Progetto Ingegneria Software

NomadBees

Gruppo T-37:

Matteo Pontalti, Matteo Bregola,

Riccardo Libanora.

Architettura

Contenuti

Scopo del Documento

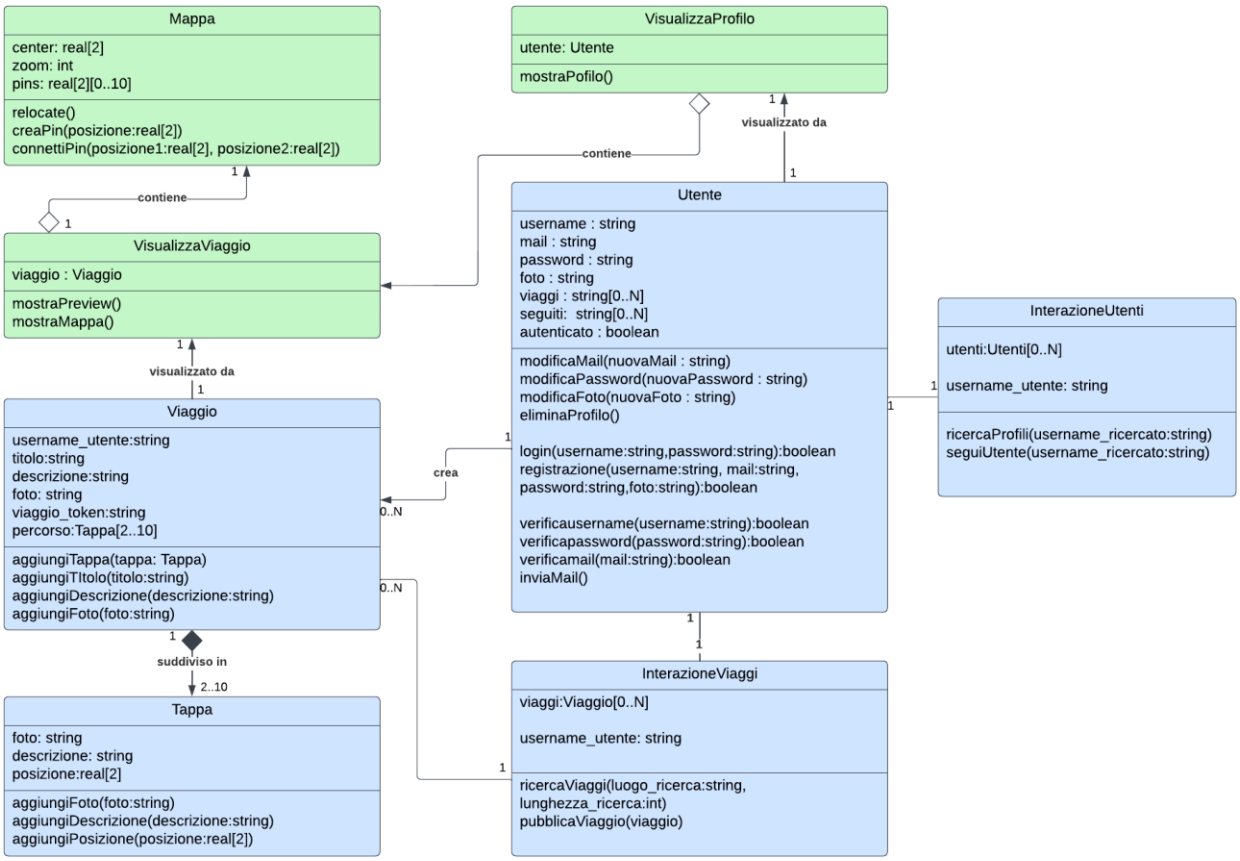
Il seguente documento presenta l'architettura del progetto NomadBees tramite l'utilizzo del diagramma delle classi in UML e codice OCL. La realizzazione di questo materiale è frutto del lavoro eseguito nei precedenti documenti, risulta quindi necessario avere una conoscenza di quest'ultimi per poter comprendere in maniera più efficacie le decisioni che sono state prese.

Diagramma delle Classi

In questa sezione sarà presentato e spiegato il diagramma delle classi del progetto. La prima immagine rappresenta il diagramma completo che permette un immediata visione d'insieme e poi successivamente verranno precisate le classi con i propri attributi e metodi. Gli attributi, locati nella porzione superiore della classe identificano i dati che la classe gestisce i quali possono provenire da interazioni con gli utenti, dal database o dati di "supporto" generati dalla classe stessa. I metodi invece sono le funzionalità che la classe utilizza internamente per gestire i suoi dati o che fornisce al sistema. Ogni classe amministra un particolare aspetto del progetto, l'unione di esse fornisce tutte le funzionalità esposte nei documenti precedenti. Affinché ciò avvenga è necessario che vi sia un dialogo tra le classi nella gestione dei dati: da qui nascono le associazioni presenti nel diagramma. Le associazioni garantiscono la comprensione delle interrelazioni tra le classi. Infine sono state inserite nel diagramma tre classi "orientate" al front-end, evidenziate in verde nel diagramma complessivo. Esse non creano o modificano dati salvati nel database ma poiché sono delle funzionalità importanti del sito riteniamo sia significativo presentarle in questa sezione.

Il diagramma è stato creato tramite l'Unified Modeling Language (UML).

Diagramma Complessivo

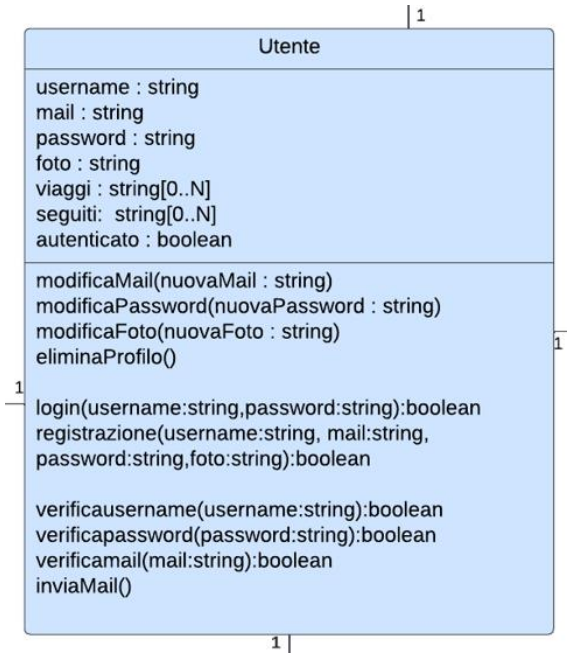


Utente

La classe Utente e Viaggio sono le due principali fondamenta del progetto *NomadBees*. Analizzando il diagramma delle componenti abbiamo notato che ogni componente gestiva dati riguardanti gli utenti o i viaggi. Il diagramma quindi è stato realizzando partendo dall'idea di mettere al centro queste due classi.

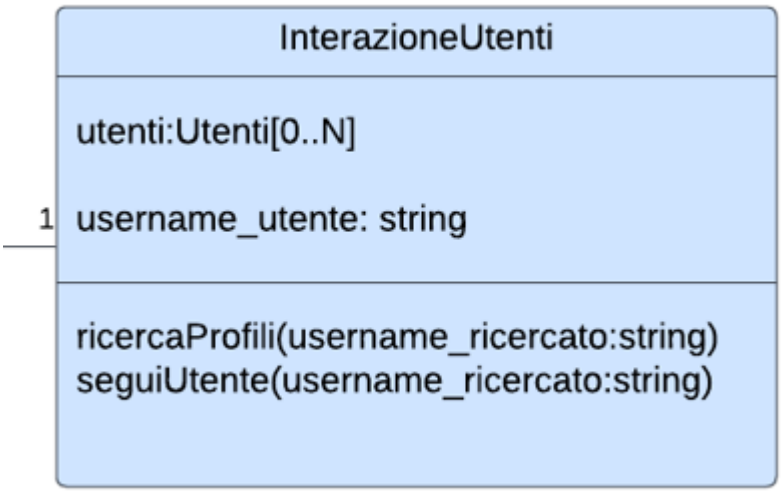
La classe utente contiene come attributi i dati collezionati durante la registrazione e l'utilizzo del sito da parte di un suo utilizzatore. Vi è inoltre un attributo *autenticato* che è stato pensato per garantire la divisione tra utenti autenticati e non e le loro rispettive funzionalità. Questo attributo può essere modificato dai metodi *login* e *registrazione*. Poiché, come specificato nei documenti precedenti, si è scelto ti rendere unici gli *username* abbiamo deciso di identificare gli utenti con quest'ultimi evitando l'utilizzo di un token.

Per quanto concerne i metodi, sono racchiuse nella classe tutte le funzionalità che permettono la modifica o l'eliminazione degli attributi sopra riportati. E' fatta eccezione per quanto riguarda l'attributo *viaggi* poiché nonostante esso sia parte delle informazioni che si hanno sull'utente, per una questione di complessità e coerenza viene gestito dalle classi *Viaggio* ed *InterazioneViaggi*. L'attributo è di tipo *string* poiché invece di salvare l'intero oggetto *Viaggio* basta salvare il token identificativo dei viaggi da lui pubblicati. Come sopra riportato sono presenti i metodi che gestiscono il *login* e la *registrazione*, quest'ultimo utilizza funzioni private di supporto per verificare che i dati inseriti nella fase di registrazione seguano i requisiti non funzionali di sicurezza riportati nel D2. Vi è inoltre il metodo *inviamail* che si occupa dell'invio di una mail di conferma per la modifica della password.



Interazione Utenti

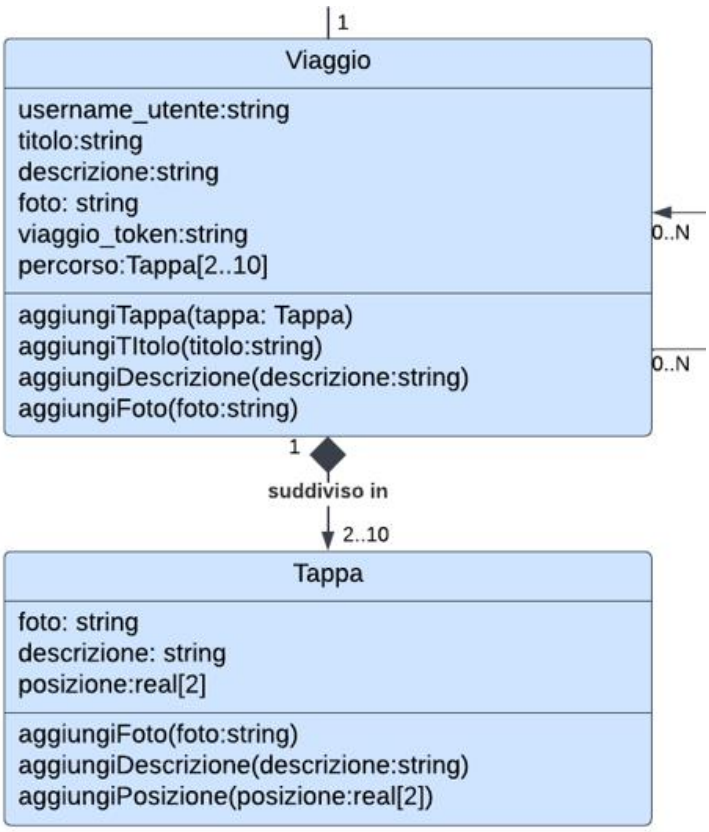
Dalla classe Utenti è nata la necessità di avere una classe che gestisse le funzionalità di interazione tra un utente e gli altri registrati al sito. La classe possiede come attributo la lista di tutti gli utenti registrati al sito e l'username dell'utente che vuole utilizzare i suoi metodi. Questo perché se un utente vuole utilizzare il metodo per seguire un utente allora tramite il suo identificativo sarà possibile aggiornare i dati dell'utente.



Viaggio

Viaggio contiene i dati che verranno salvati quando si crea un post. Durante la creazione di un post si deve aggiungere il titolo, la descrizione ed una foto. Ciò avviene tramite i tre corrispettivi metodi che prendono come parametri gli input inseriti dagli utenti. Inoltre il viaggio è formato da un minimo di 2 ad un massimo di 10 tappe, le quali possono essere aggiunte tramite il metodo *aggiungiTappa*. Il viaggio viene creato dall'interazione di uno specifico utente (del quale viene memorizzato *l'username*) con il sito, una volta che l'utente finisce la creazione del viaggio viene passato il viaggio alla classe Interazione Viaggi che ha tre compiti in relazione a questa azione:

- Determina il token identificativo del viaggio e lo salva come attributo del Viaggio;
- Salva il viaggio nella lista di tutti i viaggi;
- Salva il token viaggio nella lista dei viaggi salvati dell'utente corrispondente.

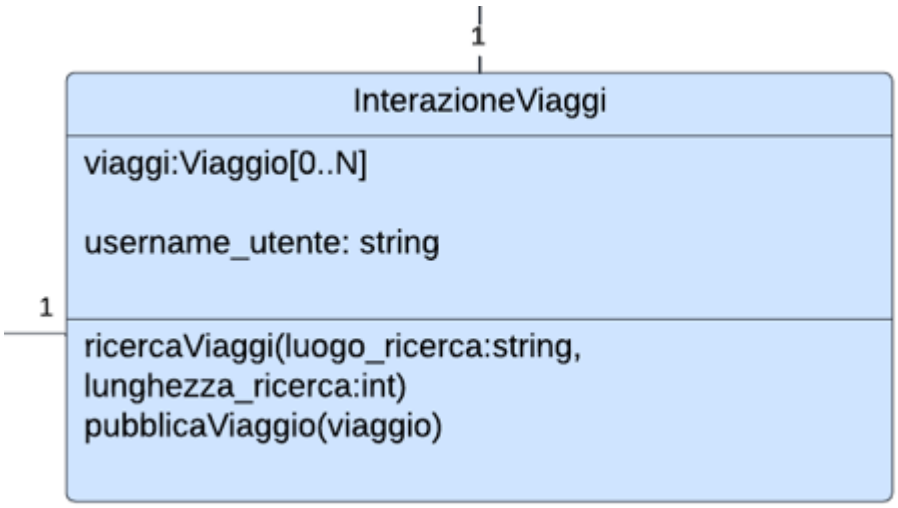


Tappa

La classe Tappa gestisce la creazione delle tappe all'interno del viaggio da parte dell'utente. *Foto* e *descrizione* sono due attributi opzionali mentre la *posizione* è obbligatorio e rappresenta le coordinate come reali. Il metodo di aggiunta della posizione verrà gestito dall'API di Google Maps.

Interazione Viaggi

La classe Interazione Viaggi si occupa della gestione di tutti i viaggi e delle funzioni che l'utente svolge su di essi. Essa infatti possiede come attributo la lista di tutti i viaggi presenti nel sito e l'username dell'utente che vuole utilizzare i metodi offerti dalla classe. Il metodo *ricercaViaggi* permette all'utente di ricercare un viaggio filtrando per nome o lunghezza del viaggio. Il metodo *pubblivaViaggi* salva il *Viaggio* creato dall'utente sia nell'insieme di tutti i viaggi del sito che in quelli dell'utente.



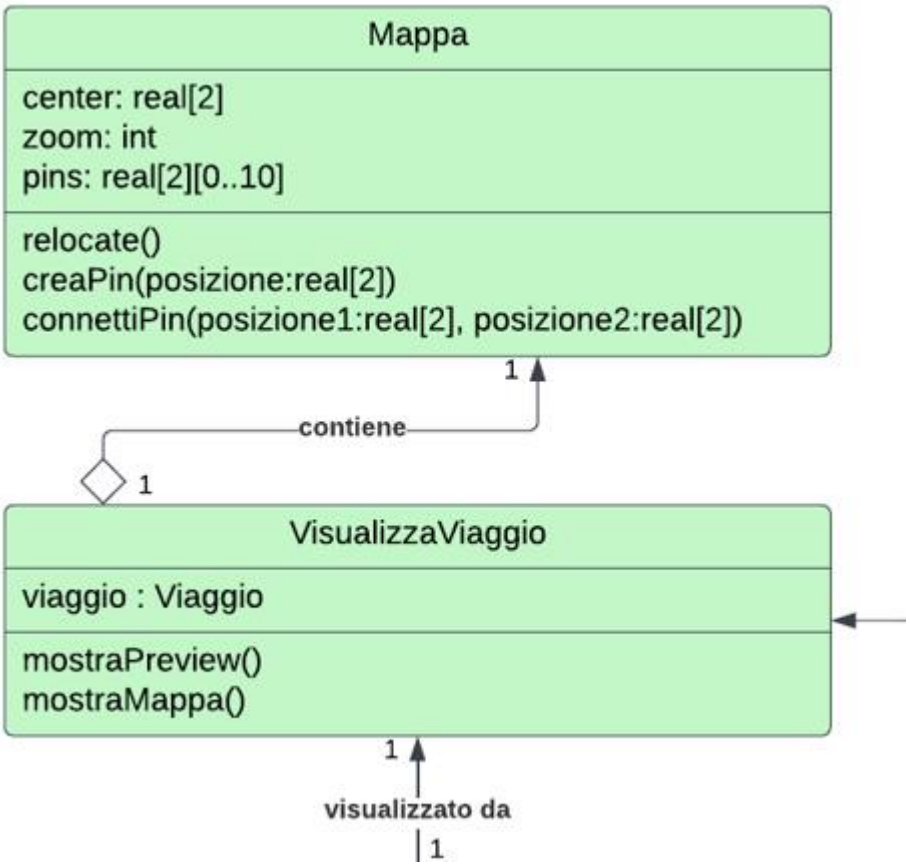
Visualizza Viaggio e Mappa

La classe *VisualizzaViaggio* si occupa di generare la preview del viaggio e sfrutta la classe *Mappa* per mostrare il percorso del viaggio. La classe Mappa viene gestita dall'API di Google Maps secondo tre attributi:

- *Center:* formato da due reali che rappresentano le coordinate del centro della mappa;
- *Zoom:* intero che indica il livello di zoom della mappa;
- *Pins:* l'insieme delle coordinate di ogni tappa.

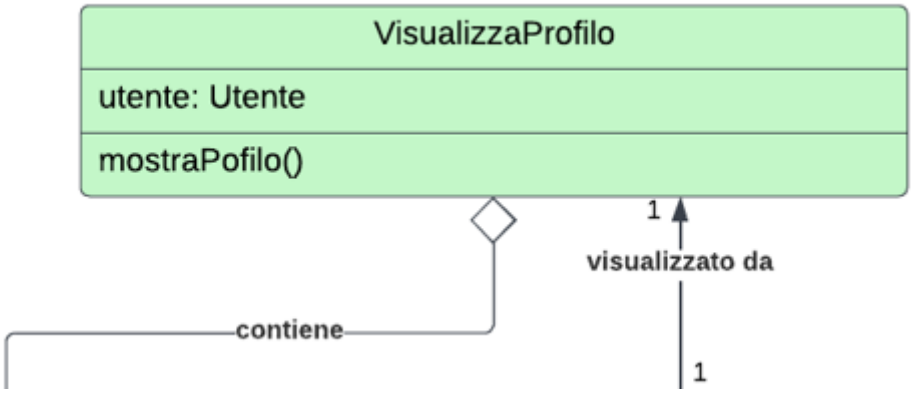
I metodi offerti sono tre:

- *Relocate:* Cambia il valore del centro della mappa ogni volta che vien aggiunto un nuovo pin;
- *CreaPin:* aggiunge un "pin" alla lista dei pins tramite i quali identifica le tappe nella mappa;
- *ConnetiPin:* connette i pin aggiunti con delle linee per mostrare il percorso.



Visualizza Profilo

La classe di visualizzazione profilo si occupa di mostrare il profilo di un utente seguito e sfrutta la classe di visualizzazione viaggio per mostrare i viaggi dell'utente.

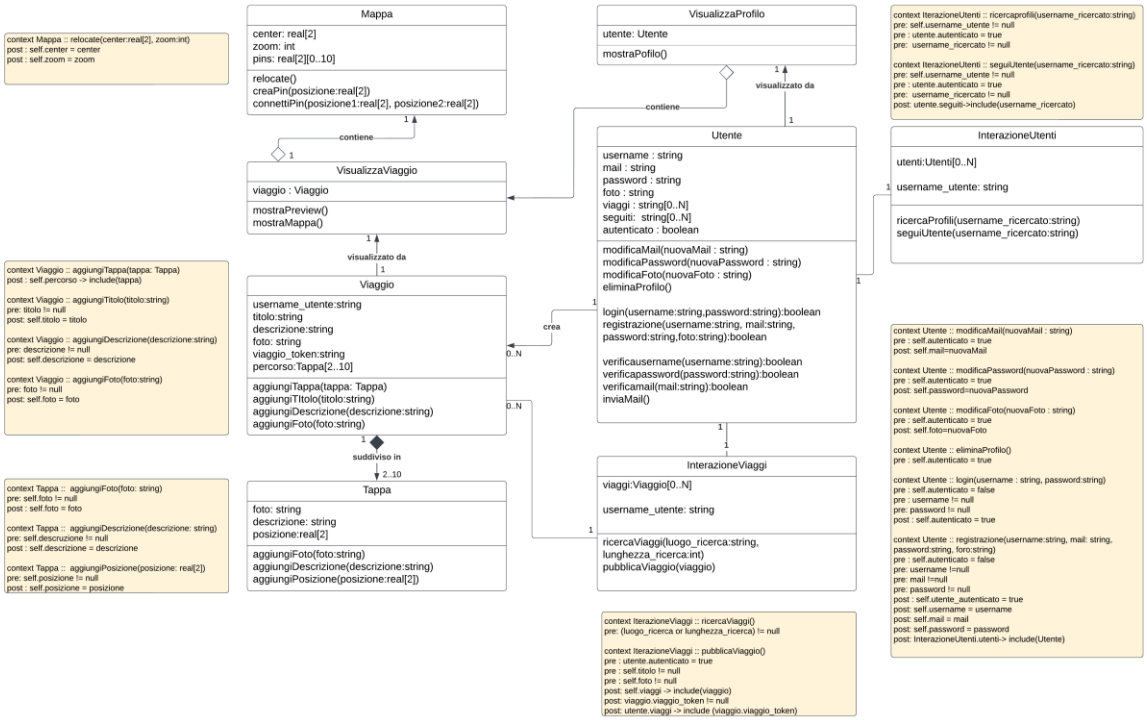


OCL

In questa sezione vengono esposte le restrizioni e determinate implicazioni logiche che regolano le interazioni logiche tra le classi e i metodi al loro interno. Questo avviene tramite un linguaggio formale che permette un'immediata ed inequivocabile comprensione delle limitazioni ovvero OCL.

Verranno presentato prima lo schema completo delle classi con le restrizioni e successivamente verranno presentati per classi soffermandosi brevemente su alcuni per aggiungere spiegazioni.

Visione Completa



Utente

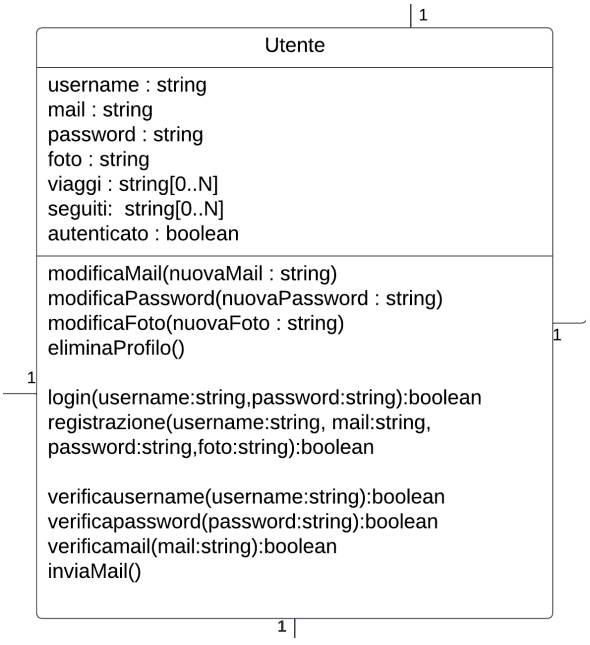
I primi tre gruppi di costraints riguardano i metodi della classe utente e sono simili in quanto ognuno ha come precondizione il fatto che gli attributi passati non debbano essere nulli e come postcondizione l'effettivo aggiornamento degli attributi dell'*Utente.* Gli altri costraints sono autoesplicativi, l'unico su cui forse vale la pena soffermarsi è il seguente:

context Utente :: registrazione(…)

post: InterazioneUtenti.utenti-> include(Utente)

Questo codice significa che una volta eseguita la registrazione è necessario passare l'utente creato alla classe *IterazioneUtenti* per poter salvarlo tra la lista degli utenti registrati.

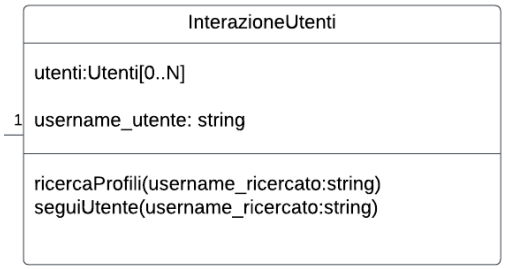
```
context Utente :: modificaMail(nuovaMail : string)
pre : self.autenticato = true
post: self.mail=nuovaMail

context Utente :: modificaPassword(nuovaPassword : string)
pre : self.autenticato = true
post: self.password=nuovaPassword

context Utente :: modificaFoto(nuovaFoto : string)
pre : self.autenticato = true
post: self.foto=nuovaFoto

context Utente :: eliminaProfilo()
pre : self.autenticato = true

context Utente :: login(username : string, password:string)
pre : self.autenticato = false
pre : username != null
pre: password != null
post : self.autenticato = true

context Utente :: registrazione(username:string, mail: string,
password:string, foro:string)
pre : self.autenticato = false
pre: username !=null
pre: mail !=null
pre: password != null
post : self.utente_autenticato = true
post: self.username = username
post: self.mail = mail
post: self.password = password
post: InterazioneUtenti.utenti-> include(Utente)
```



Interazione Utenti

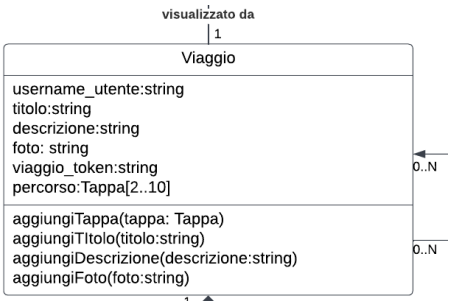
I metodi offerti da questa classe sono accessibili solo per gli utenti autenticati, quindi è necessario aggiungere questo costraint. Il costraint *post: utente.seguiti->include(username_ricercato)* indica che come post condizione del metodo *seguiUtente* è necessario aggiungere all'insieme degli utenti seguti dall'utente che sta utilizzando la funzione l'username del profile che vuole seguire.

```
context IterazioneUtenti :: ricercaprofili(username_ricercato:string)
pre: self.username_utente != null
pre : utente.autenticato = true
pre:  username_ricercato != null

context IterazioneUtenti :: seguiUtente(username_ricercato:string)
pre: self.username_utente != null
pre : utente.autenticato = true
pre:  username_ricercato != null
post: utente.seguiti->include(username_ricercato)
```

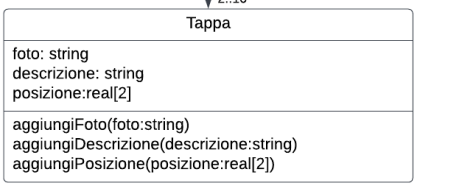


Viaggio e Tappa

```
context Viaggio :: aggiungiTappa(tappa: Tappa)
post : self.percorso -> include(tappa)

context Viaggio :: aggiungiTitolo(titolo:string)
pre: titolo != null
post: self.titolo = titolo

context Viaggio :: aggiungiDescrizione(descrizione:string)
pre: descrizione != null
post: self.descrizione = descrizione

context Viaggio :: aggiungiFoto(foto:string)
pre: foto != null
post: self.foto = foto
```



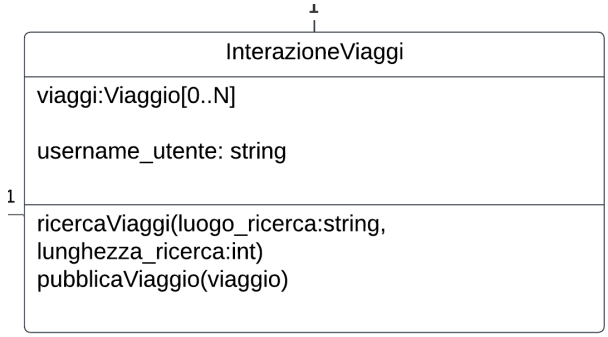

```
context Tappa ::  aggiungiFoto(foto: string)
pre: self.foto != null
post : self.foto = foto

context Tappa ::  aggiungiDescrizione(descrizione: string)
pre: self.descruzione != null
post : self.descrizione = descrizione

context Tappa ::  aggiungiPosizione(posizione: real[2])
pre: self.posizione != null
post : self.posizione = posizione
```



Iterazione Viaggi

La funzione di ricerca viaggi può avere come filtro il titolo del viaggio o la lunghezza del viaggio, quindi almeno uno dei due elementi deve essere diverso da zero. Il metodo per la pubblicazione viaggi ha come post condizione l'aggiunta del viaggio creato (nella classe *Viaggio*) tra i viaggi salvati in *viaggi* di *InterazioneViaggi* e e tra quelli dell'*Utente* (in questo caso non viene salvato il viaggio per intero ma il token all'interno della lista dei viaggi utente)

```
context IterazioneViaggi :: ricercaViaggi()
pre: (luogo_ricerca or lunghezza_ricerca) != null

context IterazioneViaggi :: pubblicaViaggio()
pre : utente.autenticato = true
pre : self.titolo != null
pre : self.foto != null
post: self.viaggi -> include(viaggio)
post: viaggio.viaggio_token != null
post: utente.viaggi -> include (viaggio.viaggio_token)
```



Mappa

```
context Mappa :: relocate(center:real[2], zoom:int)
post : self.center = center
post : self.zoom = zoom
```