SpottyThings

Documento di architettura.

Indice

Scopo del documento....3
Diagramma delle classi....4
Classi di supporto....4
Classi degli utenti del sistema e per l'autenticazione....5
Classi profilo utente e annuncio....7
Classi di ricerca annunci, risultati di ricerca e pagamento....9
Classi di interfacciamento verso i sistemi esterni di chat e mail....11
Classi di interfacciamento al database....12
Diagramma delle classi completo....13
Codice in Object Constraint Language....14
Vincoli OCL classe *GestioneRegistrazione*....14
Vincoli OCL classe GestioneAutenticazione....14
Vincoli OCL classe *PubblicazioneAnnuncio*....14
Vincoli OCL classe *GestioneRappresentazioneProfiloUtenteEsterno*....14
Vincoli OCL classe *Utente*....15
Vincoli OCL classe *LogUtente*....15
Vincoli OCL classe *Profilo*....15
Vincoli OCL classe *Statistiche*....15
Vincoli OCL classe *Amministratore*....16
Vincoli OCL classe *Transazione*....16
Vincoli OCL classe *InterfacciaSistemaPagamento*....16
Vincoli OCL classe *Recensione*....16
Vincoli OCL classe *Annuncio*....17
Vincoli OCL classe *SchedaProdotto*....17
Vincoli OCL classe *Filtro*....17
Vincoli OCL classe *Data*....17
Vincoli OCL classe *Timestamp*....17
Vincoli OCL classe *Periodo*....18
Vincoli OCL classe *Price*....18
Vincoli OCL classe *Ordinamento*....18
Diagramma delle classi con codice in OCL associato....19

Scopo del documento

Il presente documento ha lo scopo di definire l'architettura del sistema *SpottyThings*: verrà definita l'architettura del sistema mediante l'utilizzo di un diagramma delle classi.
Il diagramma delle classi rappresenta un'ulteriore salto più nel dettaglio per la specifica dei vari componenti e servizi che dovranno soddisfare i requisiti funzionali del nostro progetto, espressi nel documento *D1-T25* e ulteriormente dettagliati nel *D2-T25* con il diagramma degli use case.
Infine si userà del codice in OCL (Object Constraint Language) per definire alcuni vincoli logici a cui dovranno sottostare gli attributi e le operazioni delle classi.

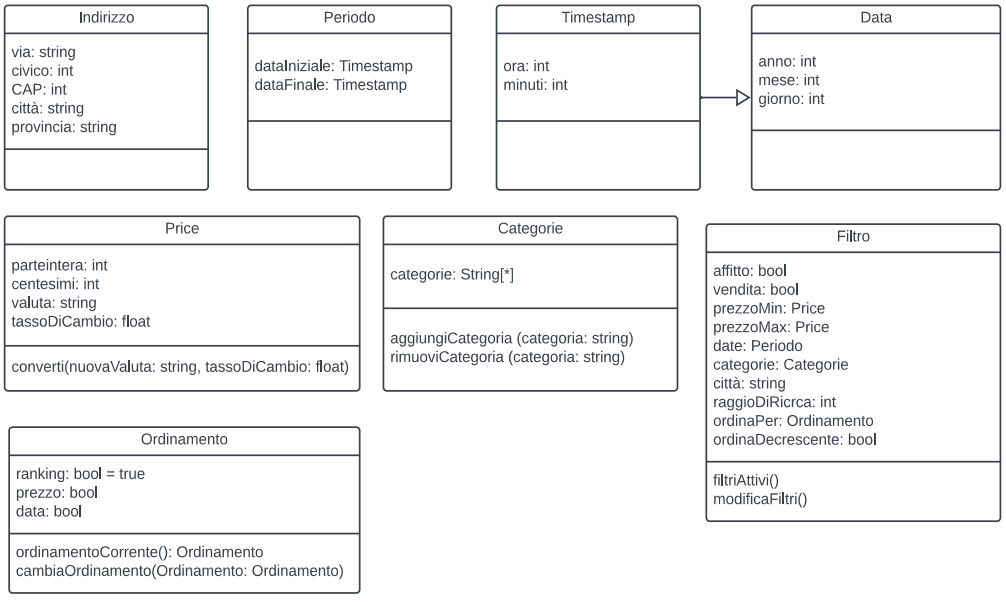
Diagramma delle classi

Nel presente capitolo rappresentiamo il nostro diagramma delle classi, raggruppandolo in gruppi di una o più classi per permettere una migliore rappresentazione.
Ogni classe, che rappresenta un ambito operativo del nostro sistema, è costituita da attributi, ovvero i dati con cui andare ad operare, e operazioni, che rappresentano le funzioni che verranno implementate a livello di codice.

Di seguito offriamo una breve descrizione di ogni classe, spiegando le operazioni e le associazioni meno intuibili.
Come già detto, per fornire migliore leggibilità rappresentiamo ogni classe rappresentata in un gruppetto di altre classi. Le classi contornate con un bordo tratteggiato sono classi con cui è presente un'associazione tra le classi descritte, ma che saranno dettagliate in seguito.

Classi di supporto

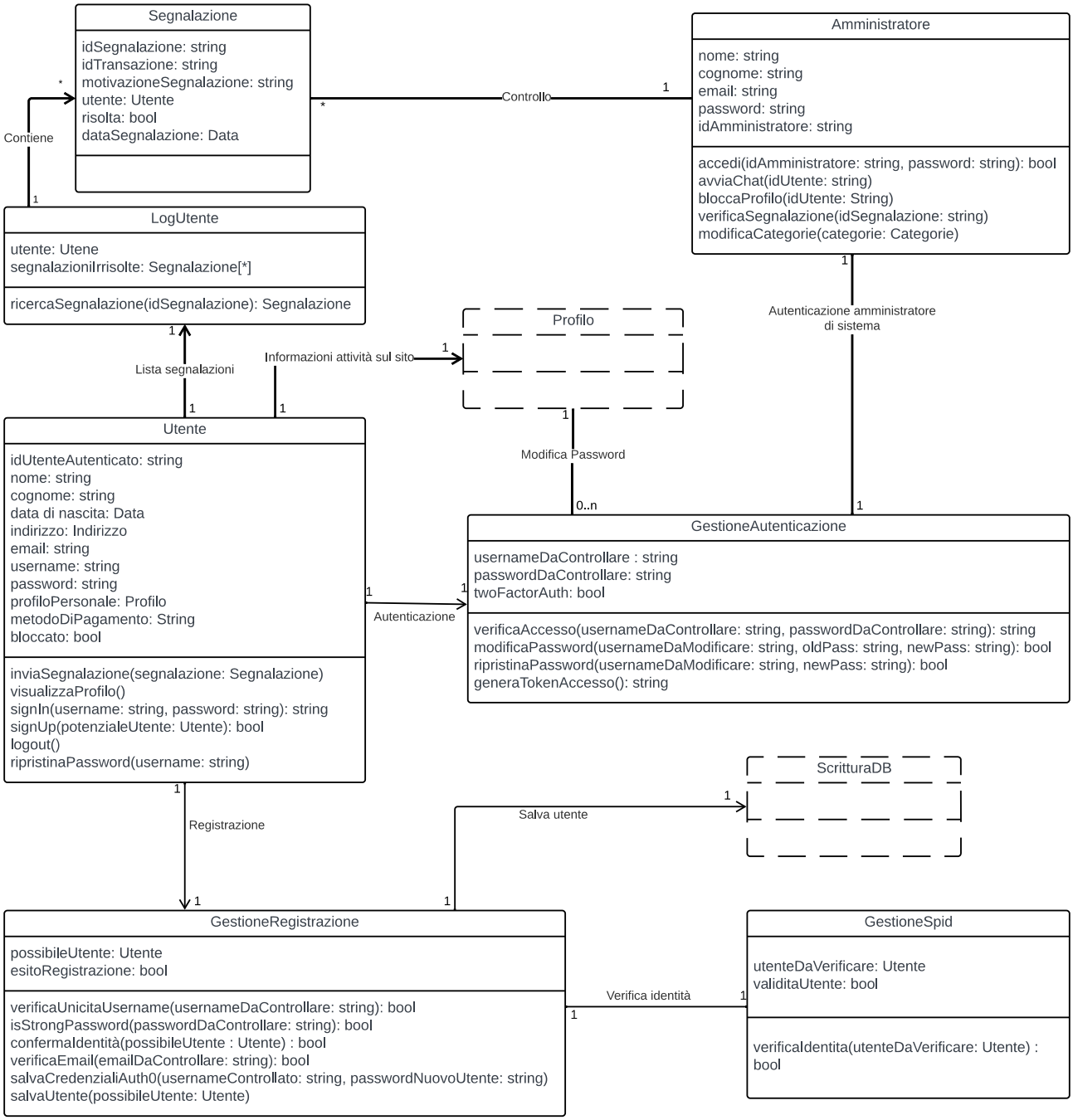
Le seguenti classi sono definite di supporto in quanto permettono di definire tipi di dato necessari nelle altre classi.
La classe *Data* permette di rappresentare le date in formato giorno, mese, anno.
La classe *TimeStamp* specializza la classe data aggiungendo gli attributi ore e minuti (ed ereditandone tutti gli altri. Serve per rappresentare il preciso momento in cui avviene una transazione).
La classe *Periodo* identifica un periodo temporale compreso tra due istanti.
La classe *Indirizzo* permette di salvare le informazioni relative ad un preciso indirizzo (di un utente oppure di un qualsiasi indirizzo generico).
La classe *Price* permette di rappresentare in modo ottimale il prezzo di un prodotto, potendo anche distinguere tra diverse valute.
La classe *Categorie* è usata per rappresentare le varie categorie di prodotti esistenti sul nostro applicativo e per gestire la modifica dell'insieme delle varie categorie da parte dell'amministratore del sito.
La classe *Ordinamento* serve a tenere traccia del corretto ordinamento in cui viene mostrata la lista di annunci ricercati dall'utente.
La classe *Filtro* serve infine per tenere traccia dei filtri applicati all'utente nella ricerca degli annunci.



Classi degli utenti del sistema e per l'autenticazione

La classe Utente è necessaria per salvare tutte i dati relativi ad un preciso utente registrato nel nostro sistema.
Oltre agli attributi che si necessita salvare per tenere traccia degli utenti, c'è anche l'attributo *idUtenteAutenticato* che rappresenta il token d'accesso della sessione dell'utente (generato dopo il login dalla classe *GestioneAutenticazione)*: se questo attributo non è NULL l'utente è attualmente loggato, altrimenti la sua sessione è scaduta ed è necessario effettuare di nuovo l'accesso.
Ad ogni utente è associato un profilo, per salvare le varie informazioni riguardanti le attività che intraprende nel nostro sito web. Per questo motivo è presente un'associazione con la classe *Profilo*.
Infine in questa classe c'è il metodo *signUp* per effettuare la registrazione e il metodo *signIn* per effettuare il login.
Ogni volta che un utente si registra si esegue il metodo *signUp* e si crea un'istanza della classe *GestioneRegistrazione*, in cui questa si verificano le validità dei dati inseriti dall'utente con opportuni metodi e in caso di esito positivo si procede al salvataggio dell'utente nel database (effettuato dalla classe *ScritturaDB*, descritta in fondo, come evidenziato dall'associazione) e nei database di Auth0. Inoltre c'è un'associazione con la classe *GestioneSpid* che si interfaccia con il sistema esterno SPID per verificare l'identità dell'utente.
Ogni volta invece che un utente accede al proprio account si esegue il metodo *signIn*: le credenziali inserite vengono controllate opportunamente dai metodi della classe *GestioneAutenticazione*.
Ogni volta che si effettua il login viene creata un'istanza della classe *GestioneAutenticazione* (come evidenziato dall'associazione) e ne si utilizzano i metodi per verificare il matching tra username e password, interfacciandosi con Auth0, dove sono memorizzati questi dati in maniera sicura. Entrambe le classi si interfacciano con il sistema subordinato Auth0.
L'ultima associazione dell'utente è con la classe *LogUtente*, in cui sono memorizzate tutte le segnalazioni non risolte dell'utente, associata a sua volta con la classe *Segnalazione*, per gestire i dati e le informazioni di una segnalazione.

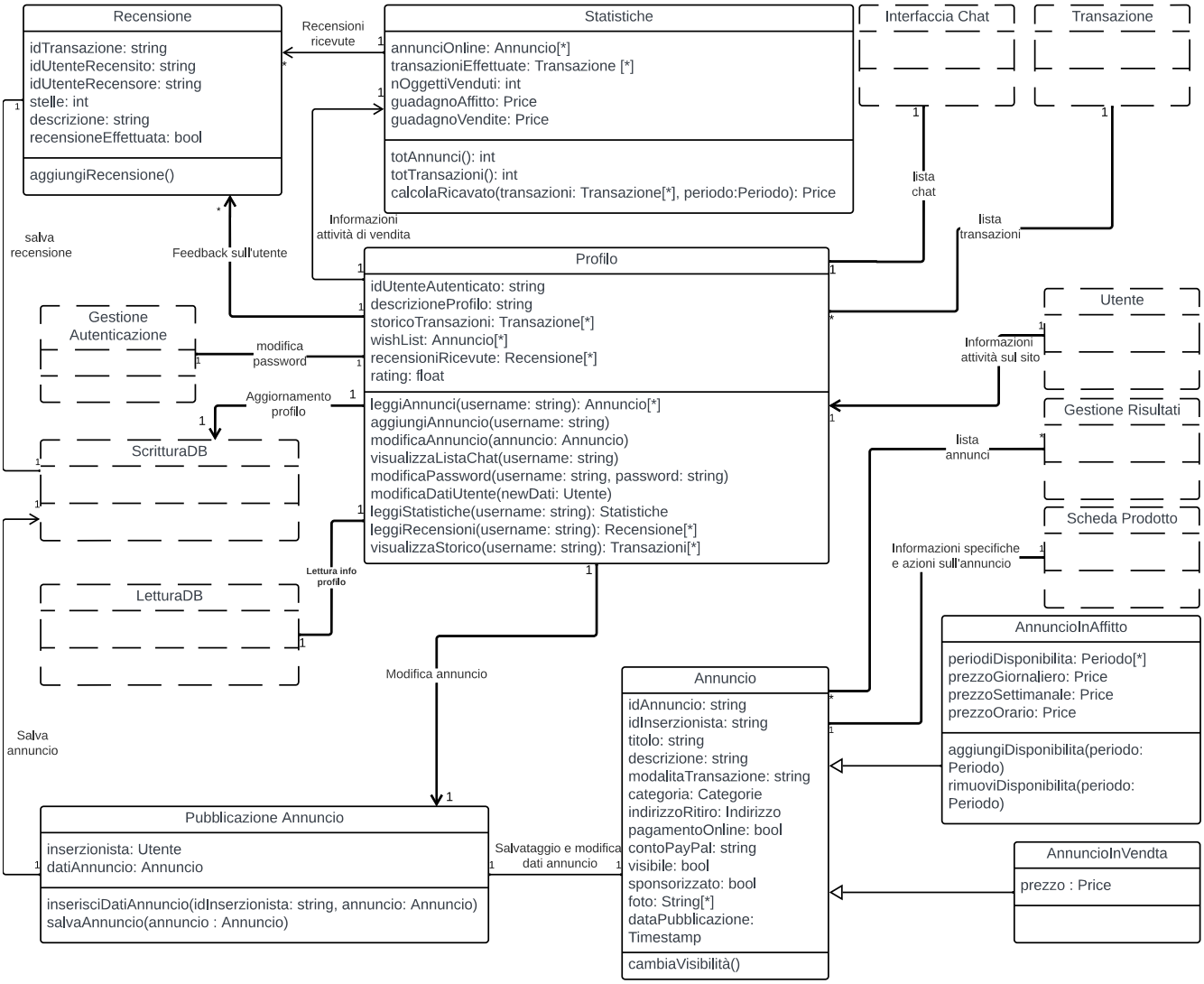
La classe *Amministratore* rappresenta invece l'utente amministratore del sistema insieme alle varie operazioni che può effettuare, in materia di intermediazione nelle dispute.


Segnalazione
idSegnalazione: string
idTransazione: string
motivazioneSegnalazione: string
utente: Utente
risolta: bool
dataSegnalazione: Data
Contiene
Controllo
Amministratore
nome: string
cognome: string
email: string
password: string
idAmministratore: string
accedi(idAmministratore: string, password: string): bool
avviaChat(idUtente: string)
bloccaProfilo(idUtente: String)
verificaSegnalazione(idSegnalazione: string)
modificaCategorie(categorie: Categorie)
LogUtente
utente: Utene
segnalazioniIrrisolte: Segnalazione[*]
ricercaSegnalazione(idSegnalazione): Segnalazione
Lista segnalazioni
Informazioni attività sul sito
Profilo
Autenticazione amministratore di sistema
Utente
idUtenteAutenticato: string
nome: string
cognome: string
data di nascita: Data
indirizzo: Indirizzo
email: string
username: string
password: string
profiloPersonale: Profilo
metodoDiPagamento: String
bloccato: bool
inviaSegnalazione(segnalazione: Segnalazione)
visualizzaProfilo()
signIn(username: string, password: string): string
signUp(potenzialeUtente: Utente): bool
logout()
ripristinaPassword(username: string)
Modifica Password
GestioneAutenticazione
usernameDaControllare : string
passwordDaControllare: string
twoFactorAuth: bool
verificaAccesso(usernameDaControllare: string, passwordDaControllare: string): string
modificaPassword(usernameDaModificare: string, oldPass: string, newPass: string): bool
ripristinaPassword(usernameDaModificare: string, newPass: string): bool
generaTokenAccesso(): string
Autenticazione
ScritturaDB
Salva utente
Registrazione
GestioneRegistrazione
possibileUtente: Utente
esitoRegistrazione: bool
verificaUnicitaUsername(usernameDaControllare: string): bool
isStrongPassword(passwordDaControllare: string): bool
confermaIdentità(possibileUtente : Utente) : bool
verificaEmail(emailDaControllare: string): bool
salvaCredenzialiAuth0(usernameControllato: string, passwordNuovoUtente: string)
salvaUtente(possibileUtente: Utente)
Verifica identità
GestioneSpid
utenteDaVerificare: Utente
validitaUtente: bool
verificaIdentita(utenteDaVerificare: Utente) : bool

Classi profilo utente e annuncio

La classe *Profilo* contiene tutti i dati e i metodi necessari all'utente per gestire le informazioni del proprio profilo sul sito e le proprie attività. È univocamente associata ad un singolo utente (infatti è presente una relazione con la classe *Utente*), oltre che con la classe *Statistiche* che serve per tenere traccia delle azioni di vendita intraprese dall'utente nel sistema (ad esempio questa classe al compito di computare i dati relativi allo storico delle vendite dell'utente).
È presente anche un'associazione con la classe *Recensione*, che gestisce i dati di una recensione relativa ad una transazione fatta all'utente, per rappresentare le recensioni lasciati dall'utente.
Entrambe le classi *Profilo* e *Recensione* presentano una associazione verso la classe *ScritturaDB* perché richiedono di effettuare delle operazioni di scrittura (in un caso salvataggio di una recensione, nell'altro modifica delle varie informazioni del profilo). Ogni volta che viene fatto un salvataggio da una di queste due classi si istanzia un oggetto della classe *ScritturaDB* e si chiama il metodo corrispondente per scrivere sul database.
Infine la classe *Profilo* è associata alla classe *LetturaDB* per leggere dal database le informazioni di un dato profilo, similmente a come si fa in fase di scrittura, *GestioneAutenticazione*, per gestire il cambiamento della password, e *InterfacciaChat* a cui richiede la lista di chat attive dall'utente.

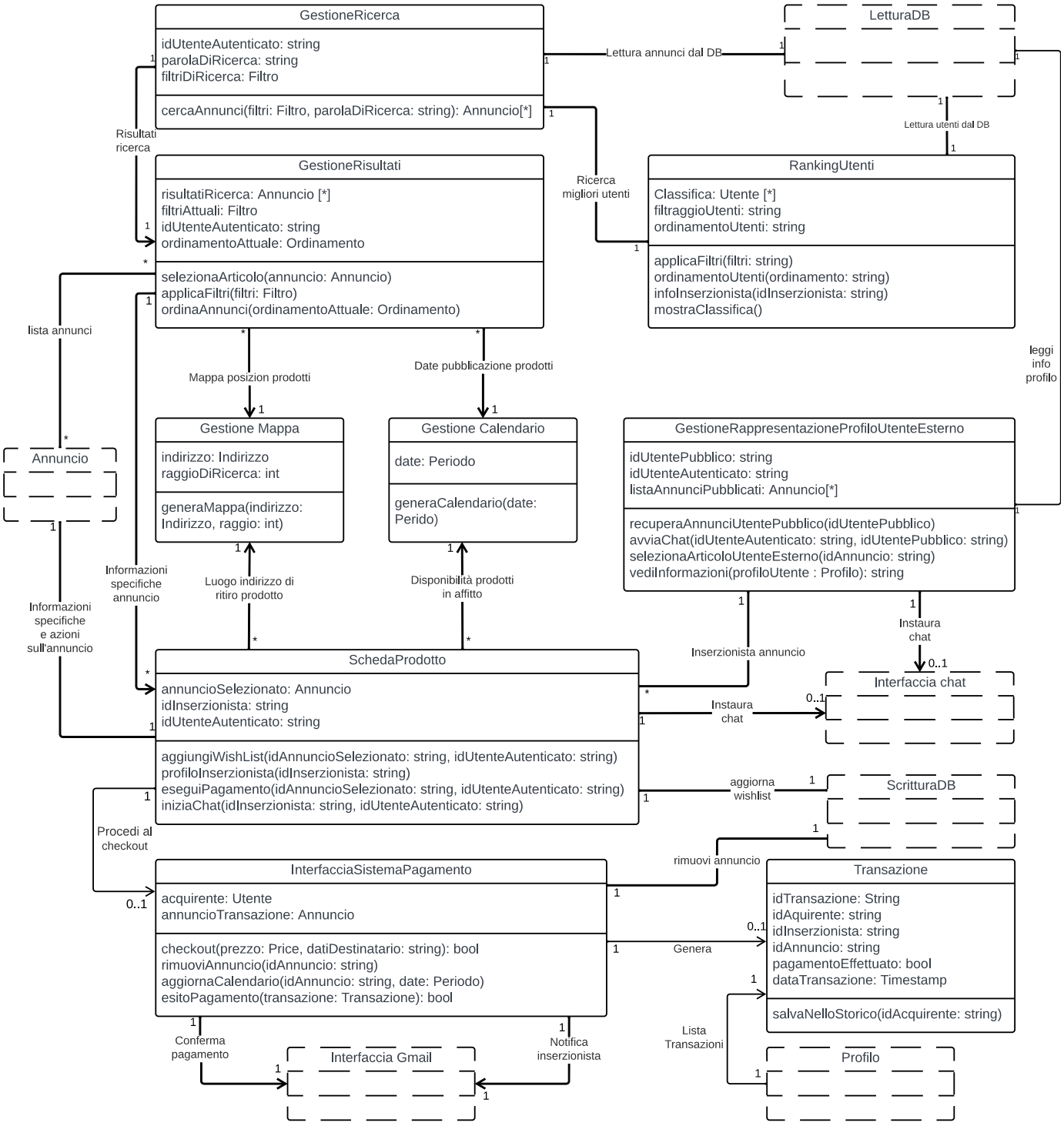
La classe *PubblicazioneAnnuncio* contiene i metodi utilizzati dall'utente durante la creazione di un nuovo annuncio, per questo motivo possiede una relazione con la classe *Annuncio,* che serve a gestire le informazioni di uno specifico annuncio caricato sul sito. Questa classe è astratta in quanto viene specializzata in *AnnuncioInAffitto* e *AnnuncioInVendita* a seconda della tipologia di transazione dell'annuncio.


Recensione
idTransazione: string
idUtenteRecensito: string
idUtenteRecensore: string
stelle: int
descrizione: string
recensioneEffettuata: bool
aggiungiRecensione()
Recensioni ricevute
Statistiche
annunciOnline: Annuncio[*]
transazioniEffettuate: Transazione [*]
nOggettiVenduti: int
guadagnoAffitto: Price
guadagnoVendite: Price
totAnnunci(): int
totTransazioni(): int
calcolaRicavato(transazioni: Transazione[*], periodo:Periodo): Price
Interfaccia Chat
Transazione
lista chat
lista transazioni
Informazioni attività di vendita
salva recensione
Feedback sull'utente
Profilo
idUtenteAutenticato: string
descrizioneProfilo: string
storicoTransazioni: Transazione[*]
wishList: Annuncio[*]
recensioniRicevute: Recensione[*]
rating: float
leggiAnnunci(username: string): Annuncio[*]
aggiungiAnnuncio(username: string)
modificaAnnuncio(annuncio: Annuncio)
visualizzaListaChat(username: string)
modificaPassword(username: string, password: string)
modificaDatiUtente(newDati: Utente)
leggiStatistiche(username: string): Statistiche
leggiRecensioni(username: string): Recensione[*]
visualizzaStorico(username: string): Transazioni[*]
Gestione Autenticazione
modifica password
Aggiornamento profilo
ScritturaDB
LetturaDB
Lettura info profilo
Utente
Informazioni attività sul sito
Gestione Risultati
lista annunci
Scheda Prodotto
Informazioni specifiche e azioni sull'annuncio
AnnuncioInAffitto
periodiDisponibilita: Periodo[*]
prezzoGiornaliero: Price
prezzoSettimanale: Price
prezzoOrario: Price
aggiungiDisponibilita(periodo: Periodo)
rimuoviDisponibilita(periodo: Periodo)
Modifica annuncio
Annuncio
idAnnuncio: string
idInserzionista: string
titolo: string
descrizione: string
modalitaTransazione: string
categoria: Categorie
indirizzoRitiro: Indirizzo
pagamentoOnline: bool
contoPayPal: string
visibile: bool
sponsorizzato: bool
foto: String[*]
dataPubblicazione: Timestamp
cambiaVisibilità()
Salva annuncio
Pubblicazione Annuncio
inserzionista: Utente
datiAnnuncio: Annuncio
inserisciDatiAnnuncio(idInserzionista: string, annuncio: Annuncio)
salvaAnnuncio(annuncio : Annuncio)
Salvataggio e modifica dati annuncio
AnnuncioInVendta
prezzo : Price

Classi di ricerca annunci, risultati di ricerca e pagamento

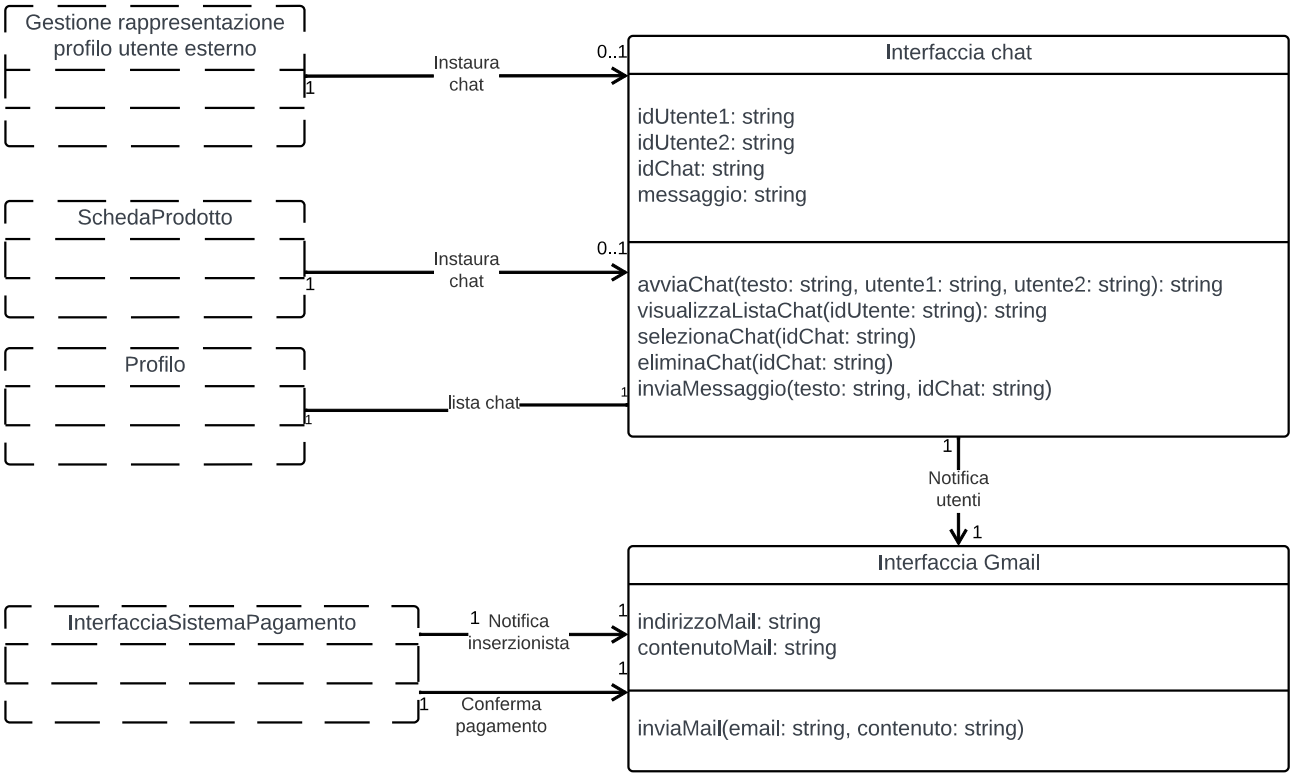
La classe *GestioneRicerca* gestisce la ricerca dei prodotti in base a particolari filtri e richiede l'accesso al database per leggere la lista di annunci corrispondenti agli specifici criteri di ricerca. Per ogni ricerca effettuata, si istanzia la classe *LetturaDB* che si occuperà di fare accesso direttamente al database e ricercare i corrispettivi annunci (come specificato dall'associazione).
L'altra associazione è verso la classe *GestioneRisultati*: gli annunci ottenuti infatti vengono forniti a questa classe che si occupa solamente di gestire le varie operazioni possibili su questa lista: ordinamento e filtraggio sui risultati. Da questa lista si può selezionare un preciso annuncio: in questo caso si crea un oggetto della classe *SchedaProdotto* che permette di gestire tutte le operazioni che si possono fare sul preciso annuncio selezionato e sull'inserzionista proprietario del prodotto.
È presente un'associazione con la classe *InterfacciaSistemaPagamento* che si occupa di interfacciarsi al sistema di PayPal usato per le transazioni di denaro in seguito ad un acquisto online sul sito. Questa classe in caso di esito positivo del pagamento si preoccupa anche di rimuovere l'annuncio acquistato: ecco perché c'è un'associazione con la classe *ScritturaDB* (per andare a usare i metodi di modifica sul database).
In questa parte di diagramma abbiamo anche la parte *Transazione* che rappresenta i dati e le informazioni relative ad un acquisto effettuato sul sito, che vengono salvate nel profilo dell'utente.
Abbiamo inoltre una classe *RankingUtenti* per gestire la classifica dei migliori venditori del sito: questa classe si occupa di cercare gli utenti con il ranking maggiore (istanziando opportunamente un oggetto della classe *LetturaDB* e usando i suoi metodi per leggere direttamente dal database) e gestire la lista degli utenti.
La classe *GestioneRappresentazioneProfiloUtenteEsterno* viene utilizzata per gestire l'interazione tra gli utenti, come ad esempio l'avvio di una chat. Ogni volta un utente richiede informazioni specifiche di un altro utente viene istanziato un oggetto di questa classe che recupera i dati del profilo e permette di avviare una conversazione o visualizzarne gli annunci.
Infine abbiamo le classi di interfacciamento con i sistemi esterni *GestioneMappa* e *GestioneCalendario*.

In tutte le classi è presente l'attributo *idUtenteAutenticato* che rappresenta l'attuale sessione d'accesso dell'utente. Se questo attributo è diverso da NULL significa che l'utente è loggato e che la sua sessione è ancora valida, e quindi possono essere eseguiti i metodi che richiedono che l'utente abbiamo effettuato l'accesso.


GestioneRicerca
idUtenteAutenticato: string
parolaDiRicerca: string
filtriDiRicerca: Filtro
cercaAnnunci(filtri: Filtro, parolaDiRicerca: string): Annuncio[*]
Lettura annunci dal DB
LetturaDB
Lettura utenti dal DB
Risultati ricerca
GestioneRisultati
risultatiRicerca: Annuncio [*]
filtriAttuali: Filtro
idUtenteAutenticato: string
ordinamentoAttuale: Ordinamento
selezionaArticolo(annuncio: Annuncio)
applicaFiltri(filtri: Filtro)
ordinaAnnunci(ordinamentoAttuale: Ordinamento)
Ricerca migliori utenti
RankingUtenti
Classifica: Utente [*]
filtraggioUtenti: string
ordinamentoUtenti: string
applicaFiltri(filtri: string)
ordinamentoUtenti(ordinamento: string)
infoInserzionista(idInserzionista: string)
mostraClassifica()
lista annunci
Mappa posizion prodotti
Date pubblicazione prodotti
leggi info profilo
Annuncio
Gestione Mappa
indirizzo: Indirizzo
raggioDiRicerca: int
generaMappa(indirizzo: Indirizzo, raggio: int)
Gestione Calendario
date: Periodo
generaCalendario(date: Perido)
GestioneRappresentazioneProfiloUtenteEsterno
idUtentePubblico: string
idUtenteAutenticato: string
listaAnnunciPubblicati: Annuncio[*]
recuperaAnnunciUtentePubblico(idUtentePubblico)
avviaChat(idUtenteAutenticato: string, idUtentePubblico: string)
selezionaArticoloUtenteEsterno(idAnnuncio: string)
vediInformazioni(profiloUtente : Profilo): string
Informazioni specifiche annuncio
Luogo indirizzo di ritiro prodotto
Disponibilità prodotti in affitto
Informazioni specifiche e azioni sull'annuncio
Inserzionista annuncio
Instaura chat
SchedaProdotto
annuncioSelezionato: Annuncio
idInserzionista: string
idUtenteAutenticato: string
aggiungiWishList(idAnnuncioSelezionato: string, idUtenteAutenticato: string)
profiloInserzionista(idInserzionista: string)
eseguiPagamento(idAnnuncioSelezionato: string, idUtenteAutenticato: string)
iniziaChat(idInserzionista: string, idUtenteAutenticato: string)
Interfaccia chat
Instaura chat
aggiorna wishlist
ScritturaDB
Procedi al checkout
rimuovi annuncio
InterfacciaSistemaPagamento
acquirente: Utente
annuncioTransazione: Annuncio
checkout(prezzo: Price, datiDestinatario: string): bool
rimuoviAnnuncio(idAnnuncio: string)
aggiornaCalendario(idAnnuncio: string, date: Periodo)
esitoPagamento(transazione: Transazione): bool
Transazione
idTransazione: String
idAquirente: string
idInserzionista: string
idAnnuncio: string
pagamentoEffettuato: bool
dataTransazione: Timestamp
salvaNelloStorico(idAcquirente: string)
Genera
Conferma pagamento
Notifica inserzionista
Interfaccia Gmail
Lista Transazioni
Profilo

Classi di interfacciamento verso i sistemi esterni di chat e mail

La classe *InterfacciaChat* ha a che fare con il sistema subordinato che fornisce una chat tra due utenti ed è associata con le varie classi in cui è necessario istituire un collegamento diretto tramite chat tra due utenti ossia *GestioneRappresentazioneProfiloUtenteEsterno* e *SchedaProdotto.* Dalla classe *Profilo* c'è un'ulteriore associazione per chiedere la lista delle chat attive di un particolare utente.
Ogni volta si mostra necessaria avviare una chat tra due utenti, viene istanziato un oggetto di questa classe che con i suoi metodi comunica con il sistema esterno inizializzando e fornendo la chat.
Infine la classe *InterfacciaGmail* si occupa di interfacciarsi col sistema esterno *GmailAPI.* Ogni qual volta una classe necessita di inviare una mail all'utente istanzia un oggetto di questa classe fornendo indirizzo mail e contenuto della mail.
Alcune di queste associazioni sono con *InterfacciaChat* (per notificare l'avvio di una nuova chat) e con *InterfacciaSistemaPagamento* (per notificare acquirente e inserzionista del corretto esito di una transazione).



Classi di interfacciamento al database

Le classi *ScritturaDB* e *LetturaDB* sono due classi senza attributi che hanno la collezione di tutti i metodi usati dalla nostra applicazione per interfacciarsi al database: rispettivamente la prima classe serve per effettuare operazioni di scrittura, modifica e cancellazione, mentre la seconda classe operazioni di lettura.
Questa scelta è dovuta a motivi di modularità del codice: nel caso dovesse cambiare il database esterno con cui ci si interfaccia, non è necessario modificare i vari metodi in tutte le classi, ma solo i metodi in queste due classi.
Quando da una classe generica si effettua un'operazione di salvataggio dati o di ricerca, ciò che viene fatto è istanziare un oggetto di una di queste due classi e usare i metodi della classe corrispondente, coi parametri opportunamente passati dalla classe principale "chiamante".
Come detto questa scelta permette di fornire maggiore indipendenza al sistema rispetto ai sistemi esterni con cui ci si interfaccia.

Le classi che sono associate con la classe *ScritturaDB* sono:

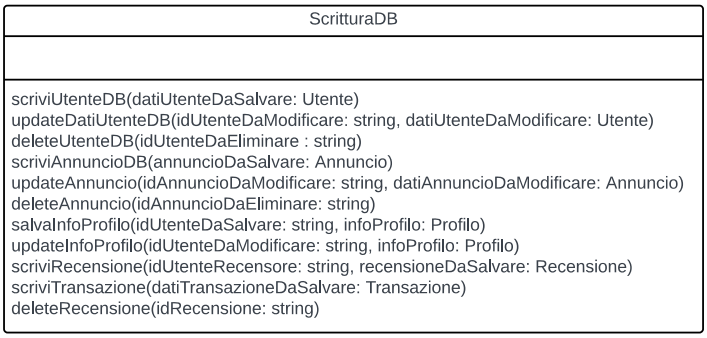
- *Profilo*
- *Recensione*
- *PubblicazioneAnnuncio*
- *GestioneRegistrazione*
- *SchedaProdotto*
- *InterfacciaSistemaPagamento*

e riguardano tutte le classi che effettuano operazioni di scrittura, modifica o cancellazione nel database.

Le classi che sono associate con la classe *LetturaDB* sono:

- *Profilo*
- *RankingUtenti*
- *GestioneRicerca*
- *GestioneRappresentazioneProfiloUtenteEsterno*

e riguardano tutte le classi che effettuano operazioni lettura dati nel database.



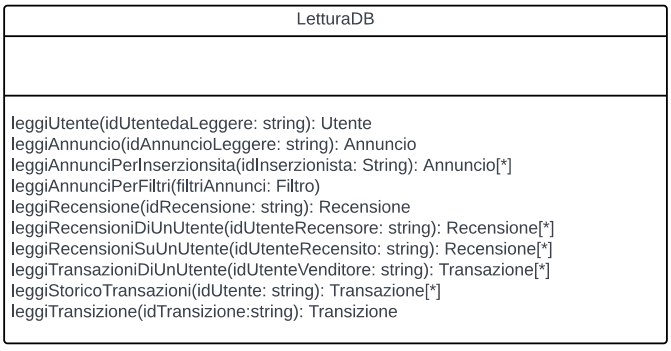
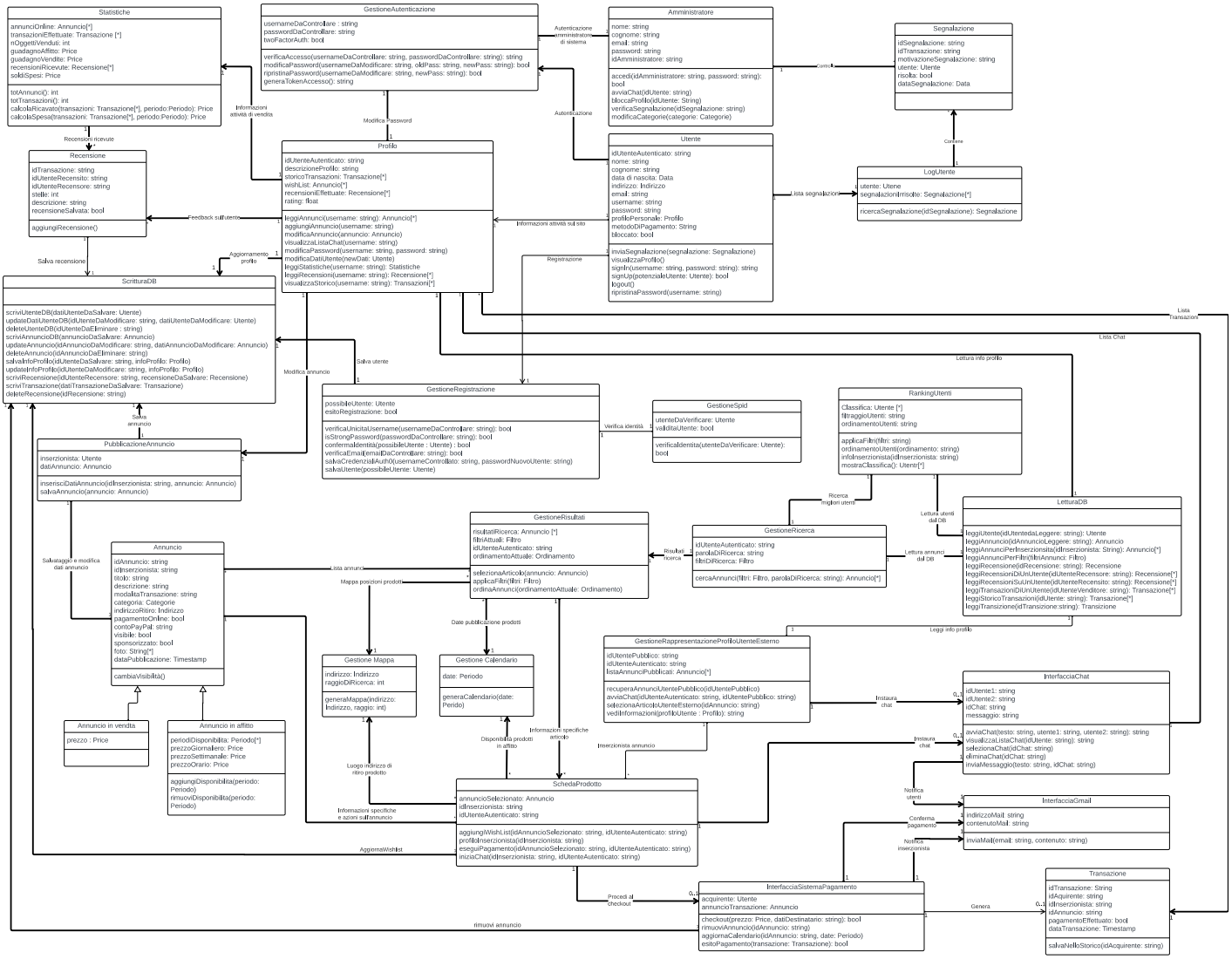
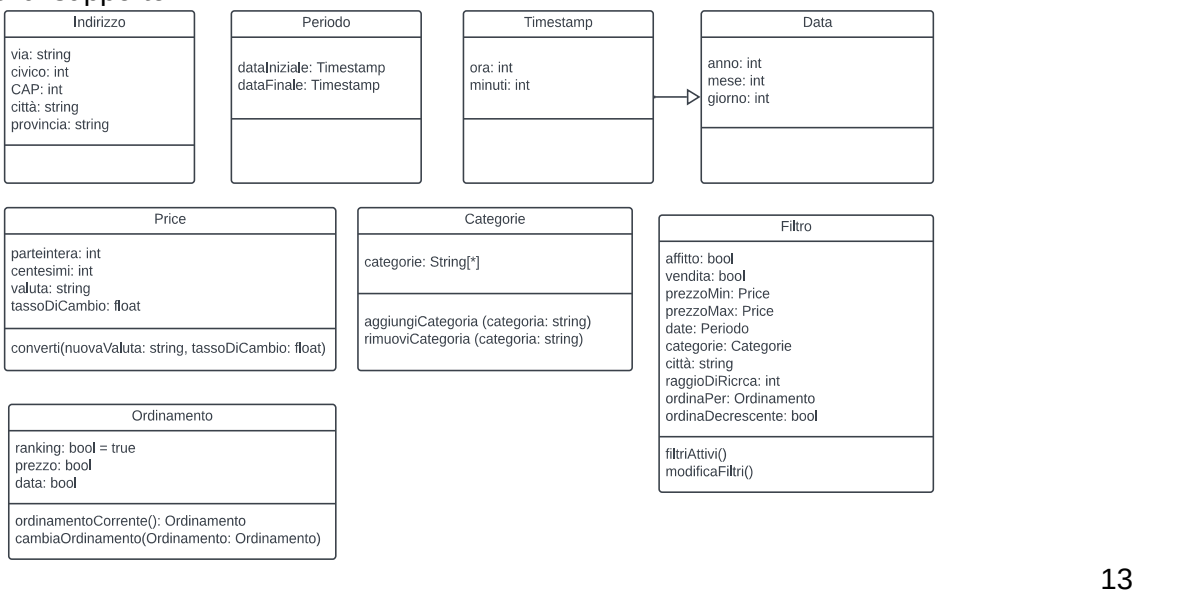


Diagramma delle classi completo

Di seguito si offre la visualizzazione del diagramma delle classi del nostro progetto, nella sua interessa, evidenziando le associazioni tra le varie classe precedentemente descritte e rappresentate.



Con le classi di supporto:



Codice in Object Constraint Language

In questo capitolo descriviamo in modo formale i vincoli di logica necessari in alcune classi.
Questi vincoli riguardano invarianti (ovvero vincoli sugli attributi) e pre/post condizioni (ovvero vincoli sull'esecuzione delle operazioni).
Per evitare ambiguità, vengono espressi con il linguaggio formale OCL (Object Constraint Language).

Vincoli OCL classe *GestioneRegistrazione*

Prima di salvare un utente, è necessario che i vari step della registrazione siano stati completati con successo.

context *GestioneRegistrazione::salvaUtente(datiUtente: Utente)*
pre*: verificaEmail(email: string) = true*
AND isStrongPassword(passwordDaControllare: string) = true
AND verificaUnicitaUsername(username: string) = true
AND confermaIdentita(possibileUtente: Utente) = true
post*: esitoRegistrazione = true*

Vincoli OCL classe GestioneAutenticazione

Prima di modificare una password, è necessario fornire correttamente la vecchia password e fornire una nuova password concorde con i requisiti di sicurezza.

context *GestioneAutenticazione::modificaPassword(usernameDaModificare: string, oldpass: string, newPass: string)*
pre*: oldpass = passwordDaControllare AND isStrongPassword(newPass: string) = true*

Vincoli OCL classe *PubblicazioneAnnuncio*

Questo vincolo specifica gli attributi di un annuncio che devono essere assolutamente specificati prima di salvare un annuncio nel database.

context *PubblicazioneAnnuncio::salvaAnnuncio(annuncio: Annuncio)*
pre*: titolo != NULL*
descrizione != NULL
modalitaTransizione != NULL
categoria != NULL
indirizzoRitiro != NULL
post*: annuncio.visibile = true*

Vincoli OCL classe *GestioneRappresentazioneProfiloUtenteEsterno*

Questo vincolo specifica che prima di avviare una chat tra due utenti è obbligatorio specificare gli username dei due utenti.

context *GestioneRappresentazioneProfiloUtenteEsterno::avviaChat(idUtenteAutenticato: string, idUtentePubblico: string)*
pre*: idUtenteAutenticato != NULL AND idUtentePubblico != NULL*

Vincoli OCL classe *Utente*

Questi vincoli specificato che per effettuare il logout, inviare una segnalazione e visualizzare il proprio profilo personale è necessario aver effettuato l'accesso al sistema (*idUtenteAutenticato* deve essere non nullo), mentre *idUtenteAutenticato* deve essere nullo (cioè è necessario non aver fatto l'accesso al sistema) prima di fare la registrazione e il login.

***context** Utente::logout()*
***pre**: idUtenteAutenticato != NULL*

***context** Utente::signIn(Username: string, password: string): bool*
***pre**: idUtenteAutenticato = NULL AND bloccato = false*
***post**: idUtenteAutenticato != NULL*

***context** Utente::signUp(potenzialeUtente: Utente)*
***pre**: idUtenteAutenticato = NULL*

***context** Utente::inviaSegnalazione(segnalazione: Segnalazione)*
***pre**: idUtenteAutenticato != NULL*
***post**: LogUtente.segnalazioniIrrisolte→append(segnalazione)*

***context** Utente::visualizzaProfilo()*
***pre**: idUtenteAutenticato != NULL*

Vincoli OCL classe *LogUtente*

Questo vincolo specifica che tutte le segnalazioni di un utente salvate nel log, devono essere ancora irrisolte.

***context** LogUtente inv: segnalazioniIrrisolte->forAll(Segnalazione.risolta = false)*

Vincoli OCL classe *Profilo*

Questo vincolo specifica che si può modificare un annuncio solo del proprio profilo.

***context** Profilo::modificaAnnuncio(annuncio: Annuncio)*
***pre**: annuncio.usernameInserzionista = Utente.username*

***context** Profilo **inv**: rating>=0 AND rating <= 5*

Vincoli OCL classe *Statistiche*

Ogni volta che si calcola il ricavato dalle transazioni effettuate da un utente, è chiaramente necessario che ci sia almeno una transazione effettuata.

***context** Statistiche::calcolaRicavato(transazioni: Transazione[*], periodo: Periodo)*
***pre**: transazioni->isEmpty() = false*
***post:** guadagnoAffitti >= 0 AND guadagnoVendite >= 0*

***context** Statistiche::calcolaSpese(transazioni: Transazione[*], periodo: Periodo)*
***pre**: transazioni->isEmpty() = false*

***post**: soldiSpesi >= 0*

Vincoli OCL classe *Amministratore*

L'amministratore prima di eseguire tutti i suoi metodi deve aver effettuato l'accesso al sistema, con le sue credenziali da amministratore.

***context** Amministratore::avviaChat(idUtente: string)*
***pre**: accedi(idAmministratore: string, password: string) = true AND idUtente != NULL*

***context** Amministratore::verificaSegnalazione(idSegnalazione: string)*
***pre**: accedi(idAmministratore: string, password: string) = true*

***context** Amministratore::modificaCategorie(categorie: Categorie)*
***pre**: accedi(idAmministratore: string, password: string) = true*

***context** Amministratore::bloccaProfilo(idUtente: string)*
***pre**: accedi(idAmministratore: string, password: string) = true*
***post**: Utente.bloccato = true*

Vincoli OCL classe *Transazione*

Una transazione può essere salvata nello storico degli ordini di un utente solo dopo che è avvenuto il pagamento per il prodotto.

***context** Transazione::salvaNelloStorico(idAcquirente: string)*
***pre**: pagamentoEffettuato = true*

Vincoli OCL classe *InterfacciaSistemaPagamento*

Solo dopo che è stato effettuato il pagamento per un annuncio in vendita, è possibile procedere alla rimozione di tale annuncio dal sito.

***context** InterfacciaSistemaPagamento::rimuoviAnnuncio(idAnnuncio: string)*
***pre**: transazione.pagamentoEffettuato = true*

Vincoli OCL classe *Recensione*

Questi vincoli specificano invarianti e valori iniziali per alcuni attributi tipici di una recensione, oltre che una condizione necessaria all'esecuzione di un metodo.

***context** Recensione:: recensioneEffettuata: bool **init**: false*

***context** Recensione::aggiungiRecensione()*
***pre**: recensioneEffettuata = false*
***post**: recensioneEffettuata = true*

***context** Recensione **inv**: stelle >= 0 AND stelle <=5*

***context** Recensione **inv**: descrizione->length() <= 200*

Vincoli OCL classe *Annuncio*
Questo vincolo specifica delle invarianti sui vari attributi di un annuncio.

context *Annuncio*
inv*: descrizione->length() <= 350*
inv*: foto->size() <= 10*
inv*: Categorie.categorie->includes(categoria) = true*
inv*: if pagamentoOnline = true then contoPayPal != NULL*

Vincoli OCL classe *SchedaProdotto*
Quando si visualizza un particolare prodotto è possibile aggiungerlo alla wish list, comprarlo pagando online o iniziare una chat con l'inserzionista solo se si ha effettuato il login nel sistema.

context *SchedaProdotto::aggiungiWishList(idAnnuncioSelezionato: string, idUtenteAutenticato: string)*
pre*: idUtenteAutenticato != NULL*
post*: Utente.Profilo.wishList->append(Annuncio)*

context *SchedaProdotto::eseguiPagamento(idAnnuncioSelezionato: string, idUtenteAutenticato: string)*
pre*: idUtenteAutenticato != NULL AND Annuncio.pagamentoOnline = true*

context *SchedaProdotto::iniziaChat(idInserzionista: string, idUtenteAutenticato: string)*
pre: idUtenteAutenticato != NULL

Vincoli OCL classe *Filtro*
Un invariante che specifica i limiti del possibile raggio di ricerca con cui si può filtrare un prodotto.

context *Mappa inv: raggioDiRicerca > 0 AND raggioDiRicerca < 40000*

Vincoli OCL classe *Data*
Questo vincolo specifica le invarianti logiche sugli attributi di una data.

context *Data*
inv*: anno >= 1900 AND anno <= 2300*
inv*: mese >= 1 AND mese <= 12*
inv*: giorno >= 1 AND giorno <= 31*

Vincoli OCL classe *Timestamp*
Questo vincolo specifica le invarianti logiche sugli attributi che caratterizzano un'ora del giorno.

context *Timestamp*
inv*: ora >= 1 AND ora <= 24*
inv*: minuti >= 0 AND minuti <= 59*

Vincoli OCL classe *Periodo*

Questo vincolo specifica una condizione sempre valida sui due attributi di questa classe.

context *Periodo* ***inv****: dataFinale > dataIniziale*

Vincoli OCL classe *Price*

Questo vincolo specifica condizioni logiche sul valore degli attributi della classe *Price*.

context *Price*
inv*: parteintera >= 0*
inv*: partedecimale >= 0 AND partedecimale <= 99*
inv*: tassoDiCambio >= 0*

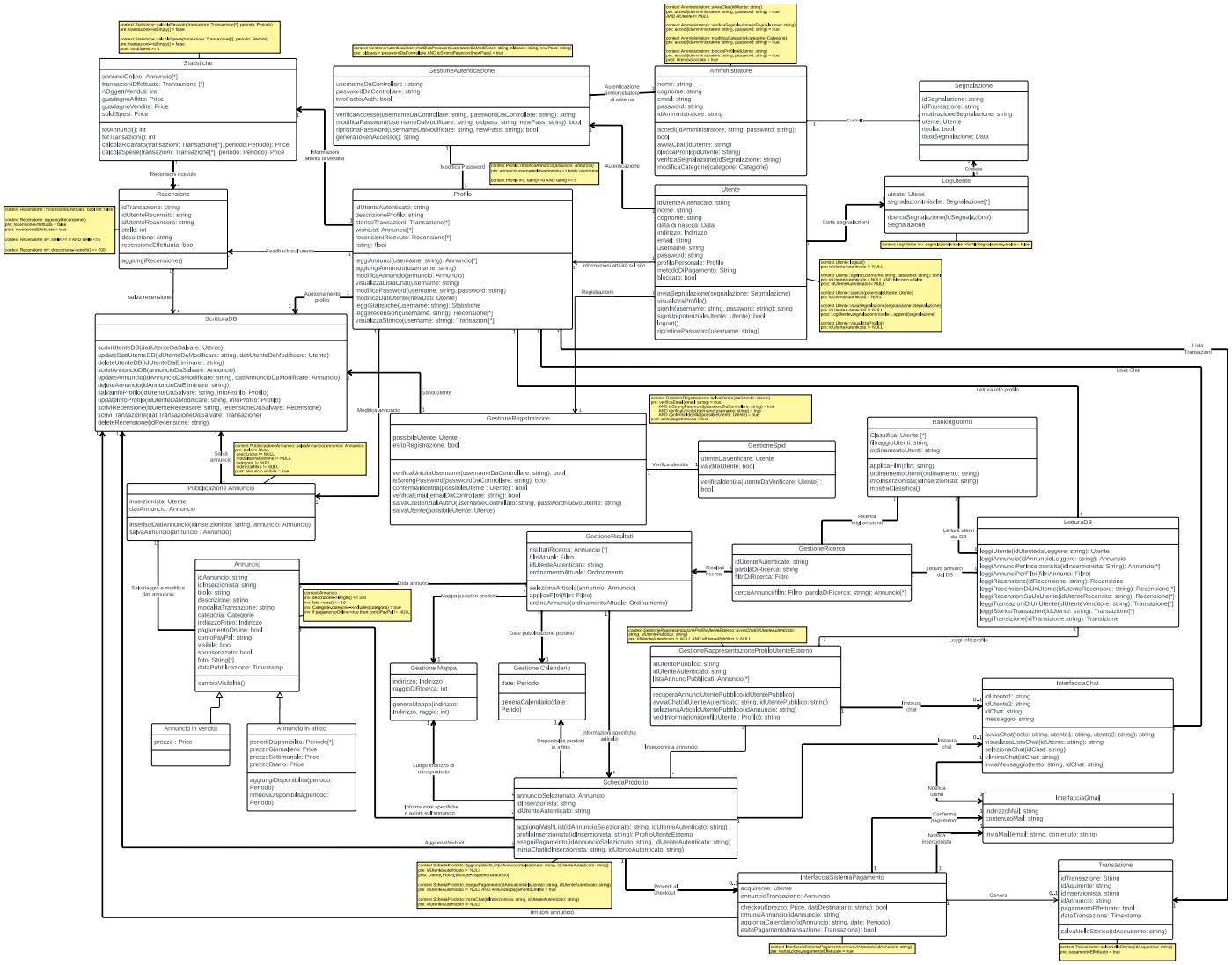
Vincoli OCL classe *Ordinamento*

Questo vincolo specifica che se un qualsiasi attributo della classe *Ordinamento* è *true*, gli altri devono essere *false*.

context *Ordinamento*
inv*: if ranking = true then data = false AND prezzo = false*
inv*: if prezzo = true then ranking = false AND data = false*
inv*: if data = true then ranking = false AND prezzo = false*

Diagramma delle classi con codice in OCL associato

Per conclude mostriamo il nostro diagramma delle classi evidenziando il rispettivo codice OCL per ognuna di esse.



Con le classi di supporto: