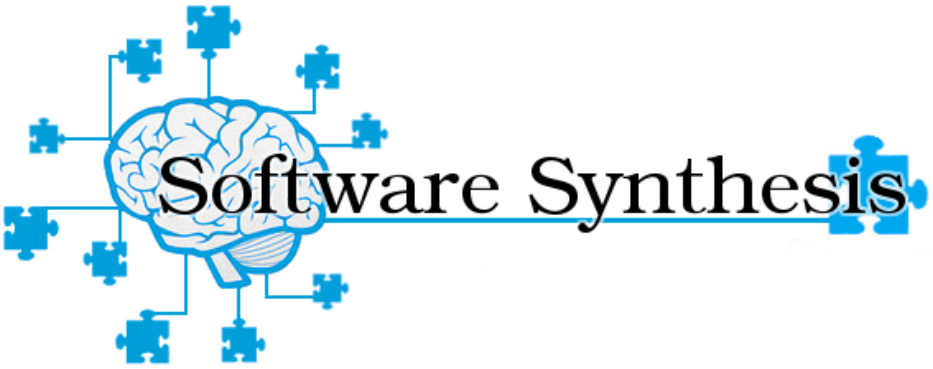

Test e misurazioni

Informazioni sul documento	
Nome file:	test_e_misurazioni.1.0.pdf
Versione:	1.0
Data creazione:	2013-02-20
Data ultima modifica:	2013-03-23
Stato:	Non approvato
Uso:	Esterno
Lista di distribuzione:	Prof. Tullio Vardanega Prof. Riccardo Cardin Dott. Gregorio Piccoli Team SoftwareSynthesis
Redattori:	Stefano Farronato
Approvato da:	Andrea Rizzi
Verificatori:	Diego Beraldin

Storia delle modifiche

Versione	Descrizione intervento	Membro	Ruolo	Data
1.0	Approvazione documento	Andrea Rizzi	Responsabile	2013-03-23
0.9	correzione errori segnalati dal verificatore	Stefano Farronato	Verificatore	2013-03-23
0.8	verifica correttezza lessico ortografica e corrispondenza test con quanto descritto	Diego Beraldin	Verificatore	2013-03-23
0.7	aggiunti test sul *model*	Marco Schivo	Verificatore	2013-03-23
0.6	aggiunti test su capitolo e test coverage	Stefano Farronato	Verificatore	2013-03-22
0.5	stesura capitoli sui test di sistema e di utilizzo, aggiunti test su capitolo 4.1	Stefano Farronato	Verificatore	2013-03-22
0.4	stesura capitolo relativo ai test sul *presenter*	Stefano Farronato	Verificatore	2013-03-21
0.3	stesura capitolo relativo ai test sul *model*	Maroo Schivo	Verificatore	2013-03-21
0.2	stesura introduzione con "scopo dei test" e preambolo su "Metriche sul codice"	Stefano Farronato	Verificatore	2013-03-20
0.1	Creazione del documento e stesura delle sezioni "Introduzione" e "Riferimenti".	Stefano Farronato	Verificatore	2013-03-20

Indice

1 Introduzione

1.1 Scopo del prodotto

Con il progetto "MyTalk" si intende un sistema software di comunicazione tra utenti mediante browser senza la necessità di installazione di plugin e/o software esterni. L'utilizzatore avrà la possibilità di interagire con un altro utente tramite una comunicazione audio - audio/video - testuale e, inoltre, ottenere delle statistiche sull'attività in tempo reale.

1.2 Scopo del documento

Il documento ha lo scopo di definire e riportare l'esito dei test effettuati sul codice prodotto, attraverso misurazioni quantitative valutate attraverso metriche definite al fine di assicurare che la qualità del prodotto risultante sia coerente con i requisiti redatti e rispetti quanto redatto nel documento *piano_di_qualifica.3.0.pdf*. I test predisposti saranno descritti esaustivamente sia nella forma che nei risultati ottenuti, al fine di renderne più chiara e oggettiva possibile la comprensione.

1.3 Glossario

Al fine di evitare incomprensioni dovute all'uso di termini tecnici nei documenti, viene redatto e allegato il documento *glossario.3.0.pdf* dove vengono definiti e descritti tutti i termini marcati con una sottolineatura.

2 Riferimenti

2.1 Normativi

piano_di_qualifica.3.0.pdf allegato;

norme_di_progetto.3.0.pdf allegato;

specifica_tecnica.2.0.pdf allegato;

definizione_di_prodotto.1.0.pdf allegato.

analisi_dei_requsiti.3.0.pdf allegato;

2.2 Informativi

Capitolato d'appalto: MyTalk, v1.0, redatto e rilasciato dal proponente Zucchetti s.r.l. reperibile all'indirizzo `http://www.math.unipd.it/~tullio/IS-1/2012/Progetto/C1.pdf`;

testo di consultazione: *Software Engineering (8th edition) Ian Sommerville, Pearson Education / Addison Wesley*;

glossario.3.0.pdf allegato.

2.3 Metriche sul codice

Per la parte *Model* prima di tutto sono state calcolate, al fine di quantificare la "dimensione" di un componente, due metriche: il *Number Of Classes* (NOC) e il *Number Of Method* (NOM). NOC rappresenta il numero totale delle classi che sono rappresentate all'interno del package, mentre il NOM rappresenta il numero dei metodi conteggiati, sempre per package.

La tabella 1 riporta le misurazioni su tali metriche per i package del prodotto MyTalk, evidenziandone i più corposi dal punto di vista implementativo.

Package	NOC	NOM
org.softwaresynthesis.mytalk.server.abook	3	55
org.softwaresynthesis.mytalk.server.abook.servlet	12	33
org.softwaresynthesis.mytalk.authentication	5	22
org.softwaresynthesis.mytalk.authentication.servlet	3	10
org.softwaresynthesis.mytalk.connection	2	10
org.softwaresynthesis.mytalk.dao	4	18

Tabella 1: Misurazioni metriche NOC e NOM

A un livello di dettaglio più elevato, si è deciso di esporre il *Total Line Of Code* (TLOC) e il *Method Lines Of Code* (MLOC). Come suggeriscono i nomi, TLOC rappresenta il numero totale di linee di codice (effettive, ovvero senza contare quelle commentate) all'interno di un'unità di compilazione, MLOC definisce invece il numero totale di linee di codice (sempre effettive) che definiscono i corpi dei metodi.

Come per NOC e NOM riportiamo la tabella **??** con le misurazioni su tali metriche relative ai package di MyTalk.

Package	TLOC	MLOC
org.softwaresynthesis.mytalk.server.abook	390	277
org.softwaresynthesis.mytalk.server.abook.servlet	889	730
org.softwaresynthesis.mytalk.authentication	347	237
org.softwaresynthesis.mytalk.authentication.servlet	225	185
org.softwaresynthesis.mytalk.connection	132	102
org.softwaresynthesis.mytalk.dao	632	138

Tabella 2: Misurazioni metriche TLOC e MLOC del server

Coerentemente con le disposizioni del documento *norme_di_progetto.3.0.pdf* si riporta l'indice di complessità ciclomatica (media) per ogni package. Tale misura è direttamente legata al numero di cammini linearmente indipendenti che compongono il grafo di controllo di flusso. Tale indice dovrebbe essere compreso tra 0 e 10 per definire il codice nella norma.

Infine il *Lack Of Cohesion Of Methods* (LCOM3) indica il livello (medio) di coesione dei metodi: più tale livello risulta basso migliore risulterà essere la progettazione della classe. I valori possono quindi variare tra lo 0 (che indica la massima coesione possibile) e 2 (che ne identifica il valore minimo).

Package	CC	LCOM3
org.softwaresynthesis.mytalk.server	0	0
org.softwaresynthesis.mytalk.server.abook	1,073	0,834
org.softwaresynthesis.mytalk.server.abook.servlet	2,2	0
org.softwaresynthesis.mytalk.authentication	1,682	0,18
org.softwaresynthesis.mytalk.authentication.servlet	1,8	0
org.softwaresynthesis.mytalk.connection	1,769	0,25
org.softwaresynthesis.mytalk.dao	3,895	0

Tabella 3: Misurazioni metriche CC e LCOM3 del server

La complessità ciclomatica media risulta essere di 1,868 con un massimo di 9 in un metodo di `org.softwaresynthesis.mytalk.connection`, precisamente in `PushInbound.java`.

Per la parte presenter vengono presentate due metriche già descritte per la parte *model*: la complessità ciclomatica (media) e il TLOC, che ricordiamo essere il numero totale di linee di codice effettive, ovvero senza contare quelle non vuote o commentate.

Package	CC	TLOC
AccountSettingPanelPresenter.js	2	170
AdressBookPanelPresenter.js	2,95	467
CallHistoryPanelPresenter.js	1,25	78
CommunicationPanelPresenter.js	1,92	220
ContactPanelPresenter.js	0,3	173
GroupPanelPresenter.js	1,5	48
LoginPanelPresenter.js	2	275
MainPanelPresenter.js	1	15
MessagePanelPresenter.js	1,25	125
PresenterMediator.js	1,38	156
RegisterPanelPresenter.js	3,08	246
SearchResultPanelPresenter.js	2,25	67
ToolsPanelPresenter.js	1,14	82

Tabella 4: Misurazioni metriche TLOC e MLOC del presenter

Il valore complessivo che se ne ricava è quindi una complessità ciclomatica media pari a 1,69 con un massimo di 9 in `RegisterPanelPresenter.js`.

3 Scopo dei test

I test riportati hanno lo scopo di dimostrare in modo oggettivo la bontà e la qualità di quanto prodotto a livello di codice. Tali test servono inoltre a rilevare *bug* non evidenziati (o non possibili da evidenziare) durante l'analisi statica del codice e vengono effettuati sui package relativi alla parte di *Model* e *Presenter* del modello MVP generale.

Le verifiche relative alla parte *View* saranno al contrario effettuati successivamente a quelli sui componenti sopracitati, in quanto sarà necessario avere la certezza che tali componenti risultino stabili e privi (per quanto possibile) di *bug*.

Si specifica inoltre che tali verifiche saranno effettuati tramite i test di sistema specificati nel documento *anilisi_ dei_ requisiti.3.0.pdf* allegato.

Il team ha seguito la seguente sequenza di passi per la realizzazione e l'utilizzo delle varie funzioni di test:

- individuazione preliminare delle funzionalità da testare;
- analisi, progettazione e successiva implementazione delle funzioni di test per le funzionalità individuate, stabilendo i dati di *input* e specificati gli *output* attesi;
- esecuzione delle funzioni di test create.
- analisi dei risultati dei test, confrontato i risultati con quelli attesi ed effettuato la correzione di eventuali anomalie riscontrate, verificata infine la ripetibilità del test stesso;
- registrazione dei risultati ottenuti.

I test per la parte *Model* sono stati predisposti ed eseguiti usufruendo della libreria JUnit, in quanto questa parte della struttura è implementata mediante codice Java.

Per il componente *Presenter* si è deciso invece di usufruire di QUnit, coerentemente con il linguaggio scelto (JavaScript) che lo implementa. Tale franework risulta pratico in quanto permette di individuare facilmente quale metodo analizzato ha portato al fallimento del test, inoltre permette una netta separazione tra codice testato e i test effettivi.

4 Model

Per l'esecuzione di questi test si è usata, coerentemente con le norme di progetto, la libreria JUnit.I test su questa parte del sistema hanno lo scopo di verificare che la persistenza dei dati all'interno del server sia gestita correttamente dal database, la corretta gestione dell'autenticazione degli utenti al sistema, e le funzionalità di connessione.

Ogni test è inoltre stato eseguito più volte in modo da avere uno spettro più ampio di misurazioni e produrre una verifica più stabile e oggettiva possibile.

4.1 Package `org.softwaresinthesis.mytalk.server.autentication`

4.1.1 AESAlgorithmTest

Verifica la Classe: `AESAlgorithm`.
Descrizione:l'obiettivo di questo test è assicurare che l'algoritmo di cittografia AES a 128 bit utilizzato dal sistema MyTalk risulti affidabile e riesca a codificare e decodificare una stringa passata.

Tale test si compone di un due metodi:

- `testEncodeAndDecode()` effettua la codifica di una stringa passata, effettua successivamente la decodifica della stessa e la controlla con l'originale per verificarne l'uguaglianza.
- `testToString()` verifica il metodo *toString* che opera la conversione in stringa degli oggetti della classe.

Risultato del test: non sono stati rilevati errori.

4.1.2 AuthenticationDataTest

Verifica la Classe: `AuthenticationData`.
Descrizione: l'obiettivo di questo test è assicurare l'uguaglianza tra due oggetti che rappresentano le credenziali dell'utente inserite volutamente con gli stessi dati. Successivamente verrà effettuata anche una negazione di tale test per assicurarne l'efficacia.
Tale test si compone di due metodi:

- `testEquals()` testa che due oggetti diversi contenenti gli stessi dati passati al metodo restituiscano *true* solo se sono effettivamente uguali;

- `testDifferent()`testa che due oggetti diversi contenenti diversi dati passati al metodo restituiscano *true* solo se sono effettivamente differenti.

Risultato del test: non sono stati rilevati errori.

4.1.3 AuthenticationModuleTest

Verifica la Classe: `AuthenticationModule`.
Descrizione: l'obiettivo del test è verificare le funzionalità di autenticazione offerte dal sistema server dell'applicativo.

Tale test si compone di 3 metodi:

- `testLogin()` verifica che l'accesso al sistema mediante le opportune credenziali avvenga con successo.
- `testLogout()` verifica che il modulo di autenticazione abbia il comportamento atteso nel momento in cui un utente autenticato richiede di uscire.

Risultato del test: non sono stati rilevati errori.

4.1.4 CredentialLoaderTest

Verifica la Classe: `CredentialLoarder`.
Descrizione: l'obiettivo di questo test è assicurare che il vettore in cui celle contiente tutte le credenziali d'accesso dell'utente (coppia *username-password*) venga popolato correttamente.
Tale test si compone di un solo metodo:

- `testLoarder()` tale metodo carica nel vettore i campi *username* e *password* e controlla che siano effettivamente stati inseriti e i dati siano coerenti (uguali) con quelli iniziali.

Risultato del test: non sono stati rilevati errori.

4.1.5 PrincipalImplTest

Verifica la Classe: `PrincipalImpl`.
Descrizione: l'obiettivo del test è verificare la corretta gestione dell'attributo `element` all'interno della classe `PrincipalImpl`. A tal fine si controlla la possibilità di recuperare l'elemento interno e la definizione del metodo che testa l'uguaglianza fra due istanze di questa classe.

Tale test si compone di tre metodi:

- `testGetName()` verifica che sia recuperato in maniera corretta l'elemento memorizzato all'interno del `PrincipalImpl`.
- `testEqual()` verifica che il test di uguaglianza fra istanze della classe dia i risultati attesi.
- `testToString()` verifica la corretta conversione in stringa delle istanze di questa classe.

Risultato del test: non sono stati rilevati errori.

4.2 Package `org.softwaresinthesis.mytalk.server.authentication.servlet`

4.2.1 LoginServletTest

Verifica l'oggetto: `LoginServlet`.
Descrizione: l'obiettivo di questo test è verificare la possibilità di effettuare il login da parte di un utente ricevuta tramite richiesta HTTP POST I possibili *input* riguardano dati passati (*username* e *password*) che possono essere corretti o non (campi dati vuoti, o che non corrispondono a nessun utente realmente registrato al sistema). Tale test si compone di tre metodi:

- `testLoginCorrectUser()` verifica l'effettivo *login* di un utente registrato nel sistema, tramite l'inserimento di dati corretti.
- `testLoginNotExistsUser()` verifica l'effettivo fallimento del *login* di un utente non registrato al sistema.
- `testLoginWrongUser()` verifica l'effettivo fallimento del *login* di un utente registrato al sistema, ma che non ha inserito dati corretti per il *login* stesso.

Risultato del test: non sono stati rilevati errori.

4.2.2 LogoutServletTest

Verifica l'oggetto: `LogoutServlet`.
Descrizione:l'obiettivo di questo test è verificare l'effettivo logout da parte di un utente autenticato tramite richiesta HTTP POST. Tale test si compone di un solo metodo:

- `testLogoutCorrectUser()` verifica l'effettivo *logout* da parte di un utente registrato nel sistema.

Risultato del test: non sono stati rilevati errori.

4.2.3 RegisterServletTest

Verifica l'oggetto: `RegisterServlet`.
Descrizione: l'obiettivo di questo test è la verifica che l'operazione di registrazione da parte di un utente ricevuta tramite richiesta HTTP POST, i possibili *input* riguardano dati passati per la registrazione stessa, che possono essere corretti o non (campi dati obbligatori vuoti, o errati). Tale test si compone di due metodi:

- `testRegisterCorrectUser()` verifica l'effettiva registrazione di un utente al sistema una volta inseriti dati corretti in *input*.
- `testRegisterWrongUser()` verifica l'effettivo fallimento nella registrazione di un utente al sistema una volta inseriti dati errati in *input*.

Risultato del test: non sono stati rilevati errori.

4.3 Package `org.softwaresinthesis.mytalk.server.abook`

4.3.1 AdressBookEntryTest

Verifica la Classe: `AdressBookEntry`.
Descrizione: l'obiettivo di questo test è verificare la corretta gestione di un'istanza di `AddressBookEntry`. Tale test si compone di due metodi:

- `testToJson()` tale metodo riceve una stringa in formato json di una *entry* della rubrica e verifica che quest'ultima sia equivalente a quella ottenuta dalla chiamata al metodo *ToJson* sull'istanza della classe *AddressBookEntry*;

- `testEquals()` viene creato un oggetto `AdressBookEntry` e viene confrontato con l'istanza dell'oggetto che ha invocato il metodo, restituendo *true* se e solo se sono uguali.

Risultato del test: non sono stati rilevati errori.

4.3.2 GroupTest

Verifica la Classe: `Group`.
Descrizione:l'obiettivo di questo test è verificare la corretta gestione di un istanza di `Group`. Tale test si compone di quattro metodi:

- `testToJson()` tale metodo riceve una stringa in formato json di una *entry* della rubrica e verifica che quest'ultima sia equivalente a quella ottenuta dalla chiamata al metodo `ToJson` sull'istanza della classe `Group`.
- `testId()` verifica l'effettivo inserimento del campo *id* mediante il metodo `getId()`.
- `testName()` verifica l'effettivo inserimento del campo *name* mediante il metodo `getName()`.
- `testEquals()` viene creato un oggetto `AddressBookEntry` e viene confrontato con l'istanza dell'oggetto che ha invocato il metodo, restituendo *true* se e solo se sono uguali.

Risultato del test: non sono stati rilevati errori.

4.3.3 UserDataTest

Verifica la Classe: `UserData`.
Descrizione: l'obiettivo di questo test è verificare la corretta gestione di un'istanza di `UserData`. La classe inizializza inoltre un istanza di tale classe, impostandone i campi membro attraverso i metodi *set*. Successivamente mediante i metodi *get* vengono verificati gli effettivi inserimenti.
Tale test si compone di otto metodi:

- `testToJson()` tale metodo riceve una stringa in formato JSON di una *entry* della rubrica e verifica che quest'ultima sia equivalente a quella ottenuta dalla chiamata al metodo *ToJson* sull'istanza della classe `UserData`.
- `testId()` verifica l'effettivo inserimento del campo *id* mediante il metodo `setId` e il suo recupero tramite `getId`;
- `testEmail()` verifica l'effettivo inserimento del campo *email* mediante il metodo `setMail` e il suo recupero tramite `getMail`;
- `testPassword()` verifica l'effettivo inserimento del campo *password* mediante il metodo `setPassword` e il suo recupero tramite `getPassword`;
- `testQuestion()` verifica l'effettivo inserimento del campo *question* mediante il metodo `setQuestion` e il suo recupero tramite `getQuestion`;
- `testAnswer()` verifica l'effettivo inserimento del campo *answer* mediante il metodo `setAnswer` e il suo recupero tramite `getAnswer`;
- `testName()` verifica l'effettivo inserimento del campo *name* mediante il metodo `setName` e il suo recupero tramite `getName`;
- `testSurname()` verifica l'effettivo inserimento del campo *surname* mediante il metodo `setSurname` e il suo recupero tramite `getSurname`;
- `testPicturePath()` verifica l'effettivo inserimento del campo *picturePath* mediante il metodo `setPath` e il suo recupero tramite `getPath`;

- `testEquals()` verranno creati due oggetti `UserData` contenenti gli stessi dati, tale metodo restituirà *true* se e solo se sono effettivamente uguali.

Risultato del test: non sono stati rilevati errori.

4.3.4 AdressBookEntryTest

Verifica la Classe: `AdressBookEntry`.
Descrizione: l'obiettivo del test è verificare la corretta gestione di un'istanza di `AddressBookEntry`. Tale test si compone di due metodi:

- `testToJson()` tale metodo riceve una stringa in formato JSON di una *entry* della rubrica e verifica che quest'ultima sia equivalente a quella ottenuta dalla chiamata al metodo `toJson` sull'istanza della classe `AddressBookEntry`;
- `testEquals()` viene creato un oggetto `AdressBookEntry` e viene confrontato con l'istanza dell'oggetto che ha invocato il metodo, restituendo *true* se e solo se sono uguali.

Risultato del test: non sono stati rilevati errori.

4.4 Package `org.softwaresinthesis.mytalk.server.abook.servlet`

4.4.1 AddressBookDoAddContactServletTest

Verifica l'oggetto: `AddressBookDoAddContactServlet`.
Descrizione: l'obiettivo di questo test è verificare la possibilità di aggiungere un contatto `IUserData` all'interno di una rubrica utente.

La possibilità di richiamare `AddressBookDoAddContactServlet` prende come precondizione l'autenticazione dell'utente richiedente, e quindi l'effettiva esistenza della sua rubrica, i possibili *input* riguardano dati passati mediante `XMLHttpRequest` che possono essere corretti (campi dati non vuoti e che si verifica appartengano ad un `IUserData` realmente esistente) o non (campi dati vuoti, o che non corrispondono a nessun utente realmente registrato al sistema). Tale test si compone di tre metodi:

- `testAddCorrectContact()` verifica l'effettiva aggiunta di un contatto nella rubrica del richiedente, a partire da dati corretti (realmente appartenenti all'utente aggiunto).
- `testAddNotExistsContact()` verifica la rilevazione dell'errore, scaturito dall'impossibilità di aggiungere un utente non registrato nel sistema alla rubrica del richiedente.
- `testAddWrongData()` verifica la rilevazione dell'errore, scaturito dall'impossibilità di aggiungere un utente `IUserData` definito da dati non completi (campi dati vuoti) parzialmente o nella loro totalità.

Risultato del test: non sono stati rilevati errori.

4.4.2 AddressBookDoRemoveContactServletTest

Verifica l'oggetto: `AddressBookDoRemoveContactServlet`.
Descrizione: l'obiettivo di questo test è verificare la possibilità di rimuovere un utente presente nella rubrica del richiedente. Come per il test della *Servlet* precedentemente descritta, anche in questo caso si assume come precondizione l'autenticazione dell'utente richiedente e quindi l'effettiva esistenza della sua rubrica.

I possibili *input* riguardano dati passati mediante `XMLHttpRequest`, che possono essere corretti (campi dati non vuoti e che si verifica appartengano ad un `IUserData` realmente esistente e presente nella rubrica dell'utente) o non (campi dati vuoti, o che non corrispondono a nessun utente realmente registrato al sistema, o comunque corrispongono ad un utente registrato, ma

il cui contatto non è presente nella rubrica del richiedente). Tale test si compone di quattro metodi:

- `testRemoveCorrectUser()` verifica l'effettiva eliminazione di un contatto presente nella rubrica del richiedente, a partire da dati corretti (realmente appartenenti all'utente da eliminare).
- `testRemoveNotExistContact()` verifica la rilevazione dell'errore, causato dall'impossibilità di rimuovere un utente non registrato nel sistema, dalla rubrica del richiedente.
- `testRemoveNotFriendContact()` verifica la rilevazione dell'errore, scaturito dall'impossibilità di rimuovere un contatto registrato al sistema, ma non presente nella rubrica del richiedente.
- `testRemoveWrongData()` verifica la rilevazione dell'errore, scaturito dall'impossibilità di rimuovere un utente `IUserData` definito da dati non completi (campi dati vuoti) parzialmente o nella loro totalità.

Risultato del test: non sono stati rilevati errori.

4.4.3 AddressBookDoCreateGroupServletTest

Verifica l'oggetto: *AddressBookDoCreateGroupServlet.*
Descrizione: l'obiettivo di questo test è verificare la possibilità di creare un gruppo `IGroup` all'interno di una rubrica utente. La possibilità di richiamare `AddressBookDoCreateGroupServlet` prende come precondizione l'autenticazione dell'utente richiedente, e quindi l'effettiva esistenza della sua rubrica. I possibili *input* riguardano dati passati mediante `XMLHttpRequest`, che possono essere corretti (campi dati non vuoti) o non (campi dati vuoti). Tale test si compone di due metodi:

- `testAddCorrectGroup()` verifica l'effettiva creazione di un gruppo nella rubrica del richiedente a partire da dati corretti (campi dati non vuoti).
- `testAddWrongData()` verifica la rilevazione dell'errore, scaturito dall'impossibilità di aggiungere un gruppo `IGroup` definito da dati non completi (campi dati vuoti) parzialmente o nella loro totalità.

Risultato del test: non sono stati rilevati errori.

4.4.4 AddressBookDoDeleteGroupServletTest

Verifica l'oggetto: *AddressBookDoDeleteGroupServlet.*
Descrizione: l'obiettivo di questo test è verificare la possibilità di eliminare un gruppo presente nella rubrica del richiedente. Come per il test della *Servlet* precedentemente descritta, si assume come precondizione l'autenticazione dell'utente richiedente e quindi l'effettiva esistenza della sua rubrica.

I possibili *input* riguardano dati passati mediante `XMLHttpRequest`, che possono essere corretti (campi dati non vuoti e che si verifica appartengano ad un `IGroup` realmente esistente e presente nella rubrica dell'utente) o non (campi dati vuoti, o che comunque non corrispondono a nessun gruppo realmente registrato al sistema, o comunque corrispongono ad un gruppo esistente, ma non presente nella rubrica del richiedente). Tale test si compone di quattro metodi:

- `testRemoveCorrectGroup()` verifica l'effettiva eliminazione di un gruppo presente nella rubrica del richiedente a partire da dati corretti (realmente appartenenti al gruppo da eliminare).

- `testRemoveNotExistGroup()` verifica la rilevazione dell'errore, scaturito dall'impossibilità di rimuovere un gruppo non presente nel sistema, dalla rubrica del richiedente.
- `testRemoveNotFriendContact()` verifica la rilevazione dell'errore, scaturito dall'impossibilità di rimuovere un gruppo presente nel sistema, ma non presente nella rubrica del richiedente.
- `testRemoveWrongData()` verifica la rilevazione dell'errore, scaturito dall'impossibilità di rimuovere un gruppo `IGroup` definito da dati non completi (campi dati vuoti) parzialmente o nella loro totalità.

Risultato del test: non sono stati rilevati errori.

4.4.5 AddressBookDoInsertInGroupServletTest

Verifica l'oggetto: `AddressBookDoInsertInGroupServlet`.
Descrizione: l'obiettivo di questo test è verificare la possibilità di aggiungere un contatto `IUserData` all'interno di un gruppo `IGroup` presente nella rubrica del richiedente. La possibilità di richiamare `AddressBookDoInsertInGroupServlet` prende come precondizione l'autenticazione dell'utente richiedente e quindi l'effettiva esistenza della sua rubrica.

I possibili *input* riguardano dati passati mediante `XMLHttpRequest` che possono essere corretti (campi dati non vuoti e che si verifica appartengano ad un `IUserData` realmente esistente e ad un `IGroup` realmente presente) o non (campi dati vuoti, o che comunque non corrispondono a nessun utente realmente registrato al sistema, o gruppo presente nella rubrica del richiedente). Tale test si compone di cinque metodi:

- `testAddCorrectContact()` verifica l'effettiva aggiunta di un contatto nella rubrica del richiedente (a gruppo precisato), a partire da dati corretti (realmente appartenenti all'utente aggiunto).
- `testAddNotExistsContact()` verifica la rilevazione dell'errore, scaturito dall'impossibilità di aggiungere un utente non registrato nel sistema, alla rubrica del richiedente (nel gruppo precisato).
- `testAddNotExistsGroup()` verifica la rilevazione dell'errore, scaturito dall'impossibilità di aggiungere un utente, in un gruppo che non è di proprietà del richiedente.
- `testAddWrongUserData()` verifica la rilevazione dell'errore, scaturito dall'impossibilità di aggiungere un utente `IUserData` definito da dati non completi (campi dati vuoti) parzialmente o nella loro totalità.
- `testAddWrongGroupData()` verifica la rilevazione dell'errore, scaturito dall'impossibilità di aggiungere un utente `IUserData` in un gruppo `IGroup` definito da dati non completi (campi dati vuoti) parzialmente o nella loro totalità.

Risultato del test: non sono stati rilevati errori.

4.4.6 AddressBookDoRemoveFromGroupServletTest

Verifica l'oggetto: `AddressBookDoRemoveFromGroupServlet`.
Descrizione: l'obiettivo di questo test è verificare la possibilità di rimuovere un conttatto da un gruppo presente nella rubrica del richiedente. Come per il test della *Servlet* precedentemente descritta, si assume come precondizione l'autenticazione dell'utente richiedente, e quindi l'effettiva esistenza della sua rubrica.

I possibili *input* riguardano dati passati mediante `XMLHttpRequest`, che possono essere corretti (campi dati non vuoti e che si verifica appartengano ad un `IGroup` realmente esistente e

presente nella rubrica dell'utente, e ad un contatto registrato nel sistema) o non (campi dati vuoti, che non corrispondono a nessun gruppo realmente presente nel sistema, che corrispongono ad un gruppo esistente ma non presente nella rubrica del richiedente, che non corrispondono a nessun contatto registrato nel sistema). Tale test si compone di quattro metodi:

- `testRemoveCorrectContact()` verifica l'effettiva eliminazione di un contatto presente (nel gruppo precisato) nella rubrica del richiedente, a partire da dati corretti (realmente appartenenti alla coppia contatto-gruppo).
- `testRemoveNotExistUser()` verifica la rilevazione dell'errore, scaturito dall'impossibilità di rimuovere un contatto non presente nel gruppo precisato.
- `testRemoveNotExistGroup()` verifica la rilevazione dell'errore, scaturito dall'impossibilità di rimuovere un utente da un gruppo non presente nella rubrica dell'utente.
- `testRemoveWrongData()` verifica la rilevazione dell'errore, scaturito dall'impossibilità di rimuovere un utente `IUserData` in un gruppo `IGroup` definito da dati non completi (campi dati vuoti) parzialmente o nella loro totalità.

Risultato del test: non sono stati rilevati errori.

4.4.7 AddressBookDoBlockServletTest

Verifica l'oggetto: `AddressBookDoBlockServlet`.
Descrizione: l'obiettivo di questo test è verificare la possibilità di bloccare un contatto all'interno di una rubrica utente. La possibilità di richiamare `AddressBookDoAddContactServlet` prende come precondizione l'autenticazione dell'utente richiedente e quindi l'effettiva esistenza della sua rubrica.

I possibili *input* riguardano dati passati mediante `XMLhttpRequest`, che possono essere corretti (campi dati non vuoti e che si verifica appartengano ad un `IUserData` realmente esistente) o non (campi dati vuoti, o che comunque non corrispondono a nessun utente realmente registrato al sistema).

- `testBlockCorrectContact()` verifica l'avvenuto blocco del contatto precisato, nella rubrica del richiedente, a partire da dati corretti (realmente appartenenti all'utente da bloccare).
- `testBlockNotExistsContact()` verifica la rilevazione dell'errore, scaturito dall'impossibilità di bloccare un utente non registrato nel sistema.
- `testWrongData()` verifica la rilevazione dell'errore, scaturito dall'impossibilità di bloccare un utente `IUserData` definito da dati non completi (campi dati vuoti) parzialmente o nella loro totalità.

Risultato del test: non sono stati rilevati errori.

4.4.8 AddressBookDoUnblockServletTest

Verifica l'oggetto: *AddressBookDoUnblockServlet*.
Descrizione: l'obiettivo di questo test è verificare la possibilità di sbloccare un contatto `IUserData` (bloccato) all'interno di una rubrica utente. La possibilità di richiamare `AddressBookDoUnblockServlet` prende come precondizione l'autenticazione dell'utente richiedente, e quindi l'effettiva esistenza della sua rubrica.

I possibili *input* riguardano dati passati mediante `XMLHttpRequest`, che possono essere corretti (campi dati non vuoti e che si verifica appartengano ad un `IUserData` realmente esistente, bloccato, e presente nella rubrica del richiedente) o non (campi dati vuoti, o che comunque non corrispondono a nessun utente realmente registrato al sistema).

Tale test si compone di tre metodi:

- `testUnblockCorrectContact()` verifica l'avvenuto sblocco del contatto precisato, nella rubrica del richiedente, a partire da dati corretti (realmente appartenenti all'utente da sbloccare).
- `testUnblockNotExistsContact()` verifica la rilevazione dell'errore, scaturito dall'impossibilità di sbloccare un utente non registrato nel sistema, o comunque non appartenente alla rubrica del richiedente.
- `testWrongData()` verifica la rilevazione dell'errore, scaturito dall'impossibilità di sbloccare un utente `IUserData` definito da dati non completi (campi dati vuoti) parzialmente o nella loro totalità.

Risultato del test: non sono stati rilevati errori.

4.4.9 AddressBookDoSearchServletTest

Verifica l'oggetto: `AddressBookDoSearchServlet`.
Descrizione: l'obiettivo di questo test è verificare la possibilità di ritornare una lista di contatti presenti nella rubrica del richiedente. La possibilità di richiamare `AddressBookGetContactsServlet` prende come precondizione l'autenticazione dell'utente richiedente, e quindi l'effettiva esistenza della sua rubrica.

I possibili input riguardano dati passati mediante `XMLHttpRequest`, che possono essere corretti (campi dati non vuoti) o non (campi dati vuoti). Tale test si compone di tre metodi:

- `testGetSearchContact()` verifica se gli utentei ottenuti in seguito all'operazione di ricerca sono tutti e soli gli utenti registrati nella rubrica del richiedente, e aventi o comee nome, cognome o mail, la parola ricercata.
- `testGetSearchNotExistsContact()` verifica che, dato un input corretto, e relativo esclusivamente ad utenti presenti nel sistema ma non nella rubrica del richiedente, il numero di utenti restituiti dalla ricerca sia 0.
- `testWrongData()` verifica che, dati in input valori scorretti, il numero di utenti restituiti dalla ricerca sia 0.

Risultato del test: non sono stati rilevati errori.

4.4.10 AddressBookGetContactsServletTest

Verifica l'oggetto: *AddressBookGetContactsServlet*.
Descrizione: l'obiettivo di questo test è verificare la possibilità di ritornare un contatto presente nella rubrica del richiedente. La possibilità di richiamare `AddressBookGetContactsServlet` prende come precondizione l'autenticazione dell'utente richiedente, e quindi l'effettiva esistenza della sua rubrica.

I possibili input riguardano dati passati mediante `XMLHttpRequest`, che possono essere corretti (campi dati non vuoti e che si verifica appartengano ad un `IUserData` realmente esistente e presente nella rubrica del richiedente) o non (campi dati vuoti, o che comunque non corrispondono a nessun utente realmente registrato al sistema). Tale test si compone di tre metodi:

- `testGetCorrectContact()` verifica se l'utente ottenuto come output del metodo doPost() della *servlet* è effettivamente il contatto a cui appartengono i dati corretti.
- `testGetNotExistsContact()` verifica la rilevazione dell'errore, scaturito dall'impossibilità di ottenere un utente non registrato nel sistema, o comunque non appartenente alla rubrica del richiedente.

- `testWrongData()` verifica la rilevazione dell'errore, scaturito dall'impossibilità di ottenere un utente `IUserData` definito da dati non completi (campi dati vuoti) parzialmente o nella loro totalità.

Risultato del test: non sono stati rilevati errori.

4.4.11 AddressBookGetGroupsServletTest

Verifica l'oggetto: `AddressBookGetGroupsServlet`.
Descrizione: l'obiettivo di questo test è verificare la possibilità di ritornare una lista di contatti `IUserData` presenti nella rubrica del richiedente, associati ai relativi gruppi di appartenenza.

La possibilità di richiamare `AddressBookGetGroupsServlet` prende come precondizione l'autenticazione dell'utente richiedente, e quindi l'effettiva esistenza della sua rubrica. Tale test si compone di un solo metodo:

- *testGetGroupContact()* verifica se gli utentei ottenuti in associazione ai gruppi di appertenneza, sono tutti e soli gli utenti-gruppi presenti nella rubrica del richiedente.

Risultato del test: non sono stati rilevati errori.

4.5 org.softwaresinthesis.mytalk.server.dao

4.5.1 HybernateUtilTest

Verifica la Classe: `HybernateUtil`.
Descrizione:l'obiettivo di questo test è assicurare la corretta esecuzione dei metodi della classe `HibernateUtil`.
Tale test si compone di due metodi:

- `testIstance()` l'obiettivo di questo test è assicurare l'effettiva presenza di un unica istanza della classe textitHibernateUtil mediante la creazione di due istanze e verificando che entrambi puntino alla stessa istanza dell'oggetto;
- `testGetFactory()` ha lo scopo di verificare la presenza di una `SessionFactory` configurata correttamente per la comunicazione con il database.

Risultato del test: non sono stati rilevati errori.

4.5.2 UserDataDAOTest

Verifica la Classe: `UserDataDAO`.
Descrizione:l'obiettivo di questo test è assicurare la corretta esecuzione delle operazioni CRUD (*Create, Read, Update, Delete*) per gli oggetti di tipo `UserData` verso il database.
Tale test si compone di cinque metodi:

- `testInsert()` tale metodo ha lo scopo di verificare il corretto inserimento di uno `UserData` all'interno del database;
- `testGetByEmail()` ha lo scopo di verificare il prelevamento dal database di uno `UserData` tramite l'indirizzo mail con cui si è registrato nel sistema;
- `testUpdate()` verifica se le modifiche ad un oggetto `UserData` siano effettive nel database;
- `testDelete()` verifica l'effettiva cancellazione di uno `UserData` dal database;
- `testSearchGeneric()` verifica che sia presente uno `UserData` nel database mediante la ricerca fatta cercando un parametro di *input* che ha una corrispondenza con uno o più campi tra *nome, cognome, email*.

Risultato del test: non sono stati rilevati errori.

4.5.3 AddressBookEntryDAOTest

Verifica la Classe: `AddressBookEntryDAO`.
Descrizione:l'obiettivo di questo test è assicurare la corretta esecuzione delle operazioni CUD (*Create, Update, Delete*) per gli oggetti di tipo `AddressBookEntry` nel database.
Tale test si compone di tre metodi:

- `testInsert()` tale metodo ha lo scopo di verificare il corretto inserimento di un `AddressBookEntry` all'interno del database;
- `testDelete()` tale metodo ha lo scopo di verificare la corretta rimozione di un `AddressBookEntry` dal database;
- `testUpdate()` tale metodo ha lo scopo di verificare il corretto aggiornamento di un `AddressBookEntry` nel database nel caso sia stato modificato uno dei campi dati.

Risultato del test: non sono stati rilevati errori.

4.5.4 GroupDAOTest

Verifica la Classe: `GroupDAO`.
Descrizione:l'obiettivo di questo test è assicurare la corretta esecuzione delle operazioni CRUD (*Create,Read, Update, Delete*) per gli oggetti di tipo `Group` e di ricerca di tali oggetti nel database.
Tale test si compone di sei metodi:

- `testInsert()` tale metodo ha lo scopo di verificare il corretto inserimento di un `Group` all'interno del database;
- `testDelete()` tale metodo ha lo scopo di verificare la corretta rimozione di un `Group` dal database;
- `testUpdate()` tale metodo ha lo scopo di verificare il corretto aggiornamento di un `Group` nel database nel caso sia stato modificato uno dei campi dati.
- `testGetByID()` tale metodo ha lo scopo di verificare la corretta ricerca di un `Group` nel database tramite l'inserimento di un ID valido.
- `testGetByOwner()` tale metodo ha lo scopo di verificare la corretta ricerca di un `Group` nel database tramite il passaggio di un `Long` che corrisponde all'ID del proprietario del gruppo.
- `testGetByOwnerAndName()` tale metodo ha lo scopo di verificare la corretta ricerca di un `Group` nel database tramite il passaggio di un `Long` che corrisponde all'ID del proprietario del gruppo che di una stringa che corrisponde al nome del gruppo in formato `String`.

Risultato del test: non sono stati rilevati errori.

5 Presenter

I test sul presenter mirano a verificare che le classi di tale componente rispettino il comportamento atteso, recuperando i dati tramite l'interazione con il *model*, componente di cui i test sono riportati nella sezione precedente.

Questa serie di test, come anticipato nelle prime pagine del documento, è stata eseguita mediante il *framework Qunit*. Tali verifiche sono state implementate allo scopo di non testare soltanto il comportamento della specifica funzione, ma anche verificare lo stato di avanzamento del prodotto e il numero di bug evidenziati ancora esistenti. Ovviamente al termine di tali verifiche il risultato dovrà risultare positivo, ovvero i test dovranno essere *tutti* superati.

5.1 Package `org.softwaresynthesis.mytalk.clientpreenter.guicontrol`

5.1.1 AccountSettingsPanelPresenterTest

Verifica l'oggetto: `AccountSettingsPanelPresenter`.
Descrizione: verifica la corretta inizializzazione di *AccountSettingPanel*, ovvero che l'albero sia stato costruito correttamente, e che il contenuto dei vari nodi sia stato inserito coerentemente. Verifica inoltre la corretta inizializzazione del *form* per la modifica dei dati personali e l'invio dei dati in forma corretta al server. Tale verifica è composta da quattro test:

- `testCreatePanel()` crea l'elemento *AccountSettingPanel*, estrae la lista dei suoi figli, ne controlla il numero (deve essere *tre*) e che il primo sia un immagine, il secondo una lista e il terzo un *button*. Successivamente controlla che gli elementi della lista vengano visualizzati correttamente presentando quindi il contenuto corrispondente ai dati memorizzati nello *stub* di `CommunicationCenter`.
- `testBuildQueryString()` controlla che venga creata la stringa di interrogazione da inviare alla servlet sulla base dei dati memorizzati nell'array associativo passato in ingresso come parametro al metodo `buildQueryString` del presenter.
- `testHasSomethingChanged()` verifica che sia rilevato correttamente un cambiamento (nel nome, nel cognome o nel percorso dell'immagine) tra l'array associativo passato in ingresso e quanto memorizzato in uno *stub* di `CommunicationCenter`.
- `testOnChangeButtonPressed()` controlla il comportamento del presenter nel momento in cui si verifica l'evento nell'interfaccia grafica di pressione del pulsante per la modifica dei dati. In particolare, il test verifica che il pannello sia trasformato in un *form* con i tre campi di inserimento necessari corredati dalle relative etichette.

Risultato del test: non sono stati rilevati errori.

5.1.2 AddressBookPanelPresenterTest

Verifica l'oggetto: `AddressBookPanelPresenter`.
Descrizione: l'obiettivo di questo test è la verifica della correttezza di tutte le funzionalità offerte da *AddressBookPanel*.
Tale verifica è composta da sedici test:

- `testDeleteGroup()` il test controlla che avvenga la corretta eliminazione di un gruppo dalla rubrica, tramite l'uso di uno *stub*. Il test verifica anche l'effettivo sollevamento di un'eccezione quando si tenta di eliminare un gruppo non presente nella rubrica.
- `testInitialize()` controlla che la rubrica venga costruita correttamente. Verifica in particolare che la rubrica sia composta di cinque figli dove il primo è un *<h1>* e gli altri quattro sono * *. Controlla poi che ognuno dei cinque figli sia costruito correttamente.
- `testSetup()` controlla il corretto funzionamento di *setup()* ovvero che avvenga correttamente il popolamento della rubrica con i contatti, che vengono scaricati da uno *stub* che simula la presenza di una *servlet* lato server. Il test avviene tramite la creazione di uno *stub* che emula i contatti dell'utente presenti nella sua rubrica. Il test consiste nel controllo che avvenga l'inserimento dei contatti nella lista contenuta nella struttura dell'*AdressBookPanel*. In particolare, oltre a controllare che il numero di contatti sia giusto (devono essere due), si verifica che sia corretto il loro nome, il loro cognome.
- `testHide()` verifica il comportamento del presenter nel momento in cui il pannello deve essere nascosto, vale a dire impostando correttamente le proprietà di visualizzazione dell'elemento grafico associato.

- `testAddContact()` il test verifica che avvenga il corretto inserimento di un contatto nella rubrica, tramite l'uso di uno *stub*. Il test verifica anche l'effettivo sollevamento di un'eccezione quando si tenta di inserire un contatto già presente nella rubrica.
- `testRemoveContact()` il test verifica che avvenga la corretta eliminazione di un contatto, tramite l'uso di *stub*. Il test verifica anche l'effettivo sollevamento di un'eccezione quando si tenta di eliminare un contatto non presente nella rubrica.
- `testApplyFilterByString()` verifica il funzionamento del filtro per i contatti. Il test controlla che data in *input* una stringa vengano restituiti i contatti che contengono la stringa nel nome, nel cognome o nella mail. Il test controlla il funzionamento per ogni possibile parametro (nome, cognome, mail). Viene inoltre verificato che il risultato del filtraggio sia nullo qualora la stringa inserita non sia presente in nessuno dei campi di nessun contatto della rubrica.
- `testApplyFilterByString()` verifica il funzionamento del filtro per i contatti. Il test controlla che data in *input* una stringa vengano restituiti i contatti che contengono la stringa nel nome, nel cognome o nella mail. Il test controlla il funzionamento per ogni possibile parametro (nome, cognome, mail). Viene inoltre verificato che il risultato del filtraggio sia nullo qualora la stringa inserita non sia presente in nessuno dei campi di nessun contatto della rubrica.
- `testAddGroup()` il test verifica che avvenga il corretto inserimento di un gruppo nella rubrica, tramite l'uso di uno *stub*. Il test verifica anche l'effettivo sollevamento di un'eccezione quando si tenta di inserire un gruppo già presente nella rubrica.
- `testBlockUser()` il test controlla che il controlla il corretto funzionamento del blocco di un contatto tramite l'uso di uno *stub* per la *servlet* coivolta nell'operazione. Viene inoltre controllato l'effettivo sollevamento di un'eccezione nel caso in cui l'utente che si desidera bloccare sia già bloccato.
- `testUnlockUser()` il test controlla che il controlla il corretto funzionamento dello sblocco di un contatto viene effettuato il blocco di un contatto tramite l'uso di uno *stub*. Il test controlla anche l'effettivo sollevamento di un'eccezione nel caso in cui l'utente che si desidera sbloccare risulta essere già sbloccato.
- `testApplyFilterByGroup()` verifica il funzionamento del filtro per i gruppi. Il test controlla che data in *input* il nome del gruppo vengano restituiti i contatti appartenenti a quel gruppo.
- `testAddContactInGroup()` il test verifica che avvenga il corretto inserimento di un contatto all'interno gruppo, tramite l'uso di uno *stub*. Il test controlla anche l'effettivo sollevamento di un'eccezione quando si tenta di inserire un contatto già presente nel gruppo in cui si tenta di aggiungerlo.
- `testDeleteContactFromGroup()` il test verifica che avvenga la corretta eliminazione di un contatto da un gruppo, tramite l'uso di uno *stub*. Il test controlla anche l'effettivo sollevamento di un'eccezione quando si tenta di eliminare dal gruppo un contatto che non gli appartiene.
- `testGetGroupsWhereContactsIs()` verifica che il nome del gruppo in cui l'utente *stub* è inserito sia corrispondente con quello in cui è effettivamente presente.
- `showFiltered()` il test controlla la corretta restituzione dei contatti filtrati. In particolare verifica che il numero di contatti restituiti sia corretto.

- `testContactAlreadyPresent()` verifica la presenza di un utente nella lista dei contatti presenti nella rubrica dell'utente garantendone la presenza effettiva.
- `testRemoveContactFromGroup()` il test verifica che avvenga la corretta eliminazione di un contatto da un gruppo, tramite l'uso di uno *stub*. Il test controlla anche l'effettivo sollevamento di un'eccezione quando si tenta di eliminare dal gruppo un contatto che non gli appartiene.

Risultato del test: non sono stati rilevati errori.

5.1.3 CallHistoryPanelPresenterTest

Verifica l'oggetto: `CallHistoryPanelPresenter`.
Descrizione: l'obiettivo di questo test è verificare la corretta creazione del sotto-albero che ha radice nell'elemento *CallHistoryPanel*.
Tale verifica è composta da un solo test:

- `testCreatePanel()` controlla che CallHistoryPanel abbia un unico figlio e che quest'ultimo sia effettivamente una lista (che corrisponde allo storico delle chiamate).

Risultato del test: non sono stati rilevati errori.

5.1.4 CommunicationPanelPresenterTest

Verifica l'oggetto: `CommunicationPanelPresenter`.
Descrizione: l'obiettivo di questo test è verificare la corretta creazione del sotto-albero che ha radice nell'elemento *CommunicationPanel*.
Tale verifica è composta da cinque test:

- `testCreatePanel()` controlla che il numero dei figli del *CommunicationPanel* siano esattamente due, che entrambi siano `<div>` e che il valore dell'attributo `id` del primo figlio sia "divCall" mentre del secondo "divChat". Verifica i figli dell'elemento per la visualizzazione delle chiamate (due elementi `<video>` e un ulteriore elemento `<div>` per la visualizzazione delle statistiche) e i figli dell'elemento per la visualizzazione delle chat.
- `testUpdateTimer()` verifica che sia correttamente visualizzata la stringa passata come parametro in ingresso al metodo `updateTimer`. A tal fine è utilizzato uno *stub* per simulare per la vista, che espone verso il presenter la medesima interfaccia del pannello vero.
- `testGetMyVideo()` verifica l'accesso alla parte di interfaccia grafica incaricata di visualizzare l'*input* proveniente dalla webcam dell'utente, utilizzando ancora uno *stub* per la vista mancante.
- `testGetOtherVideo()` verifica l'accesso alla parte di interfaccia grafica incaricata di visualizzare l'*input* proveniente dal client con cui si è instaurata una comunicazione.
- `testUpdateStarts()` verifica la corretta visualizzazione delle statistiche di comunicazione, tanto per i dati in ingresso che per i dati in uscita. A tal fine è utilizzato uno *stub* per simulare la presenza della vista durante l'esecuzione del test.

Risultato del test: non sono stati rilevati errori.

5.1.5 ContactPanelPresenterTest

Verifica l'oggetto: `ContactPanelPresenter`.
Descrizione: l'obiettivo di questo test è verificare la corretta creazione del sotto-albero che ha radice nell'elemento *ContactPanel*.
Tale verifica è composta da due test:

- `testCreatePanel()` crea l'elemento *ContactPanel*, estrae la lista dei suoi figli e controlla per ogni figlio che le informazioni siano coerenti con quelle inserite in *input*.
- `testDisplay()` controlla la modifica dell'interfaccia grafica nel momento in cui deve essere visualizzato il *ContactPanel* contenente i dati di un contatto selezionato. In particolare, il test controlla la visualizzazione dei dati (nome, cognome, indirizzo email) e la corretta impostazione dell'attributo `src` dell'elemento immagine presente nel pannello.

Risultato del test: non sono stati rilevati errori.

5.1.6 GroupPanelPresenterTest

Verifica l'oggetto: `GroupPanelPresenter`.
Descrizione: l'obiettivo di questo test è verificare la corretta istanziazione del *GroupPanel*, verificandone pertanto la corretta esecuzione della *form*.
Tale verifica è composta da due test:

- `testCreatePanel()` verifica la corretta visualizzazione iniziale del pannello, in particolare, verificando che sia creata la lista dei gruppi presenti nella rubrica dell'utente associato al client.
- `testDisplayContactList()` verifica la corretta visualizzazione dell'elenco dei gruppi della rubrica nel pannello dell'interfaccia utente, inserendo per ogni gruppo una voce che ne riporta il nome e che è corredata dai pulsanti di amministrazione (per l'eliminazione di un gruppo).

Risultato del test: non sono stati rilevati errori.

5.1.7 LoginPanelPresenterTest

Verifica l'oggetto: `LoginPanelPresenter`.
Descrizione: l'obiettivo di questo test è verificare la corretta istanziazione del *LoginPanel* popolato con dei dati coerenti rispetto allo scopo della *form*. Tale verifica è composta da sette test:

- `testInitialize()` mediante diverse asserzioni `equal` controlla l'effettiva correttezza della *form* creata, sia nella forma che nel funzionamento.
- `testLogin()` effettua una prova di login, mediante l'inserimento di uno *username* e una *password* corretti, verificando che tali dati siano accettati correttamente.
- `testGetUsername()` verifica che, dato uno *username* in *input* corretto generato come *stub*, sia possibile recuperarlo correttamente. Se lo *username* non è una mail valida, o non è inserito nella *form*, il test verifica che venga sollevata un'eccezione.
- `testGetPassword()` verifica che, data una *password* in *input* corretta generata come *stub*, sia possibile recuperarla correttamente.Se la *password* non è inserita nella *form*, il test verifica che venga sollevata un'eccezione.
- `testHide()` controlla che venga correttamente nascosto il *form* di *login* dopo l'effettiva autenticazione o di richiesta di registrazione.
- `testBuildRetrivePasswordForm()` tale test verifica la corretta creazione della form richiamata per il recupero della password tramite risposta segreta, successivamente controlla che venga richiamata la domanda correttamente impostata a cui rispondere.

- `testHasAnsweredCorrectly()` tale test verifica la corretta ricezione della risposta segreta (e la correttezza della risposta stessa) associata per il recupero password. Viene inoltre effettuato un test di negazione della stessa.

Risultato del test: non sono stati rilevati errori.

5.1.8 MainPanelPresenterTest

Verifica l'oggetto: `MainPanelPresenter`.
Descrizione:l'obiettivo di questo test è verificare la corretta creazione del sotto-albero che ha radice nell'elemento *MainPanel* e che il contenuto dei vari nodi sia stato inserito coerentemente, infine deve consentire la corretta visualizzazione dei sotto-pannelli.

Tale verifica è composta da tre test:

- `testInitialize()` mediante diversi metodi *equal* controlla l'effettiva correttezza della *form* creata, sia nella forma che nel funzionamento.
- `testDisplayChildPanel()` verifica la corretta visualizzazione di un elemento interno al pannello principale.
- `testHide()` controlla che venga correttamente nascosto il *form* principale prima dell'autenticazione.

5.1.9 MessagePanelPresenterTest

Verifica l'oggetto: `MessagePanelPresenter`.
Descrizione: l'obiettivo di questo test è verificare la corretta creazione del sotto-albero che ha radice nell'elemento *MessagePanel* e che il contenuto dei vari nodi sia stato inserito coerentemente.
Tale verifica è composta da un solo test:

- `testCreatePanel()` crea l'elemento *MessagePanel* e estrae la lista dei figli del pannello verificando che contenga due figli:il primo dev'essere di tipo *video* mentre il secondo di tipo *div*. Successivamente estraggo la lista dei figli del primo figlio (il suo figlio deve essere *source*) e del secondo figlio (il cui figlio deve essere *ul*).

Risultato del test: non sono stati rilevati errori.

5.1.10 RegisterPanelPresenterTest

Verifica l'oggetto: `RegisterPanelPresenter`.
Descrizione:l'obiettivo di questo test è verificare la corretta creazione del sotto-albero che ha radice nell'elemento *RegisterPanel*, la possibilità di nascondere correttamente l'elemento *RegisterPanel*, il corretto recupero dei dati inseriti nella *form* d'iscrizione e infine i test relativi al recupero delle singole informazioni passate in fase di registrazione.
Tale verifica è composta da dieci test:

- `testInitialize()` mediante diversi metodi `equal` controlla l'effettiva correttezza della *form* creata, sia nella forma che nel funzionamento.
- `testHide()` controlla che venga correttamente nascosto il *form* di registrazione.
- `testRegister()` mediante diversi metodi `equal` controlla l'effettiva registrazione dei dati inseriti nella *form*, confrontandoli con quelli passati in ingresso.

- `testGetPicturePath()` crea una *form* compilata correttamente con l'immagine utente da inserire, successivamente prova a recuperarla mediante `getPicturePath`, verificando se è stata inserita correttamente (o meno).
- `testGetUsername()` crea una *form* compilata correttamente con lo *username*, e prova a recuperarlo mediante `getUsername`, verificando se è stato inserito correttamente (o meno).
- `testGetPassword()` crea una *form* compilata correttamente con il campo *password*, e prova a recuperarlo mediante `getPassword`, verificando se è stata inserita correttamente (o meno). Il test verifica anche il sollevamento di un'eccezione nel momento in cui non sia stata inserito alcun valore in questo campo.
- `testGetQuestion()` crea una *form* compilata correttamente con il campo *question*, e prova a recuperarlo mediante `getQuestion`, verificando se è stata inserita correttamente (o meno). Dal momento che si tratta di un dato obbligatorio, il test verifica anche che sia sollevata un'eccezione nel caso in cui non sia fornito alcun valore.
- `testGetAnswer()` crea una *form* compilata correttamente con il campo *answer*, e prova a recuperarlo mediante `getAnswer`, verificando se è stata inserita correttamente (o meno). Dal momento che si tratta di un dato obbligatorio, il test verifica anche che sia sollevata un'eccezione nel caso in cui questo valore non sia fornito.
- `testGetName()` crea una *form* compilata correttamente con il campo *name*, e prova a recuperarlo mediante `getName`, verificando se è stato inserito correttamente (o meno).
- `testGetSurname()` crea una *form* compilata correttamente con il campo *surname*, e prova a recuperarlo mediante `getSurname`, verificando se è stata inserita correttamente (o meno).

Risultato del test: non sono stati rilevati errori.

5.1.11 SearchResultPanelPresenterTest

Verifica l'oggetto: `SearchResultPanelPresenter`.
Descrizione: l'obiettivo di questo test è verificare che sia correttamente creato l'elemento grafico *SearchResultPanel* e il comportamento del presenter nel momento in cui tale pannello deve essere aggiornato. Tale verifica è composta da due test:

- `testCreatePanel()` verifica che il pannello sia costruito e visualizzato in accordo con le specifiche della progettazione e, in particolare, che sia presente la lista dei contatti che corrispondono ai risultati della ricerca.
- `testDisplayContactList()` verifica che sia visualizzata la lista dei contatti passata come parametro in ingresso al metodo `displayContactList` del presenter.

5.1.12 ToolsPanelPresenterTest

Verifica l'oggetto: `ToolsPanelPresenter`.
Descrizione: l'obiettivo di tale test è verificare che sia correttamente creato il pannello degli strumenti dell'interfaccia grafica, nonché il comportamento del presenter nel momento in cui dalla vista è invocato il metodo che corrisponde all'uscita dal sistema. Tale verifica è composta da tre test:

- `testInitialize()` mediante diversi metodi *equal* controlla l'effettiva correttezza della *form* creata, sia nella forma che nel funzionamento.
- `testHide()` controlla che venga correttamente nascosto il *form* degli strumenti.
- `testLogout()` controlla l'effettivo *logout* dell'utente autenticato al sistema.

6 Test Coverage

Tale test è stato effettuato mediante il *plugin* per Eclipse *eclemma* per la parte Java (quindi nel *Model*) e con *JSCoverage* per la parte *JavaScript* (per il presenter) definendo quindi le percentuali di codice che è stato sottoposto a test per i singoli componenti dell'applicazione.

Componente	Copertura
Model	56,26%
Presenter	67,74%

La percentuale di copertura rilevata è relativamente bassa nella parte *Model*, tale risultato si giustifica dal numero elevato di metodi `getter` e `setter` che non sono stati tutti analizzati tramite test di unità (ma prelevati e verificati a campione), ma solo tramite analisi statica del codice. Per la parte presenter al contrario si può ricondurre il risultato alle numerose funzioni anonime impossibili da testare e invocate soltanto durante l'effettivo funzionamento dell'applicazione.

7 Test di Sistema

Nell'attività di validazione che verrà affrontata nella successiva (e ultima) fase del progetto, saranno effettuati i test per accertare le effettive funzionalità offerte dal prodotto MyTalk.
Tali verifiche sono state definite e pianificate già durante l'attività di analisi dei requisiti e sono reperibili nel documento relativo allegato (*analisi_dei_requisiti.3.0.pdf*) in cui sono elencati in una pratica tabella che ne descrive (per ognuno) le modalità di verifica e i requisiti ad essi associati. In questo modo verrà assicurato il tracciamento tra requisiti e test, pertanto tutti i requisiti soddisfatti saranno verificati mediante un test specifico.

8 Test di Utilizzo

Al fine di verificare che l'interfaccia del prodotto sia più intuitiva e *user-friendly* possibile, sono stati predisposte delle simulazioni da proporre ad un campione di utenza selezionata (cinque persone con conoscenze informatiche medio-basse). Essendo test non oggettivi non verranno catalogati come vere e proprie verifiche, ma saranno semplici *feedback* su come migliorare o confermare la parte visiva di MyTalkin base ai comportamenti assunti dai *tester* durante la prova.