di tipo `presenter.display.BaseDisplay` che è l'interfaccia Display che la BaseView implementa.
- `WidgetPresenter menuPresenter`
  Variabile che si riferisce al presenter del menù.
- `WidgetPresenter bodyPresenter`
  Variabile che si riferisce al presenter del corpo centrale della view.
- `FooterPresenter footerPresenter`
  Variabile che si riferisce al presenter del footer della view.
- `ContactListPresenter contacListPresenter`
  Variabile che si riferisce al presenter della lista contatti.

Metodi

+ `BaseViewPresenter(WSConnection wsconnection, HandlerManager eventBus)`
  Costruttore della classe. Si occupa di inizializzare i campi del BaseViewPresenter.
+ `void doSessionControl()`
  Metodo che fa un controllo sulla validità della sessione. Il metodo definisce la chiamata asincrona `AsyncCallback<String>` necessaria alla richiesta di invocazione del metodo `checkSession()` esposto dall'interfaccia `rpcservice.RPCService`. Se la RPC ha successo il sistema deve automaticamente richiedere l'autenticazione dell'utente modificando l'identificativo corrente dell'utente e lanciando un evento di tipo `LoginRequestEvent`, altrimenti inizializzare il sistema cone un utente anonimo.
+ `void init()`
  Metodo che si occupa di inizializzare la BaseView. Controlla se è presente una sessione (tramite l'invocazione del metodo `doSessionControl`) e aggiunge il footer alla view.
+ `void deleteContactList()`
  Metodo per rimuovere la lista contatti.

+ `void switchCall()`
  Metodo che si occupa di eseguire lo switch view del corpo centrale inserendo la CallView in seguito alla richiesta da parte della classe `AppController` in seguito alla ricezione di un evento `CallPageRequest`.

+ `void switchCallRequest()`
  Metodo che si occupa di eseguire lo switch view del corpo centrale quando si riceve una chiamata, nel costruttore del CallPresenter setterò a `true` il campo `Boolean request` in seguito alla richiesta da parte della classe `AppController` in seguito alla ricezione di un evento `CallRequestEvent`.

+ `void switchContactManagement()`
  Metodo che si occupa di eseguire lo switch view del corpo centrale inserendo la ContactManagementView in seguito alla richiesta da parte della classe `AppController` in seguito alla ricezione di un evento `ContactManagementPageRequest`.

+ `void switchEditProfile()`
  Metodo che si occupa di eseguire lo switch view del corpo centrale inserendo la EditProfileView in seguito alla richiesta da parte della classe `AppController` in seguito alla ricezione di un evento `ProfileManagementPageRequestEvent`.

+ `void switchFriendContactView()`
  Metodo che si occupa di eseguire lo switch view del corpo centrale inserendo la ContactView in seguito alla richiesta da parte della classe `AppController` in seguito alla ricezione di un evento `FriendContactProfilePageRequestEvent`.

+ `void refreshFooter()`
  Metodo che si occupa di richiedere il refresh del footer con il nuovo id identificativo in seguito alla richiesta da parte della classe `AppController` in seguito alla ricezione di un evento `IdChangedEvent`.

+ `void switchHome()`
  Metodo che si occupa di eseguire lo switch view del corpo centrale inserendo la HomeView in seguito alla richiesta da parte della classe `AppController` in seguito alla ricezione di un evento `HomePageRequestEvent`.

+ `void switchLogin()`
  Metodo che si occupa di eseguire lo switch view del corpo centrale inserendo la LoginView in seguito alla richiesta da parte della classe `AppController` in seguito alla ricezione di un evento `LoginPageRequestEvent`.

+ `void switchLoggedIn()`
  Questa funzione si occua di eseguire lo switch del sistema da stato non autenticato a stato atuenticato (cambiando il menù e aggiungendo la lista contatti). Si occupa anche di settare a `true` il campo `signedIn`.

+ `void switchLoggedOut()`
  Metodo invocato dopo che si effettua il logout. Toglie la lista utenti e cambia la menu view.

`+ void switchMessage()`
Metodo che si occupa di eseguire lo switch view del corpo centrale inserendo la MessageView in seguito alla richiesta da parte della classe `AppController` in seguito alla ricezione di un evento `MessagePageRequestEvent`.

`+ void switchRegistration()`
Metodo che si occupa di eseguire lo switch view del corpo centrale inserendo la RegistrationView in seguito alla richiesta da parte della classe `AppController` in seguito alla ricezione di un evento `RegistrationPageRequestEvent`.

`+ void switchPasswordRecover()`
Metodo che si occupa di eseguire lo switch view del corpo centrale inserendo la RecoverPasswordView in seguito alla richiesta da parte della classe `AppController` in seguito alla ricezione di un evento `PasswordRecoverPageRequestEvent`.

`+ void changeContactStatus(String name,int status)`
Metodo invocato quando un contatto nella lista contatti cambia il suo stato (online/offline). Questo metodo si occupa di richiedere al `contacListPresenter` di cambiare lo stato di un contatto.

`+ void removeContactFromContactList(String contact)`
Metodo che si occupa di richiedere l'eliminazione di un contatto dalla lista contatti. Questo metodo si occupa di richiedere al `contacListPresenter` di rimuovere un contatto da essa.

`+ void refreshContactList()`
Metodo che si occupa di richiedere il refresh della lista contatti quando viene aggiunto un utente accetta una richiesta di amicizia.

2.7.2 client.presenter.WidgetPresenter - Classe

Funzione

Questa interfaccia è l'interfaccia comune a tutti le classi presenter (tranne per la classe `BaseViewPresenter`)

Relazione d'uso con altri moduli

- L'interfaccia è utilizzata da:

  it.hourglass.myTalk.client.presenter.MessagePresenter

- L'interfaccia è implementata da:

  it.hourglass.myTalk.client.presenter.AnnoMenuPresenter

  it.hourglass.myTalk.client.presenter.BaseViewPresenter

  it.hourglass.myTalk.client.presenter.CallPresenter

  it.hourglass.myTalk.client.presenter.ContactListPresenter

  it.hourglass.myTalk.client.presenter.ContactManagementPresenter

  it.hourglass.myTalk.client.presenter.ContactProfilePresenter

  it.hourglass.myTalk.client.presenter.EditProfilePresenter

  it.hourglass.myTalk.client.presenter.FriendContactPresenter

  it.hourglass.myTalk.client.presenter.FriendMessagePresenter

  it.hourglass.myTalk.client.presenter.HomePresenter

  it.hourglass.myTalk.client.presenter.LoginPresenter

  it.hourglass.myTalk.client.presenter.MessagePresenter

  it.hourglass.myTalk.client.presenter.PasswordRecoverPresenter

  it.hourglass.myTalk.client.presenter.RegistrationPresenter

  it.hourglass.myTalk.client.presenter.SignedMenuPresenter

  it.hourglass.myTalk.client.presenter.TextMessagePresenter

  it.hourglass.myTalk.client.presenter.UserListPresenter

Metodi

`+ abstract AbsolutePanel getView()`

Questo metodo sarà implementato da ogni presenter per restituire il pannello contenente che il presenter controlla.

2.7.3 client.presenter.AnnoMenuPresenter - Classe

Funzione

Questa classe rappresenta il presenter per il menù anonimo. Esso si occupa di collegare i pulsanti della `view.AnnoMenuView` e di definire il comportamento che il sistema deve avere quando uno di quei pulsanti viene premuto.

Relazione d'uso con altri moduli

- La classe è utilizzata da:

  `it.hourglass.myTalk.client.presenter.BaseViewPresenter`

- La classe utilizza:

  `it.hourglass.myTalk.client.event.CallPageRequestEvent`

  `it.hourglass.myTalk.client.event.HomePageRequestEvent`

  `it.hourglass.myTalk.client.event.LoginPageRequestEvent`

  `it.hourglass.myTalk.client.event.PasswordRecoverPageRequestEvent`

  `it.hourglass.myTalk.client.event.RegistrationPageRequestEvent`

  `it.hourglass.myTalk.client.presenter.display.AnnoMenuDisplay`

  `it.hourglass.myTalk.client.presenter.WidgetPresenter`

  `com.google.gwt.event.dom.client.ClickHandler`

  `com.google.gwt.event.dom.client.HasClickHandlers`

  `com.google.gwt.event.shared.HandlerManager`

  `com.google.gwt.user.client.ui.AbsolutePanel`

- La classe implementa l'interfaccia :

  `it.hourglass.myTalk.client.presenter.WidgetPresenter`

Attributi

- `HandlerManager eventBus`
  Riferimento all'eventBus presenter in `client.appcontroller.AppController`. Esso sarà utilizzato per lanciare eventi.
- `AnnoMenuDisplay display`
  Questa variabile fa riferimento alla AnnoMenuView. Essa è di tipo `presenter.display.AnnoMenuD` cioè l'interfaccia che `view.AnnoMenuView` implementa.

Metodi

+ `AnnoMenuPresenter(HandlerManager eventBus,AnnoMenuDisplay display)`
  Costruttore della classe. Esso si occupa di inizializzare la classe e richiedere il collegamento dei bottoni con la view (il cui riferimento è contenuto in `display`) tramite il metodo `bind()`.

- `void bind()`
  Questo metodo si occupa di aggiungere il comportamento che il programma deve avere quando si richiede una pagina tramite il menù . Questo avviene aggiungendo un `ClickHandler` ad ogni Button presente nella view il cui riferimento è contenuto in display recuperandoli tramite le funzioni definite in `AnnoMenuView`. Più precisamente quando viene effettuata una richiesta tramite la pressione di un bottone per la richiesta di cambio della pagina verrà lanciato il corrispettivo evento che in `client.appcontroller.AppController` è stato aggiunto all'eventBus per richiedere lo switch del corpo centrale della view.

+ `AbsolutePanel getView()`
  Questo metodo si occupa di restituire l'AbsolutePanel contente l'istanza di `view.AnnoMenuView`. Questo avviene richiedendo il metodo `public AbsolutePanel getView()` definito nell'interfaccia `AnnoMenuDisplay` tramite la variabile `display`.

2.7.4 client.presenter.SignedMenuPresenter - Classe

Funzione

Questa classe rappresenta il presenter per il menù di un utente che si è autenticato. Esso si occupa di collegare i pulsanti della `view.SignedMenuView` e di definire il comportamento che il sistema deve avere quando uno di quei pulsanti viene premuto.

Relazione d'uso con altri moduli

- La classe è utilizzata da:

  `it.hourglass.myTalk.client.presenter.BaseViewPresenter`

- La classe utilizza:

  `it.hourglass.myTalk.client.RPCService.RPCService`

  `it.hourglass.myTalk.client.RPCService.RPCServiceAsync`

  `it.hourglass.myTalk.client.event.HomePageRequestEvent`

  `it.hourglass.myTalk.client.event.CallPageRequestEvent`

  `it.hourglass.myTalk.client.event.LogoutRequestEvent`

  `it.hourglass.myTalk.client.event.ProfileManagementPageRequestEvent`

  `it.hourglass.myTalk.client.presenter.WidgetPresenter`

  `it.hourglass.myTalk.client.presenter.display.SignedMenuDisplay`

  `com.google.gwt.event.dom.client.ClickEvent`

  `com.google.gwt.event.dom.client.ClickHandler`

  `com.google.gwt.event.dom.client.HasClickHandlers`

  `com.google.gwt.event.shared.HandlerManager`

  `com.google.gwt.user.client.rpc.AsyncCallback`

  `com.google.gwt.user.client.ui.AbsolutePanel`

- La classe implementa l'interfaccia :

  `it.hourglass.myTalk.client.presenter.WidgetPresenter`

Attributi

- `HandlerManager eventBus`
  Riferimento all'eventBus presenter in `client.appcontroller.AppController`.Esso sarà utilizzato per lanciare eventi.
- `SignedMenuDisplay display`
  Questa variabile fa riferimento alla SignedMenuView. Essa è di tipo `presenter.display.SignedMe` cioè l'interfaccia che `view.SignedMenuView` implementa.

Metodi

+ `SignedMenuPresenter(HandlerManager eventBus,SignedMenuDisplay display)`
  Costruttore della classe. Esso si occupa di inizializzare la classe e richiedere il collegamento dei bottoni (il cui riferimento è contenuto in `display`) con la view tramite il metodo `bind()`.
- `void bind()`
  Questo metodo si occupa di aggiungere il comportamento che il programma deve avere quando si richiede una pagina tramite il menù . Questo avviene aggiungendo un `ClickHandler` ad ogni Button presente nella view il cui riferimento è contenuto in display recuperandoli tramite le funzioni definite in `SignedMenuDisplay`. Più precisamente quando viene effettuata una richiesta tramite la pressione di un bottone per la richiesta di cambio della pagina verrà lanciato il corrispettivo evento che in `client.appcontroller.AppController` è stato aggiunto all'eventBus per richiedere lo switch del corpo centrale della view.
+ `AbsolutePanel getView()`
  Questo metodo si occupa di restituire l'AbsolutePanel contente l'istanza di `AnnoMenuView`. Questo avviene richiedendo il metodo `public AbsolutePanel getView()` definito nell'interfaccia `SignedMenuDisplay` tramite la variabile `display`.
+ `void doResetSession()`
  Questo metodo si occupa di eliminare la sessione attiva nel server. Per fare questo il metodo utilizza il metodo `resetSession()` della RPC.

2.7.5 client.presenter.CallPresenter - Classe

Funzione

Questa classe funge da presenter alla view `view.CallView`. Questa classe si occupa di gestire la logica per la creazione di una chiamata.

Relazione d'uso con altri moduli

- La classe è utilizzata da:

  `it.hourglass.myTalk.client.presenter.BaseViewPresenter`

- La classe utilizza:

  `it.hourglass.myTalk.client.Call`

  `it.hourglass.myTalk.client.communication.WSConnection`

  `com.google.gwt.event.dom.client.ClickEvent`

  `com.google.gwt.event.dom.client.ClickHandler`

  `com.google.gwt.event.dom.client.HasClickHandlers`

  `com.google.gwt.event.shared.HandlerManager`

  `com.google.gwt.user.client.ui.AbsolutePanel`

  `com.google.gwt.user.client.ui.TextArea`

  `com.google.gwt.user.client.ui.TextBox`

  `com.google.gwt.media.client.Video`

  `it.hourglass.myTalk.client.presenter.WidgetPresenter`

  `it.hourglass.myTalk.client.presenter.display.CallDisplay`

- La classe implementa l'interfaccia :

  `it.hourglass.myTalk.client.presenter.WidgetPresenter`

Attributi

- `WSConnection wsconnection`
  Variabile che si riferisce all'oggetto di tipo WSConnection.
- `HandlerManager eventBus`
  Riferimento all'eventBus presente in `client.appcontroller.AppController`. Esso sarà utilizzato per lanciare eventi.
- `CallDisplay display`
  Riferimento alla view `view.CallView` di cui il presenter gestisce la logica. Essa è di tipo `presenter.display.CallDisplay` cioè l'interfaccia che `view.CallView` implementa.

Metodi

+ `CallPresenter(WSConnection wsconnection, HandlerManager eventBus, CallDisplay display,boolean request)`
  Costruttore della classe. Si occupa di inizializzare la variabili. Se il campo `request` è vero si occuperà di fare partire automaticamente una chiamata invocando il metodo `wsconnection.doCall()` definito in `client.communication.WSConnection`.

+ `void bind()`
  Questo metodo si occupa di aggiungere la logica ai pulsanti della view `view.CallView` il cui riferimento è contenuto in `display`. Questo avviene aggiungendo un `ClickHandler` ad ogni bottone recuperandolo tramite la funzione definita in `CallDisplay`. Se viene premuto il pulsante per richiedere una chiamata si deve estrarre il nome dell'utente che si vuole chiamare e il tipo di stream della chiamata (questo avviene tramite i metodi definiti in `CallDisplay`) per poi inizializzare una nuova chiamata tramite il metodo `wsconnection.doCall` definito in `client.communication.WSConnection()`. Se viene invece premuto il pulsante per richiedere la chiusura di una chiamata si dovrà invocare il metodo `wsconnection.closeCall()` definito in `client.communication.WSConnection`.

+ `void show()`
  Metodo per richiedere di rendere visibile l'elemento video e le statistiche della chiamata nella view.

+ `void sendChatMessage()`
  Metodo invocato quando si invia un messaggio dalla chat. Questo metodo si occupa di di invocare il metodo `sendMessage()` definito in `client.communication.WSConnection` per l'invio di un messaggio in chat ad un utente. Infine si occupa di stampare il messaggio nell'area apposita della view.

+ `AbsolutePanel getView()`
  Questo metodo restituisce il pannello contente la `view.ContactListView` che il presenter gestisce. Questo avviene richiedendo il metodo `getView()` definito nell'interfaccia `CallDisplay` tramite `display`.

2.7.6 client.presenter.CallPopupPresenter - Classe

Funzione

Questa classe è il presenter alla view `view.CallPopupView` e si occupa della parte logica per la ricezione di una chiamata (accettare, rifiutare o ignorare una chiamata).

Relazione d'uso con altri moduli

- La classe è utilizzata da:

  `it.hourglass.myTalk.client.appcontroller.AppController`

- La classe utilizza:

  `it.hourglass.myTalk.client.event.CallRequestEvent`

  `it.hourglass.myTalk.client.event.CallRequestRefusedEvent`

  `it.hourglass.myTalk.client.presenter.display.CallPopupDisplay`

  `com.google.gwt.event.dom.client.ClickEvent`

  `com.google.gwt.event.dom.client.ClickHandler`

  `com.google.gwt.event.dom.client.HasClickHandlers`

  `com.google.gwt.event.shared.HandlerManager`

  `com.google.gwt.user.client.Timer`

Attributi

- `HandlerManager eventBus`
  Riferimento all'eventBus presenter in `client.appcontroller.AppController`. Esso sarà utilizzato per lanciare gli eventi.
- `CallPopupDisplay display`
  Variabile che si riferisce all'oggetto di tipo `view.CallPopupView`, che rappresenta la view (il popup) che appare quando si riceve una chiamata.
- `boolean answered`
  Booleano che contiene `true` se il destinatario risposto alla chiamata (utilizzato per il controllo del timer per ritenere ignorata una chiamata).

Metodi

+ `CallPopupPresenter(WSConnection wsconnection, HandlerManager eventBus, CallPopupDisplay display)`
  Costruttore della classe. Si occupa di inizializzare le variabili e di inizializzare il timer per ignorare una chiamata (se non rispondo entro 30 secondi la chiamata viene ritenuta ignorata). Infine si occupa di collegare i pulsanti della view corrispettiva `CallPopup` (il cui riferimento è contenuto in `display`) tramite il metodo `public void bind()`.

`+ void bind()`

Questo metodo si occupa di aggiungere la logica ai pulsanti della view `view.CallPopup` il cui riferimento è contenuto in `display`. Questo avviene aggiungendo un `ClickHandler` ad ogni bottone recuperandoli tramite le funzioni definite in `CallPopupDisplay`.
In base al pulsante che ho premuto lancerò un evento (`CallRequestEvent` se ho accettato o `CallRequestRefusedEvent` se ho rifiutato) tramite `eventBus`.

2.7.7 client.presenter.ContactListPresenter - Classe

Funzione

Questa classe funge da presenter alla view `view.ContactListView`. Questa classe si occupa di gestire la logica per la creazione della lista contatti di un utente che si è autenticato.

Relazione d'uso con altri moduli

- La classe è utilizzata da:

  `it.hourglass.myTalk.client.presenter.BaseViewPresenter`

- La classe utilizza:

  `it.hourglass.myTalk.client.shared.Profile`

  `com.google.gwt.event.dom.client.ClickEvent`

  `com.google.gwt.event.dom.client.ClickHandler`

  `com.google.gwt.event.dom.client.HasClickHandlers`

  `com.google.gwt.event.shared.HandlerManager`

  `com.google.gwt.user.client.ui.AbsolutePanel`

  `it.hourglass.myTalk.client.presenter.ContactListPresenter`

  `it.hourglass.myTalk.client.view.UserListView`

  `it.hourglass.myTalk.client.communication.WSConnection`

  `it.hourglass.myTalk.client.presenter.display.ContactListDisplay`

  `it.hourglass.myTalk.client.event.ContactManagementPageRequestEvent`

  `java.util.HashMap`

  `java.util.Map.Entry`

- La classe implementa l'interfaccia :

  `it.hourglass.myTalk.client.presenter.WidgetPresenter`

Attributi

- `HashMap<String, Boolean> contactList`
  Variabile che contiene la friendList presente in `Profile`.
- `WSConnection wsconnection`
  Variabile che si riferisce all'oggetto di tipo WSConnection.
- `HandlerManager eventBus`
  Riferimento a una variabile che avrà la funzione di ricevitore di eventi.
- `ContactListDisplay display`
  Variabile che si riferisce all'oggetto di tipo `view.ContactListView` di cui il presenter gestisce la logica. Essa è di tipo `ContactListDisplay` cioè l'interfaccia che `view.ContactListView` implementa.

- HashMap<String, UserListPresenter> friendContactList
  Questa HashMap contiene tutti gli `UserListPresenter` che vengono creati in seguito all'inizializzazione della classe.

Metodi

+ ContactListPresenter(HandlerManager eventBus, WSConnection wsconnection, ContactListDisplay display)
  Costruttore della classe ContactListPresenter. Esso si occupa di recuperare la friendList da `client.shared.Profile`. Infine si occupa di chiamare i metodi initList() e addListView() e di collegare i pulsanti della view corrispettiva `view.ContactListView` (il cui riferimento è contenuto in display) tramite il metodo bind().

+ void initList()
  Metodo che inizializza la lista contatti. Per ogni contatto presente in contactList il metodo inizializza un'istanza di `UserListPresenter` inserendola nell'hashMap friendContactList .

+ void addListView()
  Metodo che si occupa di aggiungere alla ContactListView tutte le istanze di UserListView contenute in friendContactList.

+ void bind()
  Questo metodo si occupa di aggiungere la logica al pulsanti della view `view.ContactListView` il cui riferimento è contenuto in display. Questo avviene aggiungendo un `ClickHandler` ad ogni bottone recuperandoli tramite le funzioni definite in `ContactListDisplay`. In particolare si occupa di aggiungere la logica per la richiesta della pagina di gestione delle amicizie. Questo avviene aggiungendo un `ClickHandler` al bottone per la richiesta recuperandolo tramite il metodo getContactManagmentPage() definito in `ContactListDisplay`. Se si preme tale pulsante si lancia un evento di tipo `ContactManagementPageRequestEvent` tramite eventBus .

+ AbsolutePanel getView()
  Metodo che ritorna la view `viewContactListView` che il presenter gestisce. Questo avviene richiedendo il metodo getView() definito nell'interfaccia `ContactListDisplay` tramite display.

+ void removeContactList()
  Metodo che rimuove la lista contatti.

+ void changeStatus(String name,Boolean status)
  Metodo invocato quando un utente della nostra lista cambia il suo stato (online o offline).

+ void removeContact(String contact)
  Metodo invocato per rimuovere un contatto dalla lista amici. Esso si occupa di eliminare il contatto dalla friendContactList e di rimuovere dalla `ContactListView` l'oggetto `UserListView` di tale utente.

2.7.8 client.presenter.FooterPresenter - Classe

Funzione

Questa classe funge da presenter alla FooterView e permette di aver accesso e di controllare i suoi elementi.

Relazione d'uso con altri moduli

- La classe è utilizzata da:

  `it.hourglass.myTalk.client.presenter.BaseViewPresenter`

- La classe utilizza:

  `com.google.gwt.user.client.ui.AbsolutePanel`

  `it.hourglass.myTalk.client.presenter.display.FooterDisplay`

Attributi

- `HandlerManager eventBus`
  Riferimento a una variabile che avrà la funzione di ricevitore di eventi.
- `FooterDisplay display`
  Variabile che si riferisce all'oggetto di tipo `view.FooterView` di cui il presenter gestisce la logica. Essa è di tipo `FooterDisplay` cioè l'interfaccia che `view.FooterView` implementa.

Metodi

+ `FooterPresenter(HandlerManager eventBus,FooterDisplay display)`
  Costruttore della classe. Si occupa di inizializzare le variabili.
+ `String getUsername()`
  Metodo che ritorna il proprio username.
+ `setUsername(String username)`
  Metodo che imposta il proprio username.
+ `void refreshFooter(String Id)`
  Metodo che effettua il refresh del footer impostando un nuovo username da visualizzare.
+ `AbsolutePanel getView()`
  Questo metodo restituisce il pannello contente la `view.FooterView` che il presenter gestisce. Questo avviene richiedendo il metodo `getView()` definito nell'interfaccia `FooterDisplay` tramite `display`.

2.7.9 client.presenter.UserListPresenter - Classe

Funzione

Questa classe funge da presenter alla view `view.UserListView`. Per ogni contatto presente nella lista contatti di un utente la classe `ContactListPresenter` si occupa di creare un'istanza di questa classe. Questa classe gestisce la logica per richiedere una chiamata rapida, la richiesta del popup per l'invio di un messaggio e la richiesta per la visualizzazione del profilo di un determinato utente.

Relazione d'uso con altri moduli

- La classe è utilizzata da:

  `it.hourglass.myTalk.client.presenter.ContactListPresenter`

- La classe utilizza:

  `it.hourglass.myTalk.client.communication.WSConnection`

  `it.hourglass.myTalk.client.event.CallPageRequestEvent`

  `it.hourglass.myTalk.client.event.FriendContactProfilePageRequest`

  `it.hourglass.myTalk.client.shared.Profile`

  `it.hourglass.myTalk.client.view.PopupMessageView`

  `it.hourglass.myTalk.client.presenter.display.UserListDisplay`

  `com.google.gwt.event.dom.client.ClickEvent`

  `com.google.gwt.event.dom.client.ClickHandler`

  `com.google.gwt.event.dom.client.HasClickHandlers`

  `com.google.gwt.event.shared.HandlerManager`

  `com.google.gwt.user.client.ui.AbsolutePanel`

- La classe implementa l'interfaccia :

  `it.hourglass.myTalk.client.presenter.WidgetPresenter`

Attributi

- `WSConnection wsconnection`
  Variabile che si riferisce all'oggetto di tipo WSConnection.
- `HandlerManager eventBus`
  Riferimento all'eventBus presente in `client.appcontroller.AppController`. Esso sarà utilizzato per lanciare gli eventi.
- `Entry<String, Boolean> contactInformation`
  Questa `Entry` deriva dall'iterazione della mappa `friendList` in `ContactListPresenter`. Essa contiene il nome e lo stato di un particolare utente.
- `UserListDisplay display`
  Riferimento alla `view.UserListView` di cui il presenter gestisce la logica. Essa è di tipo `UserListDisplay` cioè l'interfaccia che `view.UserListView` implementa.

Metodi

+ `UserListPresenter(HandlerManager eventBus,WSConnection wsConnection, Entry<String, Boolean> contactInformation,UserListDisplay view)`
  Costruttore della classe `UserListPresenter`. Si occupa di inizializzare i campi dati e di invocare il metodo `bind()`.

- `void bind()`
  Questo metodo si occupa di collegare la logica dei bottoni della `view.UserListView` il cui riferimento è contenuto in `display`. Questo avviene aggiungendo un `ClickHandler` ad ogni bottone recuperandolo tramite la funzione definita in `UserListDisplay`. Per ogni utente saranno presenti tre bottoni: uno per la chiamata rapida (si imposterà il campo `rapidCall` in `client.shared.User` e si lancerà l'evento `CallPageRequestEvent` tramite `eventBus`), un bottone per richiedere il popup per inviare un messaggio (`PopupMessagePresenter`) e un bottone per visualizzare il profilo di un utente (per quest'ultimo si imposterà il campo `ViewFriendProfile` in `client.shared.User` e si lancerà l'evento `CallPageRequestEvent` tramite `eventBus`).

+ `AbsolutePanel getView()`
  Questo metodo restituisce il pannello contente la `view.ContactListView` che il presenter gestisce. Questo avviene richiedendo il metodo `getView()` definito nell'interfaccia `UserListDisplay` tramite `display`.

+ `void changeStatus(boolean status)`
  Questo metodo si occupa di cambiare lo status dell'utente. Cambierà il valore in `contactInformation` e invocherà il metodo `changeStatus(boolean status)` su `display` per richiedere la modifica della view.

+ `void remove()`
  Questo metodo si occupa di rimuovere il pannello della `view.UserListView` dalla view `view.ContactListView` padre.

2.7.10 client.presenter.ContactManagmentPresenter - Classe

Funzione

Questa classe funge da presenter alla view `view.ContactManagmentView`. Questa classe si occupa di gestire la logica per la gestione delle amicizie di un utente. Questo presenter si occupa di visualizzare le richieste di amicizia di un utente (dando anche la possibilità di accettarle o rifiutarle) e di eliminare un'amicizia che era stata precedentemente accettata.

Relazione d'uso con altri moduli

- La classe è utilizzata da:

  `it.hourglass.myTalk.client.presenter.BaseViewPresenter`

- La classe utilizza:

  `java.util.ArrayList`

  `java.util.HashMap`

  `java.util.List`

  `java.util.Map.Entry`

  `it.hourglass.myTalk.client.RPCService.RPCService`

  `it.hourglass.myTalk.client.RPCService.RPCServiceAsync`

  `com.google.gwt.event.shared.HandlerManager`

  `it.hourglass.myTalk.client.shared.Message`

  `it.hourglass.myTalk.client.shared.Profile`

  `it.hourglass.myTalk.client.view.FriendContactView`

  `it.hourglass.myTalk.client.view.FriendMessageView`

  `it.hourglass.myTalk.client.presenter.display.ContactManagementDisplay`

  `com.google.gwt.event.dom.client.ClickEvent`

  `com.google.gwt.event.dom.client.ClickHandler`

  `com.google.gwt.event.dom.client.HasClickHandlers`

  `com.google.gwt.user.client.rpc.AsyncCallback`

  `com.google.gwt.user.client.ui.AbsolutePanel`

  `com.google.gwt.user.client.ui.HasValue`

- La classe implementa l'interfaccia :

  `it.hourglass.myTalk.client.presenter.WidgetPresenter`

Attributi

- `HandlerManager eventBus`
  Riferimento all'eventBus presenter in `client.appcontroller.AppController`. Esso sarà utilizzato per lanciare gli eventi.

- `ContactManagementDisplay display`
  Riferimento alla view `ContactManagmentView` di cui il presenter gestisce la logica. Essa è di tipo `ContactManagmentDisplay` cioè l'interfaccia che `view.ContactManagmentView` implementa.
- `HashMap<String, Boolean> contactMap`
  Riferimento alla `friendList` presente nella classe `client.shared.Profile`.
- `List<FriendContactPresenter> friendContact`
  Questa lista conterrà tutte le istanze di `FriendContactPresenter` che la classe si occupa di creare. Per ogni amico presente nella `friendList` presente nella classe `client.shared.Profile` verrà creata un istanza di `FriendContactPresenter`.
- `List<FriendMessagePresenter> friendMessage`
  Questa lista conterrà tutte le istanze di `FriendMessagePresenter` che la classe si occupa di creare. Per ogni richiesta di amicizia presente nei messaggi estratti in `client.shared.Profile` verrà creata un istanza di `FriendMessagePresenter`.

Metodi

- `+ ContactManagementPresenter(HandlerManager eventBus,ContactManagementDisplay display)`
  Costruttore della classe. Si occupa di inizializzare i campi dati e di estrarre le amicizie e le richieste di amicizia dal Profilo. Le amicizie e i messaggi si otterranno utilizzando i metodi `getFriendList()` e `getMessageList()` resi disponibili dalla classe `client.shared.Profile`. Per ogni richiesta di amicizia estratta la classe creerà un istanza di `FriendMessagePresenter` e la inserirà in `friendMessage`, mentre per ogni amico estratto si creerà un istanza di `FriendContactPresenter` e la inserirà in `friendContact`. Successivamente si occuperà di aggiungere alla `view.ContactManagmentView` le view corrispettive di ogni `FriendMessagePresenter` ed `FriendContactPresenter`.
- `+ void addContact()`
  Questo metodo si occupa di aggiungere per ogni `FriendContactPresenter` presente in `friendContact` la corrispettiva view alla `ContactManagmentView`.
- `+ void addMessage()`
  Questo metodo si occupa di aggiungere per ogni `FriendMessagePresenter` presente in `friendMessage` la corrispettiva view alla `ContactManagmentView`.
- `+ void bind()`
  Questo metodo si occupa di gestire la logica dei pulsanti della `view.ContactManagmentView` il cui riferimento è contenuto in `display`. Questo avviene aggiungendo un `ClickHandler` ad ogni bottone recuperandolo tramite la funzione definita in `ContactManagmentDisplay`. Il metodo più precisamente gestirà la logica per inviare una richiesta di amicizia ad un utente registrato. Questo si otterrà aggiungendo un `ClickHandler` al pulsante `FriendshipRequestButton` (presente in `view.ContactManagmentView`) recuoerato grazie al metodo `getFriendshipRequestButton()` invocato su `ContactManagmentDisplay`

e gestendo la richiesta di conseguenza. Tale richiesta verrà effettuata invocando il metodo `friendReq()`.

`+ AbsolutePanel getView()`
Questo metodo restituisce il pannello contente la `view.ContactManagmentView` che il presenter gestisce. Questo avviene richiedendo il metodo `getView()` definito nell'interfaccia `ContactManagmentDisplay` tramite `display`.

`- void friendReq(final Message mess)`
Questo metodo si occupa di inviare il messaggio di richiesta di amicizia creato nella gestione della risposta del click del pulsante `FriendshipRequestButton` (in `view.ContactManagmentView`). Il metodo definisce la chiamata asincrona `AsyncCallback<String>` necessaria alla richiesta di invocazione del metodo `sendMessage()` esposto dall'interfaccia `it.hourglass.myTalk.rpcservice.RPCService`. In caso di successo il metodo notificherà all'utente la buona riuscita della richiesta altrimenti notificherà all'utente l'errore.

2.7.11 client.presenter.FriendMessagePresenter - Classe

Funzione

Questa classe funge da presenter per la view `view.FriendMessageView`. Questa classe si occupa di gestire la logica per accettare o rifiutare una richiesta di amicizia nella pagina di gestione delle amicizie. Un istanza di questa classe viene creata per ogni richiesta di amicizia ricevuta dalla classe `ContactManagmentPresenter`.

Relazione d'uso con altri moduli

- La classe è utilizzata da:

  `it.hourglass.myTalk.client.presenter.ContactManagmentPresenter`

- La classe utilizza:

  `it.hourglass.myTalk.client.shared.Profile`

  `it.hourglass.myTalk.client.RPCService.RPCService`

  `it.hourglass.myTalk.client.RPCService.RPCServiceAsync`

  `it.hourglass.myTalk.client.shared.Message`

  `it.hourglass.myTalk.client.shared.Profile`

  `it.google.gwt.event.dom.client.ClickEvent`

  `it.google.gwt.event.dom.client.ClickHandler`

  `it.google.gwt.event.dom.client.HasClickHandlers`

  `it.google.gwt.user.client.rpc.AsyncCallback`

  `it.google.gwt.user.client.ui.AbsolutePanel`

  `it.hourglass.myTalk.client.presenter.display.FriendMessageDisplay`

- La classe implementa l'interfaccia :

  `it.hourglass.myTalk.client.presenter.WidgetPresenter`

Attributi

- `FriendMessageDisplay display`
  Riferimento alla view `view.FriendMessageView` di cui il presenter gestisce la logica. Essa è di tipo `FriendMessageDisplay` cioè l'interfaccia che `view.FriendMessageView` implementa.

- `Message friendReq`
  Riferimento al messaggio di richiesta di amicizia. Esso contiene l'identificativo necessario a mandare la risposta.

Metodi

+ `FriendMessagePresenter(FriendMessageDisplay display, Message friendReq)`
  Costruttore della classe FriendMessagePresenter. Si occupa dell'inizializzazione dei campi dati e di collegare i pulsanti della view corrispettiva (il cui riferimento è contenuto in `display`) tramite l'invocazione del metodo `bind`.

`+ void bind()`
Questo metodo si occupa di aggiungere la logica ai pulsanti della view `view.FriendMessageView` il cui riferimento è contenuto in `display` . Questo avviene aggiungendo un `ClickHandler` ad ogni bottone recuperandolo tramite la funzione definita in `FriendMessageDisplay`. In particolare si occupa di collegare i pulsanti per accettare o rifiutare una richiesta di amicizia. La risposta sarà gestita tramite l'invocazione del metodo `friendReqReply()`.

`- void friendReqReply(boolean accepted)`
Questo metodo si occupa di inviare la risposta ad un messaggio di amicizia. In esso si definisce l'AsyncCallback<Boolean> necessaria per l'invocazione del metodo `friendshake` della RPC. Se la RPC ha successo verrà segnalato all'utente il risultato e se la richiesta di amicizia ha avuto successo invoco il metodo `client.shared.Profile.addFriend()` per mandare una notifica all'utente a cui ho accettato l'amicizia.

`+ AbsolutePanel getView()`
Questo metodo restituisce il pannello contente la `view.FriendMessageView` che il presenter gestisce. Questo avviene richiedendo il metodo `getView()` definito nell'interfaccia `FriendMessageDisplay` tramite `display`.

2.7.12 client.presenter.EditProfilePresenter - Classe

Funzione

Costruisce ed espone le sottoclassi designate alle modifiche delle varie sezioni del profilo (avatar, dati personali, dati estesi) oltre che fornire l'accesso all'amministrazione dei messaggi.

Relazione d'uso con altri moduli

- La classe utilizza:

  `it.hourglass.myTalk.client.event.MessagePageRequestEvent`

  `it.hourglass.myTalk.client.presenter.profilemanagement.AvatarProfileManagementPresenter`

  `it.hourglass.myTalk.client.presenter.profilemanagement.ExtendedDataProfileManagementPresenter`

  `it.hourglass.myTalk.client.presenter.profilemanagement.PersonalDataProfileManagementPresenter`

  `it.hourglass.myTalk.client.view.profilemanagement.AvatarProfileManagementView`

  `it.hourglass.myTalk.client.view.profilemanagement.ExtendedDataProfileManagementView`

  `it.hourglass.myTalk.client.view.profilemanagement.PersonalDataProfileView`

  `it.hourglass.myTalk.client.presenter.display.EditProfileDisplay`

- La classe è utilizzata da:

  `it.hourglass.myTalk.client.presenter.BaseViewPresenter`

  `it.hourglass.myTalk.client.view.EditProfileView`

Attributi

- `final HandlerManager eventBus`
  Riferimento all'eventBus presenter in `client.appcontroller.AppController`. Esso sarà utilizzato per lanciare gli eventi.
- `final EditProfileDisplay display`
  Variabile che si riferisce all'oggetto di tipo `view.EditProfileView` di cui il presenter gestisce la logica. Essa è di tipo `EditProfileDisplay` cioè l'interfaccia che `view.EditProfilView` implementa.
- `AvatarProfileManagementPresenter avatarPresenter`
  Classe designata alla modifica dell'avatar del profilo.
- `PersonalDataProfileManagementPresenter personalDataPresenter`
  Classe designata alla modifica dei dati personali del profilo.

- `ExtendedDataProfileManagementPresenter extendedDataPresenter`
  Classe designata alla modifica dei dati estesi del profilo.

Metodi

+ `EditProfilePresenter(HandlerManager eventBus, EditProfileDisplay display)`
  Costruttore della classe. Assegna alle variabili i parametri passati ed evoca i metodi `init()` e `bind()` per l'inizializzazione.

- `void init()`
  Vengono create e assegnate le istanze di presenter necessarie da associare ai campi privati della classe. Esse gestiranno la modifica delle varie sezioni del Profilo dell'utente.

- `void bind()`
  Aggiunge il comportamento che il programma deve avere quando viene cliccato il tasto per la visualizzazione dei messaggi.

+ `AbsolutePanel getView()`
  Questo metodo si occupa di restituire l'AbsolutePanel contente l'istanza di `view.EditProfileView`. Questo avviene richiedendo il metodo `public AbsolutePanel getView()` definito nell'interfaccia `EditProfileDisplay` tramite `display`.

2.7.13 client.presenter.FriendContactPresenter - Classe

Funzione

Questa classe funge da presenter alla view `view.FriendContactView`. Questa classe si occupa di gestire la logica per eliminare un utente dalla lista amici. Un istanza di questa classe viene creata per ogni amico presente nella lista amici dalla classe `ContactManagmentPresenter` alla richiesta di accesso della pagina di gestione delle amicizie.

Relazione d'uso con altri moduli

- La classe è utilizzata da:

  `it.hourglass.myTalk.client.presenter.ContactManagmentPresenter`

- La classe utilizza:

  `com.google.gwt.user.client.rpc.AsyncCallback`

  `it.hourglass.myTalk.client.RPCService.RPCService`

  `it.hourglass.myTalk.client.RPCService.RPCServiceAsync`

  `com.google.gwt.event.shared.HandlerManager`

  `it.hourglass.myTalk.client.event.RemoveFriendContactEvent`

  `it.hourglass.myTalk.client.shared.Profile`

  `it.hourglass.myTalk.client.presenter.display.FriendContactDisplay`

  `com.google.gwt.event.dom.client.ClickEvent`

  `com.google.gwt.event.dom.client.ClickHandler`

  `com.google.gwt.event.dom.client.HasClickHandlers`

  `com.google.gwt.user.client.ui.AbsolutePanel`

- La classe implementa l'interfaccia :

  `it.hourglass.myTalk.client.presenter.WidgetPresenter`

Attributi

- `String friendId`
  Contiene l'identificativo dell'utente amico.
- `HandlerManager eventBus`
  Riferimento all'eventBus presenter in `client.appcontroller.AppController`. Esso sarà utilizzato per lanciare eventi.
- `FriendContactDisplay display`
  Riferimento alla view `view.FriendContactView` di cui il presenter gestisce la logica. Essa è di tipo `FriendContactDisplay` cioè l'interfaccia che `view.FriendContactView` implementa.

Metodi

+ `FriendContactPresenter(HandlerManager eventBus,FriendContactDisplay display, String friendUniqueId)`
  Costruttore della classe `FriendContactPresenter`. Si occupa di inizializzare i campi dati e di collegare i pulsanti della view corrispettiva (il cui riferimento è contenuto in `display`) tramite l'invocazione del metodo `bind()`.

- `void bind()`
  Questo metodo si occupa di aggiungere la logica al pulsante della `view.FriendContactView` il cui riferimento è contenuto in `display` per richiedere l'eliminazione di un utente dalla lista amici. Questo avviene aggiungendo un `ClickHandler` al bottone recuperandolo tramite le funzioni definite in `FriendContactDisplay`. Se premuto tale bottone si invoca il metodo `deleteFriend()` che si occupa di gestire la richiesta.

- `void deleteFriend()`
  Questo metodo si occupa della richiesta di eliminazione di una amicizia dalla propria lista amici. In esso si definisce l'AsyncCallback<Boolean> necessaria all'invocazione del metodo `removeFriend()` della RPC. Se la RPC ha successo l'utente è stato rimosso dalla lista utenti nel database e dalla lista utenti locale. Infine gli sarà notificata l'eliminazione tramite l'invocazione del metodo `client.Profile.removeFriend()`.

+ `AbsolutePanel getView()`
  Questo metodo restituisce il pannello contente la `view.FriendContactView` che il presenter gestisce. Questo avviene richiedendo il metodo `getView()` definito nell'interfaccia `FriendContactDisplay` tramite `display`.

2.7.14 client.presenter.ContactProfilePresenter - Classe

Funzione

Questa classe funge da presenter alla view `view.ContactProfileView`. Si occupa di gestire la logica per il recupero del profilo di un altro utente e l'inizializzazione della vista contenente i dati estratti.

Relazione d'uso con altri moduli

- La classe è utilizzata da:

  `it.hourglass.myTalk.client.presenter.BaseViewPresenter`

- La classe utilizza:

  `it.hourglass.myTalk.client.RPCService.RPCService`

  `it.hourglass.myTalk.client.RPCService.RPCServiceAsync`

  `it.hourglass.myTalk.client.event.HomePageRequestEvent`

  `it.hourglass.myTalk.client.shared.Profile`

  `it.hourglass.myTalk.client.shared.User`

  `it.hourglass.myTalk.client.presenter.display.ContactProfileDisplay`

  `com.google.gwt.event.shared.HandlerManager`

  `com.google.gwt.user.client.rpc.AsyncCallback`

  `com.google.gwt.user.client.ui.AbsolutePanel`

- La classe implementa l'interfaccia :

  `it.hourglass.myTalk.client.presenter.WidgetPresenter`

Attributi

- `ContactProfileDisplay display`
  Riferimento alla view `view.ContactProfileView` di cui il presenter gestisce la logica.

Metodi

+ `ContactProfilePresenter(HandlerManager eventBus,ContactProfileDisplay display)`
  Costruttore della classe `ContactProfilePresenter`. Si occupa di inizializzare il campo dato `display` e di recuperare il profilo dell'user amico richiesto. Tale richiesta è fatta in risposta alla richiesta da parte dell'utente tramite il presenter `UserListPresenter`. La richiesta di recupero viene fatta eseguendo il metodo `recover()`.

- `void recover()`
  Questo metodo si occupa del recupero del profilo di un utente per la sua successiva visualizzazione. In esso si definisce l'AsyncCallback<User> necessaria all'invocazione del metodo `fetchFriendProfile()` della RPC. Questo metodo

recupera il nome dell'utente di cui si è richiesto il profilo e si invoca il metodo `recover()` dell'RPC. Se l'RPC ha avuto successo si inizializza la view `view.ContactProfileView` (il cui riferimento è contenuto in `display`) con i dati recuperati tramite l'invocazione del metodo `init()`.

`+ AbsolutePanel getView()`

Questo metodo restituisce il pannello contente la `view.ContactProfileView` che il presenter gestisce. Questo avviene richiedendo il metodo `getView()` definito nell'interfaccia `ContactProfileDisplay` tramite `display`.

2.7.15 client.presenter.HomePresenter - Classe

Funzione

Questa classe funge da presenter alla view `view.HomeView`. Questa classe non si occupa di gestire una logica particolare ma ha il compito di fornire il presenter alla HomeView.

Relazione d'uso con altri moduli

- La classe è utilizzata da:

  `it.hourglass.myTalk.client.presenter.BaseViewPresenter`

- La classe utilizza:

  `it.hourglass.myTalk.client.RPCService.RPCService`

  `it.hourglass.myTalk.client.RPCService.RPCServiceAsync`

  `it.hourglass.myTalk.client.event.HomePageRequestEvent`

  `it.hourglass.myTalk.client.shared.Profile`

  `it.hourglass.myTalk.client.shared.User`

  `it.hourglass.myTalk.client.presenter.display.HomeDisplay`

  `com.google.gwt.event.shared.HandlerManager`

  `com.google.gwt.user.client.rpc.AsyncCallback`

  `com.google.gwt.user.client.ui.AbsolutePanel`

- La classe implementa l'interfaccia :

  `it.hourglass.myTalk.client.presenter.WidgetPresenter`

Attributi

`- HomeDisplay display`
Riferimento alla view `HomeView` di cui il presenter gestisce la logica.

Metodi

`+ AbsolutePanel getView()`
Questo metodo restituisce il pannello contente la `HomeView` che il presenter gestisce. Questo avviene richiedendo il metodo `getView()` definito nell'interfaccia `HomeDisplay` tramite `display`.

2.7.16 client.presenter.LoginPresenter - Classe

Funzione

Questa classe funge da presenter alla view `LoginView`. Questa classe si occupa di gestire la logica per il login nel sistema da parte di un utente.

Relazione d'uso con altri moduli

- La classe è utilizzata da:

  `it.hourglass.myTalk.client.presenter.BaseViewPresenter`

- La classe utilizza:

  `it.hourglass.myTalk.client.shared.Profile`

  `it.hourglass.myTalk.client.RPCService.RPCService`

  `it.hourglass.myTalk.client.RPCService.RPCServiceAsync`

  `it.hourglass.myTalk.client.communication.WSConnection`

  `it.hourglass.myTalk.client.event.LoginRequestEvent`

  `it.hourglass.myTalk.client.presenter.WidgetPresenter`

  `it.hourglass.myTalk.client.presenter.display.LoginDisplay`

  `com.google.gwt.event.dom.client.ClickHandler`

  `com.google.gwt.event.dom.client.ClickEvent`

  `com.google.gwt.event.dom.client.HasClickHandlers`

  `com.google.gwt.event.shared.HandlerManager`

  `com.google.gwt.user.client.rpc.AsyncCallback`

  `com.google.gwt.user.client.ui.AbsolutePanel`

  `com.google.gwt.user.client.ui.HasValue`

- La classe implementa l'interfaccia :

  `it.hourglass.myTalk.client.presenter.WidgetPresenter`

Attributi

- `HandlerManager eventBus`
  Riferimento all'eventBus presenter in `client.appcontroller.AppController`. Esso sarà utilizzato per lanciare eventi.

- `LoginDisplay display`
  Riferimento alla view `LoginView` di cui il presenter gestisce la logica. Essa è di tipo `LoginDisplay` cioè l'interfaccia che `view.LoginView` implementa.

Metodi

+ `LoginPresenter(WSConnection wsconnection, HandlerManager eventBus, LoginDisplay display)`
  Costruttore della classe. Si occupa di inizializzare i campi dati e di collegare la logica ai pulsanti della view corrispettiva (`view.LoginView`) il cui riferimento è contenuto in `display` tramite l'invocazione del metodo `bind()`.

+ `void bind()`
  Questo metodo si occupa di aggiungere la logica al pulsante di richiesta di autenticazione della view `LoginView` il cui riferimento è contenuto in `display`.Questo avviene aggiungendo un `ClickHandler` a tale bottone recuperandolo tramite le funzioni definite in `LoginDisplay`. Quando un utente preme tale bottone il sistema dovrà estrarre i dati di accesso inseriti e invocare il metodo `doLoginControl()` il quale si occuperà del controllo dei dati inseriti.

+ `void doLoginControl(final String username,final String password)`
  Questo metodo si occupa di controllare i dati di accesso inseriti dall'utente. In esso si definisce l'AsyncCallback<Boolean> necessaria all'invocazione del metodo `checkLogin()` della RPC. Se la RPC ha successo e i dati inseriti sono validi il metodo si occuperà di aggiornare l'identificativo dell'utente corrente (tramite l'invocazione del metodo `setUniqueId()` della classe `client.shared.User`) e lancerà un evento di tipo `LoginRequestEvent` tramite `eventBus`. Se i dati inseriti non sono corretti invece il metodo si occuperà di segnalarlo all'utente.

+ `AbsolutePanel getView()`
  Questo metodo restituisce il pannello contente la `FriendContactView` che il presenter gestisce. Questo avviene richiedendo il metodo `getView()` definito nell'interfaccia `LoginDisplay` tramite `display`.

2.7.17 client.presenter.MessagePresenter - Classe

Funzione

Questa classe funge da presenter alla view `MessageView`. Questa classe si occupa di gestire la logica per l'estrazione e la visualizzazione dei messaggi inviati a un utente.

Relazione d'uso con altri moduli

- La classe è utilizzata da:

  `it.hourglass.myTalk.client.presenter.BaseViewPresenter`

- La classe utilizza:

  `it.hourglass.myTalk.client.shared.Message`

  `it.hourglass.myTalk.client.shared.Profile`

  `it.hourglass.myTalk.client.view.TextMessageView`

  `it.hourglass.myTalk.client.presenter.display.MessageDisplay`

  `java.util.ArrayList`

  `java.util.List`

  `com.google.gwt.user.client.ui.AbsolutePanel`

- La classe implementa l'interfaccia :

  `it.hourglass.myTalk.client.presenter.WidgetPresenter`

Attributi

- `MessageDisplay display`
  Riferimento alla view `view.MessageView` di cui il presenter gestisce la logica. Essa è di tipo `MessageDisplay` cioè l'interfaccia che `view.MessageView` implementa.

- `List<TextMessagePresenter> textMessage`
  Questa lista contiene i riferimenti ad i `TextMessagePresenter` i quali vengono creati (e aggiunti alla lista) per ogni messaggio che l'utente ha ricevuto.

Metodi

+ `MessagePresenter(MessageDisplay display)`
  Costruttore della classe. Si occupa di inizializzare il campo dato `display` contenente la view e di estrarre i messaggi dal profilo tramite il metodo `client.shared.Profile.getMessageList()`. Una volta che i messaggi sono stati estratti per ognuno il costruttore si occupa di creare un istanza di `TextMessagePresenter` e di aggiungere tale istanza alla lista `textMessage`.
  Infine questo costruttore invocherà il metodo `addMessageView()` il quale si occupa di aggiungere alla view (il cui riferimento è contenuto in `display`) la view `TextMessageView` contenuta in ogni `TextMessagePresenter` presente nella lista `textMessage`.

- `void addMessageView()`
  Questo metodo si occupa di estrarre la view `view.TextMessageView` contenuta in ogni `TextMessagePresenter` presente nella lista `textMessage` e di aggiungerla alla view padre `view.MessageView` il cui riferimento è contenuto in `- MessageDisplay display`.

+ `AbsolutePanel getView()`
  Questo metodo restituisce il pannello contente la `MessageView` che il presenter gestisce. Questo avviene richiedendo il metodo `getView()` definito nell'interfaccia `MessageDisplay` tramite `display`.

2.7.18 client.presenter.TextMessagePresenter - Classe

Funzione

Questa classe funge da presenter alla view `view.TextMessageView`.

Relazione d'uso con altri moduli

- La classe è utilizzata da:

  `it.hourglass.myTalk.client.presenter.MessagePresenter`

- La classe utilizza:

  `it.hourglass.myTalk.client.shared.Profile`

  `it.hourglass.myTalk.client.presenter.display.TextMessageDisplay`

- La classe implementa l'interfaccia :

  `it.hourglass.myTalk.client.presenter.WidgetPresenter`

Attributi

- `Message message`
  Riferimento al messaggio che il presenter si occupa di gestire.

- `TextMessageDisplay display`
  Riferimento alla view `view.TextMessageView` di cui il presenter gestisce la logica. Essa è di tipo `TextMessageDisplay` cioè l'interfaccia che `view.TextMessageView` implementa.

Metodi

+ `TextMessagePresenter(TextMessageDisplay display, Message message)`
  Costruttore della classe. Si occupa di inizializzare i campi dati e di collegare la logica del pulsante per eliminare un messaggio della view `view.TextMessageView` il cui riferimento è contenuto in `display` tramite l'invocazione del metodo `bind()`.

+ `void bind()`
  Questo metodo si occupa di aggiungere la logica al pulsante di richiesta di eliminazione di un messaggio dalla lista dei messaggi. Questo avviene aggiungendo un `ClickHandler` a tale bottone recuperandolo tramite le funzioni definite in `TextMessageDisplay`. Quando un utente preme tale bottone il sistema dovrà invocare il metodo `deleteMessage()` il quale si occuperà della richiesta.

- `void deleteMessage()`
  Questo metodo si occupa di eliminare un messaggio dalla lista dei messaggi locale e richiedere l'eliminazione dello stesso messaggio dal database. In esso si definisce l'AsyncCallback<String> necessaria all'invocazione del metodo `checkLogin()` della RPC. Se la RPC ha successo si eliminerà il messaggio dalla lista messaggi locale (contenuta nella variabile `messagelist` della classe `client.shared.Profile`), dal database e infine viene eliminata la view `TextMessageView` contenuta nella `view.MessageView`.

`+ AbsolutePanel getView()`

Questo metodo restituisce il pannello contente la `TextMessageView` che il presenter gestisce. Questo avviene richiedendo il metodo `getView()` definito nell'interfaccia `TextMessageDisplay` tramite `display`.

2.7.19 client.presenter.PasswordRecoverPresenter - Classe

Funzione

Questa classe funge da presenter alla classe `client.view.PasswordRecoverView` e permette di aver accesso e di controllare i suoi elementi. La sua funzionalità consiste di offrire la possibilità di recuperare la password all'Utente dell'applicativo, mediante successivi passi che includono l'inserimento dell'indirizzo email, del codice segreto spedito ad esso e l'inserimento di una nuova password.

Relazione d'uso con altri moduli

- La classe è utilizzata da:

  `it.hourglass.myTalk.client.presenter.BaseViewPresenter`

- La classe utilizza:

  `it.hourglass.myTalk.client.communication.WSConnection`

  `it.hourglass.myTalk.client.rpcservice.RPCService`

  `it.hourglass.myTalk.client.rpcservice.RPCServiceAsync`

  `it.hourglass.myTalk.client.event.HomePageRequestEvent`

  `it.hourglass.myTalk.client.event.LoginRequestEvent`

  `it.hourglass.myTalk.client.presenter.WidgetPresenter`

  `it.hourglass.myTalk.client.presenter.display.PasswordRecoverDisplay`

Attributi

- `HandlerManager eventBus`
  Riferimento a una variabile che avrà la funzione di ricevitore di eventi.
- `PasswordRecoverDisplay display`
  Variabile che si riferisce all'oggetto di tipo `client.view.PasswordRecoverView` che costituisce la view per il recupero password.
- `RPCServiceAsync RPCall`
  Proxy per la RPC.
- `String idUtente`
  Stringa contenente l'identificativo univoco dell'utente.
- `ValuesCheck VC`
  Oggetto con la funzione di verificare la correttezza dei campi inseriti.

Metodi

+ `PasswordRecoverPresenter(WSConnection wsconnection, HandlerManager eventBus, PasswordRecoverDisplay display)`
  Costruttore della classe. Inizializza i campi.

+ `void bind()`
  Metodo che si occupa di assegnare al click di un bottone della `client.view.PasswordRecoverView` un metodo della classe `PasswordRecoverPresenter`. Un bottone invocherà il metodo per verificare la correttezza dell'email, un altro invocherà il metodo per recuperare il codice di sicurezza inserito e un altro ancora per confermare la validità della password di conferma.

+ `void doValidation(String email)`
  Metodo che riceve come parametro una email valida e si occupa di invocare il metodo `setValidation()` dell'RPC per verificare che la email sia salvata nel database: in caso contrario viene segnalato un'errore a video. Se la email corrisponde viene cambiata la view e viene salvato l'identificativo univoco dell'utente a cui è associata la email ritornato dal metodo `setValidation()` dell'RPC.

+ `void doRequestRecover()`
  Metodo che controlla se la email inserita per il cambio password sia valida. In caso contrario si segnala a video un errore. Se la email risulta valida viene passata come parametro al metodo `doValidation()`.

+ `void doCheckValidation(String uniqueId, String validation)`
  Metodo che controlla se il codice inserito corrisponda a quello realmente generato dal server della piattaforma **MyTalk** per consentire il cambio password. Il tutto avviene mediante la chiamata al metodo `checkValidation()` dell'RPC e passando come parametri l'identificativo univoco dell'utente (in modo da rendere possibili le *query* nel *database*) e il codice di sicurezza. In caso di riscontro positivo la view verrà cambiata altrimenti comparirà a video una stringa che segnala i dettagli sull'impossibilità di effettuare l'operazione richiesta.

+ `void doContinueRequest()`
  Metodo che recupera il codice di sicurezza inserito e lo passa come parametro al metodo `doCheckValidation()`.

+ `void doChangePassword(String uniqueId, String password)`
  Metodo che si occuppa di invocare il metodo `setPassword()` dell'RPC per cambiare la password e per cambiare la view passando a quella principale (home).

+ `void doSaveNewPassword()`
  Recupera dalla terza view la nuova password e la sua conferma. Inizialmente controlla che la nuova password sia valida assieme alla sua conferma altrimenti ne segnala a video la causa dell'errore. Se è valida viene invocato il metodo `doChangePassword()` che si occuperà di invocare i metodi appropriati per il cambio password.

+ `AbsolutePanel getView()`
  Metodo che ritorna la view dell'oggetto `display`.

2.7.20 client.presenter.PopupMessagePresenter - Classe

Funzione

Questa classe funge da presenter alla classe `client.view.PopupMessageView` e permette di aver accesso e di controllare i suoi elementi. Nello specifico, permette la creazione e l'invio di un nuovo messaggio ad un Utente amico.

Relazione d'uso con altri moduli

- La classe è utilizzata da:

  `it.hourglass.myTalk.client.presenter.UserListPresenter`

- La classe utilizza:

  `it.hourglass.myTalk.client.communication.WSConnection`

  `it.hourglass.myTalk.client.rpcservice.RPCService`

  `it.hourglass.myTalk.client.rpcservice.RPCServiceAsync`

  `it.hourglass.myTalk.client.shared.Message`

  `it.hourglass.myTalk.client.shared.Profile`

  `it.hourglass.myTalk.client.view.PopupInformation`

  `it.hourglass.myTalk.client.presenter.display.PopupMessageDisplay`

Attributi

- `String recipient`
  Stringa contenente il nome del destinatario.
- `String sender`
  Stringa contenente il nome del mittente.
- `PopupMessageDisplay display`
  Variabile che si riferisce all'oggetto di tipo `client.view.PopupMessageView` che costituisce la view che consente la creazione del messaggio.

Metodi

+ `PopupMessagePresenter(String sender,String receiver, PopupMessageDisplay display)`
  Costruttore della classe. Inizializza le variabili.
+ `void bind()`
  Aggiunge ai relativi bottoni l'evento di invio del messaggio o per chiudere la finestra del messaggio.
- `static RPCServiceAsync getService()`
  Metodo che ritorna una variabile di tipo `RPCServiceAsync` per la chiamata di procedura remota.

`+ void sendMessage(final Message mess)`

Metodo che perfeziona l'invio del messaggio vero e proprio (coincidente con l'inserimento della voce nel *database*) utilizzando un'apposita RPC. Al ritorno da quest'ultima viene notificato il *server WS* così che a sua volta possa informare il destinatario dell'arrivo di un messaggio.

2.7.21 client.presenter.RegistrationPresenter - Classe

Funzione

Classe che si occupa del controllo e dell'utilizzo dei dati inseriti dall'utente durante la registrazione. Nel momento in cui l'utente ha perfezionato la registrazione, tutti i dati inseriti vengono controllati. Se non vi sono errori viene perfezionata l'operazione di registrazione all'applicativo. In caso contrario gli errori vengono segnalati ed è possibile reiterarla.

Relazione d'uso con altri moduli

- La classe è utilizzata da:

  `it.hourglass.myTalk.client.presenter.BaseViewPresenter`

  `it.hourglass.myTalk.client.view.RegistrationView`

- La classe utilizza:

  `it.hourglass.myTalk.client.RPCService.RPCService`

  `it.hourglass.myTalk.client.RPCService.RPCServiceAsync`

  `it.hourglass.myTalk.client.shared.User`

  `it.hourglass.myTalk.client.presenter.display.RegistrationDisplay`

- La classe implementa:

  `it.hourglass.myTalk.client.presenter.WidgetPresenter`

Attributi

- `boolean allChecked`
  Booleano a cui viene assegnato valore `true` se sono stati inseriti tutti i campi e sono corretti, `false` altrimenti.
- `RegistrationDisplay display`
  Variabile che si riferisce all'oggetto di tipo `RegistrationView` che costituisce la view per la registrazione.
- `ValuesCheck VC`
  Oggetto utilizzato per verificare la correttezza dei campi inseriti.
- `String uniqueId`
  Stringa contenente l'id inserito dall'utente.
- `String email`
  Stringa contenente l'email inserita dall'utente.
- `String password`
  Stringa contenente la password inserita dall'utente.
- `String rePassword`
  Stringa contenente la conferma della password inserita dall'utente.

- `String name`
  Stringa contenente il nome dell'utente, da esso inserita in fase di registrazione.
- `String lastName`
  Stringa contenente il cognome dell'utente, da esso inserita in fase di registrazione.
- `String birthday`
  Stringa contenente il giorno della data di nascita dell'utente, da esso inserita in fase di registrazione.
- `String birthmonth`
  Stringa contenente il mese della data di nascita dell'utente, da esso inserita in fase di registrazione.
- `String birthyear`
  Stringa contenente l'anno della data di nascita dell'utente, da esso inserita in fase di registrazione.
- `Date birthdate`
  Stringa contenente la data di nascita dell'utente composta dal giorno, mese e anno che l'utente ha inserito nelle
- `char gender`
  Carattere che indica il sesso dell'utente.

Metodi

+ `RegistrationPresenter(RegistrationDisplay display)`
  Costruttore della classe. Riceve in parametro un oggetto di tipo display che costituisce di fatto la view attraverso la quale l'utente può inserire i dati.
+ `AbsolutePanel getView()`
  Metodo che ritorna la view dell'oggetto display.
+ `void bind()`
  Metodo che associa un evento al click del tasto Registrati. Alla pressione del tasto Registrati vengono ricavati tutti i dati inseriti dall'utente e realizzati i successivi controlli.
+ `void checkValues()`
  Metodo che verifica se i dati inseriti dall'utente per la registrazione sono stati inseriti e sono corretti.
+ `RPCServiceAsync getService()`
  Metodo che ritorna il riferimento all'RPC.
+ `void register(User user)`
  Metodo che effettua, dopo che è stata verificata la correttezza dei dati, la registrazione vera e propria passando i dati inseriti al server.
- `void checkValidation()`
  Metodo che controlla la correttezza del codice segreto inserito per la registrazione.

In caso affermativo si avviserà l'utente che può usare tutte le funzionalità extra di **MyTalk**.

2.7.22 client.presenter.ValuesCheck - Classe

Funzione

Questa classe offre metodi per il controllo della conformità ai pattern desiderati di stringhe in parametro.

Relazione d'uso con altri moduli

- La classe è utilizzata da:

  `it.hourglass.myTalk.client.presenter.RegistrationPresenter`

  `it.hourglass.myTalk.client.presenter.PasswordRecoverPresenter`

Attributi

- `String USERNAME_PATTERN`
  Stringa contenente il pattern da ricercare all'interno dell'username per verificare se è corretto.
- `String EMAIL_PATTERN`
  Stringa contenente il pattern da ricercare all'interno dell'email per verificare se è corretta.
- `String PW_PATTERN`
  Stringa contenente il pattern da ricercare all'interno della password per verificare se è corretta.

Metodi

+ `String checkUniqueId(String uniqueId)`
  Controlla che la stringa passata in parametro sia conforme al pattern corrispondente ai criteri scelti per l'username. Il pattern associato alla stringa da controllare impone che: l'username sia alfanumerico; la lunghezza non sia inferiore a 8 caratteri o maggiore di 32; non siano presenti spazi ma solo underscore.
+ `String checkPasswords(String password, String rePassword)`
  Controlla che le due password inserite siano conformi al pattern scelto e che siano uguali tra loro. Il pattern scelto per l'inserimento delle password impone che: le password siano alfanumeriche, senza spazi o segni d'interpunzione; la lunghezza delle password sia non inferiore a 8 caratteri né maggiore di 32.
+ `String checkName(String name)`
  Controlla che la stringa inserita contenente il nome dell'utente sia non vuota e che sia maggiore di 2 caratteri ma minore di 32.
+ `String checkLastName(String lastName)`
  Controlla che la stringa inserita contenente il cognome dell'utente sia non vuota e sia maggiore di 2 caratteri ma minore di 32.
+ `String checkEmail(String email)`
  Controlla che l'indirizzo email inserito sia conforme al pattern stabilito.

`+ Date checkDate(String birthyear, String birthmonth, String birthday)`
Controlla che le stringhe inserite compongano una data corrispondente al formato `yyyy-MM-dd` e valida.

`+ String validate(final String exp, String word)`
Metodo per il controllo della conformità di una stringa al pattern inserito.

2.8 Package it.hourglass.myTalk.client.presenter.profilemanagement

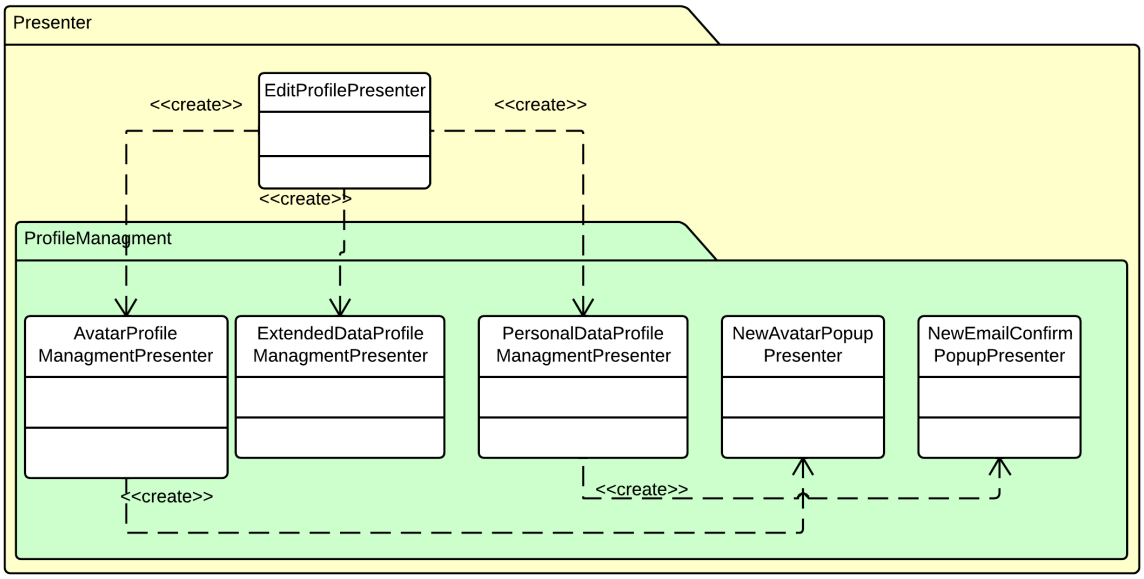

Figura 3: Package it.hourglass.myTalk.client.presenter.profilemanagement

Package che contiene le classi utilizzate dalla classe `it.hourglass.myTalk.client.presenter.EditProfilePresenter`.

2.8.1 client.presenter.profilemanagement.AvatarProfileManagementPresenter - Classe

Funzione

Questa classe funge da presenter alla view `view.AvatarProfileManagementView` (sottoview di `view.EditProfileView`). Questa classe si occupa della gestione della modifica dell'avatar del profilo di un utente.

Relazione d'uso con altri moduli

- La classe è utilizzata da:

  `it.hourglass.myTalk.client.presenter.profilemanagement.EditProfilePresenter`

- La classe utilizza:

  `it.hourglass.myTalk.client.presenter.WidgetPresenter`

  `it.hourglass.myTalk.client.shared.Profile`

  `it.hourglass.myTalk.client.shared.User`

  `it.hourglass.myTalk.client.view.ProfileManagement.NewAvatarPopupView`

  `it.hourglass.myTalk.client.presenter.display.AvatarProfileManagementDisplay`

  `com.google.gwt.event.dom.client.ClickEvent`

  `com.google.gwt.event.dom.client.ClickHandler`

  `com.google.gwt.event.dom.client.HasClickHandlers`

  `com.google.gwt.event.shared.HandlerManager`

  `com.google.gwt.user.client.ui.AbsolutePanel`

  `com.google.gwt.user.client.ui.Button`

- La classe implementa l'interfaccia :

  `it.hourglass.myTalk.client.presenter.WidgetPresenter`

Attributi

- `AvatarProfileManagementDisplay display`
  Questa variabile fa riferimento alla AvatarProfileManagementView di cui il presenter gestisce la logica. Essa è di tipo `AvatarProfileManagementDisplay` cioè l'interfaccia che
  `view.ProfileManagement.AvatarProfileManagementView` implementa.

Metodi

+ `AvatarProfileManagementPresenter(AvatarProfileManagementDisplay display)`
  Costruttore della classe. Si occupa di inizializzare `display` e di collegare il pulsante della view corrispettiva (il cui riferimento è contenuto in `display`) per la richiesta di cambio dell'avatar tramite l'invocazione del metodo `bind()`.

- `void bind()`
  Questo metodo si occupa di collegare la logica del bottone di richiesta di cambio avatar contenuto nella `view.AvatarProfileManagementView` il cui riferimento è contenuto in `display`. Questo avviene aggiungendo un `ClickHandler` al bottone recuperandolo tramite la funzione definita in `AvatarProfileManagementDisplay`. Alla pressione di tale tasto si creerà un'istanza della classe `view.NewAvatarPopupPresenter`.

+ `AbsolutePanel getView()`
  Questo metodo restituisce il pannello contente la `view.NewAvatarPopupPresenter` che il presenter gestisce. Questo avviene richiedendo il metodo `getView()` definito nell'interfaccia `AvatarProfileManagementDisplay` tramite `display`.

2.8.2 client.presenter.profilemanagement.NewAvatarPopupPresenter - Classe

Funzione

Questa classe funge da presenter alla view `view.profilemanagement.NewAvatarPopupPresenter` (sottoview di `view.EditProfileView`). Questa classe si occupa della gestione della modifica dell'avatar del profilo di un utente.

Relazione d'uso con altri moduli

- La classe è utilizzata da:

  `it.hourglass.myTalk.client.presenter.profilemanagement.EditProfilePresenter`

- La classe utilizza:

  `it.hourglass.myTalk.client.profile.Profile`

  `it.hourglass.myTalk.client.shared.User`

  `it.hourglass.myTalk.client.presenter.display.NewAvatarPopupDisplay`

  `com.google.gwt.event.dom.client.ClickEvent`

  `com.google.gwt.event.dom.client.ClickHandler`

  `com.google.gwt.event.dom.client.HasClickHandlers`

  `com.google.gwt.event.shared.HandlerManager`

  `com.google.gwt.user.client.ui.HasValue`

- La classe implementa l'interfaccia:

  `it.hourglass.myTalk.client.presenter.WidgetPresenter`

Attributi

- `HandlerManager eventBus`
  Riferimento all'eventBus presente in `client.appcontroller.AppController`. Esso sarà utilizzato per lanciare eventi.

- `NewAvatarPopupDisplay display`
  Questa variabile fa riferimento alla NewAvatarPopupView. Essa è di tipo `NewAvatarPopupDisplay` cioè l'interfaccia che `view.profilemanagement.NewAvatarPopupView` implementa.

Metodi

+ `NewAvatarPopupPresenter(NewAvatarPopupDisplay display)`
  Costruttore della classe. Si occupa di inizializzare `display` e di collegare il pulsante della view corrispettiva (il cui riferimento è contenuto in `display`) per la richiesta di conferma del nuovo url dell'avatar tramite l'invocazione del metodo `bind()`.

- `void bind()`
  Questo metodo si occupa di collegare la logica dei bottoni della `view.UserListView` il cui riferimento è contenuto in `display`. Questo avviene aggiungendo un `ClickHandler`

ad ogni bottone recuperandolo tramite la funzione definita in `NewAvatarPopupDisplay`. Se viene premuto il bottone per la chiusura del popup la modifica viene annullata. Se viene premuto il bottone di conferma della modifica dell'url il nuovo url viene estratto tramite i metodi definiti in `NewAvatarPopupDisplay` e successivamente viene salvata la modifica. Questo avviene utilizzando i metodi `changeLocalValues()` per il salvataggio del nuovo url nel profilo locale e `client.shared.Profile.` definito nella classe `client.shared.Profile`.

- `void changeLocalValues()`
  Questo metodo si occupa di salvare la modifica del nuovo ulr dell'avatar associato al profilo in locale locale tramite l'utilizzo dei metodi definiti in `it.hourglass.myTalk.client.shared.User`.

2.8.3 client.presenter.profilemanagement.PersonalDataProfileManagementPresenter - Classe

Funzione

Questa classe funge da presenter alla view `view.PersonalDataProfileManagementView` (sottoview di `view.EditProfileView`). Questa classe si occupa della gestione della modifica dei dati personali di un utente.

Relazione d'uso con altri moduli

- La classe è utilizzata da:

  `it.hourglass.myTalk.client.presenter.profilemanagement.EditProfilePresenter`

- La classe utilizza:

  `it.hourglass.myTalk.client.RPCService.RPCService`

  `it.hourglass.myTalk.client.RPCService.RPCServiceAsync`

  `it.hourglass.myTalk.client.presenter.ValuesCheck`

  `it.hourglass.myTalk.client.presenter.WidgetPresenter`

  `it.hourglass.myTalk.client.shared.Profile`

  `it.hourglass.myTalk.client.shared.User`

  `it.hourglass.myTalk.client.view.ProfileManagement.NewEmailConfirmPopupView`

  `it.hourglass.myTalk.client.presenter.display.PersonalDataProfileManagementDisplay`

  `it.google.gwt.event.dom.client.ClickEvent`

  `it.google.gwt.event.dom.client.ClickHandler`

  `it.google.gwt.event.dom.client.HasClickHandlers`

  `it.google.gwt.user.client.rpc.AsyncCallback`

  `it.google.gwt.user.client.ui.AbsolutePanel`

  `it.google.gwt.user.client.ui.HasValue`

- La classe implementa l'interfaccia :

  `it.hourglass.myTalk.client.presenter.WidgetPresenter`

Attributi

- `User user`
  Riferimento all'user corrente.

- `PersonalDataProfileManagementDisplay display`
  Questa variabile fa riferimento alla `view.PersonalDataProfileManagementView.PersonalDataProfileManagementView` di cui il presenter gestisce la logica. Essa è di tipo `PersonalDataProfileManagementDisplay` cioè l'interfaccia che `view.ProfileManagment.PersonalDataProfileManagementPresenter` implementa.

- `String email`
  Contiene l'email dell'utente recuperato dalla view dopo un cambiamento.
- `String password`
  Contiene la password dell'utente recuperata dalla view dopo un cambiamento.
- `String rePassword`
  Contiene la password di conferma che viene recuperata dalla view dopo un cambiamento.
- `String name`
  Contiene il nome dell'utente recuperato dalla view dopo un cambiamento.
- `String lastName`
  Contiene il cognome dell'utente recuperato dalla view dopo un cambiamento.
- `String birthday`
  Contiene il giorno di nascita dell'utente recuperato dalla view dopo un cambiamento.
- `String birthmonth`
  Contiene il mese di nascita dell'utente recuperato dalla view dopo un cambiamento.
- `String birthyear`
  Contiene l'anno di nascita dell'utente recuperato dalla view dopo un cambiamento.
- `char gender`
  Contiene il sesso dell'utente recuperato dalla view dopo un cambiamento.

Metodi

- `+ PersonalDataProfileManagementPresenter(PersonalDataProfileManagementDisplay display)`
  Costruttore della classe. Si occupa di inizializzare `display` e di collegare i pulsanti della view corrispettiva (il cui riferimento è contenuto in `display`) per la richiesta di modifica e di conferma di salvataggio tramite l'invocazione del metodo `bind()`.
- `- void bind()`
  Questo metodo si occupa di collegare la logica dei bottoni della `view.profilemanagement.PersonalDataProfileManagementView` il cui riferimento è contenuto in `display`. Questo avviene aggiungendo un `ClickHandler` ad ogni bottone recuperandolo tramite la funzione definita in `PersonalDataProfileManagementDisplay`. La pressione del bottone di richiesta di modifica deve rendere i campi dati della view modificabili. Invece la pressione del bottone di conferma dei dati richiede di estrarre i dati dalla view (tramite l'utilizzo dei metodi esposti dall'interfaccia `PersonalDataProfileManagementDisplay`), il controllo degli stessi utilizzando il metodo `checkValues` e, nel caso in cui i dati sono validi, richiede il salvataggio degli stessi sia nel profilo locale che nel database. Questo avviene uti-

lizzando i metodi changeLocalValues() per il salvataggio dei dati nel profilo locale e del metodo client.shared.Profile.saveUser() (definito nella classe client.shared.Profile). Se l'email associata ad un utente è stata cambiata è necessario richiedere la conferma tramite l'invio di un codice di verifica. Questo viene fatto attraverso il metodo sendValidation().

- boolean checkValues()
  Questa classe si occupa di controllare i dati estatti dalla view utilizzando i metodi definiti nella classe ValuesCheck. Se i dati non sono corretti il metodo ritornerà false altrimenti true.
- void changeLocalValues()
  Questo metodo si occupa di salvare le modifiche richieste dall'utente nel profilo locale tramite l'utilizzo dei metodi definiti in it.hourglass.myTalk.client.shared.User.
- void sendValidation(final String oldEmail, final String newEmail)
  Questo metodo si occupa di inviare il codice di verifica alla vecchia email dell'utente che ha richiesto la modifica del profilo. Il metodo definisce la chiamata asincrona AsyncCallback<String> necessaria alla richiesta di invocazione del metodo setValidation() esposto dall'interfaccia client.rpcservice.RPCService. Se la RPC ha successo si istanzierà un'istanza di NewEmailConfirmPopupPresenter il quale si occuperà della verifica del codice inviato.
+ AbsolutePanel getView()
  Questo metodo restituisce il pannello contente la view.profilemanagement.PersonalDataProfileManagementView che il presenter gestisce. Questo avviene richiedendo il metodo getView() definito nell'interfaccia PersonalDataProfileManagementDisplay tramite display.

2.8.4 client.presenter.profilemanagement.NewEmailConfirmPopupPresenter - Classe

Funzione

Questa classe funge da presenter alla view `NewEmailConfirmPopupView` e si occupa di verificare il codice di validazione per il cambio email di un utente.

Relazione d'uso con altri moduli

- La classe è utilizzata da:

  `it.hourglass.myTalk.client.presenter.profilemanagement.PersonalDataProfileManagem`

- La classe utilizza:

  `it.hourglass.myTalk.client.RPCService.RPCService`

  `it.hourglass.myTalk.client.RPCService.RPCServiceAsync`

  `it.hourglass.myTalk.client.shared.Message`

  `it.hourglass.myTalk.client.shared.Profile`

  `it.hourglass.myTalk.client.presenter.display.NewEmailConfirmPopupDisplay`

  `com.google.gwt.event.dom.client.ClickEvent`

  `com.google.gwt.event.dom.client.ClickHandler`

  `com.google.gwt.user.client.rpc.AsyncCallback`

  `com.google.gwt.user.client.ui.AbsolutePanel`

  `com.google.gwt.user.client.ui.Button`

- La classe implementa l'interfaccia :

  `it.hourglass.myTalk.client.presenter.WidgetPresenter`

Attributi

- `NewEmailConfirmPopupDisplay display`
  Questa variabile fa riferimento alla `view.profilemanagement.NewEmailConfirmPopupView` di cui il presenter gestisce la logica. Essa è di tipo `NewEmailConfirmPopupDisplay` cioè l'interfaccia che `view.ProfileManagement.NewEmailConfirmPopupPresenter` implementa.
- `String email`
  Contiene l'email da validare.
- `PersonalDataProfileManagementDisplay parentDisplay`
  Questo riferimento serve per richiedere alla pagine di gestione del profilo `view.EditProfileView` di visualizzare i dati nuovi una volta che è stata confermata la nuova email.
- `boolean encryptionNeeded`
  Questo metodo serve per richiedere la criptazione della password in caso si sia richiesto la modifica della stessa prima della conferma del cambio di email.

Metodi

+ `NewEmailConfirmPopupPresenter(String email, PersonalDataProfileManagementDisplay parentDisplay, NewEmailConfirmPopupDisplay display,boolean encryption)`
  Costruttore della classe. Si occupa di inizializzare i campi dati e di collegare i pulsanti della view corrispettiva (il cui riferimento è contenuto in `display`) per la richiesta di conferma del codice di validazione attraverso il metodo `bind()`.

- `void bind()`
  Questo metodo si occupa di collegare la logica dei bottoni della `view.profilemanagement.NewEmailConfirmPopupViw` il cui riferimento è contenuto in `display`. Questo avviene aggiungendo un `ClickHandler` ad ogni bottone recuperandolo tramite la funzione definita in `NewEmailConfirmPopupDisplay`. Se viene premuto il bottone di conferma si dovrà recuperare il codice di validazione inserito (tramite le funzioni definite in `NewEmailConfirmPopupDisplay` e di richiedere la verifica di tale codice. Questo avviene definendo l'AsyncCallback<Boolean>, necessaria per l'invocazione del metodo `checkValidation` della RPC. Se l'RPC ha successo e se è stato confermato anche il codice segreto allora si utilizza il metodo `client.shared.Profile.saveUser(encryptionNeeded)` (nota encryptionNeeded è necessario per richiedere il crypting della password) per richiedere il salvataggio delle modifiche.

+ `AbsolutePanel getView()`
  Questo metodo restituisce il pannello contente la `NewAvatarPopupPresenter` che il presenter gestisce. Questo avviene richiedendo il metodo `getView()` definito nell'interfaccia `NewEmailConfirmPopupDisplay` tramite `display`.

2.8.5 client.presenter.profilemanagement.ExtendedDataProfileManagementPresenter - Classe

Funzione

Questa classe funge da presenter alla view `view.profilemanagement.NewEmailConfirmPopupView` (sottoview di `view.EditProfileView`). Questa classe si della gestione della modifica dei dati estesi di un utente.

Relazione d'uso con altri moduli

- La classe è utilizzata da:

  `it.hourglass.myTalk.client.presenter.profilemanagement.PersonalDataProfileManagem`

- La classe utilizza:

  `it.hourglass.myTalk.client.rpcservice.RPCService`

  `it.hourglass.myTalk.client.rpcservice.RPCServiceAsync`

  `com.google.gwt.user.client.rpc.AsyncCallback`

  `it.hourglass.myTalk.client.shared.Profile`

  `it.hourglass.myTalk.client.shared.User`

  `it.hourglass.myTalk.client.presenter.display.ExtendedDataProfileManagementDisplay`

  `com.google.gwt.event.dom.client.ClickEvent`

  `com.google.gwt.event.dom.client.ClickHandler`

  `com.google.gwt.event.dom.client.HasClickHandlers`

  `com.google.gwt.user.client.ui.HasValue`

Attributi

- `User user`
  Riferimento all'user corrente.
- `ExtendedDataProfileManagementDisplay display`
  Questa variabile fa riferimento alla `view.profilemanagement.ExtendedDataProfileManagementV` di cui il presenter gestisce la logica. Essa è di tipo `ExtendedDataProfileManagementDisplay` cioè l'interfaccia che `view.ProfileManagement.ExtendedDataProfileManagementPresenter` implementa.
- `String currentLocation`
  Contiene l'indirizzo dell'utente recuperato dalla view dopo un cambiamento.
- `String hometown`
  Contiene la città di origine dell'utente recuperata dalla view dopo un cambiamento.
- `String altEmail`
  Contiene l'email alternativa dell'utente recuperata dalla view dopo un cambiamento.

- `String description`
  Contiene la descrizione dell'utente recuperata dalla view dopo un cambiamento.
- `String inspirations`
  Contiene le inspirazioni dell'utente recuperate dalla view dopo un cambiamento.
- `String interests`
  Contiene gli interessi dell'utente recuperate dalla view dopo un cambiamento.

Metodi

+ `PersonalDataProfileManagementPresenter(ExtendedDataProfileManagementDisplay display)`
  Costruttore della classe. Si occupa di inizializzare `display` e di collegare i pulsanti della view corrispettiva (il cui riferimento è contenuto in `display`) per la richiesta di modifica e di conferma di salvataggio grazie all'invocazione del metodo `bind()`.
- `void bind()`
  Questo metodo si occupa di collegare la logica dei bottoni della `view.profilemanagement.ExtendedDataProfileManagementView` il cui riferimento è contenuto in `display`. Questo avviene aggiungendo un `ClickHandler` ad ogni bottone recuperandolo tramite la funzione definita in `ExtendedDataProfileManagementDisplay`. La pressione del bottone di richiesta di modifica deve rendere i campi dati della view modificabili. La pressione del bottone di conferma dei dati richiede di estrarre i dati dalla view (tramite l'utilizzo dei metodi esposti dall'interfaccia `ExtendedDataProfileManagementDisplay`), il controllo degli stessi utilizzando il metodo `checkValues()` e, nel caso in cui i dati sono validi, richiede il salvataggio degli stessi sia nel profilo locale che nel database. Questo avviene utilizzando il metodo `changeLocalValues()` per il salvataggio dei dati nel profilo locale e il metodo `client.shared.Profile.saveUser()` definito nella classe `client.shared.Profile`.
- `boolean checkValues()`
  Questa classe si occupa di controllare i dati estatti dalla view utilizzando i metodi definiti nella classe `it.hourglass.myTalk.client.presenter.ValuesCheck`. Se i dati non sono corretti il metodo ritornerà `false` altrimenti `true`.
- `void changeLocalValues()`
  Questo metodo si occupa di salvare le modifiche richieste dall'utente nel profilo locale tramite l'utilizzo dei metodi definiti in `it.hourglass.myTalk.client.shared.User`.
+ `AbsolutePanel getView()`
  Questo metodo restituisce il pannello contente la `PersonalDataProfileManagementView` che il presenter gestisce. Questo avviene richiedendo il metodo `getView()` definito nell'interfaccia `ExtendedDataProfileManagementDisplay` tramite `display`.

2.9 Package it.hourglass.myTalk.client.view

Questo package contiene la definizione di tutte le classi contenenti ogni widget grafico. La funzione di queste classi è solo di presentazione.

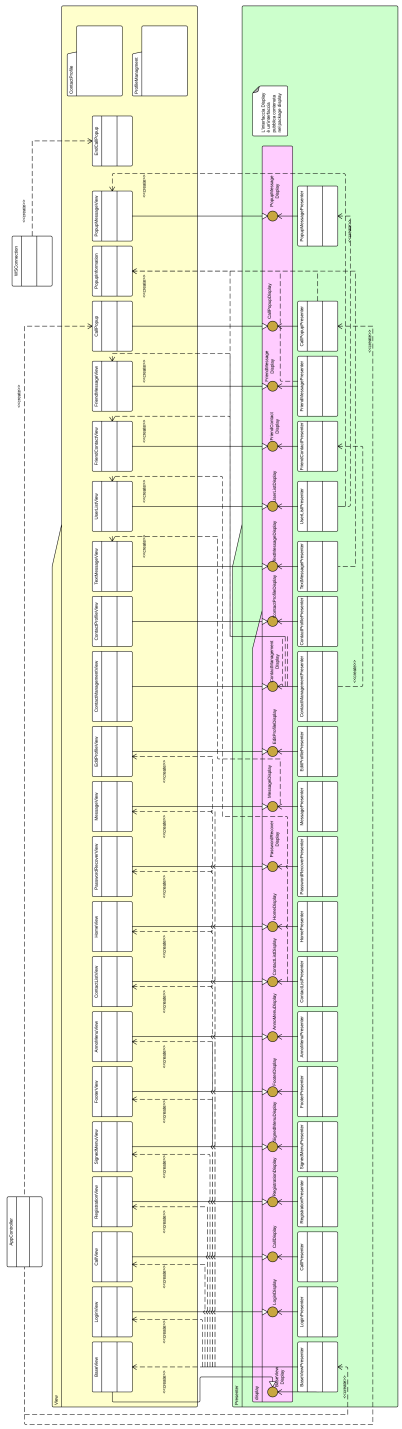

Figura 4: Package it.hourglass.myTalk.client.view

2.9.1 client.view.AnnoMenuView - Classe

Funzione

Questa classe rappresenta il widget menù degli utenti che non hanno effettuato il login e ha la funzione di permettere all'utente lo switch della view desiderata.

Relazione d'uso con altri moduli

- La classe è utilizzata da:

  `it.hourglass.myTalk.client.presenter.BaseViewPresenter`

- La classe utilizza:

  `it.hourglass.myTalk.client.presenter.AnnoMenuPresenter`

- La classe implementa:

  `it.hourglass.myTalk.client.presenter.display.AnnoMenuDisplay`

- La classe estende:

  `com.google.gwt.user.client.ui.Composite`

Attributi

- `Button HomeButton`
  Bottone per accedere alla homepage.
- `Button CallButton`
  Bottone per accedere alla pagina di chiamata.
- `Button LoginButton`
  Bottone per accedere alla pagina di login.
- `Button RegistrationButton`
  Bottone per accedere alla pagina di registrazione.
- `Button RecuperaPasswordButton`
  Bottone per accedere alla pagina di recupero password.
- `AbsolutePanel contenuto`
  Area del widget menù.

Metodi

+ `AnnoMenuView()`
  Costruttore della classe. Dispone le componenti nella view.
+ `HasClickHandlers getCallButton()`
  Metodo che restituisce il bottone `callButton`.
+ `HasClickHandlers getRegistrationButton()`
  Metodo che restituisce il bottone `registrationButton`.
+ `HasClickHandlers getLoginButton()`
  Metodo che restituisce il bottone `loginButton`.

+ `HasClickHandlers getRecuperaPasswordButton()`
  Metodo che restituisce il bottone `recuperaPasswordButton.`
+ `getHomeButton()`
  Metodo che restituisce il bottone `homeButton.`
+ `AbsolutePanel getView()`
  Metodo che restituisce il pannello `contenuto.`

2.9.2 client.view.BaseView - Classe

Funzione

Questa classe rappresenta la BaseView. Su questa classe verranno aggiunti i widget del menù, del footer e sarà possibile lo switch dei widget centrali.

Relazione d'uso con altri moduli

- La classe è utilizzata da:

  `it.hourglass.myTalk.client.presenter.BaseViewPresenter`

- La classe utilizza:

  `it.hourglass.myTalk.client.presenter.BaseViewPresenter`

  `it.hourglass.myTalk.client.presenter.WidgetPresenter`

  `it.hourglass.myTalk.client.view.FooterView`

- La classe implementa:

  `it.hourglass.myTalk.client.presenter.display.BaseViewDisplay`

- La classe estende:

  `com.google.gwt.user.client.ui.Composite`

Attributi

- `AbsolutePanel leftPanel`
  Pannello sinistro che conterrà al suo interno il pannello del menù.
- `AbsolutePanel bodyPanel`
  Pannello per il body.
- `AbsolutePanel menuPanel`
  Pannello per il menù.
- `RootPanel rootPanel`
  Pannello che contiene al suo interno il pannello sinistro (contenente il menù) e il pannello del body.

Metodi

+ `BaseView()`
  Costruttore della classe che dispone le componenti nella view.
+ `void switchBody(AbsolutePanel pannello)`
  Questo metodo si occupa di eseguire lo switch del body.
+ `void switchMenu(AbsolutePanel pannello)`
  Metodo che si occupa di eseguire lo switch del menu.
+ `void switchFooter(AbsolutePanel pannello)`
  Metodo che si occupa di eseguire lo switch del footer.
+ `void addContactList(AbsolutePanel panel)`
  Metodo per aggiungere al pannello sinistro la lista contatti.

2.9.3 client.view.CallPopup - Classe

Funzione

Questa classe rappresenta il popup di ricezione di una richiesta di chiamata.

Relazione d'uso con altri moduli

- La classe è utilizzata da:

  `it.hourglass.myTalk.client.appcontroller.AppController`

- La classe utilizza:

  `it.hourglass.myTalk.client.presenter.CallPopupPresenter`

- La classe implementa:

  `it.hourglass.myTalk.client.presenter.display.CallPopupDisplay`

Attributi

- `DialogBox popup`
  Box di dialogo per informare il destinatario che è in arrivo una chiamata. Contiene il nickname e l'avatar del chiamante accompagnati dai bottoni `accept` per accettare la chiamata e `refuse` per rifiutarla.
- `Button accept`
  Bottone per accettare la chiamata arrivata.
- `Button refuse`
  Bottone per rifiutare la chiamata arrivata.
- `String nome`
  Stringa contenente il nome del chiamante.

Metodi

+ `CallPopup(String nome)`
  Costruttore della classe. Si occupa di inizializzare i campi dati necessari e di disporre le componenti della popup.
+ `HasClickHandlers getAcceptButton()`
  Metodo che restituisce il bottone di accettazione della chiamata.
+ `HasClickHandlers getRefuseButton()`
  Metodo che restituisce il bottone di rifiuto della chiamata.
+ `close()`
  Metodo invocato per chiudere la popup.
+ `String getName()`
  Metodo invocato per ritornare il nome del chiamante.

2.9.4 client.view.CallView - Classe

Funzione

Questa classe rappresenta il widget per la chiamata.

Relazione d'uso con altri moduli

- La classe è utilizzata da:

  `it.hourglass.myTalk.client.presenter.BaseViewPresenter`

- La classe utilizza:

  `it.hourglass.myTalk.client.presenter.CallPresenter`

- La classe implementa:

  `it.hourglass.myTalk.client.presenter.display.CallDisplay`

Attributi

- `AbsolutePanel contenuto`
  Pannello contenente il widget CallView.
- `AbsolutePanel GuestVideoPanel`
  Pannello contenente l'attributo `VideoGuestPanel.`
- `AbsolutePanel LocalVideoPanel`
  Pannello contenente l'attributo `LocalVideo.`
- `AbsolutePanel callUser`
  Pannello che racchiude (se al momento non sono in corso chiamate) il textbox per inserire il nickname del chiamante e il bottone per avviare la conversazione. Se invece la chiamata è in corso questo pannello mostrerà le statistiche della chiamata, l'avviso di chi sto chiamando e il bottone per chiudere la chiamata.
- `AbsolutePanel statistics`
  Pannello contenente le statistiche.
- `AbsolutePanel ChatPanel`
  Pannello che contiene la variabile `ChatPanel`.
- `AbsolutePanel Chat`
  Pannello che contiene l'area di testo della chat.
- `AbsolutePanel VideoGuestPanel`
  Pannello che contiene la variabile `GuestVideo`.
- `Video GuestVideo`
  Contiene il riferimento al video dell'ospite.
- `Video LocalVideo`
  Contiene il riferimento al video locale.
- `TextBox textBoxInvia`
  Contiene il testo da inviare.

- `TextBox callUserTextBox`
  Contiene il nickname dell'utente che si sta chiamando.
- `TextBox callLenght`
  Contiene la durata della chiamata.
- `TextBox callByteD`
  Contiene il numero dei byte ricevuti.
- `TextBox callByteU`
  Contiene il numero dei byte inviati.
- `TextArea textAreaChat`
  TextArea della chat.
- `Button callButton`
  Bottone di chiamata.
- `Button sendButton`
  Bottone di invio messaggio della chat.
- `Button closeCall`
  Bottone per chiudere la chiamata.
- `ListBox streamRequest`
  Contiene una lista di opzioni su ciò che l'utente vuole condividere per la chiamata (Audio, Audio/Video, Audio/Video/Chat).
- `Label calee`
  Contiene l'indicazione che descrive con chi sono in chiamata.

Metodi

+ `CallView()`
  Costruttore della classe che inizializza il widget CallView disponendo le sue componenti.
+ `nascondi()`
  Metodo che si occupa di nascondere l'elemento video non locale e le statistiche di chiamata.
+ `show()`
  Metodo che si occupa di mostrare l'elemento video non locale e le statistiche della chiamata con sotto il bottone per chiuderla.
+ `HasClickHandlers getSendButton()`
  Metodo che ritorna il bottone di invio della chat.
+ `AbsolutePanel getView()`
  Metodo che ritorna il widget CallView.
+ `HasClickHandlers getCallButton()`
  Metodo che ritorna il bottone di chiamata.

+ `Video getLocalVideo()`
  Metodo che ritorna il riferimento al video locale.
+ `Video getGuestVideo()`
  Metodo che ritorna il riferimento al video dell'ospite.
+ `TextArea getChat()`
  Metodo che ritorna la textArea della chat.
+ `TextBox getByteD()`
  Metodo che ritorna la variabile `callByteD.`
+ `TextBox getByteU()`
  Metodo che ritorna la variabile `callByteU.`
+ `TextBox getLenght()`
  Metodo che ritorna la textArea per la statistica del tempo.
+ `TextBox getcallUserTextBox()`
  Metodo che ritorna il nome dell'utente da chiamare.
+ `TextBox getTextBoxInvia()`
  Metodo che ritorna il messaggio da inviare via chat.
+ `HasClickHandlers getCloseButton()`
  Metodo che ritorna il bottone della chiusura della chiamata.
+ `String getStreamRequest()`
  Metodo che ritorna una stringa contenente ciò che l'utente vuole condividere per la chiamata (Audio, Audio/Video, Audio/Video/Chat oppure solo Chat).
+ `Label getCalee()`
  Metodo che ritorna il nickname del chiamante.

2.9.5 client.view.ContactListView - Classe

Funzione

Questa classe rappresenta la view della lista contatti.

Relazione d'uso con altri moduli

- La classe è utilizzata da:

  `it.hourglass.myTalk.client.presenter.BaseViewPresenter`

- La classe utilizza:

  `it.hourglass.myTalk.client.presenter.ContactListPresenter`

- La classe implementa:

  `it.hourglass.myTalk.client.presenter.display.ContactListDisplay`

Attributi

- `AbsolutePanel contenent`
  Pannello contenente il widget ContactListView.
- `AbsolutePanel list`
  Pannello contenente la lista contatti.
- `AbsolutePanel groupNamePanel`
  Pannello contenente il pezzo di HTML che conterrà il nome del gruppo della lista contatti.
- `Button getContactManagmentPageButton`
  Bottone per accedere alla pagina di gestione dei contatti.
- `HTML groupName`
  Pezzo di HTML contenente il nome del gruppo della lista contatti.

Metodi

+ `ContactListView()`
  Costruttore della classe che inizializza il widget ContactListView disponendo le sue componenti.
+ `AbsolutePanel getView()`
  Metodo che ritorna il Widget ContactListView.
+ `void addUser(AbsolutePanel userView)`
  Metodo per aggiungere un utente nella lista.
+ `void removeContactList()`
  Metodo per rimuovere la lista contatti.
+ `HasClickHandlers getContactManagmentPage()`
  Metodo che ritorna il bottone per accedere alla gestione dei contatti.

2.9.6 client.view.ContactManagementView - Classe

Funzione

Questa classe rappresenta la view della gestione dei contatti. Sarà possibile aggiungere un amico, visualizzare la lista dei propri amici e vedere le richieste di amicizia ricevute.

Relazione d'uso con altri moduli

- La classe è utilizzata da:

  `it.hourglass.myTalk.client.presenter.BaseViewPresenter`

- La classe utilizza:

  `it.hourglass.myTalk.client.presenter.ContactManagementPresenter`

- La classe implementa:

  `it.hourglass.myTalk.client.presenter.display.ContactManagementDisplay`

Attributi

- `AbsolutePanel contactPanel`
  Pannello contenente la propria lista degli amici.
- `AbsolutePanel friendshipRequestPanel`
  Pannello contenente le richieste di amicizia ricevute.
- `AbsolutePanel contenent`
  Pannello contenente il widget ContactManagementView.
- `AbsolutePanel title`
  Pannello contenente il titolo della pagina.
- `Button friendshipRequestButton`
  Bottone per richiedere l'amicizia a un utente.
- `TextBox userRequest`
  Area di testo per scrivere il nickname dell'utente alla quale vogliamo inviarli una richiesta di amicizia.

Metodi

- `+ ContactManagementView()`
  Costruttore della classe che inizializza il widget ContactManagementView disponendo le sue componenti.
- `+ HasClickHandlers getFriendshipRequestButton()`
  Metodo che ritorna il bottone per la richiesta di amicizia.
- `+ AbsolutePanel getView()`
  Metodo che ritorna il widget ContactManagementView.
- `+ HasValue<String> getUserRequest()`
  Metodo che ritorna la variabile `userRequest`.

`+ void addFriend(AbsolutePanel panel)`
Metodo che aggiunge al relativo pannello un nuovo amico.

`+ void addFriendMessage(AbsolutePanel panel)`
Metodo che serve ad aggiungere una nuova richiesta di amicizia ricevuta alle altre.

2.9.7 client.view.ContactProfileView - Classe

Funzione

Questa classe rappresenta la view per la visualizzazione del profilo di un contatto.

Relazione d'uso con altri moduli

- La classe è utilizzata da:

  `it.hourglass.myTalk.client.presenter.BaseViewPresenter`

- La classe utilizza:

  `it.hourglass.myTalk.client.shared.User`

  `it.hourglass.myTalk.client.presenter.ContactProfilePresenter`

  `it.hourglass.myTalk.client.presenter.display.ContactProfileDisplay`

  `it.hourglass.myTalk.client.view.ContactProfile.AvatarContactProfileView`

  `it.hourglass.myTalk.client.view.ContactProfile.ExtendedDataContactProfileView`

  `it.hourglass.myTalk.client.view.ContactProfile.PersonalDataContactProfileView`

- La classe implementa:

  `it.hourglass.myTalk.client.presenter.display.ContactProfileDisplay`

Attributi

- `AbsolutePanel contenent`
  Pannello contenente il widget ContactProfileView.
- `AbsolutePanel title`
  Pannello che contiene il titolo 'Profilo'.
- `AbsolutePanel avatarPanel`
  Pannello che contiene il widget AvatarContactProfileView che servirà per l'inserimento dell'avatar.
- `AbsolutePanel personalDataPanel`
  Pannello che contiene il widget PersonalDataContactProfileView che conterrà i dati personali del contatto.
- `AbsolutePanel extendedDataPanel`
  Pannello che contiene il widget ExtendedDataContactProfileView che conterrà i dati estesi del contatto.
- `PersonalDataContactProfileView personalData`
  Riferimento ai dati personali del contatto.

- `ExtendedDataContactProfileView extendedData`
  Riferimento ai dati estesi del contatto.

Metodi

+ `ContactProfileView()`
  Costruttore della classe che inizializza il widget ContactProfileView disponendo le sue componenti.

+ `void init(User user)`
  Metodo che inizializza la view a seconda del contatto che si è scelto di visualizzare.

+ `AbsolutePanel getView()`
  Metodo che ritorna il widget ContactProfileView.

2.9.8 client.view.EditProfileView - Classe

Funzione

Questa classe rappresenta la view per la modifica del proprio profilo.

Relazione d'uso con altri moduli

- La classe è utilizzata da:

  `it.hourglass.myTalk.client.presenter.BaseViewPresenter`

- La classe utilizza:

  `it.hourglass.myTalk.client.presenter.EditProfilePresenter`

- La classe implementa:

  `it.hourglass.myTalk.client.presenter.display.EditProfileDisplay`

Attributi

- `AbsolutePanel contenent`
  Pannello contenente il widget EditProfileView.
- `AbsolutePanel title`
  Pannello che contiene il titolo 'Profilo'.
- `AbsolutePanel avatarPanel`
  Pannello che contiene il proprio avatar.
- `AbsolutePanel personalDataPanel`
  Pannello che contiene i dati principali personali.
- `AbsolutePanel extendedDataPanel`
  Pannello che contiene i dati estesi personali.
- `AbsolutePanel footerPage`
  Pannello che contiene il footer della pagina contenente un bottone per cancellare il proprio profilo e un'altro per andare a leggere i messaggi.
- `Button messageButton`
  Bottone per andare a leggere i messaggi.
- `Button deleteAccountButton`
  Bottone per eliminare il proprio profilo.

Metodi

- `+ EditProfileView()`
  Costruttore della classe che inizializza il widget EditProfileView disponendo le sue componenti.
- `+ HasClickHandlers getMessageButton()`
  Metodo che ritorna il bottone di accesso ai messaggi.
- `+ HasClickHandlers getDeleteAccountButton()`
  Metodo che ritorna il bottone di cancellazione del proprio account.

`+ AbsolutePanel getView()`
Metodo che restituisce il widget EditProfileView.

`+ void addAvatarView(AbsolutePanel panel)`
Metodo per aggiungere all'interno del relativo pannello l'avatar dell'utente.

`+ void addPersonalDataView(AbsolutePanel panel)`
Metodo per aggiungere all'interno del relativo pannello i dati personali dell'utente.

`+ void addExtendedDataView(AbsolutePanel panel)`
Metodo per aggiungere all'interno del relativo pannello i dati estesi dell'utente.

`+ void addFooter()`
Metodo che inserisce nel footer un bottone per cancellare il proprio profilo e un'altro per andare a leggere i messaggi.

2.9.9 client.view.EndCallPopup - Classe

Funzione

Questa classe rappresenta il popup che avvisa della chiusura della chiamata.

Relazione d'uso con altri moduli

- La classe è utilizzata da:

  `it.hourglass.myTalk.client.communication.WSConnection`

- La classe utilizza:

  `it.hourglass.myTalk.client.event.HomePageRequestEvent`

Attributi

- `PopupPanel popup`
  Pannello del popup che contiene l'avviso.
- `Button ok`
  Bottone che una volta cliccato indica la lettura della popup.

Metodi

+ `EndCallPopup(final HandlerManager eventBus,String callLenght)`
  Costruttore della classe. Si occupa di inizializzare i campi dati necessari per la costruzione della popup.

2.9.10 client.view.FooterView - Classe

Funzione

Questa classe rappresenta il widget FooterView attraverso la quale sarà possibile per l'utente vedere il proprio id identificativo.

Relazione d'uso con altri moduli

- La classe è utilizzata da:

  `it.hourglass.myTalk.client.presenter.BaseViewPresenter`

- La classe utilizza:

  `it.hourglass.myTalk.client.presenter.FooterPresenter`

- La classe implementa:

  `it.hourglass.myTalk.client.presenter.display.FooterDisplay`

Attributi

- `AbsolutePanel contenuto`
  Pannello contenente il widget FooterView.
- `String username`
  Stringa contenente il proprio id identificativo.
- `HTML htmlNomeutente`
  Parte di HTML che indica il proprio codice identificativo temporaneo.

Metodi

+ `FooterView(String nome)`
  Costruttore della classe. Si occupa di inizializzare i campi dati necessari al widget Footer e di disporre le sue componenti.
+ `String getUsername()`
  Metodo che restituisce l'id identificativo.
+ `AbsolutePanel getView()`
  Metodo che restituisce il widget FooterView.
+ `void setUsername(String username)`
  Metodo che si occupa di impostare l'id identificativo nel widget.
+ `void refresh()`
  Metodo che si occupa di eseguire il refresh del footer.

2.9.11 client.view.FriendContactView - Classe

Funzione

Questa classe rappresenta il widget FriendContactView attraverso la quale sarà possibile visualizzare il nickname di un nostro amico. Al fianco del suo nickname ci sarà il bottone di eliminazione nel caso lo volessimo togliere dai nostri amici.

Relazione d'uso con altri moduli

- La classe è utilizzata da:

  `it.hourglass.myTalk.client.presenter.ContactManagementPresenter`

- La classe utilizza:

  `it.hourglass.myTalk.client.presenter.FriendContactPresenter`

- La classe implementa:

  `it.hourglass.myTalk.client.presenter.display.FriendContactDisplay`

Attributi

- `AbsolutePanel contenent`
  Pannello contenente il widget FriendContactView.
- `String name`
  Stringa contenente il nickname del contatto amico.
- `Button remove`
  Bottone per rimuovere l'utente dagli amici.

Metodi

+ `FriendContactView(String name)`
  Costruttore della classe. Si occupa di inizializzare i campi dati necessari al widget FriendContact e di disporre le sue componenti.
+ `HasClickHandlers getRemoveButton()`
  Metodo che restituisce il bottone della rimozione dell'utente dagli amici.
+ `AbsolutePanel getView()`
  Metodo che restituisce il widget FriendContactView.
+ `void selfDelete()`
  Metodo per rimuovere il pannello content.

2.9.12 client.view.FriendMessageView - Classe

Funzione

Questa classe rappresenta il widget FriendMessageView attraverso la quale sarà possibile visualizzare il messaggio di una richiesta di amicizia di un utente che vuole diventare nostro amico.

Relazione d'uso con altri moduli

- La classe è utilizzata da:

  `it.hourglass.myTalk.client.presenter.ContactManagementPresenter`

- La classe utilizza:

  `it.hourglass.myTalk.client.shared.FriendMessage`

  `it.hourglass.myTalk.client.presenter.FriendMessagePresenter`

- La classe implementa:

  `it.hourglass.myTalk.client.presenter.display.FriendMessageDisplay`

Attributi

- `FriendMessage message`
  Variabile che contiene il nickname dell'utente che vuole diventare nostro amico.
- `AbsolutePanel contenent`
  Pannello contenente il widget FriendMessageView.
- `Button acceptButton`
  Bottone per accettare la richiesta di amicizia.
- `Button refuseButton`
  Bottone per rifiutare la richiesta di amicizia.

Metodi

+ `FriendMessageView(FriendMessage message)`
  Costruttore della classe. Si occupa di inizializzare i campi dati necessari al widget FriendMessage e di disporre le sue componenti.
+ `HasClickHandlers getAcceptButton()`
  Metodo che ritorna il bottone dell'accettazione della richiesta di amicizia.
+ `HasClickHandlers getRefuseButton()`
  Metodo che ritorna il bottone del rifiuto della richiesta di amicizia.
+ `AbsolutePanel getView()`
  Metodo che restituisce il widget FriendMessageView.
+ `void selfDelete()`
  Metodo per rimuovere il pannello content.

2.9.13 client.view.HomeView - Classe

Funzione

Questa classe rappresenta il widget HomeView e serve per visualizzare la homepage.

Relazione d'uso con altri moduli

- La classe è utilizzata da:

  `it.hourglass.myTalk.client.presenter.BaseViewPresenter`

- La classe utilizza:

  `it.hourglass.myTalk.client.presenter.HomePresenter`

- La classe implementa:

  `it.hourglass.myTalk.client.presenter.display.HomeDisplay`

Attributi

`- AbsolutePanel contenuto`
Pannello contenente il widget HomeView.

Metodi

`+ HomeView()`
Costruttore della classe. Si occupa di inizializzare i campi dati necessari della classe e di disporre le sue componenti.

`+ AbsolutePanel getView()`
Metodo che restituisce il widget HomeView.

2.9.14 client.view.LoginView - Classe

Funzione

Questa classe rappresenta il widget LoginView e serve per visualizzare la pagina di login.

Relazione d'uso con altri moduli

- La classe è utilizzata da:

  `it.hourglass.myTalk.client.presenter.BaseViewPresenter`

- La classe utilizza:

  `it.hourglass.myTalk.client.presenter.LoginPresenter`

- La classe implementa:

  `it.hourglass.myTalk.client.presenter.display.LoginDisplay`

Attributi

- `AbsolutePanel contenuto`
  Pannello contenente il widget LoginView.
- `TextBox usernameTextBox`
  TextBox per inserire al suo interno il proprio username per effettuare il login.
- `Button loginButton`
  Bottone per effettuare il login.
- `PasswordTextBox passwordTextBox`
  TextBox per inserire al suo interno la propria password per effettuare il login.
- `Label error`
  Label per segnalare errori durante l'operazione di login (ad esempio i dati immessi sono sbagliati).

Metodi

+ `LoginView()`
  Costruttore della classe. Si occupa di inizializzare i campi dati necessari della classe e di disporre le sue componenti.
+ `AbsolutePanel getView()`
  Metodo che restituisce il widget LoginView.
+ `HasClickHandlers getLoginButton()`
  Metodo per ritornare il bottone di login.
+ `HasValue<String> getUsername()`
  Metodo che restituisce l'username inserito.
+ `HasValue<String> getPassword()`
  Metodo che restituisce la password inserita.

`+ void showLoginError(String errore)`
Metodo per segnalare a video eventuali errori durante l'operazione di login.

2.9.15 client.view.MessageView - Classe

Funzione

Questa classe rappresenta il widget MessageView e serve per visualizzare la pagina dei messaggi.

Relazione d'uso con altri moduli

- La classe è utilizzata da:

  `it.hourglass.myTalk.client.presenter.BaseViewPresenter`

- La classe utilizza:

  `it.hourglass.myTalk.client.presenter.MessagePresenter`

- La classe implementa:

  `it.hourglass.myTalk.client.presenter.display.MessageDisplay`

Attributi

- `- AbsolutePanel contenent`
  Pannello contenente il widget MessageView.
- `- AbsolutePanel title`
  Pannello che contiene il titolo della pagina web.
- `- AbsolutePanel messageTitle`
  Pannello che contiene il sottotitolo della pagina web.
- `- AbsolutePanel messagePanel`
  Pannello che contiene i messaggi.

Metodi

- `+ MessageView()`
  Costruttore della classe. Si occupa di inizializzare i campi dati necessari della classe e di disporre le sue componenti.
- `+ addMessage(AbsolutePanel panel)`
  Metodo per aggiungere un messaggio.
- `+ AbsolutePanel getView()`
  Metodo che restituisce il widget MessageView.

2.9.16 client.view.PasswordRecoverView - Classe

Funzione

Questa classe rappresenta il widget PasswordRecoverView e serve per visualizzare la pagina del recupero password.

Relazione d'uso con altri moduli

- La classe è utilizzata da:

  `it.hourglass.myTalk.client.presenter.BaseViewPresenter`

- La classe utilizza:

  `it.hourglass.myTalk.client.presenter.PasswordRecoverPresenter`

- La classe implementa:

  `it.hourglass.myTalk.client.presenter.display.PasswordRecoverDisplay`

Attributi

- `AbsolutePanel content`
  Pannello contenente il widget PasswordRecoverView.
- `AbsolutePanel fase1`
  Pannello che contiene le componenti della prima fase per il recupero password.
- `AbsolutePanel fase2`
  Pannello che contiene le componenti della seconda fase per il recupero password.
- `AbsolutePanel fase3`
  Pannello che contiene le componenti della terza fase per il recupero password.
- `TextBox textBoxEmail`
  TextBox per inserire al suo interno il proprio indirizzo e-mail.
- `TextBox confirmCode`
  TextBox per inserire al suo interno il codice segreto inviato precedentemente via e-mail per il recupero password.
- `PasswordTextBox newPassword`
  TextBox per inserire al suo interno la nuova password.
- `PasswordTextBox newPasswordConfirm`
  TextBox per inserire al suo interno la conferma della nuova password inserita.
- `Label Email`
  Label per indicare che nel TextBox va inserita la e-mail.
- `Label Text1`
  Label che invita a compilare i campi per il recupero password.
- `Label Text2`
  Label per indicare che è stato inviato nella propria e-mail un codice segreto per il recupero password.

- `Label Text3`
  Label per indicare che nel TextBox va inserito il codice segreto arrivato via e-mail per il recupero password.
- `Label Text4`
  Label per indicare che nel TextBox va inserita la nuova password.
- `Button RequestRecover`
  Bottone per ricevere, dopo aver inserito la propria e-mail, il codice segreto per il recupero password.
- `Button ContinueButton`
  Bottone per proseguire l'operazione di recupero password dopo aver inserito il codice segreto.
- `Button ConfirmButton`
  Bottone per confermare la nuova password scelta.
- `Grid grid`
  Griglia che racchiude le label e le PasswordTextBox per l'inserimento della password e della sua conferma.
- `Label Error`
  Label che avvisa a video eventuali errori riscontrati.

Metodi

+ `PasswordRecoverView()`
  Costruttore della classe. Si occupa di inizializzare i campi dati necessari della classe e di disporre le sue componenti.
+ `AbsolutePanel getView()`
  Metodo che restituisce il widget PasswordRecoverView.
+ `HasClickHandlers getRequestButton()`
  Metodo che ritorna il bottone `RequestRecover.`
+ `HasClickHandlers getContinueButton()`
  Metodo che ritorna il bottone `ContinueButton.`
+ `HasClickHandlers getConfirmButton()`
  Metodo che ritorna il bottone `ConfirmButton.`
+ `void switchFase2()`
  Metodo che si occupa di eseguire lo switch view del contenuto per passare alla view della seconda fase.
+ `void switchFase3()`
  Metodo che si occupa di eseguire lo switch view del contenuto per passare alla view della terza fase.
+ `void showErrorFase1(String errore)`
  Metodo che si occupa di segnalare eventuali errori della prima fase.

+ `void showErrorFase2(String errore)`
  Metodo che si occupa di segnalare eventuali errori della seconda fase.
+ `void showErrorFase3(String errore)`
  Metodo che si occupa di segnalare eventuali errori della terza fase.
+ `TextBox getEmail()`
  Metodo che ritorna la e-mail inserita.
+ `PasswordTextBox getPassword()`
  Metodo che ritorna la nuova password inserita.
+ `PasswordTextBox getConfirmPassword()`
  Metodo che ritorna la conferma della nuova password inserita.
+ `TextBox getConfirmCode()`
  Metodo che ritorna il codice segreto inserito.
+ `void showLoginError(String error)`
  Metodo che inserisce nell'apposito pannello il messaggio di errore se il login è sbagliato.

2.9.17 client.view.PopupInformation - Classe

Funzione

Questa classe rappresenta il widget PopupInformation e serve per segnalare all'utente eventuali avvisi (chiamate perse, rifiuto della chiamata da parte del destinatario, arrivo di una richiesta di amicizia o di un messaggio ricevuto, eliminazione da parte di un nostro amico del nostro contatto nella sua lista contatti, conferma che un contatto è stato aggiunto alla nostra lista di amici, conferma dell'eliminazione di un messaggio, database non raggiungibile oppure avvisi di errori come richieste di amicizia inviate a sè stessi o una richiesta di un profilo non esistente).

Relazione d'uso con altri moduli

- La classe è utilizzata da:

  `it.hourglass.myTalk.client.presenter.CallPopupPresenter`

  `it.hourglass.myTalk.client.presenter.ContactManagementPresenter`

  `it.hourglass.myTalk.client.presenter.ContactProfilePresenter`

  `it.hourglass.myTalk.client.presenter.FriendMessagePresenter`

  `it.hourglass.myTalk.client.presenter.TextMessagePresenter`

  `it.hourglass.myTalk.client.presenter.PopupMessagePresenter`

  `it.hourglass.myTalk.client.communication.WSConnection`

  `it.hourglass.myTalk.client.appcontroller.AppController`

Attributi

- `PopupPanel popup`
  Box per il popup.
- `Button ok`
  Bottone per confermare la lettura della popup.
- `Timer timer`
  Variabile che dopo esser passato un pò di tempo e senza aver cliccato OK chiuderà la popup (ammesso che quest'ultima si possa chiudere da sola, ciò è dato dalla variabile autoclose).
- `Boolean autoclose`
  Booleano che conterrà `true` se la popup dopo 5 secondi che l'utente non clicca su OK dovrà chiudersi, `false` altrimenti.

Metodi

+ `PopupInformation(String message,Boolean autoclose)`
  Costruttore della classe. Si occupa di inizializzare i campi dati necessari della classe e di disporre le sue componenti.

2.9.18 client.view.PopupMessageView - Classe

Funzione

Questa classe rappresenta il widget PopupMessageView e serve per scrivere e inviare messaggi.

Relazione d'uso con altri moduli

- La classe è utilizzata da:

  `it.hourglass.myTalk.client.presenter.UserListPresenter`

- La classe utilizza:

  `it.hourglass.myTalk.client.presenter.PopupMessagePresenter`

- La classe implementa:

  `it.hourglass.myTalk.client.presenter.display.PopupMessageDisplay`

Attributi

- `DialogBox MessageDialogBox`
  Box per il popup.
- `String receiver`
  Stringa che conterrà il nickname dell'utente alla quale vogliamo inviare il messaggio.
- `TextArea textMessage`
  Box per scrivere al suo interno il messaggio da inviare.
- `Button cancel`
  Bottone per cancellare il messaggio.
- `Button send`
  Bottone per inviare il messaggio.
- `Label lenghtLabel`
  Label che servirà ad avvisare che il messaggio non deve superare la lunghezza di 255 caratteri.

Metodi

- `+ PopupMessageView(String receiver)`
  Costruttore della classe. Si occupa di inizializzare i campi dati necessari della classe e di disporre le sue componenti.
- `+ HasClickHandlers getSendButton()`
  Metodo che ritorna il bottone dell'invio del messaggio.
- `+ HasClickHandlers getCancellButton()`
  Metodo che ritorna il bottone della cancellazione del messaggio.
- `+ void remove()`
  Metodo per uscire dalla popup del messaggio.

`+ HasValue<String> getText()`
Metodo che ritorna il testo che è stato inserito nell'apposita area di testo.

`+ void error()`
Metodo che avvisa a video su una label un'errore nel caso fosse stato inserito un numero di caratteri superiore a quello consentito.

2.9.19 client.view.RegistrationView - Classe

Funzione

Questa classe rappresenta il widget RegistrationView e serve per visualizzare la pagina di registrazione.

Relazione d'uso con altri moduli

- La classe è utilizzata da:

  `it.hourglass.myTalk.client.presenter.BaseViewPresenter`

- La classe utilizza:

  `java.util.Date;`

  `it.hourglass.myTalk.client.presenter.RegistrationPresenter`

- La classe implementa:

  `it.hourglass.myTalk.client.presenter.display.RegistrationDisplay`

Attributi

- `AbsolutePanel content`
  Pannello contenente il widget RegistrationView.
- `Button registerButton`
  Bottone per confermare i dati inseriti per la registrazione.
- `Button showConfirmCodeButton`
  Bottone che serve ad accedere alla pagina contenente il textbox per inserire il codice segreto inviato via email, completando così la registrazione.
- `Button confirmCodeButton`
  Bottone per confermare il codice segreto che l'utente ha inserito nella textbox. Infatti per completare la registrazione verrà inviato via email questo codice che bisognerà inserire nella textbox menzionata sopra.
- `TextBox textBoxUsername`
  TextBox per inserire l'username.
- `PasswordTextBox textBoxPassword`
  PasswordTextBox per inserire la password.
- `PasswordTextBox textBoxRePassword`
  PasswordTextBox per reinserire la password.
- `TextBox textBoxEmail`
  TextBox per inserire la propria e-mail.
- `TextBox textBoxName`
  TextBox per inserire il proprio nome.
- `TextBox textBoxLastName`
  TextBox per inserire il proprio cognome.

- `TextBox textBoxAddress`
  TextBox per inserire il proprio indirizzo civico.
- `ListBox comboBoxDay`
  Menù a lista per selezionare il proprio giorno di nascita.
- `ListBox comboBoxMonth`
  Menù a lista per selezionare il proprio mese di nascita.
- `ListBox comboBoxYear`
  Menù a lista per selezionare il proprio anno di nascita.
- `ListBox comboBoxGender`
  Menù a lista per selezionare il proprio sesso.
- `Label wrongUsername`
  Label per avvisare eventuali errori sollevati se l'username inserito è sbagliato indicandone il motivo.
- `Label wrongPassword`
  Label per avvisare eventuali errori sollevati se la password inserita è sbagliata indicandone il motivo.
- `Label wrongEmail`
  Label per avvisare eventuali errori sollevati se l'e-mail inserita è sbagliata indicandone il motivo.
- `Label wrongName`
  Label per avvisare eventuali errori sollevati se il nome inserito è sbagliato indicandone il motivo.
- `Label wrongLastName`
  Label per avvisare eventuali errori sollevati se il cognome inserito è sbagliato indicandone il motivo.
- `Label wrongDate`
  Label per avvisare eventuali errori sollevati se la data di nascita è sbagliata indicandone il motivo.
- `Label outcome`
  Label che conferma all'utente l'avvenuta registrazione.
- `Label confirmationOutcome`
  Label che conferma oppure no la validità del codice segreto inserito.
- `TextBox confirmCode`
  TextBox per confermare il codice per la conferma della registrazione.
- `TextBox confirmUniqueId`
  TextBox per inserire l'identificativo dell'utente che si vuole registrare.

Metodi

+ `RegistrationView()`
  Costruttore della classe. Si occupa di inizializzare i campi dati necessari della classe e di disporre le sue componenti.
+ `AbsolutePanel getView()`
  Metodo che restituisce il widget RegistrationView.
+ `HasClickHandlers getRegisterButton()`
  Metodo che ritorna il bottone della registrazione.
+ `HasValue<String> getUsername()`
  Metodo che ritorna l'username inserito.
+ `HasValue<String> getEmail()`
  Metodo che ritorna l'e-mail inserita.
+ `HasValue<String> getPassword()`
  Metodo che ritorna la password inserita.
+ `HasValue<String> getName()`
  Metodo che ritorna il nome inserito.
+ `HasValue<String> getLastName()`
  Metodo che ritorna il cognome inserito.
+ `HasValue<String> getAddress()`
  Metodo che ritorna l'indirizzo civico inserito.
+ `String getGender()`
  Metodo che ritorna il sesso inserito.
+ `HasValue<String> getRePassword()`
  Metodo che ritorna la conferma della password inserita.
+ `String getBirthDay()`
  Metodo che ritorna il giorno di nascita inserito.
+ `String getBirthMonth()`
  Metodo che ritorna il mese di nascita inserito.
+ `String getBirthYear()`
  Metodo che ritorna l'anno di nascita inserito.
+ `void setWrongUsername(String error)`
  Metodo che segnala a video che l'username inserito è sbagliato indicandone il motivo.
+ `void setWrongName(String error)`
  Metodo che segnala a video che il nome inserito è sbagliato indicandone il motivo.
+ `void setWrongLastName(String error)`
  Metodo che segnala a video che il cognome inserito è sbagliato indicandone il motivo.

+ `void setWrongPassword(String error)`
  Metodo che segnala a video che la password inserita è sbagliata indicandone il motivo.
+ `void setWrongDate(String error)`
  Metodo che segnala a video che la data inserita è sbagliata indicandone il motivo.
+ `void setWrongEmail(String error)`
  Metodo che segnala a video che l'e-mail inserita è sbagliata indicandone il motivo.
+ `setOutcome(String result)`
  Metodo che segnala a video se i dati inseriti durante la registrazione e dopo averli validati sono stati salvati correttamente nel database del server.
+ `void setConfirmationOutcome(String result)`
  Metodo che segnala a video se il codice segreto inserito è valido oppure no.
+ `HasClickHandlers getShowConfirmCodeButton()`
  Metodo che ritorna il bottone `showConfirmCodeButton.`
+ `HasClickHandlers getConfirmCodeButton()`
  Metodo che ritorna il bottone `confirmCodeButton.`
+ `HasValue<String> getConfirmCode()`
  Metodo che ritorna la variabile `confirmCode.`
+ `HasValue<String> getConfirmUniqueId()`
  Metodo che ritorna la variabile `confirmUniqueId.`
+ `void changeView()`
  Metodo che cambia, dopo aver confermato i dati inseriti e dato la conferma, la view di registrazione per mostrare una nuova view contente l'indicazione per l'inserimento del codice segreto inviato via email. Una volta inserito la registrazione è stata completata.

2.9.20 client.view.SignedMenuView - Classe

Funzione

Questa classe rappresenta il widget menù per gli utenti che sono autenticati. Questo widget ha la funzione di permettere all'utente lo switch della view desiderata.

Relazione d'uso con altri moduli

- La classe è utilizzata da:

  `it.hourglass.myTalk.client.presenter.BaseViewPresenter`

- La classe utilizza:

  `it.hourglass.myTalk.client.presenter.SignedMenuPresenter`

- La classe implementa:

  `it.hourglass.myTalk.client.presenter.display.SignedMenuDisplay`

- La classe estende:

  `com.google.gwt.user.client.ui.Composite`

Attributi

- `Button HomeButton`
  Bottone per accedere alla homepage.
- `Button CallButton`
  Bottone per accedere alla pagina di chiamata.
- `Button ProfileManagementButton`
  Bottone per accedere alla pagina di gestione del proprio profilo.
- `Button LogoutButton`
  Bottone per effettuare il logout.
- `AbsolutePanel contenuto`
  Pannello contenente il widget SignedMenuView.

Metodi

+ `SignedMenuView()`
  Costruttore della classe. Si occupa di inizializzare i campi dati necessari della classe e di disporre le sue componenti.
+ `HasClickHandlers getCallButton()`
  Metodo che ritorna il bottone per accedere alla pagina di chiamata.
+ `HasClickHandlers getHomeButton()`
  Metodo che ritorna il bottone per accedere alla homepage.
+ `AbsolutePanel getView()`
  Metodo che restituisce il widget SignedMenuView.

+ `HasClickHandlers getProfileManagementButton()`
  Metodo che ritorna il bottone per accedere alla pagina di gestione del proprio profilo.
+ `HasClickHandlers getLogoutButton()`
  Metodo che ritorna il bottone per effettuare il logout.

2.9.21 client.view.TextMessageView - Classe

Funzione

Questa classe rappresenta il widget TextMessageView e serve per visualizzare il messaggio di un amico che ce l'ha inviato.

Relazione d'uso con altri moduli

- La classe è utilizzata da:

  `it.hourglass.myTalk.client.presenter.MessagePresenter`

- La classe utilizza:

  `it.hourglass.myTalk.client.shared.TextMessage`

  `it.hourglass.myTalk.client.presenter.TextMessagePresenter`

- La classe implementa:

  `it.hourglass.myTalk.client.presenter.display.TextMessageDisplay`

Attributi

- `AbsolutePanel contenent`
  Pannello contenente il widget TextMessageView (ovvero contenente il messaggio ricevuto, il nome dell'utente che ce lo ha inviato e il bottone per cancellarlo).

- `Button remove`
  Bottone per cancellare il messaggio.

Metodi

+ `TextMessageView(TextMessage message)`
  Costruttore della classe. Si occupa di inizializzare i campi dati necessari della classe e di disporre le sue componenti.

+ `AbsolutePanel getView()`
  Metodo che restituisce il widget TextMessageView.

+ `Button getRemoveButton()`
  Metodo che restituisce il bottone della eliminazione del messaggio.

+ `void selfDelete()`
  Metodo che rimuove la finestra del messaggio.

2.9.22 client.view.UserListView - Classe

Funzione

Questa classe rappresenta il widget UserListView e serve per visualizzare la lista utenti.

Relazione d'uso con altri moduli

- La classe è utilizzata da:

  `it.hourglass.myTalk.client.presenter.ContactListPresenter`

- La classe utilizza:

  `java.util.Map.Entry`

- La classe implementa:

  `it.hourglass.myTalk.client.presenter.display.UserListDisplay`

Attributi

- `AbsolutePanel contenent`
  Pannello contenente il widget UserListView.
- `boolean status`
  Booleano che indica se un utente è online oppure no.
- `Image statusImage`
  Icona che indica se un utente è online oppure no.
- `Label uniqueId`
  Label che contiene il nome dell'utente in lista.
- `PushButton callButton`
  Bottone per chiamare l'utente in lista.
- `PushButton messageButton`
  Bottone per inviare messaggi all'utente in lista.
- `PushButton profileButton`
  Bottone per vedere il profilo dell'utente in lista.

Metodi

- `+ UserListView(Entry<String, Boolean> userInformation)`
  Costruttore della classe. Si occupa di inizializzare i campi dati necessari della classe e di disporre le sue componenti.
- `+ void setStatusImage()`
  Metodo che in base allo stato dell'utente (online o offline) sceglie l'icona da inserire.
- `+ AbsolutePanel getView()`
  Metodo che restituisce il widget UserListView.
- `+ HasClickHandlers getCallButton()`
  Metodo che ritorna il bottone per chiamare l'utente in lista.

`+ HasClickHandlers getMessageButton()`
Metodo che ritorna il bottone per inviare messaggi all'utente in lista.

`+ HasClickHandlers getProfileButton()`
Metodo che ritorna il bottone per vedere il profilo dell'utente in lista.

`+ void changeStatus(boolean status)`
Metodo per cambiare lo stato di un utente (da online a offline o viceversa) che è nella lista.

`+ void remove()`
Metodo per rimuovere questo widget.

2.10 Package it.hourglass.myTalk.client.view.ContactProfile

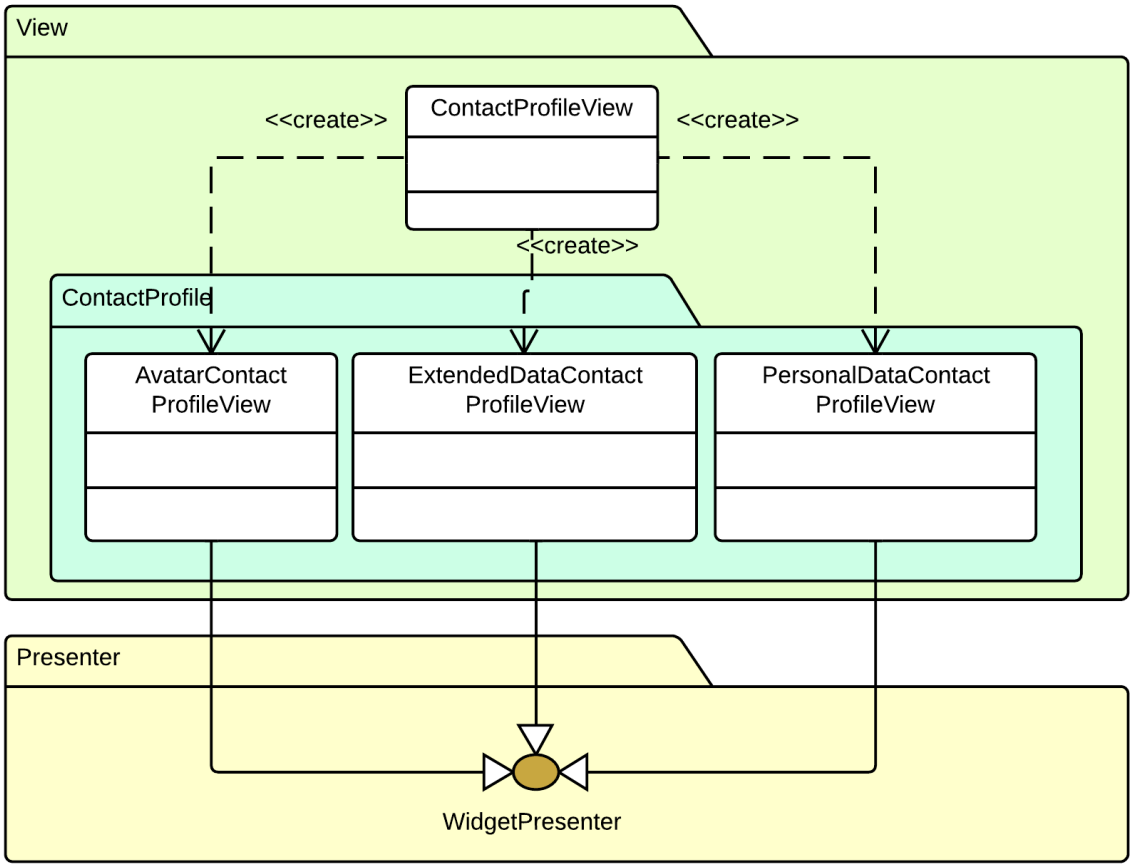

Figura 5: Package it.hourglass.myTalk.client.view.ContactProfile

2.10.1 client.view.ContactProfile.AvatarContactProfileView - Classe

Funzione

Questa classe è uno dei widget della ContactProfileView e si occupa di visualizzare l'avatar del contatto richiesto.

Relazione d'uso con altri moduli

- La classe è utilizzata da:

  `it.houglass.myTalk.client.view.ContactProfileView`

- La classe utilizza:

  `it.hourglass.myTalk.client.presenter.WidgetPresenter`

  `it.hourglass.myTalk.client.shared.Profile`

  `it.hourglass.myTalk.client.shared.User`

- La classe implementa:

  `it.hourglass.myTalk.client.presenter.WidgetPresenter`

Attributi

`AbsolutePanel contenent`
Pannello contenente l'avatar.

`Image avatar`
Immagine dell'avatar.

Metodi

`+ AvatarContactProfileView(User user)`
Costruttore della classe AvatarContactProfileView. Si occupa di inizializzare il pannello che contiene il widget inserendo al suo interno l'immagine dell'avatar.

`+ AbsolutePanel getView()`
Restituisce questo widget.

2.10.2 client.view.ContactProfile.ExtendedDataContactProfileView - Classe

Funzione

Questa classe si occupa di visualizzare le informazioni personali estese del contatto richiesto.

Relazione d'uso con altri moduli

- La classe è utilizzata da:

  `it.hourglass.myTalk.client.view.ContactProfileView`

- La classe utilizza:

  `it.hourglass.myTalk.client.shared.User`

  `it.hourglass.myTalk.client.shared.Profile`

  `it.hourglass.myTalk.client.presenter.WidgetPresenter`

- La classe implementa:

  `it.hourglass.myTalk.client.presenter.WidgetPresenter`

Attributi

`AbsolutePanel contenent`
Pannello che contiene i vari pannelli che racchiuderanno i dati personali estesi.

`AbsolutePanel cityLabelPanel`
Questa label specifica che il campo a fianco contiene la città di residenza del contatto.

`AbsolutePanel cityPanel`
Pannello che contiene la label della città di residenza del contatto.

`AbsolutePanel HometownLablePanel`
Questa label specifica che il campo a fianco contiene la città di origine del contatto.

`AbsolutePanel HometownPanel`
Pannello che contiene la label della città di origine del contatto.

`AbsolutePanel AltEmailLabelPanel`
Questa label specifica che il campo a fianco contiene l'email alternativa del contatto.

`AbsolutePanel AltEmailPanel`
Pannello che contiene la label della email alternativa del contatto.

`AbsolutePanel descriptionPanel`
Pannello che contiene il panello della descrizione del contatto.

`AbsolutePanel interestsPanel`
Pannello che contiene il pannello degli interessi del contatto.

`AbsolutePanel inspirationsPanel`
Pannello che contiene il pannello delle ispirazioni del contatto.

`AbsolutePanel descriptionTextPanel`
Pannello che contiene la descrizione del contatto.

`AbsolutePanel interestsTextPanel`
Pannello che contiene gli interessi del contatto.

`AbsolutePanel inspirationsTextPanel`
Pannello che contiene le ispirazioni del contatto.

Metodi

`+ ExtendedDataContactProfileView(User user)`
Costruttore della classe. Si occupa di inizializzare il pannello principale del widget (contenent) e di inserire gli altri pannelli che conterranno i dati personali estesi dell'utente richiesto. Successivamente inizializza le varie label contenti le informazioni dei dati personali estesi.

`+ AbsolutePanel getView()`
Restituisce questo widget.

2.10.3 client.view.ContactProfile.PersonalDataContactProfileView - Classe

Funzione

Questa classe si occupa di visualizzare le informazioni personali principali del contatto richiesto.

Relazione d'uso con altri moduli

- La classe è utilizzata da:

  `it.hourglass.myTalk.client.view.ContactProfileView`

- La classe utilizza:

  `it.hourglass.myTalk.client.shared.User`

  `it.hourglass.myTalk.client.shared.Profile`

  `it.hourglass.myTalk.client.presenter.WidgetPresenter`

- La classe implementa:

  `it.hourglass.myTalk.client.presenter.WidgetPresenter`

Attributi

`AbsolutePanel contenent`
Pannello che contiene i vari pannelli che racchiuderanno i dati personali principali.

`AbsolutePanel name_LastName_EmailPanel`
Pannello contenente tre label contenenti l'indicazione che al loro fianco si troveranno i campi del nome, cognome ed email del contatto.

`AbsolutePanel dataName_LastName_EmailPanel`
Pannello contenente il nome, cognome ed email del contatto.

`AbsolutePanel sex_BirthDate_Panel`
Pannello contenente due label contenenti l'indicazione che al loro fianco si troveranno i campi del sesso e della data di nascita del contatto.

`AbsolutePanel dataSex_BirthDatePanel`
Pannello che contiene il sesso e la data di nascita del contatto.

Metodi

`+ PersonalDataContactProfileView(User user)`
Costruttore della classe. Si occupa di inizializzare il pannello principale del widget (contenent) e di inserire gli altri pannelli che conterranno i dati personali principali dell'utente richiesto. Successivamente inizializza le varie contenti le informazioni dei dati personali principali.

`+ AbsolutePanel getView()`
Restituisce questo widget.

2.11 Package it.hourglass.myTalk.client.view.ProfileManagment

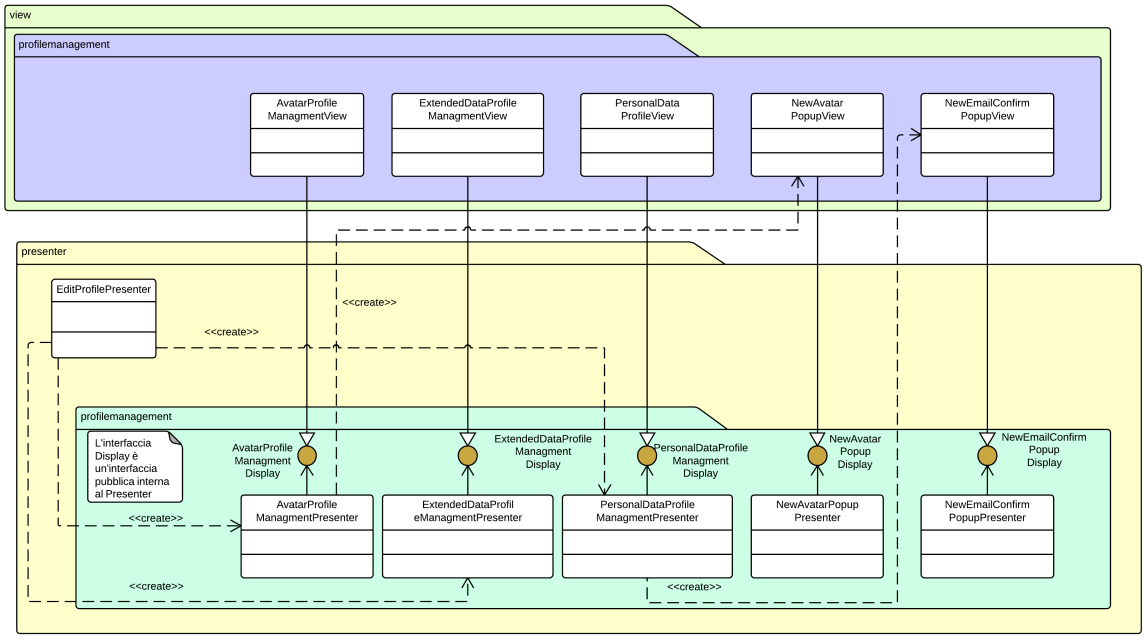

Figura 6: Package it.hourglass.myTalk.client.view.ProfileManagment

2.11.1 client.view.ProfileManagment.AvatarProfileManagmentView - Classe

Funzione

Questa classe si occupa di visualizzare l'avatar del proprio contatto e di richiedere il popup per la modifica dello stesso.

Relazione d'uso con altri moduli

- La classe è utilizzata da:

  `it.hourglass.myTalk.client.presenter.EditProfilePresenter`

- La classe utilizza:

  `it.hourglass.myTalk.client.shared.User`

  `it.hourglass.myTalk.client.profile.Profile`

  `it.hourglass.myTalk.client.presenter.ProfileManagment.AvatarProfileManagmentPresenter`

- La classe implementa:

  `it.hourglass.myTalk.client.presenter.display.AvatarProfileManagmentDisplay`

Attributi

`AbsolutePanel contenent`
Pannello contenente l'avatar e il bottone per poterlo cambiare.

`Button changeAvatarButton`
Bottone che apre una popup per poter cambiare l'avatar.

`Image avatar`
Immagine dell'avatar.

`private User user`
Riferimento all'oggetto user per poter recuperare il nostro avatar.

Metodi

`+ AvatarProfileManagmentView()`
Costruttore della classe. Si occupa di inizializzare il pannello che contiene il widget e di caricare e visualizzare il nostro avatar. Inoltre aggiunge anche il bottone `changeAvatarButton` per poter richiedere il popup per cambiare avatar.

`+ AbsolutePanel getView()`
Metodo che restituisce questo widget.

`+ Button getChangeAvatarButton()`
Bottone per restituire il bottone per il cambio avatar.

2.11.2 client.view.ProfileManagment.ExtendedDataProfileManagmentView - Classe

Funzione

Questa classe si occupa di visualizzare le informazioni estese del proprio contatto e di renderle modificabili.

Relazione d'uso con altri moduli

- La classe è utilizzata da:

  `it.hourglass.myTalk.client.presenter.EditProfilePresenter`

- La classe utilizza:

  `it.hourglass.myTalk.client.shared.User`

  `it.hourglass.myTalk.client.shared.Profile`

  `it.hourglass.myTalk.client.presenter.ProfileManagment.ExtendedDataProfileManagmentPresenter`

- La classe implementa:

  `it.hourglass.myTalk.client.presenter.display.ExtendedDataProfileManagmentDisplay`

Attributi

`User user`
Riferimento all'oggetto User che contiene tutti i nostri dati estesi.

`AbsolutePanel contenent`
Pannello che contiene i vari pannelli che racchiuderanno i dati personali estesi.

`AbsolutePanel cityLabelPanel`
Questa label specifica che il campo a fianco contiene la città di residenza del contatto.

`AbsolutePanel cityPanel`
Pannello che contiene la label della città di residenza del contatto.

`AbsolutePanel HometownLablePanel`
Questa label specifica che il campo a fianco contiene la città di origine del contatto.

`AbsolutePanel HometownPanel`
Pannello che contiene la label della città di origine del contatto.

`AbsolutePanel AltEmailLabelPanel`
Questa label specifica che il campo a fianco contiene l'email alternativa del contatto.

`AbsolutePanel AltEmailPanel`
Pannello che contiene la label della email alternativa del contatto.

`AbsolutePanel descriptionPanel`
Pannello che contiene il panello della descrizione del contatto.

`AbsolutePanel interestsPanel`
Pannello che contiene il pannello degli interessi del contatto.

`AbsolutePanel inspirationsPanel`
Pannello che contiene il pannello delle ispirazioni del contatto.

`AbsolutePanel descriptionTextPanel`
Pannello che contiene la descrizione del contatto.

`AbsolutePanel interestsTextPanel`
Pannello che contiene gli interessi del contatto.

`AbsolutePanel inspirationsTextPanel`
Pannello che contiene le ispirazioni del contatto.

`AbsolutePanel changeButtonPanel`
Bottone per modificare i dati personali estesi.

`Label error`
Label che stamperà a video eventuali errori sui dati che si è cercato di modificare.

`TextBox cityTextBox`
Area di testo che permette di modificare la locazione corrente dell'utente.

`TextBox HometownTextBox`
Area di testo che permette di modificare la città natale dell'utente.

`TextBox AltEmailTextBox`
Area di testo che permette di modificare l'email alternativa dell'utente.

`TextArea interestsTextArea`
Area di testo che permette di modificare gli interessi dell'utente.

`TextArea descriptionsTextArea`
Area di testo che permette di modificare la descrizione dell'utente.

`TextArea inspirationsTextArea`
Area di testo che permette di modificare le ispirazioni dell'utente.

`Button change`
Bottone per cambiare la view e rendere i dati estesi modificabili.

`Button confirm`
Bottone per confermare le modifiche apportate sui dati estesi.

Metodi

`+ ExtendedDataProfileManagmentView()`
Costruttore della classe. Si occupa di inizializzare il pannello che conterrà il widget (contenent) e successivamente di inizializzare tutti i pannelli. Per ogni dato da visualizzare (tranne `interests`, `descriptions`, `inspirations`) sono presenti due pannelli: un pannello a sinistra per identificare il dato e un altro pannello a destra in cui verrà visualizzato il dato. Per i campi dati `interests`, `descriptions`, `inspirations` invece viene aggiunta una TextArea contenente il testo corrispettivo. Inoltre si occupa di aggiungere le label che identificano il dato e di aggiungere

i bottoni per la modifica dei dati o per confermare le modifiche effettuate. Infine viene invocato il metodo `visualize()`.

`+ void visualize()`
Questo metodo si occupa di inserire all'interno del pannello di ogni dato la label corrispondente.

`- void clear()`
Questo metodo si occupa di eliminare il contenuto dei pannelli che contengono i dati.

`+ void change()`
Questo metodo si occupa di modificare la view dando all'utente la possibilità di modificare i campi dati del suo profilo. Al posto delle label nei pannelli vengono aggiunti dei TextBox e si rende editabile la TextArea di `interests`, `descriptions`, `inspirations`. Inoltre cambia il bottone visualizzato da `ChangeButton` a `ConfirmButton`.

`+ HasValue<String> getCity()`
Questo metodo ritorna la città di residenza dell'utente.

`+ HasValue<String> getHometown()`
Questo metodo ritorna la città di origine dell'utente.

`+ HasValue<String> getAltEmail()`
Questo metodo ritorna la email alternativa dell'utente.

`+ HasValue<String> getInterests()`
Questo metodo ritorna gli interessi dell'utente.

`+ HasValue<String> getDescriptions()`
Questo metodo ritorna la descrizione dell'utente.

`+ HasValue<String> getInspirations()`
Questo metodo ritorna le ispirazioni dell'utente.

`+ HasClickHandlers getChangeButton()`
Questo metodo ritorna il bottone `ChangeButton.`

`+ HasClickHandlers getConfirmButton()`
Questo metodo ritorna il bottone `ConfirmButton.`

`+ AbsolutePanel getView()`
Metodo che ritorna questo widget.

`+ void changeView()`
Questo metodo richiede il cambio di view passando da quella con i dati solamente visualizzati a quella per poter modificare i dati.

`+ void showError(String error)` Questo metodo permette di stampare a video un eventuale errore.

`+ void visualizeView()`
Questo metodo si occupa di cambiare la view in modo da rendere i campi dati da modificabili a solo visualizzabili.

2.11.3 client.view.ProfileManagment.NewAvatarPopupView - Classe

Funzione

Questa classe ha la funzione di popup per poter cambiare il proprio avatar.

Relazione d'uso con altri moduli

- La classe è utilizzata da:

  `it.hourglass.myTalk.client.presenter.ProfileManagment.AvatarProfileManagmentPresenter`

- La classe utilizza:

  `it.hourglass.myTalk.client.presenter.ProfileManagment.NewAvatarPopupPresenter`

- La classe implementa:

  `it.hourglass.myTalk.client.presenter.display.NewAvatarPopupDisplay`

Attributi

`DialogBox dialogBoxMail`
DialogBox che funziona da popup.

`Button load`
Bottone per caricare l'avatar scelto.

`Button cancel`
Bottone per chiudere la popup.

`FormPanel form`
Form necessaria per far funzionare la variabile `upload` in quanto è da questa struttura che viene inviato il file.

`FileUpload upload`
Visualizza una casella di testo e un pulsante sfoglia che consente all'utente di selezionare un'immagine per impostarla come nuovo avatar.

Metodi

`+ NewAvatarPopupView()`
Costruttore della classe. Si occupa di inizializzare i campi dati necessari della classe e di disporre le sue componenti.

`+ HasClickHandlers getConfirmButton()`
Metodo che ritorna il bottone per caricare l'avatar scelto.

`+ HasClickHandlers getCancelButton()`
Metodo che ritorna il bottone per chiudere la popup.

`+ void removePopup()`
Metodo che serve per chiudere la popup.

`+ FormPanel getFormPanel()`
Metodo che ritorna la variabile `form`.

`+ FileUpload getFileUpload()`
Metodo che ritorna la variabile `upload`.

2.11.4 client.view.ProfileManagment.NewEmailConfirmPopupView - Classe

Funzione

Classe che funziona da popup per il cambiamento dell'indirizzo email.

Relazione d'uso con altri moduli

- La classe è utilizzata da:

  `it.hourglass.myTalk.client.presenter.ProfileManagment.PersonalDataProfileManagmentPresenter`

- La classe utilizza:

  `it.hourglass.myTalk.client.presenter.ProfileManagment.NewEmailConfirmPopupPresenter`

- La classe implementa:

  `it.hourglass.myTalk.client.presenter.display.NewEmailConfirmPopupDisplay`

Attributi

- `DialogBox dialogBoxMail`
  DialogBox che funziona da popup.
- `TextBox code`
  Textbox che servirà a scrivere al suo interno il codice segreto che è stato inviato nella vecchia email per confermare il nuovo indirizzo email.
- `Button confirm`
  Bottone che una volta cliccato servirà a controllare se il codice inserito è corretto.
- `Button cancel`
  Bottone per chiudere la popup.
- `Label errorLabel`
  Label che nel caso il codice inserito per il cambiamento dell'email non è valido avviserà a video l'errore.

Metodi

- `+ NewEmailConfirmPopupView()`
  Costruttore della classe. Si occupa di inizializzare i campi dati necessari della classe e di disporre le sue componenti.
- `- HasClickHandlers getConfirmButton()`
  Metodo che ritornerà il bottone `confirm`.
- `- HasClickHandlers getCancelButton()`
  Metodo che ritornerà il bottone `cancel`.
- `+ HasValue<String> getSecretCode()`
  Metodo che ritornerà la variabile `code`.

`+ void setError(String error)`
Metodo che serve a impostare sulla label `errorLabel` l'eventuale errore riscontrato sulla validità del codice segreto inserito.

`+ void close()`
Metodo per chiudere la popup.

2.11.5 client.view.ProfileManagment.PersonalDataProfileView - Classe

Funzione

Questa classe si occupa di visualizzare le informazioni principali del proprio contatto e di renderle modificabili.

Relazione d'uso con altri moduli

- La classe è utilizzata da:

  `it.hourglass.myTalk.client.presenter.EditProfilePresenter`

- La classe utilizza:

  `it.hourglass.myTalk.client.shared.User`

  `it.hourglass.myTalk.client.shared.Profile`

  `it.hourglass.myTalk.client.presenter.ProfileManagment.PersonalDataProfileManagmentPresenter`

- La classe implementa:

  `it.hourglass.myTalk.client.presenter.display.PersonalDataProfileManagmentDisplay`

Attributi

`User user`
Riferimento all'oggetto User che contiene tutti i nostri dati personali.

`TextBox nameTextBox`
Area di testo per modificare il nome dell'utente.

`TextBox lastNameTextBox`
Area di testo per modificare il cognome dell'utente.

`TextBox emailTextBox`
Area di testo per modificare l'email dell'utente.

`ListBox sexListBox`
Menù a lista che permette di scegliere il sesso dell'utente.

`ListBox dayListBox`
Menù a lista che permette di scegliere il giorno di nascita dell'utente.

`ListBox monthListkBox`
Menù a lista che permette di scegliere il mese di nascita dell'utente.

`ListBox yearListBox`
Menù a lista che permette di scegliere l'anno di nascita dell'utente.

`PasswordTextBox passwordTextBox`
Area di testo per modificare la password dell'utente.

`PasswordTextBox confirmPasswordTextBox`
Area di testo per confermare la password dell'utente.

`Label error`
Label per stampare a video eventuali errori riscontrati sulla modifica dei dati.

`AbsolutePanel contenent`
Pannello che contiene i vari pannelli che racchiuderanno i dati personali principali.

`AbsolutePanel name_LastName_EmailPanel`
Pannello contenente tre label contenenti l'indicazione che al loro fianco si troveranno i campi del nome, cognome ed email dell'utente.

`AbsolutePanel dataName_LastName_EmailPanel`
Pannello contenente le aree di testo modificabili per il nome, cognome e la email.

`AbsolutePanel sex_BirthDate_PasswordPanel`
Pannello contenente due label contenenti l'indicazione che al loro fianco si troveranno i campi del sesso dell'utente, della sua data di nascita e della sua password.

`AbsolutePanel dataSex_BirthDate_PasswordPanel`
Pannello contenente le aree di testo modificabili per il sesso, la data di nascita e la password.

`AbsolutePanel changeButtonPanel`
Pannello contenente il bottone `changeButton` se sono sulla view della visualizzazione dei dati altrimenti contenente il bottone `confirmButton` se sono sulla view della modifica dei dati

`Button changeButton`
Bottone per cambiare la view e rendere i dati personali modificabili.

`Button confirmButton`
Bottone per confermare le modifiche apportate sui dati principali.

Metodi

`+ PersonalDataProfileView()`
Costruttore della classe. Si occupa di inizializzare i campi dati necessari della classe e di disporre le sue componenti.

`- void visualizeView()`
Questo metodo si occupa di cambiare la view in modo da rendere i campi dati da modificabili a solo visualizzabili.

`- void changeView()`
Questo metodo richiede il cambio di view passando da quella con i dati solamente visualizzati a quella per poter modificare i dati.

`+ void clearLabel()`
Metodo invocato dal metodo `changeView()` e serve per pulire i due pannelli della pagina dove è possibile solamente visualizzare i dati.

`+ void clearTextBox()`
Metodo invocato dal metodo `visualize()` e serve per pulire i due pannelli della pagina dove è possibile modificare i dati.

`+ HasValue<String> getNameTextBox()`
Metodo che ritorna la variabile `nameTextBox`.

`+ HasValue<String> getLastNameTextBox()`
Metodo che ritorna la variabile `lastNameTextBox`.

`+ HasValue<String> getEmailTextBox()`
Metodo che ritorna la variabile `emailTextBox`.

`+ String getSexListBox()`
Metodo che ritorna il sesso selezionato dal menù a lista della variabile `sexListBox`.

`+ String getDayListBox()`
Metodo che ritorna il giorno della data di nascita selezionato dal menù a lista della variabile `dayListBox`.

`+ String getMonthListBox()`
Metodo che ritorna il mese della data di nascita selezionato dal menù a lista della variabile `monthListBox`.

`+ String getYearListBox()`
Metodo che ritorna l'anno della data di nascita selezionato dal menù a lista della variabile `yearListBox`.

`+ HasValue<String> getpasswordTextBox()`
Metodo che ritorna la variabile `passwordTextBox`.

`+ HasValue<String> getConfirmPasswordTextBox()`
Metodo che ritorna la variabile `confirmPasswordTextBox`.

`+ HasClickHandlers getChangeButton()`
Metodo che ritorna il bottone `changeButton`.

`+ HasClickHandlers getConfirmButton()`
Metodo che ritorna il bottone `confirmButton`.

`+ AbsolutePanel getView()`
Metodo che ritorna questo widget.

`+ void change()`
Metodo che invoca il metodo `+ void changeView()` .

`public void visualize()`
Metodo per ritornare, una volta modificati i dati personali, alla view di visualizzazione di quest'ultimi.

`+ void showError(String error)`
Metodo che permette di stampare a video eventuali errori riscontrati quando si modificano i dati.

2.12 Package it.hourglass.myTalk.client.rpcservice

Contiene le definizioni di metodi e valori di ritorno asincroni che rendono possibili le chiamate asincrone tra *client* e *server*.

2.12.1 client.rpcservice.RPCService - Interfaccia

Funzione

Espone tutti i metodi che verranno poi implementati dalla corrispettiva *servlet* e saranno quindi disponibili per essere eseguiti come chiamate asincrone remote. Vengono quindi definiti parametri di ingresso e valori di ritorno, che dovranno essere rispettati dall'implementazione lato server.
I metodi sottostanti saranno descritti dettagliatamente nella classe RPCServiceImpl (sezione 2.16.1) che implementa questa interfaccia.

Relazione d'uso con altri moduli

- La classe è utilizzata da:

  `it.hourglass.myTalk.server.model.businesslogic.RPCServiceImpl`

  `it.hourglass.myTalk.client.appcontroller.AppController`

  `it.hourglass.myTalk.client.presenter.BaseViewPresenter`

  `it.hourglass.myTalk.client.presenter.ContactManagementPresenter`

  `it.hourglass.myTalk.client.presenter.ContactProfilePresenter`

  `it.hourglass.myTalk.client.presenter.FriendContactPresenter`

  `it.hourglass.myTalk.client.presenter.FriendMessagePresenter`

  `it.hourglass.myTalk.client.presenter.LoginPresenter`

  `it.hourglass.myTalk.client.presenter.PasswordRecoverPresenter`

  `it.hourglass.myTalk.client.presenter.PopupMessagePresenter`

  `it.hourglass.myTalk.client.presenter.RegistrationPresenter`

  `it.hourglass.myTalk.client.presenter.SignedMenuPresenter`

  `it.hourglass.myTalk.client.presenter.TextMessagePresenter`

  `it.hourglass.myTalk.client.shared.Profile`

  `it.hourglass.myTalk.server.model.businesslogic`

- La classe implementa:

  `com.google.gwt.user.client.rpc.RemoteService`

Metodi

```
+ String register(User user)
+ String storeUser(User user, boolean encryption)
+ String checkSession()
```

```
+ Boolean checkLogin(String userName, String password)
+ SignIn signIn(String uniqueId)
+ User fetchFriendProfile(String uniqueId)
+ String setValidation(String email)
+ Boolean checkValidation(String uniqueId, String validation)
+ Boolean setPassword(String uniqueId, String password)
+ Boolean resetSession()
+ String sendMessage(Message mess)
+ String deleteMessage(Message mess)
+ List<Message> refreshMessageList(String uniqueId)
+ String removeFriend(String userId, String friendId)
+ Boolean friendshake(Message request, boolean accepted)
```

2.12.2 client.rpcservice.RPCServiceAsync - Interfaccia

Funzione

Interfaccia richiesta per la corretta implementazione dello strumento *RPC* di *GWT*. Consente la corretta associazione tra il metodo chiamato dal *client* utilizzando la definizione in `client.rpcservice.RPCService` e il tipo corretto di `callback` ad esso associato, di ritorno dalla *servlet* relativa.

Relazione d'uso con altri moduli

- La classe è utilizzata da:

  `it.hourglass.myTalk.client.appcontroller.AppController`

  `it.hourglass.myTalk.client.presenter.BaseViewPresenter`

  `it.hourglass.myTalk.client.presenter.ContactManagementPresenter`

  `it.hourglass.myTalk.client.presenter.ContactProfilePresenter`

  `it.hourglass.myTalk.client.presenter.FriendContactPresenter`

  `it.hourglass.myTalk.client.presenter.FriendMessagePresenter`

  `it.hourglass.myTalk.client.presenter.LoginPresenter`

  `it.hourglass.myTalk.client.presenter.PasswordRecoverPresenter`

  `it.hourglass.myTalk.client.presenter.PopupMessagePresenter`

  `it.hourglass.myTalk.client.presenter.RegistrationPresenter`

  `it.hourglass.myTalk.client.presenter.SignedMenuPresenter`

  `it.hourglass.myTalk.client.presenter.TextMessagePresenter`

  `it.hourglass.myTalk.client.presenter.profilemanagement.NewEmailConfirmPopupPresenter`

  `it.hourglass.myTalk.client.presenter.profilemanagement.PersonalDataProfileManagementPresenter`

  `it.hourglass.myTalk.client.shared.Profile`

- La classe implementa:

  `com.google.gwt.user.client.rpc.AsyncCallback`

Metodi

```
+ void storeUser(User user, boolean encryption, AsyncCallback<String> callback)
+ void register(User user, AsyncCallback<String> callback)
+ void checkSession(AsyncCallback<String> callback)
+ void checkLogin(String userName, String password, AsyncCallback<Boolean>
  callback)
+ void signIn(String uniqueId, AsyncCallback<SignIn> callback)
```

```
+ void fetchFriendProfile(String uniqueId, AsyncCallback<User> callback)
+ void setValidation(String email, AsyncCallback<String> callback)
+ void checkValidation(String uniqueId, String validation,
  AsyncCallback<Boolean> callback)
+ void setPassword(String uniqueId, String password, AsyncCallback<Boolean>
  callback)
+ void resetSession(AsyncCallback<Boolean> callback)
+ void sendMessage(Message mess, AsyncCallback<String> callback)
+ void deleteMessage(Message mess, AsyncCallback<String> callback)
+ void refreshMessageList(String uniqueId, AsyncCallback<List<Message» callback)
+ void friendshake(Message request, boolean accepted, AsyncCallback<Boolean>
  callback)
+ void removeFriend(String userId, String friendId, AsyncCallback<String>
  callback)
```

2.13 Package it.hourglass.myTalk.client.shared

In questo package saranno contenute tutte le classi che risulteranno necessarie nell'interscambio di informazioni tra applicativo e server.

2.13.1 client.shared.Friendships - Classe

Funzione

Tale classe modella, in forma *POJO*, la relazione di amicizia che intercorre tra due Contatti. Oltre ad un intero rappresentante un indice biunivoco per indicare il rapporto d'amicizia, vengono memorizzati gli identificativi univoci associati agli account degli utenti interessati dalla relazione e un campo `Boolean` indicante la provvisorietà o meno della relazione. Nel momento in cui avverrà un'accettazione dell'amicizia il valore del suddetto campo è progettato per essere settato a `true`

Relazione d'uso con altri moduli

- La classe è utilizzata da:

  `it.hourglass.myTalk.server.model.businesslogic.RPCServiceImpl`

  `it.hourglass.myTalk.server.model.dao.DAO`

  `it.hourglass.myTalk.server.model.dao.DAOImpl`

- La classe implementa:

  `java.io.Serializable`

Attributi

- `Integer friendshipId`
  Identificativo univoco dell'oggetto.
- `String friend1`
  Identificativo univoco associato all'utente 1.
- `String friend2`
  Identificativo univoco associato all'utente 2.
- `Boolean accepted`
  Stato di accettazione dell'amicizia. Determina la sua provvisorietà.

Metodi

+ `Friendships()`
  Costruttore standard della classe.
+ `Friendships(String friend1, String friend2)`
  Costruttore della classe richiedente i due identificativi univoci.
+ `Friendships(String friend1, String friend2, Boolean accepted)`
  Costruttore della classe richiedente i due identificativi univoci e l'indicatore di accettazione dell'amicizia.

+ `Integer getFriendshipId()`
  Metodo che ritorna l'identificativo univoco del rapporto d'amicizia
+ `void setFriendshipId(Integer friendshipId)`
  Metodo per modificare l'identificativo univoco del rapporto d'amicizia.
+ `String getFriend1()`
  Ritorna una stringa indicante l'identificativo univoco di uno dei due utenti (1)
+ `void setFriend1(String friend1)`
  Modifica l'identificativo univoco di uno dei due utenti (1)
+ `String getFriend2()`
  Ritorna una stringa indicante l'identificativo univoco di uno dei due utenti (2)
+ `void setFriend1(String friend2)`
  Modifica l'identificativo univoco di uno dei due utenti (2)
+ `Boolean getAccepted()`
  Ritorna il valore indicante l'accettazione dell'amicizia: `true` se l'amicizia risulta accettata (definitiva), `false` altrimenti.
+ `void setAccepted(Boolean accepted)`
  Imposta il valore indicante l'accettazione dell'amicizia: `true` se l'amicizia risulta accettata (definitiva), `false` altrimenti.

2.13.2 client.shared.Message - Classe

Funzione

Costituisce un messaggio di testo che possa essere persistito nel database. E' costituito da un identificativo univoco, un mittente, un destinatario, la data di creazione, il corpo del messaggio e un indicatore che vada a distinguere se si tratta di un messaggio normale oppure una richiesta d'amicizia.

Relazione d'uso con altri moduli

- La classe è utilizzata da:

  `it.hourglass.myTalk.client.appcontroller.AppController`

  `it.hourglass.myTalk.client.presenter.ContactManagementPresenter`

  `it.hourglass.myTalk.client.presenter.FriendMessagePresenter`

  `it.hourglass.myTalk.client.presenter.MessagePresenter`

  `it.hourglass.myTalk.client.presenter.PopupMessagePresenter`

  `it.hourglass.myTalk.client.presenter.TextMessagePresenter`

  `it.hourglass.myTalk.client.profile.Profile`

  `it.hourglass.myTalk.client.rpcservice.RPCService`

  `it.hourglass.myTalk.client.rpcservice.RPCServiceAsync`

  `it.hourglass.myTalk.client.shared.SignIn`

  `it.hourglass.myTalk.client.view.FriendMessageView`

  `it.hourglass.myTalk.client.view.TextMessageView`

  `it.hourglass.myTalk.server.model.dao.DAO`

  `it.hourglass.myTalk.server.model.dao.DAOImpl`

  `it.hourglass.myTalk.server.model.businesslogic`

  `it.hourglass.myTalk.server.model.businesslogic.RPCServiceImpl`

Attributi

- `Integer messageId`
  Identificativo univoco del messaggio.
- `String sender`
  Identificativo univoco del mittente.
- `String recipient`
  Identificativo univoco del destinatario.
- `Date created`
  Data di creazione del messaggio.
- `String content`
  Contenuto testuale del messaggio.

- `Boolean friendReq`
  Determina se il messaggio in questione è o meno una richiesta d'amicizia.

Metodi

+ `Message()`
  Costruttore della classe.
+ `Message(String sender, String recipient, String content)`
  Costruttore della classe.
+ `Message(String sender, String recipient, Date created)`
  Costruttore della classe.
+ `Message(String sender, String recipient, Date created, String content, Boolean friendReq)`
  Costruttore della classe completo.
+ `Integer getMessageId(`
  Restituisce l'identificatore univoco del messaggio.
+ `void setMessageId(Integer messageId)`
  Modifica l'identificatore univoco del messaggio.
+ `String getSender()`
  Restituisce l'identificativo univoco del mittente.
+ `void setSender(String sender)`
  Modifica il mittente ricevendo in parametro il nuovo identificativo univoco.
+ `String getRecipient()`
  Restituisce l'identificativo univoco del destinatario.
+ `void setRecipient(String recipient)`
  Modifica il destinatario ricevendo in parametro il nuovo identificativo univoco.
+ `Date getCreated()`
  Restituisce la data di creazione del messaggio.
+ `void setCreated(Date created)`
  Modifica la data di creazione del messaggio.
+ `String getContent()`
  Restituisce il contenuto del messaggio.
+ `void setContent(String content)`
  Modifica il contenuto del messaggio.
+ `Boolean getFriendReq()`
  Restituisce un booleano che ha valore `true` se il messaggio è una richiesta d'amicizia, `false` altrimenti.
+ `void setFriendReq(Boolean friendReq)`
  Modifica lo status di richiesta d'amicizia di un messaggio.

2.13.3 client.shared.Profile - Classe

Funzione

Classe che mantiene i dati relativi all'Utente principale garantendone coerenza e accessibilità in tutte le componenti dell'applicativo. Mette a disposizione metodi per il loro immediato recupero e per la memorizzazione di alcune classi utilizzate per il recupero e la persistenza dei dati durante lo svolgersi dell'attività dell'Utente. Ogni altra classe del client può avere accesso ad esse da un unico punto d'accesso. Al fine di ottenere tutte le caratteristiche appena elencate, la classe presenta una struttura calcata sul *design pattern Singleton*.

Relazione d'uso con altri moduli

- La classe è utilizzata da:

  `it.hourglass.myTalk.client.view.profilemanagement.PersonalDataProfileView`

  `it.hourglass.myTalk.client.appcontroller.AppController`

  `it.hourglass.myTalk.client.presenter.BaseViewPresenter`

  `it.hourglass.myTalk.client.presenter.CallPresenter`

  `it.hourglass.myTalk.client.presenter.ContactListPresenter`

  `it.hourglass.myTalk.client.presenter.ContactManagementPresenter`

  `it.hourglass.myTalk.client.presenter.ContactProfilePresenter`

  `it.hourglass.myTalk.client.presenter.FriendContactPresenter`

  `it.hourglass.myTalk.client.presenter.FriendMessagePresenter`

  `it.hourglass.myTalk.client.presenter.LoginPresenter`

  `it.hourglass.myTalk.client.presenter.MessagePresenter`

  `it.hourglass.myTalk.client.presenter.PopupMessagePresenter`

  `it.hourglass.myTalk.client.presenter.TextMessagePresenter`

  `it.hourglass.myTalk.client.presenter.UserListPresenter`

  `it.hourglass.myTalk.client.presenter.ProfileManagment.AvatarProfileManagmentPresenter`

  `it.hourglass.myTalk.client.presenter.profilemanagement.ExtendedDataProfileManagmentPresenter`

  `it.hourglass.myTalk.client.presenter.profilemanagement.NewAvatarPopupPresenter`

  `it.hourglass.myTalk.client.presenter.profilemanagement.NewEmailConfirmPopupPresenter`

  `it.hourglass.myTalk.client.presenter.profilemanagement.PersonalDataProfileManagmentPresenter`

`it.hourglass.myTalk.client.view.profilemanagement.AvatarProfileManagmentView`

`it.hourglass.myTalk.client.view.profilemanagement.ExtendedDataProfileManagmentView`

`it.hourglass.myTalk.client.view.profilemanagement.PersonalDataProfileView`

- La classe utilizza:

  `it.hourglass.myTalk.client.wrappers.PeerConnectionCallbacks`

  `it.hourglass.myTalk.client.communication.wsconnection.CallMessageCallback`

  `it.hourglass.myTalk.client.communication.wsconnection`

  `it.hourglass.myTalk.client.communication.wsmessageBuilder`

  `it.hourglass.myTalk.client.rpcservice.RPCService`

  `it.hourglass.myTalk.client.rpcservice.RPCServiceAsync`

  `it.hourglass.myTalk.client.shared.User`

  `it.hourglass.myTalk.client.view.PopupInformation`

Attributi

- `static Profile instance`
  Contiene l'unica istanza concessa di oggetto Profile.
- `static User user`
  Puntatore all'oggetto User memorizzato.
- `static HashMap<String, Boolean> friendlist`
  Lista contenente gli identificativi univoci associati agli account dei contatti amici dell'utente.
- `static List<Message> messagelist`
  Lista contenente i propri messaggi.
- `static WSConnection ws`
  Variabile il cui scopo è mantenere i dati e i metodi necessari a stabilire una connessione col *server WS*.

Metodi

- `Profile()`
  Costruttore della classe `Profile`. E' volutamente definito private per far adottare le connotazioni del pattern *Singleton* alla classe. Così facendo, si assicura l'unicità dell'oggetto e la coerenza nell'utilizzo delle componenti che lo compongono.
- `+ static synchronized Profile getInstance()`
  Ritorna un'istanza dell'oggetto. Viene controllato se l'oggetto non sia stato ancora creato ad opera di altre componenti. In tal caso viene costruito e si ritorna un puntatore ad esso.

+ `static void setUser(User setUser)`
  Permette la sostituzione dell'elemento `User` della classe.
+ `static void setFriendList(HashMap<String, Boolean> setFriendlist)`
  Permette la sostituzione dell'elemento `friendlist` della classe. Utilizzato per associare al Profilo una lista `HashMap` di identificativi univoci di utenti amici con associato il loro status.
+ `static void setMessageList(List<Message> setMessagelist)`
  Permette la sostituzione dell'elemento `messagelist` della classe. Utilizzato per associare al Profilo una lista `List<Message>` di messaggi il cui destinatario risulti essere l'utente associato al profilo.
+ `static User getUser()`
  Ritorna un puntatore all'oggetto `User` associato al Profile.
+ `static void saveUser(boolean encrypt)`
  Stabilisce una *RPC* così da consentire il salvataggio dell'`User`. Il campo `encrypt` è utilizzato per indicare se la `password` dell'oggetto debba essere criptata o meno.
+ `static HashMap<String, Boolean> getFriendList()`
  Ritorna un puntatore all'oggetto `HashMap friendlist` associato al `Profile`.
+ `static List<Message> getMessageList()`
  Ritorna un puntatore all'oggetto `List<Message> messagelist` associato al `Profile`.
+ `static void refreshMessageList()`
  Consente l'aggiornamento della lista dei messaggi con destinatario l'utente associato al profilo. Viene instaurata una *RPC* per consentire il prelievo di una lista di messaggi aggiornata direttamente dal *DB*. Il ritorno con successo della *RPC* stessa è un oggetto `List<Message>` adatto ad essere assegnato al puntatore `messagelist`.
+ `static void removeMessage(Message msg)`
  Rimuove un messaggio dalla lista di messaggi `messagelist` con destinatario l'utente.
+ `static void addFriend(String friendId)`
  Consente l'inserimento di un nuovo contatto nella lista di contatti amici dell'utente associato al `Profile`. Viene inserita una nuova voce nella lista `friendlist` del profilo. Successivamente una notifica è inoltrata al server *WS* così che l'utente aggiunto sia a sua volta notificato dell'operazione.
+ `static void removeFriend(String friendId)`
  Rimuove un contatto dalla lista di contatti amici `friendlist` associata all'utente.
+ `static void friendReqNotificationWS(String recipient)`
  Inoltra una notifica al server del fatto che sia stata generata una richiesta d'amicizia. Nel momento in cui l'utente si trova a generare un messaggio legato a una richiesta d'amicizia rivolto ad un altro utente, si invia notizia al server *WS*, così che il destinatario ne sia notificato.

`+ static void sentMessageNotificationWS(String recipient)`
Inoltra una notifica al server del fatto che è stato generato un messaggio. Nel momento in cui l'utente si trova a generare un messaggio rivolto ad un altro utente, si invia notizia al server *WS*, così che il destinatario ne sia notificato.

`+ static void setWS(WSConnection s)`
Consente di memorizzare una connessione a server *WebSocket*.

`+ static void sendFriendlistWS()`
Costruisce un array degli amici dell'utente legato al profilo da inviare al server *WebSocket*.

`- static RPCServiceAsync getService()`
Ricava una connessione per stabilire una *RPC*.

2.13.4 client.shared.SignIn - Classe

Funzione

Classe che funge da contenitore di classi multiple. Il suo scopo è quello di consentire il passaggio di tutte le informazioni utili di un Utente mediante un'unica transazione. Ciascuna struttura in essa contenuta avrà poi scopi diversi ed indipendenti, e sarà trattata in maniera autonoma.

Relazione d'uso con altri moduli

- La classe è utilizzata da:

  `it.hourglass.myTalk.client.appcontroller.AppController`

  `it.hourglass.myTalk.client.rpcservice.RPCService`

  `it.hourglass.myTalk.client.rpcservice.RPCServiceAsync`

  `it.hourglass.myTalk.model.rpcservice.RPCServiceImpl`

- La classe utilizza:

  `it.hourglass.myTalk.client.shared.Message`

  `it.hourglass.myTalk.client.shared.User`

- La classe implementa:

  `java.io.Serializable`

Attributi

- `User user`
  Variabile costituita da un riferimento a un oggetto di tipo `User`.
- `HashMap<String, Boolean> friendlist`
  Lista di identificativi univoci associati ai contatti amici dell'utente, ciascuno associato ad un valore booleano rappresentate lo status.
- `List<Message> messagelist`
  Lista di messaggi.

Metodi

+ `SignIn()`
  Costruttore standard della classe.
+ `SignIn(User user, HashMap<String, Boolean> friendlist, List<Message> messagelist)`
  Secondo costruttore della classe. Assegna il valore alle tre variabili descritte sopra grazie ai parametri formali.
+ `SignIn(User user)`
  Terzo costruttore della classe. Assegna il valore del parametro formale alla variabile `user` e inizializza la variabile `friendlist`.

`+ SignIn(User user, List<Message> messagelist)`
Quarto costruttore della classe. Assegna il valore dei parametri formali alle variabili `user` e `messagelist` e inizializza la variabile `friendlist`.

`+ User getUser()`
Metodo che ritorna il puntatore `user`.

`+ HashMap<String, Boolean> getFriendList()`
Metodo che ritorna il puntatore `friendlist`.

`+ List<Message> getMessageList()`
Metodo che ritorna il puntatore `messagelist`.

2.13.5 client.shared.User - Classe

Funzione

Lo scopo di questa classe è mantenere tutti i dati personali che riguardino il profilo associato ad un utente, oltre ad offrire dei metodi per la modifica e il recupero degli stessi.

Relazione d'uso con altri moduli

- La classe è utilizzata da:

  it.hourglass.myTalk.client.appcontroller.AppController

  it.hourglass.myTalk.client.presenter.ContactProfilePresenter

  it.hourglass.myTalk.client.presenter.RegistrationPresenter

  it.hourglass.myTalk.client.presenter.profilemanagement.
  AvatarProfileManagmentPresenter

  it.hourglass.myTalk.client.presenter.profilemanagement.
  ExtendedDataProfileManagmentPresenter

  it.hourglass.myTalk.client.presenter.profilemanagement.
  NewAvatarPopupPresenter

  it.hourglass.myTalk.client.presenter.profilemanagement.
  NewEmailConfirmPopupPresenter

  it.hourglass.myTalk.client.presenter.profilemanagement.
  PersonalDataProfileManagmentPresenter

  it.hourglass.myTalk.client.rpcservice.RPCService

  it.hourglass.myTalk.client.rpcservice.RPCServiceAsync

  it.hourglass.myTalk.client.shared.Profile

  it.hourglass.myTalk.client.shared.SignIn

  it.hourglass.myTalk.client.view.ContactProfileView

  it.hourglass.myTalk.client.view.contactprofile.AvatarContactProfileView

  it.hourglass.myTalk.client.view.contactprofile.ExtendedDataContactProfileView

  it.hourglass.myTalk.client.view.contactprofile.PersonalDataContactProfileView

  it.hourglass.myTalk.client.view.profilemanagement.AvatarProfileManagementView

  it.hourglass.myTalk.client.view.profilemanagement.
  ExtendedDataProfileManagementView

  it.hourglass.myTalk.client.view.profilemanagement.PersonalDataProfileView

  it.hourglass.myTalk.server.model.dao.DAO

  it.hourglass.myTalk.server.model.dao.DAOImpl

  it.hourglass.myTalk.server.model.businessmodel.RPCServiceImpl

Attributi

- `String uniqueId`
  Stringa contenente l'identificativo univoco dell'utente.
- `String email`
  Stringa contenente l'email dell'utente.
- `String password`
  Stringa contenente la password dell'utente.
- `String altEmail`
  Stringa contenente l'email alternativa dell'utente.
- `String avatar`
  Stringa contenente l'indirizzo dell'avatar dell'utente.
- `String name`
  Stringa contenente il nome dell'utente.
- `String lastName`
  Stringa contenente il cognome dell'utente.
- `Date birthdate`
  Stringa contenente la data di nascita dell'utente.
- `char gender`
  Carattere che indica il sesso dell'utente.
- `String hometown`
  Stringa contenente il nome della città d'origine dell'utente.
- `String currentLocation`
  Stringa contenente il luogo in cui l'utente si trova attualmente.
- `String description`
  Stringa contenente una breve descrizione dell'utente.
- `String inspirations`
  Stringa contenente le ispirazioni dell'utente.
- `String interests`
  Stringa contenente gli interessi dell'utente.
- `String validation`
  Stringa utilizzata dal sistema nei casi in cui sia richiesta una convalida via email, associata ad uno specifico account.
- `Boolean enabled`
  Stato di abilitazione al login dell'account dell'utente.
- `String rapidCall`
  Stringa utilizzata per memorizzare contatti amici nelle chiamate da menù dei contatti.

- `String viewFriendProfile`
  Stringa utilizzata per memorizzare identificativi univoci per le chiamate da menù dei contatti.

Metodi

+ `User()`
  Costruttore standard della classe.
+ `User(String uniqueId, String email, String password, String name, String lastName, Date birthdate, char gender)`
  Secondo costruttore della classe. Inizializza le dovute variabili per mezzo dei parametri formali.
+ `User(String uniqueId, String email, String password, String altEmail, String avatar, String name, String lastName, Date birthdate, char gender, String hometown, String currentLocation, String description, String inspirations, String interests, String validation, Boolean enabled)`
  Terzo costruttore della classe. Inizializza tutte le variabili per mezzo dei parametri formali.
+ `String getUniqueId()`
  Ritorna la variabile `uniqueId`.
+ `void setUniqueId(String uniqueId)`
  Imposta la variabile `uniqueId`.
+ `String getEmail()`
  Ritorna la variabile `email`.
+ `String setEmail()`
  Imposta la variabile `email`.
+ `String getPassword()`
  Metodo che ritorna la variabile `password`.
+ `String setPassword()`
  Imposta la variabile `password`.
+ `String getAltEmail()`
  Ritorna la variabile `altEmail`.
+ `void setAltEmail(String altEmail)`
  Imposta la variabile `altEmail`.
+ `String getAvatar()`
  Ritorna la variabile `avatar`.
+ `void setAvatar(String avatar)`
  Imposta la variabile `avatar`.
+ `String getName()`
  Ritorna la variabile `name`.

+ `void setName(String name)`
  Imposta la variabile `name`.
+ `String getLastName()`
  Ritorna la variabile `lastname`.
+ `void setLastName(String lastName)`
  Imposta la variabile `lastname`.
+ `Date getBirthdate()`
  Ritorna la variabile `birthdate`.
+ `void setBirthdate(String lastName)`
  Imposta la variabile `birthdate`.
+ `char getGender()`
  Ritorna la variabile `gender`.
+ `void setGender(char gender)`
  Imposta la variabile `gender`.
+ `String getHometown()`
  Ritorna la variabile `hometown`.
+ `void setHometown(String hometown)`
  Imposta la variabile `hometown`.
+ `String getCurrentLocation()`
  Ritorna la variabile `currentLocation`.
+ `void setCurrentLocation(String currentLocation)`
  Imposta la variabile `currentLocation`.
+ `String getDescription()`
  Metodo che ritorna la variabile `description`.
+ `void setDescription(String description)`
  Imposta la variabile `description`.
+ `String getInspirations()`
  Ritorna la variabile `inspirations`.
+ `void setInspirations(String inspirations)`
  Imposta la variabile `inspirations`.
+ `String getInterests()`
  Metodo che ritorna la variabile `interests`.
+ `void setInterests(String interests)`
  Imposta la variabile `interests`.
+ `String getValidation()`
  Ritorna la variabile `validation`.
+ `void setValidation(String validation)`
  Imposta la variabile `validation`.

+ `Boolean getEnabled()`
  Ritorna la variabile `enabled`.
+ `void setEnabled(Boolean enabled)`
  Imposta la variabile `enabled`.
+ `void void setRapidCall(String rapid)`
  Imposta la variabile `rapidCall`.
+ `void setViewFriendProfile(String rapid)`
  Imposta la variabile `viewFriendProfile`.
+ `String getRapidCall()`
  Ritorna la variabile `rapidCall`.
+ `String getViewFriendProfile()`
  Ritorna la variabile `viewFriendProfile`.

2.14 Package it.hourglass.myTalk.client.wrappers

Questo package contiene la definizione delle classi wrapper delle varie funzioni di webRTC e viene utilizzato dalle classi che utilizzano le librerie RTC.

2.14.1 client.wrappers.ConsoleLog - Classe

Questa classe è il wrap dell'oggetto console Javascript e delle sue funzioni e permette di stampare stringhe nella console del browser.

Relazioni d'uso con altri moduli Questa classe verrà utilizzata dalle componenti che vorranno visualizzare una stringa sulla console del browser.

2.14.2 client.wrappers.MediaStream - Classe

Questa classe è il wrap per rappresentare lo stream in Javascript.

Relazioni d'uso con altri moduli

- La classe è utilizzata da:

  `it.hourglass.myTalk.client.communication.Call`.

2.14.3 client.wrappers.GetUserMediaUtils - Classe

Questa classe è il wrap delle funzioni di webRTC preposte alla cattura dello stream audio/video della periferica di acquisizione dell'utente. Le attività che svolge è di chiedere all'utente l'autorizzazione per avere accesso allo stream. Una volta concessa, viene restituito un riferimento allo stream di tipo MediaStream.

Relazioni d'uso con altri moduli

- La classe è utilizzata da:

  `it.hourglass.myTalk.client.communication.Call`

2.14.4 client.wrappers.PeerConnectionWrapper - Classe

Questa classe è il wrap delle funzioni di webRTC preposte alla gestione della connessione tra utenti, lo scambio dei loro stream multimediali e la gestione dello stream di dati, utilizzato nel nostro caso per la chat. Gestisce quindi tutte le possibili situazioni in una chiamata, ovvero:

- Inizio chiamata
- Fine chiamata
- Scambio di informazioni
- Canale di chat

Relazioni d'uso con altri moduli

- La classe è utilizzata da:

  `it.hourglass.myTalk.client.communication.Call`

2.14.5 client.wrappers.RTCConfiguration - Classe

Questa classe è il wrap delle funzioni di webRTC preposte alla gestione della connessione con il server ICE. Si occupa di instaurare una connessione al server ICE, ritornando la risposta di quest'ultimo.

Relazioni d'uso con altri moduli

- La classe è utilizzata da:

  `it.hourglass.myTalk.client.communication.Call`

2.14.6 client.wrappers.PeerConnectionCallbacks - Interfaccia

Questa interfaccia espone i metodi che verranno utilizzati per la chiamata e che andranno successivamente implementati da `Call`.

Relazioni d'uso con altri moduli

- L'interfaccia è utilizzata da:

  `it.hourglass.myTalk.client.communication.Call`

2.14.7 client.wrappers.RTCSessionDescription - Classe

Questa classe è il wrap del valore dell'SDP di ciascun client.

Relazioni d'uso con altri moduli

- La classe è utilizzata da:

  `it.hourglass.myTalk.client.communication.Call`

2.14.8 client.wrappers.mozRTCSessionDescription - Classe

Questa classe è il wrap del valore dell'SDP di ciascun client. Utilizzato su browser Firefox.

Relazioni d'uso con altri moduli

- La classe è utilizzata da:

  `it.hourglass.myTalk.client.communication.Call`

2.14.9 client.wrappers.SDPCreateOfferCallback - Interfaccia

Questa classe è wrap delle funzioni Javascript utilizzate per la creazione delle offerte in base al tipo di stream scelto, da mandare poi all'utente ricevente della telefonata.

Relazioni d'uso con altri moduli

- L'interfaccia è utilizzata da:

  `it.hourglass.myTalk.client.communication.Call`

2.15 Package it.hourglass.myTalk.server

Tale package funge da controparte del package `it.hourglass.myTalk.client`. In esso sono contenute tutte le classi che trovano il loro spazio di esecuzione in remoto. Ad esse sono quindi demandate funzionalità di persistenza, recupero ed elaborazione dei dati, nonché tutta la logica che rende possibile l'instaurazione di una chiamata tra due utenti, il monitoraggio degli stati degli stessi e l'inoltro di richieste tra un utente e un altro. Le descrizioni dei singoli package seguenti offriranno una visione più in dettaglio di quanto appena descritto.

2.16 Package it.hourglass.myTalk.server.model

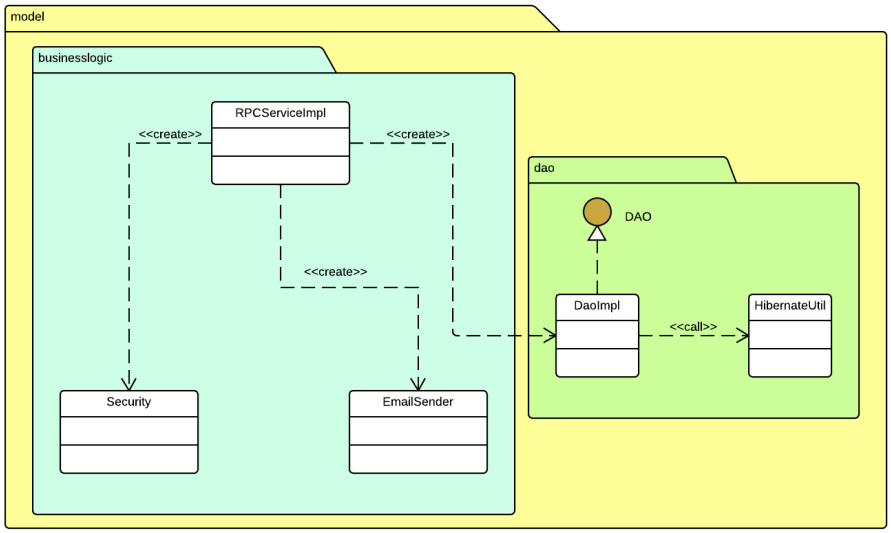

Figura 7: Package it.hourglass.myTalk.server.model

Tale package raggruppa a sua volta package e classi le cui funzionalità sono legate all'elaborazione e al prelievo dal *database* di tutti i dati legati ai profili degli utenti oltre che a offrire gli strumenti adatti alla loro modifica.

2.17 Package it.hourglass.myTalk.server.model.DAO

Package contenente tutte le classi e i file di configurazione utilizzati durante le operazioni di persistenza dei dati del *database*.

2.17.1 server.model.dao.DAO - Interfaccia

Funzione

Interfaccia dell'oggetto `DAO`. L'oggetto `DAO` ha la funzione di fornire dei metodi al resto dell'applicativo che si occupino di offrire degli immediati ed univoci punti d'accesso al *database* e quindi permettere la persistenza dei dati sfruttando il *framework Hibernate*.

Relazione d'uso con altri moduli

- La classe è utilizzata da:

  `it.hourglass.myTalk.server.model.businessmodel.RPCServiceImpl`

  `it.hourglass.myTalk.server.model.dao.DAOImpl`

- La classe utilizza:

  `it.hourglass.myTalk.client.shared.User`

  `it.hourglass.myTalk.client.shared.Message`

  `it.hourglass.myTalk.client.shared.Friendships`

Metodi

```
+ String storeUser(User user)
+ User fetchFriend(String uniqueId)
+ User fetchUserById(String uniqueId)
+ User fetchUserByEmail(String email)
+ String setValidation(String email, String validation)
+ boolean checkValidation(String uniqueId, String validation)
+ List<String> fetchFriendships(String uniqueId)
+ Friendships fetchSpecificFriendship(String friend1, String friend2)
+ List<Message> fetchMessages(String uniqueId)
+ void updateMessage(Message message)
+ void updateFriendship(Friendships friendship)
+ void deleteMessage(Message message)
+ void deleteFriendship(Friendships friendship)
```

2.17.2 server.model.dao.DAOImpl - Classe

Funzione

Classe designata all'implementazione dei metodi esposti dalla rispettiva interfaccia. Con l'utilizzo del *pattern DAO* diviene irrilevante che genere di strumenti si sia deciso di utilizzare per l'implementazione dei metodi stessi (che nella fattispecie, si tratta del *framework Hibernate*). Nel caso in cui in futuro si decidesse di adottare qualche altro strumento od offrire implementazioni alternative sarà sufficiente rispettare le definizioni esposte dall'interfaccia e utilizzate dagli altri applicativi.
Nel caso presente la classe ha lo scopo di fornire metodi utilizzabili dall'applicativo per sfruttare gli strumenti offerti da *Hibernate* automaticamente, senza necessità di conoscerne il funzionamento. Al fine di rendere unico l'accesso e l'utilizzo di tali metodi offerti si è deciso di rendere l'oggetto un *Singleton*. Grazie alla classe in questione, viene implementato il contratto esposto dall'interfaccia generale. Vengono quindi offerti al resto dell'applicativo metodi utilizzabili per effettuare operazioni elementari di lettura e scrittura sul *database*. Tali operazioni sono relative al prelievo e alla memorizzazione di dati degli Utenti, messaggi e rapporti di amicizia. I dati prelevati di volta in volta dalle differenti richieste vengono mappati, per mezzo di appositi file `xml`, in corrispettivi *POJO*, utilizzati poi nel resto dell'applicativo.
Ciascun metodo utilizza strumenti resi disponibili dal framework *Hibernate* per gestire accessi concorrenti, oltre che interpretare query *SQL*.

Relazione d'uso con altri moduli

- La classe è utilizzata da:

  `it.houglass.myTalk.server.model.businesslogic.RPCServiceImpl`

- La classe utilizza:

  `it.hourglass.myTalk.client.shared.Friendships`

  `it.hourglass.myTalk.client.shared.Message`

  `it.hourglass.myTalk.client.shared.User`

  `it.hourglass.myTalk.server.model.businesslogic.Security`

- La classe implementa:

  `it.hourglass.myTalk.server.model.dao.DAO`

Metodi

`- DAOImpl()`
E' reso private per far coincidere la struttura della classe con quella di un *Singleton*. Lo scopo è assicurare che gli accessi al *DB* da parte delle *servlet* siano sempre possibili da un unico punto d'accesso.

`+ static synchronized DAO getInstance()`
Restituisce un'istanza della classe o la crea nel caso questa non sia stata ancora inizializzata.

- `Session getSession()`
  Fornisce una sessione *Hibernate*. Permette l'ottenimento di una sessione valida sfruttando la classe di supporto `HibernateUtil`. Ogni funzionalità di *Hibernate* è raggiungibile attraverso questa.

+ `String storeUser(User user)`
  Persiste i dati di un utente nel database. Ricevuto un oggetto User in parametro, questo viene memorizzato nel *DB* associato. Viene ritornata una stringa che descrive l'esito dell'operazione.

+ `User fetchUserById(String uniqueId)`
  Ritorna un oggetto User dal *DB* cercandolo per `uniqueId`. Fornendo al metodo un identificativo univoco, questo ritorna un oggetto `User` corrispondente al record nel *DB* avente l'identificativo univoco come chiave.

+ `User fetchUserByEmail(String userEmail)`
  Ritorna un oggetto User dal *DB* cercandolo per `email`. Fornita l'email dell'Utente che si desidera trovare, viene ritornato l'oggetto `User` corrispondente alla sua voce nel *DB*.

+ `User fetchFriend(String uniqueId)`
  Ritorna un oggetto `User` dopo averne settato il campo password a stringa vuota. Utilizzato per ricevere dal database un oggetto `User` associato al campo `uniqueId` fornito e privato di password per motivi di sicurezza.

+ `String setValidation(String email, String validation)`
  Attribuisce una stringa al campo `validation` del record associato all' indirizzo email desiderato. Ricevuti in parametro indirizzo email e codice di validazione desiderati, il metodo ricerca il record associato a tale indirizzo e ne modifica il campo `validation`. Ritorna infine a quale identificativo unico il record è associato, o un messaggio d'errore se l'indirizzo email non è associato ad alcun record.

+ `boolean checkValidation(String uniqueId, String validation)`
  Controlla che il codice di validazione inserito sia quello associato all'account richiesto. Fornito un appropriato `uniqueId`, il metodo controlla che esista un account associato a questo e che il codice di validazione inserito sia quello correttamente ad esso associato. Ritorna `true` se ciò accade, `false` altrimenti.

+ `boolean setPassword(String uniqueId, String password)`
  Attribuisce la password desiderata all'account associato all'`uniqueId` fornito. Fornito un identificativo univoco e una password, quest'ultima è attribuita al record avente il relativo identificativo. Se l'account non esiste viene ritornato un valore `false`.

+ `List<String> fetchFriendships(String uniqueId)`
  Ritorna gli identificativi degli amici di un Utente. Fornito l'identificativo univoco associato all'account di un Utente è ritornata una lista (un oggetto di tipo `java.util.List`) di identificativi univoci associati agli account degli amici. Se

non esiste alcun account associato al dato identificativo univoco viene ritornato il valore `null`.

+ `Friendships fetchSpecificFriendship(String friend1, String friend2)`
  Passati due identificativi univoci in parametro, questo metodo ritorna un oggetto di tipo `Friendships` rappresentante il loro rapporto di contatti amici. Il risultato potrà eventualmente essere `null` nel caso in cui non esistesse alcuna voce di relazione tra i due identificativi univoci forniti.

+ `List<Message> fetchMessages(String uniqueId)`
  Ritorna tutti i messaggi con destinatario l'utente associato all'identificativo univoco passato in parametro. Saranno, nello specifico, `List` di `Message` costruiti su record di messaggi destinati all'utente specificato. Se non esistono messaggi associati al dato identificativo univoco viene ritornata una lista vuota.

+ `void updateMessage(Message message)`
  Crea o provoca un *update* del record su cui è costruito l'oggetto `Message` passato in parametro.

+ `void updateFriendship(Friendships friendship)`
  Crea o provoca un *update* del record su cui è costruito l'oggetto `Friendships` passato in parametro.

+ `void deleteMessage(Message message)`
  Elimina dal database il record su cui è costruito l'oggetto `Message` passato in parametro.

+ `void deleteFriendship(Friendships friendship)`
  Elimina dal database il record su cui è costruito l'oggetto `Friendships` passato in parametro.

2.17.3 server.model.dao.HibernateUtil - Classe

Funzione

Classe di inizializzazione delle sessioni di Hibernate. Il suo scopo è quello di fornire una sessione aggiornata e utile all'utilizzo delle funzionalità offerte dal *framework Hibernate*. Così facendo si garantisce una corretta concorrenza e salvaguardia da eventuali conflitti delle transazioni effettuate.

Relazione d'uso con altri moduli

- La classe è utilizzata da:

  `it.houglass.myTalk.model.DAO.DAOImpl`

Attributi

- `- static ServiceRegistry serviceRegistry`
  Consente e regola registrazioni e ottenimenti di un servizio.
- `- static final SessionFactory sessionFactory`
  Oggetto che si occupa della creazione di sessioni. Ogni transazione verso il *database* è regolata in base a una richiesta di un'istanza da esso, rendendo le operazioni auspicabilmente sincronizzate.

Metodi

- `+ static SessionFactory getSessionFactory()`
  Ritorna un riferimento alla `sessionFactory` opportunamente configurata dal costruttore.

2.18 Package it.hourglass.myTalk.server.model.businesslogic

All'interno di tale package saranno contenute tutte le componenti che andranno a costituire (e a fare da supporto) al lato *servlet* dello strumento *RPC* messo a disposizione dal framework *GWT*. Tramite chiamate remote asincrone è possibile, grazie agli strumenti presenti nel pacchetto, soddisfare tutte le richieste riguardanti il prelievo, il salvataggio, il controllo e l'elaborazione dei dati.

2.18.1 server.model.businesslogic.RPCServiceImpl - Classe

Funzione

Tale classe consiste nell'implementazione vera e propria del servizio *Remote Procedure Call* messo a disposizione dal *framework GWT*. In risposta alle chiamate remote asincrone ricevute vengono ritornati i dati richiesti dopo che siano state eseguiti i dovuti controlli ed elaborazioni.

Relazione d'uso con altri moduli

- La classe utilizza:

  `it.hourglass.myTalk.client.rpcservice.RPCService`

  `it.hourglass.myTalk.client.shared.SignIn`

  `it.hourglass.myTalk.client.shared.User`

  `it.hourglass.myTalk.client.shared.Friendships`

  `it.hourglass.myTalk.client.shared.Message`

  `it.hourglass.myTalk.server.model.dao.DAO`

  `it.hourglass.myTalk.server.model.dao.DAOImpl`

- La classe estende:

  `com.google.gwt.user.server.rpc.RemoteServiceServlet`

- La classe implementa:

  `it.hourglass.myTalk.client.RPCService.RPCService`

Attributi

- `DAO DAO`
  Istanza dell'oggetto `DAO` designato alla persistenza dei dati nel *database*.
- `Security tools`
  Istanza dell'oggetto `Security` designato a operazioni di sicurezza sui dati da persistere.

Metodi

+ `String storeUser(User user, boolean encryption)`
  Ricevuto un oggetto `it.hourglass.myTalk.client.shared.User` in parametro, questo metodo utilizza i metodi dell'oggetto `DAO` per permetterne la persistenza

nel *DB*. Viene verificato che non esistano record associati al medesimo indirizzo email dell'oggetto User che deve essere memorizzato. In caso negativo, viene invocato il metodo dell'oggetto `DAO` appropriato alla sua memorizzazione nel *DB*. Nel caso in cui il parametro `encryption` corrisponda al valore `true` il campo `password` viene criptato prima della memorizzazione.

`+ String register(User user)`
Ricevuto un oggetto `it.hourglass.myTalk.client.shared.User` in parametro, questo metodo utilizza i metodi dell'oggetto `DAO` per permetterne la persistenza nel DB. Viene verificato che non esistano record associati al medesimo indirizzo email o `uniqueId` dell'oggetto User che deve essere memorizzato. In caso negativo viene invocato il metodo dell'oggetto `DAO` appropriato alla sua memorizzazione nel *DB* non prima che il campo password sia stato criptato e il campo `validation` settato a `false`. A seguito di tale operazione viene poi inviata un'email all'indirizzo email dell'utente associato all'oggetto User in questione.

`+ Boolean checkLogin(String userName, String password)`
Verifica attraverso il metodo `loginValidation()` che username e password siano validi. Se questo è il caso, crea una nuova sessione. In caso contrario restituisce `false`.

`+ SignIn signIn(String uniqueId)`
Ricevuto uno `uniqueId` in parametro, questo metodo ritorna un oggetto `it.hourglass.myTalk.client.shared.SignIn` opportunamente costruito. L'oggetto ritornato conterrà a sua volta:

- Un oggetto `it.hourglass.myTalk.client.shared.User` associato all'`uniqueId` passato in parametro
- Un oggetto `java.util.HashMap`, composto da coppie chiave-valore di tipo `<String, Boolean>`, di cui ciascuna stringa conterrà un `uniqueId` associato ad un account amico dell'account associato allo `uniqueId` passato in parametro e di cui ciascun `Boolean` sarà inizialmente settato a `false`.
- Un oggetto `java.util.List`, costituente una collezione di oggetti `it.hourglass.myTalk.client.shared.Message` con destinatario l'utente associato al campo `uniqueId` passato in parametro.

Il metodo può infine restituire anche un valore `null` nel caso in cui non esista alcun account associato all'identificativo univoco indicato.

`+ User fetchFriendProfile(String uniqueId)`
Ritorna un oggetto `it.hourglass.myTalk.client.shared.User` costruito sul *record* del *database* associato allo `uniqueId` passato in parametro, la cui password è settata a stringa vuota.

`- boolean loginValidation(String uniqueId, String password)`
Interroga il *DB* per verificare se esista una corrispondenza della coppia di valori `uniqueId` e `password` passati in parametro. Il metodo ritorna un booleano che indica l'esito positivo (`true`) o meno (`false`) dell'operazione.

+ `String setValidation(String email)`
  Si occupa di delegare alle classi specializzate la creazione del codice di sicurezza per il cambio password e di associarlo all'email dell'utente che ha richiesto il servizio. Il metodo infine crea e invia il messaggio email con il codice di sicurezza all' email associata all'utente.

+ `Boolean checkValidation(String uniqueId, String validation)`
  Verifica che la stringa `validation` passata in parametro sia correttamente associata alla voce nel *DB* relativa al campo `uniqueId`.

+ `Boolean setPassword(String uniqueId, String password)`
  Consente la modifica del campo password del record associato all'`uniqueId` passato in parametro. Ritornerà poi l'esito dell'operazione.

+ `String sendMessage(Message mess)`
  Metodo destinato alla memorizzazione nel database di messaggi testuali o d'amicizia. Ricevuto in parametro un oggetto di tipo `it.hourglass.myTalk.client.shared.Message` si determina il tipo di messaggio che deve essere memorizzato. Nel caso questo sia una richiesta di amicizia vengono compiuti dei controlli mediante classe `DAO` così da accertarsi che l'utente destinatario esista, non risulti già amico del mittente o non sia stata già inoltrata una richiesta d'amicizia da parte dello stesso mittente. Se gli esiti sono positivi, un'amicizia non confermata è creata nel *DB*. In caso contrario vengono ritornate delle stringhe d'errore. Sia nel caso che si tratti di una richiesta d'amicizia che di un semplice messaggio, il messaggio stesso è poi persistito nel *database*.

+ `String deleteMessage(Message mess)`
  Rimuove dal database un record associato al messaggio passato in parametro.

+ `List<Message> refreshMessageList(String uniqueId)`
  Ritorna una lista aggiornata dei messaggi associati all'identificativo univoco passato in parametro.

+ `Boolean friendshake(Message request, boolean accepted)`
  Metodo finalizzato all'accettazione di richieste d'amicizia intercorse tra due utenti. L'oggetto `it.hourglass.myTalk.client.shared.Message` associato alla richiesta d'amicizia e l'esito espresso in responso alla stessa sono passati in parametro. Viene preliminarmente verificato che nel tempo intercorso fino alla risposta la richiesta d'amicizia non sia stata in qualche modo eliminata. Se non è stata eliminata allora, a seconda del responso, viene abilitata l'amicizia creata durante l'inoltro della richiesta originale oppure eliminata del tutto l'amicizia temporanea. In entrambi i casi il messaggio relativo è eliminato dal database.

+ `String removeFriend(String user, String friend)`
  Elimina il record rappresentante il rapporto di amicizia instaurato tra due utenti, i cui identificativi univoci sono passati in parametro.

+ `String checkSession()`
  Si occupa di verificare che la sessione con il client sia valida. In caso afferma-

tivo aggiorna lo stato della sessione e ritorna il parametro di sessione `username` altrimenti ritorna `null`.

`+ Boolean resetSession()`
Se esiste una sessione attiva la distrugge. Viene ritornato l'esito dell'operazione.

2.18.2 server.model.businesslogic.Security - Classe

Funzione

Classe che si occupa di fornire varie funzionalità relative alla sicurezza dell'account. Fornisce dei metodi per l'amministrazione delle password e dei codici di validazione associati a un account.

Relazione d'uso con altri moduli

- La classe è utilizzata da:

  `it.hourglass.service.model.businesslogic.RPCServiceImpl`

Attributi

- `final BasicPasswordEncryptor passwordEncryptor`
  Oggetto la cui finalità è la creazione e il controllo di *Message Digest*.
- `UUID`
  Oggetto la cui finalità è la generazione di stringhe di contenuto randomico.

Metodi

+ `String generateValidation()`
+ `String cryptPassword(String plain)`
+ `boolean checkPasswords(String plain, String encrypted)`

2.18.3 server.model.businesslogic.EmailSender - Classe

Funzione

Classe utilizzata per la costruzione di messaggi di posta istantanei rivolti all'utente, necessari per il corretto compimento di alcune operazioni messe a disposizione dall'applicativo.

Relazione d'uso con altri moduli

- La classe è utilizzata da:

  `it.hourglass.myTalk.server.model.businesslogic.RPCServiceImpl`

Attributi

- `String user`
  Identificativo univoco del dato `user` da utilizzare durante l'autenticazione.
- `String password`
  Password da utilizzare durante l'autenticazione.
- `String host`
  Host del server di posta
- `String sender`
  Mittente dell'email.
- `String subject`
  Oggetto dell'email.
- `String recipient`
  Destinatario dell'email.
- `String content`
  Contenuto dell'email.

Metodi

+ `EmailSender(String recipient, String subject, String content)`
  Costruttore completo, richiede i parametri di connessione al server di posta
+ `void inviaEmail()`
  Metodo che si occupa dell'invio effettivo della mail. Comporterà riutilizzo di codice appartenente ai *package* `java.mail.*` per l'invio di email, secondo metodologie standard.

2.19 Package it.hourglass.myTalk.server.WSServer

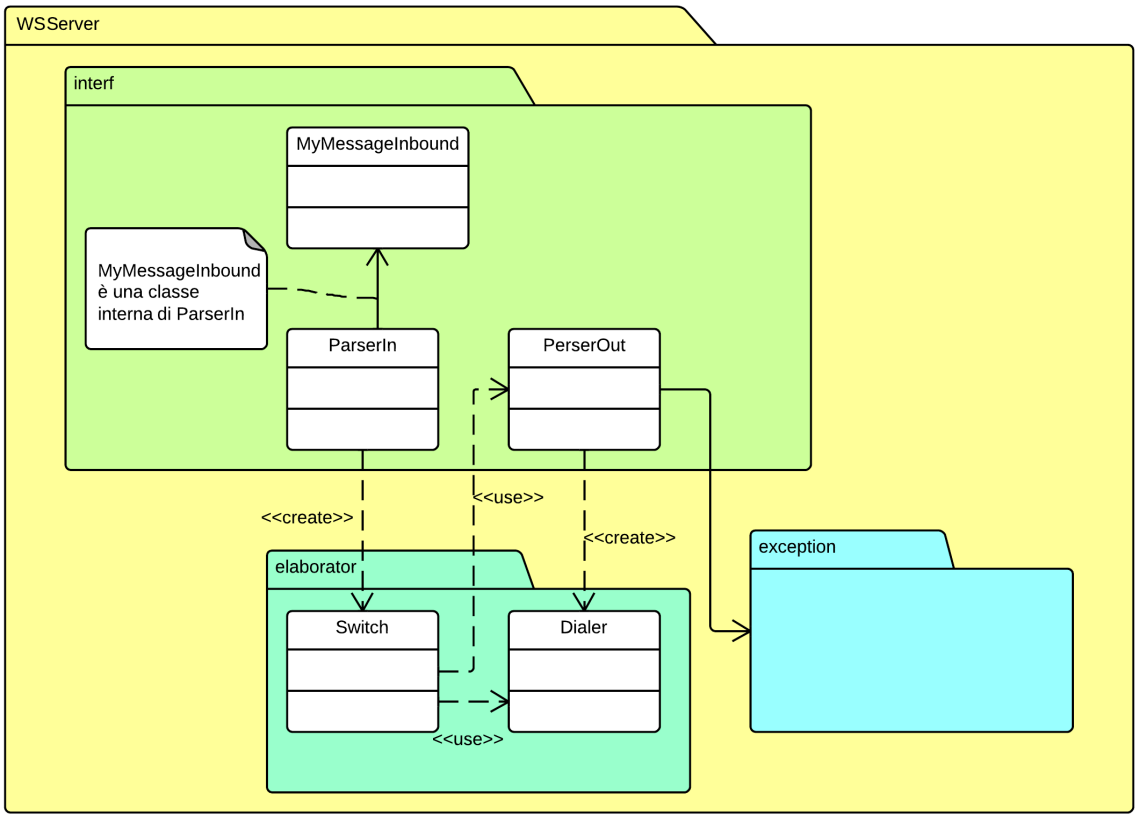

Figura 8: Package it.hourglass.myTalk.server.WSServer

Questo package contiene la servlet che si occuperà di smistare ed elaborare i JSON dei vari utenti che si collegheranno al sistema e di offire servizi in base al tipo di richiesta ricevuta.

2.20 Package it.hourglass.myTalk.server.WSServer.interf

Questo package contiene le classi preposte alla scambio di informazioni tra la servlet e i client.

2.20.1 server.WSServer.interf.ParserIn - Classe

Funzione

Questa classe ha lo scopo di aprire un Web Socket e di interpretare correttamente i JSON in ingresso inoltrando successivamente le informazioni alla classe preposta.

Relazione d'uso con altri moduli

- La classe non è utilizzata da nessun'altro componente.
- La classe utilizza:

  `elaborator.Switch`
- La classe estende

  `WebSocketServlet` Necessaria per essere individuata come una servlet.

Attributi

- `Switch switch`
  Riferimento ad un oggetto Switch, necessario per le future elaborazioni.
- `HashMap<String, Integer> typeID`
  Una mappa che contiene i codici dei vari tipi di JSON che arrivano. Vi possono essere 9 tipo di messaggio con il suo codice correlato:
  - 0 - registration
  - 1 - offer
  - 2 - answer
  - 3 - candidate
  - 4 - incoming
  - 5 - list
  - 6 - bye
  - 7 - ping
  - 8 - endcall
  - 9 - notification
- `BaseViewPresenter view`
  Riferimento alla view di base.
- `Profile profile`
  Variabile che si riferisce all'oggetto che mantiene i dati relativi all'utente principale garantendone coerenza e accessibilità in tutte le componenti dell'applicativo.

Metodi

`StreamInbound createWebSocketInbound(String string, HttpServletRequest hsr)`
E' un metodo implementato dell'interfaccia WebSocketServlet. Viene invocato quando viene richiesto un nuovo Web Socket. Il suo scopo è creare un'istanza di MyMessageInbound.

2.20.2 server.WSServer.interf.ParserIn.MyMessageInbound - Classe

Funzione

Questa classe ha lo scopo di definire un Web Socket e tutti i suoi metodi correlati.

Relazione d'uso con altri moduli

- La classe è utilizzata da :

  `interf.ParserIn`

- La classe utilizza :

  `interf.ParserIn`

- La classe estende

  `WebSocketServlet` Utilizzata per definire tutti i metodi necessari all'utilizzo del Web Socket.

Attributi

`WsOutbound myoutbound`
L'istanza vera e propria del Web Socket.

Metodi

`+ void onOpen(WsOutbound outbound)`
Metodo utilizzato quando viene creato il Web Socket. Assegna l'istanza a `myoutbound`.

`+ void onClose(int status)`
Metodo utilizzato quando viene chiuso il Web Socket. Viene inviato a switch un messaggio per avvisare della disconnessione del Web Socket e quindi del client ad esso collegato.

`+ void onTextMessage(CharBuffer cb)`
Metodo utilizzato quando viene ricevuto un messaggio testuale. Viene individuato il tipo di messaggio ed estratti i dati che esso conteneva per poi essere inviati al metodo `goSwitch()` dell'istanza switch di Switch.

2.20.3 server.WSServer.interf.ParserOut - Classe

Funzione

Questa classe ha lo scopo di assemblare i JSON che verranno inviati poi ai client.

Relazione d'uso con altri moduli

- La classe è utilizzata da :

  `elaborator.Dialer`

- La classe non utilizza alcuna classe.

Metodi

`+ static synchronized void sendToClient(WsOutbound ws, String s)`
Metodo utilizzato per inviare il JSON al client del Web Socket `ws`.

`+ static synchronized void sendToClient(WsOutbound ws, String s)`
Metodo utilizzato per inviare un'informazione generica al client del Web Socket `ws`.

2.21 Package it.hourglass.myTalk.server.WSServer.elaborator

Questo package contiene le classi preposte all'elaborazione dei dati in input per produrre un output atteso o un servizio.

2.21.1 server.WSServer.elaborator.Switch - Classe

Funzione

La classe ha lo scopo di eseguire l'elaborazione corretta dei dati in base al tipo di messaggio ricevuto.

Relazione d'uso con altri moduli

- La classe è utilizzata da :

  `interf.ParserIn`

- La classe utilizza :

  `interf.ParserOut`

Attributi

`- Dialer d`

Riferimento ad un oggetto Dialer, necessario per far eseguire l'elaborazione.

Metodi

`+ void goSwitch(int type,JSONObject data,WsOutbound t)`

Metodo utilizzato per scegliere l'elaborazione corretta in base al tipo di messaggio. Il parametro `data` contiene i dati necessari per la corretta elaborazione.

2.21.2 server.WSServer.elaborator.Dialer - Classe

Funzione

La classe ha lo scopo di eseguire l'elaborazione dei dati e di produrre un output o un servizio atteso.

Relazione d'uso con altri moduli

- La classe è utilizzata da :

  `interf.Switch`

- La classe utilizza :

  `interf.ParserOut`

Attributi

- `HashMap<String, List<String» regUserFriends`
  E' una mappa che associa ad un nickname una lista di contatti.
- `HashMap<String, WsOutbound> regUserWs`
  E' una mappa che associa ad un nickname un riferimento al Web Socket.

Metodi

- `List reverseSearch(HashMap<String, WsOutbound> t, Object w)`
  Metodo utilizzato per la ricerca inversa in una tabella. Dato un oggetto ne restituisce la chiave.

`synchronized void register(WsOutbound ws)`
Metodo utilizzato per elaborare la richiesta di registrazione presso il server. Viene creato un id temporaneo di 6 caratteri che verrà utilizzato come nick temporaneo per l'utente che ha inoltrato la richiesta.

`synchronized void register(WsOutbound ws, String s)`
Metodo utilizzato per elaborare la richiesta di registrazione presso il server. L'utente viene registrato con il nickname da lui comunicato. Una volta registrato sul server gli viene notificato chi è online.

`void sendOffer(JSONObject obj,String sdp,String type, String o, String r)`
Metodo utilizzato per inviare all'utente `r` l'offerta di chiamata inviata da `o`. Viene sollevata un eccezione nel caso il ricevente non sia registrato presso il server.

`void sendCandidate(JSONObject obj, String o,String r)`
Metodo utilizzato per inviare una richiesta di inizio chiamata.

`void sendIncoming(String o, String r, int s,int str)`
Metodo utilizzato per inviare messaggi di tipo Incoming da `o` a `r`. Se l'incoming è di accettazione(s=2) allora viene inviato anche il tipo della chiamata con `str`, altrimenti viene inviata senza.

`synchronized void byeUser(String s)`
Metodo utilizzato per rimuovere un utente tramite il suo nickname dalla tabella degli utenti registrati. Una volta eliminato (se l'utente è registrato) viene notificato a tutti i suoi contatti l'avvenuta disconnessione.

`synchronized void byeUser(WsOutbound n)`
Metodo utilizzato per rimuovere un utente tramite il riferimento al suo web socket dalla tabella degli utenti registrati. Una volta eliminato (se l'utente è registrato) viene notificato a tutti i suoi contatti l'avvenuta disconnessione.

`+ void sendList(String mynick, List<String> list)`
Metodo utilizzato per associare una lista di contatti ad un nickname tramite l'aggiunta di `list` nella lista `regUserFriends`.

`- void getFriendStatus(WsOutbound ws, List<String> l)`
Metodo utilizzato per sapere lo stato di tutti i contatti online.

`- List<String> sendNotificationOn(String n)`
Metodo utilizzato per notificare a tutti i contatti dell'utente `n` il suo stato attivo. Il metodo ritorna una lista di contatti online.

`- void sendNotificationOff(String n)`
Metodo utilizzato per notificare a tutti i contatti dell'utente `n` il suo stato offline.

`+ void sendEndCall(String nick, String info )`
Metodo utilizzato per informare l'utente `nick` della fine della chiamata. `Info` può contenere informazioni aggiuntive sul termine di quest'ultima.

`+ void sendNotification (String o, String r, int i)`
Metodo utilizzato per inviare vari tipi di notifiche a `r`. Il tipo della notifica viene identificato dalla variabile `i`.

2.22 Package it.hourglass.myTalk.server.WSServer.exception

Questo package contiene la gestione di alcune eccezioni che possono scatenarsi durante l'elaborazione dei JSON.

2.22.1 server.WSServer.elaborator.alreadyExist - Classe

Funzione

Questa classe ha lo scopo di definire l'eccezione che si scatenerà quando l'utente che si cerca di inserire è già presente.

Relazione d'uso con altri moduli

- La classe è utilizzata da :

  `interf.Switch`

  `interf.Dialer`

- La classe utilizza :

  `interf.ParserOut`

Metodi

`+ alreadyExist(WsOutbound ws)`

Costruttore dell'eccezione. Manda un messaggio al client per avvertirlo dell'errore.

2.22.2 server.WSServer.elaborator.notFound - Classe

Funzione

Questa classe ha lo scopo di definire l'eccezione che si scatenerà quando l'utente che si cerca non è stato trovato.

Relazione d'uso con altri moduli

- La classe è utilizzata da :

  `interf.Switch`

  `interf.Dialer`

- La classe utilizza :

  `interf.ParserOut`

Metodi

`+ alreadyExist(WsOutbound ws)`

Costruttore dell'eccezione. Manda un messaggio al client per avvertirlo dell'errore.

3 Tracciamento della relazione requisiti - classi

Un tracciamento delle le relazioni intercorrenti tra requisiti e classi può essere trovato nel documento *Specifica Tecnica v.3.0, Sezione 8*. Tramite analisi successive a una prima fase di progettazione, è stato infatti in tale sede possibile tracciare le correlazioni tra componenti logiche e classi effettivamente esistenti con un elevato grado di dettaglio. Anche tutte queste ultime sono infatti state individuate e ricondotte ai requisiti che soddisfano. Si evita quindi di riportare tale tracciamento anche in questa sede per prevenire ridondanza.