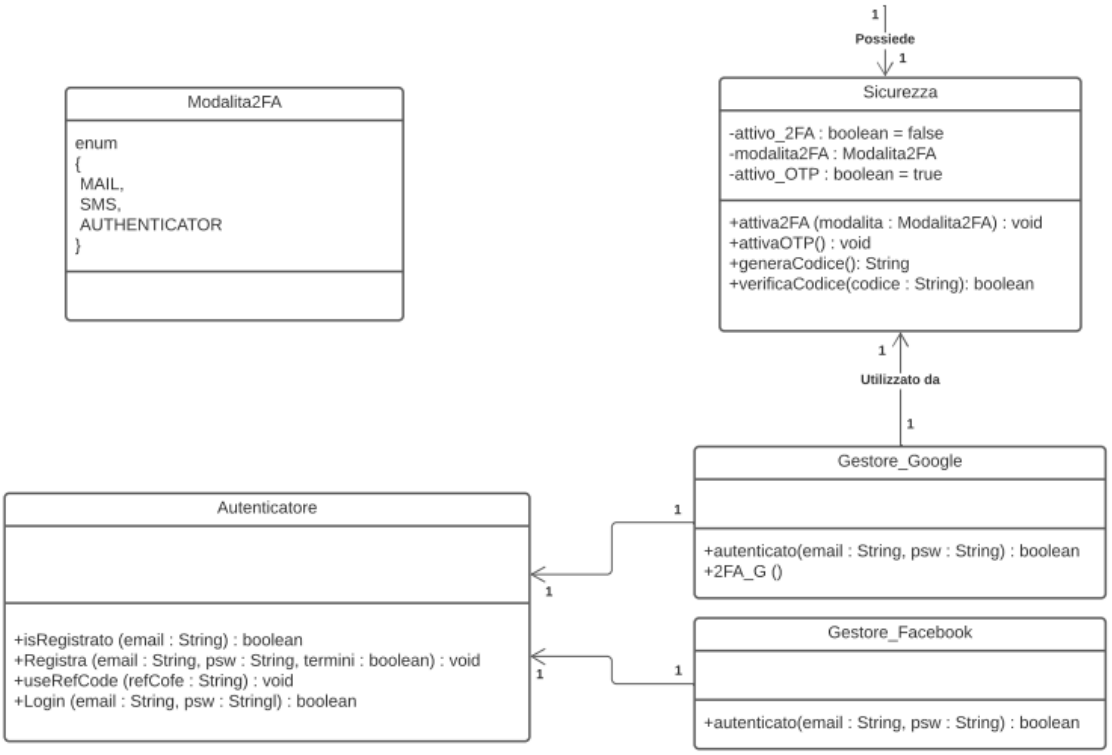
Scopo del documento:

In questo documento verrà definita l'architettura necessaria per la creazione dell'applicazione **Work4Me**. Per descrivere queste informazioni in maniera più formale, in supporto al linguaggio naturale, verranno utilizzati dei diagrammi delle classi e del codice Object Constraint Language (OCL).
Utilizzando i documenti precedenti, e in particolar modo del documento D3 contenente il diagramma di contesto dell'applicazione, verranno definite le classi da implementare e i vincoli da porre alle stesse. Le classi e i vincoli verranno rappresentate rispettivamente attraverso il diagramma delle Classi e tramite il linguaggio OCL.

1. Diagramma delle Classi

In questo capitolo verranno rappresentate le classi richieste per la realizzazione del sistema software Work4Me. Ognuna delle classi verrà identificata da un nome, da una serie di attributi e da dei metodi. Gli attributi sono caratterizzati da un nome, un tipo e rappresentano i dati gestiti da ogni classe. I metodi invece rappresentano le operazioni eseguibili da ogni classe e vengono caratterizzati da un nome, dei dati in input, ognuno con il proprio tipo, e un dato in output, anch'esso con il relativo dato.
Ogni classe può essere collegata ad altre classi tramite delle associazioni, che rappresentano come queste si relazionano tra loro.
Di seguito verranno elencate le classi individuate, una loro descrizione e la loro rappresentazione nel diagramma delle classi

1.1. Autenticazione:



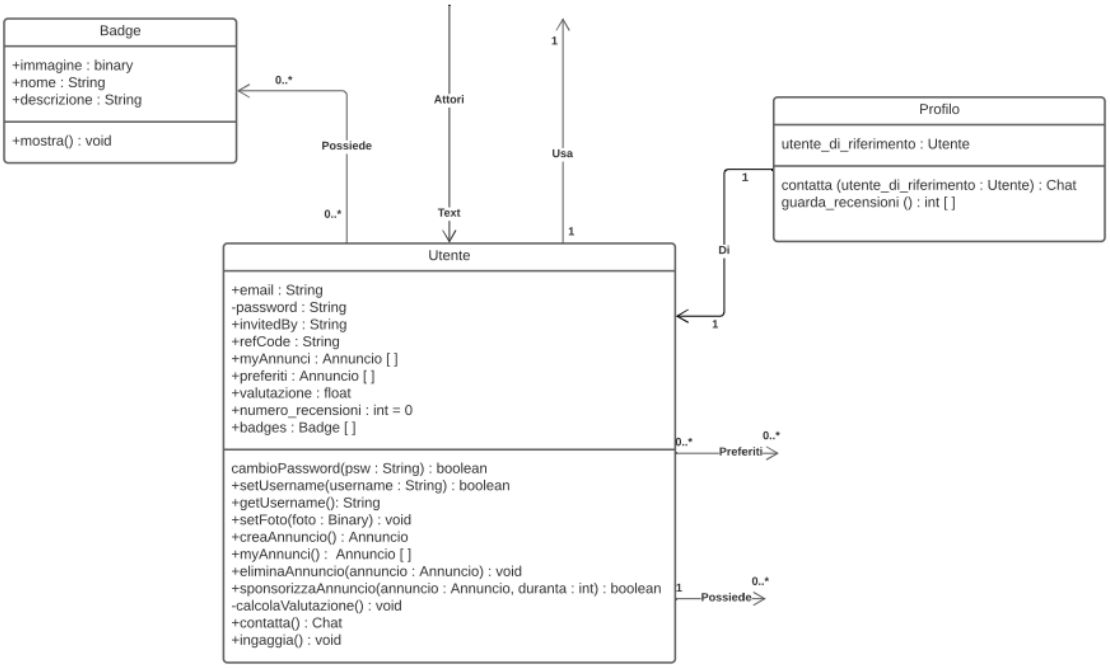
Nel diagramma di contesto l'utente ha la possibilità di autenticarsi con diversi mezzi quali la combinazione username e password, account Google e account Facebook.

Questa funzionalità è stata quindi rappresentata all'interno del diagramma delle classi tramite la classe Autenticatore, Gestore Facebook e Gestore Google.
La classe **Autenticazione** non presenta attributi ma solamente dei metodi che le permettono di effettuare la registrazione di un nuovo utente ed effettuare il Login. Sarà inoltre in grado di verificare se una combinazione di email e password date corrispondano ad un utente già registrato.
Abbiamo inoltre creato delle classi apposite che permettono di interfacciarsi con **Facebook** e **Android** per l'accesso e la creazione di un account; in aggiunta **Google** offre anche il servizio 2FA tramite l'applicazione proprietaria Authenticator.

1.2. Sicurezza:

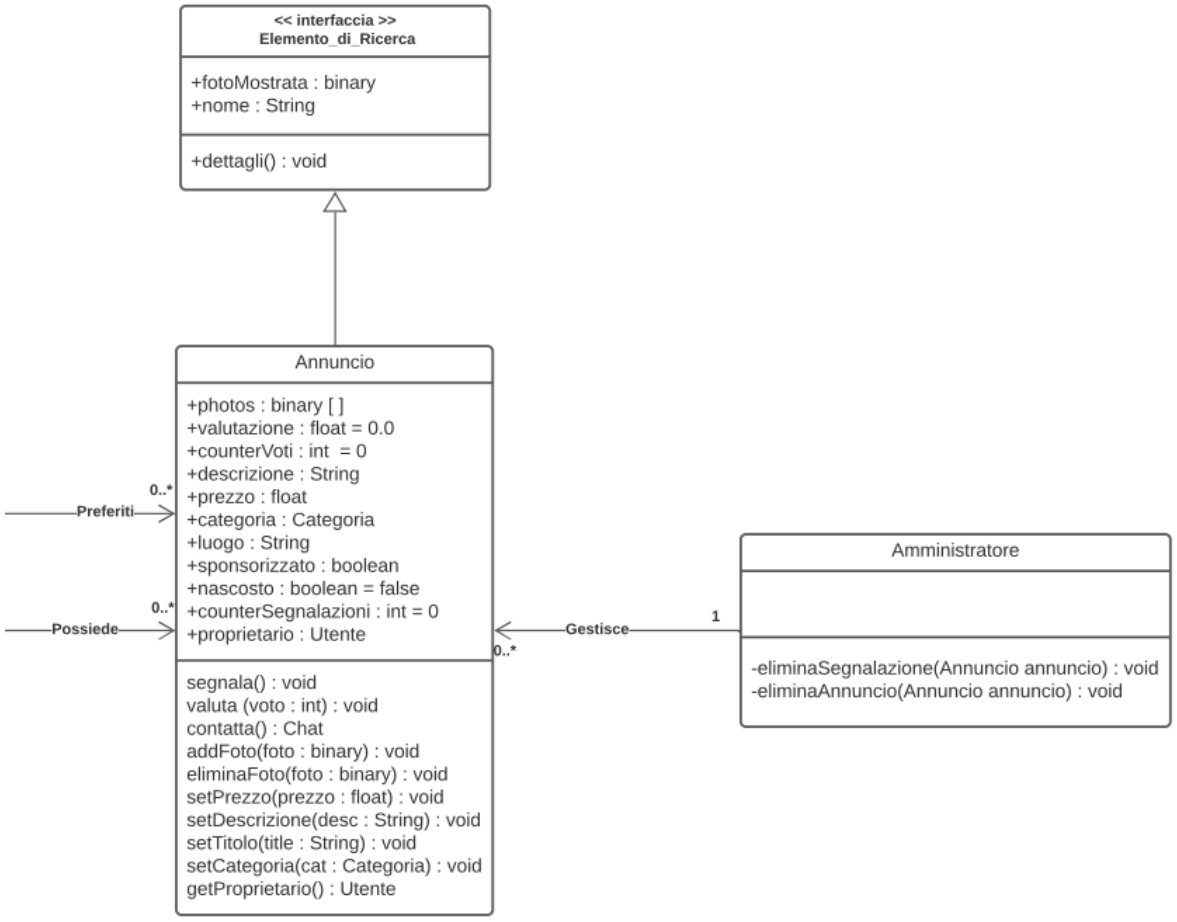
L'applicazione deve inoltre garantire un certo livello di sicurezza durante l'accesso e il reset della password. Queste funzionalità sono state rappresentate nel diagramma delle classi all'interno di Sicurezza. Questo componente permette di scegliere la tipologia di sicurezza desiderata, attivando la tecnica **2FA** o quella **OTP**. Sarà poi possibile generare un codice di sicurezza e verificarlo.
In particolar modo il metodo per l'attivazione del 2FA richiede di specificare il tipo di modalità tra le varie disponibili. Per queste informazioni verrà utilizzata una apposita enumeration **Modalita2FA**

1.3. Utente:



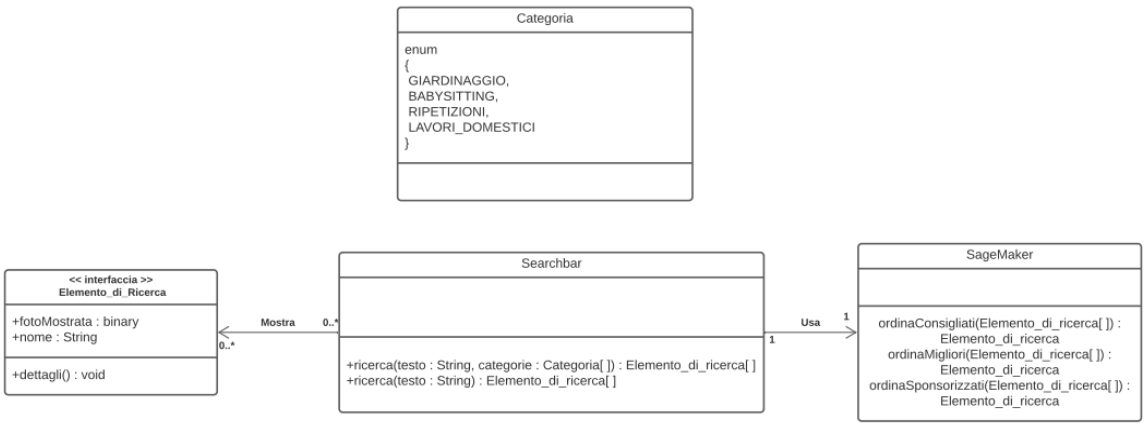
La classe **Utente** contiene tutte le informazioni personali di quest'ultimo, gli annunci pubblicati, quelli preferiti, i badge e il referral code. La classe Utente permette di modificare i dati personali e di creare e eliminare i propri annunci.
Implementerà poi il sistema di valutazione utente che è formato dalla media delle valutazioni degli annunci calcolata tramite l'apposito metodo.
Ad ogni utente verrà poi associata anche una classe **Profilo**. Questa classe è una specializzazione della classe interfaccia Elemento di ricerca, dalla quale eredita alcuni attributi e dei metodi. L'interfaccia **Elemento di ricerca** rappresenta le informazioni mostrate durante una ricerca tramite la search bar, come un'anteprima della pagina dell'utente. Ogni anteprima conterrà il nome utente e la sua immagine di profilo e sarà possibile passare direttamente alla chat con l'utente in questione.
Ad ogni utente verranno assegnati un certo numero di **Badge**, gestiti dall'omonima classe, ognuno dei quali caratterizzato da un titolo, una breve descrizione e un'immagine.

1.4. Annuncio e Amministrazione:



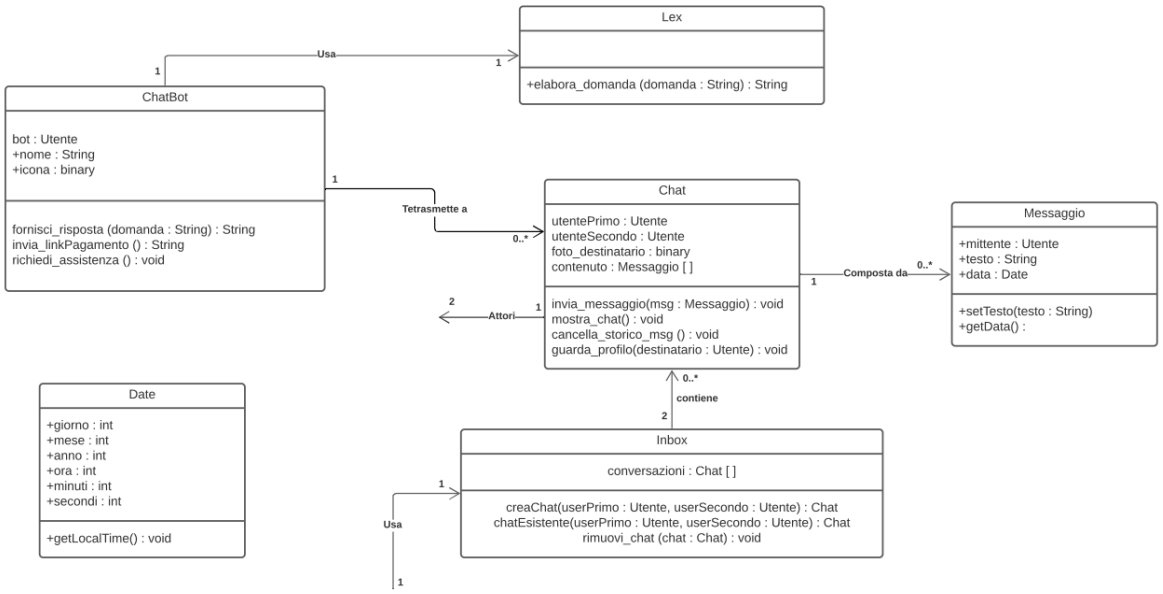
Come specificato precedentemente l'utente potrà creare svariati annunci, ognuno dei quali conterrà un riferimento al suo creatore, le sue caratteristiche (foto, prezzo, categoria, valutazione...) e vari metodi per la sua gestione.
Fornirà inoltre i metodi per permettere agli utenti di valutare e segnalare l'annuncio in questione e la possibilità di mettersi in contatto con il creatore dell'annuncio
Se l'**Annuncio** risultasse inappropriato allora potrà essere segnalato, il conto delle segnalazioni viene tenuto internamente all'annuncio.
Esiste inoltre una classe **Amministratore**, con il compito di eliminare o riabilitare l'annuncio in caso questa riceva un certo numero di segnalazioni, utilizzando due appositi metodi.
Anche l'Annuncio "Is-A" **Elemento di ricerca** dove l'attributo nome rappresenta il titolo dell'annuncio, mentre la foto è la prima foto inserita nello stesso.

1.5. Searchbar:



Gli utenti dovranno poter ricercare all'interno dell'applicativo i **Profili** di altri utenti e degli **Annunci**. Per compiere queste operazioni verrà utilizzata la classe **Searchbar**, che permetterà la ricerca tramite parole chiave e con la possibilità di aggiungere dei filtri agli elementi di ricerca mostrati. Per selezionare le categorie all'interno di una ricerca verrà utilizzata un'apposita enumeration **Categorie**
I vari annunci visualizzati verranno inoltre ordinati in tre sottocategorie, "Suggeriti", "I più richiesti" e "Sponsorizzati", attraverso i sistemi offerti da **SageMaker.**

1.6. Chat:



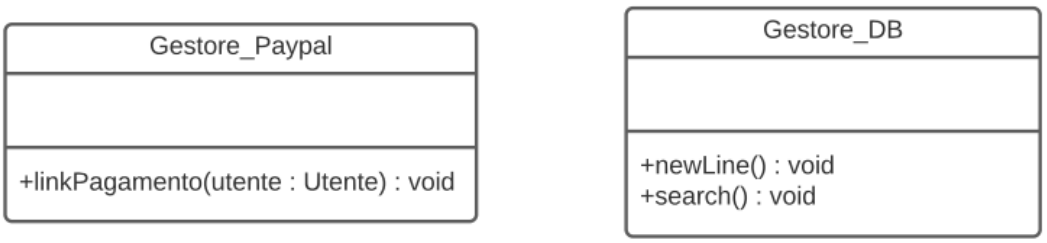
Come specificato nei documenti precedenti gli utenti devono poter comunicare tra loro grazie ad una chat interna all'applicazione. Nel diagramma delle classi questa funzionalità è ottenuta tramite le classi Inbox, Chat e Messaggio.
Inbox contiene le varie Chat e ne permette la loro creazione, eliminazione e darà la possibilità di cercare se una determinata Chat con un'altro utente è già stata creata o meno.
La **Chat** è definita da due utenti e la lista dei messaggi inviati, permette l'invio dei messaggi, la loro eliminazione e la possibilità di passare al profilo utente di uno dei due membri della chat. Ogni utente potrà avere più chat, ma potrà avere al massimo una ed una sola chat verso un altro utente.
Messaggio rappresenta ogni singolo messaggio inviato contenente un riferimento al mittente, il testo del suo contenuto e la data di invio.
Verrà quindi utilizzata una classe ausiliaria Date, che verrà utilizzata per rappresentare le informazioni relative alla data (giorno, mese, anno, ora, minuti, secondi) e darà la possibilità di recuperare la data attuale.

1.7. ChatBot:

Come riportato in precedenza, l'applicativo dovrà rendere disponibile un servizio di messaggistica con un assistente virtuale, il **ChatBot**. Questo dovrà ricevere dagli utenti delle domande su argomenti vari alle quali fornirà delle risposte automatiche adatte. Per fare ciò al chatbot verrà assegnato un apposito utente creato in precedenza e che utilizzerà per scambiare messaggi.
Per queste operazioni si appoggia, tramite degli appositi metodi API, alla classe **Lex**, la quale si interfaccia ai sistemi di Amazon per riorganizzare, grazie all'IA di Amazon Lex, le domande poste dagli utenti e per riadattare le richieste e trovare una risposta adatta.
Ultima attività svolta dal ChatBot, gestita dalla funzione invia_linkPagamento() appoggiandosi alle API **PayPal**, è l'invio del link per il pagamento di un altro utente,

sempre attraverso i servizi di messaggistica offerti dalle classi al punto precedente.

1.8. Classi ausiliarie:



Vengono infine utilizzate due classi **Gestore_DB** e **Gestore_Paypal** che avranno lo scopo di interfacciarsi rispettivamente ai sistemi del database, per l'inserimento e ricerca di informazioni nello stesso, e ai sistemi di Paypal, per la creazione dei link per i pagamenti in seguito all'ingaggio di un certo utente.

2. Codice in Object Constraint Language

In questo capitolo si andrà a descrivere i vincoli previsti per alcune operazioni delle classi precedentemente descritte. Per fare ciò utilizzeremo il linguaggio OCL (Object Constraint Language) dato che questi particolari concetti non sono esprimibili tramite UML.

2.1. Sicurezza:

Nell'ambito della sicurezza, saranno disponibili due modalità di sicurezza: 2FA e OTP. In ogni momento, non sarà possibile non avere nessuna o entrambe le modalità attive

```
context Sicurezza::attiva2FA()
post : self.attivo_OTP = false && self.attivo_2FA = true

context Sicurezza::attivaOTP()
post : self.attivo_2FA = false && self.attivo_OTP = true
```

2.2. Utente:

Per quanto riguarda la classe Utente, sono presenti alcuni vincoli per limitare le sue funzionalità solo agli utenti registrati. Per fare ciò si effettueranno dei controlli su l'username dell'utente in uso, e in caso questo sia "null", non gli sarà possibile fare una serie di attività.

context Utente::cambioPassword()
pre : self.email != null

context Utente::setUsername()
pre : self.email != null

context Utente::setFoto()
pre : self.email != null

context Utente::creaAnnuncio()
pre : self.email != null

context Utente::eliminaAnnuncio()
pre : self.email != null

context Utente::myAnnunci()
pre : self.email != null

2.3. Annuncio:

Nella classe Annuncio verranno assegnati alcuni vincoli dovuti alle limitazioni poste sui metodi che modificano le caratteristiche, quali prezzo, descrizione, foto e così via, o la rimozione delle inserzioni. In questo modo solo il proprietario dello stesso può accedervi.

context Utente::eliminaAnnuncio()
pre : self.username == Annuncio.proprietario.getUsername()

context Annuncio::addFoto()
pre : self.proprietario.getUsername() == utente.getUsername()

context Annuncio::eliminaFoto()
pre : self.proprietario.getUsername() == utente.getUsername()

context Annuncio::setPrezzo()
pre : self.proprietario.getUsername() == utente.getUsername()

context Annuncio::setDescrizione()
pre : self.proprietario.getUsername() == utente.getUsername()

context Annuncio::setTitolo()
pre : self.proprietario.getUsername() == utente.getUsername()

context Annuncio::setCategoria()
pre : self.proprietario.getUsername() == utente.getUsername()

Sono inoltre presenti dei vincoli in modo tale che il proprietario di un annuncio non possa auto-ingaggiarsi o auto-contattarsi.

context Annuncio::valuta()
pre : self.proprietario.getUsername() != utente.getUsername()

context Annuncio::contatta()
pre : self.proprietario.getUsername() != utente.getUsername()

context Annuncio::segnala()
pre : self.proprietario.getUsername() != utente.getUsername()

Oltre ai vincoli precedenti, viene posto un invariante secondo cui se un annuncio raggiunge il numero di 10 segnalazioni viene nascosto dalle ricerche e viene messo in attesa di revisione, altrimenti rimane visibile.

context Annuncio inv:
(nascosto == true && counterSegnalazioni > 9) || (nascosto == false && counterSegnalazioni < 10)

2.4. Amministratore

Vengono posti dei vincoli relativi alle segnalazioni anche nella classe Amministratore. Questi regolano i metodi per rifiutare una segnalazione, in modo che una volta che questo è stato chiamato, il contatore apposito venga resettato.

context Amministratore :: eliminaSegnalazione()
post : Annuncio.counterSegnalazioni = 0

2.5. Gestore_Google

In questa classe viene posto un vincolo in modo che i servizi di Google Authenticator vengano erogati solo se l'apposita modalità 2FA è attiva

context Gestore_Google :: 2FA_G ()
pre : Sicurezza.modalita2FA == AUTHENTICATOR

3. Diagramma delle Classi con codice OCL

In questo capitolo finale viene riportato il diagramma delle classi completo, con l'aggiunta del linguaggio OCL già descritto nel capitolo 2.