



responsabile@hourglabs.org

Esito delle verifiche

Versione 2.0

Informazioni documento

Nome	<i>Esito_dei_Test_2.0.pdf</i>
Versione	2.0
Data creazione	2013-07-01
Data ultima modifica	2013-07-10
Stato del Documento	Formale ad uso esterno
Redazione	Riccardo Cesarotto Giovanni Morlin
Verifica	Thomas Rossetto
Approvazione	Gioele Lorenzo Cresce
Distribuzione	hourglass Prof. Tullio Vardanega Prof. Riccardo Cardin

Registro delle modifiche

Data	Versione	Ruolo	Descrizione	Autore
2013-09-19	2.0	Responsabile	Approvazione del documento	Thomas Rossetto
2013-09-18	1.1	Verificatore	Integrazione a seguito della RQ, e aggiornamento della sezione test di unità	Paolo Bustreo
2013-07-14	1.0	Responsabile	Approvazione Documento	Gioele Lorenzo Cresce
2013-07-13	0.5	Verificatore	Verifica Documento	Thomas Rossetto
2013-07-12	0.4	Progettista	Aggiunta sezione test di integrazione	Riccardo Cesarotto
2013-07-12	0.3	Progettista	Aggiunta sezione copertura test unità	Riccardo Cesarotto
2013-07-12	0.2	Progettista	Aggiunta sezione componenti lato Client	Riccardo Cesarotto
2013-07-2	0.1	Progettista	Aggiunta sezione componenti lato Server	Giovanni morlin

Indice

1	Scopo del documento	4
1.1	Riferimenti	4
1.1.1	Normativi	4
1.1.2	Informativi	4
2	Esito dell'Analisi statica del codice	5
3	Esiti Test di unità	11
3.1	Ambiente di lavoro	11
3.2	Componenti lato server	11
3.2.1	Test di componenti nel package it.hourglass.myTalk.server.wssserver . .	11
3.2.2	Test di componenti nel package it.hourglass.myTalk.server.model . . .	14
3.3	Componenti lato client	17
3.3.1	Test di componenti nel package it.hourglass.myTalk.client.shared . . .	17
3.3.2	Test di componenti nel package it.hourglass.myTalk.client.presenter . .	17
4	Copertura Test di unità	23
5	Esiti Test di integrazione	25

1 Scopo del documento

Lo scopo di questo documento è di presentare l'esito dei test aggiornato al 2013-09-18: saranno forniti riassunti tabellari riguardanti l'esito dei test con descrizioni dettagliate del loro scopo e di eventuali errori riscontrati, inoltre vengono allegati gli esiti dell'analisi statica del codice e informazioni sulla sua copertura.

1.1 Riferimenti

1.1.1 Normativi

- Le Norme di Progetto, disponibili nel documento *Norme di Progetto_v4.0.pdf*.

1.1.2 Informativi

- Il Piano di Qualifica, disponibili nel documento *Piano di Qualifica_v4.0.pdf*.

2 Esito dell'Analisi statica del codice

Nelle tabelle seguenti vi è una sintesi riguardante le misure statiche sul codice effettuate sulle metriche definite in sezione 5 del *Piano di Qualifica* e calcolate utilizzando *Metrics*.

Metrica	Totale	Media	Massimo
Complessità ciclomatica	-	1,354	12
Numero di parametri	-	0,659	16
Numero di campi dati per classe	439	4,526	25
Numero livelli di annidamento	-	1,204	5
Indice di utilità	-	10,733	50
Indice di dipendenza	-	9,467	32

Tabella 2: Sintesi metriche GWT dell'applicativo generale

Metrica	Totale	Media	Massimo
Complessità ciclomatica	0	1	1
Numero di parametri	-	0	0
Numero di campi dati per classe	0	0	0
Numero livelli di annidamento	0	1	1
Indice di utilità	0	-	-
Indice di dipendenza	1	-	-

Tabella 3: Sintesi metriche GWT per il package `it.hourglass.myTalk.client`

Metrica	Totale	Media	Massimo
Complessità ciclomatica	-	1,696	12
Numero di parametri	-	0,261	2
Numero di campi dati per classe	3	3	3
Numero livelli di annidamento	-	1,391	4
Indice di utilità	1	-	-
Indice di dipendenza	1	-	-

Tabella 4: Sintesi metriche GWT per il package `it.hourglass.myTalk.client.appcontroller`

Metrica	Totale	Media	Massimo
Complessità ciclomatica	-	1,845	11
Numero di parametri	-	1,517	11
Numero di campi dati per classe	28	9,333	19
Numero livelli di annidamento	-	1,379	4
Indice di utilità	8	-	-
Indice di dipendenza	3	-	-

Tabella 5: Sintesi metriche GWT per il package `it.hourglass.myTalk.client.communication`

Metrica	Totale	Media	Massimo
Complessità ciclomatica	-	1	1
Numero di parametri	-	0,559	2
Numero di campi dati per classe	3	0,188	2
Numero livelli di annidamento	-	1	1
Indice di utilità	14	-	-
Indice di dipendenza	32	-	-

Tabella 6: Sintesi metriche GWT per il package `it.hourglass.myTalk.client.event`

Metrica	Totale	Media	Massimo
Complessità ciclomatica	-	0	-
Numero di parametri	-	0	-
Numero di campi dati per classe	0	0	-
Numero livelli di annidamento	-	0	-
Indice di utilità	17	-	-
Indice di dipendenza	4	-	-

Tabella 7: Sintesi metriche GWT per il package `it.hourglass.myTalk.client.rpcservice`

Metrica	Totale	Media	Massimo
Complessità ciclomatica	-	1,496	7
Numero di parametri	-	0,65	4
Numero di campi dati per classe	87	4,143	15
Numero livelli di annidamento	-	1,483	3
Indice di utilità	7	-	-
Indice di dipendenza	22	-	-

Tabella 8: Sintesi metriche GWT per il package `it.hourglass.myTalk.client.presenter`

Metrica	Totale	Media	Massimo
Complessità ciclomatica	-	0	-
Numero di parametri	-	0	-
Numero di campi dati per classe	0	0	-
Numero livelli di annidamento	-	0	-
Indice di utilità	50	-	-
Indice di dipendenza	25	-	-

Tabella 9: Sintesi metriche GWT per il package `it.hourglass.myTalk.client.presenter.display`

Metrica	Totale	Media	Massimo
Complessità ciclomatica	-	1,632	7
Numero di parametri	-	0,579	4
Numero di campi dati per classe	35	7	16
Numero livelli di annidamento	-	1,684	4
Indice di utilità	1	-	-
Indice di dipendenza	5	-	-

Tabella 10: Sintesi metriche GWT per il package `it.hourglass.myTalk.client.presenter.profilemanagment`

Metrica	Totale	Media	Massimo
Complessità ciclomatica	-	1,079	6
Numero di parametri	-	0,285	2
Numero di campi dati per classe	155	6,739	25
Numero livelli di annidamento	-	1,046	3
Indice di utilità	16	-	-
Indice di dipendenza	23	-	-

Tabella 11: Sintesi metriche GWT per il package `it.hourglass.myTalk.client.view`

Metrica	Totale	Media	Massimo
Complessità ciclomatica	-	1,167	2
Numero di parametri	-	0,5	1
Numero di campi dati per classe	20	6,667	13
Numero livelli di annidamento	-	1,167	2
Indice di utilità	1	-	-
Indice di dipendenza	3	-	-

Tabella 12: Sintesi metriche GWT per il package `it.hourglass.myTalk.client.view.contactprofile`

Metrica	Totale	Media	Massimo
Complessità ciclomatica	-	1,22	11
Numero di parametri	-	0,06	1
Numero di campi dati per classe	56	11,2	24
Numero livelli di annidamento	-	1,04	2
Indice di utilità	3	-	-
Indice di dipendenza	5	-	-

Tabella 13: Sintesi metriche GWT per il package `it.hourglass.myTalk.client.view.profilemanagment`

Metrica	Totale	Media	Massimo
Complessità ciclomatica	-	1,011	2
Numero di parametri	-	0,912	16
Numero di campi dati per classe	31	6,2	18
Numero livelli di annidamento	-	1,022	2
Indice di utilità	35	-	-
Indice di dipendenza	1	-	-

Tabella 14: Sintesi metriche GWT per il package `it.hourglass.myTalk.client.shared`

Metrica	Totale	Media	Massimo
Complessità ciclomatica	0	2,056	4
Numero di parametri	0	1	2
Numero di campi dati per classe	0	0	0
Numero livelli di annidamento	0	1,833	3
Indice di utilità	1	-	-
Indice di dipendenza	3	-	-

Tabella 15: Sintesi metriche GWT per il package `it.hourglass.myTalk.server.model.dao`

Metrica	Totale	Media	Massimo
Complessità ciclomatica	0	2,64	10
Numero di parametri	-	1,44	3
Numero di campi dati per classe	12	3	7
Numero livelli di annidamento	-	1,96	5
Indice di utilità	1	-	-
Indice di dipendenza	4	-	-

Tabella 16: Sintesi metriche GWT per il package `it.hourglass.myTalk.server.model.businesslogic`

Metrica	Totale	Media	Massimo
Complessità ciclomatica	-	1,057	2
Numero di parametri	-	0,8	3
Numero di campi dati per classe	9	1,125	6
Numero livelli di annidamento	-	0,286	2
Indice di utilità	6	-	-
Indice di dipendenza	10	-	-

Tabella 17: Sintesi metriche GWT per il package `it.hourglass.myTalk.client.wrappers`

Dalla tabella 2 si può notare che solamente la misura del numero di livelli di annidamento non oltrepassa il limite imposto dalle metriche descritte nel paragrafo 5.1 del documento *Piano di Qualifica v2.4.pdf*. Le misure invece che superano il limite imposto sono la complessità ciclomatica (12 invece di 10), il numero di parametri (16 invece di 8) e il numero di campi dati per classe (25 invece di 12). La misura del numero di parametri sfiora il limite imposto a causa dei package `it.hourglass.myTalk.client.shared` e `it.hourglass.myTalk.client.communication`. Il primo package infatti contiene la classe `User` con un gran numero di campi dati e con costruttori che richiedono un elevato numero di parametri mentre il secondo contiene le classi `Call` e `WSConnection` che sfiorano questa misura a causa dello stesso motivo. Quindi se escludiamo questi due package da tale misura otteniamo un valore massimo di 4 (vedi tabella 8). La misura del numero di campi dati invece oltrepassa il limite imposto a causa delle seguenti classi:

- `Call`, facente parte del package `it.hourglass.myTalk.client.communication`, poichè ha bisogno di un elevato numero di campi dati per salvare le statistiche di chiamata e per il corretto funzionamento di quest'ultima;
- `CallView` e `RegistrationView`, facenti parte del package `it.hourglass.myTalk.client.view`, poichè hanno bisogno di un gran numero di elementi grafici per racchiudere rispettivamente i dati della chiamata e dell'utente;
- `ExtendedDataContactProfileView`, facente parte del package `it.hourglass.myTalk.client.view.contactprofile`, poichè ha bisogno di un gran numero di elementi grafici per racchiudere i dati personali estesi dell'utente;
- `PersonalDataProfileView` e `ExtendedDataProfileManagementView`, facenti parte del package `it.hourglass.myTalk.client.view.profilemanagement`, poichè hanno bisogno di un gran numero di elementi grafici per modificare i dati personali principali ed estesi dell'utente;
- `RegistrationPresenter`, facente parte del package `it.hourglass.myTalk.client.presenter`, poichè deve controllare la validità dei dati personali che l'utente ha inserito durante la registrazione;
- `PersonalDataProfileManagementPresenter`, facente parte del package `it.hourglass.myTalk.client.presenter.profilemanagment`, poichè si deve occupare della gestione della modifica dei dati personali dell'utente;

- `User`, facente parte del package `it.hourglass.myTalk.client.shared`, poichè deve mantenere tutti i dati personali che riguardino il profilo associato a un utente.

3 Esiti Test di unità

Di seguito vengono riportati gli esiti dei test di unità effettuati sulle componenti software. Legenda delle abbreviazioni utilizzate per i nomi dei pacchetti software:

ELB: it.hourglass.myTalk.server.wsserver.elaborator
ITF: it.hourglass.myTalk.server.wsserver.interf
DAO: it.hourglass.myTalk.server.model.dao
SHA: it.hourglass.myTalk.client.shared;
PRE: it.hourglass.myTalk.client.presenter
TEST: it.hourglass.myTalk.test
BSL: it.it.hourglass.myTalk.server.model.businesslogic

3.1 Ambiente di lavoro

Per quanto riguarda le componenti lato server si è utilizzato solamente il *framework JUnit* con la librerie aggiuntive fornite da *Mockito* in `mockito-all-1.9.5.jar` per potere creare degli oggetti *Mock*. Riguardo le componenti lato client si è utilizzato, oltre a queste ultime risorse, anche le librerie fornite da *gwtmockito* in `gwtmockito-1.1.0.jar` che hanno permesso al team di realizzare oggetti *Mock* anche in ambiente *GWT* e le librerie di *gwt-test-utils* contenute in `ingwt-test-utils.jar` per testare alcune funzionalità disponibili sempre con *GWT*.

3.2 Componenti lato server

3.2.1 Test di componenti nel package it.hourglass.myTalk.server.wsserver

Registrazione nickname lato server

- **Classe che realizza i test:** TEST.DialerTest
- **Classi testate:** ELB.Dialer
- **Descrizione:** Verifica della funzionalità principali della classe Dialer.
- **Funzionalità testate:**
 - **regTest():** Verifica che un nickname ricevuto venga correttamente ed effettivamente registrato nel registro della classe Dialer
 - **byeUser1Test():** Verifica che un nickname ricevuto venga correttamente ed effettivamente cancellato nel registro della classe Dialer attraverso il metodo `byeUser(WsConnection ws)` della classe ELB.Dialer.
 - **byeUser2Test():** Verifica che un nickname ricevuto venga correttamente ed effettivamente cancellato nel registro della classe Dialer attraverso il metodo `byeUser(WsConnection ws, String nick)` della classe ELB.Dialer.

- `regTest()`: Verifica che un nickname ricevuto venga correttamente ed effettivamente registrato nel registro della classe Dialer
 - `sendNotificationOnTest()`: Verifica che il metodo `sendNotificationOn()` e `sendNotificationOff()` della classe Dialer eseguano correttamente utilizzando una lista di stringhe appositamente creata
 - `sendOfferTest()`: Verifica che un nickname ricevuto venga correttamente ed effettivamente registrato nel registro della classe Dialer
 - `sendCandidateTest()`: Verifica che proceda correttamente l'invio di un candidato attraverso il metodo `sendCandidate()` della classe Dialer
 - `getFriendStatusTest()`: Verifica che venga eseguito correttamente il metodo `getFriendStatus()` della classe Dialer con parametri appositamente creati.
- **Stato:** Superato
 - **Errori:** Nessuno.

Elaborazione JSON

- **Classe che realizza i test:** TEST.SwitchTest
- **Classi testate:** ELB.Switch
- **Descrizione:** Verifica della corretta interpretazione dei JSON in arrivo.
- **Funzionalità testate:**
 - `goSwitchTest6()`: Attraverso la creazione di un JSON ad-hoc per questo test, si verifica che il metodo `goSwitch()` della classe Switch interpreti correttamente il tipo del JSON (in questo caso di tipo 6) e ne esegua le corrette istruzioni cioè l'eliminazione del nickname dal registro dei nickname della classe ELB.Dialer.
 - `goSwitchTest0()`: Attraverso la creazione di un JSON ad-hoc per questo test, si verifica che il metodo `goSwitch()` della classe Switch interpreti correttamente i JSON (in questo caso di tipo 0) e ne esegua le corrette istruzioni cioè la registrazione del nick contenuto nel JSON nel registro dei nickname della classe ELB.Dialer.
 - `goSwitchTest5()`: Attraverso la creazione di un JSON ad-hoc per questo test, si verifica che il metodo `goSwitch()` della classe Switch interpreti correttamente i JSON (in questo caso di tipo 5) e ne esegua le corrette istruzioni cioè l'invio della lista amici relativa al nickname contenuta nel JSON tramite il metodo `sendList(String onick, List<String> l)` della classe ELB.Dialer.
 - `goSwitchTest4()`: Attraverso la creazione di un JSON ad-hoc per questo test, si verifica che il metodo `goSwitch()` della classe Switch interpreti correttamente i JSON (in questo caso di tipo 4) e ne esegua le corrette istruzioni cioè invii

correttamente il nick e le informazioni contenute nel JSON attraverso il metodo `sendIncoming(String onick, String rnick, String status, String stream)` della classe `ELB.Dialer`.

- `goSwitchTest123()`: Attraverso la creazione di un JSON ad-hoc per questo test, si verifica che il metodo `goSwitch()` della classe `Switch` interpreti correttamente i JSON (in questo caso di tipo 1, 2 e 3) e ne esegua di conseguenza le corrette istruzioni.

- **Stato:** Superato
- **Errori:** Nessuno.

Analisi tipi dei JSON

- **Classe che realizza i test:** `TEST.ParserInTest`
- **Classi testate:** `ITF.ParserIn`
- **Descrizione:** Verifica della corretta analisi del tipo dei JSON in arrivo.
- **Funzionalità testate:**
 - `getIDTypeTest()`: Creando un'apposita stringa si verifica che ne venga interpretato l'esatto tipo, in questo caso dalla stringa `offer` viene ricavato il corretto tipo 1.
- **Stato:** Superato
- **Errori:** Nessuno.

Comunicazioni Websocket

- **Classe che realizza i test:** `TEST.ParserOutTest`
- **Classi testate:** `ITF.ParserOut`
- **Descrizione:** Verifica dell'invio di messaggi di testo via Websocket.
- **Funzionalità testate:**
 - `sendToClientTest()`: Verifica che il metodo `sendToClient(WsConnection wsconnection, String message)` invii correttamente un messaggio di testo via Websocket analizzando i messaggi stampati nella Console Java.
- **Stato:** Superato
- **Errori:** Nessuno.

3.2.2 Test di componenti nel package `it.hourglass.myTalk.server.model`

Operazioni con il DAO

- **Classe che realizza i test:** `TEST.DaoImplTest`
- **Classi testate:** `DAO.DaoImpl`
- **Descrizione:** Verifica il corretto funzionamento delle operazioni di recupero e salvataggio dei dati sul database dell'applicativo software, inoltre testa la validazione di un utente con dei dati fittizi appositamente creati.
- **Funzionalità testate:**
 - `testStore()`: Verifica che il metodo `storeUser(User)` della classe `DAO.DaoImpl` salvi effettivamente un utente nel database con il quale il DAO si collega.
 - `testcheckValidation()`: Verifica che il metodo `testcheckValidation(String uniqueId, String Validation)` della classe `DAO.DaoImpl` reperisca correttamente i dati per poter controllare che il codice di validazione corrisponda con il codice dell'utente in questione.
 - `testfetchMessages()`: Verifica che il metodo `fetchMessages(String uniqueId)` della classe `DAO.DaoImpl` reperisca effettivamente la lista dei messaggi dell'utente in questione.
 - `testFriendship()`: Verifica che i metodi `updateFriendship(Friendships fr)` e `deleteFriendship(Friendships fr)` della classe `DAO.DaoImpl` gestiscano correttamente l'aggiornamento e l'eliminazione delle amicizie di un utente.
- **Stato:** Superato
- **Errori:** Nessuno.

Accertamento delle funzionalità di crittazione e decrittazione

- **Classe che realizza i test:** `TEST.SecurityTest`
- **Classi testate:** `BSL.Security`
- **Descrizione:** Verifica di alcune funzionalità della classe `Security`.
- **Funzionalità testate:**
 - `generateValidationTest()`: Verifica che il metodo `generateValidation(String)` generi una stringa randomizzata.
 - `generateValidationTest()`: Verifica che il metodo `generateValidation(String)` generi un codice di sicurezza a partire da un message digest passato come parametro del metodo. La verifica avviene controllando che il message digest e il codice generato siano differenti.

- `checkPasswordTest()`: Verifica che il metodo `checkPassword(String,String)` dia esito positivo solo nel caso in cui vengono passati, come parametri: il codice di sicurezza generato in precedenza dal metodo `generateValidation(String)` e il message digest usato per creare il codice di sicurezza.
- **Stato:** Superato
- **Errori:** Nessuno.

Verifica della correttezza dei metodi remoti

- **Classe che realizza i test:** TEST.RPCServiceTest
- **Classi testate:** BSL.RPCServiceImpl
- **Descrizione:** Verifica di alcune funzionalità della classe BSL.RPCServiceImpl.
- **Funzionalità testate:**
 - `storeUserTest()`: Verifica che il metodo `storeUser(User,boolean)` non permetta di associare una email già in uso da un'altro utente precedentemente iscritto alla piattaforma. Il controllo avviene determinando se la stringa di errore ritornata dal metodo di test coincide con quella una attesa.
 - `registerTest()`: Verifica che il metodo `register()` non permetta di associare una email già in uso da un'altro utente precedentemente iscritto alla piattaforma. Il controllo avviene determinando se la stringa di errore ritornata dal metodo di test coincide con quella una attesa.
 - `checkLoginTest()`: Verifica che il metodo `checkLogin(String,String)` ritorni come parametro false, quando vengono usati come parametri di invocazione del metodo: username e password non registrati nella piattaforma. Il controllo avviene determinando se la stringa di errore ritornata dal metodo di test coincide con quella una attesa.
 - `sendMessageTest()`: Verifica che il metodo `sendMessage(Message)` non spedisca con successo un messaggio con campo destinatario nullo. Il controllo avviene determinando se la stringa di errore ritornata dal metodo di test coincide con quella una attesa.
 - `deleteMessageTest()`: Verifica che il metodo `deleteMessage(Message)` non cancelli un messaggio con campo destinatario nullo. Il controllo avviene determinando se la stringa di errore ritornata dal metodo di test coincide con quella una attesa.
 - `friendshakeTest()`: Verifica che il metodo `friendshake(Message,boolean)` non riesca ad inviare un messaggio, con campo destinatario settato a null, ad un altro utente della piattaforma.

- `SignInTest()`: Verifica che il metodo `SignIn(String)` ritorni un'oggetto `SignIn` non vuoto, quando invoco il metodo di test con uno `uniqueId` valido.
 - `fetchFriendProfileTest()`: Verifica che il metodo `fetchFriendProfile(String)` ritorni un'oggetto vuoto, nel caso in cui il parametro passato al metodo di test, che identifica univocamente un'utente, non risulta corrispondere a nessuno di essi.
 - `setValidationTest()`: Verifica che il metodo `setValidation(String)` ritorni l'identificativo univoco di utente registrato nella piattaforma, passando come parametro l'indirizzo email di quell'utente.
 - `checkValidationTest()`: Verifica che il metodo `checkValidation(String,String)` ritorni un parametro booleano settato a `false`, dato che vengono usati dei parametri non validi nell'invocazione del metodo di test.
 - `setPasswordTest()`: Verifica che il metodo `setPassword(String,String)` non riesca a settare la password di un'utente non presente nella piattaforma.
 - `removeFriendTest()`: Verifica che il metodo `removeFriendTest(String,String)` riesca a rimuovere dalla lista contatti dell'utente richiedente, un suo amico.
- **Stato:** Superato
 - **Errori:** Nessuno.

Verifico se l'invio delle email avviene correttamente

- **Classe che realizza i test:** `TEST.EmailSenderTest`
 - **Classi testate:** `BSL.EmailSender`
 - **Descrizione:** Verifica di alcune funzionalità della classe `BSL.EmailSender`.
 - **Funzionalità testate:**
 - `testSendEmail()`: Verifica che il metodo `EmailSender(String,String,String)` riesca a inviare una mail con il campo mittente impostato alla mail utilizzata dall'applicativo per inviare i messaggi con il codice di sicurezza per il cambio password. La verifica avviene semplicemente lanciando il test e rilevando che questo non segnala errori durante la sua esecuzione.
 - `testSendEmail2()`: Verifica che il metodo `SendEmail()` non riesca ad inviare una mail con il campo mittente non valido. Viene riscontrato un'eccezione a run-time che però viene correttamente gestita. L'esito finale risulta quindi soddisfacente.
- **Stato:** Superato
 - **Errori:** Nessuno.

3.3 Componenti lato client

3.3.1 Test di componenti nel package `it.hourglass.myTalk.client.shared`

Gestione messaggi e lista amici dei profili utente

- **Classe che realizza i test:** `TEST.ProfileTest`
- **Classi testate:** `SHA.Profile`
- **Descrizione:** Verifica la corretta gestione della lista amici e lista messaggi di un profilo associato ad un utente.
- **Funzionalità testate:**
 - `FriendTest()`: Verifica che attraverso i metodi della classe soggetta al test sia possibile aggiungere effettivamente un amico alla lista amici di un utente creato ad-hoc. Vengono inoltre chiamati metodi di tipo get e set della classe `Profile` per garantire la loro correttezza nell'esecuzione.
 - `MessageTest()`: Viene aggiunto un messaggio creato appositamente per il test in questione alla lista messaggi del profilo di un utente, una volta chiamati i metodi `sendMessageNotificationWS(String uniqueId)` e `refreshMessageList()` di `SHA.Profile` si verifica che la lista messaggi non sia vuota, ma sia stato aggiunto il messaggio creato precedentemente. Vengono inoltre chiamati metodi di tipo get e set della classe `User` per garantire la loro correttezza nell'esecuzione.
 - `SignInTest()`: Viene aggiunto un messaggio creato appositamente per il test in questione alla lista messaggi del profilo di un utente, una volta chiamati i metodi `sendMessageNotificationWS(String uniqueId)` e `refreshMessageList()` di `SHA.Profile` si verifica che la lista messaggi non sia vuota, ma sia stato aggiunto il messaggio creato precedentemente.
- **Stato:** Superato
- **Errori:** Nessuno.

3.3.2 Test di componenti nel package `it.hourglass.myTalk.client.presenter`

Gestione della view base

- **Classe che realizza i test:** `TEST.BaseViewPreTest`
- **Classi testate:** `PRE.BaseViewPresenter`
- **Descrizione:** Verifica la corretta gestione della view base dopo la chiamata di alcuni metodi della classe stessa.

- **Funzionalità testate:**
 - `removeContactFromContactListTest()`: Verifica che dopo aver chiamato il metodo `removeContactFromContactList(String uniqueId)` della classe da testare, venga lanciato un evento del tipo `HomePageRequestEvent.class`.
 - `switchEditProfileTest()`: Verifica che dopo aver chiamato il metodo `switchEditProfile` della classe da testare, venga lanciato un evento del tipo `HomePageRequestEvent.class`.
- **Stato:** Superato
- **Errori:** Nessuno.

Gestione stato UserList

- **Classe che realizza i test:** `TEST.UserlistPreTest`
- **Classi testate:** `PRE.UserlistPresenter`
- **Descrizione:** Verifica la corretta gestione dello status di un oggetto della classe `PRE.UserlistPresenter`
- **Funzionalità testate:**
 - `changeStatusTest()`: Verifica che il metodo `changeStatus()` della classe da testare cambi effettivamente lo stato dell'oggetto `contactInformation` all'interno del corpo del metodo.
- **Stato:** Superato
- **Errori:** Nessuno.

Gestione sessioni con il server e menù

- **Classe che realizza i test:** `TEST.SignedMenuPreTest`
- **Classi testate:** `PRE.SignedMenuPresenter`
- **Descrizione:** Verifica la creazione di una sessione con il server
- **Funzionalità testate:**
 - `SessionTest()`: Verifica che il metodo `doResetSession()` della classe da testare crei effettivamente una sessione con il server.
- **Stato:** Superato
- **Errori:** Nessuno.

Gestione invio messaggi

- **Classe che realizza i test:** TEST.PopupMessagePreTest
- **Classi testate:** PRE.PopupMessagePresenter
- **Descrizione:** Verifica il corretto comportamento del presenter in questione, all'invio di un messaggio.
- **Funzionalità testate:**
 - `PopupMessagePresenterTest()`: Verifica che il metodo `sendMessage(Message msg)` della classe da testare applichi correttamente le proprie istruzioni.
- **Stato:** Superato
- **Errori:** Nessuno.

Accertamento delle funzionalità di controllo dei dati in fase di registrazione

- **Classe che realizza i test:**RegistrationPresenterTest
- **Classi testate:**PRE.RegistrationPresenter
- **Descrizione:**Verifica di alcune funzionalità della classe `RegistrationPresenter`.
- **Funzionalità testate:**
 - `getAllCheckedTest()`: Verifica che il metodo `getAllChecked()` ritorni un booleano settato a true in seguito alla compilazione corretta e completa dei campi necessari alla registrazione. Per effettuare questo test è stato necessario creare un mock della classe `display` e ridefinire i metodi che recuperano i dati inseriti nella view dell'applicativo, in modo da ritornare solo dati validi, così da rendere il test superabile.
- **Stato:** Superato
- **Errori:** Nessuno.

Accertamento nell'uso dei parametri di accesso alla piattaforma

- **Classe che realizza i test:** TEST.PresenterLoginTest
- **Classi testate:** PRE.LoginPresenter

- **Descrizione:** Verifica di alcune funzionalità della classe `LoginPresenter`.
- **Funzionalità testate:**
 - `doLoginControlTest()`: Verifica che il metodo `doLoginControl(String,String)` invochi la chiamata all'RPC per accertarsi che i parametri usati per il metodo siano già presenti nel Database della piattaforma, e corrispondano in maniera univoca ad un utente. Il seguente test avviene utilizzando diversi mock per simulare il comportamento della parte grafica attraverso la quale il presenter recupera i dati inseriti dall'utente, per autenticarsi alla piattaforma.
- **Stato:** Superato
- **Errori:** Nessuno.

Verifico le operazioni disponibili sulla lista contatti

- **Classe che realizza i test:** `TEST.ContactListPresenterTest`
- **Classi testate:** `PRE.ContactListPresenter`
- **Descrizione:** Verifica di alcune funzionalità della classe `ContactListPresenter`. I vari test prima creano dei mock necessari a sostituire i parametri per generare un'istanza della classe testata. Viene caricata in un secondo momento un utente fittizio che sarà il soggetto dei test assieme alla lista che lo contiene. Queste operazioni vengono ripetute all'inizio di ogni test.
- **Funzionalità testate:**
 - `removecontactTest()`: Verifica che il metodo `removeContact()` elimini un contatto dalla lista delle amicizie.
 - `addUserTest()`: Verifica che il metodo `addUser()` aggiunga un contatto alla lista delle amicizie.
 - `removeContactTest()`: Verifica che il metodo `removeContact()` elimini la lista dei contatti dalla view.
- **Stato:** Superato
- **Errori:** Nessuno.

Accertamento della verifica dei parametri nella fase di registrazione

- **Classe che realizza i test:** `TEST.TestValuesCheck`
- **Classi testate:** `PRE.ValuesCheck`

- **Descrizione:** Verifica di alcune funzionalità della classe `ValuesCheck`.
- **Funzionalità testate:**
 - `testcheckUniqueId(String)`: Verifica che il metodo `checkUniqueId(String)` non accetti uno username non valido, durante la fase di registrazione alla piattaforma. La verifica viene accertata mediante il controllo del messaggio di errore ritornato dal metodo classe soggetta ai test.
 - `testcheckPassword(String,String)`: Verifica che il metodo `checkPassword(String)` non accetti una password non valida, durante la fase di registrazione alla piattaforma. La verifica viene accertata mediante il controllo del messaggio di errore ritornato dal metodo classe soggetta ai test.
 - `testcheckName(String)`: Verifica che il metodo `checkPassword(String)` non accetti un campo nome lasciato vuoto, durante la fase di registrazione, alla piattaforma. La verifica viene accertata mediante il controllo del messaggio di errore ritornato dal metodo della classe soggetta ai test.
 - `testcheckLastName(String)`: Verifica che il metodo `checkLastName(String)` non accetti un campo cognome lasciato vuoto, durante la fase di registrazione, alla piattaforma. La verifica viene accertata mediante il controllo del messaggio di errore ritornato dal metodo della classe soggetta ai test.
 - `testcheckEmail()`: Verifica che il metodo `checkEmail(String)` non accetti una email non valida.
- **Stato:** Superato
- **Errori:** Nessuno.

Accertamento della verifica dei parametri inseriti dall'utente

- **Classe che realizza i test:** `TEST.ExtendedDataProfileTest`
- **Classi testate:** `PRE.ProfileManagment.ExtendedDataProfileManagmentPresenter`
- **Descrizione:** Verifica di alcune funzionalità della classe `ExtendedDataProfileManagmentPresenter`.
- **Funzionalità testate:**
 - `checkValueTest()`: Verifica che il metodo `checkValue()` controlli adeguatamente i campi dati modificati dall'utente. I dati in questione fanno riferimento ai campi che un utente può aggiungere, modificare una volta autenticato. Il test viene effettuato con l'ausilio di diversi mock necessari in primis per l'istanziamento di un'oggetto appartenente alla classe di test e poi per ridefinire i parametri di ritorno della view, così da poter restituire oggetti ad-hoc, in questo caso un

parametro di lunghezza superiore a quella consentita così da far fallire il test, ricevendo un risultato atteso.

- **Stato:** Superato
- **Errori:** Nessuno.

Verifico che i costruttori delle classi `NewAvatarPpoupPresenter`, `NewEmailConfirmPopupPresenter` non diano luogo a eccezioni incontrollate

- **Classe che realizza i test:** `TEST.OtherProfileManagment`
- **Classi testate:** `PRE.ProfileManagment.NewAvatarPpoupPresenter`, `PRE.ProfileManagment.NewEmailConfirmPopupPresenter`
- **Descrizione:** Verifica di alcune funzionalità della classe `ExtendedDataProfileManagmentPresenter`.
- **Funzionalità testate:**
 - `checkConstructors()`: Verifico i costruttori delle classi `NewAvatarPpoupPresenter`, `NewEmailConfirmPopupPresenter`.
- **Stato:** Superato
- **Errori:** Nessuno.

Accertamento della verifica dei parametri inseriti dall'utente

- **Classe che realizza i test:** `TEST.PersonalDataProfileManagmentPresenterTest`
- **Classi testate:** `PRE.PersonalDataProfileManagmentPresenter`
- **Descrizione:** Verifica di alcune funzionalità della classe `PRE.PersonalDataProfileManagmentPresenter`.
- **Funzionalità testate:**
 - `checkValueTest()`: Verifica che il metodo `checkValue()` appartenente alla classe di test controlli nel modo adeguato, senza dare luogo ad imprevisti, i dati raccolti dalla view associata.
- **Stato:** Superato
- **Errori:** Nessuno.

4 Copertura Test di unità

La copertura dei test di unità non tiene conto della gran parte dei metodi di tipo get e set che hanno per loro natura una correttezza intrinseca, inoltre sono stati esclusi packages come stabilito nel documento *Piano di Qualifica_v4.0.pdf* alla sezione 9.2.

Nella pagina seguente vengono riportati gli esiti dei package contenenti componenti per le quali sono stati effettuati test di unità.

Si può notare come in questi package sottoposti a test, il livello di copertura sia superiore alla soglia del 75%. Ciò non è stato possibile per il package `it.hourglass.myTalk.presenter` dato che molte classi al suo interno non presentano una logica con una complessità tale da giustificare la fruizione di risorse del team per essere testate.

it.hourglass.myTalk.server









Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
elaborator		72%		72%	10	38	39	154	1	17	0	2
interf		58%		n/a	7	13	19	37	7	13	2	5
test		93%		50%	7	26	0	117	0	19	0	4
exception		45%		n/a	0	2	5	10	0	2	0	2
Total	307 of 1.337	77%	17 of 50	66%	24	79	63	318	8	51	2	13

Figura 1: Esito test di copertura nel package `it.hourglass.myTalk.server.wssserver`
`it.hourglass.myTalk.server.model.dao`







Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
DAOImpl		67%		58%	6	23	52	140	1	17	0	1
HibernateUtil		88%		n/a	1	3	1	7	1	3	0	1
DAOImplTest		100%		50%	5	10	0	39	0	5	0	1
Total	125 of 572	78%	10 of 22	55%	12	36	53	186	2	25	0	3

Figura 2: Esito test di copertura nel package `it.hourglass.myTalk.server.model.dao`
`it.hourglass.myTalk.client.shared`


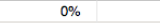

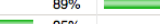
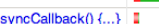
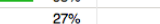

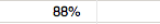

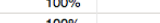

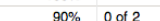


Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
Profile.new AsyncCallback() {...}		0%		100%	3	3	6	6	3	3	1	1
Profile		89%		100%	4	20	6	40	4	19	0	1
User		95%		n/a	3	39	5	83	3	39	0	1
Profile.new AsyncCallback() {...}		27%		n/a	2	3	4	6	2	3	0	1
SignIn		88%		n/a	2	7	2	18	2	7	0	1
Message		100%		n/a	0	16	0	38	0	16	0	1
Friendships		100%		n/a	0	11	0	23	0	11	0	1
Total	55 of 558	90%	0 of 2	100%	14	99	21	211	14	98	1	7

Figura 3: Esito test di copertura nel package `it.hourglass.myTalk.client.shared`












Element	Coverage	Covered Instructio...	Missed Instructions	Total Instructions
it.hourglass.myTalk.server.model.businesslogic		79,0 %	757	958
Security.java		100,0 %	29	29
EmailSender.java		95,1 %	8	163
RPCServiceImpl.java		77,3 %	573	741
FileServiceImpl.java		0,0 %	25	25
it.hourglass.myTalk.client.presenter.ProfileManagment		78,6 %	469	597
NewEmailConfirmPopupPresenter.java		91,9 %	6	74
ExtendedDataProfileManagmentPresenter.java		90,6 %	15	159
NewAvatarPopupPresenter.java		82,4 %	9	51
PersonalDataProfileManagmentPresenter.java		79,3 %	56	271
AvatarProfileManagmentPresenter.java		0,0 %	42	42

Figura 4: Esito test di copertura nel package `it.hourglass.myTalk.presenter.ProfileManagment`
e di `it.hourglass.myTalk.server.model.businesslogic`

5 Esiti Test di integrazione

Verifica delle componenti presenter che usano chiamate a metodi remoti

- **Classe che realizza i test:** TestPresenterIntegrations
- **Classi testate:** Anche se non vengono istanziate, il codice dei metodi di test proviene dai metodi dei presenter che invocano le chiamate remote al server. Perciò le classi testate sono (le sigle PRE, BSL e DAO fanno riferimento alla legenda contenuta nella sezione 3):
 - PRE.LoginPresenter,
 - PRE.BaseViewPresenter,
 - PRE.ContactProfilePresenter,
 - PRE.FriendMessagePresenter,
 - PRE.RegistrationPresenter,
 - PRE.TextMessagePresenter,
 - PRE.SignedMenuPresenter,
 - PRE.ContactManagmentPresenter,
 - PRE.PopupMessagePresenter,
 - PRE.PasswordRecoverPresenter,
 - BSL.*,
 - DAO.*
- **Descrizione:** Vengono tralasciate tutte le invocazioni ai metodi delle view dei vari presenter in modo da testare esclusivamente il comportamento delle componenti presenter e model, facendo riferimento all'architettura MVP adottata nel progetto.
- **Funzionalità testate:**
 - **testLoginPresenter():** Verifica che il metodo `checkLogin(String,String,AsyncCallback<Boolean>` riesca ad essere invocato senza sollevare errori a run-time. Il metodo `checkLogin(String,String,AsyncCallback<Boolean>` della classe `BSL.ServiceImpl` a sua volta invoca altri metodi della classe `DaoImpl`, dimostrando quindi l'interazione tra la classe `PRE.LoginPresenter`, e le classi sul server `BSL.ServiceImpl`, `DAO.DaoImpl` che operano nella base di dati. La correttezza viene dimostrata dal fatto che non sono presenti eccezioni non gestite a run-time e il test termina prima del periodo di attesa per il ritorno della chiamata remota dal server una volta invocata, dimostrando che non c'è rischio di loop.
 - **testBaseViewPresenter():** Verifica che il metodo `checkSession(AsyncCallback<String>)` venga invocato ed operi senza sollevare eccezioni o loop infiniti, infatti termina sempre prima del tempo di attesa per il ritorno della RPC dal server.

- `testContactProfilePresenter()`: Verifica che il metodo `fetchFriendProfile(String, AsyncCallback<User>)` una volta invocato usando come parametro uno `uniqueId` inesistente, non restituisca il riferimento a nessun utente registrato sulla piattaforma. Viene quindi ritornato un riferimento nullo dato che lo `uniqueId` non è valido. La verifica viene fatta sul tipo ritornato dal metodo `fetchFriendProfile(String, AsyncCallback<User>)` controllando che sia pari a `null`.
- `testFriendContactPresenter()`: Verifica che il metodo `removeFriend(String, AsyncCallback<String>)` una volta invocato con dei parametri non validi, restituisca una stringa di errore, come premeditato.
- `testRegistrationPresenter()`: Verifica che il metodo `checkValidation(String, String, AsyncCallback<Boolean>)` funzioni senza incappare in eccezioni incontrollate. La funzione controlla se i due parametri stringa corrispondono rispettivamente allo `uniqueId` del utente registrato, e al suo codice di sicurezza, in risposta il metodo ritorna un parametro `Boolean`, in caso positivo si procederà con lo switch della view, altrimenti si richiederà di reinserire il codice di sicurezza.
- `testTextMessagePresenter()`: Verifica che il metodo `deleteMessage(Message, AsyncCallback<String>)` non possa cancellare un messaggio che ha come mittente e destinatario due utenti non presenti nella piattaforma. Il metodo ritorna una stringa di errore che avvisa l'utente dell'errore riscontrato durante il tentativo di cancellazione.
- `testSignedMenuPresenter()`: Verifica che il metodo `resetSession(AsyncCallback<Boolean>)` resetti, se presente, una sessione attiva tra il client e il server. Viene controllato il valore del tipo di ritorno dal metodo remoto `resetSession(AsyncCallback<Boolean>)`, e dato che questo corrisponde alle attese il metodo dimostra un comportamento atteso.
- `testContactManagmentPresenter()`: Verifica che il metodo `sendMessage(Message, AsyncCallback<String>)` avverta l'utente nel caso in cui stia per inoltrare una richiesta di amicizia ad utente non registrato nella piattaforma. Il test avviene settando a `true` il parametro dell'oggetto message che identifica la richiesta di amicizia e inserendo nel campo dell'utente destinatario un identificativo univoco non valido, così che il metodo ritorni un messaggio di errore.
- `testPopupMessagePresenter()`: Verifica che il metodo `sendMessage(Message, AsyncCallback<String>)` possa inviare un messaggio. Il parametro di tipo `Message` viene creato a partire dalla pressione di un bottone sulla view dell'applicativo, non sarà quindi necessario inserire il nome del destinatario del messaggio, in questo modo non sarà necessario verificare la bontà del parametro che identifica il mittente dato che sarà creato dalla view stessa. Finita la premessa, il test da esito positivo anche se il parametro non risulta essere

registrato, ma come detto in precedenza questo esito era atteso per via di come è stato strutturata la view.

- `testPasswordRecoverPresenter()`: Verifica che i metodi
 - * `setValidation(String, AsyncCallback<String>`,
 - * `checkValidation(String, String, AsyncCallback<Boolean>`,
 - * `setPassword(String, String, AsyncCallback<String>`

riescano con successo a cambiare la password di utente che esegue correttamente tutto il procedimento per il cambio password. In primis viene invocato il metodo `setValidation(String, AsyncCallback<String>)` con un parametro non valido per l'indirizzo email dell'utente richiedente del cambio password. Il test come in preventivo fallisce, e non dimostra nessun comportamento anomalo o inatteso. Una volta verificata l'email viene inviato per email un codice di sicurezza da inserire nel secondo passaggio per cambiare password.

Il metodo `checkValidation(String, String, AsyncCallback<Boolean>)` si occupa di inviare l'identificativo univoco dell'utente e il suo codice di sicurezza. Il test verifica che non vi siano eccezioni non gestite o che la chiamata al metodo remoto non ritorni più al client, ma durante l'esecuzione non si registrano comportamenti anomali. Il metodo `setPassword(String, String, AsyncCallback<String>)` viene invocato per effettuare il cambio password passando come parametri l'identificativo univoco dell'utente e la nuova password da lui scelta per sostituire quella precedente. Il metodo durante l'esecuzione non fa registrare comportamenti anomali che possano far supporre ad errori.

- **Stato:** Superato
- **Errori:** Nessuno.