MyTalk

Sofware di comunicazione tra utenti senza requisiti di installazione

clockworkTeam7@gmail.com

Specifica Tecnica

v 3.0

Informazioni sul documento

Nome documento	Specifica Tecnica
Versione documento	v 3.0
Data creazione	2013/01/14
Data ultima modifica	2013/07/15
Uso documento	Esterno
Redazione	Bain Giacomo Furlan Valentino La Bruna Agostino
Verifica	Zohouri Haghian Pardis
Approvazione	Gavagnin Jessica
Lista distribuzione	gruppo *Clockwork* Zucchetti SPA Prof. Tullio Vardanega

Sommario

Questo documento vuol definire l'architettura generale che il prodotto dovrà avere.

Diario delle modifiche

Autore	Modifica	Data	Versione
Gavagnin Jessica	Approvazione del documento	2013/07/15	v 3.0
Zohouri Haghian Pardis	Verifica del documento	2013/07/14	v 2.6
La Bruna Agostino	Aggiunta tabella con spiegazione dettagliata dei messaggi	2013/07/11	v 2.5
Furlan Valentino	Modificato diagramma Dao, Client e Server	2013/07/09	v 2.4
Furlan Valentino	Creata appendice di Backbone	2013/07/06	v 2.3
Furlan Valentino	Modificati svantaggi Backbone, modificato diagramma MV*	2013/07/01	v 2.2
Bain Giacomo	Modificati svantaggi SQLite	2013/06/27	v 2.1
Furlan Valentino	Approvazione del documento	2013/03/15	v 2.0
Bain Giacomo	Verifica del documento	2013/03/08	v 1.8
Gavagnin Jessica	Modifica del capitolo 9	2013/03/07	v 1.7
Zohouri Haghian Pardis	Modifica del capitolo 8	2013/03/07	v 1.6
Zohouri Haghian Pardis	Modifica del capitolo 7	2013/03/05	v 1.5
Gavagnin Jessica	Modifica del capitolo 6	2013/03/05	v 1.4
Gavagnin Jessica	Modifica del capitolo 5	2013/03/04	v 1.3
Gavagnin Jessica	Modifica del capitolo 4	2013/03/03	v 1.2
Gavagnin Jessica	Modifica del capitolo 3	2013/03/01	v 1.1
La Bruna Agostino	Verifica documento e approvazione	2013/01/29	v 1.0
Palmisano Maria Antonietta	Controllo dei tracciamenti	2013/01/29	v 0.16
Bain Giacomo	Controllo concettuale capitolo 9 e 10	2013/01/28	v 0.15
Palmisano Maria Antonietta	Controllo concettuale fino al capitolo 6	2013/01/28	v 0.14
Bain Giacomo	Controllo ortografico, strutturale e sintattico	2013/01/26	v 0.13
Furlan Valentino	Stesura capitolo 9	2013/01/25	v 0.12
Gavagnin Jessica	Stesura capitolo 10	2013/01/25	v 0.11
Ceseracciu Marco	Stesura capitolo 7	2013/01/24	v 0.10
Furlan Valentino	Stesura capitolo 8	2013/01/24	v 0.9
Ceseracciu Marco	Bozza capitolo 7	2013/01/23	v 0.8
Bain Giacomo	Stesura capitolo 2	2013/01/22	v 0.7
Gavagnin Jessica	Stesura capitolo Server	2013/01/21	v 0.6
Zohouri Haghian Pardis	Stesura capitolo Client	2013/01/21	v 0.5
Bain Giacomo	Stesura capitolo 4	2013/01/21	v 0.4
Gavagnin Jessica	Bozza capitolo 3	2013/01/17	v 0.3
Ceseracciu Marco	Bozza capitolo 2	2013/01/16	v 0.2

Zohouri Haghian Pardis	Creazione documento, stesura sezione Introduzione	2013/01/15	v 0.1

Indice

1 Introduzione **1**
1.1 Scopo del documento . . . 1
1.2 Scopo del prodotto . . . 1
1.3 Glossario . . . 1
1.4 Riferimenti . . . 1
1.4.1 Normativi . . . 1
1.4.2 Informativi . . . 1

2 Architettura Generale **2**
2.1 Client . . . 2
2.2 Server . . . 3
2.3 Architettura dell'intero sistema . . . 3
2.4 Stile architetturale del server . . . 5

3 Design Pattern **7**
3.1 Singleton . . . 7
3.2 DAO . . . 7
3.3 MV* . . . 8
3.4 Observer . . . 9

4 Strumenti Utilizzati **11**
4.1 Java . . . 11
4.2 SQLite . . . 11
4.2.1 Differenze tra SQLite e MySQL . . . 12
4.3 HTML5 e CSS3 . . . 12
4.4 JavaScript . . . 12
4.5 Backbone.js . . . 12
4.6 WebSocket . . . 13
4.7 WebRTC . . . 13
4.8 QUnit . . . 13
4.9 RequireJs . . . 14
4.10 SinonJs . . . 14

5 Comunicazione Client-Server **15**

6 Client **17**
6.1 View . . . 17
6.1.1 mytalk.client.view.AuthenticationView . . . 18
6.1.2 mytalk.client.view.CallView . . . 18
6.1.3 mytalk.client.view.ChatView . . . 19
6.1.4 mytalk.client.view.ContactView . . . 20
6.1.5 mytalk.client.view.FileView . . . 20
6.1.6 mytalk.client.view.FunctionsView . . . 21
6.1.7 mytalk.client.view.NotificationView . . . 21
6.1.8 mytalk.client.view.RecordMessageView . . . 22

6.1.9 mytalk.client.view.SideView . . . 22
6.1.10 mytalk.client.view.StatisticsView . . . 23
6.1.11 mytalk.client.view.TutorialView . . . 23
6.1.12 mytalk.client.view.UserDataView . . . 24
6.2 Communication . . . 24
6.2.1 mytalk.client.communication.AuthenticationCommunication 25
6.2.2 mytalk.client.communication.CallCommunication . . . 25
6.2.3 mytalk.client.communication.ChatCommunication . . . 25
6.2.4 mytalk.client.communication.ContactsCommunication . . 26
6.2.5 mytalk.client.communication.FileCommunication . . . 26
6.2.6 mytalk.client.communication.NotificationCommunication 27
6.2.7 mytalk.client.communication.RecordMessageCommunication 27
6.2.8 mytalk.client.communication.TutorialCommunication . . . 28
6.2.9 mytalk.client.communication.UserDataCommunication . . 28
6.3 Collection . . . 29
6.3.1 mytalk.client.collection.ContactsCollection . . . 29
6.3.2 mytalk.client.collection.RecordMessagesCollection . . . 29
6.3.3 mytalk.client.collection.TextMessagesCollection . . . 30
6.3.4 mytalk.client.collection.TutorialsCollection . . . 30
6.4 Model . . . 31
6.4.1 mytalk.client.model.ContactModel . . . 31
6.4.2 mytalk.client.model.RecordMessageModel . . . 32
6.4.3 mytalk.client.model.TextMessageModel . . . 32
6.4.4 mytalk.client.model.TutorialModel . . . 32
6.4.5 mytalk.client.model.UserModel . . . 33

7 Server 34
7.1 Transfer Layer . . . 35
7.1.1 mytalk.server.transfer.ListenerTransfer . . . 36
7.1.2 mytalk.server.transfer.AuthenticationTransfer . . . 36
7.1.3 mytalk.server.transfer.CallTransfer . . . 37
7.1.4 mytalk.server.transfer.ChatTransfer . . . 38
7.1.5 mytalk.server.transfer.FileTransfer . . . 38
7.1.6 mytalk.server.transfer.RecordMessageTransfer . . . 38
7.1.7 mytalk.server.transfer.UserTransfer . . . 39
7.2 Manager Layer . . . 40
7.2.1 mytalk.server.usermanager.AuthenticationManager . . . 40
7.2.2 mytalk.server.usermanager.UserManager . . . 40
7.2.3 mytalk.server.functionmanager.Converter . . . 41
7.3 Data Layer . . . 42
7.3.1 mytalk.server.dao.JavaConnectionSQLite . . . 42
7.3.2 mytalk.server.dao.RecordMessageDao . . . 42
7.3.3 mytalk.server.dao.RecordMessageDaoSQL . . . 42
7.3.4 mytalk.server.dao.TutorialsDaoSQL . . . 43
7.3.5 mytalk.server.dao.UserDao . . . 43
7.3.6 mytalk.server.dao.UserDaoSQL . . . 44

7.3.7 mytalk.server.shared.RecordMessage . . . 44
7.3.8 mytalk.server.shared.Tutorials . . . 45
7.3.9 mytalk.server.shared.User . . . 45
7.3.10 mytalk.server.shared.UserList . . . 45

8 Tracciamento componenti-requisiti 47

9 Tracciamento requisiti-componenti 50

10 Diagramma delle attività 52
10.1 Utente non autenticato . . . 52
10.2 Registrazione . . . 53
10.3 Autenticazione . . . 54
10.4 Utente autenticato . . . 55
10.5 Modifica dati account . . . 56
10.6 Contattare un indirizzo IP . . . 57
10.7 Contattare un utente . . . 58
10.8 Chiamata . . . 59
10.9 Registrazione messaggio . . . 60
10.10 Gestione delle notifiche . . . 60

11 Prototipo interfaccia utente 62

A Backbone.js 66
A.0.1 Dipendenze . . . 66
A.1 Backbone.Model . . . 66
A.2 Backbone.Collection . . . 67
A.3 Backbone.View . . . 68
A.4 Backbone.Router . . . 69
A.5 Backbone.Events . . . 69
A.6 Underscore.js Templates . . . 70

Elenco delle figure

1 Architettura generale 2
2 Architettura generale dell'intero sistema 4
3 Rappresentazione Three Tier 5
4 Rappresentazione DAO 7
5 Rappresentazione MV* 8
6 Rappresentazione Observer 9
7 Diagramma dei package dell'architettura del Client 17
8 Diagramma dei package dell'architettura del Server 35
9 Diagramma delle attività di un utente non autenticato 52
10 Diagramma delle attività di registrazione 53
11 Diagramma delle attività di autenticazione 54
12 Diagramma delle attività che mostra le operazioni a disposizione di un utente autenticato 55
13 Diagramma delle attività di modifica dei dati account 56
14 Diagramma delle attività per comunicare con un indirizzo IP . . 57
15 Diagramma delle attività dei servizi di un utente 58
16 Diagramma delle attività della chiamata 59
17 Diagramma delle attività della registrazione di un messaggio . . 60
18 Diagramma delle attività di gestione delle notifiche 60
19 Pagina iniziale 62
20 Pagina dopo login 63
21 Schermata videochiamata 64
22 Schermata chiamata audio 64
23 Schermata comunicazione testuale 65
24 Schermata Videoconferenza 65

Elenco delle tabelle

1 Messaggi JSON passati tra Client e Server 16
2 Tracciamento tra componenti e requisiti 49
3 Tracciamento tra requisiti e componenti 51

1 Introduzione

1.1 Scopo del documento

Il presente documento ha lo scopo di definire la progettazione ad alto livello che il prodotto dovrà avere. Verranno presentati i vari design pattern utilizzati nella creazione del prodotto, l'architettura generale secondo la quale saranno organizzate le varie componenti software e il tracciamento tra le componenti software ed i requisiti.

1.2 Scopo del prodotto

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

1.3 Glossario

I termini tecnici o di uso non comune sono presenti nel documento allegato `Glossario_v2.0.pdf`. Tali riferimenti vengono evidenziati tramite sottolineatura alla prima occorrenza del termine nel documento.

1.4 Riferimenti

1.4.1 Normativi

- Norme di Progetto (allegato `Norme_di_Progetto_v4.0.pdf`)

1.4.2 Informativi

- Analisi dei Requisiti (allegato `Analisi_dei_Requisiti_v4.0.pdf`)
- SWEBOK - Chapter 3: Software Design: `http://www.computer.org/protal/web/swebok/html/ch3`
- SWEBOK - Chapter 11: Software Quality: `http://www.computer.org/protal/web/swebok/html/ch11`
- Software Engineering - Chapter 11: Architectural design - Ian Sommerville - 8th ed. (2006)

2 Architettura Generale

L'applicativo è stato diviso in due sistemi, ovvero il client e il server, come indicato nella figura 1. L'indice di accoppiamento tra questi due sottosistemi sarà tenuto al minimo necessario.

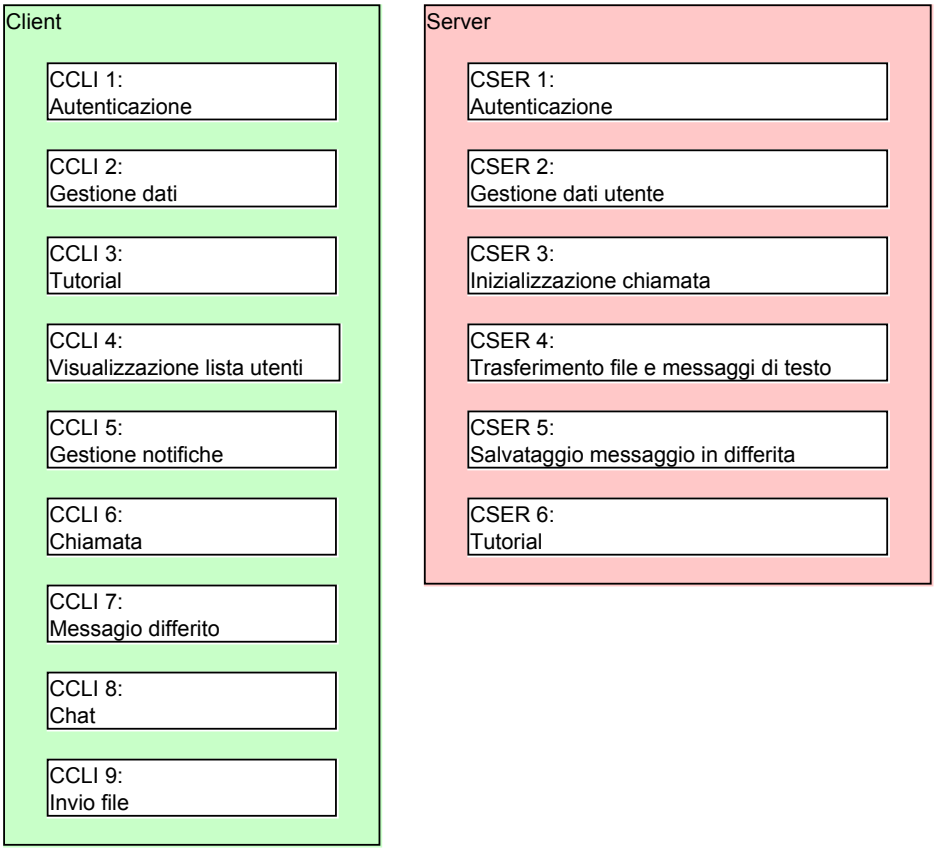

Figura 1: Architettura generale

2.1 Client

Si occuperà di gestire l'interfaccia visibile all'utente le cui operazioni verranno poi inviate al server. Il lato client è stato suddiviso nel seguente modo:

- **CCLI1:** autenticazione
- **CCLI2:** gestione dati
- **CCLI3:** tutorial

- **CCLI4:** visualizzazione lista utenti
- **CCLI5:** gestione notifiche
- **CCLI6:** chiamata
- **CCLI7:** messaggio differito
- **CCLI8:** chat
- **CCLI9:** invio file

2.2 Server

Si occuperà di gestire le richieste fatte e di inviare una risposta al client. Il server è stato suddiviso in

- **CSER1:** autenticazione
- **CSER2:** gestione dati utente
- **CSER3:** inizializzazione chiamata
- **CSER4:** trasferimento file e messaggi di testo
- **CSER5:** salvataggio messaggio in differita
- **CSER6:** tutorial

2.3 Architettura dell'intero sistema

Facendo riferimento alla figura 2 si può avere quindi una visione dell'architettura e delle relazioni tra gli strati:

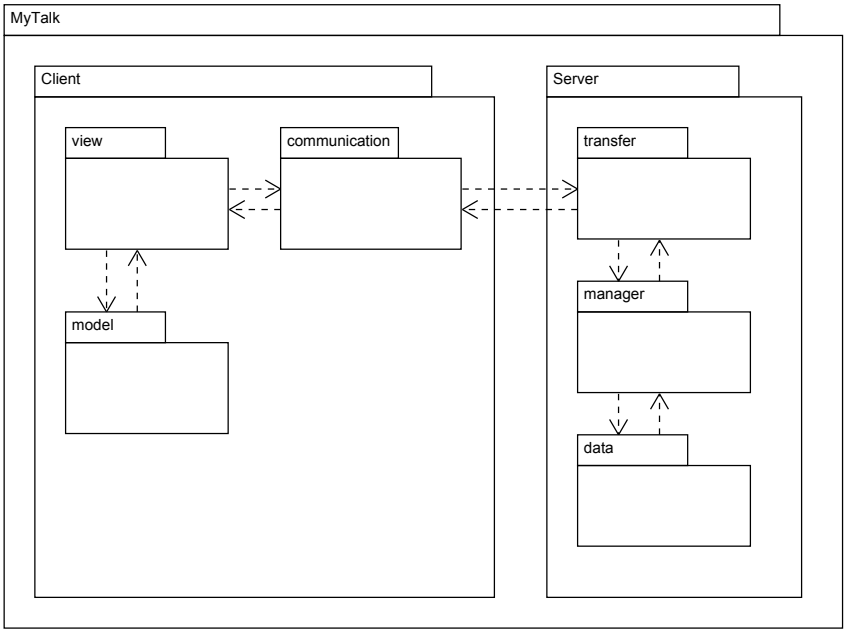

Figura 2: Architettura generale dell'intero sistema

- `mytalk` incorpora l'intero sistema
  - `mytalk.client` racchiude al suo interno il lato client di **MyTalk**
    - `mytalk.client.view` racchiude al suo interno l'interfaccia grafica e la parte logica. Questo strato viene definito dal framework Backbone.js
    - `mytalk.client.model` contiene localmente le informazioni necessarie a popolare le viste
    - `mytalk.client.communication` si occupa della gestione delle comunicazioni tra client e server e tra più client
  - `mytalk.server` racchiude al suo interno il lato server di **MyTalk**
    - `mytalk.server.transfer` gestisce il collegamento tra server e client e le operazioni di comunicazione tra peers
    - **mytalk.server.manager** racchiude al suo interno:
      - `mytalk.server.functionmanager` gestisce le operazioni di conversione dati per lo strato transfer
      - `mytalk.server.usermanager` gestisce le operazioni di comunicazione tra lo strato transfer e lo strato data
    - **mytalk.server.data** racchiude al suo interno:

- `mytalk.server.dao` gestisce la base di dati, ovvero l'inserimento, la modifica e la rimozione dei dati e la loro permanenza
- `mytalk.server.shared` contiene al suo interno una copia del database in modo da ridurre al minimo gli accessi a quest'ultimo

2.4 Stile architetturale del server

Lo stile architetturale che verrà seguito per il server è il Multitier nella forma Three Tier.

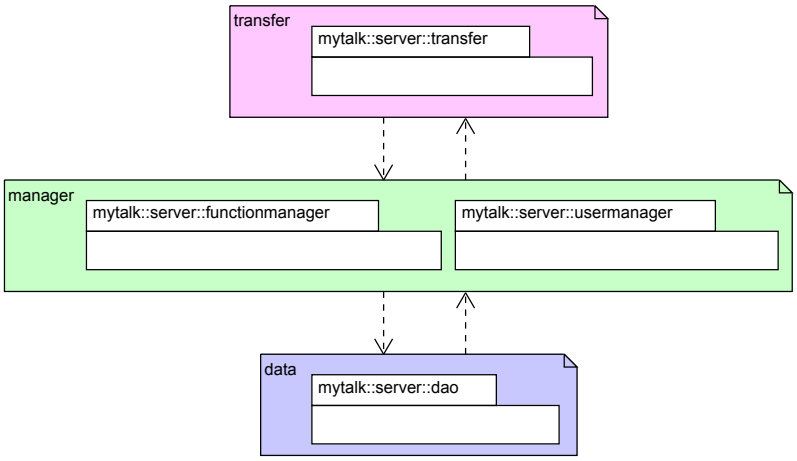

Figura 3: Rappresentazione Three Tier

- **Descrizione:** tale stile architetturale permette una disgiunzione fra i vari gruppi di entità che cooperano per l'erogazione del servizio. Un primo livello si occuperà della comunicazione con il client, un secondo livello invece si occuperà di effettuare le operazioni logiche e di comunicare con il terzo strato che si occuperà di modificare la base di dati
  - strato transfer: offrirà un'interfaccia di comunicazione con il client e i livelli sottostanti del server e si occuperà di svolgere le funzionalità di comunicazione
  - strato manager: gestirà la comunicazione tra gli strati transfer e data e la visualizzazione e la modifica delle informazioni presenti nello strato data
  - strato data: sarà un contenitore di informazioni riguardanti gli utenti e di messaggi registrati e collegamenti

- **Motivazione:** il beneficio principale di questo stile è la possibilità di aggiornare/cambiare un livello senza modificare i livelli adiacenti. La forte comunicazione rigidamente strutturata tra gli strati, che favorisce il disaccoppiamento, rende tale architettura un ottimo modello per applicazioni client-server, poiché ogni livello non esiste come unità logica a se stante, ma si adegua allo specifico ambiente di rete in cui esegue

3 Design Pattern

Presentiamo qui di seguito i design pattern che andremo ad utilizzare per la rappresentazione dell'architettura del sistema.

3.1 Singleton

- **Descrizione:** tale design pattern assicura la presenza di unica istanza della classe e fornisce un punto d'accesso globale a tale istanza, tramite costruttore privato e puntatore alla classe stessa
- **Motivazione:** utilizzeremo tale design pattern al fine di impedire la proliferazione di copie di parti del server, e quindi interferenze
- **Contesto applicativo:** verrà utilizzato dalla classe `mytalk.server.Launcher` per inizializzare correttamente le classi del nostro server e per `mytalk.server.shared.UserList` e `mytalk.server.dao.Dao`

3.2 DAO

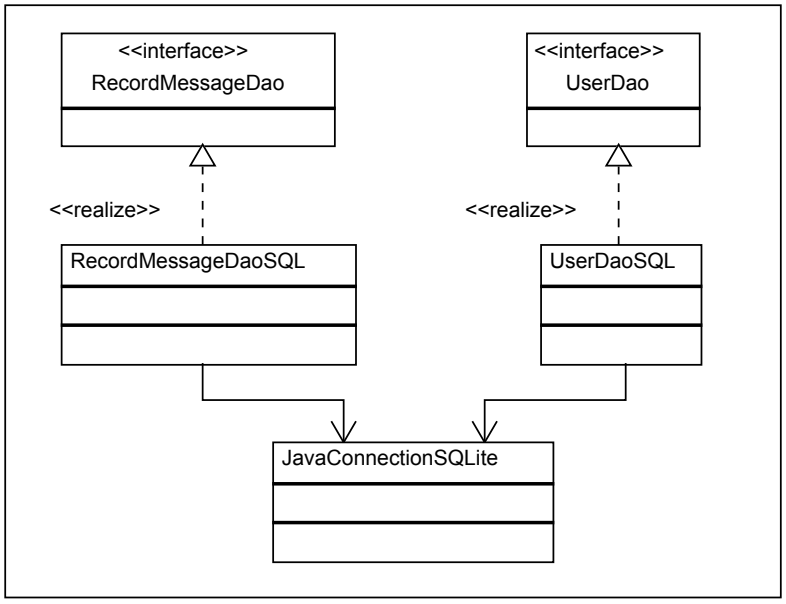

Figura 4: Rappresentazione DAO

- **Descrizione:** tale design pattern si occupa di creare un'interfaccia di accesso alla base di dati e adatta l'implementazione di tale interfaccia al database utilizzato

- **Motivazione:** utilizzeremo questo design pattern per accedere al data base ed effettuare operazioni su di esso, disaccoppiando di conseguenza la logica di business da quella di accesso ai dati
- **Contesto applicativo:** verrà utilizzato nello strato data del server per prelevare e modificare dati dal database

3.3 MV*

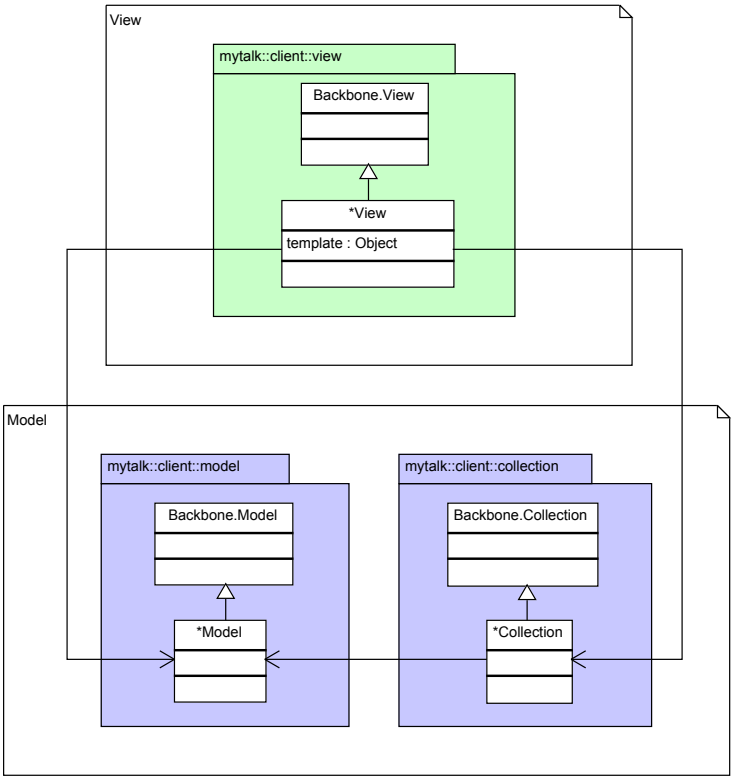

Figura 5: Rappresentazione MV*

- **Descrizione:** tale design pattern deriva da MVC e MVP con la differenza che le funzioni di controller/presenter sono integrate nella view
  - **Model:** rappresenta le informazioni di cui necessita l'applicazione
  - **View:** contiene i metodi per visualizzare l'interfaccia grafica, composta dalle informazioni richieste, e per la creazione ed assegnazione di eventi

  - **Template:** sono frammenti di codice HTML arricchiti da Underscore.js che rappresentano il layout della view

- **Motivazione:** utilizzeremo questo pattern in quanto è quello utilizzato da Backbone.js che è il framework che andremo ad utilizzare per la realizzazione della parte client perché permette lo sviluppo di applicazioni web dinamiche, strutturate e modulari
- **Contesto applicativo:** verrà utilizzato per l'architettura del lato client

3.4 Observer

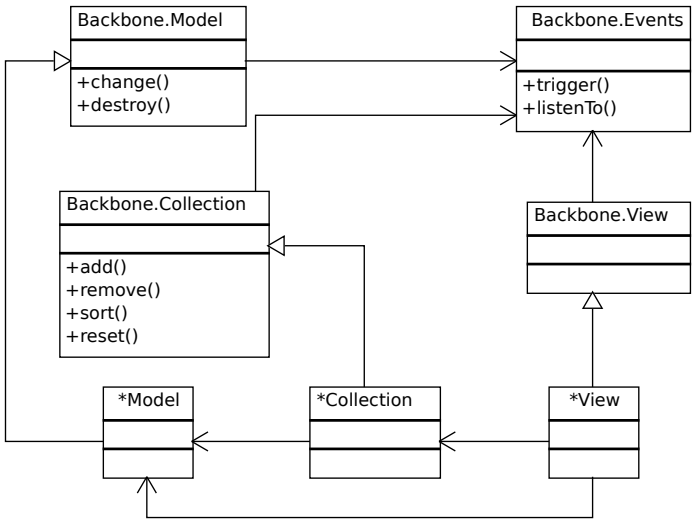

Figura 6: Rappresentazione Observer

- **Descrizione:** tale design pattern definisce la struttura che consente agli oggetti di una classe di "osservare" altri oggetti delle classi soggette a controlli. Inoltre, permette l'aggiornamento degli oggetti osservatori quando lo stato di un oggetto soggetto viene modificato
- **Motivazione:** questo design pattern riflette in tempo reale sui dati visibili all'utente qualsiasi modifica avvenga sui dati presenti nel database, indipendente dal numero di oggetti dipendenti, mantenendo un alto livello di consistenza fra classi correlate, e contemporaneamente cercando di tenere il più basso possibile il livello di accoppiamento. Questo design pattern è già implementato dal framework Backbone.js

- **Contesto applicativo:** verrà utilizzato, nel lato client, per la manutenibilità della lista utenti registrati (connessione e disconnessione degli utenti) o nel caso nuovi utenti si registrino, e per l'invio e ricezione dei messaggi testuali

4 Strumenti Utilizzati

4.1 Java

L'utilizzo di Java è legato ai requisiti di capitolato, verrà utilizzato per la parte server. È stato deciso di utilizzare la versione 7.0

- **Vantaggi:**
  - **Multipiattaforma:** grazie alla presenza della JVM si ha la sicurezza che il programma sarà eseguibile indipendentemente dal sistema operativo installato sulla macchina
  - **Indipendenza dalle risorse:** per lo stesso motivo sopra elencato l'utilizzo delle risorse fisiche sarà indipendente dal sistema operativo installato
- **Svantaggi:**
  - Nessun svantaggio rilevato

4.2 SQLite

Considerando la complessità relativamente limitata del database che occorrerà per la gestione degli utenti abbiamo deciso di utilizzare SQLite, invece del più potente MySQL, poiché lo riteniamo più adatto ai nostri scopi

- **Vantaggi:**
  - Conoscenza da parte dei componenti del gruppo del linguaggio SQL
  - Gestibile attraverso il package `java.sql` di Java
  - È multipiattaforma
  - Non richiede installazione di un server
  - Leggero e veloce
  - Maggior semplicità di impostazione
  - Ha transizioni atomiche, consistenti, isolate e durabili, anche in caso di crash di sistema o blackout
- **Svantaggi:**
  - Se il volume di dati diventasse molto ampio l'efficienza della base di dati ne risentirebbe

4.2.1 Differenze tra SQLite e MySQL

- SQLite, nella sua semplicità, permette un'installazione molto veloce ed è quasi privo di esigenze di manutenzione e configurazione, contrariamente a MySQL che richiede una installazione ed una configurazione complessa
- Attualmente il limite di dimensione massima è 2TB per database SQLite mentre per database MySQL è pari a 64TB
- Il limite di RAM utilizzata per SQLite è pari a 16 MB mentre MySQL richiede almeno 384 MB

4.3 HTML5 e CSS3

L'utilizzo di HTML5 è legato ai requisiti di capitolato e andrà a costituire insieme a CSS3 l'interfaccia web del prodotto

- **Vantaggi:**
  - HTML5 supporta le ultime tecnologie riguardanti la creazione di applicazioni web
  - Grafica più leggera e valida evitando l'utilizzo di tecnologia Flash
- **Svantaggi:**
  - HTML5 e CSS3 non attualmente standard

4.4 JavaScript

L'utilizzo di JavaScript è legato ai requisiti di capitolato. Lo andremo ad utilizzare per la parte client e in particolare per l'utilizzo di WebRTC e quindi per la comunicazione tra gli utenti.

4.5 Backbone.js[1]

Per la gestione del lato client abbiamo deciso di utilizzare il framework Backbone.js, che risulta comodo ed efficiente per la gestione di applicazioni che utilizzino pesantemente il linguaggio JavaScript.
Backbone.js ha un'architettura MV*, in quanto implementa Model e View, delegando a quest'ultima i compiti di una componente Controller tradizionale
Per il suo funzionamento Backbone.js necessita della libreria Underscore.js, di cui verrà utilizzato anche il sistema di templating, e di una libreria per la manipolazione del DOM. Tra quelle proposte abbiamo scelto di utilizzare jQuery

- **Vantaggi:**

[1] Per una descrizione più dettagliata del Framework Backbone.js si veda l'appendice A.

  - Maggiore facilità nella gestione del lato client e nella sua programmazione
  - Backbone.js è un progetto open source
  - Struttura già data dal framework

- **Svantaggi:**
  - Rimuovendo una vista viene rimosso anche il relativo elemento nel DOM, rendendo necessario il reinserimento manuale
  - Le viste, anche se rimosse, non vengono deallocate se non vengono rimossi anche gli eventi collegati ad esse

4.6 WebSocket

Per la gestione delle comunicazioni tra client e server utilizziamo il protocollo WebSocket, come richiesto dal proponente.
La tecnologia WebSocket fornisce un canale che permette la comunicazione, in entrambi i versi, attraverso una singola connessione ed è ideale per le comunicazioni client-server, mentre non è ottimale per le comunicazioni client-client. Tale protocollo è indipendente dal protocollo TCP, e grazie alla metodologia standard di inviare messaggi tra browser e server tenendo la connessione aperta, permette maggiore interazione tra browser e server, facilitando la creazione di applicazioni che forniscono contenuti in tempo reale.
Il protocollo in questione è supportato da numerosi browser, come Internet Explorer, Google Chrome, Firefox, Safari ed Opera.

4.7 WebRTC

Per la gestione delle comunicazioni tra client e client utilizziamo la tecnologia WebRTC, come richiesto dal proponente.
La tecnologia WebRTC è sviluppata per fornire metodi di comunicazione, tramite chiamate vocali, videochiamate e condivisione di file, tra due o più utenti senza caricare il server, che si occupa solo dell'inizializzazione del canale, inoltre non richiede l'installazione di plugin. Tale tecnologia è basata sul HTML5 e JavaScript.

4.8 QUnit

Framework per creare e svolgere i test di unità JavaScript

- **Vantaggi:**
  - Ha già implementate delle asserzioni
- **Svantaggi:**
  - Nessun svantaggio rilevato

4.9 RequireJs

Framework per creare e svolgere i test di unità in ambiente modulare

- **Vantaggi:**
  - Permette di effettuare test di unità su Backbone.js modulare
  - Compatibile con altri framework per svolgere test di unità (come QUnit)
- **Svantaggi:**
  - Nessun svantaggio rilevato

4.10 SinonJs

Framework che fornisce *spies*, *stubs* e *mocks* per creare e svolgere i test di unità Javascript

- **Vantaggi:**
  - Compatibile con altri framework per svolgere test di unità (come QUnit)
- **Svantaggi:**
  - Non è naturalmente adattato a WebSocket

5 Comunicazione Client-Server

La comunicazione tra il client ed il server avverrà tramite l'utilizzo di WebSocket (come indicato nella sezione 4.6). I messaggi che si andranno ad inviare saranno degli array associativi formati da coppie nome-valore. Ogni array dovrà contenere un campo di nome "type" che conterrà la tipologia di messaggio. Questo campo è particolarmente importante in quanto consente l'identificazione del messaggio da parte del metodo ricevente, che controllerà che il messaggio sia del tipo corretto prima di procedere alla sua elaborazione. Nei metodi del client gli array dovranno essere convertiti in stringhe JSON tramite l'utilizzo del metodo "JSON.stringify" prima di essere inviati e i messaggi ricevuti dovranno essere convertiti in array associativi tramite il metodo "JSON.parse". Il server invece creerà direttamente le stringhe JSON utilizzando eventualmente i metodi di conversione in stringa della classe `mytalk.server.functionmanager.Converter`.
Nella Tabella 1 vengono riportati i messaggi JSON con i rispettivi mittenti e destinatari. Il simbolo *i* indica che verrà passato un numero di elementi pari a size.

Classe	Campo type	Altri attributi	Classe che gestisce il pacchetto
AuthenticationCommunication	*login*	username, password	AuthenticationTransfer
	signUp	username, password, name, surname	
	logout	-	
CallCommunication	*call*	contact, callType,conference	CallTransfer
	addConferenceCaller	user, contact	
	addConferenceAnswer	user, contact	
	answeredCall	contact, conference	
	refuseCam	contact	
	sdp	contact, description	
	candidate	contact, candidate	
	candidateReady	contact	
	endCall	contact	
	endCallEarly	contact	
ChatCommunication	*sendText*	contact, message	ChatTransfer
ContactsCommunication	*getContacts*	-	AuthenticationTransfer
FileCommunication	*file*	file, contact	FileTransfer
NotificationCommunication	*busy*	contact	CallTransfer
	refuseCall	contact	
	refuseFile	contact	FileTransfer
RecordMessageCommunication	*sendRecord*	contact, path, date	RecordMessageTransfer
	removeRecord	contact, path, date	
TutorialCommunication	-	-	-
UserDataCommunication	*checkCredentials*	password	UserTransfer
	changeData	name, surname, password	

AuthenticationTransfer	*login*	answer, (name, surname) | error	AuthenticationCommunication
	signUp	answer, (error)	
	tutorials	size, title$i$, path$i$	TutorialCommunication
	getContacts	size, username$i$, name$i$, surname$i$, IPi	ContactsCommunication
CallTransfer	*call*	contact, callType, conference	NotificationCommunication
	answeredCall	answer, (error) | (user)	CallCommunication
	addConferenceCaller	user	
	addConferenceAnswer	user	
	offer	creato da WebRTC, contact	
	answer	creato da WebRTC, contact	
	candidate	creato da WebRTC, contact	
	candidateReady	contact	
	endCall	contact	
	endCallEarly	contact	
			NotificationCommunication
ChatTransfer	*sendText*	message, contact	ChatCommunication
	notDelivered	message, contact	
FileTransfer	*file*	file, contact	NotifiactionCommunication
	fileRefused	contact, error	FileCommunication
ListenerTransfer	-	-	-
RecordMessageTransfer	*getRecords*	size, sender$i$, message$i$, dateCreation$i$	NotificationCommunication
	removeRecord	answer, (error)	RecordMessageCommunication
	sendRecord	answer, (error)	
UserTransfer	*checkCredentials*	answer	UserDataCommunication
	changeData	answer, error	

Tabella 1: Messaggi JSON passati tra Client e Server

6 Client

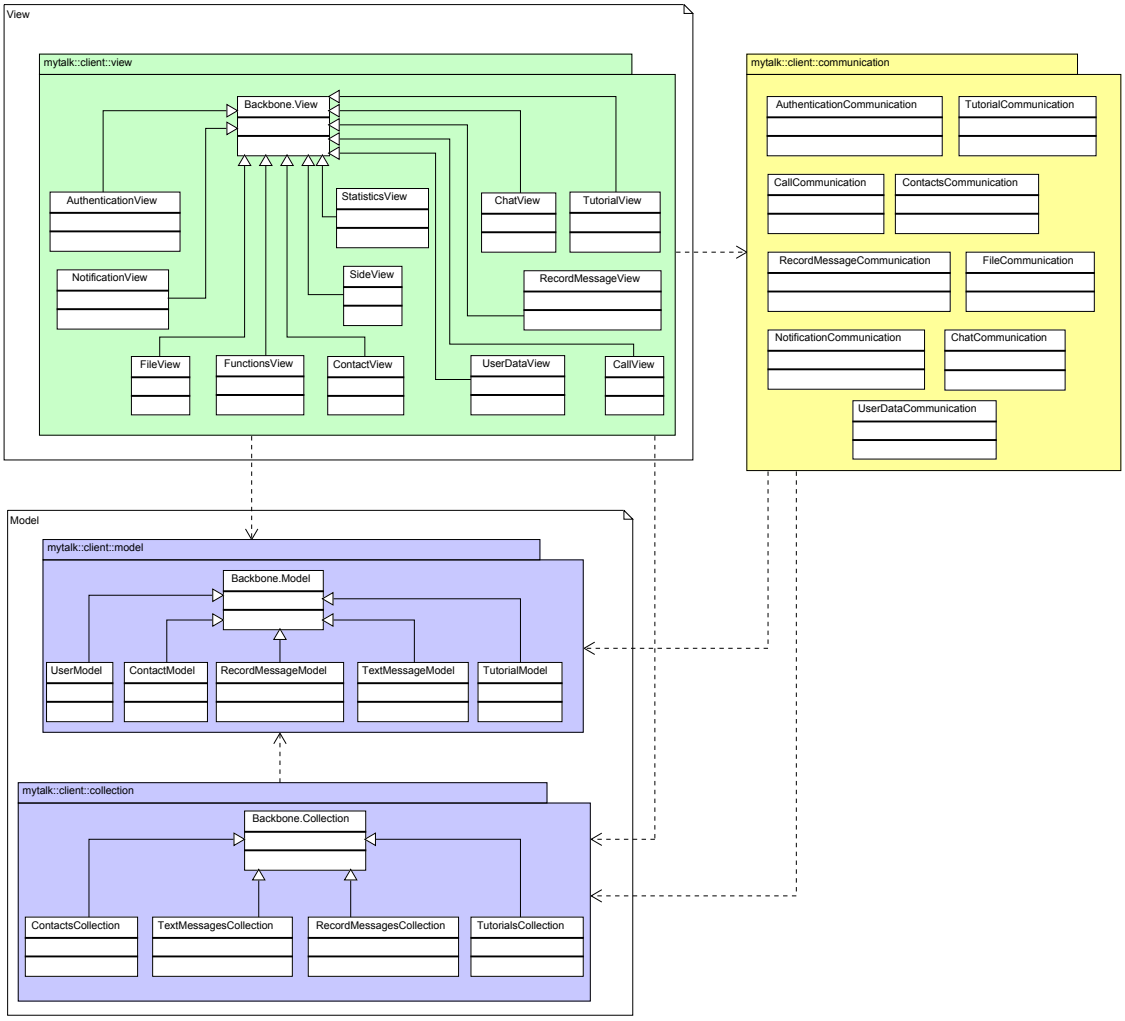

Figura 7: Diagramma dei package dell'architettura del Client

6.1 View

Tipo, obiettivo e funzione del componente: costituisce la parte del sistema che definisce ed implementa l'interfaccia web usufruibile dagli utenti. Il package `mytalk.client.view` contiene non solo le parti di interfaccia ma anche parti

logiche, come previsto dal framework Backbone.js, che prevede uno schema di tipo MV*.
Relazioni d'uso con altre componenti:

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

6.1.1 mytalk.client.view.AuthenticationView

Tipo, obiettivo e funzione del componente: la classe definisce la struttura, e la conseguente visualizzazione, della parte di pagina web che consente di effettuare l'accesso e la disconnessione dal sistema e la registrazione di un utente al server.
Relazioni d'uso con altre componenti:

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

Attività svolte e dati trattati: la classe rende disponibile all'utente un form per inserire i propri dati ed i pulsanti per scegliere l'operazione desiderata; fa in modo che vengano gestiti gli eventi per l'autenticazione, la registrazione e la modifica dei dati dell'utente e, in caso di corretta autenticazione, visualizza la lista dei contatti e si modifica per rispecchiare il fatto che ora l'utente è connesso al server.

6.1.2 mytalk.client.view.CallView

Tipo, obiettivo e funzione del componente: la classe definisce la struttura, e la conseguente visualizzazione, della parte di pagina web che consente di effettuare chiamate.
Relazioni d'uso con altre componenti:

- **La classe utilizza:**
  - `mytalk.client.communication.CallCommunication`

  - `mytalk.client.view.StatisticsView`
  - `mytalk.client.template.CallTemplate`

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

Attività svolte e dati trattati: la classe fa in modo che vengano visualizzate le corrette componenti grafiche e rende disponibili all'utente i metodi che permettono la gestione delle funzionalità legate alla comunicazione audio/video.

6.1.3 mytalk.client.view.ChatView

Tipo, obiettivo e funzione del componente: la classe definisce la struttura, e la conseguente visualizzazione, della parte di pagina web che consente di effettuare comunicazioni testuali.
Relazioni d'uso con altre componenti:

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

Attività svolte e dati trattati: la classe fa in modo che vengano visualizzate le corrette componenti grafiche e rende disponibili all'utente i metodi per la gestione della chat testuale.

6.1.4 mytalk.client.view.ContactView

Tipo, obiettivo e funzione del componente: la classe definisce la struttura, e la conseguente visualizzazione, di un singolo elemento della lista di contatti.
Relazioni d'uso con altre componenti:

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

Attività svolte e dati trattati: la classe mostra lo username di un singolo utente registrato sul server ed il suo stato attuale (online/offline), inoltre, se lo si seleziona, mostra le funzionalità di comunicazione disponibili con quello specifico utente.

6.1.5 mytalk.client.view.FileView

Tipo, obiettivo e funzione del componente: la classe definisce la struttura, e la conseguente visualizzazione, della parte di pagina web che consente di selezionare un file salvato in locale per mandarlo all'utente autenticato selezionato.
Relazioni d'uso con altre componenti:
La classe utilizza:

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

Attività svolte e dati trattati: la classe fa in modo che vengano visualizzati i componenti grafici necessari per le operazioni di selezione di un file salvato in locale e di scelta del destinatario.

6.1.6 mytalk.client.view.FunctionsView

Tipo, obiettivo e funzione del componente: la classe definisce la struttura, e la conseguente visualizzazione, della parte di pagina web che consente la selezione delle possibili operazioni di comunicazione con un singolo contatto, la visualizzazione delle informazioni legate ad esso o le operazioni di comunicazione multipla.
Relazioni d'uso con altre componenti:

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

Attività svolte e dati trattati: la classe fa in modo che vengano visualizzate le componenti grafiche che permettono la scelta delle funzionalità di comunicazione disponibili. Le operazioni di chiamata e di invio di messaggio di testo vengono rese visibili solo quando il contatto è collegato, mentre le operazioni di invio di messaggi audio/video e di visualizzazione delle informazioni del contatto sono sempre disponibili.

6.1.7 mytalk.client.view.NotificationView

Tipo, obiettivo e funzione del componente: la classe definisce la struttura, e la conseguente visualizzazione, della parte di pagina web che consente di visualizzare le notifiche di chiamata, di file o di videomessaggi in arrivo.
Relazioni d'uso con altre componenti:

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

- **La classe deriva da**:
  - `mytalk.client.Backbone.View`

Attività svolte e dati trattati: la classe fa in modo che vengano visualizzate le corrette componenti grafiche per la visualizzazione delle notifiche e rende disponibili all'utente i metodi per la gestione di quest'ultime.

6.1.8 mytalk.client.view.RecordMessageView

Tipo, obiettivo e funzione del componente: la classe definisce la struttura, e la conseguente visualizzazione, della parte di pagina web che consente di effettuare registrazioni audio o audio/video da inviare ad un utente registrato.
Relazioni d'uso con altre componenti:

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

Attività svolte e dati trattati: la classe fa in modo che vengano visualizzate le corrette componenti grafiche e rende disponibili all'utente i metodi per la registrazione di messaggi video.

6.1.9 mytalk.client.view.SideView

Tipo, obiettivo e funzione del componente: la classe definisce la struttura, e la conseguente visualizzazione, della parte di pagina web che consente la visualizzazione della lista di contatti e dei pulsanti che consentono di accedere alle funzionalità di chiamata verso un indirizzo IP e teleconferenza.
Relazioni d'uso con altre componenti:

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

  - `mytalk.client.collection.ContactsCollection`
  - `mytalk.client.model.UserModel`

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

Attività svolte e dati trattati: la classe fa in modo che venga visualizzata una lista di utenti registrati presso il server, inoltre presenterà i pulsanti relativi alla chiamata verso un indirizzo IP e alla chiamata verso più di un utente che, se premuti, dovranno mostrare le opzioni corrette.

6.1.10 mytalk.client.view.StatisticsView

Tipo, obiettivo e funzione del componente: la classe definisce la struttura della parte di pagina web che consente la visualizzazione di statistiche durante e dopo lo svolgimento di una chiamata.
Relazioni d'uso con altre componenti:

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

Attività svolte e dati trattati: la classe fa in modo che vengano visualizzate le statistiche relative della chiamata effettuata.

6.1.11 mytalk.client.view.TutorialView

Tipo, obiettivo e funzione del componente: la classe definisce la struttura, e la conseguente visualizzazione, della parte di pagina web che consente la visualizzazione dei video tutorial del prodotto **MyTalk**.
Relazioni d'uso con altre componenti:

- **La classe utilizza:**
  - `mytalk.client.communication.TutorialCommunication`
  - `mytalk.client.template.TutorialTemplate`

  - `mytalk.client.collection.TutorialsCollection`

- **La classe deriva da**:
  - `mytalk.client.Backbone.View`

Attività svolte e dati trattati: la classe fa in modo che l'utente scelga da una lista e visualizzi il video tutorial desiderato.

6.1.12 mytalk.client.view.UserDataView

Tipo, obiettivo e funzione del componente: la classe definisce la struttura, e la conseguente visualizzazione, della parte di pagina web che consente di visualizzare e di modificare i dati del proprio account.
Relazioni d'uso con altre componenti:

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

Attività svolte e dati trattati: la classe fa in modo che vengano visualizzati i dati dell'account dell'utente e rende disponibili i metodi per la modifica dei dati personali.

6.2 Communication

Tipo, obiettivo e funzione del componente: definisce ed implementa la parte del sistema che si occupa della comunicazione tra client e server.
Relazioni d'uso con altre componenti:

- **Il package utilizza:**
  - `mytalk.client.collection`
  - `mytalk.client.model`
- **Il package è utilizzato da:**
  - `mytalk.client.view`
- **Il package comunica con:**
  - `mytalk.server.transfer`

6.2.1 mytalk.client.communication.AuthenticationCommunication

Tipo, obiettivo e funzione del componente: la classe si occupa di inviare le credenziali inserite per l'accesso al server, e restituirne la risposta. Gestisce inoltre le operazioni di registrazione di un nuovo utente e di disconnessione dal sistema.
Relazioni d'uso con altre componenti:

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

Attività svolte e dati trattati: la classe si occupa della comunicazione tra client e server per quanto riguarda le operazioni di autenticazione, registrazione e disconnessione dal sistema.

6.2.2 mytalk.client.communication.CallCommunication

Tipo, obiettivo e funzione del componente: la classe si occupa di avviare la comunicazione tra utenti con il server.
Relazioni d'uso con altre componenti:

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

Attività svolte e dati trattati: la classe si occupa di avviare una chiamata tra due o più utenti.

6.2.3 mytalk.client.communication.ChatCommunication

Tipo, obiettivo e funzione del componente: la classe si occupa del trasferimento di messaggi di testo tra due o più utenti.
Relazioni d'uso con altre componenti:

- **La classe utilizza**
  - `mytalk.client.collection.TextMessagesCollection`

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

Attività svolte e dati trattati: la classe gestisce la comunicazione chat tra due o più utenti.

6.2.4 mytalk.client.communication.ContactsCommunication

Tipo, obiettivo e funzione del componente: la classe si occupa di ricevere la lista degli utenti presenti nel server.
Relazioni d'uso con altre componenti:

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

Attività svolte e dati trattati: la classe si occupa di gestire la lista di tutti gli utenti iscritti al server.

6.2.5 mytalk.client.communication.FileCommunication

Tipo, obiettivo e funzione del componente: la classe si occupa di gestire l'invio e la ricezione di file.
Relazioni d'uso con altre componenti:

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

Attività svolte e dati trattati: la classe si occupa di effettuare il trasferimento tra client di un file salvato in locale.

6.2.6 mytalk.client.communication.NotificationCommunication

Tipo, obiettivo e funzione del componente: la classe si occupa di segnalare notifiche all'utente.
Relazioni d'uso con altre componenti:

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

Attività svolte e dati trattati: la classe si occupa di notificare l'utente di chiamate in arrivo o della presenza di file o messaggi inviati da altri utenti in attesa.

6.2.7 mytalk.client.communication.RecordMessageCommunication

Tipo, obiettivo e funzione del componente: la classe si occupa di inviare e ricevere i messaggi registrati.
Relazioni d'uso con altre componenti:

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

Attività svolte e dati trattati: la classe si occupa di ricevere la richiesta di avvio registrazione e di mandare al server il compito di salvarla.

6.2.8 mytalk.client.communication.TutorialCommunication

Tipo, obiettivo e funzione del componente: la classe si occupa di ricevere l'indirizzo dei videotutorial da caricare.
Relazioni d'uso con altre componenti:

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

Attività svolte e dati trattati: la classe gestisce i videotutorial presenti nel server.

6.2.9 mytalk.client.communication.UserDataCommunication

Tipo, obiettivo e funzione del componente: la classe si occupa di gestire le operazioni di visualizzazione e modifica dei dati dell'account.
Relazioni d'uso con altre componenti:

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

Attività svolte e dati trattati: la classe gestisce l'operazione di modifica delle informazioni dell'account, quali password, nome e cognome.

6.3 Collection

Tipo, obiettivo e funzione del componente: ha lo scopo di rappresentare localmente le collezioni di oggetti necessarie per popolare le viste in modo dinamico.
Relazioni d'uso con altre componenti:

- **Il package utilizza:**
  - `mytalk.client.model`
- **Il package è utilizzato da:**
  - `mytalk.client.view`
  - `mytalk.client.communication`

6.3.1 mytalk.client.collection.ContactsCollection

Tipo, obiettivo e funzione del componente: la classe si occupa di conservare la lista degli utenti registrati al server.
Relazioni d'uso con altre componenti:

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

Attività svolte e dati trattati: la classe si occupa di conservare la lista degli utenti.

6.3.2 mytalk.client.collection.RecordMessagesCollection

Tipo, obiettivo e funzione del componente: la classe si occupa di conservare tutti i collegamenti ai messaggi differiti ricevuti dall'utente.
Relazioni d'uso con altre componenti:

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

  - `mytalk.client.view.RecordMessageView`
  - `mytalk.client.communication.NotificationCommunication`
  - `mytalk.client.communication.RecordMessageCommunication`

- **La classe deriva da:**
  - `mytalk.client.Backbone.Collection`

Attività svolte e dati trattati: la classe contiene i riferimenti ai singoli model, aggregando in un'unica collezione la lista di collegamenti ai messaggi ricevuti.

6.3.3 mytalk.client.collection.TextMessagesCollection

Tipo, obiettivo e funzione del componente: la classe si occupa di conservare tutte le conversazioni testuali.
Relazioni d'uso con altre componenti:

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

Attività svolte e dati trattati: la classe si occupa di conservare tutte le conversazioni testuali intraprese dall'utente dalla sua autenticazione.

6.3.4 mytalk.client.collection.TutorialsCollection

Tipo, obiettivo e funzione del componente: la classe si occupa di conservare la lista dei video tutorial.
Relazioni d'uso con altre componenti:

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

- **La classe deriva da:**
  - `mytalk.client.Backbone.Collection`

Attività svolte e dati trattati: la classe si occupa di conservare la lista di tutti i video tutorial presenti nel server.

6.4 Model

Tipo, obiettivo e funzione del componente: ha lo scopo di rappresentare localmente le informazioni presenti nel database del server al fine di popolare le viste in modo dinamico.
Relazioni d'uso con altre componenti:

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

6.4.1 mytalk.client.model.ContactModel

Tipo, obiettivo e funzione del componente: la classe rappresenta il singolo contatto.
Relazioni d'uso con altre componenti:

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

Attività svolte e dati trattati: la classe contiene i dati di un singolo contatto presente nel server.

6.4.2 mytalk.client.model.RecordMessageModel

Tipo, obiettivo e funzione del componente: la classe si occupa della memorizzazione del collegamento ad un singolo messaggio differito ricevuto dall'utente.
Relazioni d'uso con altre componenti:

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

Attività svolte e dati trattati: la classe si occupa di conservare il collegamento ad un singolo messaggio differito finché l'utente non lo visualizza, non lo cancella o il messaggio non scade.

6.4.3 mytalk.client.model.TextMessageModel

Tipo, obiettivo e funzione del componente: la classe si occupa della memorizzazione di un singolo messaggio di testo.
Relazioni d'uso con altre componenti:

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

Attività svolte e dati trattati: la classe si occupa di conservare per un tempo limitato le comunicazioni testuali da e verso un singolo utente.

6.4.4 mytalk.client.model.TutorialModel

Tipo, obiettivo e funzione del componente: la classe si occupa di conservare i dati riguardanti i tutorial.
Relazioni d'uso con altre componenti:

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

Attività svolte e dati trattati: la classe si occupa di conservare i dati riguardanti un singolo video tutorial.

6.4.5 mytalk.client.model.UserModel

Tipo, obiettivo e funzione del componente: la classe si occupa di tenere aggiornate localmente le informazioni dell'account attuale.
Relazioni d'uso con altre componenti:

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

Attività svolte e dati trattati: la classe si occupa di tenere le informazioni del proprio account restituendo i dati all'utente.

7 Server

Alcune annotazioni:

- La classe `mytalk.server.Laucher` ha il solo scopo di inizializzare le classi degli strati manager e data, in modo da non creare copie che comporterebbero interferenza e malfunzionamento del server. Proprio per la sua funzione di inizializzatore, si trova all'esterno degli strati e verrà modellato come Singleton
- La classe `mytalk.server.ServerMyTalk` ha lo scopo di avviare il server, utilizzando la classe Launcher, e inizializzando le classi dello strato transfer affinché il server possa comunicare con il lato client
- Il package `mytalk.server.shared` contiene le strutture dati utilizzate in tutto il server[2], per evitare ripetizioni questa relazione verrà dichiarata esplicitamente solo nella descrizione delle classi stesse, e non nelle classi degli altri package
- I dati vengono salvati sia nel package shared, sopra indicato, sia nella base di dati per ridurre l'accesso e le modifiche su quest'ultima. Anche se concettualmente considereremo la base di dati interna allo strato data, per motivi tecnici essa non sarà posizionata in nessun package
- Le classi `mytalk.server.dao.*SQL` agiscono sulla base di dati attraverso la classe JavaConnectionSQLite, questo passaggio sarà considerato implicito nelle descrizioni delle classi

[2] Ad eccezione delle classi `mytalk.server.transfer.CallTransfer`, `mytalk.server.transfer.ChatTransfer`, `mytalk.server.transfer.FileTransfer` e `mytalk.server.dao.JavaConnectionSQLite`.

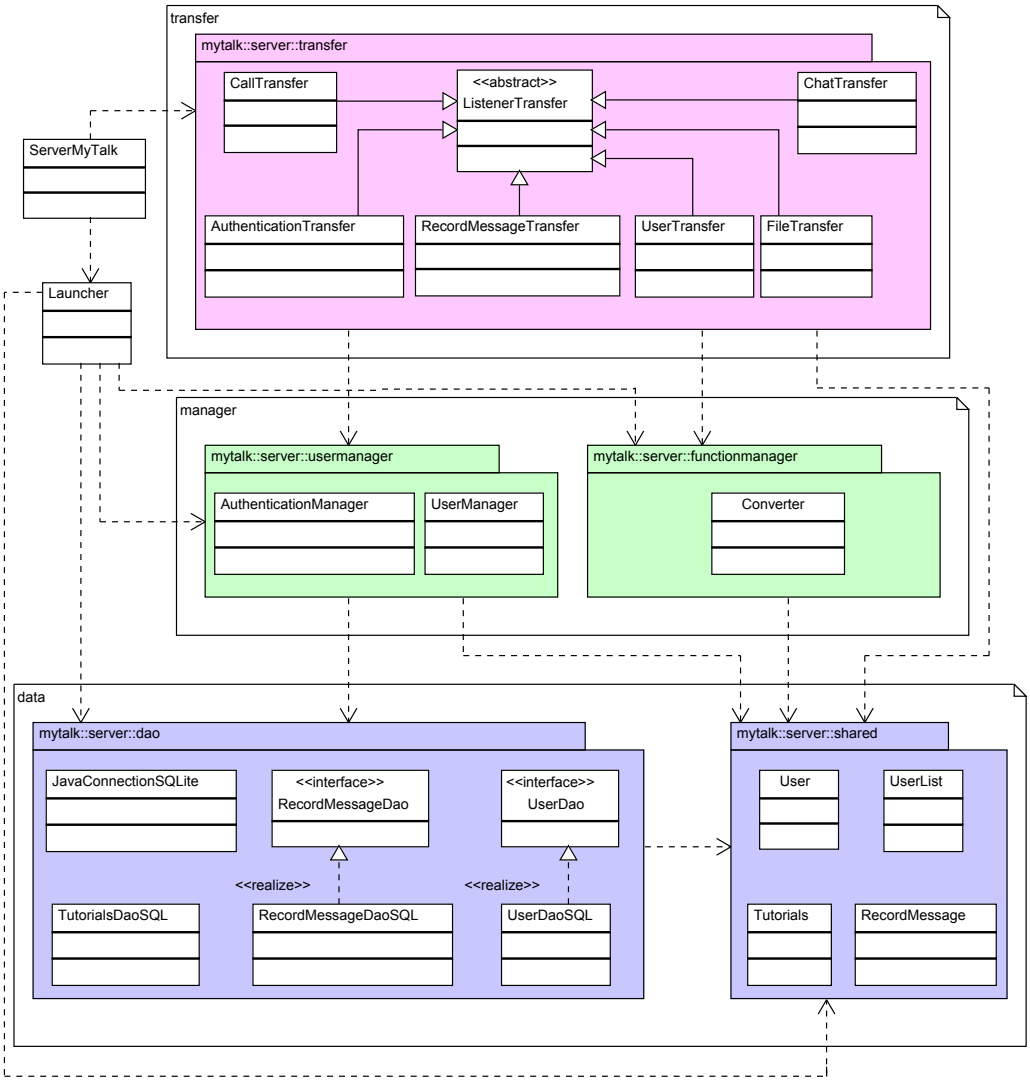

Figura 8: Diagramma dei package dell'architettura del Server

7.1 Transfer Layer

Tipo, obiettivo e funzione del componente: lo strato transfer si occupa di interfacciare il server con il client e di inizializzare le comunicazioni peer-to-peer.
Relazioni d'uso con altre componenti:

- **Utilizza:**
  - `mytalk.server.usermanager`

  - `mytalk.server.functionmanager`
  - `mytalk.server.shared`

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

7.1.1 mytalk.server.transfer.ListenerTransfer

Tipo, obiettivo e funzione del componente: è una classe astratta che implementa l'interfaccia `org.jwebsocket.listener.WebSocketServerTokenListener`, e fornisce i metodi per l'invio dei pacchetti al client.
Relazioni d'uso con altre componenti:

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

Attività svolte e dati trattati: gestisce e memorizza la collezione dei connettori che puntano agli utenti connessi, inoltre contiene i metodi per inviare un singolo pacchetto ad un dato connettore, e per inviare un singolo pacchetto in broadcast a tutti gli utenti connessi. Inizializza un'istanza della classe Converter, che viene utilizzata dalle altre classi del package per convertire gli oggetti di tipo shared in stringhe JSON.

7.1.2 mytalk.server.transfer.AuthenticationTransfer

Tipo, obiettivo e funzione del componente: la classe si occupa del trasferimento delle richieste di autenticazione e registrazione al server e della gestione dei processi, quindi la connessione/disconnessione di un computer al server.
Relazioni d'uso con altre componenti:

- **Utilizza:**

  - `mytalk.server.usermanager.AuthenticationManager`
  - `mytalk.server.usermanager.UserManager`
  - `mytalk.server.shared.Tutorials`
  - `mytalk.server.shared.User`
  - `mytalk.server.functionmanager.Converter` (ereditata da ListenerTransfer)

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

- **Comunica con:**
  - `mytalk.client.communication.AuthenticationCommunication`
  - `mytalk.client.communication.ContactsCommunication`
  - `mytalk.client.communication.TutorialCommunication`

Attività svolte e dati trattati:

- Gestisce le richieste da parte del lato client di autenticazione; se le operazioni vanno a buon fine manda gli aggiornamenti in broadcast a tutti gli utenti connessi. Permette la visione della lista degli utenti connessi, comunicando con UserManager; il vettore verrà convertito dalla classe Converter per poter essere mandato in formato corretto al client
- Gestisce la connessione al server dei vari utenti. In particolare quando un dispositivo si connette al server viene aggiunto il suo connettore alla lista presente nella classe ListenerTransfer e gli viene inviata la lista di tutorial, quando si disconnette viene rimosso il connettore dalla lista, e se non è stata fatta correttamente l'operazione di *logout* viene fatta automaticamente

7.1.3 mytalk.server.transfer.CallTransfer

Tipo, obiettivo e funzione del componente: la classe si occupa di trasferire i pacchetti tra i client che desiderano iniziare, o terminare, una comunicazione.
Relazioni d'uso con altre componenti:

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

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

Attività svolte e dati trattati: gestisce il trasferimento dei pacchetti necessari al fine di creare il canale WebRTC per la comunicazione tra i due utenti, utilizza il vettore di connettori per individuare gli utenti.

7.1.4 mytalk.server.transfer.ChatTransfer

Tipo, obiettivo e funzione del componente: la classe si occupa di gestire la comunicazione tramite messaggi di testo.
Relazioni d'uso con altre componenti:

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

Attività svolte e dati trattati: gestisce il trasferimento dei pacchetti, contenenti messaggi di testo, tra i due utenti che stanno comunicando via chat.

7.1.5 mytalk.server.transfer.FileTransfer

Tipo, obiettivo e funzione del componente: la classe si occupa della comunicazione per l'invio dei file.
Relazioni d'uso con altre componenti:

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

Attività svolte e dati trattati: si occupa del trasferimento di file tra due utenti.

7.1.6 mytalk.server.transfer.RecordMessageTransfer

Tipo, obiettivo e funzione del componente: la classe si occupa del trasferimento del messaggio registrato dal client al server e del successivo invio al destinatario.
Relazioni d'uso con altre componenti:

- **Utilizza:**
  - `mytalk.server.usermanager.UserManager`
  - `mytalk.server.shared.RecordMessage`
  - `mytalk.server.functionmanager.Converter` (ereditata da ListenerTransfer)

- **Viene utilizzata da:**
  - `mytalk.server.usermanager.UserManager`
- **Estende:**
  - `mytalk.server.transfer.ListenerTransfer`
- **Comunica con:**
  - `mytalk.client.communication.RecordMessageCommunication`
  - `mytalk.client.communication.NotificationCommunication`

Attività svolte e dati trattati: comunica con la classe UserManager per la gestione dei messaggi registrati, in particolare per le richieste da parte del client di registrazione e di cancellazione di un messaggio nel database, e per l'invio dei messaggi all'utente.

7.1.7 mytalk.server.transfer.UserTransfer

Tipo, obiettivo e funzione del componente: la classe si occupa della gestione dei dati dell'utente.
Relazioni d'uso con altre componenti:

- **Utilizza:**
  - `mytalk.server.usermanager.UserManager`
  - `mytalk.server.shared.User`
  - `mytalk.server.functionmanager.Converter` (ereditata da ListenerTransfer)
- **Estende:**
  - `mytalk.server.transfer.ListenerTransfer`
- **Comunica con:**
  - `mytalk.client.communication.UserDataCommunication`

Attività svolte e dati trattati: gestisce le richieste di modifica dei dati degli utenti inviate dal client comunicando con UserManager, se le operazioni hanno buon esito manda gli aggiornamenti in broadcast.

7.2 Manager Layer

Tipo, obiettivo e funzione del componente: lo strato manager del server si occupa delle funzionalità di comunicazione con la base di dati, effettuando tutti i controlli necessari, e di conversione dei dati da formati Java a formati JSON.
Relazioni d'uso con altre componenti:

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

7.2.1 mytalk.server.usermanager.AuthenticationManager

Tipo, obiettivo e funzione del componente: la classe si occupa di stabilire la riuscita o il fallimento di un tentativo di autenticazione o di registrazione.
Relazioni d'uso con altre componenti:

- **Utilizza:**
  - `mytalk.server.dao.UserDao`
  - `mytalk.server.dao.UserDaoSQL`
  - `mytalk.server.shared.UserList`
  - `mytalk.server.shared.User`
- **Viene utilizzata da:**
  - `mytalk.server.transfer.AuthenticationTransfer`

Attività svolte e dati trattati: comunica con la classe AuthenticationTransfer e lo strato data per la gestione della registrazione e delle operazioni di login e logout degli utenti, inoltre fornisce alla classe AuthenticationTransfer il metodo per la visualizzazione di tutti gli utenti presenti nel server.

7.2.2 mytalk.server.usermanager.UserManager

Tipo, obiettivo e funzione del componente: la classe si occupa delle operazioni di aggiornamento delle informazioni sugli utenti e di gestione lato server dei messaggi audio/video.
Relazioni d'uso con altre componenti:

- **Utilizza:**

  - `mytalk.server.dao.UserDao`
  - `mytalk.server.dao.UserDaoSQL`
  - `mytalk.server.dao.RecordMessageDao`
  - `mytalk.server.dao.RecordMessageDaoSQL`
  - `mytalk.server.shared.UserList`
  - `mytalk.server.shared.User`
  - `mytalk.server.shared.RecordMessage`

- **Viene utilizzata da:**

  - `mytalk.server.transfer.AuthenticationTransfer`
  - `mytalk.server.transfer.RecordMessageTransfer`
  - `mytalk.server.transfer.UserTransfer`

Attività svolte e dati trattati: rende possibile la comunicazione tra lo strato data e lo strato transfer per permettere la modifica dei dati degli utenti e la gestione dei messaggi audio/video.

7.2.3 mytalk.server.functionmanager.Converter

Tipo, obiettivo e funzione del componente: la classe si occupa delle operazioni di conversione da oggetti propri del server, definiti nel package `mytalk.server.shared`, a stringhe JSON.
Relazioni d'uso con altre componenti:

- **Utilizza:**

  - `mytalk.server.shared.Tutorials`
  - `mytalk.server.shared.User`
  - `mytalk.server.shared.RecordMessage`

- **Viene utilizzato da:**

  - `mytalk.server.transfer.ListenerTransfer`
  - `mytalk.server.transfer.AuthenticationTransfer`
  - `mytalk.server.transfer.RecordMessageTransfer`
  - `mytalk.server.transfer.UserTransfer`

Attività svolte e dati trattati: converte gli oggetti Tutorials, User e RecordMessage in stringhe JSON compatibili con la comunicazione tramite WebSocket.

7.3 Data Layer

Tipo, obiettivo e funzione del componente: lo strato data contiene i dati persistenti e il package `mytalk.server.dao` che è l'unico package ad operare modifiche sulla base di dati.
Relazioni d'uso con altre componenti:

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

7.3.1 mytalk.server.dao.JavaConnectionSQLite

Tipo, obiettivo e funzione del componente: la classe si occupa della connessione con il database e delle operazioni CRUD su di esso.
Relazioni d'uso con altre componenti:

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

Attività svolte e dati trattati: la classe permette la connessione con la base di dati, inoltre fornisce i metodi per la lettura, il salvataggio e la modifica della base di dati.

7.3.2 mytalk.server.dao.RecordMessageDao

Tipo, obiettivo e funzione del componente: l'interfaccia con cui comunicherà lo strato manager per la gestione dei messaggi video/audio.
Relazioni d'uso con altre componenti:

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

7.3.3 mytalk.server.dao.RecordMessageDaoSQL

Tipo, obiettivo e funzione del componente: la classe implementa l'interfaccia RecordMessageDao e fornisce le operazioni per i messaggi audio/video
Relazioni d'uso con altre componenti:

- **Utilizza:**

  - `mytalk.server.dao.JavaConnectionSQLite`
  - `mytalk.server.shared.RecordMessage`
  - `mytalk.server.shared.User`
  - `mytalk.server.shared.UserList`

- **Implementata:**
  - `mytalk.server.dao.RecordMessageDao`

- **Viene utilizzata da:**
  - `mytalk.server.usermanager.UserManager`

Attività svolte e dati trattati: si occupa del salvataggio, della cancellazione e del prelievo dei messaggi registrati presenti nella base di dati.

7.3.4 mytalk.server.dao.TutorialsDaoSQL

Tipo, obiettivo e funzione del componente: la classe si occupa di copiare in `mytalk.server.shared.Tutorials` i tutorial presenti nella base di dati.
Relazioni d'uso con altre componenti:

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

Attività svolte e dati trattati: si occupa di inizializzare, riempire e restituire l'oggetto della classe Tutorials.

7.3.5 mytalk.server.dao.UserDao

Tipo, obiettivo e funzione del componente: l'interfaccia con cui comunicherà lo strato manager per tutte le operazioni sugli utenti.
Relazioni d'uso con altre componenti:

- **Viene implementata da:**
  - `mytalk.server.dao.UserDaoSQL`

- **Viene utilizzata da:**
  - `mytalk.server.usermanager.UserManager`
  - `mytalk.server.usermanager.AuthenticationManager`

7.3.6 mytalk.server.dao.UserDaoSQL

Tipo, obiettivo e funzione del componente: la classe implementa l'interfaccia UserDao e fornisce tutte le possibili operazioni sugli utenti.
Relazioni d'uso con altre componenti:

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

Attività svolte e dati trattati: si occupa della creazione e cancellazione di un utente e della modifica dei suoi dati. Inoltre quando la classe viene istanziata per la prima volta, vengono salvati in UserList tutti gli utenti presenti nella base di dati.

7.3.7 mytalk.server.shared.RecordMessage

Tipo, obiettivo e funzione del componente: la classe viene utilizzata per contenere le informazioni di messaggio registrato, questo viene passato da una classe ad un'altra.
Relazioni d'uso con altre componenti:

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

Attività svolte e dati trattati: la classe contiene le informazioni riguardanti un messaggio, ovvero il mittente, il destinatario, l'indirizzo in cui è salvato il messaggio e la data di creazione.

7.3.8 mytalk.server.shared.Tutorials

Tipo, obiettivo e funzione del componente: la classe contiene tutti i tutorial presenti nella base di dati.
Relazioni d'uso con altre componenti:

- **Viene utilizzata da:**
  - `mytalk.server.dao.TutorialsDaoSQL`
  - `mytalk.server.transfer.AuthenticationTransfer`
  - `mytalk.server.functionmanager.Converter`

Attività svolte e dati trattati: la classe preserva la lista di collegamenti ai tutorial video.

7.3.9 mytalk.server.shared.User

Tipo, obiettivo e funzione del componente: la classe contiene le informazioni su un utente contenute nel database, ad eccezione della password, che per motivi di sicurezza si è deciso di mantenere solo nella base di dati.
Relazioni d'uso con altre componenti:

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

Attività svolte e dati trattati: la classe preserva le informazioni riguardanti gli utenti, ovvero username, nome, cognome e indirizzo IP.

7.3.10 mytalk.server.shared.UserList

Tipo, obiettivo e funzione del componente: la classe contiene la lista degli utenti registrati sul server.
Relazioni d'uso con altre componenti:

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

  - `mytalk.server.usermanager.AuthenticationManager`
  - `mytalk.server.usermanager.UserManager`

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

Attività svolte e dati trattati: la classe preserva una lista con tutti gli utenti registrati, per evitare situazioni di incoerenza e interferenza nei dati salvati.[3]

[3] Viene salvata una copia della lista degli utenti presenti nel database direttamente in locale, per evitare la creazione di più copie. Ciò implica un consumo di memoria non trascurabile nel server, ma il vantaggio è che la memoria occupata sarà sempre fissa.

8 Tracciamento componenti-requisiti

Componente	Classi	Requisiti
CCLI1	`mytalk.client.view.AuthenticationView` `mytalk.client.communication.AuthenticationCommunication`	RUFO 1 RUFO 1.1 RUFO 1.2 RUFO 1.3 RUFO 1.4 RUFO 1.5 RUFO 2 RUFO 2.1 RUFO 2.2 RUFO 8

CCLI2	`mytalk.client.view.UserDataView`
	`mytalk.client.communication.UserDataCommunication`
	`mytalk.client.model.UserModel`
		RUFF 3
		RUFF 3.1
		RUFF 3.2
		RUFF 3.3
		RUFF 3.4
		RUFF 3.5

CCLI3	`mytalk.client.view.TutorialView`
	`mytalk.client.communication.TutorialCommunication`
	`mytalk.client.collection.TutorialsCollection`
	`mytalk.client.model.TutorialModel`
		RUFF 4
		RUFF 4.1

CCLI4	`mytalk.client.view.SideView`
	`mytalk.client.view.ContactView`
	`mytalk.client.communication.ContactsCommunication`
	`mytalk.client.collection.ContactsCollection`
	`mytalk.client.model.ContactModel`
		RUFO 5
		RUFF 5.1
		RUFO 6.3

CCLI5	`mytalk.client.view.NotificationView` `mytalk.client.communication.NotificationCommunication`	RUFF 7 RUFF 7.1 RUFF 7.2 RUFF 7.3 RUFF 7.4 RUFF 7.7 RUFF 7.8 RUFF 7.9

CCLI6	`mytalk.client.view.CallView`
	`mytalk.client.view.FunctionsView`
	`mytalk.client.view.StatisticsView`
	`mytalk.client.communication.CallCommunication`
	`mytalk.client.model.ContactModel`
		RUFO 6
		RUFO 6.1
		RUFO 6.2
		RUFO 6.4
		RUFO 6.5
		RUFF 6.6
		RUFF 6.7
		RUFD 6.8
		RUFD 6.9

		RUFD 6.13 RUFO 6.15
CCLI7	mytalk.client.view.RecordMessageView mytalk.client.communication.RecordMessageCommunication	RUFF 6.10 RUFF 6.11 RUFF 7.5 RUFF 7.6
CCLI8	mytalk.client.view.ChatView mytalk.client.communication.ChatCommunication mytalk.client.collection.TextMessageCollection mytalk.client.model.TextMessageModel	RUFF 6.12
CCLI9	mytalk.client.view.FileView mytalk.client.communication.FileCommunication	RUFD 6.14 RUFD 6.14.1

CSER1	mytalk.server.transfer.ListenerTransfer mytalk.server.transfer.AuthenticationTransfer mytalk.server.functionmanager.Converter mytalk.server.usermanager.AuthenticationManager mytalk.server.usermanager.UserManager mytalk.server.dao.LoginDao mytalk.server.dao.LoginDaoSQL mytalk.server.dao.UserDao mytalk.server.dao.UserDaoSQL mytalk.server.shared.UserList mytalk.server.shared.User	RUFO 1 RUFO 1.1 RUFO 2 RUFO 5 RUFF 5.1 RUFO 8
CSER2	mytalk.server.transfer.ListenerTransfer mytalk.server.transfer.UserTransfer mytalk.server.usermanager.UserManager mytalk.server.functionmanager.Converter mytalk.server.dao.UserDao mytalk.server.dao.UserDaoSQL mytalk.server.shared.UserList mytalk.server.shared.User	RUFF 3 RUFF 3.1 RUFF 3.2 RUFF 3.3
CSER3	mytalk.server.transfer.ListenerTransfer mytalk.server.transfer.CallTransfer	RUFO 6 RUFO 6.1 RUFO 6.2 RUFO 6.4 RUFO 6.5 RUFF 6.6 RUFF 6.7 RUFD 6.13 RUFO 6.15 RUFF 7.2 RUFF 7.3
CSER4	mytalk.server.transfer.ListenerTransfer mytalk.server.transfer.ChatTransfer	RUFF 6.12 RUFD 6.14

	mytalk.server.transfer.FileTransfer	RUFF 7.7 RUFF 7.8 RUFF 7.9
CSER5	mytalk.server.transfer.RecordMessageTransfer mytalk.server.usermanager.UserManager mytalk.server.functionmanager.Converter mytalk.server.dao.RecordMessageDao mytalk.server.dao.RecordMessageDaoSQL mytalk.server.shared.RecordMessage	RUFF 6.10 RUFF 6.11 RUFF 7.5 RUFF 7.6
CSER6	mytalk.server.transfer.AuthenticationTransfer mytalk.server.functionmanager.Converter mytalk.server.dao.TutorialsDaoSQL mytalk.server.shared.Tutorials	RUFF 4 RUFF 4.1

Tabella 2: Tracciamento tra componenti e requisiti

9 Tracciamento requisiti-componenti

Requisito	Componente Client	Componente Server
RUFO 1	CCLI1	CSER1
RUFO 1.1		
RUFO 1.2		-
RUFO 1.3		
RUFO 1.4		
RUFO 1.5		
RUFO 2		CSER1
RUFO 2.1		-
RUFO 2.2		
RUFF 3	CCLI2	CSER2
RUFF 3.1		
RUFF 3.2		
RUFF 3.3		
RUFF 3.4		-
RUFF 3.5		
RUFF 4	CCLI3	CSER6
RUFF 4.1		
RUFO 5	CCLI4	CSER1
RUFF 5.1		
RUFO 6	CCLI6	CSER3
RUFO 6.1		
RUFO 6.2		
RUFO 6.3	CCLI4	-
RUFO 6.4	CCLI6	CSER3
RUFO 6.5		
RUFF 6.6		
RUFF 6.7		
RUFD 6.8		-
RUFD 6.9		
RUFF 6.10	CCLI7	CSER5
RUFF 6.11		
RUFF 6.12	CCLI8	CSER4
RUFD 6.13	CCLI6	CSER3
RUFD 6.14	CCLI9	CSER4
RUFD 6.14.1		-
RUFO 6.15	CCLI6	CSER3
RUFF 7	CCLI5	-
RUFF 7.1		
RUFF 7.2		CSER3
RUFF 7.3		
RUFF 7.4		-

RUFF 7.5	CCLI7	CSER5
RUFF 7.6		
RUFF 7.7	CCLI5	CSER4
RUFF 7.8		
RUFF 7.9		
RUFO 8	CCLI1	CSER1

Tabella 3: Tracciamento tra requisiti e componenti

10 Diagramma delle attività

In questa sezione si illustreranno i diagrammi delle attività relativi alle possibili interazioni di un utente con il front-end del prodotto **MyTalk**.

10.1 Utente non autenticato

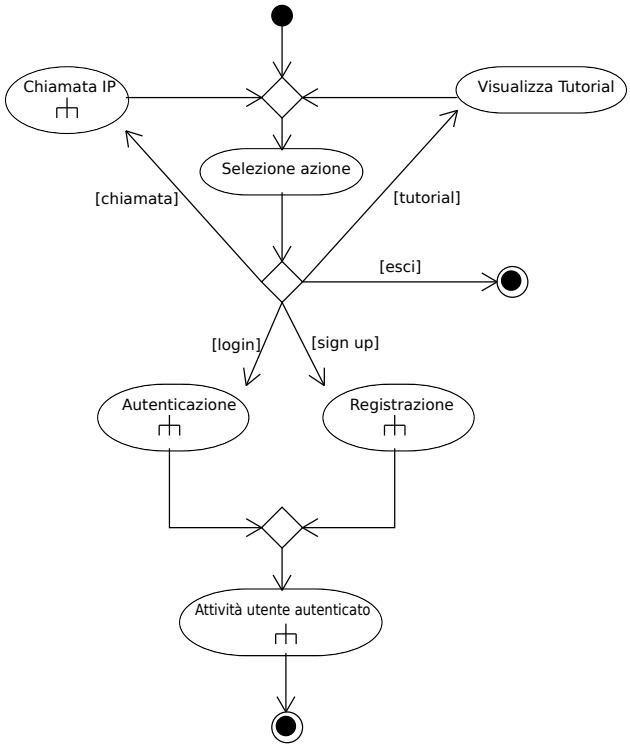

Figura 9: Diagramma delle attività di un utente non autenticato

Il diagramma rappresentato in figura 9 spiega le operazioni che un utente non autenticato potrà svolgere per interagire con il sistema. Una volta aperta la pagina web l'utente potrà decidere di guardare i video tutorial, contattare direttamente un utente tramite l'inserimento dell'indirizzo IP (vedasi sezione 10.6), iscriversi al sito (vedasi sezione 10.2) ed effettuare l'accesso con le proprie credenziali (vedasi sezione 10.3). Una volta effettuata l'autenticazione, l'utente avrà a disposizione le funzionalità offerte per l'utente autenticato (vedasi sezione 10.4). Si fa presente che una volta che una registrazione va a buon fine, viene in automatico effettuata l'autenticazione.

10.2 Registrazione

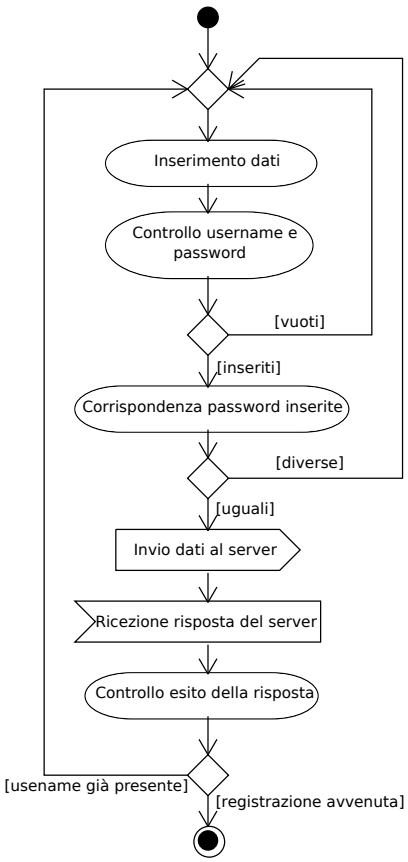

Figura 10: Diagramma delle attività di registrazione

Il diagramma rappresentato in figura 10 spiega le operazioni che un utente potrà svolgere una volta che quest'ultimo abbia deciso di iscriversi al prodotto. Il modulo di registrazione prevede l'inserimento di un username, di una password, del nome e del cognome dell'utente. L'inserimento dello username e della password sono obbligatori, mentre l'inserimento del nome e del cognome sono facoltativi. Il modulo di registrazione richiede di ripetere due volte la password per verificare che quella inserita sia corretta. In caso le due password non fossero uguali il sistema visualizzerà l'errore e richiederà nuovamente l'inserimento dei dati.
Una volta inseriti tutti campi obbligatori ed eventuali opzionali, se lo username non risulta già in uso, l'utente sarà registrato presso il server e autenti-

cato automaticamente, altrimenti verrà richiesto di reinserire almeno i campi obbligatori.

10.3 Autenticazione

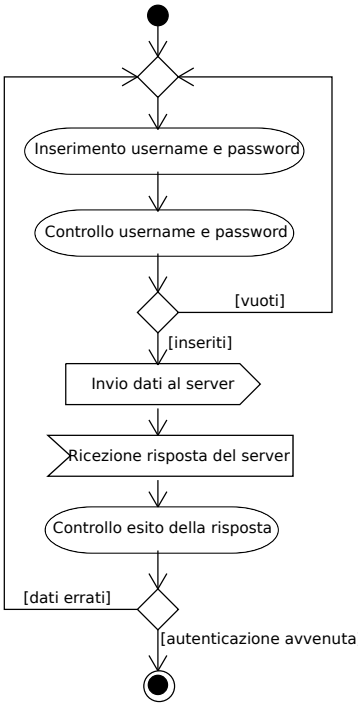

Figura 11: Diagramma delle attività di autenticazione

Il diagramma rappresentato in figura 11 spiega le operazioni che un utente potrà svolgere una volta che quest'ultimo abbia deciso di autenticarsi. Il modulo di autenticazione prevede l'immissione obbligatoria di un username e di una password. Una volta inseriti questi campi, sarà effettuato un controllo per vedere se l'utente risulta registrato presso il servizio e, in caso di esito positivo, l'utente sarà autenticato automaticamente, altrimenti dovrà inserire i dati nuovamente.

10.4 Utente autenticato

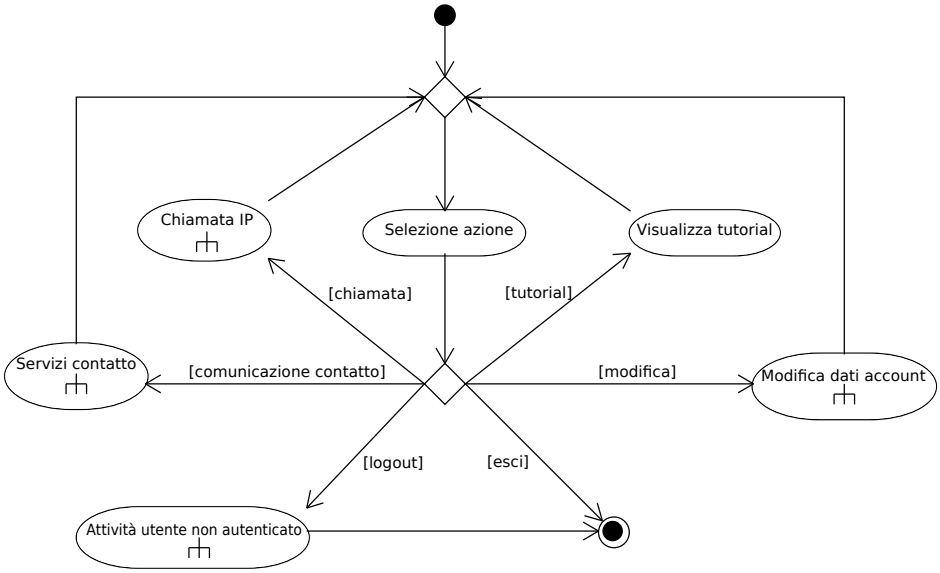

Figura 12: Diagramma delle attività che mostra le operazioni a disposizione di un utente autenticato

Il diagramma rappresentato in figura 12 spiega le operazioni che un utente autenticato potrà svolgere per interagire con il sistema. Una volta che l'utente abbia effettuato l'autenticazione potrà decidere di guardare i video tutorial, contattare direttamente un utente tramite l'inserimento dell'indirizzo IP (vedasi sezione 10.6), usufruire dei servizi messi a disposizione per contattare un utente registrato presso il server (vedasi sezione 10.7), modificare le proprie credenziali (vedasi sezione 10.3) e deautenticarsi dal sistema. Una volta effettuato il logout, l'utente avrà a disposizione solo le funzionalità offerte per l'utente non autenticato (vedasi sezione 9).

10.5 Modifica dati account

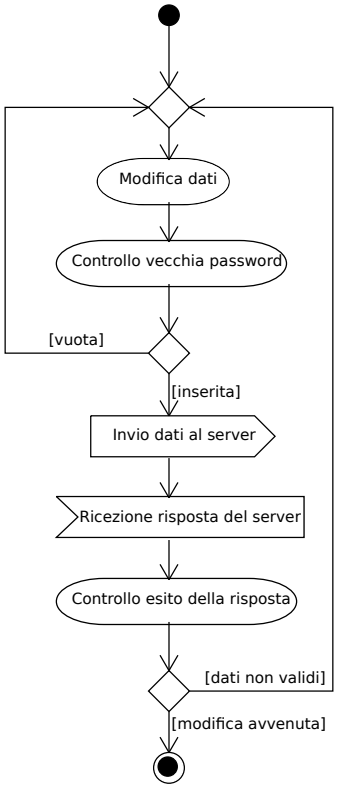

Figura 13: Diagramma delle attività di modifica dei dati account

Il diagramma rappresentato in figura 13 spiega le operazioni che un utente potrà svolgere una volta che quest'ultimo abbia deciso di modificare le proprie credenziali (i dati che si potranno cambiare saranno il nome, il cognome e la password).
Prima di effettuare eventuali modifiche ai dati, per motivi di sicurezza, verrà richiesto di inserire la password corrente. In caso la password inserita sia errata verrà richiesto nuovamente di inserirla. In caso contrario, verranno passati al server i dati da cambiare. A questo punto il server cercherà di modificare i campi presenti coi dati passati e, nel caso in cui le modifiche non avverranno a causa di un errore (come per esempio un campo dato non valido), il sistema esporrà un errore e richiederà nuovamente l'inserimento dei dati. Se invece le modifiche avverranno con successo, il server invierà un messaggio che attesterà la riuscita della modifica dei dati.

10.6 Contattare un indirizzo IP

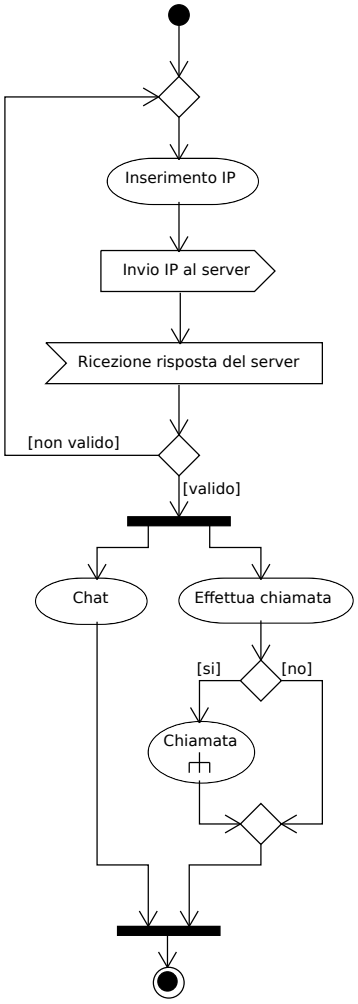

Figura 14: Diagramma delle attività per comunicare con un indirizzo IP

Il diagramma rappresentato in figura 14 spiega le operazioni che un utente potrà svolgere una volta che quest'ultimo abbia deciso di contattare una persona tramite l'inserimento dell'indirizzo IP[4]. Una volta inserito l'IP, questo verrà mandato al server che ne controllerà la validità. Se l'indirizzo non sarà valido, verrà mostrato un messaggio d'errore e sarà richiesto di inserire nuovamente l'IP. In

[4] Azione che si potrà effettuare sia da autenticati che da non autenticati.

caso contrario verrà mandata la notifica di richiesta di chiamata al destinatario e, se quest'ultimo accetterà la richiesta, verrà aperto il canale di comunicazione tra i soggetti. L'utente avrà la possibilità di scambiare messaggi testuali con la persona desiderata, e contemporaneamente, se lo desidera, effettuare una chiamata. Per guardare le funzionalità che comporta la chiamata vedasi sezione 10.8.

10.7 Contattare un utente

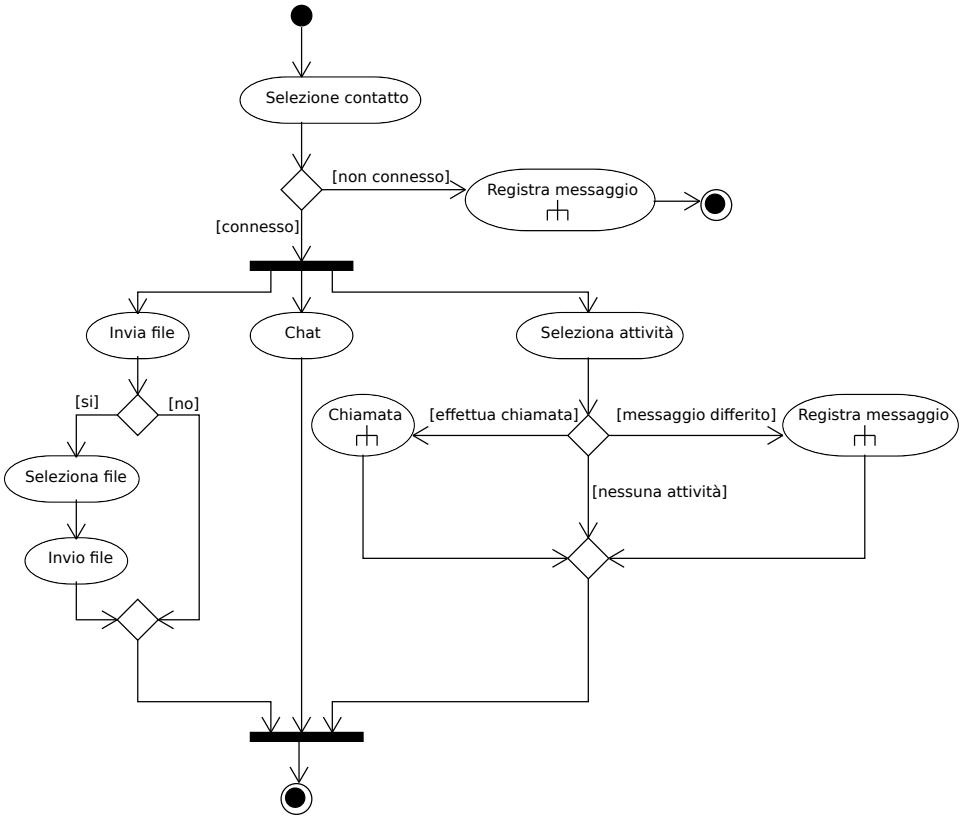

Figura 15: Diagramma delle attività dei servizi di un utente

Il diagramma rappresentato in figura 15 spiega le operazioni che un utente potrà svolgere una volta che quest'ultimo abbia deciso di contattare un altro utente registrato presso il server.
Se l'utente selezionato risultasse offline l'unico servizio messo a disposizione sarà quello di registrare un messaggio differito e inviarglielo (vedasi sezione 10.9). Nel caso in cui risultasse online si potranno scambiare dei messaggi testuali. Inoltre, parallelamente all'invio e ricezione di messaggi testuali, se lo si desidera, si

potranno inviare file e si potrà registrare un messaggio differito (vedasi sezione 10.9) oppure effettuare una chiamata (vedasi sezione 10.8). Per l'invio dei file il mittente dovrà selezionare il file desiderato per inviarlo al destinatario.

10.8 Chiamata

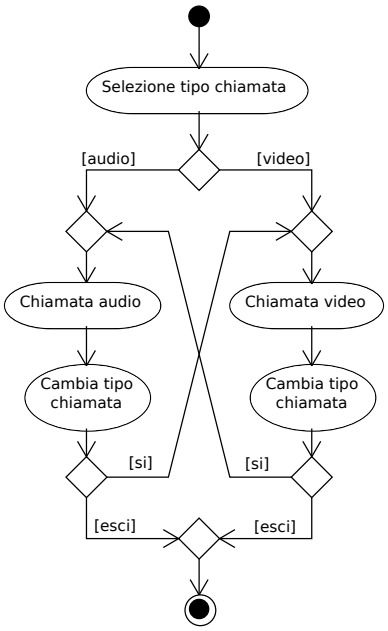

Figura 16: Diagramma delle attività della chiamata

Il diagramma rappresentato in figura 16 spiega le operazioni che un utente potrà svolgere quando deciderà di avviare una chiamata.
Quest'ultimo potrà scegliere se avviare una chiamata audio o video. Sarà possibile, una volta avviata una chiamata, cambiare la tipologia (da audio a video e viceversa) senza interrompere la chiamata. Inoltre l'utente potrà terminare la chiamata quando vuole (il termine della chiamata non implica che non si potranno scrivere messaggi testuali e/o inviare file).

10.9 Registrazione messaggio

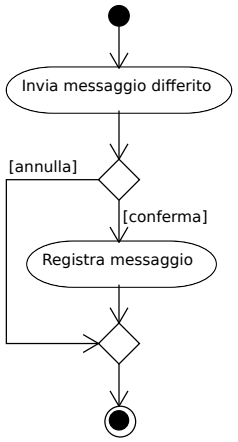

Figura 17: Diagramma delle attività della registrazione di un messaggio

Il diagramma rappresentato in figura 17 spiega le operazioni che un utente può effettuare quando deciderà di inviare un messaggio differito ad un utente registrato presso il server (non sarà possibile inviare un messaggio di questa tipologia ad un utente che non è registrato).
Il mittente a questo punto registrerà il messaggio che verrà salvato nel server.
Il messaggio differito verrà inviato al destinatario la prima volta che si autentica.

10.10 Gestione delle notifiche

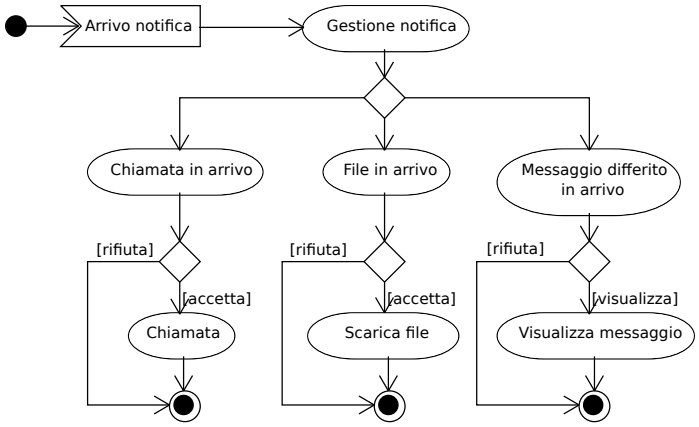

Figura 18: Diagramma delle attività di gestione delle notifiche

Il diagramma rappresentato in figura 18 spiega le operazioni che un utente potrà svolgere una volta che quest'ultimo abbia ricevuto una notifica dal server.
Se la notifica riguarderà una chiamata in arrivo, l'utente potrà scegliere se accettare (e quindi avviare la chiamata) o rifiutarla. Se la notifica riguarderà un file in arrivo l'utente potrà scegliere se accettare (e quindi scaricare) il file o rifiutarlo. Infine, se la notifica riguarda un messaggio differito in arrivo, l'utente potrà scegliere di visualizzare il messaggio o rifiutarlo.

11 Prototipo interfaccia utente

In questo capitolo andremo a descrivere come sarà l'applicazione a livello visivo e le operazioni che l'utente avrà modo di effettuare.

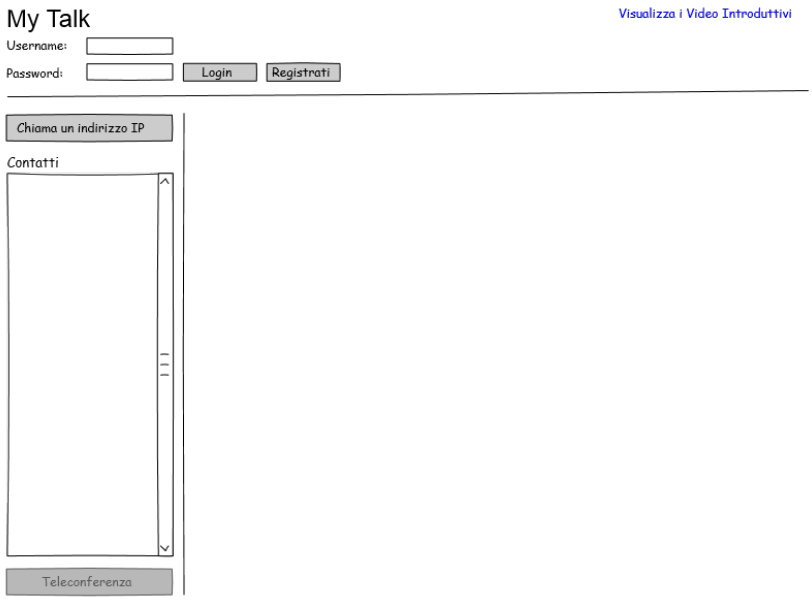

Figura 19: Pagina iniziale

Come si può notare dalla la figura 19 una volta acceduti al sito le uniche operazioni che potremo fare saranno effettuare chiamate sapendo l'indirizzo IP del destinatario, guardare i video tutorial riguardanti il prodotto **MyTalk** e ricevere le chiamate da parte di terzi. Come mostrato in la figura 20 un utente non autenticato può compiere molte delle operazioni consentite ad un utente che ha effettuato il login, ma un utente autenticato può anche chiamare gli utenti presenti nella lista ed eseguire videoconferenze con più utenti.

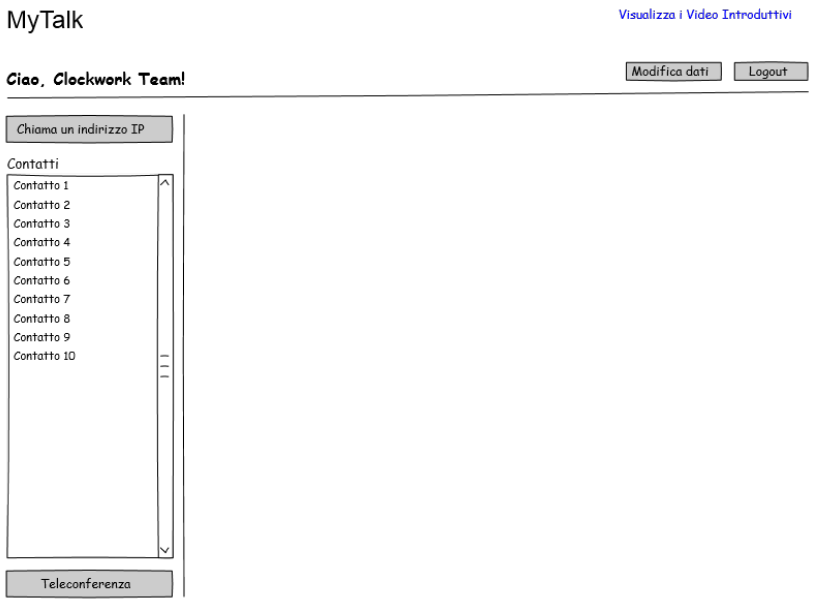

Figura 20: Pagina dopo login

Le figure 21, la figura 22 e la figura 23 mostrano le varie schermate di comunicazioni possibili con i vari utenti. Presente anche la possibilità di registrare le comunicazioni in tempo reale, o inviare messaggi anche se l'utente non è connesso.

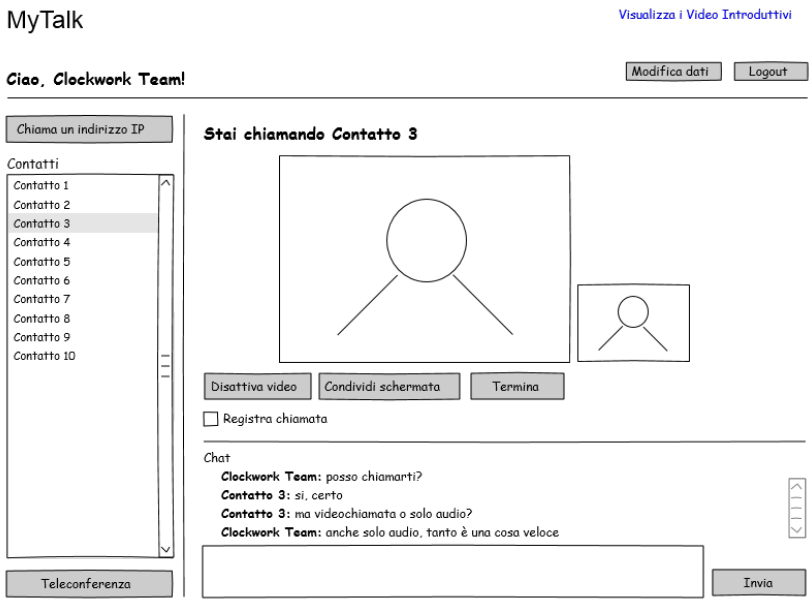

Figura 21: Schermata videochiamata

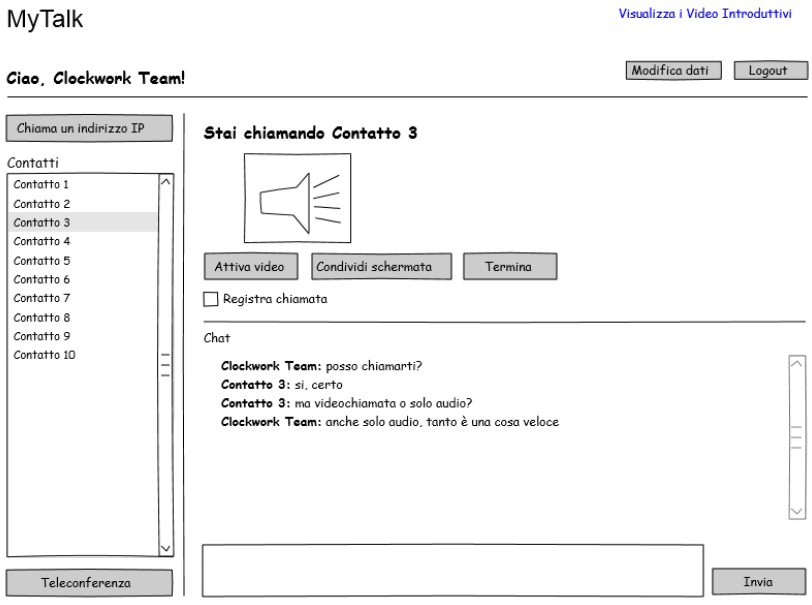

Figura 22: Schermata chiamata audio

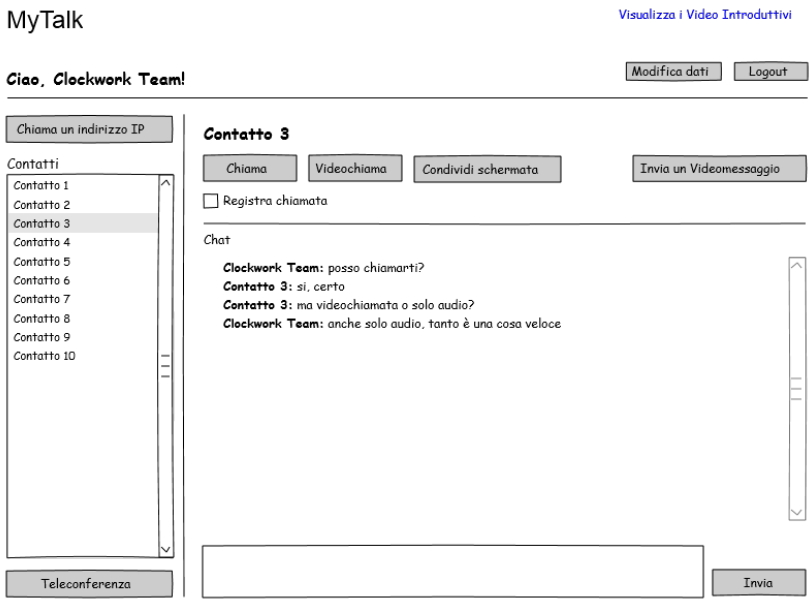

Figura 23: Schermata comunicazione testuale

la figura 24 rappresenta la schermata di videoconferenza.

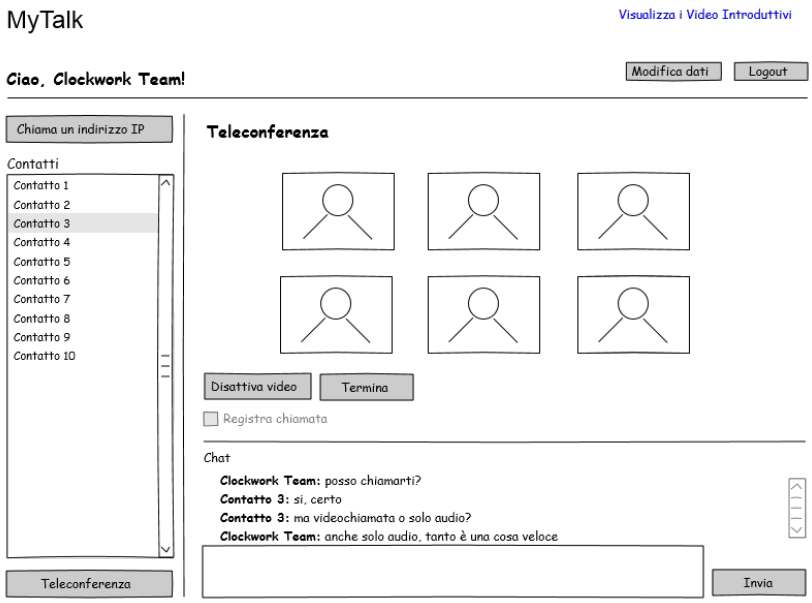

Figura 24: Schermata Videoconferenza

A Backbone.js

Backbone.js nasce come libreria di sviluppo per DocumentCloud, un servizio di condivisione e analisi di documenti, e viene rilasciata come progetto standalone verso la fine del 2010.
Gli obiettivi principali dello sviluppatore sono di realizzare una libreria minimale in grado di fornire degli strumenti di base per organizzare le proprie applicazioni e di poterla usare sia in ambiente browser che server (principalmente con Node.js).
Backbone si focalizza nel fornire metodi utili per manipolare e visualizzare i dati presenti nell'applicazione.
Per la sua architettura, Backbone.js rientra nella categoria delle librerie MV*, in quanto implementa Model e View, ma non ha un componente Controller tradizionale, delegandone i compiti alle View ed al Router. Questo approccio è abbastanza diffuso in ambito JavaScript, a causa della diversa e più complessa gestione dell'interazione utente e dello stato dell'applicazione.
I principali componenti di Backbone.js sono:

- Backbone.Model
- Backbone.Collection
- Backbone.View
- Backbone.Router
- Backbone.Events

A.0.1 Dipendenze

Per il suo funzionamento Backbone.js si appoggia ad alcune librerie esterne:

- Underscore.js ($\geq$ 1.4.3): è una libreria di utility per JavaScript che aggiunge supporto funzionale per collezioni, array e funzioni. Inoltre rende disponibile un sistema di templating HTML che è possibile usare congiuntamente alle View di Backbone.js
- jQuery ($\geq$ 1.7.0): è una libreria che facilita la manipolazione del DOM in JavaScript

A.1 Backbone.Model

Il Model di Backbone.js rappresenta un oggetto discreto contenente una serie di dati sotto forma di attributi, si tratta quindi di un singolo record. I Model vengono aggregati dalle Collection. È inoltre possibile definire funzioni personalizzate per eseguire operazioni sui dati.

```
var ExampleModel = Backbone.Model.extend({

  //attributi di default del model
  defaults: {
    content: "description", //attributo di tipo stringa
    read: false, //attributo di tipo booleano
    number: 12 //attributo numerico
  },

  //funzione di inizializzazione
  initialize: function() {
    if (!this.get("content")) {
      //assicuriamoci che il todo abbia un contenuto
      this.set({"content": this.defaults.content});
    }
  },

  //funzioni definite dall'utente
  toggle: function() {
    var currentRead = this.get("read");
    this.set({read: !currentRead});
  },

  clear: function() {
    //distrugge il model
    this.destroy();
  }
});
```

Il metodo extend() permette di aggiungere proprietà e metodi personalizzati all'istanza del modello ed eventualmente di aggiungere direttamente al costruttore proprietà relative alla classe. Inoltre imposta in modo corretto la catena di prototipizzazione in modo da poter estendere ulteriormente classi derivate.
L'attributo defaults è un oggetto con la lista degli attributi predefiniti con i rispettivi valori.
La funzione initialize() viene invocata quando viene creato il model e può essere usata al posto di ridefinire il costruttore.

A.2 Backbone.Collection

Una collection è un oggetto contenente una raccolta di modelli dello stesso tipo, attraverso il quale è possibile ordinare, filtrare e manipolare i modelli contenuti.

```
var ExampleCollection = Backbone.Collection.extend({

  //model di riferimento
  model: ExampleModel
```

```
});
```

Come per i Model il metodo extend() permette di aggiungere proprietà e metodi personalizzati alla Collection ed imposta in modo corretto la catena di prototipizzazione.
L'attributo model contiente il modello di riferimenti i cui oggetti sono aggregati dalla Collection.

A.3 Backbone.View

Le View in Backbone.js non contengono markup HTML, bensì fungono da tramite fra l'interfaccia ed i modelli, definendone la logica di interazione. La parte di templating vero e proprio è demandata a librerie esterne.

```
var ExampleView = Backbone.View.extend({
  tagName: "li",

  className: "document-row",

  template: _.template(exampleTemplate),

  events: {
    "click .icon":          "open",
    "click .button.edit":   "openEditDialog",
    "click .button.delete": "destroy"
  },

  initialize: function() {
    this.listenTo(this.model, "change", this.render);
  },

  render: function() {
    this.$el.html(this.template(this.model.attributes));
  },

  open: function() {},

  openEditDialog: function() {},

  destroy: function() {}

});
```

Gli attributi tagName e className servono a definire la proprietà el della vista, che identifica l'elemento del DOM a cui è legata e in cui verrà visualizzata. Altri attributi che possono essere usati per definirla sono id e attributes o, in alternativa, la si può specificare in modo esplicito.

L'attributo template contiene il frammento HTML riguardante il layout vero e proprio della vista. In questo esempio si utilizza il sistema di templating di Underscore.js attraverso la sua funzione _.template().
Nell'array events vengono associati gli eventi generati dal DOM alle funzioni atte a gestirli.
La funzione initialize() viene invocata alla creazione della vista.
Nella sua implementazione di default la funzione render() non compie nessuna operazione. Questa funzione va ridefinita al fine di visualizzare correttamente la vista nella pagina web, includendo eventualmente i dati presi dai modelli.

A.4 Backbone.Router[5]

Il Router di Backbone.js ci permette di ottenere un'applicazione che risiede completamente in una singola pagina, ma che tuttavia da la possibilità di navigare fra le proprie viste come si farebbe con un'applicazione web tradizionale.
Oltre al vantaggio di eliminare i tempi di attesa durante il caricamento della pagina e di dare più scorrevolezza all'interazione, un'applicazione di questo tipo simula anche il cambiamento di pagina aggiornando l'URL della pagina nella barra degli indirizzi.

```
var Workspace = Backbone.Router.extend({
  routes: {
    "help":                 "help",    // #help
    "search/:query":        "search",  // #search/kiwis
    "search/:query/p:page": "search"   // #search/kiwis/p7
  },

  //funzioni di gestione delle Routes
  help: function() {
    ...
  },

  search: function(query, page) {
    ...
  }

});
```

A.5 Backbone.Events

Backbone mette a disposizione un sistema che consente di legare un oggetto ad uno o più eventi, in modo da poter lanciare delle funzioni quando questi si verificano. Oltre a poter gestire eventi del DOM possiamo legare eventi personalizzati che avvengono su oggetti.

[5] Dato che, da capitolato, ci è chiesto di creare un'applicazione su una singola pagina web i Router di Backbone.js non verranno utilizzati in **MyTalk.**

```
var Test = Backbone.View.extend({
  events: {
    'click button#start' : 'start',
    'click .field' : 'score',
    'click button#reset' : 'reset'
  },

  //funzioni di gestione degli eventi
  start: function() { ... },
  score: function(event) { ... },
  reset: function() { ... }
});

//dichiaro un oggetto
var object = {};

//estendo Backbone.Events
_.extend(object, Backbone.Events);

//specifico cosa fare se un evento avviene
object.on("alert", function(msg) {
  alert("Triggered " + msg);
});

//lancio l'evento
object.trigger("alert", "an event");
```

A.6 Underscore.js Templates

Underscore.js implementa un sistema di templating che permette di includere comandi e variabili all'interno di frammenti HTML in modo da generare pagine HTML che si adattino al contesto.

```
var compiled = _.template("hello: <%= name %>");
compiled({name : 'moe'});
=> "hello: moe"

var list = "<% _.each(people, function(name) {
  %> <li><%= name %></li> <%
}); %>";
_.template(list, {people : ['moe', 'curly', 'larry']});
=> "<li>moe</li><li>curly</li><li>larry</li>"

var template = _.template("<b><%- value %></b>");
template({value : '<script>'});
=> "<b>&lt;script&gt;</b>"
```

I simboli <% e %> racchiudono le istruzioni non HTML.
<%= name %> verrà sostituito dal valore della variabile name.