



UNIVERSITÀ
DEGLI STUDI
FIRENZE

CORSO DI LAUREA MAGISTRALE
IN INFORMATICA

Tecniche Avanzate di Programmazione - Password Manager

Valentino Marano - *valentino.marano@stud.unifi.it*

4 Febbraio 2019

Contents

1	Introduzione	1
1.1	Obiettivo	1
1.2	Strumenti utilizzati	2
2	Implementazione	3
2.1	Folder Main	4
2.1.1	Package app	4
2.1.2	Package app.db	5
2.1.3	Package app.gui	7
2.2	Folder Test	9
2.2.1	Package test.gui	9
2.2.2	Package test.db	10
3	Difficoltà riscontrate	13
4	Dettagli	15
5	Possibili Estensioni	16

Introduzione

1.1 Obiettivo

Lo scopo del presente progetto è la creazione di un'applicazione desktop per la gestione di password. Il programma offre le seguenti features:

- Salvataggio dei dati su un Database NoSQL Mongo
- Interfaccia grafica sia per il login al programma che per la gestione delle password
- Avviso in caso di presenza di password scadute
- Possibilità di ricerca tramite parte del sito o dell'utenza
- Vista di riepilogo di tutte le password e possibilità di ordinamento in base a sito, utente, password o data di scadenza
- Gestione multi-utente per consentire l'utilizzo a diverse persone sullo stesso PC mantenendo i dati separati

1.2 Strumenti utilizzati

Per lo sviluppo e il test del progetto sono stati usati i seguenti strumenti:

Maven Consente di gestire le dipendenze e la build automation del progetto.

Travis Consente di avere una CI (Continuous Integration) gestita in maniera semplice attraverso il file di configurazione. Ad ogni push sul repository Git viene eseguita una build completa. Al termine viene inviato il riepilogo della code coverage a Coveralls e il riepilogo della code quality a Sonarcloud.

Coveralls Consente di consultare in maniera semplice il riepilogo della code coverage avendo sempre sotto controllo eventuali porzioni di codice non testate.

Sonar Consente di analizzare la qualità del codice segnalando tutte le anomalie riscontrate con la corrispettiva severity e il tempo di “debito” accumulato.

Docker Consente di usare dei contenitori all’interno dei quali è possibile eseguire server, database, singole applicazioni, sistemi operativi, ...









PIT Consente di verificare la completezza dei test preparati. Questo tool infatti genera una serie di mutazioni al codice delle classi del progetto che vengono quindi sottoposte ai test preparati. Eventuali mutazioni “sopravvissute” ai test vengono segnalate poiché indicano una probabile incompletezza nei test.

SWTBot È un tool per il test di interfacce grafiche basate su SWT, Eclipse o GEF (Graphical Editing Framework). Consente durante i test di simulare il comportamento di un utente che interagisce con l’interfaccia: fa click sui pulsanti, scrive nei campi editabili, vede cosa è presente nella finestra, ...

Fongo Il programma utilizza il Database Mongo come server per il salvataggio dei dati. Per evitare di dover mockare tutti i metodi delle API Java di Mongo, negli Unit Test è stato utilizzato **Fongo** che consente di simulare il server Mongo mettendo a disposizione praticamente tutte le medesime funzionalità.

Implementazione

In questo capitolo sarà esposta la struttura del progetto che, come si vede dall'immagine, è composta da due folder distinte: *src/main/java* per le classi che compongono il programma e *src/test/java* per le classi di unit/integration test.

```
▼  > com.valentino.tap.password_manager [Password Manager master]
  ▼  > src/main/java
    ►  > com.valentino.tap.password_manager.app
    ►  com.valentino.tap.password_manager.app.db
    ►  > com.valentino.tap.password_manager.app.gui
  ▼  > src/test/java
    ►  > com.valentino.tap.password_manager.test.db
    ►  > com.valentino.tap.password_manager.test.gui
```

2.1 Folder Main

Il progetto è diviso in 3 package differenti: *app*, *app.db* e *app.gui*.

2.1.1 Package app

Questo package gestisce la parte generale dell'applicazione. Qui sono contenute 3 classi:

Password Questa classe modella l'oggetto password (composta da sito, utente, password e data di scadenza) e ne contiene i relativi metodi getter/setter.



PasswordManager Questa classe contiene le funzionalità per la gestione delle password: aggiunta, aggiornamento, cancellazione, verifica esistenza e ricerca tramite parte del sito o dell'utenza.



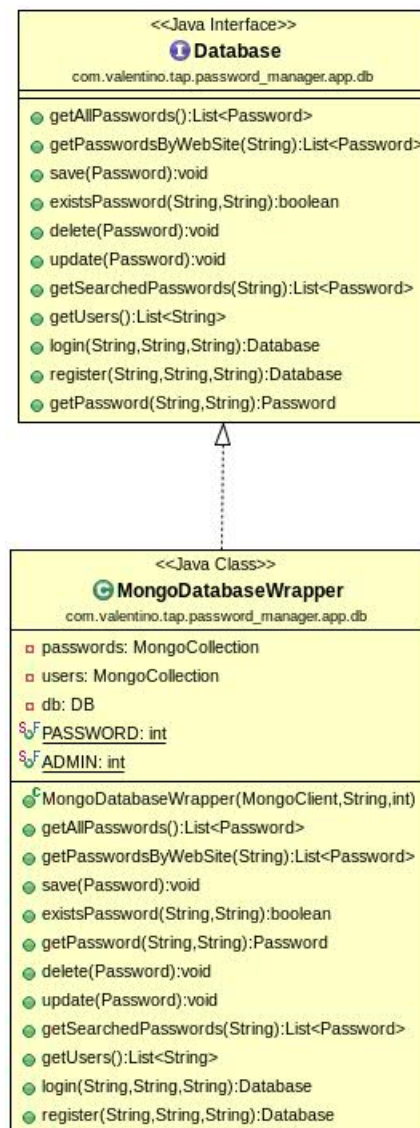
Application Questa classe contiene semplicemente il metodo principale dell'applicazione: mostra la finestra di accesso e, una volta effettuato il login/registrazione, mostra la finestra per la gestione delle password presenti per l'utente.

2.1.2 Package app.db

In questo package sono presenti le classi per la gestione dell'interazione con il Database.

Database Questa interfaccia contiene la definizione di tutti i metodi richiamati per interagire con il database. Avendo definito un'interfaccia è molto più semplice l'aggiunta di nuove tipologie di database con cui interagire: sarà sufficiente creare una nuova classe che implementi tale interfaccia.

MongoDatabaseWrapper Questa classe è attualmente l'unica implementazione dell'interfaccia **Database** e contiene tutti i metodi per l'interazione con il database NoSQL Mongo. Viene utilizzata nel progetto sia per la gestione dell'accesso iniziale aprendo una connessione al database *admin* (per recuperare l'elenco degli utenti presenti e registrare eventuali nuovi utenti), sia per la gestione delle password aprendo una connessione al database relativo all'utente loggato.

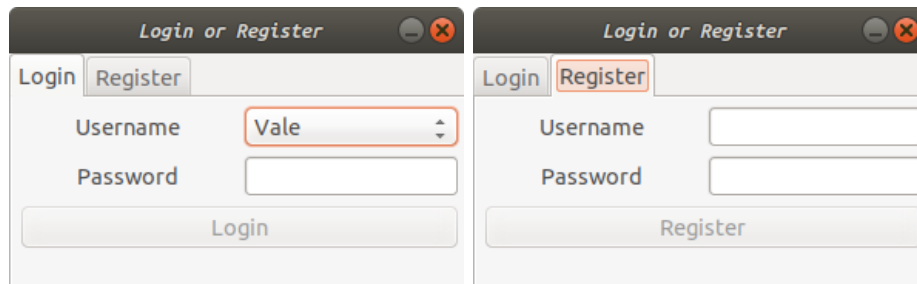


DBUser Questa classe modella l'utente del database Mongo per consentire la gestione del login (recuperando l'elenco degli utenti esistenti) e la registrazione di nuovi utenti.

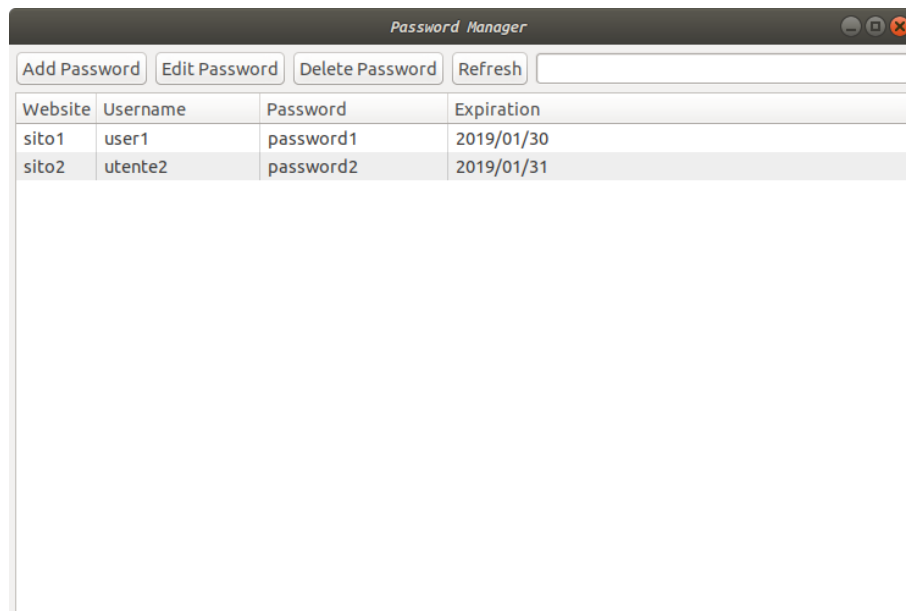
2.1.3 Package app.gui

In questo package sono presenti le classi per la gestione dell'interfaccia grafica e di tutte le funzionalità collegate.

LoginGUI Mostra la finestra iniziale per il login di utenti registrati o la registrazione di nuovi utenti.



PasswordManagerGUI Contiene l'interfaccia principale per la gestione delle password: i pulsanti con le varie funzionalità, la barra di ricerca e la tabella con l'elenco delle password salvate in cui si può modificare l'ordinamento con un click sulla colonna secondo cui si vuole effettuare l'ordinamento.



EditDialog In questa classe è presente la finestra che viene mostrata per la creazione di una nuova password o per la modifica di una password esistente.

Add a new password

Website:

Username:

Password:

Expiration:

lun	mar	mer	gio	ven	sab	dom
31	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	31	1	2	3
4	5	6	7	8	9	10

Cancel Ok

Edit an existing password

Website:

Username:

Password:

Expiration:

lun	mar	mer	gio	ven	sab	dom
31	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	31	1	2	3
4	5	6	7	8	9	10

Cancel Ok

AlertDialog Inizialmente per la gestione dei messaggi di avviso o errore era stata usata la classe `org.eclipse.swt.widgets.MessageBox` ma, visto che tale classe non era testabile con **SWTBot** a causa di un [bug aperto](#), è stata implementata una classe custom.

Labels Questa classe contiene le “costanti” per la parte grafica: label, hint, titoli, messaggi, ... In questo modo è possibile richiamare lo stesso testo in diversi punti

del programma in maniera sempre consistente evitando eventuali disallineamenti che potrebbero nascere dalla ridondanza.

2.2 Folder Test

La cartella contenente i test è composta da due package: *test.gui* e *test.db*.

2.2.1 Package test.gui

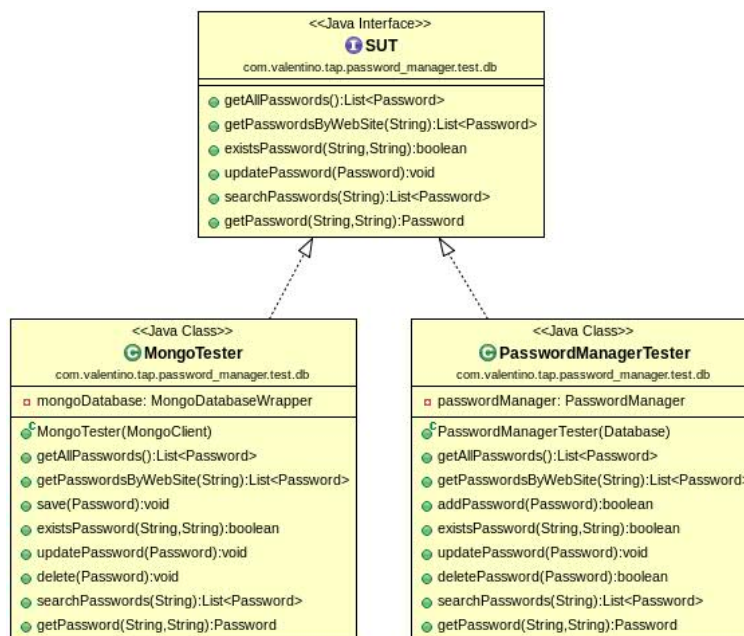
In questo package sono presenti le classi per il test dell'interfaccia grafica. Per tali test è stato utilizzato il tool ***SWTBot*** menzionato nel capitolo precedente.

LoginGUITestIT Verifica che sia correttamente mostrata la finestra con i tutti gli elementi attesi ed effettua i test per quanto riguarda l'accesso al programma: si testa il login con le corrette credenziali, il mancato accesso con credenziali errate, la corretta registrazione di un nuovo utente e la mancata registrazione in caso di utente omonimo già presente. Viene inoltre verificato che dopo aver effettuato l'accesso con successo (sia tramite login che tramite registrazione di un nuovo utente), venga correttamente mostrata la finestra principale del programma.

PasswordManagerGUITest Verifica che sia correttamente mostrata la finestra con tutti gli elementi attesi e ne testa tutte le funzionalità. Si verifica inoltre che in caso di comportamento errato da parte dell'utente vengano mostrati i corretti messaggi d'errore: per esempio se si tenta di inserire due password per lo stesso sito/utenza o si tenta di inserire una password con data di scadenza errata.

2.2.2 Package test.db

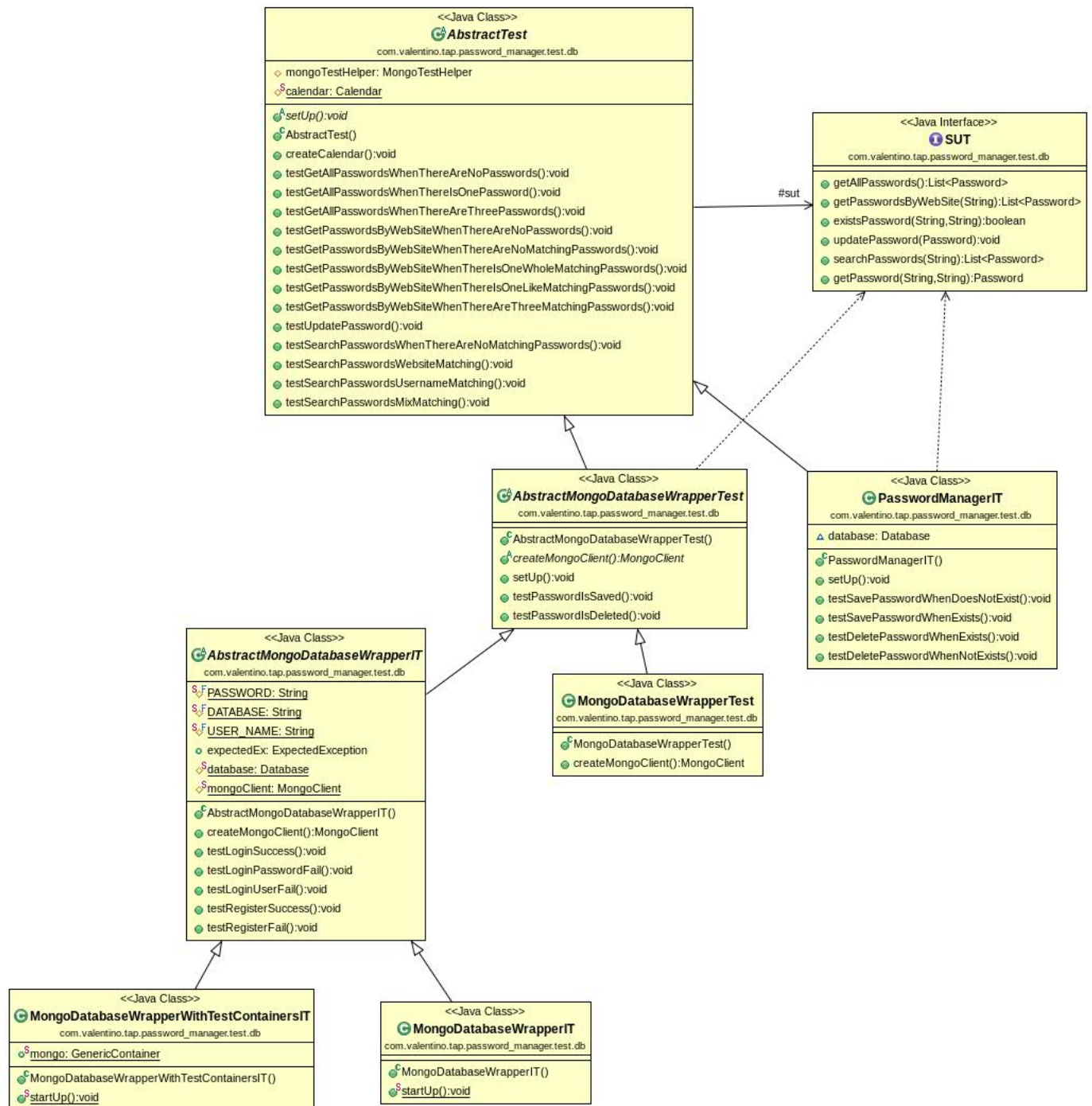
In questo package troviamo i test principali sulle funzionalità backend del programma e sull'interazione con il database. Sono stati implementati sia degli unit test che degli integration test. Alcuni test effettuati nelle varie classi risultano però molto simili o identici, per tale motivo in seguito ad alcuni refactoring, tutti i test comuni sono stati raggruppati in classi astratte. Prima di procedere vediamo quindi le dipendenze tra le varie classi appartenenti a tale package riportate in figura 2.1 a pagina 12. Come possiamo notare la classe in cui è presente la maggior implementazione è *AbstractTest*. In ogni test si fa riferimento all'interfaccia *SUT* che rappresenta appunto il *Software Under Test*, in questo modo i test sono generici e indipendenti dalla classe che stiamo testando. Tale interfaccia è implementata dalle due classi *PasswordManagerTester* e *MongoTester*: la prima implementa le funzionalità necessarie richiamando i corrispettivi metodi della classe *PasswordManager*, la seconda i metodi della classe *MongoDatabaseWrapper*.



Il metodo `testUpdatePassword`, per esempio, sarà richiamato sia durante i test della classe *PasswordManagerIT* che durante i test delle classi che estendono la classe astratta *AbstractMongoDatabaseWrapperTest*: nel primo caso verificherà il corretto aggiornamento della password quando viene richiesto dalla classe *PasswordManager*, nel secondo caso quando viene richiesto dalla classe *MongoDatabaseWrapper*. Centralizzando i test nella classe astratta

è stato quindi possibile ridurre la quantità di codice scritta per i medesimi test e semplificare la scrittura di successivi test ad ogni aggiunta di nuove funzionalità.

Per quanto riguarda la classe *MongoDatabaseWrapper* sono stati implementati sia Unit test che Integration Test, infatti la classe astratta *AbstractMongoDatabaseWrapperTest* è estesa dalla classe *MongoDatabaseWrapperTest* (Unit Test) e dalla classe *AbstractMongoDatabaseWrapperIT* (Integration Test). Si può notare che anche quest'ultima è una classe astratta poiché gli stessi Integration Test (principalmente per scopo didattico) sono eseguiti due volte: per quanto riguarda la classe *MongoDatabaseWrapperIT* connettendosi a un'istanza *Mongo* presente in un docker avviato da *Maven*, per quanto riguarda la classe *MongoDatabaseWrapperWithTestContainersIT* lanciando un apposito docker con l'istanza di *Mongo*. Nella classe *AbstractMongoDatabaseWrapperIT* sono presenti i test sulle funzionalità di *MongoDatabaseWrapper* per quanto riguarda la gestione degli accessi: login, registrazione nuovi utenti, verifica utenti presenti, ... Tali test infatti, con lo stesso principio visto in precedenza, vengono quindi eseguiti sia sul docker lanciato da *Maven* che sul docker lanciato dalla classe *MongoDatabaseWrapperWithTestContainersIT*. Per concludere quindi possiamo osservare che le classi *MongoDatabaseWrapperTest*, *MongoDatabaseWrapperIT* e *MongoDatabaseWrapperWithTestContainersIT* contengono solo una differente creazione del MongoClient che viene utilizzato per i test.



Difficoltà riscontrate

Di seguito sono esposte le principali difficoltà incontrate:

- Inizialmente non era molto naturale la scrittura e l'esecuzione dei test prima dell'implementazione stessa ma risultava quasi una forzatura. Dopo non molto però si sono visti i vantaggi sia in termini di velocità di sviluppo (anche grazie alle molteplici funzionalità offerte da Eclipse) sia in termini di correttezza di design e modello.
- Altra difficoltà è stata l'utilizzo del Database Mongo per la gestione degli utenti e degli accessi. Pur avendo buona padronanza (per lavoro) di SQL e PL/SQL su Oracle ho avuto qualche difficoltà a orientarmi e comprendere l'organizzazione del database Mongo, la creazione degli utenti con i relativi permessi e il recupero degli utenti esistenti.
- Le password scadute vengono evienziate dal programma tramite il colore rosso nel riepilogo principale e tramite una pop-up con il relativo messaggio come possiamo vedere dalle seguenti immagini.



Non è stato però possibile testare la corretta colorazione delle righe relative alle password scadute con *SWTBot* in quanto gli elementi delle tabelle non vengono gestiti, a differenza di altri Widget, come istanze della classe `org.eclipse.swt.widgets.Control`, di conseguenza non viene correttamente recuperato da *SWTBot* il colore. Per tale motivo i test relativi al colore delle righe della tabella all'interno della classe *PasswordManagerGUI* sono stati commentati ed è stato riportato il riferimento alla [nota](#).

- Infine, anche se abbastanza normale, altre difficoltà sono nate dall'utilizzo di strumenti nuovi da dover imparare. Inizialmente per esempio ho perso diverso tempo per l'aggiunta di *SWTBot* alle dipendenze visto che non è presente su *Maven Central* o per comprendere l'organizzazione e la gestione del file *pom*.

Dettagli

Il progetto è presente su [Github](#), nel file *README* sono presenti anche i badge con i collegamenti verso [Travis](#), [Coveralls](#) e [Sonarcloud](#).

È possibile effettuare la build anche da terminale usando il comando

```
mvn -f com.valentino.tap.password_manager/pom.xml clean verify -Pjacoco  
↪ coveralls:report sonar:sonar
```

In questo modo saranno sarà effettuata la build con tutti i test; avendo attivato l'apposito profilo sarà inoltre effettuata la code coverage tramite **JaCoCo** e il report sarà inviato a **Coveralls**. Al termine della build, l'applicazione viene inclusa in un apposito contenitore ed è stato aggiunto un docker-compose file per gestire l'avvio sia del docker con l'applicazione che del docker con l'istanza di **Mongo** necessaria. Per l'avvio di tali docker è stato predisposto lo script *password_manager.sh*.

Da riga di comando è inoltre possibile effettuare i mutation test con il comando

```
mvn -f com.valentino.tap.password_manager/pom.xml docker:start  
↪ org.pitest:pitest-maven:mutationCoverage docker:stop
```

Questo consente l'avvio del docker con **Mongo** prima dei mutation test e la chiusura di tale docker al termine.

Possibili Estensioni

Elenchiamo di seguito alcune delle possibili estensioni:

- Sviluppo di un'app per smartphone sincronizzabile con il software presente sul PC.
- Possibilità di sincronizzare le password su un server in modo da avere sempre un backup.
- Possibilità di importare/esportare il contenuto del Database tramite dump protetto.
- Possibilità di testare la robustezza delle proprie password con appositi tool che effettuano attacchi brute force, attacchi al dizionario o simili.
- Possibilità di generare password randomiche impostando alcuni parametri (es. lunghezza massima, caratteri speciali, ...)
- Aggiunta del pulsante che copia la password automaticamente in modo da averla subito a disposizione per incollarla senza doverla selezionare.