UNIVERSITÀ DEGLI STUDI DI TRENTO

Progetto “InTouch”

Documento finale
Corso di Programmazione ad Oggetti
A.A. 2014/2015

SDM Team

Sommario

Informazioni sul gruppo

Nome del gruppo: **SDM Team**

Componenti:

- **Lebedev Sergey** *matricola 166036*
- **Pernpruner Marco** *matricola 164782*
- **Rocco Davide** *matricola 165680*

Responsabile: **Pernpruner Marco**

Informazioni sul documento

Data di termine stesura: **XX/01/2015**

Versione: **0.1**

Informazioni sul progetto

Idea progettuale

L'idea di realizzare il progetto in questione è nata prendendo spunto dal già esistente social network *Facebook*.

Di seguito una breve descrizione della piattaforma in questione:

> *"Gli utenti possono accedere al sito previa una registrazione gratuita, durante la quale vengono richiesti dati personali come nome, cognome, data di nascita e indirizzo email. Il sito chiarisce che l'inserimento obbligatorio della data di nascita serve esclusivamente "per favorire una maggiore autenticità e consentire l'accesso ai vari contenuti in base all'età". Completata la registrazione, gli utenti possono creare un profilo personale, includere altri utenti nella propria rete sociale, aggiungendoli come amici, e scambiarsi messaggi, anche via chat, incluse le notifiche automatiche quando questi aggiornano i propri profili.*
>
> *Inoltre gli utenti possono fondare e unirsi a gruppi per condividere interessi in comune con altri utenti, organizzati secondo il luogo di lavoro, la scuola, l'università o altre caratteristiche, condividere contenuti multimediali ed utilizzare varie applicazioni presenti sul sito. Per personalizzare il proprio profilo l'utente può caricare una foto, chiamata immagine del profilo, con la quale può rendersi riconoscibile. Può inoltre fornire ulteriori informazioni, come il comune di nascita (esempio: Città natale: Roma) e quello di residenza (esempio: Vive a Torino), la scuola frequentata, il proprio datore di lavoro, l'orientamento religioso e quello politico, la propria situazione sentimentale e molte altre."*
>
> *(dalla voce Facebook di Wikipedia, l'enciclopedia libera)*

Al fine di adattare il lavoro alle risorse disponibili all'interno del gruppo, sono state naturalmente vagliate attentamente le funzionalità da implementare.

Descrizione

Il progetto si propone di realizzare una piattaforma sociale in grado di far interagire tra loro gli utenti iscritti.

Per poter utilizzare attivamente l'applicazione è necessario registrarsi. Chiunque lo desideri può farlo, selezionando l'apposita funzione nella schermata di autenticazione; per la procedura di registrazione, è necessario inserire il proprio nome, cognome, indirizzo *email* e la *password* personale scelta, che dovrà essere immessa ad ogni successivo *login*.

Una volta registrati, selezionando nuovamente l'apposita funzione nella schermata di autenticazione, è possibile eseguire il *login* inserendo il proprio indirizzo *email* – trattato dal programma come *username* univoco identificativo di ogni utenza – e la propria *password*, così come immessi all'atto della registrazione.

Verificata la corrispondenza tra *username* e *password*, il sistema permetterà all'utente di accedere al proprio menu principale, da cui potrà scegliere quali azioni intraprendere.

Ogni utente può effettuare richieste di "amicizia" ad altri utenti; una volta confermata tale richiesta da parte del destinatario nell'apposita sezione, i due utenti in questione – che diventeranno "amici" – saranno

in grado di visualizzare informazioni e pubblicazioni reciproche. Naturalmente, è anche possibile rifiutare richieste di "amicizia" nel caso in cui non si vogliano condividere le proprie informazioni con l'utente richiedente; in questo caso non sarà più possibile stringere amicizia con quel determinato utente.

Gli utenti hanno la possibilità di creare stati personali per condividere pensieri o emozioni con i propri amici. Tali stati – denominati "*post*" – verranno visualizzati da tutti gli utenti presenti nella lista di amici dell'autore all'interno della bacheca di quest'ultimo e della loro bacheca generale.

In particolare, accedendo alla propria bacheca generale gli utenti possono visualizzare tutti i *post* dei propri "amici", insieme al nominativo dell'utente autore del *post*, alla data e ora di pubblicazione, ad eventuali commenti e all'elenco di eventuali altri utenti che hanno indicato il proprio apprezzamento per il *post* in questione.

Qualunque utente può infatti apporre un proprio commento o indicare il proprio apprezzamento ad uno qualsiasi dei *post* dei propri "amici".

Ogni utente può inoltre aggiungere informazioni personali che saranno visualizzati da tutti i propri amici sul proprio profilo, ad esempio data di nascita, sesso, professione e situazione sentimentale; nel caso in cui l'utente non modifichi queste informazioni, esse resteranno "Non definite" di *default.*

Requisiti

Requisito #1: Funzionale

L'applicazione deve permettere all'utente di:

- Registrarsi ed autenticarsi nella piattaforma
- Gestire le proprie "amicizie" con altri utenti
- Visualizzare e gestire la propria bacheca e le proprie informazioni personali
- Visualizzare la bacheca e le informazioni personali di un amico
- Visualizzare tutti i post pubblicati dai propri amici con relativi commenti e apprezzamenti
- Pubblicare *post* personali
- Commentare ed esprimere il proprio apprezzamento a *post* dei propri "amici"

Requisito #2: Funzionale

L'applicazione deve eseguire un controllo sul carattere immesso ogni qualvolta viene richiesto di effettuare una delle scelte presentate da un menu. In particolare, deve impedire l'inserimento di caratteri non numerici e/o non inclusi nel *range* di scelte disponibili.

Requisito #3: Funzionale

L'applicazione non deve permettere all'utente l'immissione di determinati caratteri speciali all'interno dell'*indirizzo email* indicato in fase di registrazione. Dal momento che l'applicazione crea una cartella per ogni utente, avente come nome proprio l'indirizzo *email*, la presenza negli stessi di caratteri speciali non consentiti da *Windows* come nomi di cartelle causerebbe problemi di funzionamento.

Requisito #4: Funzionale

L'applicazione non deve permettere all'utente l'immissione di caratteri utilizzati come delimitatori nei *file* di testo, al fine di impedire l'erronea lettura delle informazioni in fase di importazione iniziale. A tal fine, è stato modificato il formato dei file da *.csv* (*Comma Separated Values*) a *.tsv* (*Tab Separated Values*), dal momento che l'utilizzo di virgole e punti e virgola all'interno di campi testuali è alquanto frequente.

Requisito #5: Funzionale

L'applicazione deve memorizzare in un *database* (*file* di testo, in questo caso) i dati inseriti nel corso di una sessione, importandoli poi all'inizio di sessione successiva al fine di non perdere alcun dato.

Requisito #6: Temporale

Il gruppo SDM Team deve consegnare tempestivamente:

- Il documento relativo all'idea progettuale
- Il documento di analisi
- Il documento di progettazione

- Il documento finale con il codice sorgente definitivo

Requisito #7: Economico

Il progetto viene realizzato dal gruppo senza alcun introito economico, solamente ai fini didattico-culturali relativi al corso di Programmazione ad Oggetti.

Requisito #8: Livello di Servizio

L'applicazione deve essere funzionante e progettata per essere teoricamente sempre accessibile.

Requisito #9: Organizzativo

Il gruppo SDM Team deve collaborare per la stesura del codice, provvedendo ad allocare – in fase di progettazione – ad ogni attività le opportune risorse, compatibilmente con la complessità delle mansioni.

Sono inoltre previste frequenti riunioni tra i membri del gruppo per provvedere alla stesura dei vari documenti, oltre che per delineare lo stato complessivo dei lavori.

Requisito #10: di Sicurezza

Il progetto non si propone di attuare alcuna procedura di sicurezza per proteggere dati sensibili relativi agli utenti. Al contrario, dato l'utilizzo ai fini didattici di file *.tsv* come *database*, i dati degli utenti saranno visibili in chiaro sul file di testo.

Tuttavia, l'applicazione deve comunque essere sicura e non arrecare danni al sistema sul quale viene avviata.

Inoltre, viene garantita la riservatezza in fase di immissione della *password* all'atto del *login*, sostituendo ogni carattere digitato con un asterisco (*).

Requisito #11: Tecnologico

L'utente deve avere a disposizione la seguente strumentazione:

- Computer con sistema operativo Windows 7/8
- Monitor
- Mouse
- Tastiera italiana

N.B.: Dal momento che per la creazione e la rimozione di file e cartelle vengono utilizzate alcune funzioni come *mkdir()* e *rmdir()* che fanno riferimento a specifici comandi di *Windows*, per il corretto funzionamento dell'applicazione è fondamentale l'utilizzo di tale sistema operativo.

Requisito #12: di Utilizzo

L'applicazione deve essere facilmente comprensibile ed utilizzabile da qualunque utente lo desideri.

Deve inoltre essere fluida e non causare rallentamenti.

Cambiamenti rispetto al documento di progettazione

Elementi modificati

- Descrizione → aggiunta nota sul fatto che se una richiesta di amicizia viene rifiutata non sarà più possibile stringere amicizia con quel determinato utente
- Requisiti → aggiunto #3
- Casi d'uso → aggiunti casi d'uso 6-16-17, corretti altri
- Schermate di interazione → aggiornate in base alla versione finale del codice
- Pianificazione delle attività → inserite le durate reali relative alla fase di sviluppo e *testing*
- Durate totali attività → aggiunte
- Diagrammi di Gantt e PERT → adattati alle durate reali delle attività di sviluppo e *testing*
- Diagramma ad oggetti → aggiunti attributi e metodi derivanti dall'avanzamento nello sviluppo
- Basi di dati → modificato il formato dei *file*, aggiornate e descritte in modo più dettagliato
- Librerie → aggiunta libreria *config.h*
- Bibliografia → aggiunto riferimento a descrizione file *.tsv*

Pianificazione delle attività

Attività	Durata stimata		Durata reale		Assegnatario
	Effort	Totale	*Effort*	Totale	
Analisi					
Stesura del documento dei requisiti	8 ore	5 giorni	10 ore	5 giorni	Tutti
Realizzazione diagramma della classi UML	5 ore	2 giorni	3 ore	3 giorni	Tutti
Realizzazione diagramma dei casi d'uso UML	2 ore	1 giorno	4 ore	2 giorni	Pernpruner
Stesura del documento di analisi	7 ore	2 giorni	6 ore	2 giorni	Tutti
Realizzazione del diagramma di Gantt	3 ore	1 giorno	3 ore	1 giorno	Pernpruner, Rocco
Realizzazione del diagramma di PERT	3 ore	1 giorno	3 ore	1 giorno	Pernpruner, Rocco
Progettazione					
Definizione classe "InTouch"	2 ore	1 giorno	2 ore	1 giorno	Pernpruner
Definizione classe "Utente"	4 ore	3 giorni	2 ore	2 giorni	Pernpruner
Definizione classe "Bacheca"	2 ore	2 giorni	2 ore	1 giorno	Rocco
Definizione classe "Profilo"	2 ore	2 giorni	2 ore	1 giorno	Rocco
Definizione classe "Post"	3 ore	2 giorni	2 ore	1 giorno	Lebedev
Definizione classe "Commento"	3 ore	2 giorni	2 ore	1 giorno	Lebedev
Definizione classe "Amicizia"	2 ore	2 giorni	3 ore	2 giorni	Tutti
Realizzazione del documento di progettazione	10 ore	3 giorni	25 ore	11 giorni	Tutti
Sviluppo					
Implementazione classe "InTouch"	2 ore	1 giorno	1 ora	1 giorno	Pernpruner
Implementazione classe "Utente"	2 ore	1 giorno	3 ore	1 giorno	Pernpruner
Implementazione classe "Bacheca"	2 ore	1 giorno	2 ore	1 giorno	Rocco
Implementazione classe "Profilo"	2 ore	1 giorno	3 ore	1 giorno	Rocco
Implementazione classe "Post"	2 ore	1 giorno	3 ore	2 giorni	Lebedev
Implementazione classe "Commento"	2 ore	1 giorno	2 ore	1 giorno	Lebedev
Implementazione classe "Amicizia"	4 ore	1 giorno	2 ore	1 giorno	Pernpruner
Prima aggregazione codice	10 ore	4 giorni	7 ore	3 giorni	Tutti
Registrazione utente	5 ore	2 giorni	2 ore	1 giorno	Pernpruner
Login utente	5 ore	3 giorni	4 ore	1 giorno	Pernpruner
Logout utente	4 ore	2 giorni	1 ora	1 giorno	Pernpruner
Visualizzazione informazioni personali	4 ore	2 giorni	3 ore	1 giorno	Rocco
Modifica informazioni personali	4 ore	2 giorni	3 ore	1 giorno	Rocco
Visualizzazione bacheca personale	4 ore	2 giorni	6 ore	2 giorni	Rocco
Creazione post	6 ore	3 giorni	4 ore	2 giorni	Lebedev
Aggiunta commenti	6 ore	3 giorni	3 ore	2 giorni	Lebedev
Aggiunta "mi piace"	4 ore	2 giorni	6 ore	3 giorni	Lebedev
Richiesta amicizia	10 ore	5 giorni	10 ore	4 giorni	Lebedev, Pernpruner
Accettazione/rifiuto richieste amicizia	6 ore	3 giorni	4 ore	2 giorni	Lebedev
Cancellazione amicizie	4 ore	2 giorni	6 ore	3 giorni	Lebedev
Visualizzazione post amici	10 ore	5 giorni	6 ore	2 giorni	Pernpruner
Visualizzazione bacheca amici	5 ore	3 giorni	3 ore	1 giorno	Rocco
Visualizzazione informazioni amici	4 ore	2 giorni	3 ore	1 giorno	Rocco
Seconda aggregazione codice	15 ore	5 giorni	10 ore	4 giorni	Tutti
Testing					
Test fine progetto	10 ore	2 giorni	14 ore	3 giorni	Tutti
Conclusione					
Stesura documento finale			5 giorni		Tutti

Riepilogo

Totale durata per ogni attività:

Attività	Durata	
	Effort	Totale
Analisi	29 ore	14 giorni
Progettazione	40 ore	20 giorni
Sviluppo	97 ore	42 giorni
Testing	14 ore	3 giorni
Conclusione		5 giorni
TOTALE	**180 ore (senza doc)**	**84 giorni**

Basi di dati

Come basi di dati, vengono utilizzati *file* di formato *.tsv* con informazioni separate da carattere di tabulazione ("\t").

Tale decisione nasce dalla possibilità concreta che all'interno di post e commenti vengano usati sia virgole (",") che punti e virgola (";"), rendendo poco sicuro l'utilizzo di tali caratteri come delimitatori.

> *"TSV is an alternative to the common comma-separated values (CSV) format, which often causes difficulties because of the need to escape commas – literal commas are very common in text data, but literal tab stops are infrequent in running text."*
>
> *(Tab-separated values, http://en.wikipedia.org)*

Dati generali

Tali *file* verranno inseriti in una cartella denominata *File* contenuta nella cartella principale dell'applicazione. In tale cartella verranno inseriti i *file* generali: *utenti.tsv*, *post.tsv*.

Dati utente

Percorso: *[cartella codice]/File/Dati utente/*

In tale percorso verrà creata una cartella per ogni utente, avente come nome l'indirizzo *email* dell'utente a cui appartiene.

Nella cartella di ogni utente, verranno creati i file definiti "*(uno per ogni utente registrato)*" nella descrizione sottostante, quindi: *profilo.tsv*, *amicizie.tsv*.

Dati post

Percorso: *[cartella codice]/File/Dati post/*

In tale percorso verrà creata una cartella per ogni post, avente come nome l'ID numerico progressivo del post a cui appartiene.

Nella cartella di ogni post, verranno creati i file definiti "*(uno per ogni post)*" nella descrizione sottostante, quindi: *likes.tsv*, *commenti.tsv*.

Descrizione file utilizzati

Descrizione dei *file* utilizzati come basi di dati:

utenti.tsv	
file generale	
ID utente Nome Cognome Email Password	
Informazioni contenute	
ID utente	identificatore univoco progressivo generato automaticamente
Nome	come inserito all'atto della registrazione
Cognome	come inserito all'atto della registrazione

Indirizzo email	come inserito all'atto della registrazione
Password	come inserita all'atto della registrazione

post.tsv	
file generale	
ID post Email autore Data ora Testo	
Informazioni contenute	
ID post	identificatore univoco progressivo generato automaticamente
Email autore	identificatore univoco dell'utente che ha pubblicato il *post*
Data e ora di pubblicazione	data e ora di pubblicazione del *post*
Testo	testo del post

profilo.tsv	
uno per ogni utente registrato	
Sesso Professione Situazione sentimentale GG/MM/AAAA nascita Luogo nascita	
Informazioni contenute	
Sesso	solo se modificato successivamente, altrimenti "ND"
Professione	solo se modificata successivamente, altrimenti "ND"
Situazione sentimentale	solo se modificata successivamente, altrimenti "ND"
Data di nascita	solo se modificata successivamente, altrimenti "1/1/0"
Luogo di nascita	solo se modificato successivamente, altrimenti "ND"

amicizie.tsv	
uno per ogni utente registrato	
ID amicizia Email Status Ruolo	
Informazioni contenute	
ID amicizia	identificatore univoco progressivo generato automaticamente
Email altro utente	identificatore univoco dell'utente di colui a cui appartiene il file
Status	stato della richiesta di amicizia (*accettata/rifiutata/in attesa*)
Ruolo	ruolo dell'utente a cui appartiene il file all'interno della richiesta di amicizia (*mittente/destinatario*)

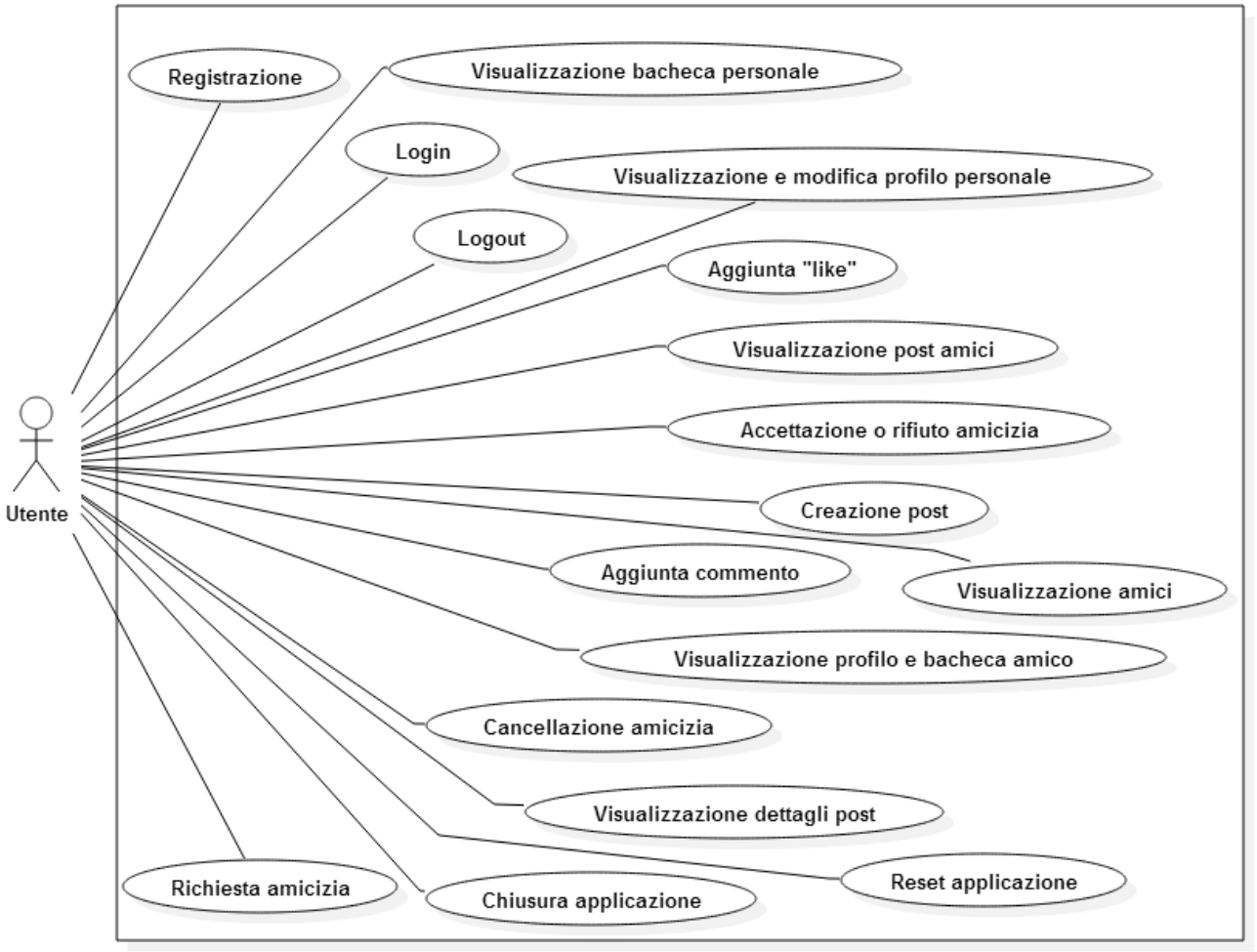
commenti.tsv	
uno per ogni post	
ID commento Email Data ora Testo	
Informazioni contenute	
ID commento	identificatore univoco progressivo generato automaticamente
Email autore	identificatore univoco dell'utente che ha pubblicato il commento
Data e ora di pubblicazione	data e ora di pubblicazione del *post*
Testo	testo del commento

likes.tsv

uno per ogni post	
Email autore like	
Informazioni contenute	
Email autore like	identificatore univoco dell'utente che ha espresso il proprio apprezzamento

Casi d'uso

Diagramma UML



Descrizione testuale

1. Registrazione di un nuovo utente – Utente
    a) Il sistema mostra all'utente la schermata di accesso → *InTouch::schermata_autenticazione()*
    b) L'utente seleziona la funzione "Registrati" → *InTouch::registrazione()*
    c) L'utente immette il suo nome, cognome, indirizzo e-mail e password
    d) L'utente conferma quanto immesso
    e) Il sistema verifica che l'e-mail inserita non sia già in utilizzo → *InTouch::utente_esiste()*
        i. Se lo fosse, riporta direttamente alla schermata di autenticazione mostrando un messaggio di errore → *InTouch::schermata_autenticazione()*
    f) Il sistema mostra una schermata di conferma registrazione
    g) Il sistema torna alla schermata di autenticazione → *InTouch::schermata_autenticazione()*

2. Login di un utente – Utente
    a) Il sistema mostra all'utente la schermata di accesso → *InTouch::schermata_autenticazione()*
    b) L'utente seleziona la funzione "Autenticati" → *InTouch::login()*
    c) L'utente immette il suo indirizzo e-mail e la sua password
    d) L'utente conferma quanto immesso

e) Il sistema verifica che l'utente esista → *InTouch::utente_esiste()*
    i. Se l'utente non esiste, riporta alla schermata di autenticazione mostrando un messaggio di errore → *InTouch::schermata_autenticazione()*
f) Il sistema verifica la correttezza di e-mail e password → *InTouch::check_login()*
    i. Se l'utente esiste ma la *password* non coincide riporta direttamente alla schermata di autenticazione mostrando un messaggio di errore → *InTouch::schermata_autenticazione()*
g) Il sistema mostra la schermata iniziale → *Utente::schermata_iniziale()*

3. Richieste di amicizia – Utente
    a) Il sistema mostra la schermata iniziale → *Utente::schermata_iniziale()*
    b) L'utente seleziona la voce "Gestisci amicizie" (tasto 1) → *Utente::gestisci_amicizie()*
    c) All'utente viene presentato un menù di scelta, seleziona "Richiedi amicizia" (tasto 1)
    d) Il sistema mostra all'utente l'elenco di tutti gli utenti iscritti → *Utente::richiedi_amicizia()*
        i. Se non sono presenti altri utenti a cui è possibile richiedere l'amicizia, il sistema riporta direttamente alla schermata di gestione amicizie mostrando un apposito messaggio → *Utente::gestisci_amicizie()*
    e) L'utente digita il numero corrispondente alla persona da aggiungere nella propria lista di amici
    f) Il sistema mostra un messaggio di conferma
    g) Il sistema mostra la schermata di gestione amicizie → *Utente::gestisci_amicizie()*

4. Accettazione o rifiuto di amicizie pervenute – Utente
    a) Il sistema mostra la schermata iniziale → *Utente::schermata_iniziale()*
    b) L'utente seleziona la voce "Gestisci amicizie" (tasto 1) → *Utente::gestisci_amicizie()*
    c) All'utente viene presentato un menù di scelta, seleziona "Accetta/Rifiuta amicizia" (tasto 2)
    d) Il sistema mostra all'utente l'elenco delle richieste pervenute → *Utente::accetta_rifiuta_amicizia()*
        i. Se non sono presenti richieste di amicizia in attesa di risposta, il sistema riporta direttamente alla schermata di gestione amicizie mostrando un apposito messaggio → *Utente::gestisci_amicizie()*
    e) L'utente digita il numero della persona da selezionare
    f) Il sistema richiede se si desidera accettare o rifiutare l'amicizia selezionata
    g) L'utente digita A per accettare, R per rifiutare
    h) Il sistema mostra un messaggio di conferma
    i) Il sistema mostra la schermata di gestione amicizie → *Utente::gestisci_amicizie()*

5. Cancellazione di amicizie – Utente
    a) Il sistema mostra la schermata iniziale → *Utente::schermata_iniziale()*
    b) L'utente seleziona la voce "Gestisci amicizie" (tasto 1) → *Utente::gestione_amicizie()*
    c) All'utente viene presentato un menù di scelta, seleziona "Cancella amicizia" (tasto 3)
    d) Il sistema mostra all'utente l'elenco di tutti gli amici → *Utente::cancella_amicizia()*
        i. Se l'utente non ha ancora amici, il sistema riporta direttamente alla schermata di gestione amicizie mostrando un apposito messaggio → *Utente::gestisci_amicizie()*
    e) L'utente digita il numero della persona da togliere nella propria lista di amici
    f) Il sistema mostra un messaggio di conferma
    g) Il sistema mostra la schermata di gestione amicizie → *Utente::gestisci_amicizie()*

6. Visualizzazione dei propri amici – Utente
    a) Il sistema mostra la schermata iniziale → *Utente::schermata_iniziale()*
    b) L'utente seleziona la voce "Gestisci amicizie" (tasto 1) → *Utente::gestione_amicizie()*
    c) All'utente viene presentato un menù di scelta, seleziona "Visualizza amicizie" (tasto 4)
    d) Il sistema mostra all'utente l'elenco di tutti gli amici → *Utente::visualizza_amici()*

i. Se l'utente non ha ancora amici, il sistema riporta direttamente alla schermata di gestione amicizie mostrando un apposito messaggio → *Utente::gestisci_amicizie()*

e) Alla pressione del tasto 0 da parte dell'utente, il sistema torna alla schermata di gestione amicizie → *Utente::gestisci_amicizie()*

7. Visualizzazione dei post dei propri amici – Utente
   a) Il sistema mostra la schermata iniziale → *Utente::schermata_iniziale()*
   b) L'utente seleziona la voce "Visualizza i post degli amici" (tasto 2) → *Utente::visualizza_bacheca_generale()*
   c) Il sistema mostra all'utente l'elenco dei post dei propri amici
      i. Se non sono presenti post di propri amici, il sistema riporta direttamente alla schermata iniziale mostrando un apposito messaggio→ *Utente::schermata_iniziale()*
   d) Al termine della visualizzazione, l'utente preme il tasto 0 per tornare alla schermata iniziale
   e) Il sistema mostra la schermata iniziale → *Utente::schermata_iniziale()*

8. Visualizzazione dei dettagli di un post (commenti e utenti a cui piace) – Utente
   a) Il sistema mostra la schermata "Visualizza i post degli amici" (tasto 2), la propria bacheca (tasto 3) o la bacheca di un amico (tasto 5, numero dell'amico, tasto 1)
   b) Il sistema mostra all'utente l'elenco dei post → *Post::visualizza_post_light()*
   c) L'utente digita il numero del post di cui desidera visualizzare i dettagli
   d) Il sistema visualizza i dettagli del post → *Post::visualizza_post()*
   e) Il sistema chiede inoltre se si desidera commentare, mettere/togliere "mi piace" o tornare alla schermata iniziale
   f) Al termine della visualizzazione, l'utente preme il tasto 0 per tornare alla bacheca completa
   g) Il sistema mostra la bacheca completa → *Post::visualizza_post_light()*

9. Aggiunta di un commento ad un post – Utente
   a) Il sistema mostra la schermata "Visualizza i post degli amici" (tasto 2), la propria bacheca (tasto 3) o la bacheca di un amico (tasto 5, numero dell'amico, tasto 1)
   b) Il sistema mostra all'utente l'elenco dei post → *Post::visualizza_post_light()*
   c) L'utente digita il numero del post che desidera commentare
   d) Il sistema mostra i dettagli del post → *Post::visualizza_post()*
   e) Il sistema chiede se commentare o esprimere apprezzamento
   f) L'utente digita il tasto 1 per commentare → *Post::commenta_post()*
   g) Il sistema propone all'utente una schermata per la compilazione del commento
   h) L'utente digita il testo del commento
   i) Il sistema chiede per una conferma (tasto 2), dando la possibilità di modificare il testo del commento (tasto 1) o di annullare l'operazione e tornare alla visualizzazione della bacheca (tasto 0)
   j) L'utente conferma con il tasto 2
   k) Il sistema mostra la schermata di cui al punto d), in base alla selezione precedentemente effettuata
   l) Al termine della visualizzazione, l'utente preme il tasto 0 per tornare alla visualizzazione della bacheca completa
   m) Il sistema mostra la bacheca completa → *Post::visualizza_post_light()*

10. Aggiunta del proprio apprezzamento ad un post – Utente
    a) Il sistema mostra la schermata "Visualizza i post degli amici" (tasto 2), la propria bacheca (tasto 3) o la bacheca di un amico (tasto 5, numero dell'amico, tasto 1)
    b) Il sistema mostra all'utente l'elenco dei post
    c) L'utente digita il numero del post per cui desidera esprimere apprezzamento
    d) Il sistema mostra i dettagli del post → *Post::visualizza_post()*

e) Il sistema chiede se commentare o esprimere apprezzamento
f) L'utente digita il tasto 2 per mettere "mi piace" → *Post::aggiungi_like()*
g) Il sistema mostra un messaggio di conferma
h) Il sistema mostra la schermata di cui al punto d), in base alla selezione precedentemente effettuata
i) Al termine della visualizzazione, l'utente preme il tasto 0 per tornare alla visualizzazione della bacheca completa
j) Il sistema mostra la bacheca completa → *Post::visualizza_post_light()*

11. Visualizzazione della propria bacheca (propri post) – Utente
    a) Il sistema mostra la schermata iniziale → *Utente::schermata_iniziale()*
    b) L'utente seleziona la voce "Visualizza la tua bacheca" (tasto 3)
    c) Il sistema mostra all'utente l'elenco dei propri post pubblicati → *Utente::visualizza_bacheca()*
        i. Se l'utente non ha ancora pubblicato post, il sistema riporta alla schermata iniziale mostrando un apposito messaggio → *Utente::schermata_iniziale()*
    d) Al termine della visualizzazione, l'utente preme il tasto 0 per tornare alla schermata iniziale
    e) Il sistema mostra la schermata iniziale → *Utente::schermata_iniziale()*

12. Visualizzazione e modifica delle proprie informazioni personali – Utente
    a) Il sistema mostra la schermata iniziale → *Utente::schermata_iniziale()*
    b) L'utente seleziona la voce "Visualizza e modifica il tuo profilo" (tasto 4)
    c) Il sistema mostra all'utente le proprie informazioni personali → *Utente::visualizza_profilo()*
    d) Se l'utente desidera modificare le proprie informazioni preme il tasto 1, altrimenti preme il tasto 0 per tornare alla schermata iniziale
    e) [eventuale modifica delle informazioni] → *Utente::modifica_profilo()*
    f) Il sistema mostra la schermata iniziale → *Utente::schermata_iniziale()*

13. Visualizzazione di profilo e bacheca di un amico – Utente
    a) Il sistema mostra la schermata iniziale → *Utente::schermata_iniziale()*
    b) L'utente seleziona la voce "Visualizza il profilo e la bacheca di un amico" (tasto 5)
    c) Il sistema propone all'utente l'elenco dei propri amici
    d) L'utente seleziona l'amico di cui visualizzare il profilo o la bacheca
    e) Il sistema propone all'utente la scelta se visualizzare la bacheca o il profilo dell'amico selezionato, oppure se tornare alla schermata di selezione dell'amico (00.5.X)
    f) L'utente risponde a tale richiesta
    g) Il sistema propone all'utente la schermata contenente la soluzione scelta
    h) Se è stata scelta la bacheca (opzione 1) digitando il numero del post è data la possibilità all'utente di commentare tale post (schermata 00.2.X.1)
    i) Al termine della visualizzazione, l'utente può scegliere se visualizzare anche l'altra alternativa premendo il tasto 0, che lo farà tornare al menù di scelta 00.5.X

    Si riparte dal punto (e) fino a quando la scelta dell'utente è (0)

    j) Il sistema mostra la schermata iniziale → *Utente::schermata_iniziale()*

14. Creazione di un post personale – Utente
    a) Il sistema mostra la schermata iniziale → *Utente::schermata_iniziale()*
    b) L'utente seleziona la voce "Crea post" (tasto 6) → *Bacheca::aggiungi_post()*
    c) Il sistema propone all'utente una schermata per la compilazione del post
    d) L'utente inserisce il contenuto del post
    e) Al termine della digitazione, il sistema chiede all'utente se confermare e finalizzare il post (tasto 1) o annullare il processo tornando alla schermata iniziale (tasto 0)
    f) Il sistema mostra un messaggio di conferma se la creazione è andata a buon fine, o d'errore in caso contrario
    g) Il sistema mostra la schermata iniziale → *Utente::schermata_iniziale()*

15. Logout – Utente
    a) Il sistema mostra la schermata iniziale → *Utente::schermata_iniziale()*
    b) L'utente seleziona la voce "Logout" (tasto 7)
    c) Il sistema mostra una schermata di conferma dell'avvenuta disconnessione → *Utente::logout()*
    d) Il sistema mostra la schermata di autenticazione → *InTouch::schermata_autenticazione()*

16. Chiusura applicazione – Utente
    a) Il sistema mostra la schermata di autenticazione → *InTouch::schermata_autenticazione()*
    b) L'utente seleziona la voce "Chiudi applicazione" (tasto 3)
    c) La finestra dell'applicazione si chiude → *exit(1)*

17. Reset applicazione – Utente (in realtà finalizzato al *testing*)
    a) Il sistema mostra la schermata di autenticazione → *InTouch::schermata_autenticazione()*
    b) L'utente seleziona la voce "Reset applicazione" (tasto 4)
    c) L'applicazione cancella tutti i file presenti, eliminando di fatto ogni traccia relativa a utenti, post, profili, commenti, *likes* e amicizie → *InTouch::reset()*
    d) Il sistema mostra a video la lista di tutti i file cancellati, con eventuali messaggi di errore
    e) Il sistema attende la pressione di un tasto da parte dell'utente → *system("PAUSE")*
    f) La finestra dell'applicazione di chiude in seguito alla pressione di un tasto da parte dell'utente

Schermate di interazione

***(0)** Schermata di accesso:*

Benvenuto in InTouch!
Seleziona cosa vuoi fare:

1. Registrati ***[UC1]***
2. Autenticati ***[UC2]***
3. Chiudi applicazione ***[UC16]***
4. Reset applicazione ***[UC17]***

(0.1)** Registrati* ***[UC1]

Inserisci il tuo nome (max X caratteri): ___
Inserisci il tuo cognome (max X caratteri): ___
Inserisci il tuo indirizzo e-mail (max X caratteri): ___
Inserisci una password (max X caratteri): ___

Premi 1 per confermare la registrazione,
premi 0 per annullare e tornare alla schermata di accesso. ___

Messaggio di conferma: Utente registrato correttamente!

(0.2)** Autenticati* ***[UC2]

Inserisci il tuo indirizzo e-mail: ___
Inserisci la tua password: ___

Messaggio di conferma: Login riuscito!

(0.3)** Chiudi applicazione* ***[UC16]

[Chiusura della finestra]

(0.4)** Reset applicazione* ***[UC17]

Cancello [*path files*]/[*nome file*]
Cancello [*path files*]/[*nome file*]
Cancello [*path files*]/[*nome file*]
[…]

Premere un tasto per continuare . . .

[Chiusura della finestra]

(00) *Schermata iniziale*

Benvenuto in InTouch!

[*eventuale notifica relativa a richieste di amicizia in attesa*]

Seleziona la funzione desiderata:

1. Gestisci amicizie
2. Visualizza i post degli amici ***[UC7]***
3. Visualizza la tua bacheca ***[UC11]***
4. Visualizza e modifica il tuo profilo ***[UC12]***
5. Visualizza il profilo e la bacheca di un amico ***[UC13]***
6. Crea post ***[UC14]***
7. Logout ***[UC14]***

(00.1) *Gestisci amicizie*

Seleziona la funzione desiderata:

1. Richiedi amicizia ***[UC3]***
2. Accetta/rifiuta amicizia ***[UC4]***
3. Cancella amicizia ***[UC5]***
4. Visualizza amicizie ***[UC6]***

Premi 0 per tornare alla schermata iniziale. ___

(00.1.1) *Richiedi amicizia* ***[UC3]***

Lista utenti a cui puoi richiedere l'amicizia:

- #1 - [Cognome Nome]
- #2 - [Cognome Nome]
- […]
- #N - [Cognome Nome]

Selezione utente a cui chiedere l'amicizia,

Premi 0 per tornare alla schermata precedente ___

Messaggio di conferma: Amicizia richiesta correttamente!

(00.1.2) *Accetta/rifiuta amicizie* ***[UC4]***

Lista richieste ricevute:

- #1 - [Cognome Nome]
- #2 - [Cognome Nome]
- […]
- #N - [Cognome Nome]

Selezione utente del quale accettare o rifiutare l'amicizia

Premi 0 per tornare alla schermata precedente ___

(00.1.2.X) *Accetta/rifiuta amicizia dell'utente specificato* ***[UC4]***

Accettare o rifiutare l'amicizia? (A/R) __

Messaggio di conferma: Amicizia accettata[/rifiutata] correttamente!

(00.1.3) *Cancella amicizie* ***[UC5]***

Lista amici:

- 1 - [Cognome Nome]
- 2 - [Cognome Nome]

- […]
- N - [Cognome Nome]

Seleziona numero dell'amico da cancellare
Premi 0 per tornare alla schermata precedente ___

Messaggio di conferma: Amicizia cancellata correttamente! Un rifiuto e' per sempre!

(00.2) *Visualizza i post degli amici* ***[UC6]***

POST #1
Autore: [Cognome Nome] ([indirizzo *email*])
Data: [Data e ora di pubblicazione]
"[Testo del post]"
[N commenti]
[N *likes*]

POST #2
Autore: [Cognome Nome] ([indirizzo *email*])
Data: [Data e ora di pubblicazione]
"[Testo del post]"
[N commenti]
[N *likes*]

POST #N
Autore: [Cognome Nome] ([indirizzo *email*])
Data: [Data e ora di pubblicazione]
"[Testo del post]"
[N commenti]
[N *likes*]

Per visualizzare i dettagli di un post e interagire con esso digitarne il numero,
Per tornare alla schermata iniziale premi 0 ___

(00.2.X) *Interagisci con un post* ***[UC8]***

POST #X ####################
Autore: [Cognome Nome] ([indirizzo *email*])
Data: [Data e ora di pubblicazione]
"[Testo del post]"

[Commenti]
Commento #X ####################
Autore: [Cognome Nome] ([indirizzo *email*])
Data: [Data e ora di pubblicazione]
"[Testo del commento]"
…

Utenti a cui piace questo post:
[Cognome Nome]
[Cognome Nome]
…

Per commentare il post selezionato premi 1,
per mettere[/togliere] "mi piace" premi 2,
per tornare alla schermata iniziale premi 0 ___

(00.2.X.1) *Commenta un post* ***[UC9]***

Inserisci testo:

Per confermare ed aggiungere il commento al post premi 2

Per modificare il testo del commento premi 1
Per annullare e tornare alla schermata precedente premi 0 ___

Messaggio di conferma: Commento pubblicato!
Messaggio di conferma annullamento pubblicazione: Pubblicazione del commento annullata!

(00.2.X.2) *Metti "mi piace" ad un post* ***[UC10]***

Se aggiunto: Liked!
Se rimosso: Non ti piace piu'!

(00.3) *Visualizza la tua bacheca* ***[UC11]***

Bacheca di [Nome Cognome]

POST #X
Autore: [Cognome Nome] ([indirizzo *email*])
Data: [Data e ora di pubblicazione]
"[Testo del post]"
[N commenti]
[N *likes*]

Post #N
Autore: [Cognome Nome] ([indirizzo *email*])
Data: [Data e ora di pubblicazione]
"[Testo del post]"
[N commenti]
[N *likes*]

Per visualizzare i dettagli di un post e interagire con esso digitarne il numero [schermata 00.2.X]
Per tornare alla schermata precedente premi 0 ___

(00.4) *Visualizza e modifica il tuo profilo* ***[UC12]***

[Nome Cognome]
Sesso: […] (*default*: ND)
Professione: […] (*default*: ND)
Situazione sentimentale: […] (*default*: ND)
Data di nascita: […] (*default*: 01/01/0
Luogo di nascita: […] (*default*: ND)
Per modificare il tuo profilo premi 1
Per tornare alla schermata iniziale premi 0 ___

(00.4.1) *Modifica il tuo profilo* ***[UC12]***

[Nome Cognome]
1. Sesso: […] (*default*: ND)
2. Professione: […] (*default*: ND)
3. Situazione sentimentale: […] (*default*: ND)
4. Data di nascita: […] (*default*: 01/01/0
5. Luogo di nascita: […] (*default*: ND)
Per modificare i campi selezionare il numero corrispondente
Per tornare alla visualizzazione del profilo premi 0 ___

(00.5) *Visualizza il profilo e bacheca di un amico* ***[UC13]***

Lista amici:

- #X - [Cognome Nome]
- […]
- #N - [Cognome Nome]

Selezionare il numero dell'amico per visualizzare le sue informazioni
Per tornare alla schermata iniziale premi 0. ___

(00.5.X) *Selezionato l'amico di cui visualizzare il profilo o la bacheca*

Per visualizzare la sua bacheca premi 1
Per visualizzare il suo profilo premi 2
Per tornare alla selezione dell'amico premi 3
Per tornare alla schermata iniziale premi 0 ___

(00.5.X.1) *Visualizza la bacheca dell'amico selezionato* ***[UC13]***

Bacheca di [Nome Cognome]

POST #X
Autore: [Cognome Nome] ([indirizzo *email*])
Data: [Data e ora di pubblicazione]
"[Testo del post]"
[N commenti]
[N *likes*]

POST #N
Autore: [Cognome Nome] ([indirizzo *email*])
Data: [Data e ora di pubblicazione]
"[Testo del post]"
[N commenti]
[N *likes*]

Per visualizzare i dettagli di un post e interagire con esso digitarne il numero [schermata 00.2.X], per tornare alla schermata iniziale premi 0. ___

(00.5.X.2) *Visualizza il profilo dell'amico selezionato* ***[UC13]***

[Nome cognome]
Sesso: X
Professione: […]
Situazione sentimentale: […]
Data di nascita: […]
Luogo di nascita: […]

Per tornare alla schermata precedente premi 0 [schermata 00.5.X]. ___

(00.6) *Crea post* ***[UC14]***

Inserire testo post: ___

Per confermare e aggiungere il post alla tua Bacheca premi 1,
Per annullare e tornare alla schermata iniziale premi 0 ___

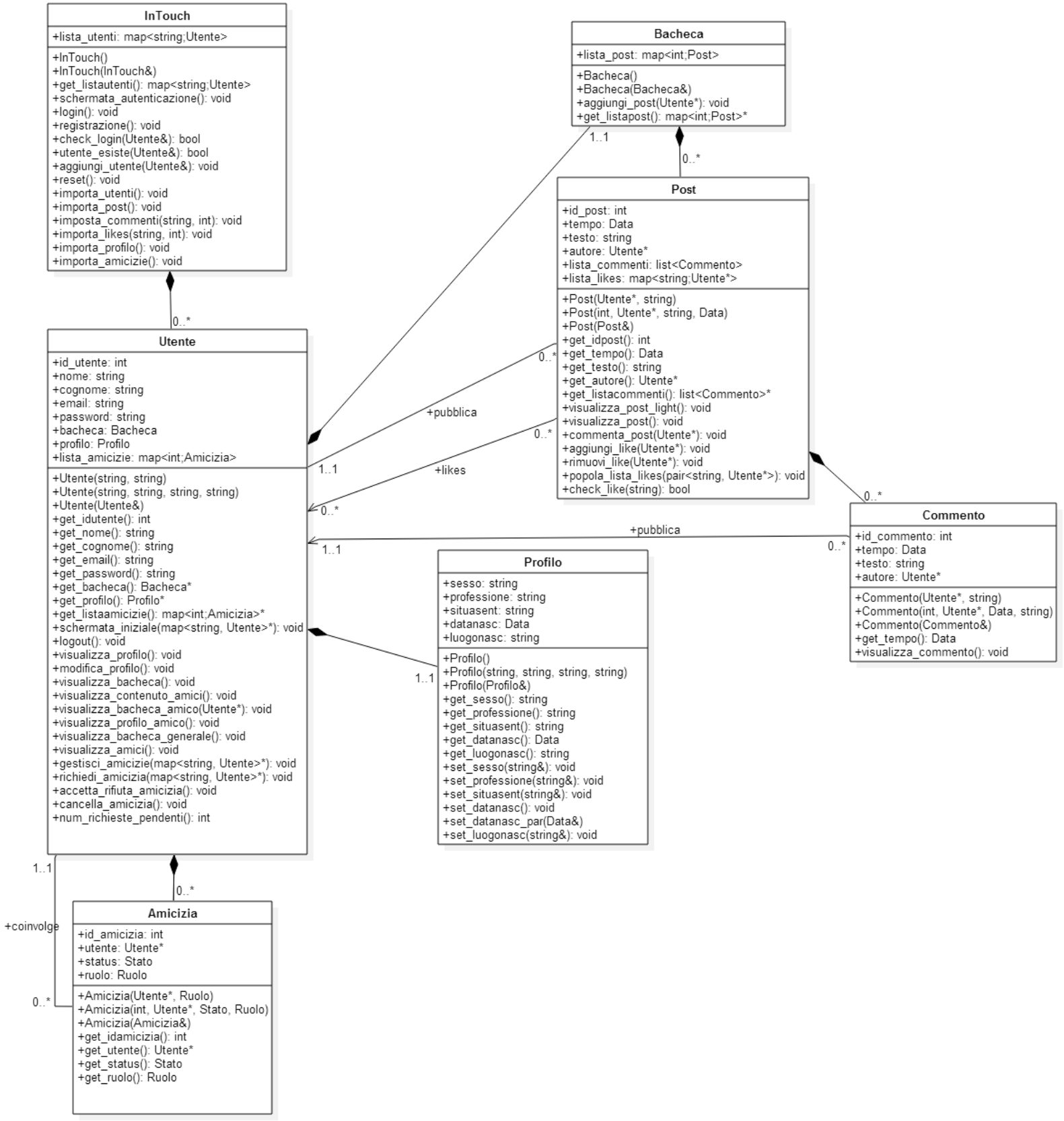
Messaggio di conferma dell'aggiunta: Post pubblicato!
Messaggio di conferma dell'annullamento: Pubblicazione del post annullata!

(00.7) *Esci* ***[UC15]***

Logout effettuato!
[(0) Schermata di accesso]

Diagramma ad oggetti

Diagramma UML



Descrizione testuale

InTouch	
main class del progetto, contiene i metodi che permettono all'utente di autenticarsi ed accedere all'applicazione.	
Attributi	
lista_utenti: map<string,Utente>	La classe InTouch contiene l'elenco degli utenti registrati nell'applicazione. Tali utenti si trovano in un contenitore *map* ove la chiave è costituita dall'indirizzo *email* immesso all'atto della registrazione, trattato come identificatore univoco; a questa chiave è associata l'intera istanza della classe utente corrispondente.
Metodi	
InTouch()	Costruttore di *default* della classe, contiene le funzioni che importano i dati da file di testo all'apertura dell'applicazione (quindi all'allocazione di un'istanza di tale classe): *importa_utenti()*, […]
InTouch(InTouch&)	Costruttore di copia.
map<string,Utente> get_listautenti()	Restituisce la lista degli utenti registrati nell'applicazione.
void schermata_autenticazione()	Costituisce la prima schermata che viene visualizzata una volta avviata l'applicazione, ovvero quella di autenticazione. Propone all'utente la possibilità di registrarsi, di autenticarsi o di chiudere l'applicazione.
void login()	Costituisce la schermata di login, ove viene chiesto all'utente il proprio indirizzo *email* e la propria *password*, la cui esattezza verrà poi controllata da appositi metodi. Nel caso il *login* abbia successo, rimanda alla schermata principale.
void registrazione()	Costituisce la schermata di registrazione, ove viene chiesto all'utente di inserire il proprio nome, cognome e indirizzo *email*, oltre a scegliere una password. Una volta terminata tale operazione, previa conferma dell'utente esso viene aggiunto all'elenco degli utenti tramite apposito metodo.
void check_login(Utente&)	Verifica se gli attributi *email* e *password* della variabile utente passata (che potrà contenere solamente tali due informazioni grazie al costruttore a due parametri appositamente presente nella relativa classe) corrispondono a quelle dichiarate in fase di registrazione; in tal caso, rimanda l'utente alla schermata principale personale, altrimenti rende un messaggio di errore e rimanda alla pagina di autenticazione.
void utente_esiste(Utente&)	Verifica se l'attributo *email* della variabile utente passata esiste all'interno della lista degli utenti, ovvero se l'utente è effettivamente registrato o meno. Se chiamato in fase di registrazione, serve ad impedire la creazione di più utenze con lo stesso indirizzo *email*. Se chiamato in fase di *login*, verifica che l'utente sia effettivamente registrato e in tal caso rimanda al metodo *check_login* per verificare la corrispondenza della *password* inserita, altrimenti rende un messaggio di errore e rimanda alla pagina di autenticazione.

void aggiungi_utente(Utente&)	Al termine dell'immissione dei dati all'atto della registrazione, essi sono inseriti in una variabile temporanea di tipo Utente tramite il costruttore a quattro parametri. Tale variabile viene passata al presente metodo che provvede ad inserire i dati del nuovo utente all'interno del contenitore *lista_utenti*, creando inoltre un nuovo *record* nel *database* "*utenti.tsv*".
void reset()	Permette il *reset* dell'applicazione, ovvero la cancellazione di tutti i dati attualmente presenti. Costituisce unico modo sicuro per l'inizializzazione dell'applicazione, dal momento che provvede ad eliminare in modo sicuro tutti i *file* e le cartelle relativi.
void importa_utenti()	Il presente metodo viene chiamato all'interno del costruttore di *default* della classe *InTouch* per caricare all'interno della lista di utenti i dati presenti nel *database* "*utenti.tsv*", in modo tale da non perdere quanto inserito in una sessione alla chiusura della finestra.
void importa_post()	Il presente metodo viene chiamato all'interno del costruttore di *default* della classe *InTouch* per caricare all'interno della bacheca di ogni utente i dati presenti nel *database* "*post.tsv*", in modo tale da non perdere quanto inserito in una sessione alla chiusura della finestra.
void importa_commenti(string, int)	Il presente metodo viene chiamato all'interno della funzione *importa_post()* per importare i commenti relativi ad ogni *post* esistente. Tali informazioni vengono prese dal *database* "*commenti.tsv*" relativo ad ogni *post*, in modo tale da non perdere quanto inserito in una sessione alla chiusura della finestra.
void importa_likes(string, int)	Il presente metodo viene chiamato all'interno della funzione *importa_post()* per importare i *likes* relativi ad ogni *post* esistente. Tali informazioni vengono prese dal *database* "*likes.tsv*" relativo ad ogni *post*, in modo tale da non perdere quanto inserito in una sessione alla chiusura della finestra.
void importa_profilo()	Il presente metodo viene chiamato all'interno del costruttore di *default* della classe *InTouch* per caricare le informazioni del profilo di ogni utente dal *database* "*profilo.tsv*" di ogni utente, in modo tale da non perdere quanto inserito in una sessione alla chiusura della finestra.
void importa_amicizie()	Il presente metodo viene chiamato all'interno del costruttore di *default* della classe *InTouch* per caricare all'interno della lista di amici relativo ad ogni utente i dati presenti nel *database* "*amicizie.tsv*" di ogni utente, in modo tale da non perdere quanto inserito in una sessione alla chiusura della finestra.

Utente	
classe contenente gli attributi distintivi degli utenti registrati.	
Attributi	
id_utente: int	Identificatore univoco progressivo di ogni utenza creata. La progressione è resa possibile dalla variabile globale *id_u* che viene incrementata all'aggiunta di ogni nuovo utente alla

	lista_utenti.
nome: string	Nome dell'utente, come inserito all'atto della registrazione.
cognome: string	Cognome dell'utente, come inserito all'atto della registrazione.
email: string	Indirizzo *email* dell'utente, come inserito all'atto della registrazione. Tale attributo costituirà l'identificatore univoco relativo ad ogni utente all'interno del contenitore *lista_utenti*. Verrà utilizzato per il *login* all'interno dell'applicazione
password: string	*Password* dell'utente, come inserita all'atto della registrazione. Verrà utilizzata per il *login* all'interno dell'applicazione
bacheca: bacheca	Inclusione 1..1 della classe *Bacheca*. Rappresenta la bacheca dell'utente, contenente l'elenco dei post da esso pubblicato.
profilo: Profilo	Inclusione 1..1 della classe *Profilo*. Rappresenta il profilo dell'utente, contenente le proprie informazioni personali che verranno mostrate tramite il metodo *visualizza_profilo* e potranno essere modificate con il metodo *modifica_profilo*.
lista_amicizie: map<int,Amicizia>	Contenitore *map* di ogni singola amicizia che coinvolge l'utente a cui appartiene l'istanza.
Metodi	
Utente(string, string)	Costruttore specifico a due parametri, utilizzato per i metodi *check_login* e *verifica_utente* della classe *InTouch*. In seguito al tentativo di *login*, all'interno dell'istanza di *Utente* così creata, vengono infatti immessi *email* e *password*; tale variabile viene poi passata ai sopracitati metodi per il controllo all'interno della *lista_utenti*.
Utente(string, string, string, string)	Costruttore specifico a quattro parametri, utilizzato per la fase di registrazione. Ultimata l'immissione dei dati e confermata la volontà di iscriversi all'applicazione da parte dell'utente, infatti, viene creata un'istanza di *Utente* contenente le quattro stringhe di dati immessi. Tale variabile viene poi passata alla funzione *aggiungi_utente* di *InTouch* che provvede ad aggiungere i dati del nuovo utente alla *lista_utenti* e al file "*utenti.tsv*".
Utente(Utente&)	Costruttore di copia.
int get_idutente()	Permette l'accesso all'attributo privato *id_utente* anche dall'esterno.
string get_nome()	Permette l'accesso all'attributo privato *nome* anche dall'esterno.
string get_cognome()	Permette l'accesso all'attributo privato *cognome* anche dall'esterno.
string get_email()	Permette l'accesso all'attributo privato *email* anche dall'esterno.
string get_password()	Permette l'accesso all'attributo privato *password* anche dall'esterno.
Bacheca* get_bacheca()	Restituisce un puntatore alla bacheca dell'utente. Utilizzato principalmente per la funzione *importa_post()* della classe

	InTouch, per poter inserire nelle corrette bacheche i post presi da *file*.
Profilo* get_profilo()	Restituisce un puntatore al profilo dell'utente.
map<int,Amicizia>* get_listaamicizie()	Restituisce un puntatore alla lista di amicizie dell'utente.
void schermata_iniziale(map<string,Utente>*)	Costituisce la schermata contenente il menu principale, che viene mostrata all'utente una volta completata con successo la fase di *login*. Viene passato come parametro un puntatore alla lista di utenti registrati, altrimenti inaccessibile poiché contenuta in *InTouch*.
void logout()	Permette all'utente di effettuare il *logout* dall'applicazione.
void visualizza_profilo()	A seconda di come chiamata, mostra all'utente le proprie informazioni personali o quelle di un altro utente. In particolare, stamperà a schermo: nome, cognome (come da registrazione), sesso, professione, situazione sentimentale, data di nascita e luogo di nascita. Se non modificate, di *default* le informazioni personali sono impostate su "*Non definito*".
void modifica_profilo()	Permette all'utente la modifica del proprio profilo, anche solamente di una voce al suo interno.
void visualizza_bacheca()	Mostra all'utente la bacheca dell'istanza della classe *Utente* chiamante. In particolare, può essere utilizzata per visualizzare la propria bacheca o quella di un amico, stampandone a video i post relativi.
void visualizza_contenuto_amici()	Mostra l'elenco dei propri amici e permette di visualizzarne la bacheca e il profilo facendo riferimento alle funzioni *visualizza_bacheca_amico()* e *visualizza_profilo_amico().*
void visualizza_bacheca_amico(Utente*)	Visualizza la bacheca di un amico, mostrando tutti i post da esso pubblicati. Viene passato come parametro un puntatore all'utente che visualizza la bacheca dell'amico per l'aggiunta di eventuali commenti o *likes*.
void visualizza_profilo_amico()	Visualizza il profilo di un amico, mostrando sesso, professione, situazione sentimentale, data e luogo di nascita.
void visualizza_bacheca_generale()	Visualizza la bacheca generale, ovvero l'elenco di tutti i post pubblicati dai propri amici.
void visualizza_amici()	Visualizza nome e cognome degli amici dell'utente oggetto dell'istanza chiamante.
void gestisci_amicizie(map<string, Utente>)	Menu che presenta all'utente le funzioni per gestire le amicizie.
void richiedi_amicizia(map<string, Utente>)	Stampa a video l'elenco degli utenti registrati, permettendo all'utente di selezionare a chi richiedere l'amicizia tramite selezione del numero corrispondente.
void accetta_rifiuta_amicizia()	Mostra all'utente l'elenco delle eventuali richieste di amicizia ricevute, permettendogli di accettare/rifiutare ognuna di esse.
void cancella_amicizia()	Permette all'utente di eliminare amici dal proprio elenco, tramite selezione del numero corrispondente.
int num_richieste_pendenti()	Restituisce il numero di richieste pendenti possedute dall'utente in questione. Viene utilizzata per visualizzare

	un'eventuale notifica nella schermata iniziale.

Bacheca	
classe contenente i metodi per visualizzare una specifica bacheca ed aggiungere un nuovo post.	
Attributi	
lista_post: map<int,Post>	La classe Bacheca contiene l'elenco dei post pubblicati dall'utente a cui appartiene la bacheca in questione. Tali post si trovano in un contenitore *map* ove la chiave è costituita da un identificatore univoco progressivo intero associato ad ogni post, a cui viene affiancata l'intera istanza della classe *Post* corrispondente.
Metodi	
Bacheca()	Costruttore di *default* della classe *Bacheca*, svuota la lista del *post* contenuta nell'istanza inizializzata al fine di evitare l'impiego di memoria sporca.
Bacheca(Bacheca&)	Costruttore di copia.
void aggiungi_post(Utente*)	Richiede all'utente a cui appartiene la bacheca il testo del post da pubblicare e lo aggiunge alla stessa. Viene passato come parametro un puntatore all'utente pubblicante il post, così da semplificare le operazioni di memorizzazione dell'autore e successivo accesso agli attributi dello stesso.
map<int,Post>* get_listapost	Restituisce un puntatore alla lista dei post della bacheca dell'utente. Utilizzato principalmente per la funzione *importa_post()* della classe *InTouch*, per poter inserire nelle corrette bacheche i post presi da *file*.

Profilo	
classe contenente informazioni aggiuntive sull'utente, che compariranno nella sezione del profilo ad esso relativo.	
Attributi	
sesso: string	Campo dedicato al sesso dell'utente, di *default*: "*Non definito*".
professione: string	Campo dedicato alla professione dell'utente, di *default*: "*Non definita*".
situasent: string	Campo dedicato alla situazione sentimentale dell'utente, di *default*: "*Non definita*".
datanasc: data	Campo dedicato alla data di nascita dell'utente.
luogonasc: string	Campo dedicato al luogo di nascita dell'utente, di *default*: "*Non definito*".
Metodi	
Profilo()	Costruttore di *default* del profilo personale di ogni utente; imposta i campi su "*ND*".
Profilo(string, string, string, string)	Costruttore specifico utilizzato per impostare gli attributi *sesso*, *professione*, *situasent* e *luogonasc* del profilo.
Profilo(Profilo&)	Costruttore di copia.
string get_sesso()	Permette l'accesso all'attributo privato *sesso* anche dall'esterno.

string get_professione()	Permette l'accesso all'attributo privato *professione* anche dall'esterno.
string get_situasent()	Permette l'accesso all'attributo privato *situazione_sent* anche dall'esterno.
Data get_datanasc()	Permette l'accesso all'attributo privato *data_nascita* anche dall'esterno.
string get_luogonasc()	Permette l'accesso all'attributo privato *luogo_nascita* anche dall'esterno.
void set_sesso(string&)	Imposta l'attributo *sesso* al valore passato alla funzione.
void set_professione(string&)	Imposta l'attributo *professione* al valore passato alla funzione.
void set_situasent(string&)	Imposta l'attributo *situazione_sent* al valore passato alla funzione.
void set_datanasc()	Richiama la funzione *imposta_data()* della classe *Data*.
void set_datanasc_par(Data&)	Imposta l'attributo *tempo* con il valore passato come parametro.
void set_luogonasc(string&)	Imposta l'attributo *luogo_nasc* al valore passato alla funzione.

Post	
classe contenente gli attributi distintivi di ogni post esistente	
Attributi	
id_post: int	Identificatore univoco progressivo intero di ogni post che viene creato; viene utilizzato come chiave nel contenitore *map lista_post* presente all'interno di ogni bacheca.
tempo: Data	Data e ora di pubblicazione del post.
testo: string	Testo del post.
autore: Utente*	Puntatore all'utente autore del post, in modo tale da poterne trovare le informazioni ove necessario.
lista_commenti: list<Commento>	La classe *Post* contiene l'elenco dei commenti pubblicati dagli utenti sul post in questione. Tali commenti si trovano in un contenitore *list* in cui sono contenute istanze della classe *Commento*.
lista_likes: map<string,Utente*>	Contiene la lista degli utenti che hanno espresso il proprio *like* per il post in questione. In particolare, tali dati sono memorizzati in un contenitore *map* avente come chiave l'indirizzo email dell'utente (per facilitare eventuali rimozioni del *like*), a cui viene associato un puntatore all'utente stesso.
Metodi	
Post(Utente*, string)	Costruttore specifico a due parametri, inizializza un'istanza della classe *Post* passando un puntatore all'utente autore ed il testo del post; data e ora vengono automaticamente impostate.
Post(int, Utente*, string, Data)	Costruttore specifico a quattro parametri, utilizzato per importare l'elenco generale dei post dal *database* "*post.tsv*".
Post(Post&)	Costruttore di copia.
int get_idpost()	Restituisce l'ID del post relativo.
Data get_tempo()	Restituisce la data di pubblicazione del post relativo.
string get_testo()	Restituisce il testo del post relativo.

Utente* get_autore()	Restituisce un puntatore all'autore del post relativo.
list<Commento>* get_listacommenti()	Restituisce un puntatore alla lista di commenti relativa al *post*.
void visualizza_post_light()	Visualizza il post in modo riassuntivo, indicando il numero di commenti e di *likes* presenti.
void visualizza_post()	Visualizza il post in modo dettagliato, mostrando l'elenco di utenti che hanno espresso il proprio apprezzamento e l'elenco di commenti relativi.
void commenta_post(Utente*)	Permette all'utente di aggiungere un commento al post, chiedendone il testo; viene passato un puntatore all'utente autore del post per una memorizzazione più efficace.
void aggiungi_like(Utente*)	Aggiunge l'utente al contenitore *map lista_likes*, ad indicare che l'utente ha espresso il proprio apprezzamento per il post in questione.
void rimuovi_like(Utente*)	Rimuove l'utente dal contenitore *map lista_likes*, ad indicare che l'utente ha rimosso il proprio apprezzamento per il post in questione.
void popola_lista_likes(pair<string,Utente*>)	???
bool check_like(string)	Restituisce un valore *booleano* a seconda che l'utente abbia o meno espresso il proprio "mi piace" per il post in questione. Viene utilizzato per mostrare la dicitura "mettere" o "togliere" mi piace a seconda dello stato attuale.

Commento	
classe contenente gli attributi distintivi di ogni commento esistente relativo ad ogni specifico post	
Attributi	
id_commento: int	Identificatore univoco progressivo intero di ogni commento che viene creato; viene utilizzato come chiave nel contenitore *map lista_commenti* presente all'interno di ogni post.
tempo: data	Data e ora di pubblicazione del commento.
testo: string	Testo del commento.
autore: Utente*	Puntatore all'utente autore del post, in modo tale da poterne trovare le informazioni ove necessario.
Metodi	
Commento(Utente*, string)	Costruttore specifico a un parametro, inizializza un'istanza della classe *Commento* impostando l'attributo *testo* con la stringa passata.
Commento(int, Utente*, Data, string)	Costruttore specifico a quattro parametri, utilizzato per importare i commenti da *file* di testo all'atto dell'inizializzazione.
Commento(Commento&)	Costruttore di copia.
Data get_tempo()	Restituisce la data e l'ora di pubblicazione del commento.
void visualizza_commento()	Permette la visualizzazione del commento, con la relative informazioni (data di pubblicazione ed autore).

Amicizia

association class relativa all'associazione tra i vari utenti, contiene informazioni sul mittente e il destinatario della specifica richiesta e uno *status* che ne identifica lo stato corrente (accettata/in attesa/rifiutata)	
Attributi	
id_amicizia: int	Identificatore univoco progressivo intero di ogni commento che viene creato; viene utilizzato come chiave nel contenitore *map lista_amicizie* presente all'interno di ogni utente.
Utente: Utente*	Puntatore all'utente che ha inviato la richiesta di amicizia.
status: Stato	Stato della richiesta di amicizia. Può assumere tre valori: • *X* di *default*, se la richiesta è in stallo • *A* se la richiesta viene accettata • *R* se la richiesta viene rifiutata
ruolo: Ruolo	Ruolo dell'utente in cui è contenuta la lista all'interno della richiesta di amicizia. Può assumere due valori: • *MITTENTE* se l'utente ha inviato la richiesta di amicizia • *DESTINATARIO* se l'utente ha ricevuto la richiesta di amicizia
Metodi	
Amicizia(Utente*, Ruolo)	Costruttore a due parametri, permette di creare un'istanza della classe *Amicizia* passando il puntatore all'altro utente coinvolto e il ruolo dell'utente in cui è contenuta l'amicizia.
Amicizia(int, Utente*, Stato, Ruolo)	Costruttore a quattro parametri, utilizzato nell'importazione di amicizie da *file* di testo all'atto dell'inizializzazione.
Amicizia(Amicizia&)	Costruttore di copia.
int get_idamicizia()	Restituisce l'identificatore univoco della richiesta di amicizia.
Utente* get_utente()	Restituisce un puntatore all'utente coinvolto nell'amicizia.
Stato get_status()	Restituisce lo stato di una determinata amicizia.
Ruolo get_ruolo()	Restituisce il ruolo dell'utente in cui una determinata amicizia è contenuta.

Librerie utilizzate

Nella realizzazione del progetto sono state utilizzate le seguenti librerie:

conio.h	La libreria <conio.h> dà accesso ad un metodo alternativo di input da parte dell'utente, ovvero alla funzione **getch()**. Questa permette di ricevere input senza visualizzarlo sullo schermo, utile nel caso si vogliano nascondere campi sensibili quali la password dell'utente. Nel progetto viene utilizzata all'interno della funzione *inputPassword()*, per nascondere la digitazione della *password* in fase di *login*.
cstdlib	*Standard library* del C++, dichiara funzioni e costanti di utilità generale: allocazione della memoria, controllo dei processi, e altre funzioni generali comprendenti anche i tipi di dato.
cstring	Permette l'utilizzo della funzione *strtok()*, che divide una stringa in *token*. Tale funzione viene utilizzata a sua volta dalle funzioni di importazione da file presenti nella classe *InTouch* per suddividere le righe di testo nei relativi dati da inserire nel sistema all'apertura dell'applicazione.
fstream	Permette l'apertura di un flusso input/output su file; nel progetto

	viene utilizzata per la scrittura e la lettura su/da file *.tsv*.
iostream	Permette l'apertura di un flusso input/output su schermo; nel progetto viene utilizzata per le interazioni con l'utente.
list	Permette l'utilizzo di contenitori di tipo *list* facenti parte della *Standard Template Library*.
map	Permette l'utilizzo di contenitori di tipo *map* facenti parte della *Standard Template Library*.
set	Permette l'utilizzo di contenitori di tipo *set* facenti parte della *Standard Template Library*.
sstream	La libreria <sstream> è stata implementata per permettere al programma di avvantaggiarsi della classe **stringstream**, che abbiamo osservato reagire meglio di "cin>>" agli input numerici dell'utente mediante conversione da string a integer, che avviene sempre tramite **operator>>.** Nel progetto viene utilizzata principalmente all'interno della funzione *inputInt()*.
string	Permette l'utilizzo di tipi di dato aggiuntivi denominati *string*, con l'implementazioni di funzioni apposite e la ridefinizione dell'operatore = relativamente alle stringhe.
ctime ***time.h***	La libreria <ctime> permette al programma di prendere la data corrente senza l'intervento diretto dell'utente. Utile, se non addirittura necessario, in casi quali la creazione di post e di commenti. In particolare nel programma viene fatto uso del tipo **time_t** inizializzato con **time(0)**, che viene successivamente convertito come tempo locale in una **struct tm** grazie a **localtime**. I tipi e le struct appena nominate non vengono utilizzate direttamente per salvare i dati relativi al tempo nella classe *Data* per problemi di portabilità.

File utilizzati come librerie

All'interno del progetto è stato incluso un file "*input.h",* utilizzato come libreria personalizzata.

In particolare, all'interno di tale file sono state inserite le funzioni :

int inputInt(int, int)	Permette di effettuare un controllo sul carattere immesso ogni qualvolta viene richiesto di effettuare una delle scelte presentate da un menu (requisito #2). In particolare, vengono eseguiti i seguenti controlli: • Sul tipo di dato inserito: viene verificato che il dato inserito sia effettivamente di tipo *int*, unico ammesso per la scelta • Sul *range* di scelta del relativo menu: una volta verificata la correttezza del tipo di dato inserito, viene verificato che il dato sia anche compreso nel *range* di scelta del relativo menu (i cui valori minimo e massimo sono passati come parametri)
string inputString(int)	Passato come parametro il numero massimo di caratteri ammessi, verifica che l'immissione effettuata dall'utente non ecceda tale limite (*es. inserimento del nome dell'utente)*
string inputEmail(int)	Passato come parametro il numero massimo di caratteri ammessi, verifica che l'immissione effettuata dall'utente non ecceda tale limite, controllando inoltre che la stringa immessa non contenga determinati caratteri speciali non ammessi da Windows all'interno dei nomi delle cartelle. Viene utilizzata sulla *email* immessa all'atto della registrazione, dal momento che essa verrà poi usata come nominativo per la cartella relativa all'utente in questione.
string inputPassword(int)	Passato come parametro il numero massimo di caratteri ammessi, verifica che l'immissione della *password* effettuata dall'utente non ecceda tale limite. Inoltre, all'atto del *login*, sostituisce ogni carattere digitato nel

	campo *password* in asterischi (*), in modo tale da garantire la sicurezza nell'immissione della stessa.

È stato inoltre incluso un secondo *file* "*data.h*", utilizzato anch'esso come libreria personalizzata, al fine di gestire correttamente l'immissione delle date.

In particolare, all'interno di tale file è stata inserita una classe *Data* così formata:

Data	
classe utilizzata per gestire la corretta immissione e gestione delle date	
Attributi	
giorno: int	Attributo contenente il giorno in forma numerica.
mese: int	Attributo contenente il mese in forma numerica.
anno: int	Attributo contenente l'anno in forma numerica.
ore: int	Attributo contenente le ore in forma numerica.
minuti: int	Attributo contenente i minuti in forma numerica.
Metodi	
Data()	Costruttore di *default*, inizializza la variabile di tipo *Data* ad un valore di *default* (1/1/0 88:88).
Data(int, int, int)	Costruttore specifico a tre parametri, inizializza la variabile di tipo *Data* impostando *giorno*, *mese* ed *anno* con i valori passati, mantenendo invece *ore* e *minuti* su un valore *standard* (88:88).
Data(int, int, int, int, int)	Costruttore specifico a cinque parametri, inizializza la variabile di tipo *Data* impostando tutti i campi con i valori passati.
void imposta_data()	Chiede all'utente di inserire giorno, mese ed anno, verificandone la correttezza di *range* (1-31 per il giorno, 1-12 per il mese, anno al massimo uguale a quello attuale).
void imposta_dataOra()	Imposta la variabile chiamante alla data e ora attuale, utilizzando la libreria *time.h*. Viene utilizzato per impostare le variabili al tempo corrente al momento della pubblicazione di post e commenti.
ostream& operator << (ostream&, Data&)	Ridefinizione dell'operatore "<<": permette la stampa della data in formato GG/MM/AAAA, aggiungendo anche uno 0 come prima cifra di giorno e mese nel caso in cui questi siano compresi tra 1 e 9. Inoltre, se ora e minuti sono diversi da quelli di *default* (88:88), stampa anche ora e minuti nel formato HH:MM.

Infine, è stata inserita una terza libreria personalizzata denominata "*config.h*", contenente le impostazioni relative ai percorsi dei file utilizzati come basi di dati al fine di rendere gli stessi personalizzabili senza alterare il codice, oltre alle variabili globali relative all'ID univoco progressivo.

path_files: string	Percorso (*path*) della cartella contenente i file di testo generali
path_files_u: string	Percorso (*path*) della cartella contenente i file di testo relativi agli utenti
path_files_p: string	Percorso (*path*) della cartella contenente i file di testo relativi ai *post*
nome_file_utenti: string	Nome del file di testo contenente l'elenco degli utenti
nome_file_post: string	Nome del file di testo contenente l'elenco degli post
nome_file_commenti: string	Nome del file di testo contenente l'elenco dei commenti di un *post*
nome_file_likes: string	Nome del file di testo contenente l'elenco dei *likes* di un post
nome_file_amicizie: string	Nome del file di testo contenente l'elenco delle amicizie di un

	utente
nome_file_profilo: string	Nome del file di testo contenente le informazioni del profilo di un utente
id_a: int	Impostazione di *default* dell'ID univoco progressivo relativo alle richieste di amicizia
id_c: int	Impostazione di *default* dell'ID univoco progressivo relativo ai commenti
id_p: int	Impostazione di *default* dell'ID univoco progressivo relativo ai post
id_u: int	Impostazione di *default* dell'ID univoco progressivo relativo agli utenti

Sitografia

- **http://www.cplusplus.com/**
  Sito di riferimento per quanto riguarda tipi, funzioni e librerie
- **http://www.cplusplus.com/forum/articles/6046/**
  Controllo input su interi
- **http://www.cplusplus.com/forum/general/12234/**
  Controllo input su interi
- **https://www.daniweb.com/software-development/cpp/threads/398829/restricting-string-input-size-c-vs-c-strings**
  Controllo lunghezza sulle stringhe in input
- **http://stackoverflow.com/questions/21017600/getch-exception-in-enter-and-backspace-keys**
  Scrivere asterischi per non visualizzare a schermo la password
- **http://www.cplusplus.com/forum/unices/25744/**
  Implementare backspace su digitazione password
- **http://en.wikipedia.org/wiki/Tab-separated_values**
  Descrizione di file .tsv