EATANDRATE

Progetto Linguaggi Dinamici

Simone Richetti, mat. 88664
Daniele Toschi, mat. 88607

EATANDRATE: INTRODUZIONE

EATANDRATE è un'applicazione Web che si propone di implementare un sito di recensioni di ristoranti e simili: si vuole dare la possibilità agli utenti di votare e recensire le varie attività e trovare il ristorante ideale mediante una ricerca con filtri e il supporto delle mappe Google Maps. I proprietari possono registrare le proprie attività con relative informazioni, aggiungere fotografie, rimanere aggiornati sulle recensioni ricevute e rispondere mediante un sistema di notifiche.

Abbiamo sviluppato la nostra applicazione utilizzando Python3 ed in particolare il framework Django, versione 1.11. Django necessita di un database per salvare e accedere alle informazioni dell'applicazione: come DBMS abbiamo scelto di utilizzare sqlite3.

I requisiti in termini di moduli Python con relative versioni sono indicati nel file *requirements.txt*, le istruzioni per l'installazione dell'applicazione e dei pacchetti, per caricare i dati contenuti nel dump del DB e per utilizzare l'applicazione sono contenuti nel README.

Nello sviluppo abbiamo pensato alla compatibilità con sistemi UNIX e Windows, non garantiamo quindi la compatibilità con sistemi differenti.

SCELTE IMPLEMENTATIVE

Riportiamo la traccia del progetto e la commentiamo per chiarire la nostra interpretazione e mostrare le scelte fatte nell'implementarla:

Applicazione Web che gestisce un sistema di recensioni in stile Tripadvisor (recensioni comprendenti sia testo che votazioni numeriche).
Presenza di utenti registrati (proprietari attività e clienti) ed anonimi:
• Gli utenti anonimi possono soltanto leggere le recensioni già presenti
• Gli utenti registrati possono inserire proprie recensioni, e possono indicare se recensioni di altri utenti sono state utili o meno (possibilità di indicazione positiva o negativa)
• Gli utenti registrati proprietari di attività possono inserire/modificare i dati relativi ad una attività – comprensivi di fotografie. Ricevono inoltre una notifica ogni volta che una delle loro strutture viene valutata e possono rispondere.

Partendo da queste indicazioni e sfruttando i meccanismi di login e permessi di Django, abbiamo così diviso le funzionalità a disposizione delle varie tipologie di utenti:

- **Utenti non loggati**: gli utenti non loggati possono vedere i profili delle attività con relative informazioni, immagini e recensioni. Possono inoltre effettuare ricerche. Non possono invece effettuare recensioni, indicare una recensione come utile/inutile, registrare le proprie attività;
- **Utenti registrati recensori/normali**: sono gli utenti che effettuano la registrazione al sito per contribuire con le proprie recensioni. Possono pertanto effettuare tutte le azioni degli utenti non loggati e in più possono recensire un'attività, indicare una recensione di un altro utente come utile/inutile e gestire il proprio profilo. Non possono però registrare un'attività;
- **Utenti registrati proprietari**: sono gli utenti che si registrano per aggiungere le proprie attività. Possono quindi registrare una o più attività con le relative informazioni, aggiungere e rimuovere fotografie, ricevere notifiche sulle recensioni e rispondere, gestire il loro profilo. A questo si aggiungono tutte le funzionalità concesse agli utenti non loggati. Non possono però recensire né indicare le recensioni come utili.

Adottare un sistema di reputazione degli utenti basato sui giudizi di utilità dati dagli altri utenti (Es. questa recensione è stata utile?)

Ad ogni utente recensore è associato un punteggio di affidabilità: alla registrazione viene dato un punteggio di partenza, che viene incrementato/decrementato ogni volta che una recensione dell'utente viene valutata come utile/inutile.

Adottare e visualizzare un sistema di ranking complessivo delle attività, pesato anche in base alla reputazione degli utenti.
Dotare il sistema di un sistema di ricerca che permetta di selezionare attività in base a diverse caratteristiche (locazione, tipologia, servizi offerti, …)

Sono stati implementati 2 tipi di ricerca: una ricerca semplice, che consiste in una ricerca testuale mediante una TextBox, e una ricerca complessa, che permette la ricerca testuale sul nome, ricerca testuale sulla città e di filtrare i risultati per tipologia di ristorante cercato.

Possibilità di visualizzare per ogni attività la tag cloud delle parole più frequenti contenute nell'insieme dei testi di tutte le recensioni.
Visualizzazione su una mappa geografica delle attività considerate

Il termine “attività considerate” lascia molta libertà nell’utilizzo delle mappe, abbiamo quindi scelto di inserirle in 2 pagine della nostra applicazione:

1. Nella pagina di ogni attività è presente una mappa con un marker nella posizione del ristorante;
2. Quando vengono visualizzati i risultati di una ricerca, sotto all’elenco delle attività è presente una mappa che contiene un marker per ogni risultato: ciascun marker mostra le informazioni basilari dell’attività associata e un link alla relativa pagina.

Sistema di recommendation basato sui giudizi dati dagli utenti (es. consiglio ad un utente una attività recensita bene da utenti che hanno dato giudizi simili ai suoi su altre attività, o che hanno caratteristiche simili a quelle da lui cercate o recensioni positive)

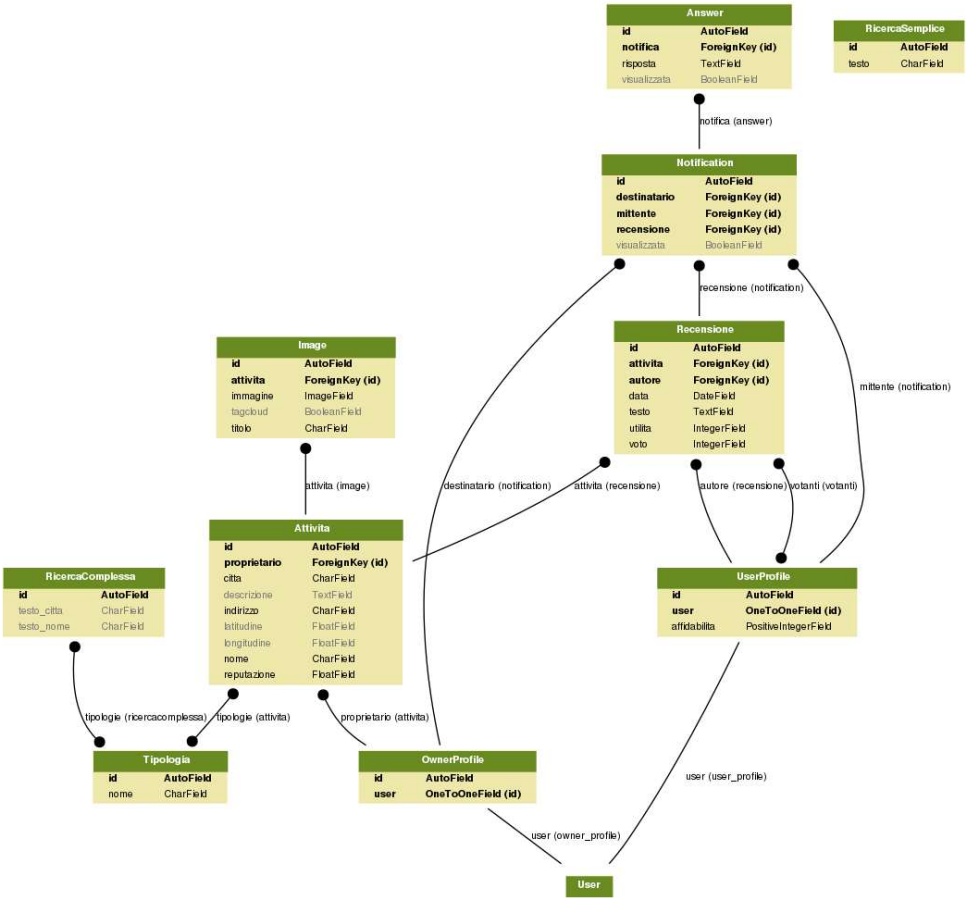
Nel sito EATANDRATE sono presenti due meccanismi di raccomandazione: uno estremamente naïve che mostra le 5 attività con reputazione più alta, l’altro invece basato sull’utente loggato (solo se recensore): vengono mostrate le attività più apprezzate dagli utenti che hanno dato giudizi simili ai suoi nelle attività che questo ha recensito.

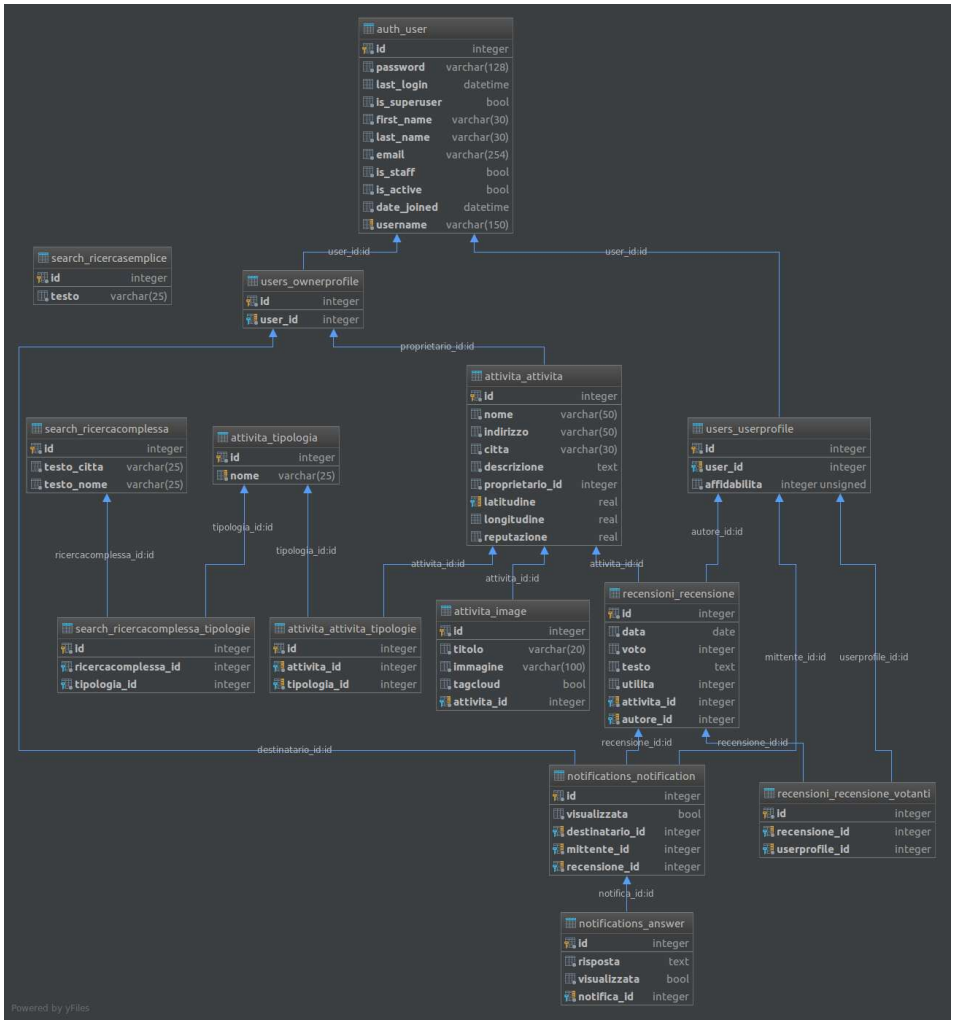
APPLICAZIONI DJANGO

Le funzionalità richieste dell’applicazione sono state implementate in maniera modulare mediante alcune applicazioni Django. Le principali sono:

- attività
- recensioni
- users
- notifications
- search

Prima di entrare nel dettaglio di ciascuna di esse, introducendo i modelli creati e le funzionalità principali con diagrammi UML delle principali operazioni e screenshot del sito, riportiamo la struttura del database e il diagramma UML delle classi più importanti per la gestione di EATANDRATE:





SOPRA, DIAGRAMMA UML DELLE CLASSI. SOTTO, STRUTTURA DEL DB

Attività

MODELS	VIEWS	
• Attività • Image • Tipologia	• index • AttView • recensisci • modify	• add • add_image • delete_image

Attività è l'applicazione principale del nostro progetto, e anche la più grande e complessa in termini di funzionalità implementate e gestite. Qui vengono gestiti:

- La creazione di una nuova attività da parte del proprietario (view *add*)
- La modifica dei dati di un'attività da parte del proprietario (view *modify*)
- L'aggiunta e l'eliminazione delle immagini di un'attività da parte del proprietario (views *add_image* e *delete_image*, model *Image*)
- Il processo di creazione di una recensione (view *recensisci*) e il conseguente aggiornamento della reputazione dell'attività
- Creazione e aggiornamento della tagcloud di un'attività (models *Attivita* e *Image*)
- La visualizzazione della pagina principale, la generazione delle attività "consigliate" e la visualizzazione delle pagine delle attività

La gestione delle immagini è integrata in questa applicazione perché legata esclusivamente alle attività. Inoltre, sono presenti controlli sui permessi dell'utente loggato legati alle azioni disponibili esclusivamente a certe tipologie di utenti (es. accesso alla view *recensisci* solo se l'utente loggato è un utente recensore, alla view *add* solo se proprietario): questi controlli sono implementati mediante decoratori per le funzioni messi a disposizione da Django.

Recensioni

MODELS	VIEWS	
• Recensione	• vote • vote_pos	• vote_neg

In questa applicazione viene implementato il modello per le recensioni e vengono gestiti i processi di indicazione di utilità/inutilità delle recensioni (views *vote_pos*/*vote_neg*): si controlla ad

esempio che la recensione non sia già stata votata dall'utente, o che l'utente non sia l'autore stesso della recensione. Infine, viene gestita la conseguente modifica all'affidabilità dell'utente.

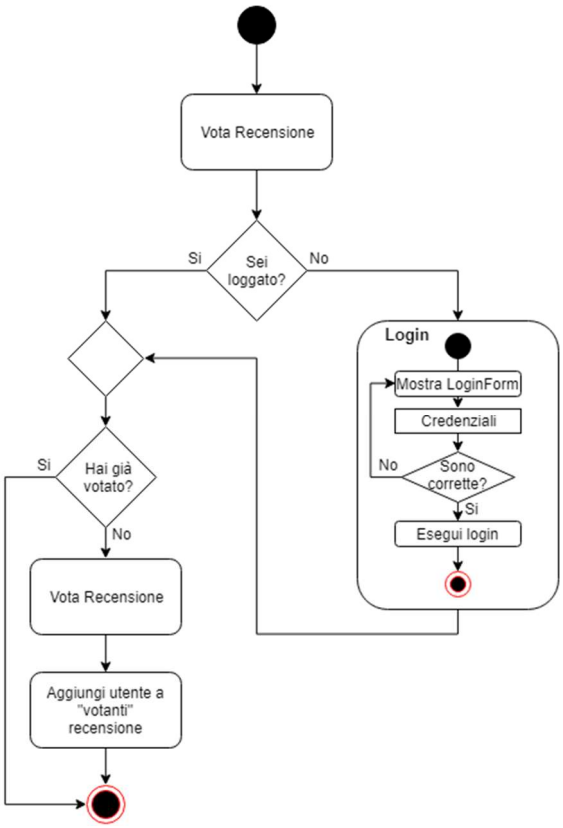


DIAGRAMMA UML DI ATTIVITÀ: VOTAZIONE DI UNA RECENSIONE

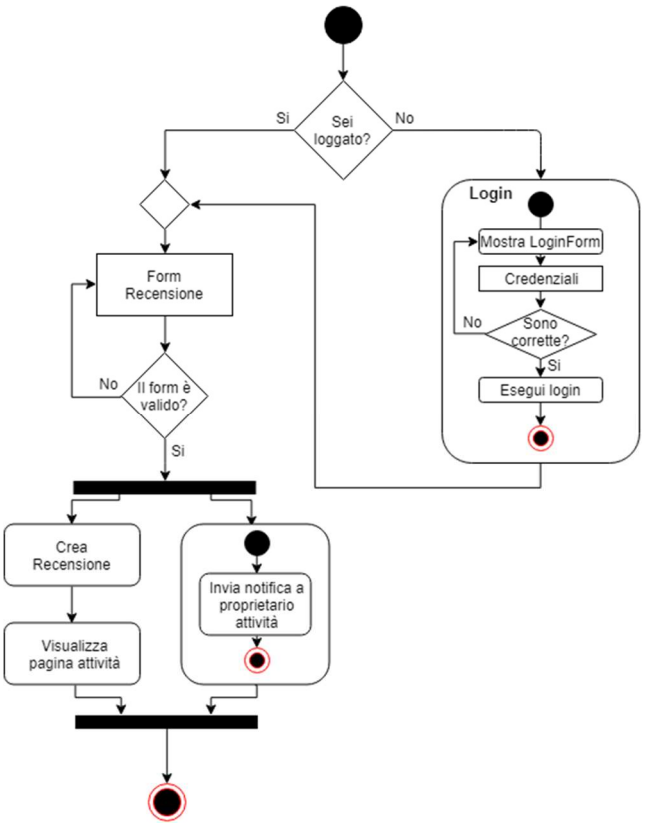


DIAGRAMMA UML DI ATTIVITÀ: CREAZIONE NUOVA RECENSIONE

Users

MODELS	VIEWS	
• UserProfile • OwnerProfile	• my_profile • user_profile • register	• register_owner • modify_profile • delete_profile

L'applicazione *users* si occupa della gestione di tutte le funzionalità legate agli utenti, come ad esempio:

- La creazione di utenti recensori o proprietari e dei profili (views *register* e *register_owner*)
- L'assegnamento dei permessi in base al tipo di utente
- La modifica dei propri dati del profilo utente (views *modify_profile*)

- La visualizzazione delle pagine degli utenti e della propria pagina del profilo (views *my_profile* e *user_profile*)
- L'eliminazione del proprio account (view *delete_profile*)

Riguardo all'eliminazione di un utente abbiamo scelto di seguire le linee guida suggerite da Django: l'utente non viene cancellato dal database, ma viene settato il suo flag *active* a *False*. Questo fa sì che tutti gli elementi che hanno l'utente come Foreign Key non vengano eliminate dal database, mantenendo di conseguenza recensioni, voti o attività, ma l'utente non potrà più effettuare il login, così da risultare a tutti gli effetti disattivato.

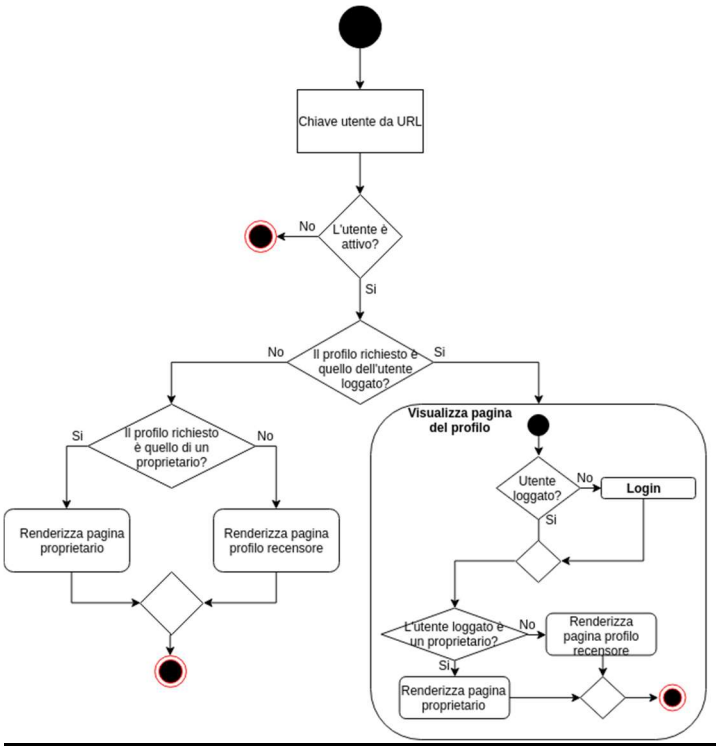


DIAGRAMMA UML DI ATTIVITÀ: VISUALIZZAZIONE DELLA PAGINA DI UN UTENTE

PAGINA DEL PROFILO (GRAFICA RESPONSIVE)

Notifications

MODELS	VIEWS	
• notification • answer	• reply • delete	• delete_notification • delete_answer

Nell'applicazione *notifications* viene gestito il sistema di notifiche. Si crea un segnale che invia automaticamente una notifica al proprietario di un'attività quando questa viene recensita (vedi file *signals.py*) e si gestisce l'eventuale risposta all'autore della recensione. Anche in questo caso,

una notifica non viene eliminata definitivamente dal database, ma si setta il flag *visualizzata* a *True* di modo che non venga più mostrata all'utente destinatario.

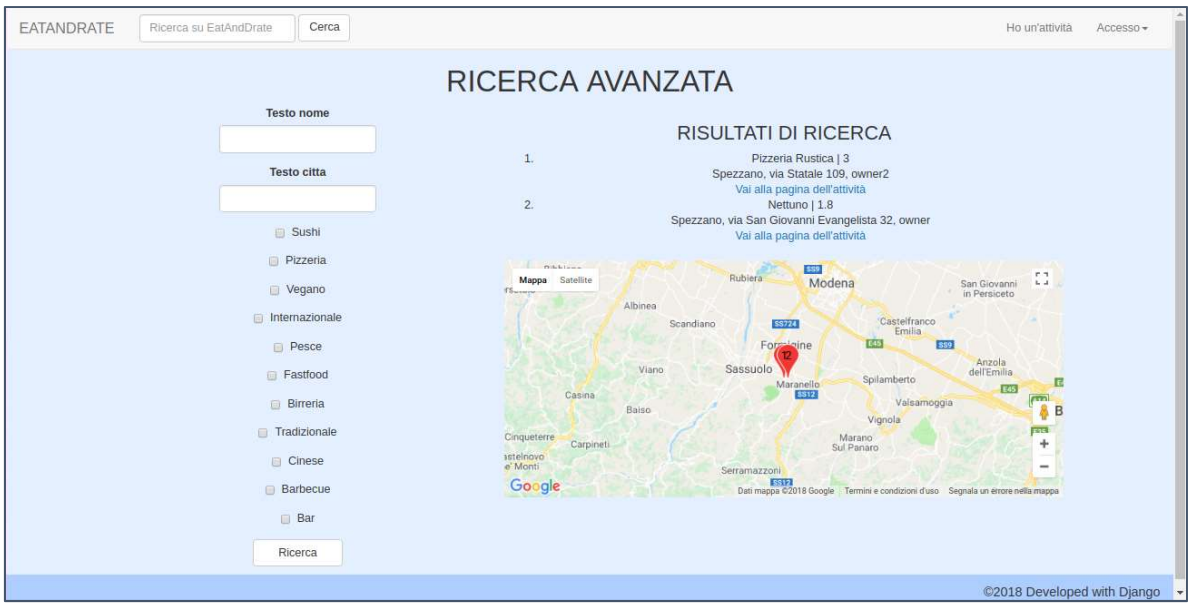
Search

MODELS	VIEWS		
• RicercaSemplice • RicercaComplessa	• simple_search • complex_search	• show_results • simple_results	• complex_results

L'applicazione *search* implementa il sistema di ricerca delle attività. Sono stati pensati 2 tipi di ricerca:

- **Ricerca semplice**: accessibile mediante una barra di ricerca situata nella navbar del sito, si tratta di una ricerca testuale e vengono mostrate tra i risultati tutte le attività che fanno matching nel nome, nell'indirizzo o nella città con la stringa cercata.
- **Ricerca complessa**: accessibile nella pagina dei risultati di un'altra ricerca e pensata come ricerca avanzata e filtraggio dei risultati: è possibile fare ricerche specifiche su nomi e città e filtrare i risultati per tipologia di appartenenza.

Le funzioni *simple_search* e *complex_search* ricevono la richiesta contenente i parametri della ricerca e creano la relativa entità nel DB, *simple_results* e *complex_results* calcolano quali sono le attività che soddisfano i parametri e infine *show_results* visualizza i risultati.



VISUALIZZAZIONE RISULTATI DI RICERCA CON MAPPA E POSSIBILITÀ DI EFFETTUARE RICERCA AVANZATA

GRAFICA

La grafica di EATANDRATE è stata realizzata principalmente integrando i template Django per il caricamento dei contenuti dal database con il framework Bootstrap (versione 3), che ci permette di fornire una grafica responsive e più user friendly al sito. Eventuali correzioni CSS alle classi Bootstrap sono state inserite direttamente nei tag HTML degli oggetti mediante l'attributo *style* piuttosto che in una stylesheet separata.

Abbiamo utilizzato anche Javascript e JQuery per implementare alcune funzionalità, quali:

- Inserimento di mappe e marker
- Contatore caratteri nella box di riposta ad una notifica

CONCLUSIONI

Le funzionalità richieste dalla traccia sono state implementate e testate, per alcune di esse sono stati creati test ad hoc nei file *tests.py*.

Eventuali sviluppi futuri possono essere sotto il punto di vista della grafica, che potrebbe essere resa più moderna e user friendly, o sotto il punto di vista delle funzionalità: aggiungere foto del profilo, salvare immagini nel database in modo da renderle più facilmente portabili, aggiungere informazioni ai profili degli utenti o potenziare lo strumento delle mappe.