N2L - Need 2 Log

Programma di utility per lo storage di credenziali personali.

Relazione del progetto di P.O.I.S.

Davide Quattrocchi
MATRICOLA: 273654
d.quattrocchi@campus.uniurb.it

A.A.:2016/17

Professore: Edoardo Bontà

Università degli Studi di Urbino "Carlo Bo"

Questo documento è stato preparato con LaTeX.

Indice

Codici sorgente

Capitolo 1

Specifica del problema

Si vuole realizzare un sistema che dia la possibilità di salvare in modo sicuro gli account posseduti dall'utilizzatore in modo tale che egli non debba ricordare ogni singola credenziale, username o password personale.
Per poter rispettare suddetta specifica, il sistema dovrà possedere tali caratteristiche e comportamenti:

- Tutti i dati sensibili forniti dall'utente dovranno essere salvati all'interno di un database criptato.
- L'accesso al sistema e ai dati deve essere permesso solo dopo l'immissione e la verifica di una password, conservata in modo sicuro.
- Il sistema deve dare la possibilità di modificare ed eliminare i dati presenti nel database.
- Il sistema deve permettere di cambiare le impostazioni dell'applicazione, tra cui anche la password di accesso.

Capitolo 2

Specifica dei requisiti

Osservando la specifica del problema risulta evidente quanto l'applicazione debba possedere efficienti sistemi di sicurezza, per preservare i dati dell'utilizzatore da un eventuale attacco informatico sui dati.

In particolare, si è deciso di proteggere le informazioni che l'utente affida al sistema utilizzando questi metodi:

- La password di accesso al programma non verrà conservata in chiaro tra le impostazioni della applicazione, ma ne verrà conservato l'hash solo in fase di esecuzione, calcolato tramite una apposita funzione crittografica di hash (i.e.: SHA-256).
- L'hash in questione verrà utilizzato come chiave di accesso e de/criptazione del database: così facendo, ci si affiderà al sistema di gestione della base di dati per accedere alle informazioni sicuramente.
- L'applicazione può permettere di cambiare la password di accesso al sistema da parte dell'utente, così che l'utilizzatore riduca le probabilità che un attaccante riesca a sorpassare i sistemi di sicurezza tramite attacchi basati su associazioni e confronti (bruteforcing, dictionary attack, rainbow tables attack, ecc).

2.1 Organizzazione dell'applicativo

Si è deciso di separare, tramite la creazione di una DLL, il sistema di gestione dei dati dalla interfaccia utente, in modo tale che il sistema sia concettualmente più ordinato e che siano più facilmente implementabili futuri miglioramenti e aggiornamenti del programma.
Si è dunque deciso di creare uno strato di comunicazione tra il DBMS scelto per lo storage dei dati e la applicazione che implementa la DLL, in modo tale da evitare che, con successive modifiche, le differenti versioni rischino di effettuare dei *breaking changes*.

2.2 Diagramma e descrizione dei casi d'uso

Di seguito vengono riportati i casi d'uso della applicazione basata su GUI, implementata tramite Windows Forms, per comprendere al meglio le funzionalità dell'applicativo.
Vengono riportati anche tutti i template dei casi d'uso sopra descritti.

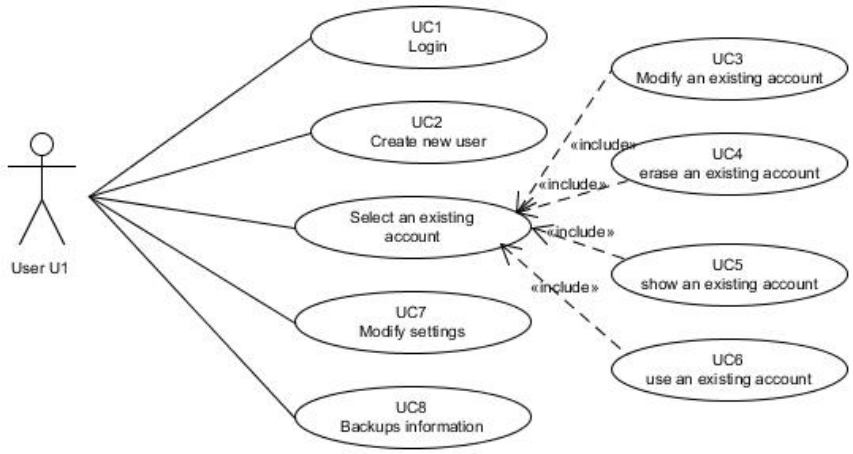

Figura 2.1: Diagramma dei casi d'uso tramite GUI da parte dell'utente

Poiché il processo di sviluppo utilizzato *(sezione 4.1)* prevede lo sviluppo orientato ai test, non è necessario effettuare test black-box sulla libreria (durante lo sviluppo vengono creati test white-box che coprono tutti i possibili flussi d'esecuzione) ed è quindi superfluo descrivere i template dei casi d'uso della libreria.

Use Case: Login
ID: UC1
ACTOR: U1 User
Preconditions: * U1 is not logged in yet
Basic course of events: 1. U1 insert passphrase 2. The system verifies the password and, if, correct, decrypt the database and show the element wich is in it
Postconditions: * U1 is logged into the system. * The database was decrypted and the data is accessible * U1 is in the home page.
Alternative paths: * if 2. fails, system shows an error message, then goes back to the login page.

Figura 2.2: Template relativo al caso d'uso UC1

Use Case: Create new account
ID: UC2
ACTOR: U1 User
Preconditions: * U1 was logged in
Basic course of events: 1. U1 select to create a new account. 2. U1 select the type of account. 3. U1 insert all the data related to the new account. 4. The system verify the correctiveness of the data inserted. 5. The system update the database.
Postconditions: * The system have a new element in the database. * U1 is at the home page
Alternative paths: * U1 decide to erase the current action, the system ask a confirm, so U1 was sended to the home page. * If 4 fails, the system shows an error message and the user was invited to insert the correct data. * if 5 fails, the system shows an error message, user was invited to check settings and then to reinsert data.

Figura 2.3: Template relativo al caso d'uso UC2

Use Case: Modify an existing account
ID: UC3
ACTOR: U1 User
Preconditions: * U1 was logged in * Database has at least one account stored.
Basic course of events: 1. U1 select to modify an existing account. 2. U1 select the account from a list. 3. The system request the new data and U1 insert it. 4. The system verify the correctiveness of what inserted. 5. The system shows an alert message requesting a confirm. 6. The system update the data. 7. The system update the database.
Postconditions: * Database was changed. * U1 is in the home page.
Alternative paths: * U1 decide to erase the current action, the system ask a confirm, so U1 was sended to the home page. * If 4 fails, the system shows an error message and the user was invited to insert the correct data. * if 7 fails, the system shows an error message, user was invited to check settings and then to reinsert data.

Figura 2.4: Template relativo al caso d'uso UC3

Use Case: Erase an existing account
ID: UC4
ACTOR: U1 User
Preconditions: * U1 was logged in * Database has at least one account stored.
Basic course of events: 1. U1 select to erase an existing account. 2. U1 select the account from a list. 3. The system request a confirm after proceed. 4. The system update the data. 5. The system update the database.
Postconditions: * Database was changed. * U1 is in the home page.
Alternative paths: * If U1 decide to erase the current action, the system ask a confirm, so U1 was sended to the home page. * if 5 fails, the system shows an error message, user was invited to check settings and then to retry.

Figura 2.5: Template relativo al caso d'uso UC4

Use Case: Show an existing account
ID: UC5
ACTOR: U1 User
Preconditions: * U1 was logged in * Database has at least one account stored.
Basic course of events: 1. U1 select to show an existing account. 2. U1 select the account from a list. 3. The system send a query to the db and show the requested data.
Postconditions: * U1 has the information searched.
Alternative paths: * If U1 decide to erase the current action, the system ask a confirm, so U1 was sended to the home page. * if 3 fails, the system shows an error message, user was invited to check settings and then to retry.

Figura 2.6: Template relativo al caso d'uso UC5

Use Case: Use an existing account
ID: UC6
ACTOR: U1 User
Preconditions: * U1 was logged in * Database has at least one account stored.
Basic course of events: 1. U1 select to use info related to an existing account. 2. U1 select the account from a list. 3. The system send a query to the db and show the account's related data. 4. User was able to use the data. (e.g.: user was sended to the default browser and then he/she was automatically logged into the account).
Postconditions: * U1 is in the home page. * U1 is in the requested account (if function is available).
Alternative paths: * If U1 decide to erase the current action, the system ask a confirm, so U1 was sended to the home page. * if 3 fails, the system shows an error message, user was invited to check settings and then to retry. * if 4. fails but the function was available, user nedds to insert requested data manually.

Figura 2.7: Template relativo al caso d'uso UC6

Use Case: Modify settings
ID: UC7
ACTOR: U1 User
Preconditions: * U1 was logged in
Basic course of events: 1. U1 choose to modify some settings 2. U1 select what needed a modify and do it. 3. The system show an alert message and request a confirm for the action. 4. The system update the settings.
Postconditions: * Program's settings was changed. * U1 is in the home page.
Alternative paths:* * If U1 decide to erase the current action, the system ask a confirm, so U1 was sended to the home page. * if 4 fails, the system shows an error message, user was invited to check settings and then to retry.

Figura 2.8: Template relativo al caso d'uso UC7

Use Case: Backups information
ID: UC8
ACTOR: U1 User
Preconditions: * U1 was logged in * Database has at least one account stored.
Basic course of events: 1. U1 choose to do a backup of the data. 2. The system show an alert message and request a confirm for the action. 3. The system ask at U1 to choose the path and the name of the backup. 4. The system do the backup and save it in the required path.
Postconditions: * U1 is in the home page. * U1 have the requested backup.
Alternative paths:* * If U1 decide to erase the current action, the system ask a confirm, so U1 was sended to the home page. * If 3. fails, the system shows an error message, user was invited to verify the inserted data an then to retry. * if 4. fails, the system shows an error message, user was invited to check settings and then to retry.

Figura 2.9: Template relativo al caso d'uso UC8

Capitolo 3

Analisi e progettazione

3.1 Linguaggio di Programmazione

Per la realizzazione della interfaccia grafica e della libreria verrà utilizzato il linguaggio di programmazione C# (C-sharp).
Tale linguaggio è di tipo *multi paradigma*: la sua struttura e sintassi sono ispirati da vari linguaggi precedenti a questo per nascita, evolvendoli ripulendo il codice da simbolismi ed elementi decorativi.
Il linguaggio opera all'interno del Framework *.NET*, che è una suite di prodotti di tecnologia di programmazione ad oggetti.

3.2 Ambiente di sviluppo

Per lo sviluppo dell'applicazione è stato scelto di utilizzare come IDE *Visual Studio Express 2017*: sviluppato da Microsoft (esattamente come il linguaggio di programmazione selezionato), funzionante su Sistemi Operativi Microsoft Windows, supporta perfettamente il linguaggio di programmazione scelto.

3.3 Database e DBMS

Per la memorizzazione dei dati si è scelto di utilizzare SQLite, per la sua semplicità d'utilizzo, la sua persistenza in locale e per la sua gestione ottimale.
Infatti tutta la base di dati (tabelle, query, report etc..) è presente in un unico file, garantendo così la portabilità dei dati.
Quando l'applicazione viene eseguita, si controlla se esiste un database dal nome "*DBN2L.sqlite3*" e se esiste si procede ad effettuare l'accesso; in caso contrario si crea.
Il percorso del file è una sottodirectory della libreria, dove si trova cioè il file .dll necessario al funzionamento degli applicativi.
La libreria è stata progettata per implementare, a livello concettuale, qualunque altro DBMS si voglia, dando anche la possibilità di affiancarli tra loro.

3.4 GUI

Per implementare le funzionalità offerte dal sistema e per rendere l'utilizzo di tali funzioni semplici per l'utente classico, si è deciso di creare una applicazione con interfaccia grafica.
Tale GUI è stata realizzata tramite l'utilizzo delle classi delle Windows Forms del framework .NET.
Tra le classi grafiche utilizzate nel programma troviamo le seguenti:

- *Form*: Rappresenta una finestra o una finestra di dialogo che compone l'interfaccia utente di un'applicazione.
- *MessageBox*: Visualizza una finestra di messaggio, nota anche come finestra di dialogo, che presenta un messaggio all'utente.
  Si tratta di una finestra modale, che impedisce altre azioni nell'applicazione fino alla sua chiusura.
  Un oggetto *System.Windows.Forms.MessageBox* può contenere testo, pulsanti e simboli che forniscono istruzioni e informazioni all'utente.
- *SaveFileDialog*: Richiede all'utente di selezionare un percorso per salvare un file.
- *PictureBox*: Rappresenta un controllo casella di immagine di Windows per la visualizzazione di un'immagine.

Sono stati introdotti in totale 7 Form:

- *DetailedInfo*: Form utile a mostrare all'utente tutte le informazioni relative ad un account selezionato tra quelli disponibili e/o a modificarlo.
- *FormAboutBox*: Form della finestra "informazioni su" relative alla applicazione.
- *FormCreate*: Form utile a creare un nuovo account da inserire nella lista di account esistenti.
- *FormFirstAccess*: Form utile a settare le impostazioni iniziali (username e password) prima della creazione del database.
- *FormLogin*: Form utile a permettere all'utente di accedere al sistema.
- *FormMainMenu*: Form principale della applicazione, dove vengono svolte le principali azioni.
- *FormOption*: Form utile a modificare le impostazioni necessarie per il corretto funzionamento della applicazione (username e password).

3.5 Pattern usati

In fase di progettazione sono stati individuati dei problemi tipicamente ricorrenti, ai quali si è deciso di trovare soluzione tramite la implementazione dei design pattern.
Nello specifico, si è ritenuto necessario dover utilizzare i seguenti:

3.5.1 Singleton

Singleton è un pattern creazionale basato su oggetti che risolve il problema di assicurare che per una classe esista un'unica istanza accessibile con semplicità, facendo in modo che sia la classe stessa ad istanziarsi.
Singleton è applicabile quando:

- Deve esistere un'unica istanza di una classe accessibile da un riferimento noto a tutti gli utilizzatori;
- L'unica istanza deve essere estesa con sottoclassi e anch'esse devono ammettere un'unica istanza.

Per far funzionare correttamente il pattern, il costruttore della classe deve essere privato (nel caso in cui si debba verificare la prima precondizione) o protetto (nel caso in cui si debba verificare la seconda).
L'implementazione di Singleton consiste nella creazione di un metodo pubblico e statico (in genere chiamato Instance) che restituisce il riferimento all'istanza, non accessibile direttamente dall'esterno, dell'oggetto (in genere contenuta nell'attributo privato _instance), se il riferimento è nullo, invoca il costruttore prima di resituirlo.
Questo pattern sarà utilizzato all'interno della classe Core.DBConnector.

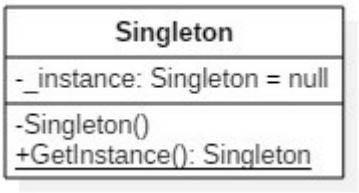

Figura 3.1: Diagramma UML del Design Pattern Singleton

3.5.2 Template Method

Template Method è un pattern comportamentale basato su classi che permette di definire la struttura di un algoritmo lasciando alle sottoclassi il compito di implementarne alcuni passi come preferiscono, personalizzando in queste il comportamento senza dover riscrivere più volte il codice in comune.
Template Method è applicabile quando:

- si vuole implementare la parte comune di un algoritmo una volta sola, lasciando alle sottoclassi il compito di implementare il comportamento che può variare;
- il comportamento comune di più classi può essere inserito in una classe a parte per evitare di scrivere più volte lo stesso codice;

Questo pattern verrà utilizzato dalle classi Core.QueryCreator e Core.SQLiteQueryCreator.

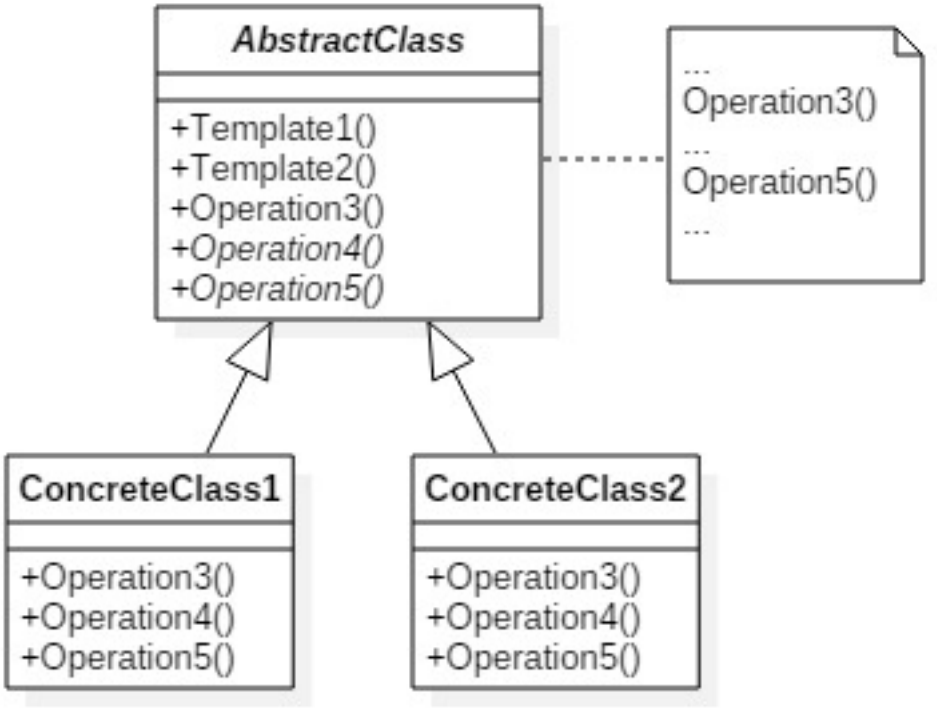

Figura 3.2: Diagramma UML del Design Pattern Template Method

3.5.3 Architettura n-tier

Una *architettura n-tier*, o *multi-tier*, è una architettura software in cui le varie funzionalità sono disaccoppiate a livello logico su vari livelli in comunicazione diretta tra loro solo tramite i livelli tra loro adiacenti mediante il modello client-server. Un esempio classico di queste architetture è quello delle applicazioni web, queste infatti hanno 3 livelli logici separati tra loro che sono:

1. logica di presentazione
2. elaborazione dei processi
3. persistenza dei dati

Il livello 1 è in comunicazione solo con il livello 2, il livello 2 è in comunicazione con il livello 1 e il livello 3, il livello 3 è in comunicazione solo con il livello 2.
Questa architettura viene utilizzata all'interno della soluzione associata a questo report, in quanto la libreria *Core* funge da intermediario tra la base di dati e un qualsiasi programma che voglia sfruttare tale libreria (in questo caso il nostro progetto con intefaccia grafica).

$$DBMS \longleftrightarrow N2L.Core \longleftrightarrow GUI$$

3.6 Classi

Di seguito verranno forniti i diagrammi UML relativi ad ogni singola classe creata all'interno di ogni progetto, divisi per questi.

3.6.1 N2L.Core

Le classi che compongono la libreria DLL.

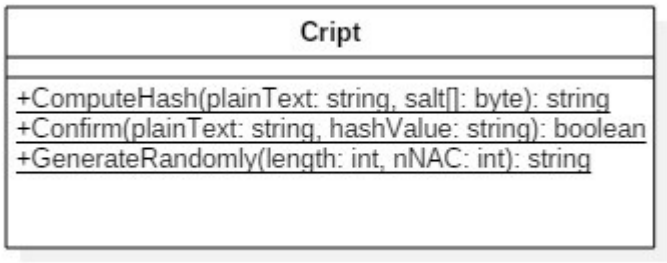

Figura 3.3: Diagramma UML della classe Cript

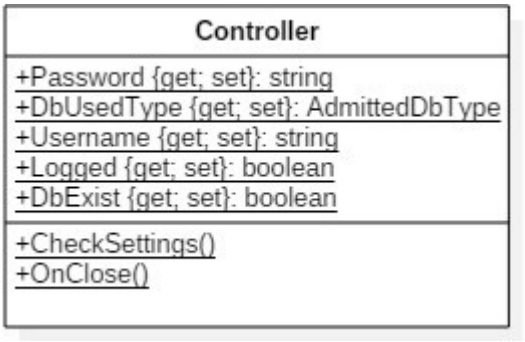

Figura 3.4: Diagramma UML della classe Controller

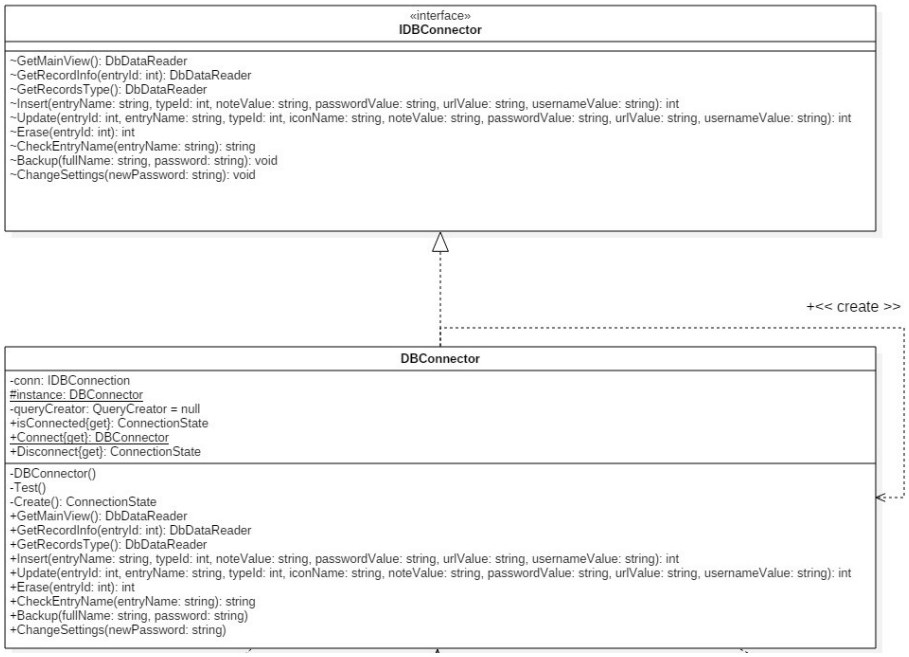

Figura 3.5: Diagramma UML della classe DBConnector e della sua interfaccia

Figura 3.6: Diagramma UML della classe QueryCreator, della sua interfaccia e di una sua implementazione

3.6.2 N2L.Core.DataStruct

Le classi che compongono le strutture dati usate nella libreria.

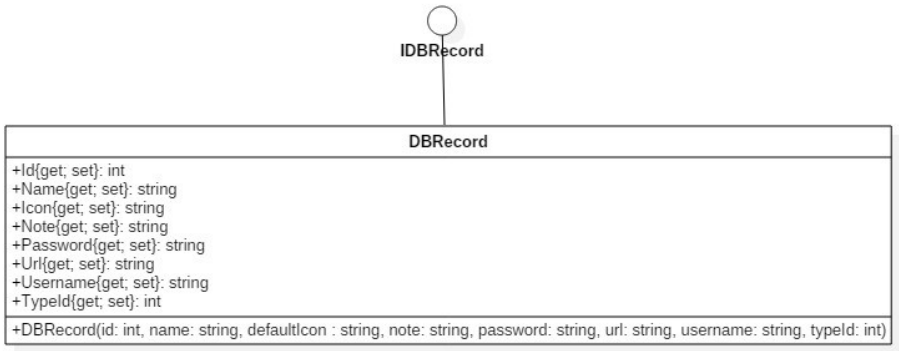

Figura 3.7: Diagramma UML della classe DBRecord e della sua interfaccia

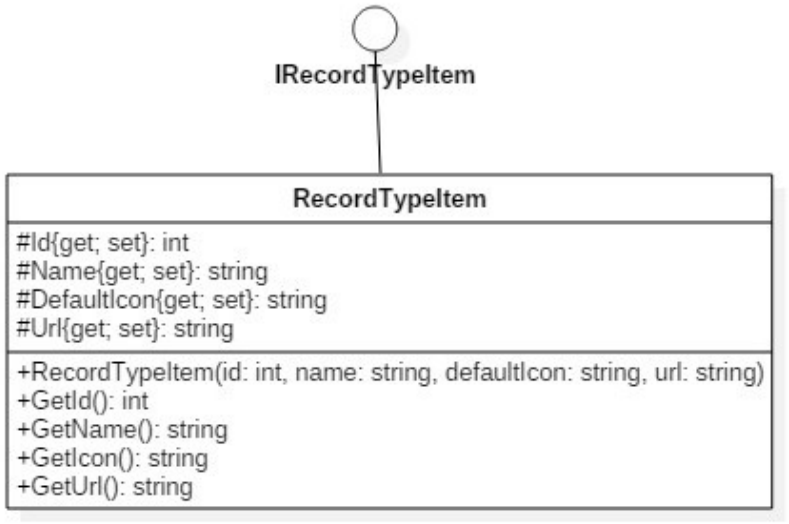

Figura 3.8: Diagramma UML della classe RecordTypeItem e della sua interfaccia

3.6.3 N2L.DesktopFormsGUI

Le classi che compongono l'implementazione della GUI.

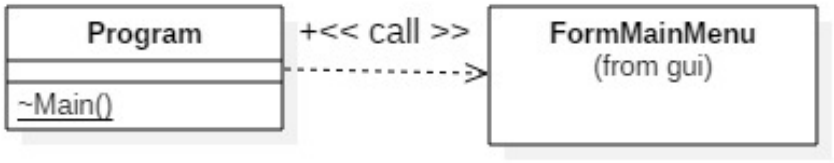

Figura 3.9: Diagramma UML della classe Program

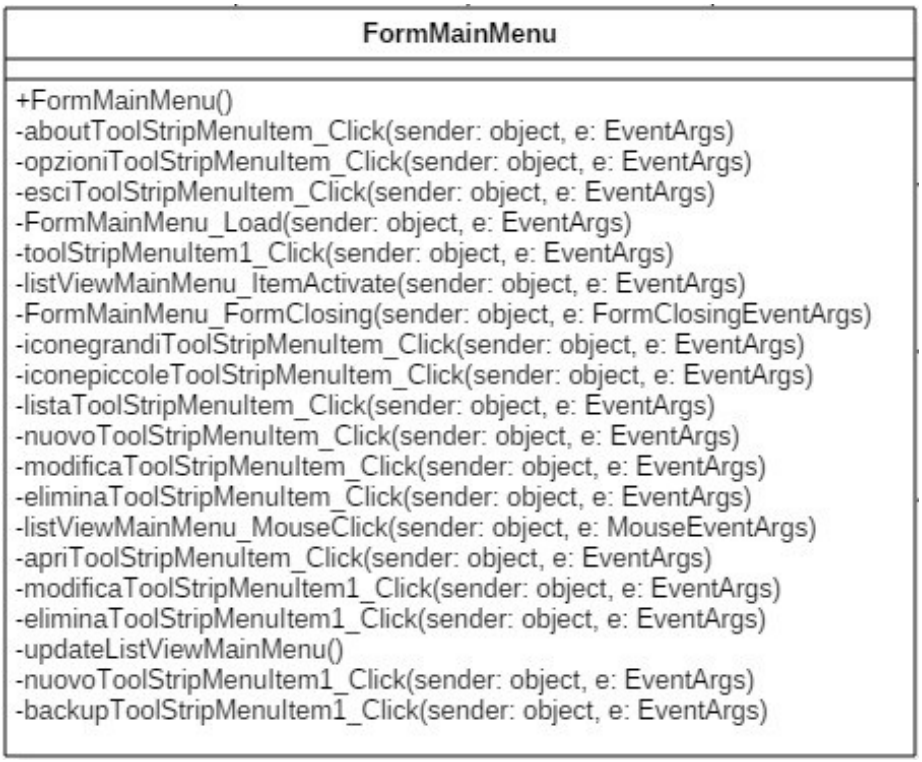

Figura 3.10: Diagramma UML della classe FormMainMenu

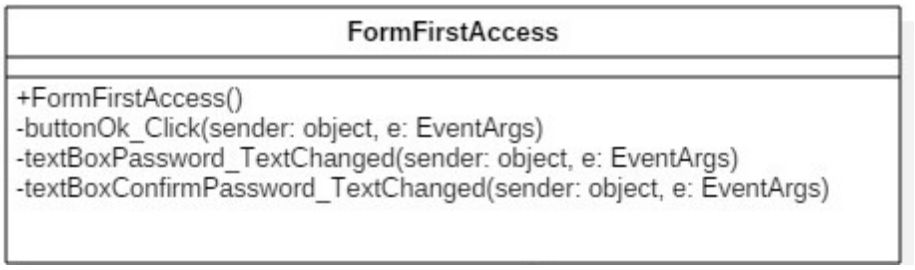

Figura 3.11: Diagramma UML della classe FormFirstAccess

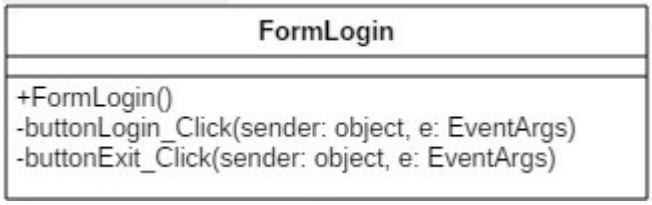

Figura 3.12: Diagramma UML della classe FormLogin

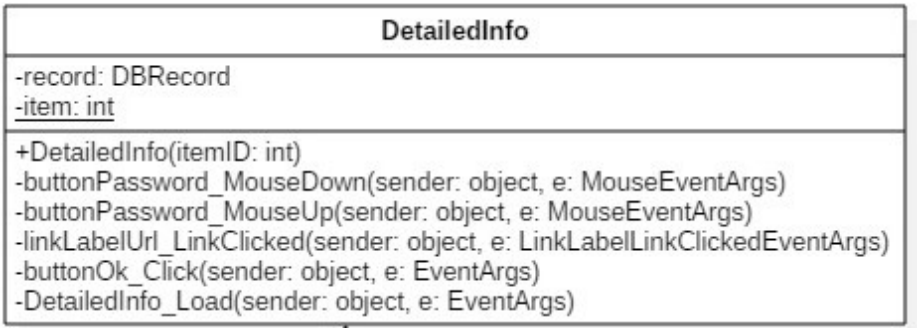

Figura 3.13: Diagramma UML della classe DetailedInfo

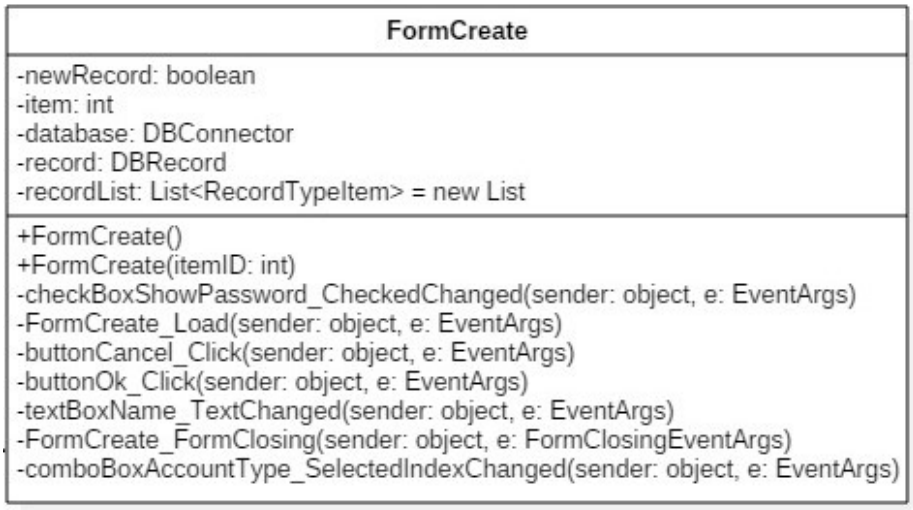

Figura 3.14: Diagramma UML della classe FormCreate

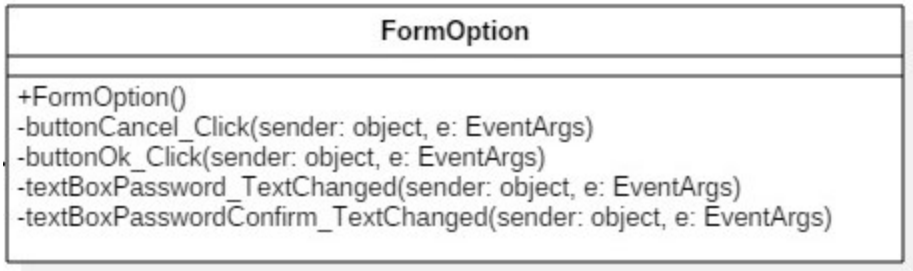

Figura 3.15: Diagramma UML della classe FormOption

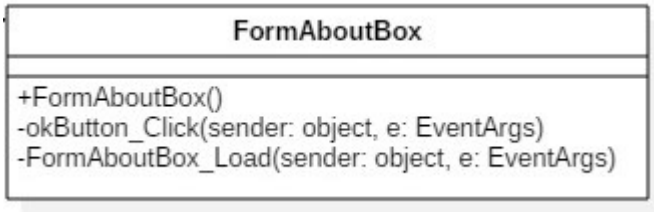

Figura 3.16: Diagramma UML della classe FormAboutBox

3.6.4 N2L.ConsoleTest

La classe che compone l'applicazione di test della DLL.

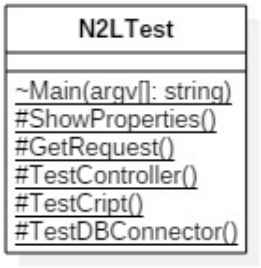

Figura 3.17: Diagramma UML della classe N2L.ConsoleTest.N2LTest

3.6.5 Diagrammi delle classi completi

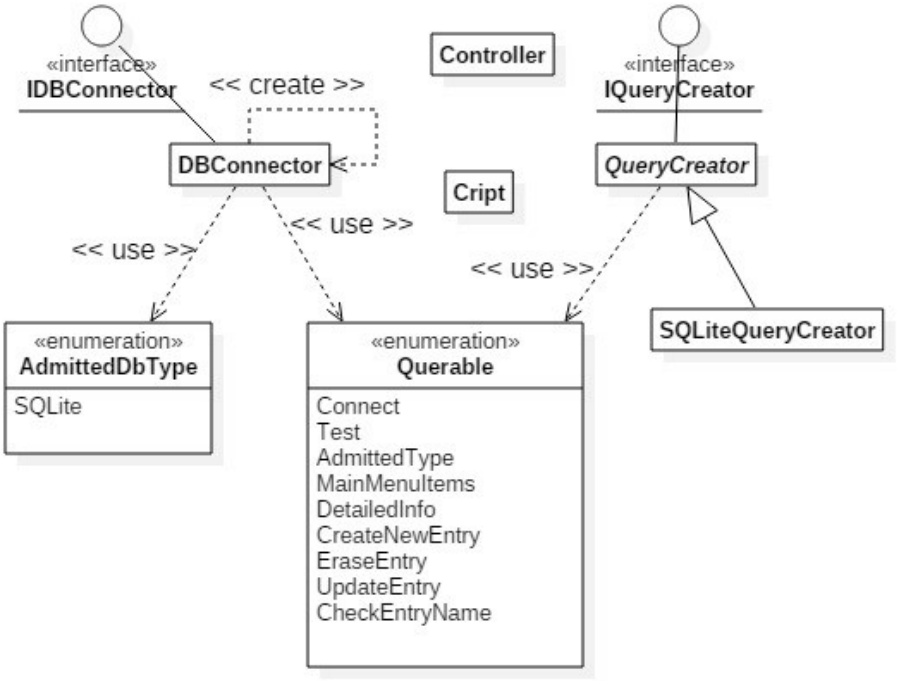

Figura 3.18: Diagramma UML del namespace Core

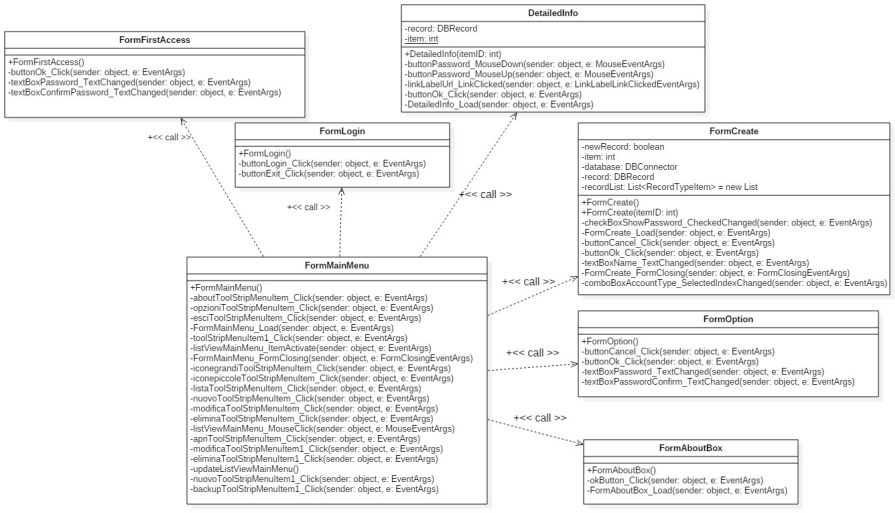

Figura 3.19: Diagramma UML del namespace DesktopFormsGUI.gui

Capitolo 4

Implementazione

4.1 Processo di sviluppo

Un processo di sviluppo è un insieme di attività (principali e secondarie) atte a ottenere una migliore pianificazione e gestione durante lo sviluppo di software.
Il processo di sviluppo utilizzato, visto che è presente un solo sviluppatore (situazione che obbliga ad escludere dunque le pratiche di pair programming e simili), è Extreme Programming (XP).
Extreme Programming è un processo di sviluppo *agile*, ossia un processo che predilige pianificazione adattativa, sviluppo evolutivo, consegne anticipate, miglioramento continuo ed è in grado di rispondere ai cambiamenti in maniera rapida e flessibile.

4.2 Classi della libreria "*Core*"

Classi presenti:

- Controller.cs
- Cript.cs
- DBConnector.cs
- IDBConnector.cs
- IQueryCreator.cs
- QueryCreator.cs
- SQLiteQueryCreator.cs
- DataStruct/IDBRecord.cs
- DataStruct/DbRecord.cs
- DataStruct/IRecordTypeItem.cs
- DataStruct/RecordTypeItem.cs

Codice 4.1: Classe Controller.cs

```
1  using System;
2  using System.IO;
3  using Core.Properties;
4  using System.Data.Common;
5
6  namespace Core
7  {
8      /// <summary>
9      /// Fornisce due metodi statici e 5 proprietà usati per
       mantenere aggiornate le impostazioni del sistema in
       apertura ed in chiusura della applicazione.
10     /// </summary>
11     public class Controller
12     {
13         /// <summary>
14         /// Proprietà usata per restituire e configurare
       l'impostazione "password".
15         /// </summary>
16         public static string Password
17         {
18             get { return Settings.Default.password; }
19             set
20             {
21                 Settings.Default.password = value;
22             }
23         }
24         /// <summary>
25         /// Proprietà usata per restituire e configurare
       l'impostazione "db_used_type".
26         /// </summary>
27         public static AdmittedDbType DbUsedType
28         {
29             get { return
       (AdmittedDbType)Settings.Default.db_used_type; }
30             set
31             {
32                 Settings.Default.db_used_type = (int)value;
33                 Settings.Default.Save();
34             }
35         }
36         /// <summary>
37         /// Proprietà usata per restituire e configurare
       l'impostazione "username".
38         /// </summary>
39         public static string Username
40         {
41             get { return Settings.Default.username; }
42             set
43             {
44                 Settings.Default.username = value;
45                 Settings.Default.Save();
46             }
47         }
```

```
48          /// <summary>
49          /// Proprietà usata per restituire e configurare
    l'impostazione "logged".
50          /// </summary>
51          public static bool Logged
52          {
53              get { return Settings.Default.logged; }
54              set
55              {
56                  Settings.Default.logged = value;
57                  Settings.Default.Save();
58              }
59          }
60          /// <summary>
61          /// Proprietà usata per restituire e configurare
    l'impostazione "db_exist".
62          /// </summary>
63          public static bool DbExist
64          {
65              get { return Settings.Default.db_exist; }
66              set
67              {
68                  Settings.Default.db_exist = value;
69                  Settings.Default.Save();
70              }
71          }
72
73          /// <summary>
74          /// verifica che il database esista e modifica il
    valore db_exist in false nel caso in cui non esistesse
75          /// </summary>
76          public static void CheckSettings()
77          {
78              switch
    ((AdmittedDbType)Settings.Default.db_used_type)
79              {
80                  case AdmittedDbType.SQLite:
81                      if
    (!File.Exists(Settings.Default.db_path_sqlite))
82                      {
83                          Settings.Default.db_exist = false;
84                      }
85                      else
86                      {
87                          Settings.Default.db_exist = true;
88                      }
89                      Settings.Default.password =
    String.Empty;
90                      break;
91                  default:
92                      break;
93              }
94              Settings.Default.logged = false;
95          }
```

```
96          /// <summary>
97          /// modifica il valore logged delle impostazioni in
    false
98          /// </summary>
99          public static void OnClose()
100         {
101             try
102             {
103                 switch
    ((AdmittedDbType)Settings.Default.db_used_type)
104                 {
105                     case AdmittedDbType.SQLite:
106                         Settings.Default.password =
    String.Empty;
107                         break;
108                     default:
109                         break;
110                 }
111                 var db = DBConnector.Connect.Disconnect;
112             }
113             catch (DbException)
114             {
115
116             }
117             finally
118             {
119                 Settings.Default.logged = false;
120                 Settings.Default.Save();
121             }
122         }
123     }
124 }
```

Codice 4.2: Classe Cript.cs

```
1  using System;
2  using System.Text;
3  using System.Security.Cryptography;
4
5
6  namespace Core
7  {
8      /// <summary>
9      /// Fornisce 3 metodi static che permettono di
       calcolare un hash, confermarlo e generare randomicamente
       una password
10     /// restituiscono null se qualcosa non è corretto.
11     /// </summary>
12     public class Cript
13     {
14         /// <summary>
15         /// Calcola il valore hash relativo ad una password
16         /// </summary>
```

```
17          /// <param name="plainText">la password in chiaro
    di cui calcolare l'hash</param>
18          /// <param name="salt">i valori dei salt bytes.
    null se non presenti</param>
19          /// <returns>l'hash relativo a plainText</returns>
20          public static string ComputeHash(string plainText,
    byte[] salt)
21          {
22              byte[] result = null;
23
24              if (!String.IsNullOrEmpty(plainText))
25              {
26                  byte[] SaltBytes = null;
27                  int SaltLength = 8;//16;
28
29                  if (salt != null)
30                  {
31                      SaltBytes = salt;
32                  }
33                  else
34                  {
35                      SaltBytes = new byte[SaltLength];
36                      SaltBytes =
    Convert.FromBase64String("1Amas4lT");
37                  }
38                  byte[] plainData =
    ASCIIEncoding.UTF8.GetBytes(plainText);
39                  byte[] plainDataAndSalt = new
    byte[plainData.Length + SaltBytes.Length];
40
41                  for (int i = 0; i < plainData.Length; i++)
42                      plainDataAndSalt[i] = plainData[i];
43                  for (int i = 0; i < SaltBytes.Length; i++)
44                      plainDataAndSalt[plainData.Length + i]
    = SaltBytes[i];
45                  byte[] hashValue = null;
46                  SHA256Managed sha = new SHA256Managed();
47                  hashValue =
    sha.ComputeHash(plainDataAndSalt);
48                  sha.Dispose();
49                  result = new byte[hashValue.Length +
    SaltBytes.Length];
50                  for (int i = 0; i < hashValue.Length; i++)
51                      result[i] = hashValue[i];
52                  for (int i = 0; i < SaltBytes.Length; i++)
53                      result[hashValue.Length + i] =
    SaltBytes[i];
54              }
55              return
    (String.IsNullOrEmpty(plainText))?"":Convert.ToBase64String(result);
56          }
57          /// <summary>
58          /// Controlla se un dato hash è corrispondente ad
    una data password
```

```
59          /// </summary>
60          /// <param name="plainText">la password da
    verificare</param>
61          /// <param name="hashValue">l'hash da
    confrontare</param>
62          /// <returns>true se combaciano. false
    altrimenti.</returns>
63          public static bool Confirm(string plainText, string
    hashValue)
64          {
65              bool retVal = false;
66              if (!String.IsNullOrEmpty(hashValue))
67              {
68                  byte[] hashBytes =
    Convert.FromBase64String(hashValue);
69                  int hashSize = 32;
70                  byte[] saltBytes = new
    byte[hashBytes.Length - hashSize];
71                  for (int i = 0; i < saltBytes.Length; i++)
72                      saltBytes[i] = hashBytes[hashSize + i];
73                  string newHash = ComputeHash(plainText,
    saltBytes);
74                  retVal = String.Equals(newHash, hashValue);
75              }
76              return retVal;
77          }
78          /// <summary>
79          /// Metodo utile a creare una password
    peudorandomicamente
80          /// </summary>
81          /// <param name="length">lunghezza della password
    desiderata</param>
82          /// <param name="nNAC">numero di caratteri non
    alfanumerici che è possibile immettere</param>
83          /// <returns>password generata</returns>
84          public static string GenerateRandomly(int length,
    int nNAC)
85          {
86              const string ANC =
    "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789";
87              const string NANC =
    "!@#$%^&*()_-+=[{]};:<>|./?";
88              Random r = new Random();
89              char[] result = new char[length];
90              char[] tmpNanc = null;
91              char[] tmpAnc = null;
92
93              if (nNAC != 0)
94              {
95                  tmpNanc = new char[nNAC];
96                  for (int i = 0; i < tmpNanc.Length; i++)
97                      tmpNanc[i] = NANC[r.Next() %
    NANC.Length];
98              }
```

```
99              if (length != nNAC)
100             {
101                 tmpAnc = new char[length - nNAC];
102                 for (int i = 0; i < tmpAnc.Length; i++)
103                     tmpAnc[i] = ANC[r.Next() % ANC.Length];
104             }
105
106             for (int i = 0, rc; i < length; i++)
107             {
108                 if (tmpNanc == null && tmpAnc != null)
109                 {
110                     int ind = r.Next() % tmpAnc.Length;
111                     result[i] = tmpAnc[ind];
112                     string tmp = new
        string(tmpAnc).Remove(ind, 1);
113                     tmpAnc = null;
114                     tmp.CopyTo(0, (tmpAnc = new
        char[tmp.Length]), 0, tmp.Length);
115                 }
116                 else if (tmpNanc != null && tmpAnc == null)
117                 {
118                     int ind = r.Next() % tmpNanc.Length;
119                     result[i] = tmpNanc[ind];
120                     string tmp = new
        string(tmpNanc).Remove(ind, 1);
121                     tmpNanc = null;
122                     tmp.CopyTo(0, (tmpNanc = new
        char[tmp.Length]), 0, tmp.Length);
123                 }
124                 else if (tmpNanc != null && tmpAnc != null)
125                 {
126                     rc = r.Next(0, length);
127                     if (tmpAnc.Length == 0)
128                     { //NON ALPHANUMERIC CHARACTER
129                         int ind = r.Next() % tmpNanc.Length;
130                         result[i] = tmpNanc[ind];
131                         string tmp = new
        string(tmpNanc).Remove(ind, 1);
132                         tmpNanc = null;
133                         tmp.CopyTo(0, (tmpNanc = new
        char[tmp.Length]), 0, tmp.Length);
134                     }
135                     else if (tmpNanc.Length == 0)
136                     { //ALPHANUMERIC CHARACTER
137                         int ind = r.Next() % tmpAnc.Length;
138                         result[i] = tmpAnc[ind];
139                         string tmp = new
        string(tmpAnc).Remove(ind, 1);
140                         tmpAnc = null;
141                         tmp.CopyTo(0, (tmpAnc = new
        char[tmp.Length]), 0, tmp.Length);
142                     }
143                     else if (tmpAnc.Length != 0 &&
        tmpNanc.Length != 0)
```

```
144                     {
145                         if (rc % 2 != 0)
146                         { //NON ALPHANUMERIC CHARACTER
147                             int ind = r.Next() %
    tmpNanc.Length;
148                             result[i] = tmpNanc[ind];
149                             string tmp = new
    string(tmpNanc).Remove(ind, 1);
150                             tmpNanc = null;
151                             tmp.CopyTo(0, (tmpNanc = new
    char[tmp.Length]), 0, tmp.Length);
152                         }
153                         else
154                         { //ALPHANUMERIC CHARACTER
155                             int ind = r.Next() %
    tmpAnc.Length;
156                             result[i] = tmpAnc[ind];
157                             string tmp = new
    string(tmpAnc).Remove(ind, 1);
158                             tmpAnc = null;
159                             tmp.CopyTo(0, (tmpAnc = new
    char[tmp.Length]), 0, tmp.Length);
160                         }
161                     }
162                 }
163                 else
164                 {
165                     return null;
166                 }
167             }
168             return new string(result);
169         }
170     }
171 }
```

Codice 4.3: Interfaccia IDBConnector.cs

```
1  using System.Data.Common;
2
3  namespace Core
4  {
5      /// <summary>
6      /// Enumerato utile a definire in modo fisso quale tipo
       di database è possibile utilizzare
7      /// </summary>
8      public enum AdmittedDbType
9      {
10         SQLite = 1
11     }
12     /// <summary>
13     /// Interfaccia che rappresenta le operazioni che
       devono poter essere eseguite
14     /// sull'origine dati della applicazione.
15     /// </summary>
```

```
16      internal interface IDBConnector
17      {
18          /// <summary>
19          /// Rappresenta un metodo utile ad ottenere una
        istanza di System.Data.Common.DbDataReader
20          /// contenente tutti i record presenti nel database
        locale con le loro informazioni generali
21          /// </summary>
22          /// <returns>Ritorna un oggetto
        System.Data.Common.DbDataReader</returns>
23          DbDataReader GetMainView();
24          /// <summary>
25          /// Rappresenta un metodo utile ad ottenere una
        istanza di tipo System.Data.Common.DbDataReader
26          /// contenente tutte le informazioni presenti nel
        database locale relative al record indicato
27          /// </summary>
28          /// <param name="entryId">identificativo del record
        nel database</param>
29          /// <returns>Ritorna un oggetto
        System.Data.Common.DbDataReader</returns>
30          DbDataReader GetRecordInfo(int entryId);
31          /// <summary>
32          /// Rappresenta un metodo utile ad ottenere dal db
        tutti i tipi predefiniti già esistenti
33          /// e preconfigurati leggibili tramite una istanza
        di System.Data.Common.DbDataReader
34          /// </summary>
35          /// <returns>Ritorna un oggetto
        System.Data.Common.DbDataReader</returns>
36          DbDataReader GetRecordsType();
37
38          /// <summary>
39          /// Rappresenta un metodo utile a creare un nuovo
        record all'interno del database
40          /// </summary>
41          /// <param name="entryName">nome del record</param>
42          /// <param name="typeId">tipologia del
        record</param>
43          /// <param name="noteValue">note aggiuntive</param>
44          /// <param name="passwordValue">password del
        record</param>
45          /// <param name="urlValue">url del sito associato
        al record</param>
46          /// <param name="usernameValue">username associata
        al record</param>
47          /// <returns>numero di righe inserite a seguito del
        comando</returns>
48          int Insert(string entryName, int typeId, string
        noteValue, string passwordValue, string urlValue, string
        usernameValue);
49          /// <summary>
50          /// Rappresenta un metodo utile ad
        aggiornare/modificare un record già esistente nel
```

```
        database
51          /// </summary>
52          /// <param name="entryId">id del record all'interno
        del database</param>
53          /// <param name="entryName">nome del record</param>
54          /// <param name="typeId">Id del tipo predefinito
        associato al record</param>
55          /// <param name="iconName">nome dell'icona
        associata al record</param>
56          /// <param name="noteValue">note aggiuntive</param>
57          /// <param name="passwordValue">password del
        record</param>
58          /// <param name="urlValue">url del sito associato
        al record</param>
59          /// <param name="usernameValue">username associato
        al record</param>
60          /// <returns>numero di righe modificate a seguito
        del comando</returns>
61          int Update(int entryId, string entryName, int
        typeId, string iconName, string noteValue, string
        passwordValue, string urlValue, string usernameValue);
62          /// <summary>
63          /// Rappresenta un metodo utile a cancellare tutti
        gli elementi associati ad un record dal database.
64          /// </summary>
65          /// <param name="entryId">Id del record presente
        nel database</param>
66          /// <returns>numero di righe eliminate a seguito
        del comando</returns>
67          int Erase(int entryId);
68          /// <summary>
69          /// Rappresenta un metodo utile a verificare che il
        nome associato
70          /// ad un record che si sta per inserire sia univoco
71          /// </summary>
72          /// <param name="entryName">nome del record</param>
73          /// <returns>nome del record eventualmente
        modificato per rispettare l'univocità</returns>
74          string CheckEntryName(string entryName);
75          /// <summary>
76          /// Rappresenta un metodo utile a creare una copia
        di backup del database
77          /// </summary>
78          /// <param name="fullName"> nome completo di path
        del database di backup</param>
79          /// <param name="password"> password che si vuole
        settare per proteggere il nuovo db</param>
80          void Backup(string fullName, string password);
81          /// <summary>
82          /// Rappresenta un metodo utile a
        modificare/assegnare una nuova password a protezione del
        database locale.
83          /// </summary>
84          /// <param name="newPassword"> nuova
```

```
    password</param>
85          void ChangeSettings(string newPassword);
86      }
87  }
```

Codice 4.4: Classe DBConnector.cs

```
1   using System;
2   using System.IO;
3   using System.Data;
4   using System.Data.Common;
5   using System.Data.SQLite;
6   using Core.Properties;
7
8   namespace Core
9   {
10      /// <summary>
11      /// Rappresenta la connessione all'origine dati
        dell'applicazione con SQLite,
12      /// effettuabile tramite una sola possibile istanza
        dell'oggetto.
13      /// L'unicità della connessione è assicurata tramite il
        pattern Singleton.
14      /// </summary>
15      public class DBConnector : IDBConnector
16      {
17          /// <summary>
18          /// Rappresenta l'istanza reale di collegamento al
        database tramite l'interfaccia DbConnection
19          /// </summary>
20          private IDbConnection conn;
21          /// <summary>
22          /// Rappresenta l'unica istanza di DBConnector
        creabile
23          /// </summary>
24          protected static DBConnector instance;
25          /// <summary>
26          /// Rappresenta l'istanza del creatore di stringhe
        di query da presentare al database tramite la istanza
        conn
27          /// </summary>
28          private QueryCreator queryCreator = null;
29
30          /// <summary>
31          /// Costruttore della classe DBConnector.
32          /// </summary>
33          private DBConnector()
34          {
35              switch
        ((AdmittedDbType)Settings.Default.db_used_type)
36              {
37                  case (AdmittedDbType.SQLite):
38                      queryCreator = new SQLiteQueryCreator();
39                      break;
```

```
40                  default:
41                      queryCreator = null;
42                      break;
43              }
44          }
45
46          /// <summary>
47          /// Descrive lo stato corrente della connessione
    all'origine dati dell'applicazione
48          /// </summary>
49          public ConnectionState isConnected
50          {
51              get
52              {
53                  return ((instance.conn != null ) ?
54                      instance.conn.State :
55                      ConnectionState.Closed);
56              }
57          }
58
59          /// <summary>
60          /// Fornisce accesso alla connessione alla origine
    dati della applicazione
61          /// </summary>
62          public static DBConnector Connect
63          {
64              get
65              {
66                  try
67                  {
68                      if (instance == null)
69                      {
70                          instance = new DBConnector();
71                          instance.conn = new
    SQLiteConnection(instance.queryCreator.Create(Querable.Connect));
72                          switch
    ((AdmittedDbType)Settings.Default.db_used_type)
73                          {
74                              case (AdmittedDbType.SQLite):
75                              default:
76                                  switch
    (File.Exists(Settings.Default.db_path_sqlite))
77                                  {
78                                      case true:
79
    instance.conn.Open();
80                                          break;
81                                      default:
82                                          instance.Create();
83                                          break;
84                                  }
85                                  break;
86                          }
87                          instance.Test();
```

```
88                      }
89                      if (instance.isConnected !=
    ConnectionState.Open)
90                      {
91                          instance.conn.Open();
92                      }
93                  }
94                  catch (DbException dbe)
95                  {
96                      ConnectionState a = instance.Disconnect;
97                      throw dbe;
98                  }
99                  return instance;
100             }
101         }
102         /// <summary>
103         /// Permette la disconnessione dalla origine data
    della applicazione, chiudendo la comunicazione in modo
    sicuro
104         /// </summary>
105         public ConnectionState Disconnect
106         {
107             get
108             {
109                 try
110                 {
111                     instance.conn.Close();
112                     var toReturn = instance.isConnected;
113                     instance.conn.Dispose();
114                     instance.conn = null;
115                     instance = null;
116                     return toReturn;
117                 }
118                 catch (DbException dbe)
119                 {
120                     throw dbe;
121                 }
122             }
123         }
124
125         /// <summary>
126         /// Verifica che i parametri inseriti (per esempio
    la password) siano corretti e che la connessione sia
    apribile.
127         /// Una System.Data.SQLite.DbException viene
    prodotta se degli errori sono riscontrati.
128         /// </summary>
129         private void Test()
130         {
131             try
132             {
133                 switch
    ((AdmittedDbType)Settings.Default.db_used_type)
134                 {
```

```
135                     case (AdmittedDbType.SQLite):
136                         var a = new
    SQLiteCommand(queryCreator.Create(Querable.Test),
    (SQLiteConnection)conn);
137                         a.VerifyOnly();
138                         break;
139                     default:
140                         break;
141                 }
142 
143             }
144             catch (DbException dbe)
145             {
146                 throw dbe;
147             }
148         }
149         /// <summary>
150         /// Se non è ancora stato creato il database,
    provvede a creare il file usando uno script sql presente
151         /// nella cartella Resources
152         /// </summary>
153         /// <returns>Lo stato della connessione</returns>
154         private ConnectionState Create()
155         {
156             try
157             {
158                 switch
    ((AdmittedDbType)Settings.Default.db_used_type)
159                 {
160                     case (AdmittedDbType.SQLite):
161 
    SQLiteConnection.CreateFile(Settings.Default.db_path_sqlite);
162 
    ((SQLiteConnection)instance.conn).Open();
163                         SQLiteCommand query = new
    SQLiteCommand(File.ReadAllText(Settings.Default.db_scriptpath_sqlite),
    (SQLiteConnection)instance.conn);
164                         query.ExecuteNonQuery();
165 
    ((SQLiteConnection)instance.conn).ChangePassword(Settings.Default.password);
166                         break;
167                     default:
168                         break;
169                 }
170             }
171             catch (DbException dbe)
172             {
173                 throw dbe;
174             }
175             return conn.State;
176         }
177 
178         /// <summary>
179         /// Metodo utile ad ottenere una istanza di tipo
```

```
        System.Data.SQLite.SQLiteDataReader contenente tutti
180         /// i record presenti nel database locale con le
        loro informazioni generali, leggibile tramite
181         /// una istanza di System.Data.Common.DbDataReader
182         /// </summary>
183         /// <returns>Ritorna un oggetto
        System.Data.Common.DbDataReader</returns>
184         public DbDataReader GetMainView()
185         {
186             DbDataReader toReturn;
187             try
188             {
189                 switch
        ((AdmittedDbType)Settings.Default.db_used_type)
190                 {
191                     case (AdmittedDbType.SQLite):
192                         toReturn = new
        SQLiteCommand(queryCreator.Create(Querable.MainMenuItems),
        (SQLiteConnection)conn).ExecuteReader();
193                         break;
194                     default:
195                         toReturn = null;
196                         break;
197                 }
198                 return toReturn;
199             }
200             catch (DbException dbe)
201             {
202                 throw dbe;
203             }
204         }
205         /// <summary>
206         /// Metodo utile ad ottenere una istanza di tipo
        System.Data.SQLite.SQLiteDataReader contenente tutte
207         /// le informazioni presenti nel database locale
        relative al record indicato
208         /// leggibile tramite una istanza di
        System.Data.Common.DbDataReader
209         /// </summary>
210         /// <param name="entryId">identificativo del record
        nel database</param>
211         /// <returns>Ritorna un oggetto
        System.Data.Common.DbDataReader</returns>
212         public DbDataReader GetRecordInfo(int entryId)
213         {
214             DbDataReader toReturn;
215
216             try
217             {
218                 switch
        ((AdmittedDbType)Settings.Default.db_used_type)
219                 {
220                     case (AdmittedDbType.SQLite):
221                         toReturn = new
```

```
        SQLiteCommand(queryCreator.Create(Querable.DetailedInfo,
        entryId), (SQLiteConnection)conn).ExecuteReader();
222                         break;
223                     default:
224                         toReturn = null;
225                         break;
226                 }
227                 return toReturn;
228             }
229             catch (DbException dbe)
230             {
231                 throw dbe;
232             }
233         }
234         /// <summary>
235         /// Metodo utile ad ottenere dal db tutti i tipi
        predefiniti già esistenti e preconfigurati
236         /// leggibili tramite una istanza di
        System.Data.Common.DbDataReader
237         /// </summary>
238         /// <returns>Ritorna un oggetto
        System.Data.Common.DbDataReader</returns>
239         public DbDataReader GetRecordsType()
240         {
241             DbDataReader toReturn;
242 
243             try
244             {
245                 switch
        ((AdmittedDbType)Settings.Default.db_used_type)
246                 {
247                     case (AdmittedDbType.SQLite):
248                         toReturn = new
        SQLiteCommand(queryCreator.Create(Querable.AdmittedType),
249 
        (SQLiteConnection)instance.conn).ExecuteReader();
250                         break;
251                     default:
252                         toReturn = null;
253                         break;
254                 }
255                 return toReturn;
256             }
257             catch (DbException dbe)
258             {
259                 throw dbe;
260             }
261         }
262 
263         /// <summary>
264         /// Metodo utile a creare un nuovo record
        all'interno del database
265         /// </summary>
266         /// <param name="entryName">nome del record</param>
```

```
267         /// <param name="typeId">tipologia del
    record</param>
268         /// <param name="noteValue">note aggiuntive</param>
269         /// <param name="passwordValue">password del
    record</param>
270         /// <param name="urlValue">url del sito associato
    al record</param>
271         /// <param name="usernameValue">username associata
    al record</param>
272         /// <returns>numero di righe inserite a seguito del
    comando</returns>
273         public int Insert(string entryName, int typeId,
    string noteValue, string passwordValue, string urlValue,
    string usernameValue)
274         {
275             DbCommand sql;
276             int toReturn;
277             try
278             {
279                 string queryString =
    queryCreator.Create(Querable.CreateNewEntry, entryName,
    typeId, noteValue, passwordValue, urlValue,
    usernameValue);
280                 switch
    ((AdmittedDbType)Settings.Default.db_used_type)
281                 {
282                     case (AdmittedDbType.SQLite):
283                         sql = new
    SQLiteCommand(queryString,
    (SQLiteConnection)instance.conn);
284                         toReturn = sql.ExecuteNonQuery();
285                         break;
286                     default:
287                         toReturn = 0;
288                         sql = null;
289                         break;
290                 }
291                 return toReturn;
292             }
293             catch (DbException dbe)
294             {
295                 throw dbe;
296             }
297         }
298         /// <summary>
299         /// Metodo utile ad aggiornare/modificare un record
    già esistente nel database
300         /// </summary>
301         /// <param name="entryId">id del record all'interno
    del database</param>
302         /// <param name="entryName">nome del record</param>
303         /// <param name="typeId">Id del tipo predefinito
    associato al record</param>
304         /// <param name="iconName">nome dell'icona
```

```
associata al record</param>
305         /// <param name="noteValue">note aggiuntive</param>
306         /// <param name="passwordValue">password del
record</param>
307         /// <param name="urlValue">url del sito associato
al record</param>
308         /// <param name="usernameValue">username associato
al record</param>
309         /// <returns>numero di righe modificate a seguito
del comando</returns>
310         public int Update(int entryId, string entryName,
int typeId, string iconName, string noteValue, string
passwordValue, string urlValue, string usernameValue)
311         {
312             DbCommand sql;
313             int toReturn;
314             try
315             {
316                 string queryString =
queryCreator.Create(Querable.UpdateEntry, entryId,
entryName, typeId, iconName, noteValue, passwordValue,
urlValue, usernameValue);
317                 switch
((AdmittedDbType)Settings.Default.db_used_type)
318                 {
319                     case (AdmittedDbType.SQLite):
320                         sql = new
SQLiteCommand(queryString,
(SQLiteConnection)instance.conn);
321                         toReturn = sql.ExecuteNonQuery();
322                         break;
323                     default:
324                         toReturn = 0;
325                         break;
326                 }
327
328                 return toReturn;
329             }
330             catch (DbException dbe)
331             {
332                 throw dbe;
333             }
334         }
335         /// <summary>
336         /// Metodo utile a cancellare tutti gli elementi
associati ad un record dal database.
337         /// </summary>
338         /// <param name="entryId">Id del record presente
nel database</param>
339         /// <returns>numero di righe eliminate a seguito
del comando</returns>
340         public int Erase(int entryId)
341         {
342             DbCommand sql;
```

```
343             int toReturn;
344 
345             try
346             {
347                 string queryString =
    queryCreator.Create(Querable.EraseEntry, entryId);
348                 switch
    ((AdmittedDbType)Settings.Default.db_used_type)
349                 {
350                     case (AdmittedDbType.SQLite):
351                         sql = new
    SQLiteCommand(queryString,
    (SQLiteConnection)instance.conn);
352                         toReturn = sql.ExecuteNonQuery();
353                         break;
354                     default:
355                         toReturn = 0;
356                         break;
357                 }
358                 return toReturn;
359             }
360             catch (DbException dbe)
361             {
362                 throw dbe;
363             }
364         }
365         /// <summary>
366         /// Metodo utile a verificare che il nome associato
    ad un record che si sta per
367         /// inserire sia univoco
368         /// </summary>
369         /// <param name="entryName">nome del record</param>
370         /// <returns>nome del record eventualmente
    modificato per rispettare l'univocità</returns>
371         public string CheckEntryName(string entryName)
372         {
373             DbCommand sql;
374             string toReturn = String.Empty;
375 
376             try
377             {
378                 string queryString =
    queryCreator.Create(Querable.CheckEntryName, entryName);
379                 switch
    ((AdmittedDbType)Settings.Default.db_used_type)
380                 {
381                     case (AdmittedDbType.SQLite):
382                         sql = new
    SQLiteCommand(queryString, (SQLiteConnection)conn);
383                         using (SQLiteDataReader data =
    (SQLiteDataReader)sql.ExecuteReader())
384                         {
385                             while (data.Read())
386                             {
```

```
387                                 toReturn =
    data[0].ToString();
388                             }
389                         }
390                         break;
391                     default:
392                         break;
393                 }
394 
395             }
396             catch (DbException dbe)
397             {
398                 throw dbe;
399             }
400             return toReturn;
401         }
402         /// <summary>
403         /// Metodo utile a creare una copia di backup del
    database
404         /// </summary>
405         /// <param name="fullName"> nome completo di path
    del database di backup</param>
406         /// <param name="password"> password che si vuole
    settare per proteggere il nuovo db</param>
407         public void Backup(string fullName, string password)
408         {
409             try
410             {
411                 switch
    ((AdmittedDbType)Settings.Default.db_used_type)
412                 {
413                     case (AdmittedDbType.SQLite):
414 
    SQLiteConnection.CreateFile(fullName);
415                         using (var destination = new
    SQLiteConnection("Data Source=" + fullName + ";
    Version=3;"))
416                         {
417                             destination.Open();
418                             if
    (!String.IsNullOrEmpty(password))
419                             {
420 
    destination.ChangePassword(password);
421                             }
422 
    ((SQLiteConnection)instance.conn).BackupDatabase(destination,
    "main", "main", -1, null, 500);
423                             destination.Close();
424                             destination.Dispose();
425                         }
426                         break;
427                     default:
428                         break;
```

```
                }
            }
            catch (DbException dbe)
            {
                throw dbe;
            }
        }
        /// <summary>
        /// Metodo utile a modificare/assegnare una nuova
    password a protezione del database locale.
        /// </summary>
        /// <param name="newPassword"> nuova
    password</param>
        public void ChangeSettings(string newPassword)
        {
            try
            {
                switch
    ((AdmittedDbType)Settings.Default.db_used_type)
                {
                    case (AdmittedDbType.SQLite):

    ((SQLiteConnection)instance.conn).ChangePassword(newPassword);
                        Settings.Default.password =
    newPassword;
                        break;
                    default:
                        break;
                }
            }
            catch (DbException dbe)
            {
                throw dbe;
            }
        }
    }
}
```

Codice 4.5: Interfaccia IQueryCreator.cs

```
namespace Core
{
     /// <summary>
     /// Enumerato utile a definire in modo fisso quale
    tipo di query è necessario restituire
     /// </summary>
    public enum Querable
    {
        /// <summary>
        /// Se è necessario effettuare la connessione al DB.
        /// </summary>
        Connect = 1,
        /// <summary>
```

```
13          /// Se è necessario testare la connessione e le
        credenziali con cui si accede al DB.
14          /// </summary>
15          Test,
16          /// <summary>
17          /// Se è necessario conoscere tutti i tipi
        predefiniti presenti nel DB e le loro relative
        informazioni
18          /// </summary>
19          AdmittedType,
20          /// <summary>
21          /// Se è necessario conoscere le informazioni base
        di tutti i record per popolare il menù principale
22          /// </summary>
23          MainMenuItems,
24          /// <summary>
25          /// Se è necessario conoscere le informazioni
        dettagliate relative ad un record del DB
26          /// </summary>
27          DetailedInfo,
28          /// <summary>
29          /// Se è necessario creare un nuovo record
30          /// </summary>
31          CreateNewEntry,
32          /// <summary>
33          /// Se è necessario eliminare un record già
        esistente
34          /// </summary>
35          EraseEntry,
36          /// <summary>
37          /// Se è necessario aggiornare un record già
        esistente
38          /// </summary>
39          UpdateEntry,
40          /// <summary>
41          /// Se è necessario verificare la correttezza di un
        nome da associare ad un nuovo record
42          /// </summary>
43          CheckEntryName
44      };
45      /// <summary>
46      /// Interfaccia che fornisce un metodo basato su
        quattro overload utile a ed inoltrare la corretta query
        sql al database dell'applicativo.
47      /// </summary>
48      internal interface IQueryCreator
49      {
50          /// <summary>
51          /// Overload per richiedere la stringa di
        connessione, la stringa di test
52          /// della connessione, i tipi ammissibili come
        parametro dalla classe o per
53          /// richiedere di restituire tutti i record utili
        nella schermata principale
```

```
54          /// </summary>
55          /// <param name="query">Connect o Test o
        AdmittedType o MainMenuItems</param>
56          /// <returns>string - query selezionata</returns>
57          string Create(Querable query);
58          /// <summary>
59          /// Overload per richiedere tutte le informazioni
        su un record o per eliminarlo
60          /// </summary>
61          /// <param name="query">DetailedInfo o
        EraseEntry</param>
62          /// <param name="entryId">ID del record
        selezionato</param>
63          /// <returns></returns>
64          string Create(Querable query, int entryId);
65          /// <summary>
66          /// Overload per richiedere l'inserimento di un
        nuovo record.
67          /// </summary>
68          /// <param name="query">CreateNewEntry</param>
69          /// <param name="entryName">nome del record
        assegnato</param>
70          /// <param name="typeId">tipologia base di
        account</param>
71          /// <param name="noteValue">valore della nota
        associato</param>
72          /// <param name="passwordValue">valore della
        password associata</param>
73          /// <param name="urlValue">valore dell'URL
        associato</param>
74          /// <param name="usernameValue">valore
        dell'username associato</param>
75          /// <returns></returns>
76          string Create(Querable query, string entryName, int
        typeId, string noteValue, string passwordValue, string
        urlValue, string usernameValue);
77          /// <summary>
78          /// Overload per richiedere la modifica di un
        record già esistente
79          /// </summary>
80          /// <param name="query">UpdateEntry</param>
81          /// <param name="entryId">id del record da
        modificare</param>
82          /// <param name="entryName">nuovo nome da
        associare</param>
83          /// <param name="iconName">nuova icona da
        associare</param>
84          /// <param name="noteValue">nuovo valore delle note
        da associare</param>
85          /// <param name="passwordValue">nuovo valore della
        password da associare</param>
86          /// <param name="urlValue">nuovo valore dell'url da
        associare</param>
87          /// <param name="usernameValue">nuovo username da
```

```
    associare</param>
88          /// <returns></returns>
89          string Create(Querable query, int entryId, string
    entryName, int typeId, string iconName, string
    noteValue, string passwordValue, string urlValue, string
    usernameValue);
90          /// <summary>
91          /// Overload per richiedere la verifica e la
    eventuale modifica del nome del nuovo record
92          /// che si sta per inserire nel database
93          /// </summary>
94          /// <param name="query"></param>
95          /// <param name="entryName"></param>
96          /// <returns></returns>
97          string Create(Querable query, string entryName);
98      }
99  }
```

Codice 4.6: Classe QueryCreator.cs

```
1   using System;
2
3   namespace Core
4   {
5       /// <summary>
6       /// Classe astratta che dichiara il metodo pubblico
        ereditato dall'interfaccia e i metodi protetti utili
        alle classi concrete.
7       /// </summary>
8       internal abstract class QueryCreator : IQueryCreator
9       {
10          /// <summary>
11          /// Metodo per richiedere la stringa di
        connessione, la stringa di test
12          /// della connessione, i tipi ammissibili come
        parametro dalla classe o per
13          /// richiedere di restituire tutti i record utili
        nella schermata principale
14          /// </summary>
15          /// <param name="query">Connect o Test o
        AdmittedType o MainMenuItems</param>
16          /// <returns>string - query selezionata</returns>
17          public string Create(Querable query)
18          {
19              string toReturn = string.Empty;
20
21              switch (query)
22              {
23                  case Querable.Connect:
24                      toReturn = Connect();
25                      break;
26                  case Querable.Test:
27                      toReturn = Test();
28                      break;
```

```
29                  case Querable.AdmittedType:
30                      toReturn = AdmittedType();
31                      break;
32                  case Querable.MainMenuItems:
33                      toReturn = MainMenuItems();
34                      break;
35              }
36
37              return toReturn;
38          }
39          /// <summary>
40          /// Metodo per richiedere tutte le informazioni su
    un record o per eliminarlo
41          /// </summary>
42          /// <param name="query">DetailedInfo o
    EraseEntry</param>
43          /// <param name="entryId">ID del record
    selezionato</param>
44          /// <returns>string - query per richiedere tutti i
    dati di un elemento del db</returns>
45          public string Create(Querable query, int entryId)
46          {
47              string toReturn = string.Empty;
48
49              switch (query)
50              {
51                  case Querable.DetailedInfo:
52                      toReturn = DetailedInfo(entryId);
53                      break;
54                  case Querable.EraseEntry:
55                      toReturn = EraseEntry(entryId);
56                      break;
57              }
58
59              return toReturn;
60          }
61          /// <summary>
62          /// Metodo per richiedere l'inserimento di un nuovo
    record.
63          /// </summary>
64          /// <param name="query">CreateNewEntry</param>
65          /// <param name="entryName">nome del record
    assegnato</param>
66          /// <param name="typeId">tipologia base di
    account</param>
67          /// <param name="noteValue">valore della nota
    associato</param>
68          /// <param name="passwordValue">valore della
    password associata</param>
69          /// <param name="urlValue">valore dell'URL
    associato</param>
70          /// <param name="usernameValue">valore
    dell'username associato</param>
71          /// <returns>string - query per inserire un nuovo
```

```
        elemento nel db</returns>
72          public string Create(Querable query, string
        entryName, int typeId, string noteValue, string
        passwordValue, string urlValue, string usernameValue)
73          {
74              string toReturn = string.Empty;
75
76              if (query.Equals(Querable.CreateNewEntry))
77              {
78                  toReturn = CreateNewEntry(entryName,
        typeId, noteValue, passwordValue, urlValue,
        usernameValue);
79              }
80
81              return toReturn;
82          }
83          /// <summary>
84          /// Metodo per richiedere la modifica di un record
        già esistente
85          /// </summary>
86          /// <param name="query">UpdateEntry</param>
87          /// <param name="entryId">id del record da
        modificare</param>
88          /// <param name="entryName">nuovo nome da
        associare</param>
89          /// <param name="iconName">nuova icona da
        associare</param>
90          /// <param name="noteValue">nuovo valore delle note
        da associare</param>
91          /// <param name="passwordValue">nuovo valore della
        password da associare</param>
92          /// <param name="urlValue">nuovo valore dell'url da
        associare</param>
93          /// <param name="usernameValue">nuovo username da
        associare</param>
94          /// <returns>string - query per modificare un
        record</returns>
95          public string Create(Querable query, int entryId,
        string entryName, int typeId, string iconName, string
        noteValue, string passwordValue, string urlValue, string
        usernameValue)
96          {
97              string toReturn = string.Empty;
98
99              if (query.Equals(Querable.UpdateEntry))
100             {
101                 toReturn = UpdateEntry(entryId, entryName,
        typeId, iconName, noteValue, passwordValue, urlValue,
        usernameValue);
102             }
103
104             return toReturn;
105         }
106         /// <summary>
```

```
107         /// Metodo per richiedere la verifica e la
    eventuale modifica del nome del nuovo record
108         /// che si sta per inserire nel database
109         /// </summary>
110         /// <param name="query"></param>
111         /// <param name="entryName"></param>
112         /// <returns>string - query per richiedere la
    verifica/modifica di un nome</returns>
113         public string Create(Querable query, string
    entryName)
114         {
115             string toReturn = String.Empty;
116             string entry = (String.IsNullOrEmpty(entryName)
    ? "Account" : entryName);
117 
118             if (query.Equals(Querable.CheckEntryName))
119             {
120                 toReturn = CheckEntryName(entry);
121             }
122 
123             return toReturn;
124         }
125 
126         /// <summary>
127         /// Restituisce la query per connettersi alla
    istanza di DB selezionata
128         /// </summary>
129         /// <returns>string - Query per connettersi al
    DB</returns>
130         protected abstract string Connect();
131         /// <summary>
132         /// Restituisce la query per testare la connessione
    alla istanza di DB selezionata
133         /// </summary>
134         /// <returns>string - Query per testare la
    connessione al DB</returns>
135         protected abstract string Test();
136         /// <summary>
137         /// Restituisce la query per render noti tutti i
    tipi predefiniti disponibili
138         /// </summary>
139         /// <returns>string - Query per accedere a tutta la
    view type</returns>
140         protected abstract string AdmittedType();
141         /// <summary>
142         /// Restituisce la query per rendere noti tutti i
    record presenti e impostare il main menu
143         /// </summary>
144         /// <returns>string - query per accedere a tutta la
    view main</returns>
145         protected abstract string MainMenuItems();
146         /// <summary>
147         /// Restituisce la query per rendere note tutte le
    informazioni relative ad un record
```

```
148         /// </summary>
149         /// <param name="entryId">id del record da
    ricercare</param>
150         /// <returns>string - query per accedere alla view
    DetailedInfo</returns>
151         protected abstract string DetailedInfo(int entryId);
152         /// <summary>
153         /// Restituisce la query per validare il nome del
    record prima di tentare di inserirlo
154         /// </summary>
155         /// <param name="entryName">nome da associare al
    nuovo record</param>
156         /// <returns>string - query per verificare la
    validità di un nome prima di inserirlo nel db</returns>
157         protected abstract string CheckEntryName(string
    entryName);
158         /// <summary>
159         /// Restituisce la query per inserire un nuovo
    recod nel db
160         /// </summary>
161         /// <param name="entryName">nuovo nome da
    associare</param>
162         /// <param name="typeId">tipologia base di
    account</param>
163         /// <param name="noteValue">nuovo valore delle note
    da associare</param>
164         /// <param name="passwordValue">nuovo valore della
    password da associare</param>
165         /// <param name="urlValue">nuovo valore dell'url da
    associare</param>
166         /// <param name="usernameValue">nuovo username da
    associare</param>
167         /// <returns>string - query per inserire in tutte
    le tabelle i valori relativi</returns>
168         protected abstract string CreateNewEntry(string
    entryName, int typeId, string noteValue, string
    passwordValue, string urlValue, string usernameValue);
169         /// <summary>
170         /// Restituisce la query per eliminare tutte le
    informazioni riferite ad un record
171         /// </summary>
172         /// <param name="entryId">id del record da
    eliminare</param>
173         /// <returns>string - query per eliminare i dati da
    ogni tabella</returns>
174         protected abstract string EraseEntry(int entryId);
175         /// <summary>
176         /// Restituisce la query per modificare i dati
    specificati di un record
177         /// </summary>
178         /// <param name="entryId">id del record da
    modificare</param>
179         /// <param name="entryName">nuovo nome da
    associare</param>
```

```
180         /// <param name="typeId">id del tipo da
    modificare</param>
181         /// <param name="iconName">nuova icona da
    associare</param>
182         /// <param name="noteValue">nuovo valore delle note
    da associare</param>
183         /// <param name="passwordValue">nuovo valore della
    password da associare</param>
184         /// <param name="urlValue">nuovo valore dell'url da
    associare</param>
185         /// <param name="usernameValue">nuovo username da
    associare</param>
186         /// <returns>string - query per modificare i dati
    di un record nelle tabelle specificate</returns>
187         protected abstract string UpdateEntry(int entryId,
    string entryName, int typeId, string iconName, string
    noteValue, string passwordValue, string urlValue, string
    usernameValue);
188     }
189 }
```

Codice 4.7: Classe SQLiteQueryCreator.cs

```
1  using System;
2  using Core.Properties;
3
4  namespace Core
5  {
6      /// <summary>
7      /// Classe che implementa, specificamente per i
       database di tipo SQLite, il metodo pubblico ereditato
       dall'interfaccia e i metodi protetti ereditati dalla
       classe astratta.
8      /// </summary>
9      internal class SQLiteQueryCreator : QueryCreator
10     {
11         internal SQLiteQueryCreator()
12         {
13         }
14
15         /// <summary>
16         /// Restituisce la query per connettersi alla
       istanza di DB SQLite
17         /// </summary>
18         /// <returns>string - Query per connettersi al DB
       SQLite</returns>
19         protected override string Connect()
20         {
21             string toReturn = "Data Source=" +
       Settings.Default.db_path_sqlite +
22                               "; Version=3; Password=" +
       Settings.Default.password;
23             Console.WriteLine(toReturn);
24
```

```
25              return(toReturn);
26          }
27          /// <summary>
28          /// Restituisce la query per testare la connessione
    alla istanza di DB SQLite
29          /// </summary>
30          /// <returns>string - Query per testare la
    connessione al DB SQLite</returns>
31          protected override string Test()
32          {
33              return "SELECT * FROM sqlite_master;";
34          }
35          /// <summary>
36          /// Restituisce la query per render noti tutti i
    tipi predefiniti disponibili
37          /// </summary>
38          /// <returns>string - Query per accedere a tutta la
    view type</returns>
39          protected override string AdmittedType()
40          {
41              return ("SELECT * FROM `type` ORDER BY `id` ASC
    LIMIT 0, 50000;");
42          }
43          /// <summary>
44          /// Restituisce la query per rendere noti tutti i
    record presenti e impostare il main menu
45          /// </summary>
46          /// <returns>string - query per accedere a tutta la
    view main</returns>
47          protected override string MainMenuItems()
48          {
49              return ("SELECT * FROM `main` ORDER BY `ID` ASC
    LIMIT 0, 50000;");
50          }
51          /// <summary>
52          /// Restituisce la query per rendere note tutte le
    informazioni relative ad un record
53          /// </summary>
54          /// <param name="entryId">id del record da
    ricercare</param>
55          /// <returns>string - query per accedere alla view
    DetailedInfo</returns>
56          protected override string DetailedInfo(int entryId)
57          {
58              return ("SELECT * FROM `DetailedInfo`" +
59                  ((entryId > 0 || !entryId.Equals(null)) ?
    (" WHERE `ID` = " + entryId) : ("")) +
60                  " ORDER BY `ID` ASC LIMIT 0, 50000;");
61          }
62          /// <summary>
63          /// Restituisce la query per validare il nome del
    record prima di tentare di inserirlo
64          /// </summary>
65          /// <param name="entryName">nome da associare al
```

```
nuovo record</param>
66          /// <returns>string - query per verificare la
validità di un nome prima di inserirlo nel db</returns>
67          protected override string CheckEntryName(string
entryName)
68          {
69              string toReturn = String.Empty;
70
71              if (!String.IsNullOrEmpty(entryName))
72              {
73                  toReturn = "SELECT CASE" +
74                      "(SELECT count(name) FROM entry WHERE "
+
75                          "(name GLOB(\'" + entryName + "\')
OR name GLOB(\'" + entryName + "_*\'))) " +
76                              "WHEN 0 THEN \'" + entryName +
"\' " +
77                              "ELSE PRINTF(\'" + entryName +
"_%d\'," +
78                                  "(SELECT count(name) FROM
entry WHERE" +
79                                      "(name GLOB(\'" +
entryName + "\') OR name GLOB(\'" + entryName +
"_*\'))))" +
80                  " END;";
81              }
82              return toReturn;
83          }
84          /// <summary>
85          /// Restituisce la query per inserire un nuovo
recod nel db
86          /// </summary>
87          /// <param name="entryName">nuovo nome da
associare</param>
88          /// <param name="typeId">tipologia base di
account</param>
89          /// <param name="noteValue">nuovo valore delle note
da associare</param>
90          /// <param name="passwordValue">nuovo valore della
password da associare</param>
91          /// <param name="urlValue">nuovo valore dell'url da
associare</param>
92          /// <param name="usernameValue">nuovo username da
associare</param>
93          /// <returns>string - query per inserire in tutte
le tabelle i valori relativi</returns>
94          protected override string CreateNewEntry(string
entryName, int typeId, string noteValue, string
passwordValue, string urlValue, string usernameValue)
95          {
96              string entry =
(String.IsNullOrEmpty(entryName)) ? ("(SELECT
PRINTF(\'Record_%d\', (SELECT max(id) + 1 FROM
`entry`)))") : ('\'' + entryName + '\'');
```

```
97              int type = (typeId.Equals(null)) ? (1) :
    (typeId);
98              string note = (String.IsNullOrEmpty(noteValue))
    ? ("NULL") : ('\'' + noteValue + '\'');
99              string password =
    (String.IsNullOrEmpty(passwordValue)) ? ("NULL") : ('\''
    + passwordValue + '\'');
100             string url = (String.IsNullOrEmpty(urlValue) ||
    urlValue.Equals("NULL")) ?
101                     ("SELECT `url` FROM `type` WHERE
    `type`.`id` = " +
102                         "(SELECT `type_id` FROM `entry`
    WHERE `entry`.`id` = " +
103                             "(SELECT max(id) FROM
    `entry`))") :
104                     ('\'' + urlValue + '\'');
105             string username =
    (String.IsNullOrEmpty(usernameValue)) ? ("NULL") : ('\''
    + usernameValue + '\'');
106
107             return String.Format("BEGIN TRANSACTION;\n" +
108                 "INSERT INTO `entry` VALUES ((SELECT
    max(id) + 1 FROM `entry`), {0}, {1});\n" +
109                 "INSERT INTO `icon` VALUES ((SELECT max(id)
    + 1 FROM `icon`), (SELECT max(id) FROM `entry`)," +
110                     "(SELECT `default_icon` FROM `type`
    WHERE `type`.`id` = (SELECT `type_id` FROM `entry` WHERE
    `entry`.`id` = (SELECT max(id) FROM `entry`))));\n" +
111                 "INSERT INTO `note` VALUES ((SELECT max(id)
    + 1 FROM `note`),{2},(SELECT max(id) FROM `entry`));\n" +
112                 "INSERT INTO `password` VALUES ((SELECT
    max(id) + 1 FROM `password`),{3},(SELECT max(id) FROM
    `entry`));\n" +
113                 "INSERT INTO `url` VALUES ((SELECT max(id)
    + 1 FROM `url`), ({4}), (SELECT max(id) FROM
    `entry`));\n" +
114                 "INSERT INTO `username` VALUES ((SELECT
    max(id) + 1 FROM `username`),{5},(SELECT max(id) FROM
    `entry`));\n" +
115                 "COMMIT;", entry, type, note, password,
    url, username);
116         }
117         /// <summary>
118         /// Restituisce la query per eliminare tutte le
    informazioni riferite ad un record
119         /// </summary>
120         /// <param name="entryId">id del record da
    eliminare</param>
121         /// <returns>string - query per eliminare i dati da
    ogni tabella</returns>
122         protected override string EraseEntry(int entryId)
123         {
124             return String.Format("BEGIN TRANSACTION;\n" +
125                 "DELETE FROM `entry` WHERE `id` = {0};\n" +
```

```
126                 "DELETE FROM ‘icon‘ WHERE ‘id‘ = {0};\n" +
127                 "DELETE FROM ‘note‘ WHERE ‘id‘ = {0};\n" +
128                 "DELETE FROM ‘password‘ WHERE ‘id‘ =
    {0};\n" +
129                 "DELETE FROM ‘url‘ WHERE ‘id‘ = {0};\n" +
130                 "DELETE FROM ‘username‘ WHERE ‘id‘ =
    {0};\n" +
131                 "COMMIT;", entryId);
132         }
133         /// <summary>
134         /// Restituisce la query per modificare i dati
    specificati di un record
135         /// </summary>
136         /// <param name="entryId">id del record da
    modificare</param>
137         /// <param name="entryName">nuovo nome da
    associare</param>
138         /// <param name="typeId">id del tipo da
    modificare</param>
139         /// <param name="iconName">nuova icona da
    associare</param>
140         /// <param name="noteValue">nuovo valore delle note
    da associare</param>
141         /// <param name="passwordValue">nuovo valore della
    password da associare</param>
142         /// <param name="urlValue">nuovo valore dell'url da
    associare</param>
143         /// <param name="usernameValue">nuovo username da
    associare</param>
144         /// <returns>string - query per modificare i dati
    di un record nelle tabelle specificate</returns>
145         protected override string UpdateEntry(int entryId,
    string entryName, int typeId, string iconName, string
    noteValue, string passwordValue, string urlValue, string
    usernameValue)
146         {
147             string toReturn = String.Empty;
148             if (!String.IsNullOrEmpty(entryName))
149             {
150                 toReturn = String.IsNullOrEmpty(toReturn) ?
    "BEGIN TRANSACTION;\n" : toReturn;
151                 toReturn = String.Concat(toReturn,
    String.Format("UPDATE ‘entry‘ SET ‘name‘ = '{0}',
    ‘type_id‘ = {1} WHERE ‘id‘ = {2};\n", entryName, typeId,
    entryId));
152             }
153             if (!String.IsNullOrEmpty(iconName))
154             {
155                 toReturn = String.IsNullOrEmpty(toReturn) ?
    "BEGIN TRANSACTION;\n" : toReturn;
156                 toReturn = String.Concat(toReturn,
    String.Format("UPDATE ‘icon‘ SET ‘name‘ = '{0}' WHERE
    ‘id‘ = {1};\n", iconName, entryId));
157             }
```

```
158             if (!String.IsNullOrEmpty(noteValue))
159             {
160                 toReturn = String.IsNullOrEmpty(toReturn) ?
    "BEGIN TRANSACTION;\n" : toReturn;
161                 toReturn = String.Concat(toReturn,
    String.Format("UPDATE `note` SET `value` = '{0}' WHERE
    `id` = {1};\n", noteValue, entryId));
162             }
163             if (!String.IsNullOrEmpty(passwordValue))
164             {
165                 toReturn = String.IsNullOrEmpty(toReturn) ?
    "BEGIN TRANSACTION;\n" : toReturn;
166                 toReturn = String.Concat(toReturn,
    String.Format("UPDATE `password` SET `value` = '{0}'
    WHERE `id` = {1};\n", passwordValue, entryId));
167             }
168             if (!String.IsNullOrEmpty(urlValue))
169             {
170                 toReturn = String.IsNullOrEmpty(toReturn) ?
    "BEGIN TRANSACTION;\n" : toReturn;
171                 toReturn = String.Concat(toReturn,
    String.Format("UPDATE `url` SET `value` = '{0}' WHERE
    `id` = {1};\n", urlValue, entryId));
172             }
173             if (!String.IsNullOrEmpty(usernameValue))
174             {
175                 toReturn = String.IsNullOrEmpty(toReturn) ?
    "BEGIN TRANSACTION;\n" : toReturn;
176                 toReturn = String.Concat(toReturn,
    String.Format("UPDATE `username` SET `value` = '{0}'
    WHERE `id` = {1};\n", usernameValue, entryId));
177             }
178
179             if (!String.IsNullOrEmpty(toReturn))
180             {
181                 toReturn = String.Concat(toReturn,
    "COMMIT;");
182             }
183
184             return toReturn;
185         }
186     }
187 }
```

Codice 4.8: Interfaccia DataStruct/IDBRecord.cs

```
1  namespace Core.DataStruct
2  {
3      /// <summary>
4      /// Interfaccia che rappresenta le informazioni utili a
       descrivere un record
5      /// </summary>
6      public interface IDBRecord
7      {
```

```
8           /// <summary>
9           /// Proprietà che rappresenta l'id associato al
    record
10          /// </summary>
11          int Id { get; set; }
12          /// <summary>
13          /// Proprietà che rappresenta il nome associato al
    record
14          /// </summary>
15          string Name { get; set; }
16          /// <summary>
17          /// Proprietà che rappresenta l'icona associata al
    record
18          /// </summary>
19          string Icon { get; set; }
20          /// <summary>
21          /// Proprietà che rappresenta le note associate al
    record
22          /// </summary>
23          string Note { get; set; }
24          /// <summary>
25          /// Proprietà che rappresenta la password
    dell'account salvato nel record
26          /// </summary>
27          string Password { get; set; }
28          /// <summary>
29          /// Proprietà che rappresenta lo url associato al
    record
30          /// </summary>
31          string Url { get; set; }
32          /// <summary>
33          /// Propreità che rappresenta lo username
    dell'account salvato nel record
34          /// </summary>
35          string Username { get; set; }
36          /// <summary>
37          /// Proprietà che rappresenta l'id del tipo
    predefinito associato al record
38          /// </summary>
39          int TypeId { get; set; }
40      }
41  }
```

Codice 4.9: Classe DataStruct/DBRecord.cs

```
1   namespace Core.DataStruct
2   {
3       /// <summary>
4       /// Classe che rappresenta le informazioni relative ad
    un record
5       /// </summary>
6       public class DBRecord : IDBRecord
7       {
8           /// <summary>
```

```
9           /// Proprietà che rappresenta l'id associato al
    record
10          /// </summary>
11          public int Id { get; set; }
12          /// <summary>
13          /// Proprietà che rappresenta il nome associato al
    record
14          /// </summary>
15          public string Name { get; set; }
16          /// <summary>
17          /// Proprietà che rappresenta l'icona associata al
    record
18          /// </summary>
19          public string Icon { get; set; }
20          /// <summary>
21          /// Proprietà che rappresenta le note associate al
    record
22          /// </summary>
23          public string Note { get; set; }
24          /// <summary>
25          /// Proprietà che rappresenta la password
    dell'account salvato nel record
26          /// </summary>
27          public string Password { get; set; }
28          /// <summary>
29          /// Propreità che rappresenta lo url associato al
    record
30          /// </summary>
31          public string Url { get; set; }
32          /// <summary>
33          /// Propreità che rappresenta lo username
    dell'account salvato nel record
34          /// </summary>
35          public string Username { get; set; }
36          /// <summary>
37          /// Proprietà che rappresenta l'id del tipo
    predefinito associato al record
38          /// </summary>
39          public int TypeId { get; set; }
40
41          /// <summary>
42          /// Costruttore della classe DBRecord
43          /// </summary>
44          /// <param name="id">identificato del record sul
    db</param>
45          /// <param name="name">nome del record</param>
46          /// <param name="defaultIcon">icona associata al
    record</param>
47          /// <param name="note">note associate al
    record</param>
48          /// <param name="password">password associata al
    record</param>
49          /// <param name="url">url associato al
    record</param>
```

```
50          /// <param name="username">username associata al
    record</param>
51          /// <param name="typeId">identificativo del tipo
    predefinito associato al record</param>
52          public DBRecord(int id, string name, string
    defaultIcon, string note, string password, string url,
    string username, int typeId)
53          {
54              Id = id;
55              Name = name;
56              Icon = defaultIcon;
57              Note = note;
58              Password = password;
59              Url = url;
60              Username = username;
61              TypeId = typeId;
62          }
63      }
64  }
```

Codice 4.10: Interfaccia DataStruct/IRecordTypeItem.cs

```
1   namespace Core.DataStruct
2   {
3       /// <summary>
4       /// Interfaccia che rappresenta un tipo predefinito di
    record presente sul database
5       /// </summary>
6       public interface IRecordTypeItem
7       {
8           /// <summary>
9           /// Restituisce il codice identificativo del record.
10          /// </summary>
11          /// <returns></returns>
12          int GetId();
13          /// <summary>
14          /// Restituisce il nome del record.
15          /// </summary>
16          /// <returns>nome del record</returns>
17          string GetName();
18          /// <summary>
19          /// Restituisce il nome dell'icona di riferimento
    senza il percorso.
20          /// </summary>
21          /// <returns>nome dell'icona senza
    percorso</returns>
22          string GetIcon();
23          /// <summary>
24          /// Restituisce l'url associato al record di
    default.
25          /// </summary>
26          /// <returns>url predefinito associato al
    record</returns>
27          string GetUrl();
```

```
28      }
29  }
```

Codice 4.11: Classe DataStruct/RecordTypeItem.cs

```
1   namespace Core.DataStruct
2   {
3       /// <summary>
4       /// Classe che rappresenta un tipo predefinito di
        record presente sul database
5       /// </summary>
6       public class RecordTypeItem : IRecordTypeItem
7       {
8           /// <summary>
9           /// Rappresenta il codice identificativo del record.
10          /// </summary>
11          protected int Id { get; set; }
12          /// <summary>
13          /// Rappresenta il nome associato al record.
14          /// </summary>
15          protected string Name { get; set; }
16          /// <summary>
17          /// Rappresenta il nome dell'icona di riferimento
        senza il suo percorso.
18          /// </summary>
19          protected string DefaultIcon { get; set; }
20          /// <summary>
21          /// Rappresenta l'URL associato al record.
22          /// </summary>
23          protected string Url { get; set; }
24
25          /// <summary>
26          /// Costruttore della classe RecordTypeItem.
27          /// </summary>
28          /// <param name="id">id del record</param>
29          /// <param name="name">nome predefinito</param>
30          /// <param name="defaultIcon">icona
        predefinita</param>
31          /// <param name="url">url predefinito</param>
32          public RecordTypeItem(int id, string name, string
        defaultIcon, string url)
33          {
34              Id = id;
35              Name = name;
36              DefaultIcon = defaultIcon;
37              Url = url;
38          }
39
40          /// <summary>
41          /// Restituisce il codice identificativo del record.
42          /// </summary>
43          /// <returns></returns>
44          public int GetId()
45          {
```

```
46              return Id;
47          }
48          /// <summary>
49          /// Restituisce il nome del record.
50          /// </summary>
51          /// <returns>nome del record</returns>
52          public string GetName()
53          {
54              return Name;
55          }
56          /// <summary>
57          /// Restituisce il nome dell'icona di riferimento
    senza il percorso.
58          /// </summary>
59          /// <returns>nome dell'icona senza
    percorso</returns>
60          public string GetIcon()
61          {
62              return DefaultIcon;
63          }
64          /// <summary>
65          /// Restituisce l'url associato al record di
    default.
66          /// </summary>
67          /// <returns>url predefinito associato al
    record</returns>
68          public string GetUrl()
69          {
70              return Url;
71          }
72      }
73  }
```

4.3 Classi del progetto "*N2L-Need_2_Log*"

Classi presenti:

- Program.cs
- gui/DetailedInfo.cs
- gui/FormAboutBox.cs
- gui/FormCreate.cs
- gui/FormFirstAccess.cs
- gui/FormLogin.cs
- gui/FormMainMenu.cs
- gui/FormModify.cs
- gui/FormOption.cs

Codice 4.12: Classe Program.cs

```
1  using System;
2  using System.Windows.Forms;
3  using Core;
4
5  namespace N2L_Need_2_Log
6  {
7      static class Program
8      {
9          /// <summary>
10         /// Punto di ingresso principale dell'applicazione.
11         /// </summary>
12         [STAThread]
13         static void Main()
14         {
15             Application.EnableVisualStyles();
16
       Application.SetCompatibleTextRenderingDefault(false);
17             Controller.CheckSettings();
18             Application.Run(new gui.FormMainMenu());
19         }
20     }
21 }
```

Codice 4.13: Classe DetailedInfo.cs

```
1  using System;
2  using System.IO;
3  using System.Reflection;
4  using System.Data.Common;
5  using System.Diagnostics;
6  using System.ComponentModel;
7  using System.Drawing;
8  using System.Windows.Forms;
9  using Core;
10 using Core.DataStruct;
11
12 namespace N2L_Need_2_Log.gui
13 {
14     /// <summary>
```

```
15      /// Classe che definisce il comportamento del form
        DetailedInfo
16      /// </summary>
17      public partial class DetailedInfo : Form
18      {
19          private DBRecord record;
20          private static int item;
21
22          /// <summary>
23          /// Costruttore della classe DetailedInfo
24          /// </summary>
25          /// <param name="itemID"></param>
26          public DetailedInfo(int itemID)
27          {
28              InitializeComponent();
29              item = itemID;
30          }
31
32          private void buttonPassword_MouseDown(object
        sender, MouseEventArgs e)
33          {
34              if(e.Button == MouseButtons.Left)
35              {
36                  textBoxPassword.UseSystemPasswordChar =
        false;
37              }
38          }
39          private void buttonPassword_MouseUp(object sender,
        MouseEventArgs e)
40          {
41              textBoxPassword.UseSystemPasswordChar = true;
42          }
43          private void linkLabelUrl_LinkClicked(object
        sender, LinkLabelLinkClickedEventArgs e)
44          {
45              try
46              {
47                  Process.Start(linkLabelUrl.Text);
48              }
49              catch
50                  (Win32Exception noBrowser)
51              {
52                  if (noBrowser.ErrorCode == -2147467259)
53                      MessageBox.Show(noBrowser.Message);
54              }
55              catch (Exception other)
56              {
57                  MessageBox.Show(this, "Error with your
        browser.",other.Message, MessageBoxButtons.OK);
58                  this.linkLabelUrl.Enabled = false;
59              }
60          }
61          private void buttonOk_Click(object sender,
        EventArgs e)
```

```
62          {
63              this.Close();
64          }
65          private void DetailedInfo_Load(object sender,
    EventArgs e)
66          {
67              try
68              {
69                  DBConnector database = DBConnector.Connect;
70                  using (DbDataReader data =
    database.GetRecordInfo(item))
71                  {
72                      while (data.Read())
73                      {
74                          record = new DBRecord(item,
    data["NAME"].ToString(),
75                              data["IMAGE"].ToString(),
    data["NOTE"].ToString(),
76                              data["PASSWORD"].ToString(),
    data["URL"].ToString(),
77                              data["USERNAME"].ToString(),
    Convert.ToInt32(data["TYPE_ID"].ToString()));
78                          this.Text = record.Name + " - " +
79
    Path.GetFileNameWithoutExtension(Assembly.GetExecutingAssembly().CodeBase);
80                          this.labelName.Text = record.Name;
81                          this.recordIcon.Image =
    Image.FromFile("..\\..\\..\\Core\\Resources\\Pers\\Icon\\"
    + record.Icon);
82                          this.recordIcon.MaximumSize = new
    Size(128, 128);
83                          this.richTextBoxNote.Text =
    record.Note;
84                          this.textBoxPassword.Text =
    record.Password;
85                          this.linkLabelUrl.Text = record.Url;
86                          this.textBoxUsername.Text =
    record.Username;
87                      }
88                  }
89              }
90              catch (DbException dbe)
91              {
92                  MessageBox.Show(this, dbe.Message,
93                      "Errore SQL n. " + dbe.ErrorCode,
    MessageBoxButtons.OK);
94              this.Close();
95              }
96              catch (Exception ex)
97              {
98                  MessageBox.Show(this, ex.Message, "Generic
    Error!", MessageBoxButtons.OK);
99                  this.Close();
100             }
```

```
        }
    }
}
```

Codice 4.14: Classe FormAboutBox.cs

```
using System;
using System.Reflection;
using System.Windows.Forms;

namespace N2L_Need_2_Log.gui
{
    /// <summary>
    /// Classe che definisce il comportamento del form FormAboutBox
    /// </summary>
    partial class FormAboutBox : Form
    {
        /// <summary>
        /// Costruttore della classe FormAboutBox
        /// </summary>
        public FormAboutBox()
        {
            InitializeComponent();
        }

        #region Funzioni di accesso attributo assembly
            private string AssemblyTitle
            {
                get
                {
                    object[] attributes = Assembly.GetExecutingAssembly().GetCustomAttributes(typeof(AssemblyTitleAttribut
    false);
                    if (attributes.Length > 0)
                    {
                        AssemblyTitleAttribute titleAttribute = (AssemblyTitleAttribute)attributes[0];
                        if (titleAttribute.Title != "")
                        {
                            return titleAttribute.Title;
                        }
                    }
                    return System.IO.Path.GetFileNameWithoutExtension(Assembly.GetExecutingAssembly().CodeB
                }
            }
            private string AssemblyVersion
            {
                get
                {
                    return Assembly.GetExecutingAssembly().GetName().Version.ToString();
                }
```

```
43              }
44              private string AssemblyDescription
45              {
46                  get
47                  {
48                      object[] attributes =
    Assembly.GetExecutingAssembly().GetCustomAttributes(typeof(AssemblyDescriptionAt
    false);
49                      if (attributes.Length == 0)
50                      {
51                          return "";
52                      }
53                      return
    ((AssemblyDescriptionAttribute)attributes[0]).Description;
54                  }
55              }
56              private string AssemblyProduct
57              {
58                  get
59                  {
60                      object[] attributes =
    Assembly.GetExecutingAssembly().GetCustomAttributes(typeof(AssemblyProductAttrib
    false);
61                      if (attributes.Length == 0)
62                      {
63                          return "";
64                      }
65                      return
    ((AssemblyProductAttribute)attributes[0]).Product;
66                  }
67              }
68              private string AssemblyCopyright
69              {
70                  get
71                  {
72                      object[] attributes =
    Assembly.GetExecutingAssembly().GetCustomAttributes(typeof(AssemblyCopyrightAttr
    false);
73                      if (attributes.Length == 0)
74                      {
75                          return "";
76                      }
77                      return
    ((AssemblyCopyrightAttribute)attributes[0]).Copyright;
78                  }
79              }
80              private string AssemblyCompany
81              {
82                  get
83                  {
84                      object[] attributes =
    Assembly.GetExecutingAssembly().GetCustomAttributes(typeof(AssemblyCompanyAttrib
    false);
85                      if (attributes.Length == 0)
```

```
86                      {
87                          return "";
88                      }
89                      return
    ((AssemblyCompanyAttribute)attributes[0]).Company;
90                  }
91              }
92          #endregion
93
94          private void okButton_Click(object sender,
    EventArgs e)
95          {
96              this.Close();
97          }
98          private void FormAboutBox_Load(object sender,
    EventArgs e)
99          {
100             this.Text = String.Format("Informazioni su
    {0}", AssemblyTitle);
101             this.labelProductName.Text = AssemblyProduct;
102             this.labelVersion.Text =
    String.Format("Versione {0}", AssemblyVersion);
103             this.labelCopyright.Text = AssemblyCopyright;
104             this.labelCompanyName.Text = AssemblyCompany;
105             this.textBoxDescription.Text =
    AssemblyDescription;
106         }
107     }
108 }
```

Codice 4.15: Classe FormCreate.cs

```
1  using System;
2  using System.Collections.Generic;
3  using System.Drawing;
4  using System.Linq;
5  using System.Data.Common;
6  using System.Windows.Forms;
7  using Core;
8  using Core.DataStruct;
9
10 namespace N2L_Need_2_Log.gui
11 {
12     /// <summary>
13     /// Classe che definisce il comportamento del form
    FormCreate
14     /// </summary>
15     public partial class FormCreate : Form
16     {
17         private bool newRecord;
18         private int item;
19         private DBConnector database;
20         private DBRecord record;
```

```
21          private List<RecordTypeItem> recordList = new
        List<RecordTypeItem>();
22
23          /// <summary>
24          /// Costruttore della classe FormCreate utile per
        creare un nuovo record
25          /// </summary>
26          public FormCreate()
27          {
28              InitializeComponent();
29              newRecord = true;
30          }
31          /// <summary>
32          /// Costruttore della classe FormCreate utile per
        modificare un record già esistente
33          /// </summary>
34          /// <param name="itemID">id del record che si vuole
        modificare</param>
35          public FormCreate(int itemID)
36          {
37              InitializeComponent();
38              newRecord = false;
39              item = itemID;
40          }
41
42          private void
        checkBoxShowPassword_CheckedChanged(object sender,
        EventArgs e)
43          {
44              if (this.checkBoxShowPassword.Checked == true)
45              {
46                  this.textBoxPassword.UseSystemPasswordChar
        = true;
47                  this.checkBoxShowPassword.Text = "Nascondi";
48              }
49              else
50              {
51                  this.textBoxPassword.UseSystemPasswordChar
        = false;
52                  this.checkBoxShowPassword.Text = "Mostra";
53              }
54          }
55          private void FormCreate_Load(object sender,
        EventArgs e)
56          {
57              try
58              {
59                  this.database = DBConnector.Connect;
60                  using (DbDataReader data =
        database.GetRecordsType())
61                  {
62                      while (data.Read())
63                      {
64                          recordList.Add(new
```

```
        RecordTypeItem(Convert.ToInt32(data["id"].ToString()),
65                                 data["name"].ToString(),
        data["default_icon"].ToString(),
        data["url"].ToString()));
66
        this.comboBoxAccountType.Items.Add(this.recordList.Last().GetName());
67                      }
68                  }
69                  this.comboBoxAccountType.SelectedIndex = 0;
70                  if(!newRecord)
71                  {
72                      DBConnector database =
        DBConnector.Connect;
73                      using (DbDataReader data =
        database.GetRecordInfo(item))
74                      {
75                          while (data.Read())
76                          {
77                              record = new DBRecord(item,
        data["NAME"].ToString(),
78
        data["IMAGE"].ToString(), data["NOTE"].ToString(),
79
        data["PASSWORD"].ToString(), data["URL"].ToString(),
80
        data["USERNAME"].ToString(),
        Convert.ToInt32(data["TYPE_ID"].ToString()));
81                          }
82
        this.comboBoxAccountType.SelectedIndex = record.TypeId
        -1;
83                          RecordTypeItem itemRec =
        recordList.ElementAt(this.comboBoxAccountType.SelectedIndex);
84                          this.Text = "N2L - Need2Log | " +
        record.Name;
85                          this.textBoxName.Text = record.Name;
86                          this.pictureBox.Image =
        Image.FromFile("..\\..\\..\\Core\\Resources\\Pers\\icon\\"
        + itemRec.GetIcon());
87                          this.pictureBox.MaximumSize = new
        Size(128, 128);
88                          this.richTextBoxNote.Text =
        record.Note;
89                          this.textBoxPassword.Text =
        record.Password;
90                          this.textBoxUrl.Text = record.Url;
91                          this.textBoxUsername.Text =
        record.Username;
92                      }
93                  }
94              }
95              catch(DbException dbe)
96              {
97                  throw dbe;
```

```
98              }
99          }
100         private void buttonCancel_Click(object sender,
    EventArgs e)
101         {
102             this.Close();
103         }
104         private void buttonOk_Click(object sender,
    EventArgs e)
105         {
106             if (newRecord)
107             {
108                 try
109                 {
110                     string entryName =
    database.CheckEntryName(this.textBoxName.Text);
111                     this.textBoxName.Text = entryName;
112                     int typeId =
    this.comboBoxAccountType.SelectedIndex + 1;
113                     string noteValue =
    this.richTextBoxNote.Text;
114                     string passwordValue =
    this.textBoxPassword.Text;
115                     string urlValue = this.textBoxUrl.Text;
116                     string usernameValue =
    this.textBoxUsername.Text;
117                     database.Insert(entryName, typeId,
    noteValue, passwordValue, urlValue, usernameValue);
118                     this.Close();
119                 }
120                 catch (DbException dbe)
121                 {
122                     string message = "Errore in creazione
    nuovo record!\nErrore: " + dbe.Message;
123                     string caption = "SQLite error n.: " +
    dbe.ErrorCode;
124                     MessageBoxButtons buttons =
    MessageBoxButtons.OK;
125                     DialogResult result;
126                     result = MessageBox.Show(this, message,
    caption, buttons);
127                 }
128             }
129             else
130             {
131                 try
132                 {
133                     if (this.textBoxName.Text !=
    record.Name)
134                     {
135                         string entryName =
    database.CheckEntryName(this.textBoxName.Text);
136                         this.textBoxName.Text = entryName;
137                         record.Name = entryName;
```

```
138                     }
139                     record.TypeId =
    this.comboBoxAccountType.SelectedIndex + 1;
140                     record.Note = this.richTextBoxNote.Text;
141                     record.Password =
    this.textBoxPassword.Text;
142                     record.Url = this.textBoxUrl.Text;
143                     record.Username =
    this.textBoxUsername.Text;
144                     using (DbDataReader data =
    database.GetRecordsType())
145                     {
146                         RecordTypeItem item =
    recordList.ElementAt(this.comboBoxAccountType.SelectedIndex);
147                         record.Icon = item.GetIcon();
148                     }
149                     database.Update(record.Id, record.Name,
    record.TypeId, record.Icon, record.Note,
    record.Password, record.Url, record.Username);
150                     this.Close();
151                 }
152                 catch(DbException dbe)
153                 {
154                     string message = "Errore in modifica
    del record!\nErrore: " + dbe.Message;
155                     string caption = "SQLite error n.: " +
    dbe.ErrorCode;
156                     MessageBoxButtons buttons =
    MessageBoxButtons.OK;
157                     DialogResult result;
158                     result = MessageBox.Show(this, message,
    caption, buttons);
159                 }
160             }
161         }
162         private void textBoxName_TextChanged(object sender,
    EventArgs e)
163         {
164             if (String.IsNullOrEmpty(this.textBoxName.Text))
165             {
166                 this.Text = "N2L - Need2Log";
167             }
168             else
169             {
170                 this.Text = "N2L - Need2Log | " +
    this.textBoxName.Text;
171             }
172         }
173         private void FormCreate_FormClosing(object sender,
    FormClosingEventArgs e)
174         {
175
176         }
177         private void
```

```
        comboBoxAccountType_SelectedIndexChanged(object sender,
        EventArgs e)
178         {
179             RecordTypeItem item =
        recordList.ElementAt(this.comboBoxAccountType.SelectedIndex);
180             if (newRecord)
181             {
182                 this.textBoxName.Text = item.GetName();
183             }
184             this.pictureBox.Image =
        Image.FromFile("..\\..\\..\\Core\\Resources\\Pers\\icon\\"
        + item.GetIcon());
185             this.pictureBox.MaximumSize = new Size(128,
        128);
186             this.textBoxUrl.Text = item.GetUrl();
187         }
188     }
189 }
```

Codice 4.16: Classe FormFistAccess.cs

```
1  using System;
2  using System.Data.Common;
3  using System.Windows.Forms;
4  using Core;
5
6  namespace N2L_Need_2_Log.gui
7  {
8      /// <summary>
9      /// Classe che definisce il comportamento del form
       FormFirstAccess
10     /// </summary>
11     public partial class FormFirstAccess : Form
12     {
13         /// <summary>
14         /// costruttore della classe FormFirstAccess
15         /// </summary>
16         public FormFirstAccess()
17         {
18             InitializeComponent();
19         }
20
21         private void buttonOk_Click(object sender,
       EventArgs e)
22         {
23             if
       (String.IsNullOrEmpty(this.textBoxUsername.Text) ||
       String.Equals(this.textBoxUsername.Text, ""))
24             {
25                 Controller.Username = "User";
26                 Properties.Settings.Default.Save();
27             }
28             if
       (!this.textBoxPassword.Text.Equals(this.textBoxConfirmPassword.Text))
```

```
29              {
30                  string message = "Le password non
    corrispondono, impossibile settare la password.";
31                  string caption = "Errore in input!";
32                  MessageBoxButtons buttons =
    MessageBoxButtons.OK;
33                  DialogResult result;
34                  result = MessageBox.Show(this, message,
    caption, buttons);
35              }
36              else
37              {
38                  Controller.Password =
    Cript.ComputeHash(this.textBoxPassword.Text, null);
39                  try
40                  {
41                      var db = DBConnector.Connect;
42                      Controller.Logged = true;
43                      this.Close();
44                  }
45                  catch (DbException dbe)
46                  {
47                      MessageBox.Show(this, dbe.Message,
    "Errore in creazione DB n. " +
48                          dbe.ErrorCode,
    MessageBoxButtons.OK);
49                      this.Close();
50                  }
51                  catch (Exception ex)
52                  {
53                      MessageBox.Show(this, ex.Message,
    "Generic Error!", MessageBoxButtons.OK);
54                      this.Close();
55                  }
56              }
57          }
58          private void textBoxPassword_TextChanged(object
    sender, EventArgs e)
59          {
60              if (this.textBoxPassword.Text != String.Empty
    && this.textBoxConfirmPassword.Text != String.Empty)
61              {
62                  if
    (this.textBoxPassword.Text.Equals(this.textBoxConfirmPassword.Text))
63                  {
64                      this.labelPasswordError.Visible = false;
65                      this.labelPasswordConfirmed.Visible =
    true;
66                  }
67                  else
68                  {
69                      this.labelPasswordConfirmed.Visible =
    false;
70                      this.labelPasswordError.Visible = true;
```

```
71                  }
72              }
73              else
74              {
75                  this.labelPasswordConfirmed.Visible = false;
76                  this.labelPasswordError.Visible = false;
77              }
78          }
79          private void
    textBoxConfirmPassword_TextChanged(object sender,
    EventArgs e)
80          {
81              if (this.textBoxPassword.Text != String.Empty
    && this.textBoxConfirmPassword.Text != String.Empty)
82              {
83                  if
    (this.textBoxPassword.Text.Equals(this.textBoxConfirmPassword.Text))
84                  {
85                      this.labelPasswordError.Visible = false;
86                      this.labelPasswordConfirmed.Visible =
    true;
87                  }
88                  else
89                  {
90                      this.labelPasswordConfirmed.Visible =
    false;
91                      this.labelPasswordError.Visible = true;
92                  }
93              }
94              else
95              {
96                  this.labelPasswordConfirmed.Visible = false;
97                  this.labelPasswordError.Visible = false;
98              }
99          }
100     }
101 }
```

Codice 4.17: Classe FormLogin.cs

```
1  using System;
2  using System.Data.Common;
3  using System.Windows.Forms;
4  using Core;
5
6  namespace N2L_Need_2_Log
7  {
8      /// <summary>
9      /// Classe che definisce il comportamento del form
       FormLogin
10     /// </summary>
11     public partial class FormLogin : Form
12     {
13         /// <summary>
```

```
14          /// Costruttore della classe FormLogin
15          /// </summary>
16          public FormLogin()
17          {
18              InitializeComponent();
19          }
20
21          private void buttonLogin_Click(object sender,
    EventArgs e)
22          {
23              Controller.Password =
    Cript.ComputeHash(this.textBoxPwd.Text, null);
24              try
25              {
26                  var db = DBConnector.Connect;
27                  this.labelWrongPassword.Visible = false;
28                  string message = "Bentornato! Premi Invio
    per accedere";
29                  string caption = "Ciao " +
    Controller.Username;
30                  MessageBoxButtons buttons =
    MessageBoxButtons.OK;
31                  DialogResult result;
32                  result = MessageBox.Show(this, message,
    caption, buttons);
33
34                  Controller.Logged = true;
35
36                  this.Close();
37              }
38              catch (DbException)
39              {
40                  Controller.Logged = false;
41                  this.labelWrongPassword.Visible = true;
42                  this.textBoxPwd.SelectAll();
43              }
44          }
45          private void buttonExit_Click(object sender,
    EventArgs e)
46          {
47              Controller.OnClose();
48              this.Close();
49          }
50      }
51  }
```

Codice 4.18: Classe FormMainMenu.cs

```
1  using System;
2  using System.Data.Common;
3  using System.Drawing;
4  using System.Windows.Forms;
5  using Core;
6
```

```
7  namespace N2L_Need_2_Log.gui
8  {
9      /// <summary>
10     /// Classe che definisce il comportamento del form
       FormMainMenu
11     /// </summary>
12     public partial class FormMainMenu : Form
13     {
14         /// <summary>
15         /// costruttore della classe FormMainMenu
16         /// </summary>
17         public FormMainMenu()
18         {
19             InitializeComponent();
20         }
21
22         private void aboutToolStripMenuItem_Click(object
       sender, EventArgs e)
23         {
24             FormAboutBox fab = new FormAboutBox();
25             fab.Activate();
26             fab.ShowDialog(this);
27         }
28         private void opzioniToolStripMenuItem_Click(object
       sender, EventArgs e)
29         {
30             FormOption fo = new FormOption();
31             fo.Activate();
32             fo.ShowDialog(this);
33         }
34         private void esciToolStripMenuItem_Click(object
       sender, EventArgs e)
35         {
36             this.Close();
37         }
38         private void FormMainMenu_Load(object sender,
       EventArgs e)
39         {
40             this.Text =
       System.IO.Path.GetFileNameWithoutExtension(System.Reflection.Assembly.GetExecuti
41             this.listViewMainMenu.View =
       Properties.Settings.Default.MainMenuView;
42             switch (this.listViewMainMenu.View)
43             {
44                 case (View.LargeIcon):
45
       this.iconegrandiToolStripMenuItem.Checked = true;
46
       this.iconepiccoleToolStripMenuItem.Checked = false;
47                     this.listaToolStripMenuItem.Checked =
       false;
48                     break;
49                 case (View.SmallIcon):
50
```

```
        this.iconegrandiToolStripMenuItem.Checked = false;
51
        this.iconepiccoleToolStripMenuItem.Checked = true;
52                      this.listaToolStripMenuItem.Checked =
    false;
53                      break;
54                  case (View.List):
55
        this.iconegrandiToolStripMenuItem.Checked = false;
56
        this.iconepiccoleToolStripMenuItem.Checked = false;
57                      this.listaToolStripMenuItem.Checked =
    true;
58                      break;
59                  default:
60                      break;
61              }
62              //Sign In or Sign Up
63              if (!Controller.DbExist)
64              {
65                  FormFirstAccess ffa = new FormFirstAccess();
66                  ffa.Activate();
67                  ffa.ShowDialog(this);
68              }
69              else
70              {
71                  FormLogin fl = new FormLogin();
72                  fl.Activate();
73                  fl.ShowDialog(this);
74              }
75              //If user correctly signed in
76              if (Controller.Logged == true)
77              {
78                  try
79                  {
80                      updateListViewMainMenu();
81                  }
82                  catch(DbException dbe)
83                  {
84                      string message = "Errore durante la
    ricerca dei record presenti!\nErrore: " + dbe.Message;
85                      string caption = "SQLite error n.: " +
    dbe.ErrorCode;
86                      MessageBoxButtons buttons =
    MessageBoxButtons.OK;
87                      DialogResult result;
88                      result = MessageBox.Show(this, message,
    caption, buttons);
89                      this.Close();
90                  }
91              }
92              else
93              {
94                  this.Close();
```

```
95              }
96
97          }
98          private void toolStripMenuItem1_Click(object
        sender, EventArgs e)
99          {
100             SaveFileDialog saveFileDialog = new
        SaveFileDialog();
101             saveFileDialog.AddExtension = true;
102             saveFileDialog.DefaultExt = "sqlite3";
103             saveFileDialog.Title = "Backup";
104             saveFileDialog.InitialDirectory =
        Environment.GetFolderPath(Environment.SpecialFolder.Personal);
105             saveFileDialog.Filter = "file di database
        SQLite (*.sqlite3)|*.sqlite3|Tutti i file (*.*)|*.*";
106             if (saveFileDialog.ShowDialog(this) ==
        DialogResult.OK)
107             {
108                 string fileName = saveFileDialog.FileName;
109                 DBConnector db = DBConnector.Connect;
110                 db.Backup(fileName, String.Empty);
111             }
112         }
113         private void listViewMainMenu_ItemActivate(object
        sender, EventArgs e)
114         {
115             int Item =
        Convert.ToInt32(listViewMainMenu.FocusedItem.SubItems[1].Text);
116             DetailedInfo detailedInfoForm = new
        DetailedInfo(Item);
117             detailedInfoForm.Activate();
118             detailedInfoForm.ShowDialog(this);
119         }
120         private void FormMainMenu_FormClosing(object
        sender, FormClosingEventArgs e)
121         {
122             Controller.OnClose();
123         }
124         private void
        iconegrandiToolStripMenuItem_Click(object sender,
        EventArgs e)
125         {
126             this.listViewMainMenu.View = View.LargeIcon;
127             Properties.Settings.Default.MainMenuView =
        this.listViewMainMenu.View;
128             Properties.Settings.Default.Save();
129             this.iconegrandiToolStripMenuItem.Checked =
        true;
130             this.iconepiccoleToolStripMenuItem.Checked =
        false;
131             this.listaToolStripMenuItem.Checked = false;
132         }
133         private void
        iconepiccoleToolStripMenuItem_Click(object sender,
```

```
        EventArgs e)
134         {
135             this.listViewMainMenu.View = View.SmallIcon;
136             Properties.Settings.Default.MainMenuView =
        this.listViewMainMenu.View;
137             Properties.Settings.Default.Save();
138             this.iconegrandiToolStripMenuItem.Checked =
        false;
139             this.iconepiccoleToolStripMenuItem.Checked =
        true;
140             this.listaToolStripMenuItem.Checked = false;
141         }
142         private void listaToolStripMenuItem_Click(object
        sender, EventArgs e)
143         {
144             this.listViewMainMenu.View = View.List;
145             Properties.Settings.Default.MainMenuView =
        this.listViewMainMenu.View;
146             Properties.Settings.Default.Save();
147             this.iconegrandiToolStripMenuItem.Checked =
        false;
148             this.iconepiccoleToolStripMenuItem.Checked =
        false;
149             this.listaToolStripMenuItem.Checked = true;
150         }
151         private void nuovoToolStripMenuItem_Click(object
        sender, EventArgs e)
152         {
153             FormCreate createRecordForm = new FormCreate();
154             createRecordForm.Activate();
155             createRecordForm.ShowDialog(this);
156             updateListViewMainMenu();
157         }
158         private void modificaToolStripMenuItem_Click(object
        sender, EventArgs e)
159         {
160             try
161             {
162                 int Item =
        System.Convert.ToInt32(listViewMainMenu.FocusedItem.SubItems[1].Text);
163                 FormCreate modifyRecordForm = new
        FormCreate(Item);
164                 modifyRecordForm.Activate();
165                 modifyRecordForm.ShowDialog(this);
166                 updateListViewMainMenu();
167             }
168             catch(Exception)
169             {
170
171             }
172         }
173         private void eliminaToolStripMenuItem_Click(object
        sender, EventArgs e)
174         {
```

```
175             ListViewItem focusedItem =
    listViewMainMenu.FocusedItem;
176             if (focusedItem != null)
177             {
178                 int itemID =
    Convert.ToInt32(focusedItem.SubItems[1].Text);
179                 string message = "Sei sicuro di voler
    eliminare l'account " +
180                     focusedItem.Text + "?\n" +
181                     "Questa operazione non è annullabile
    una volta eseguita.";
182                 string caption = "Conferma cancellazione";
183                 MessageBoxButtons buttons =
    MessageBoxButtons.YesNo;
184                 DialogResult result;
185                 result = MessageBox.Show(this, message,
    caption, buttons);
186                 if (result == DialogResult.Yes)
187                 {
188                     try
189                     {
190                         DBConnector.Connect.Erase(itemID);
191                     }
192                     catch (DbException dbe)
193                     {
194                         message = "Errore durante
    cancellazione del record selezionato!\nErrore: " +
    dbe.Message;
195                         caption = "SQLite error n.: " +
    dbe.ErrorCode;
196                         buttons = MessageBoxButtons.OK;
197                         result = MessageBox.Show(this,
    message, caption, buttons);
198                     }
199                 }
200                 updateListViewMainMenu();
201             }
202         }
203         private void listViewMainMenu_MouseClick(object
    sender, MouseEventArgs e)
204         {
205             ListView listView = sender as ListView;
206             if (e.Button == MouseButtons.Right)
207             {
208                 ListViewItem item = listView.GetItemAt(e.X,
    e.Y);
209                 if (item != null)
210                 {
211                     item.Selected = true;
212                     item.Focused = true;
213                     this.contextMenuStrip1.Show(listView,
    e.Location);
214                 }
215             }
```

```
216         }
217         private void apriToolStripMenuItem_Click(object
    sender, EventArgs e)
218         {
219             int Item =
    Convert.ToInt32(listViewMainMenu.FocusedItem.SubItems[1].Text);
220             DetailedInfo detailedInfoForm = new
    DetailedInfo(Item);
221             detailedInfoForm.Activate();
222             detailedInfoForm.ShowDialog(this);
223         }
224         private void
    modificaToolStripMenuItem1_Click(object sender,
    EventArgs e)
225         {
226             int Item =
    Convert.ToInt32(listViewMainMenu.FocusedItem.SubItems[1].Text);
227             FormCreate modifyRecordForm = new
    FormCreate(Item);
228             modifyRecordForm.Activate();
229             modifyRecordForm.ShowDialog(this);
230             updateListViewMainMenu();
231         }
232         private void eliminaToolStripMenuItem1_Click(object
    sender, EventArgs e)
233         {
234             ListViewItem focusedItem =
    listViewMainMenu.FocusedItem;
235             int itemID =
    Convert.ToInt32(focusedItem.SubItems[1].Text);
236             string message = "Sei sicuro di voler eliminare
    l'account " + focusedItem.Text + "?\n" +
237                 "Questa operazione non è annullabile una
    volta eseguita.";
238             string caption = "Conferma cancellazione";
239             MessageBoxButtons buttons =
    MessageBoxButtons.YesNo;
240             DialogResult result;
241             result = MessageBox.Show(this, message,
    caption, buttons);
242             if (result == DialogResult.Yes)
243             {
244                 try
245                 {
246                     DBConnector.Connect.Erase(itemID);
247                 }
248                 catch (DbException dbe)
249                 {
250                     message = "Errore durante cancellazione
    del record selezionato!\nErrore: " + dbe.Message;
251                     caption = "SQLite error n.: " +
    dbe.ErrorCode;
252                     buttons = MessageBoxButtons.OK;
253                     result = MessageBox.Show(this, message,
```

```
caption, buttons);
254                 }
255             }
256             updateListViewMainMenu();
257         }
258         private void updateListViewMainMenu()
259         {
260             try
261             {
262                 this.listViewMainMenu.Items.Clear();
263                 var database = DBConnector.Connect;
264                 DbDataReader data = database.GetMainView();
265                 ImageList imageListSmall = new ImageList();
266                 imageListSmall.ImageSize = new Size(64, 64);
267                 ImageList imageListLarge = new ImageList();
268                 imageListLarge.ImageSize = new Size(128,
128);
269 
270                 int i = 0;
271                 while (data.Read())
272                 {
273 
imageListLarge.Images.Add(Image.FromFile("..\\..\\..\\Core\\Resources\\Pers\\ico
+ data["IMAGE"]));
274 
imageListSmall.Images.Add(Image.FromFile("..\\..\\..\\Core\\Resources\\Pers\\ico
+ data["IMAGE"]));
275                     var item =
this.listViewMainMenu.Items.Add(data["NAME"].ToString(),
data["NAME"].ToString(), i);
276 
item.SubItems.Add(data["ID"].ToString());
277 
item.SubItems.Add(data["URL"].ToString());
278                     i++;
279                 }
280 
281                 listViewMainMenu.LargeImageList =
imageListLarge;
282                 listViewMainMenu.SmallImageList =
imageListSmall;
283 
284                 if(this.listViewMainMenu.FocusedItem ==
null && listViewMainMenu.Items.Count > 0)
285                 {
286                     try
287                     {
288                         this.listViewMainMenu.FocusedItem =
this.listViewMainMenu.TopItem;
289                     }
290                     catch (InvalidOperationException) { }
291                 }
292             }
293             catch (DbException dbe)
```

```
294                 {
295                     throw dbe;
296                 }
297             }
298             private void nuovoToolStripMenuItem1_Click(object
        sender, EventArgs e)
299             {
300                 FormCreate createRecordForm = new FormCreate();
301                 createRecordForm.Activate();
302                 createRecordForm.ShowDialog(this);
303                 updateListViewMainMenu();
304             }
305             private void backupToolStripMenuItem1_Click(object
        sender, EventArgs e)
306             {
307                 SaveFileDialog saveFileDialog = new
        SaveFileDialog();
308                 saveFileDialog.AddExtension = true;
309                 saveFileDialog.DefaultExt = "sqlite3";
310                 saveFileDialog.Title = "Backup";
311                 saveFileDialog.InitialDirectory =
        Environment.GetFolderPath(Environment.SpecialFolder.Personal);
312                 saveFileDialog.Filter = "file di database
        SQLite (*.sqlite3)|*.sqlite3|Tutti i file (*.*)|*.*";
313                 if (saveFileDialog.ShowDialog(this) ==
        DialogResult.OK)
314                 {
315                     string fileName = saveFileDialog.FileName;
316                     DBConnector db = DBConnector.Connect;
317                     db.Backup(fileName, String.Empty);
318                 }
319             }
320         }
321 }
```

Codice 4.19: Classe FormModify.cs

```
1  using System;
2  using System.Collections.Generic;
3  using System.ComponentModel;
4  using System.Data;
5  using System.Drawing;
6  using System.Linq;
7  using System.Text;
8  using System.Threading.Tasks;
9  using System.Windows.Forms;
10
11 namespace N2L_Need_2_Log.gui
12 {
13     public partial class FormModify : Form
14     {
15         public FormModify(int itemID)
16         {
17             InitializeComponent();
```

```
18          }
19      }
20  }
```

Codice 4.20: Classe FormOption.cs

```
1   using System;
2   using System.Data.Common;
3   using System.Windows.Forms;
4   using Core;
5
6   namespace N2L_Need_2_Log.gui
7   {
8       /// <summary>
9       /// Classe che definisce il comportamento del form
        FormOption
10      /// </summary>
11      public partial class FormOption : Form
12      {
13          /// <summary>
14          /// Costruttore della classe FormOption
15          /// </summary>
16          public FormOption()
17          {
18              InitializeComponent();
19              this.textBoxUsername.Text = Controller.Username;
20          }
21
22          private void buttonCancel_Click(object sender,
        EventArgs e)
23          {
24              this.Close();
25          }
26          private void buttonOk_Click(object sender,
        EventArgs e)
27          {
28              if
        (!this.textBoxUsername.Text.Equals(Controller.Username))
29              {
30                  Controller.Username =
        this.textBoxUsername.Text;
31              }
32              if
        (!this.textBoxPassword.Text.Equals(this.textBoxPasswordConfirm.Text))
33              {
34                  string message = "Le password non
        corrispondono, impossibile settare/modificare la
        password.";
35                  string caption = "Errore in input!";
36                  MessageBoxButtons buttons =
        MessageBoxButtons.OK;
37                  DialogResult result;
38                  result = MessageBox.Show(this, message,
        caption, buttons);
```

```
39              }
40              else
41              {
42                  string previousPwd = Controller.Password;
43                  try
44                  {
45                      var database = DBConnector.Connect;
46                      if
    (String.IsNullOrEmpty(this.textBoxPassword.Text))
47                      {
48
    database.ChangeSettings(String.Empty);
49                      }
50                      else
51                      {
52
    database.ChangeSettings(Cript.ComputeHash(this.textBoxPassword.Text,
    null));
53
54                      }
55                      string message = "La password è stata
    correttamente settata/modificata.";
56                      string caption = "Operazione
    completata!";
57                      MessageBoxButtons buttons =
    MessageBoxButtons.OK;
58                      DialogResult result;
59                      result = MessageBox.Show(this, message,
    caption, buttons);
60                      if (result == DialogResult.OK)
61                      {
62                          this.Close();
63                      }
64                  }
65                  catch(DbException dbe)
66                  {
67                      string message = "Impossibile
    modificare la password! Errore con il DB.\n" +
    dbe.Message;
68                      string caption = "Operazione non
    riuscita!";
69                      MessageBoxButtons buttons =
    MessageBoxButtons.OK;
70                      DialogResult result;
71                      result = MessageBox.Show(this, message,
    caption, buttons);
72                      if (result == DialogResult.OK)
73                      {
74                          this.Close();
75                      }
76                  }
77              }
78
79          }
```

```
80          private void textBoxPassword_TextChanged(object
        sender, EventArgs e)
81          {
82              if(this.textBoxPassword.Text != String.Empty)
83              {
84                  this.textBoxPasswordConfirm.Enabled = true;
85              }
86              else
87              {
88                  this.textBoxPasswordConfirm.Enabled = false;
89              }
90              if (this.textBoxPassword.Text != String.Empty
        && this.textBoxPasswordConfirm.Text != String.Empty)
91              {
92                  if
        (this.textBoxPassword.Text.Equals(this.textBoxPasswordConfirm.Text))
93                  {
94                      this.labelPasswordError.Visible = false;
95                      this.labelPasswordConfirmed.Visible =
        true;
96                  }
97                  else
98                  {
99                      this.labelPasswordConfirmed.Visible =
        false;
100                     this.labelPasswordError.Visible = true;
101                 }
102             }
103             else
104             {
105                 this.labelPasswordConfirmed.Visible = false;
106                 this.labelPasswordError.Visible = false;
107             }
108
109
110         }
111         private void
        textBoxPasswordConfirm_TextChanged(object sender,
        EventArgs e)
112         {
113             if (this.textBoxPassword.Text != String.Empty
        && this.textBoxPasswordConfirm.Text != String.Empty)
114             {
115                 if
        (this.textBoxPassword.Text.Equals(this.textBoxPasswordConfirm.Text))
116                 {
117                     this.labelPasswordError.Visible = false;
118                     this.labelPasswordConfirmed.Visible =
        true;
119                 }
120                 else
121                 {
122                     this.labelPasswordConfirmed.Visible =
        false;
```

```
123                         this.labelPasswordError.Visible = true;
124                     }
125                 }
126                 else
127                 {
128                     this.labelPasswordConfirmed.Visible = false;
129                     this.labelPasswordError.Visible = false;
130                 }
131             }
132 
133 
134         }
135 }
```

4.4 Classi del progetto "*ConsoleTest*"

Classi presenti:

- N2LTest.cs

Codice 4.21: Classe N2LTest.cs

```
1  using System;
2  using System.IO;
3  using System.Net;
4  using Core;
5
6  namespace ConsoleTest
7  {
8      /// <summary>
9      /// Classe di test per verificare il corretto
       funzionamento del Core.
10     /// </summary>
11     internal class N2LTest
12     {
13         /// <summary>
14         /// Punto di ingresso principale del programma di
       test della libreria. Tale programma punta a testare
       tutte le funzionalità della DLL Core.
15         /// </summary>
16         /// <param name="args"></param>
17         internal static void Main(string[] args)
18         {
19             bool result;
20             Console.WriteLine("INIZIO DI TUTTI I TEST");
21             result = TestController();
22             Console.Write("Test su classe Controller: ");
23             Console.WriteLine((result) ? "OK" : "ERRORE");
24             if (result)
25             {
26                 result = TestCript();
27                 Console.Write("Test su classe Cript: ");
28                 Console.WriteLine((result) ? "OK" :
       "ERRORE");
29                 if (result)
30                 {
31                     result = TestDBConnector();
32                     Console.Write("Test su classe
       DBConnector: ");
33                     Console.WriteLine((result) ? "OK" :
       "ERRORE");
34                 }
35             }
36             Console.WriteLine((result) ?
37                 "TUTTI I TEST SONO STATI SUPERATI" :
38                 "ERRORE RISCONTRATO IN UNO DEI TEST");
39             Console.WriteLine("PREMI INVIO PER TERMINARE");
40             Console.ReadKey();
41         }
```

```
42
43          /// <summary>
44          /// Metodo di supporto ai test. Tramite Internet
        produce una parola in maniera randomica.
45          /// </summary>
46          /// <returns></returns>
47          protected static string GetRequest()
48          {
49              try
50              {
51                  string toReturn = String.Empty;
52                  Stream objStream;
53                  StreamReader objReader;
54                  WebRequest wr =
        WebRequest.Create("http://setgetgo.com/randomword/get.php");
55                  WebProxy myProxy = new WebProxy();
56
57                  myProxy.BypassProxyOnLocal = true;
58                  wr.Proxy = myProxy;
59                  objStream =
        wr.GetResponse().GetResponseStream();
60                  objReader = new StreamReader(objStream);
61                  toReturn = objReader.ReadLine();
62
63                  return toReturn;
64              }
65              catch (Exception)
66              {
67                  return "ErroreInGetRequest";
68              }
69          }
70          /// <summary>
71          /// Serie di test per verificare il corretto
        funzionamento della classe Controller.
72          /// </summary>
73          /// <returns> true se il db viene/è eliminato,
        altrimenti false</returns>
74          protected static bool TestController()
75          {
76              Core.Controller.CheckSettings();
77              bool DbExist = Core.Controller.DbExist;
78              if (Core.Controller.DbExist)
79              {
80                  Console.Write("ELiminazione vecchio
        Database: ");
81
        System.IO.File.Delete("..\\..\\..\\Core\\Resources\\Db\\DBN2L.sqlite3");
82                  Console.WriteLine("OK");
83                  Controller.CheckSettings();
84                  DbExist = Core.Controller.DbExist;
85              }
86              return !Core.Controller.DbExist;
87          }
88          /// <summary>
```

```
89          /// Serie di test per verificare il corretto
    funzionamento della classe Cript.
90          /// </summary>
91          /// <returns> true se la classe risponde
    correttamente, altrimenti false</returns>
92          protected static bool TestCript()
93          {
94              Random r = new Random();
95              int i = r.Next(8, 16);
96              int nNAC = r.Next(0, i);
97              string testWord = GetRequest();
98              string hash = Cript.ComputeHash(testWord, null);
99              bool result;
100
101             Console.WriteLine("\t\"Cript.ComputeHash(\"" +
    testWord + "\", null)\"->" + hash);
102             result = Cript.Confirm(testWord, hash);
103             Console.WriteLine("\t\"Cript.Confirm(\"" +
    testWord + "\", " + hash + "\")\"->" + result);
104             Console.WriteLine("\t\"Cript.GenerateRandomly("
105                 + i + ", " + nNAC + ")\"->\"" +
    Cript.GenerateRandomly(i, nNAC) + "\"");
106             return result;
107         }
108         /// <summary>
109         /// Serie di test per verificare il corretto
    funzionamento della classe DBConnector.
110         /// </summary>
111         /// <returns> true se la classe risponde
    correttamente, altrimenti false</returns>
112         protected static bool TestDBConnector()
113         {
114             Random r = new Random();
115             string password = GetRequest();
116             bool toReturn = true;
117             int entryId;
118             string entryName;
119             int typeId;
120             string iconName;
121             string noteValue;
122             string passwordValue;
123             string urlValue;
124             string usernameValue;
125             int result;
126             int j;
127             DBConnector db = DBConnector.Connect;
128
129             Console.Write("\tcreazione e connessione db: ");
130             if (db == null)
131             {
132                 toReturn = false;
133                 Console.WriteLine("FALLITO");
134             }
135             else
```

```
136             {
137                 if (db.isConnected ==
    System.Data.ConnectionState.Open)
138                     Console.WriteLine("OK");
139                 else
140                 {
141                     toReturn = false;
142                     Console.WriteLine("FALLITO");
143                 }
144             }
145             if (toReturn)
146             {
147                 Console.Write("\tCambio password in \"" +
    password + "\": ");
148                 Core.Controller.Password =
    Core.Cript.ComputeHash(password, null);
149                 db.ChangeSettings(Core.Controller.Password);
150                 Console.WriteLine("OK");
151                 Console.Write("\tDisconnessione: ");
152                 if (db.Disconnect ==
    System.Data.ConnectionState.Closed)
153                 {
154                     Console.WriteLine("OK");
155                 }
156                 else
157                 {
158                     toReturn = false;
159                     Console.WriteLine("FALLITO");
160                 }
161             }
162             if (toReturn)
163             {
164                 db = null;
165                 db = DBConnector.Connect;
166                 Console.Write("\tConessione: ");
167                 if (db != null)
168                 {
169                     if (db.isConnected ==
    System.Data.ConnectionState.Open)
170                     {
171                         Console.WriteLine("OK");
172                     }
173                 }
174                 else
175                 {
176                     toReturn = false;
177                     Console.WriteLine("FALLITO");
178                 }
179             }
180             if (toReturn)
181             {
182                 j = 0;
183                 Console.Write("\t\"GetRecordsType()\":");
184                 using (System.Data.Common.DbDataReader data
```

```
= db.GetRecordsType())
185                 {
186                     while (data.Read())
187                     {
188                         j++;
189                     }
190                     if (j > 0)
191                     {
192                         Console.WriteLine("OK");
193                     }
194                     else
195                     {
196                         toReturn = false;
197                         Console.WriteLine("FALLITO");
198                     }
199                 }
200                 if (toReturn)
201                 {
202                     Console.Write("\tCreazione di tre nuovi
contatti: ");
203                     for (int i = 0; i < 3; i++)
204                     {
205                         entryName = GetRequest();
206                         typeId = r.Next(1, j);
207                         noteValue = GetRequest();
208                         passwordValue = GetRequest();
209                         urlValue = "http://www.test.org";
210                         usernameValue = GetRequest();
211                         usernameValue =
db.CheckEntryName(usernameValue);
212                         result = db.Insert(entryName,
typeId, noteValue, passwordValue, urlValue,
usernameValue);
213                         Console.Write((i + 1) + " | ");
214                     }
215                     j = 0;
216                     using (System.Data.Common.DbDataReader
data = db.GetMainView())
217                     {
218                         while (data.Read())
219                         {
220                             j++;
221                         }
222                     }
223                     if (j > 0)
224                     {
225                         Console.WriteLine("OK");
226                     }
227                     else
228                     {
229                         toReturn = false;
230                         Console.WriteLine("FALLITO");
231                     }
232                 }
```

```
233             }
234             if (toReturn)
235             {
236                 j = 0;
237                 Console.Write("\tModifica di un elemento:
    ");
238                 entryId = r.Next(0, j);
239                 entryName = String.Empty;
240                 typeId = 1;
241                 iconName = String.Empty;
242                 noteValue = GetRequest();
243                 passwordValue = GetRequest();
244                 urlValue = "http://www." + GetRequest() +
    ".org";
245                 usernameValue = GetRequest();
246                 result = db.Update(entryId, entryName,
    typeId, iconName, noteValue, passwordValue, urlValue,
    usernameValue);
247                 for (int i = 1; i <= 3; i++)
248                 {
249                     using (System.Data.Common.DbDataReader
    data = db.GetRecordInfo(i))
250                     {
251                         while (data.Read())
252                         {
253                             j++;
254                         }
255                     }
256                 }
257                 if (j == 3)
258                 {
259                     Console.WriteLine("OK");
260                 }
261                 else
262                 {
263                     Console.WriteLine("FALLITO");
264                     toReturn = false;
265                 }
266             }
267             if (toReturn)
268             {
269                 Console.Write("\tElimino un elemento: ");
270                 if (db.Erase(r.Next(1, 3)) > 0)
271                 {
272                     Console.WriteLine("OK");
273                 }
274                 else
275                 {
276                     toReturn = false;
277                     Console.WriteLine("FALLITO");
278                 }
279             }
280             if (toReturn)
281             {
```

```
282                 Console.Write("\tCreo database di backup:
    ");
283                 if
    (File.Exists("..\\..\\..\\Core\\Resources\\Db\\backup.sqlite3"))
284
    File.Delete("..\\..\\..\\Core\\Resources\\Db\\backup.sqlite3");
285
    db.Backup("..\\..\\..\\Core\\Resources\\Db\\backup.sqlite3",
    null);
286                 if
    (File.Exists("..\\..\\..\\Core\\Resources\\Db\\backup.sqlite3"))
287                 {
288
    File.Delete("..\\..\\..\\Core\\Resources\\Db\\backup.sqlite3");
289                     Console.WriteLine("OK");
290                 }
291                 else
292                 {
293                     Console.WriteLine("FALLITO");
294                     toReturn = false;
295                 }
296             }
297             return toReturn;
298         }
299     }
300 }
```

Capitolo 5

Test

5.1 Test durante lo sviluppo

Per accertarsi che non vi fossero problemi in fase di compilazione ed assemblaggio, durante tutte le fasi di sviluppo si è proceduto costantenmente con i test di ogni classe e funzionalità, soprattutto per tutte quelle responsabili delle comunicazioni con il database.
Tale sistema ha permesso di individuare errori che sarebbero stati difficili da rilevare in altre situazioni, velocizzando drasticamente le fasi di test e rendendo l'applicazione, dunque, più rapidamente rilasciabile e "*bugless*".

5.2 Test White-Box

Tale tipologia di test si basa sull'utilizzo della conoscenza del codice in modo tale da poter riuscire a coprire ogni singolo possibile flusso di esecuzione.
Questo genere di test viene eseguito tramite il progetto "*ConsoleTest*", all'interno del quale sono stati sviluppati dei metodi il cui scopo è quello di testare ogni singola classe presente all'interno della libreria *Core* (pag. 89).
Nessun test ha riscontrato errori nel funzionamento della libreria, pertanto si può affermare che la libreria funziona correttamente.

5.3 Test Black-Box

Tale tipologia di test si basa sull'utilizzo dell'applicativo senza però essere a conoscenza della logica funzionale presente dietro.
I test black-box servono per individuare:

- *Errori nella GUI*:
  non sono stati individuati errori nella interfaccia grafica.
- *Errori nelle performance*:
  le performance sono risultate essere più che accettabili in tutte le macchine su cui l'applicativo è stato testato.
- *Errori di terminazione/crash*:
  non sono stati riscontrati crash inattesi in nessuna fase di utilizzo e test.

Il test black-box inoltre viene utilizzato per valutare i casi d'uso.
A tal proposito, verrano mostrati quattro test specifici per i casi d'uso **UC1**, **UC2**, **UC4** e **UC5**.

5.3.1 Test UC1 - Login

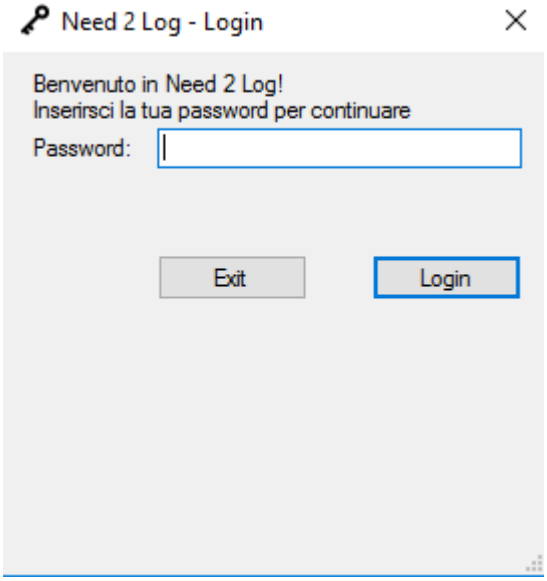

Figura 5.1: Prima screenshot del test per UC1

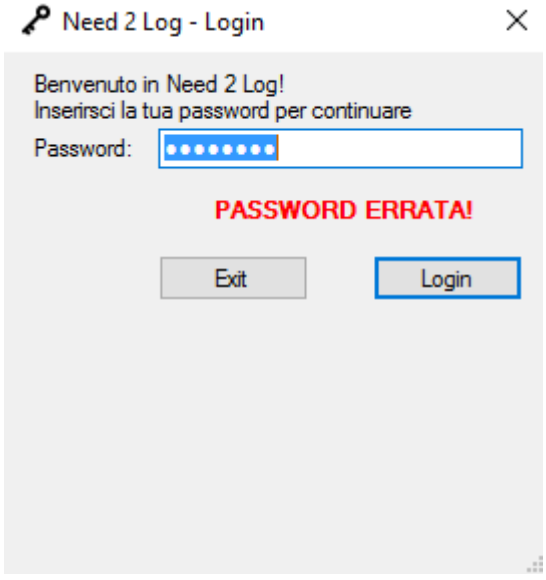

Figura 5.2: Seconda screenshot del test per UC1

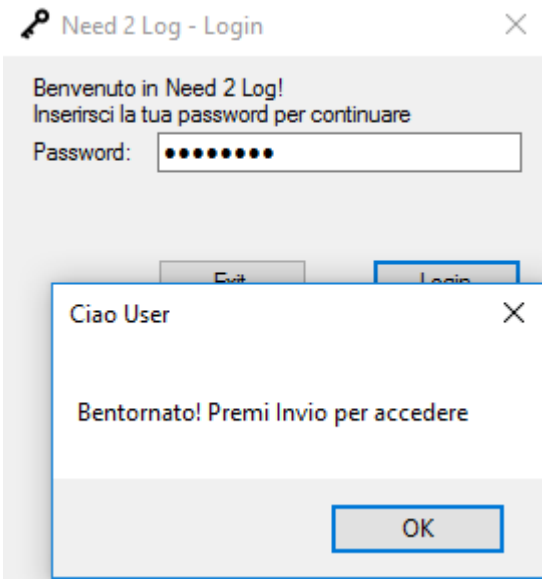

Figura 5.3: Terza screenshot del test per UC1

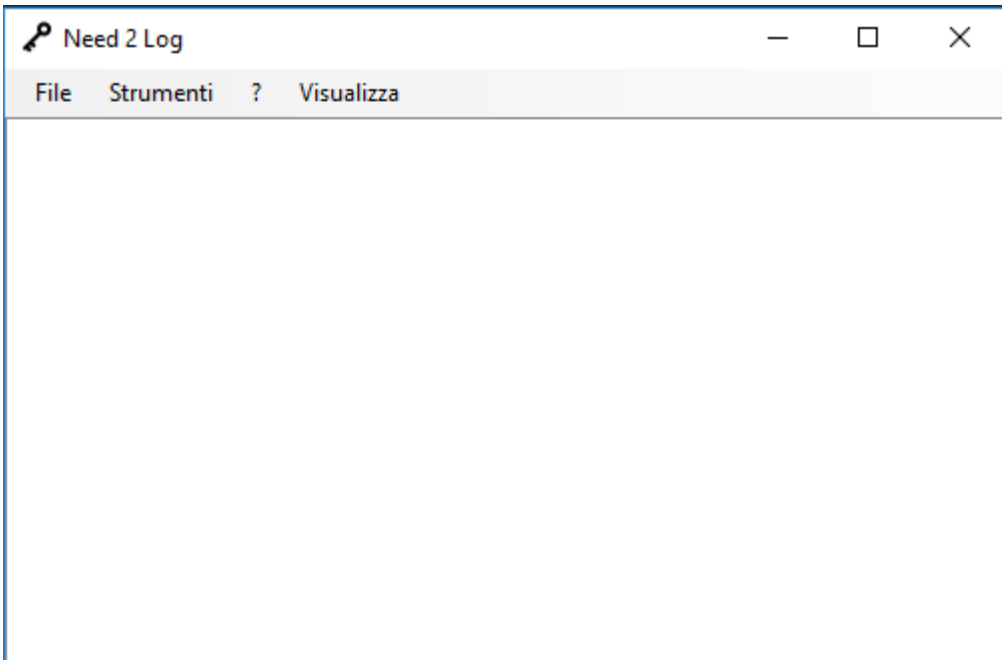

Figura 5.4: Quarta screenshot del test per UC1

5.3.2 Test UC2 - New account

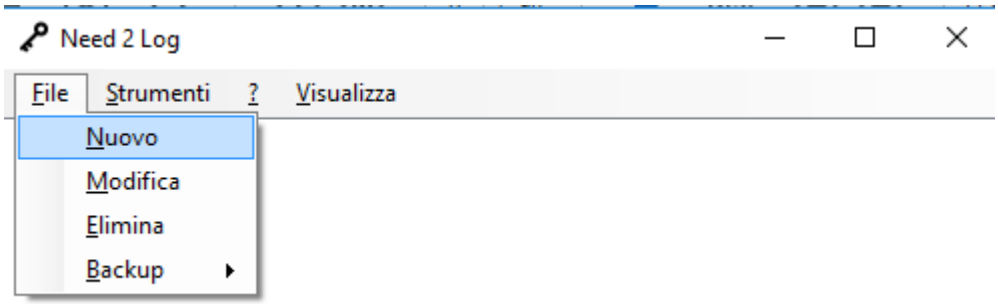

Figura 5.5: Prima screenshot del test per UC2

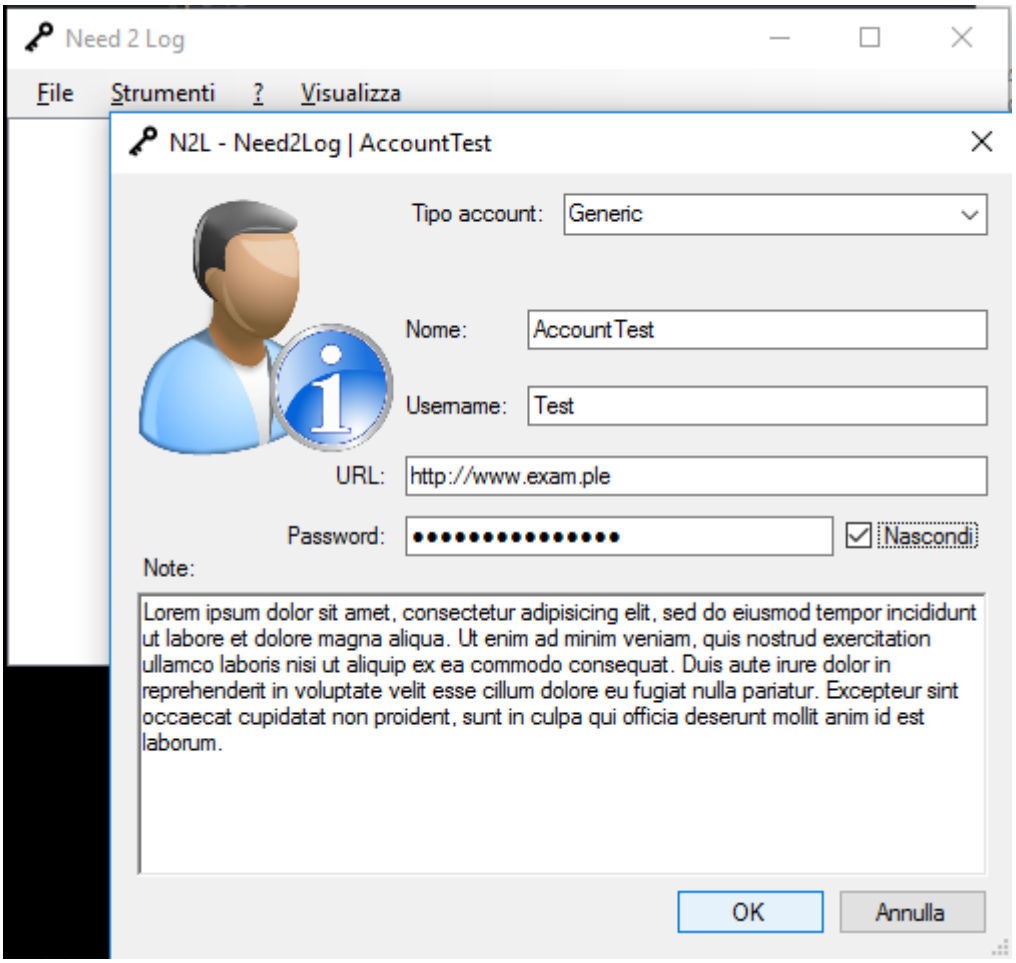

Figura 5.6: Seconda screenshot del test per UC2

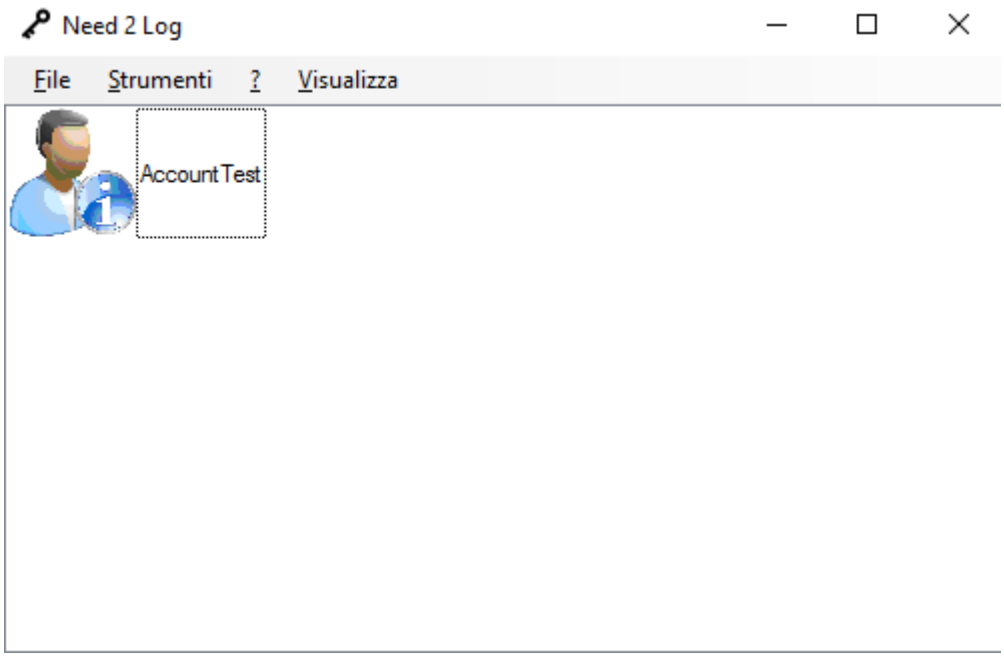

Figura 5.7: Terza screenshot del test per UC2

5.3.3 Test UC4 - Delete account

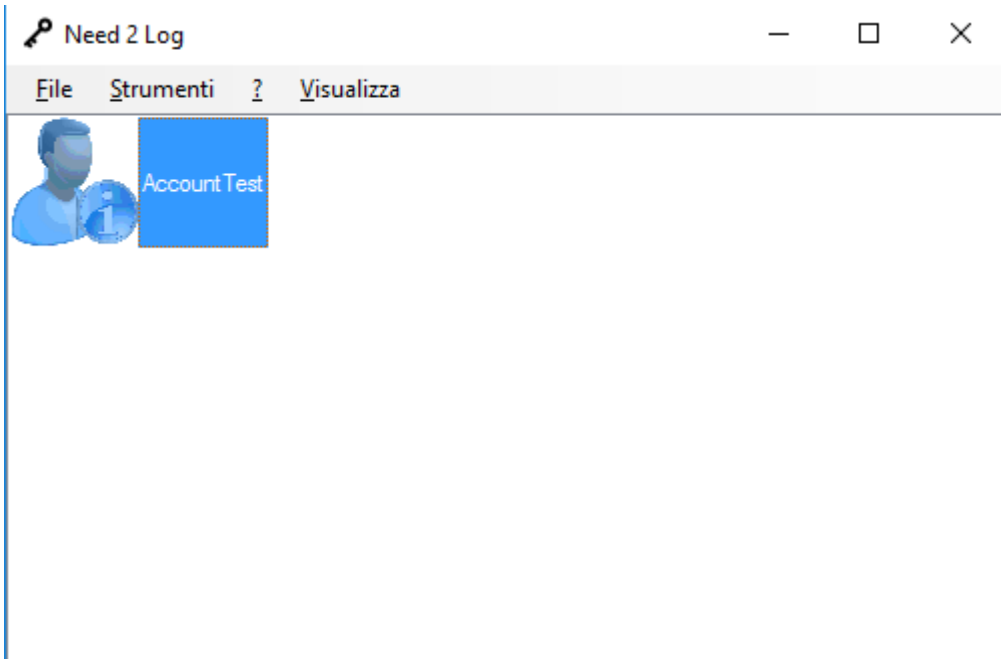

Figura 5.8: Prima screenshot del test per UC4

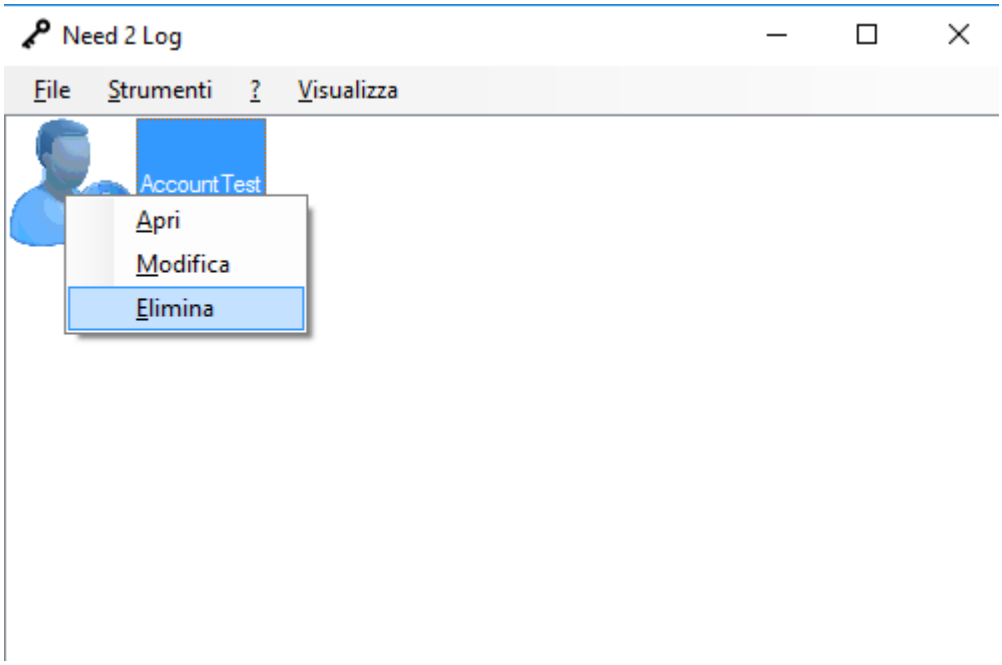

Figura 5.9: Seconda screenshot del test per UC4

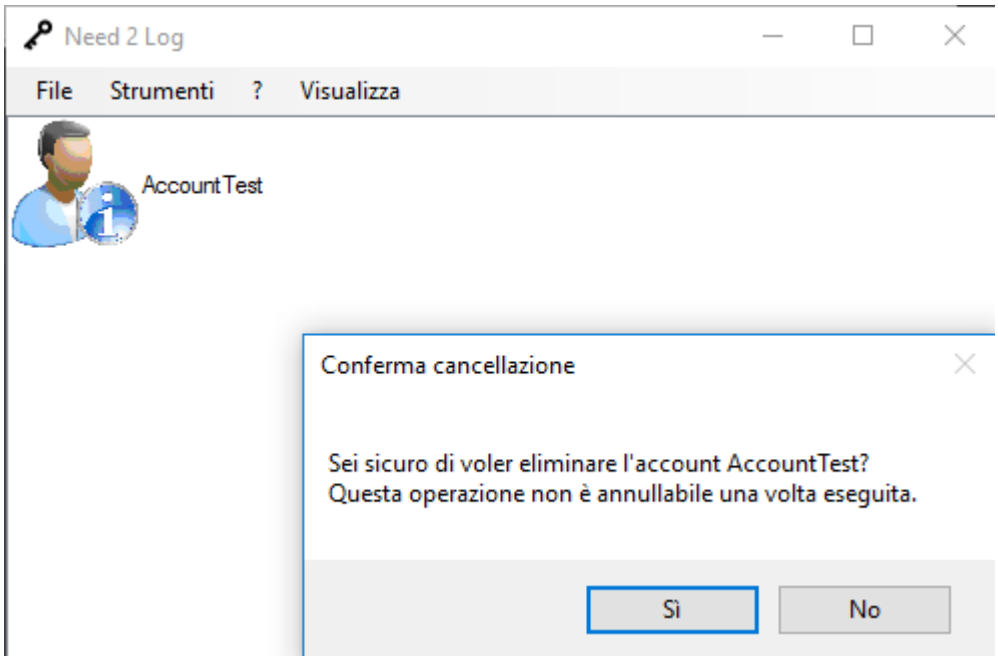

Figura 5.10: Terza screenshot del test per UC4

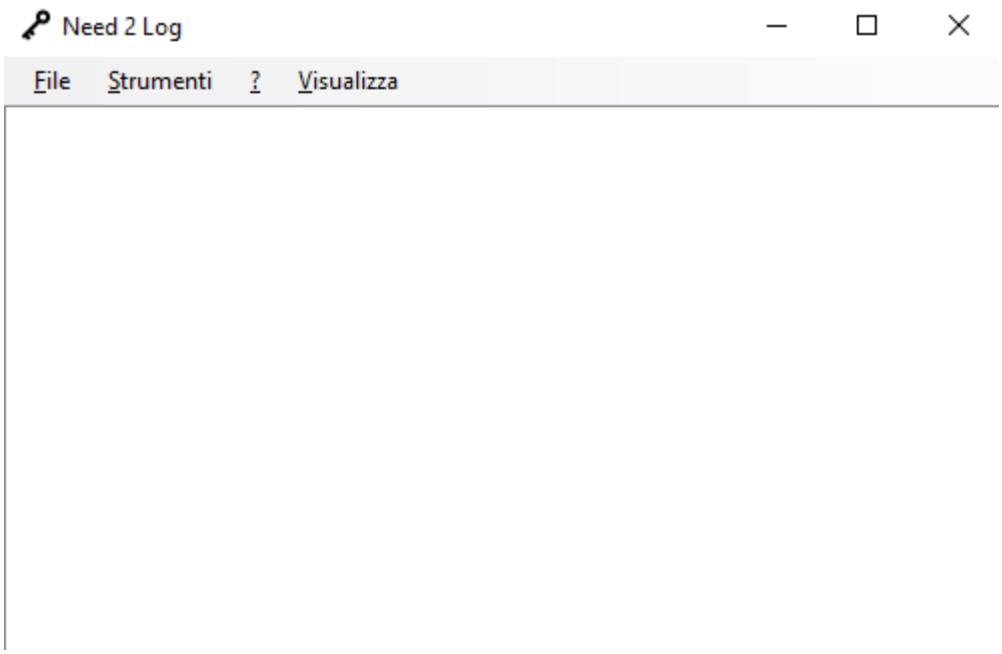

Figura 5.11: Quarta screenshot del test per UC4

5.3.4 Test UC5 - Show account

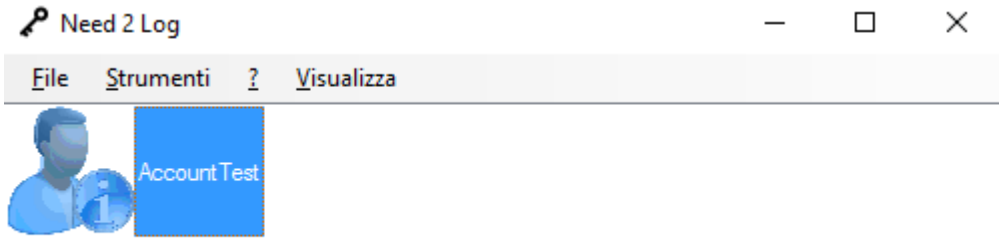

Figura 5.12: Prima screenshot del test per UC5

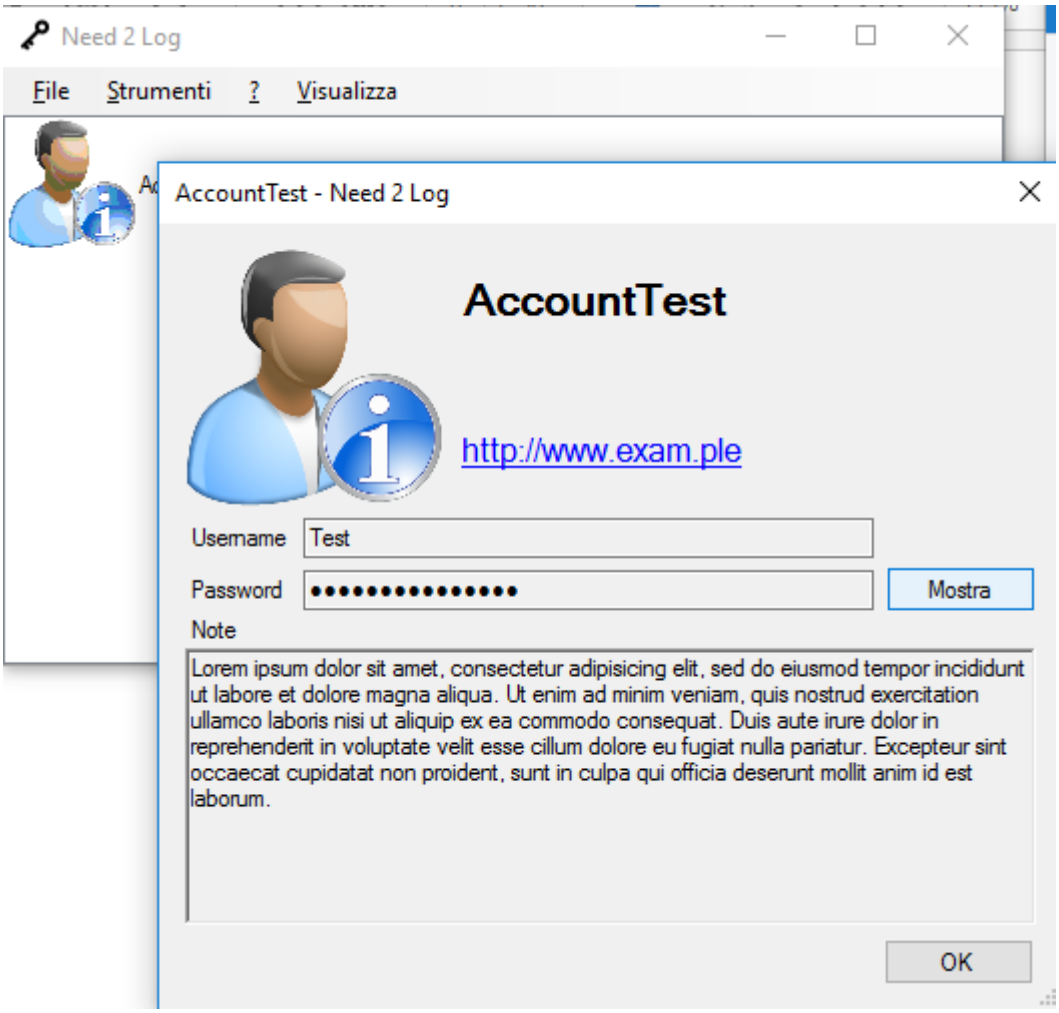

Figura 5.13: Seconda screenshot del test per UC5

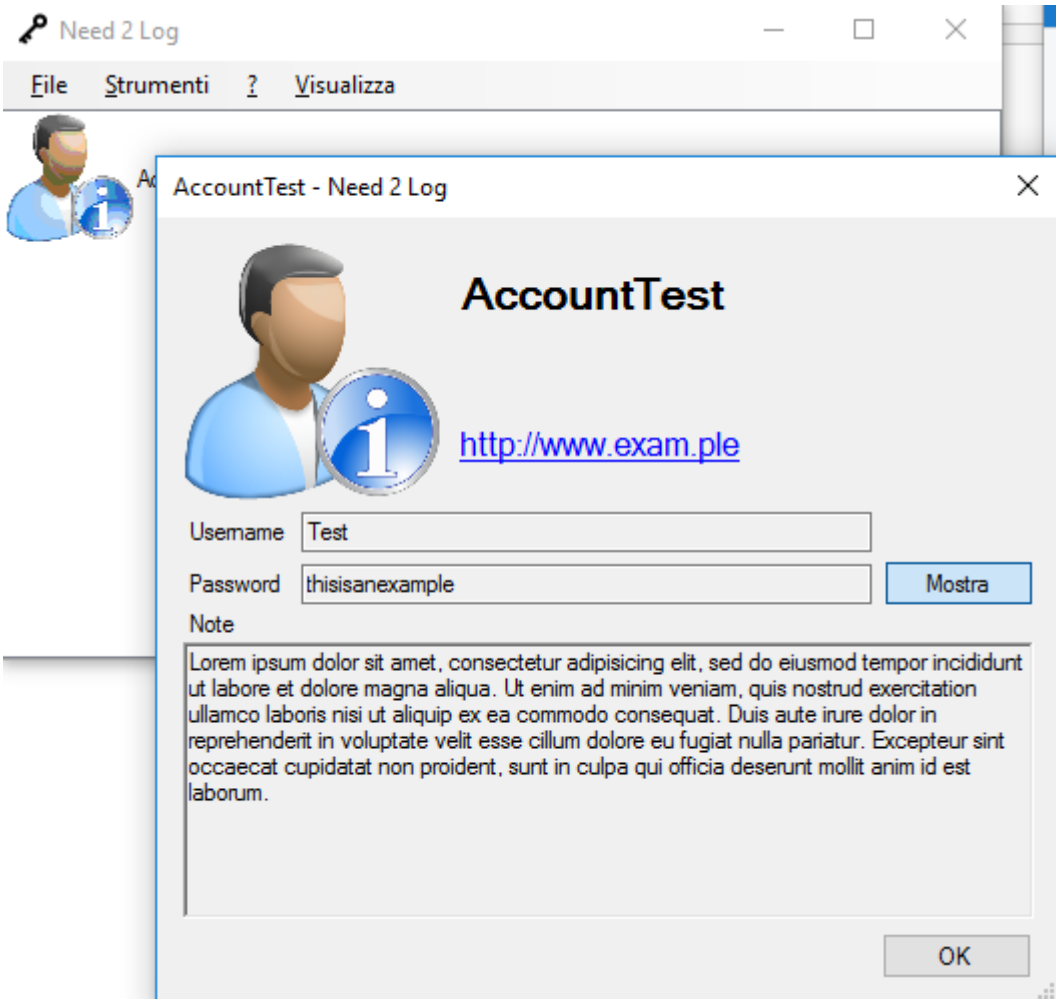

Figura 5.14: Terza screenshot del test per UC5

Capitolo 6

Compilazione ed esecuzione

6.1 Requisiti di sistema

Poiché l'applicazione non risulta onerosa in termini di potenza computazionale e di memoria occupata, i requisiti minimi coincidono con quelli di tutte le applicazioni sviluppate con il framework .NET.

6.1.1 Requisiti minimi

- Processore 1 GHz o superiore con architettura x86 o x86_64
- 512 Mb di memoria principale
- 600 Mb di spazio su disco per sistemi a 32 bit
- 1.5 Gb di spazio su disco per sistemi a 64 bit
- Sistema operativo Microsoft Windows 7 SP1 o successivo
- Piattaforma .NET 4.5.2 installata.

6.1.2 Requisiti consigliati

- Processore 1 GHz o superiore con architettura x86 o x86_64
- 512 Mb di memoria principale
- 850 Mb di spazio su disco per sistemi a 32 bit
- 2 Gb di spazio su disco per sistemi a 64 bit
- Sistema operativo Microsoft Windows 7 SP1 o successivo per ambienti desktop
- Sistema operativo Microsoft Windows Server 2008 SP2 o successivo per ambienti server
- Piattaforma .NET 4.6 installata.

Si sono effettuati test su varie macchine, confermando empiricamente che i requisiti sopra descritti risultano sufficienti all'esecuzione senza crash o rallentamenti.

6.2 Compilazione con Visual Studio

1. Navigare fino alla cartella *Need2Log* ed accedervi;
2. Avviare il file *N2L.sln* con *Visual Studio*;
3. Verificare che la configurazione sia impostata su *Release*;

4. Selezionare la voce di menù *Compila*→ *CompilaSoluzione*

6.3 Esecuzione

1. Dalla cartella *Need2Log*, navigare nel percorso *Need2Log/N2L-Need_2_Log/bin/Release*
2. Fare doppio click sul file eseguibile "*Need 2 Log.exe*"

L'applicativo è stato testato su vari elaboratori con differenti sistemi operativi e differenti configurazioni hardware per verificare se il comportamento del programma mutasse (anche a livello di performance), ma non sono state riscontrate differenze nel funzionamento.
Nello specifico, l'applicazione è stata testata su:

Sistema operativo	**RAM (GB)**	**CORE**	**Architettura**
Windows 7	4 GB	2 Core	64 bit
Windows 8.1	8 GB	4 Core	64 bit
Windows 10	5 GB	4 Core	64 bit
Windows 10	8 GB	8 Core	64 bit

Tabella 6.1: Specifiche delle macchine su cui il programma è stato eseguito