Progetto:

iSpesa

Titolo del documento:

Documento di architettura

Informazioni Documento

Nome Documento	*iSpesa_Architettura_D3*	Numero Documento	D3
Descrizione	Il documento include diagrammi delle classi e codice in OCL		

Indice

Scopo del documento

Il presente documento riporta la definizione dell'architettura del progetto iSpesa usando diagrammi delle classi in Unified Modeling Language (UML) e codice in Object Constraint Language (OCL). Nel precedente documento è stato presentato il diagramma degli use case, il diagramma di contesto e quello dei componenti. Ora, tenendo conto di questa progettazione, viene definita l'architettura del sistema dettagliando da un lato le classi che dovranno essere implementate a livello di codice e dall'altro la logica che regola il comportamento del software. Le classi vengono rappresentate tramite un diagramma delle classi in linguaggio UML. La logica viene descritta in OCL perché tali concetti non sono esprimibili in nessun altro modo formale nel contesto di UML.

1. Diagramma delle classi

Nel presente capitolo vengono presentate le classi previste nell'ambito del progetto iSpesa. Ogni componente presente nel diagramma dei componenti diventa una o più classi. Tutte le classi individuate sono caratterizzate da un nome, una lista di attributi che identificano i dati gestiti dalla classe e una lista di metodi che definiscono le operazioni previste all'interno della classe. Ogni classe può essere anche associata ad altre classi e, tramite questa associazione, è possibile fornire informazioni su come le classi si relazionano tra loro.
Riportiamo di seguito le classi individuate a partire dai diagrammi di contesto e dei componenti. In questo processo si è proceduto anche nel massimizzare la coesione e minimizzare l'accoppiamento tra classi.

1.1 Autenticazione e Registrazione

La classe **Utente** è necessaria per salvare tutte i dati relativi ad un preciso utente registrato nel nostro sistema. Questa classe prevede un metodo *registrazione*, per effettuare la registrazione, un metodo *login*, per effettuare il login, ed un metodo di logout. Ogni volta che un utente si registra si esegue il metodo *registrazione* e viene creata un'istanza della classe ***GestoreRegistrazione***, in cui si verifica la validità dei dati inseriti dall'utente con opportuni metodi (*verificaUsername*, *isPasswordSicura*, *verificaEmail*) e, in caso di esito positivo, si procede al salvataggio dell'utente nel database (effettuato dalla classe **ScritturaDBUtenti**, chiamata dal metodo *salvaUtente*) e nei database di Auth0 (chiamato dal metodo *inviaCredenzialiAuth0*).
Ogni volta invece che un utente accede al proprio account si esegue il metodo login e viene creata un'istanza della classe **GestoreAutenticazione**, che verifica che le credenziali siano corrette interfacciandosi con il sistema Auth0 (mediante il metodo *controllaAccesso*). Questa classe si occupa anche di modificare la password utente (mediante il metodo *modificaPassword*) e di interfacciarsi col sistema di ripristino della password del sistema Auth0 (mediante il metodo *ripristinaPassword*). Inoltra si occupa anche di bloccare e sbloccare l'account a seguito di vari accessi consecutivi errati (con i metodi *bloccaUtente* e *sbloccaUtente*).
GestoreAutenticazione si occupa anche della 2FA comunicando con il sistema Auth0. Sia la classe **GestoreAutenticazione** e **GestoreRegistrazione** si interfacciano con l'**InterfacciaGmail** per comunicare con l'utente mediante mail. La classe **GestoreRegistrazione** si interfaccia, inoltre, con la classe **ScritturaDBUtenti** che si occupa di salvare sul Database degli utenti i dati dell'utente. I dati dell'utente, invece, vengono ottenuti dopo il login mediante la classe **LetturaDBUtenti**.
La classe **Amministratore** rappresenta tutti gli amministratori del sistema. Essi possono soltanto fare login e logout (mediante i metodi *login* e *logout*) e non possono registrarsi né ottenere/dare altri dati.

Di seguito un'immagine rappresentante le classi coinvolte:

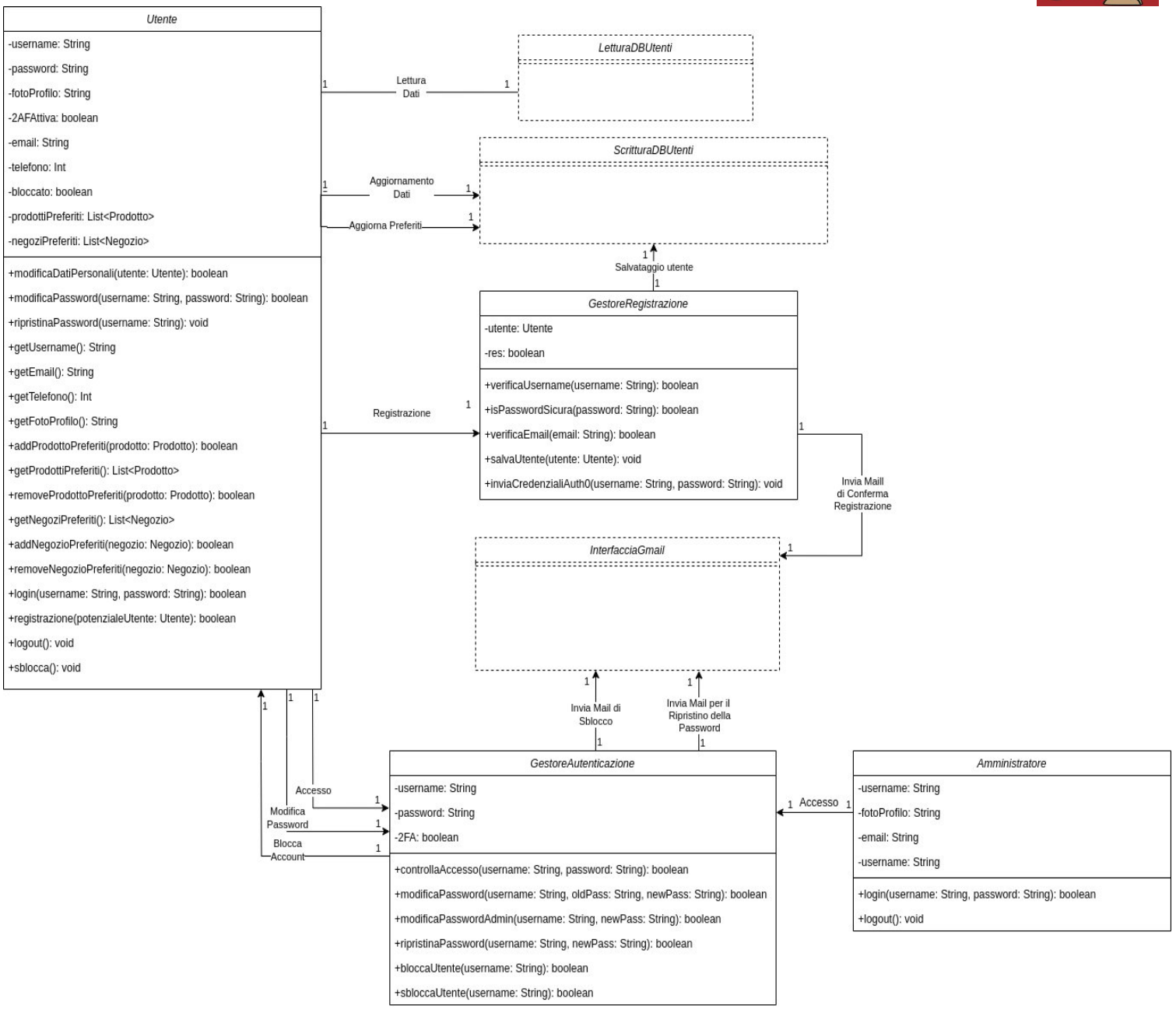


Figura 1. Diagramma delle classi, con relative associazioni, coinvolte nell'Autenticazione/Registrazione.

1.2 Pagina Utente

Sempre parlando della classe **Utente**, essa è al centro anche della visualizzazione della pagina utente. Infatti da essa si possono ottenere tutti i dati da visualizzare mediante i suoi metodi (in particolare mediante *getUsername*, *getEmail*, *getTelefono*, *getFotoProfilo*). Mediante il metodo *modificaDatiPersonali* è possibile modificare gli stessi. La classe **Utente** è associata all'**interfacciaSegnalazioni**, poiché l'utente, dalla sua pagina dedicata, può inviare delle segnalazioni alla moderazione del sito che verrà inviate (attraverso l'**InterfacciaGmail**) all'apposita email dedicata. Poiché l'utente può modificare la propria password (mediante il metodo *modificaPassword*), la classe Utente è nuovamente associata alla classe **GestoreAutenticazione** (già descritta nel punto 1.1) di cui verrà creata un'istanza invocando il metodo *modificaPassword*. Anche i dati relativi ai prodotti e/o negozi preferiti di un utente sono conservati in questa classe e possono essere ottenuti e/o modificati mediante i metodi *addProdottoPreferiti*, *getProdottiPreferiti*, *removeProdottoPreferiti*, *getNegoziPreferiti*, *addNegozioPreferiti* e *removeNegozioPreferiti*. Come già detto in precedenza i dati della classe **Utente** verranno ottenuti la classe **LetturaDBUtenti**, mentre le modifiche ai dati di un utente verranno salvate mediante la classe **ScritturaDBUtenti**.

Di seguito un'immagine rappresentante le classi coinvolte:

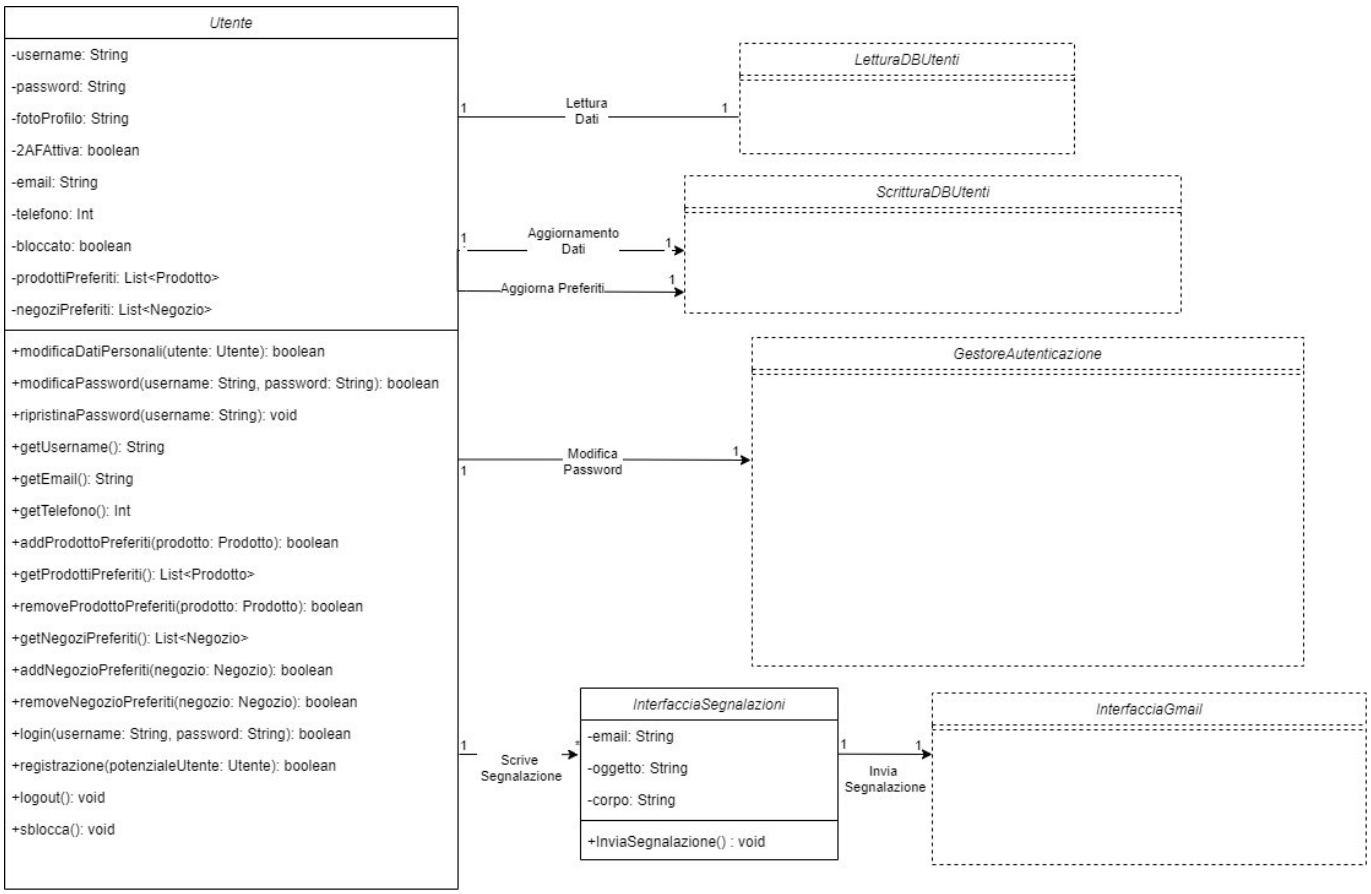


Figura 2. Diagramma delle classi, con relative associazioni, coinvolte nella Pagina Utente.

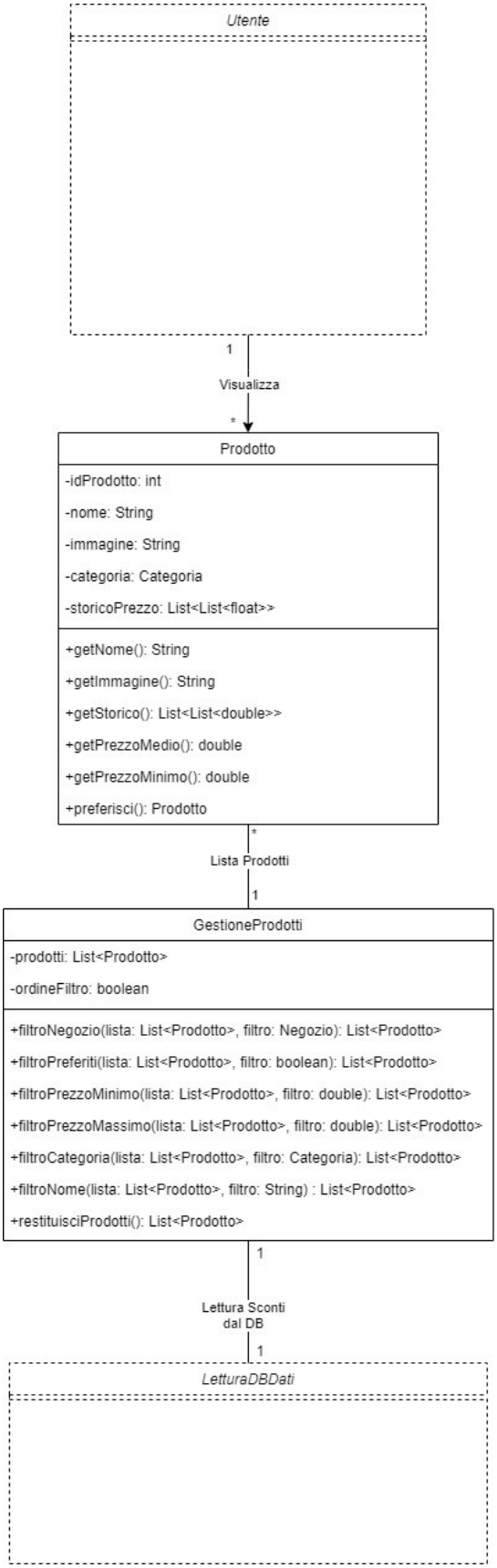


Figura 3. Diagramma delle classi, con relative associazioni, coinvolte nella Visualizzazione dei Prodotti.

1.3 Visualizzazione dei Prodotti

Il diagramma a sinistra illustra le classi coinvolte nella visualizzazione dei prodotti e le loro associazioni. Come si può notare dallo schema, la classe **Utente** è associata alla classe **Prodotto** poiché un utente può visualizzare svariati prodotti all'interno dell'apposita pagina. Poiché l'utente può anche visualizzare maggiori dettagli per ogni prodotto, la classe **Prodotto** contiene svariate informazioni e metodi. Ogni prodotto contiene al suo interno tutte le informazioni relative a nomi, immagini e categorie, non che lo storico dei prezzi dei singoli prodotti. Tutti questi parametri sono ottenibili mediante i loro appositi metodi (*getNome*, *getImmagine*, *getStorico*). Sono presenti anche due metodi aggiuntivi, *getPrezzoMinimo* e *getPrezzoMedio*. Come si può intuire dai noi essi servono ad ottenere rispettivamente il prezzo più basso e la media del prezzo. Si può notare anche la presenza di un metodo preferisci che serve a salvare un prodotto come preferito. L'ottenimento dei prodotti dal database e il loro filtro sono stati affidati alla classe **GestioneProdotti** di cui viene creata un'istanza quando l'utente apre la pagina prodotti. Tale classe filtra gli oggetti da visualizzare mediante i propri metodi (*filtroNegozio*, *filtroPreferiti*, *filtroPrezzoMinimo*, *filtroPrezzoMassimo*, *filtroCategoria*, *filtroNome*), che modificano ciò che la classe visualizza nell'apposita pagina. È presente un parametro *ordineFiltro* che stabilisce se l'ordinamento già intrinseco ai filtri deve essere crescente o decrescente. Sempre osservando il grafico si può notare come la classe **GestioneProdotti** ottenga tutti i dati necessari dalla classe **LetturaDBDati**.

1.3 Visualizzazione Negozi

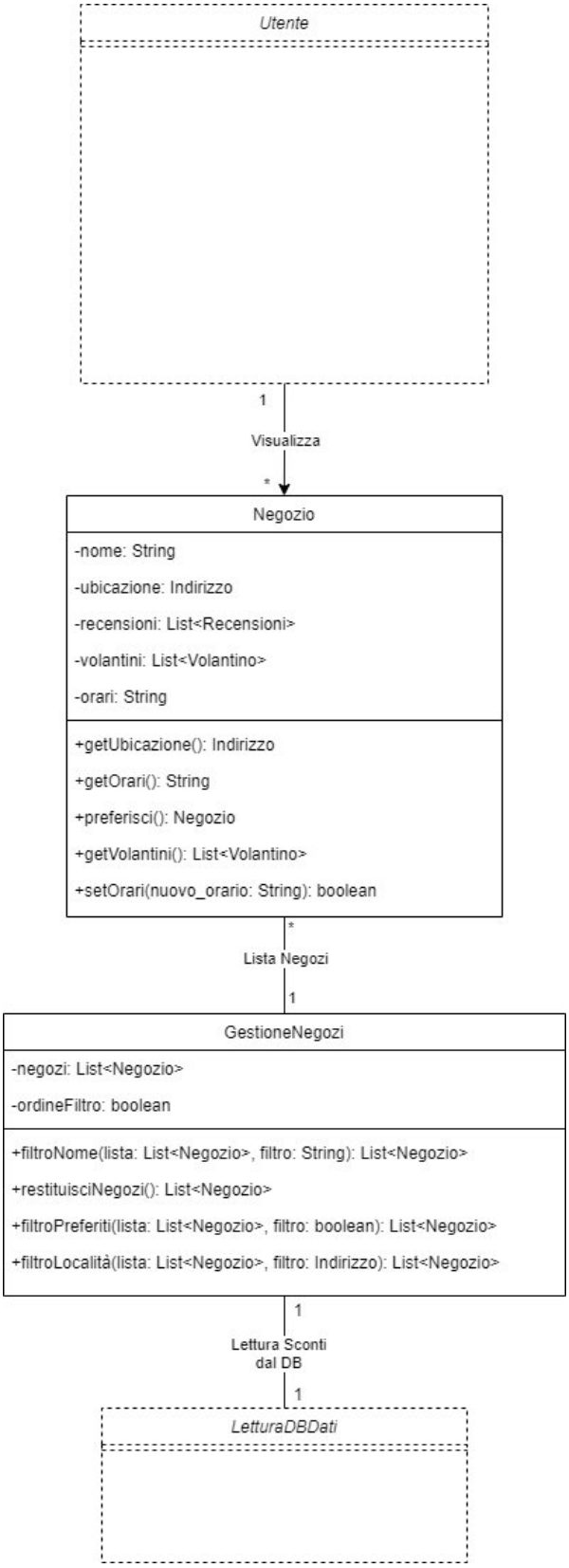


Figura 4. Diagramma delle classi, con relative associazioni, coinvolte nella Visualizzazione dei Negozi.

Il diagramma a sinistra illustra le classi coinvolte nella visualizzazione dei negozi e le loro associazioni. Come si può notare dallo schema, la classe **Utente** è associata alla classe **Negozio** poiché un utente può visualizzare svariati negozi all'interno dell'apposita pagina. Poiché l'utente può anche visualizzare maggiori dettagli per ogni negozio, la classe **Negozio** contiene svariate informazioni e metodi. Ogni negozio contiene al suo interno tutte le informazioni relative al proprio nome, l'ubicazione del negozio, l'orario di apertura e chiusura, la lista delle recensioni e la lista dei volantini associati a tale negozio. Tutti questi parametri sono ottenibili mediante i loro appositi metodi (*getUbicazione*, *getOrari*, *getVolantini*). È presente anche un metodo aggiuntivo, *setOrari*, che permette di modificare l'orario del negozio. Si può notare anche la presenza di un metodo preferisci che serve a salvare un negozio come preferito. L'ottenimento dei negozi dal database e il loro filtro sono stati affidati alla classe **GestioneNegozi** di cui viene creata un'istanza quando l'utente apre la pagina negozi. Tale classe filtra gli oggetti da visualizzare mediante i propri metodi (*filtroNome*, *filtroPreferiti*, *filtroLocalità*), che modificano ciò che la classe visualizza nell'apposita pagina. È presente un parametro *ordineFiltro* che stabilisce se l'ordinamento già intrinseco ai filtri deve essere crescente o decrescente. Sempre osservando il grafico si può notare come la classe **GestioneNegozi** ottenga tutti i dati necessari dalla classe **LetturaDBDati**.

1.4 Visualizzazione Volantini

Il diagramma a sinistra illustra le classi coinvolte nella visualizzazione dei volantini e le loro associazioni. Come si può notare dallo schema, la classe **Utente** è associata alla classe **Volantino** poiché un utente può visualizzare svariati volantini all'interno dell'apposita pagina. Ogni volantino contiene un id univico, i dati relativi al negozio a cui si riferisce, la data di fine delle offerte mostrate e una stringa contenente un url al volantino che l'utente può visualizzare, tutti ottenibili mediante i metodi *getNegozio*, *getFile*, *getDataFine*, *getIdVolantino*. L'ottenimento dei volantini dal database e il loro filtro sono stati affidati alla classe **GestioneVolantini** di cui viene creata un'istanza quando l'utente apre la pagina volantini. Tale classe filtra gli oggetti da visualizzare mediante i propri metodi (*filtroNegozio*, *filtroPreferiti*), che modificano ciò che la classe visualizza nell'apposita pagina. È presente un parametro *ordineFiltro* che stabilisce se l'ordinamento già intrinseco ai filtri deve essere crescente o decrescente. Sempre osservando il grafico si può notare come la classe **GestioneVolantini** ottenga tutti i dati necessari dalla classe **LetturaDBDati**.

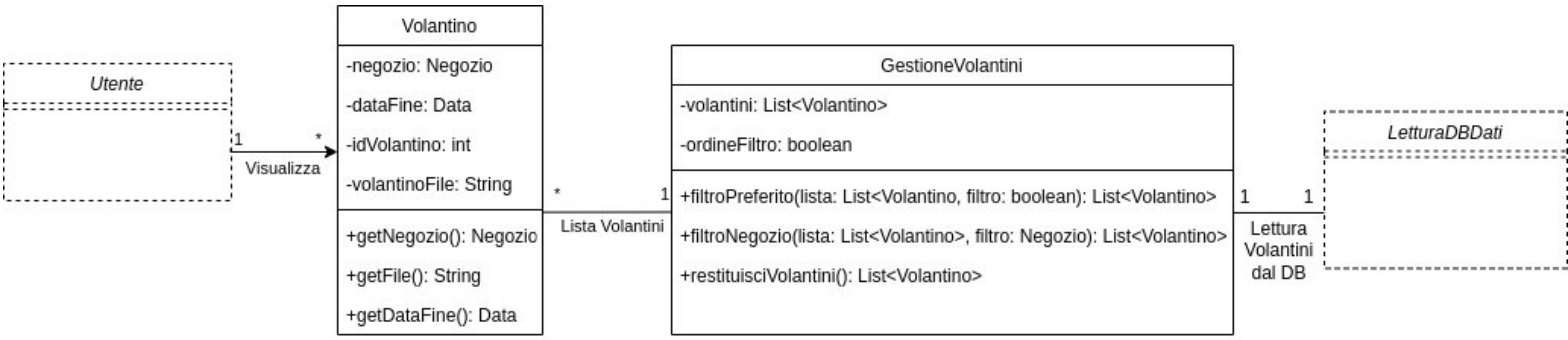


Figura 5. Diagramma delle classi, con relative associazioni, coinvolte nella Visualizzazione dei Volantini.

1.5 Visualizzazione Sconti

Il diagramma a sinistra illustra le classi coinvolte nella visualizzazione degli sconti e le loro associazioni. Come si può notare dallo schema, la classe **Utente** è associata alla classe **Sconti** poiché un utente può visualizzare svariati sconti all'interno dell'apposita pagina. Ogni sconto contiene i dati relativi al negozio a cui si riferisce, un id univoco, il proprio valore, la categoria di prodotti a cui si applica (se si applica ad una categoria) e la lista di prodotti a cui si applica, quasi tutti ottenibili mediante i metodi getNegozio, getProdotti, getValore e getCategoria. L'ottenimento degli sconti dal database e il loro filtro sono stati affidati alla classe **GestioneSconti** di cui viene creata un'istanza quando l'utente apre la pagina sconti. Tale classe filtra gli oggetti da visualizzare mediante i propri metodi (*filtroNegozio*, *filtroPreferiti*, *filtroProdotto*, *filtroCategoria*, *filtroValore*), che modificano ciò che la classe visualizza nell'apposita pagina. È presente un parametro *ordineFiltro* che stabilisce se l'ordinamento già intrinseco ai filtri deve essere crescente o decrescente. Sempre osservando il grafico si può notare come la classe **GestioneSconti** ottenga tutti i dati necessari dalla classe **LetturaDBDati**.

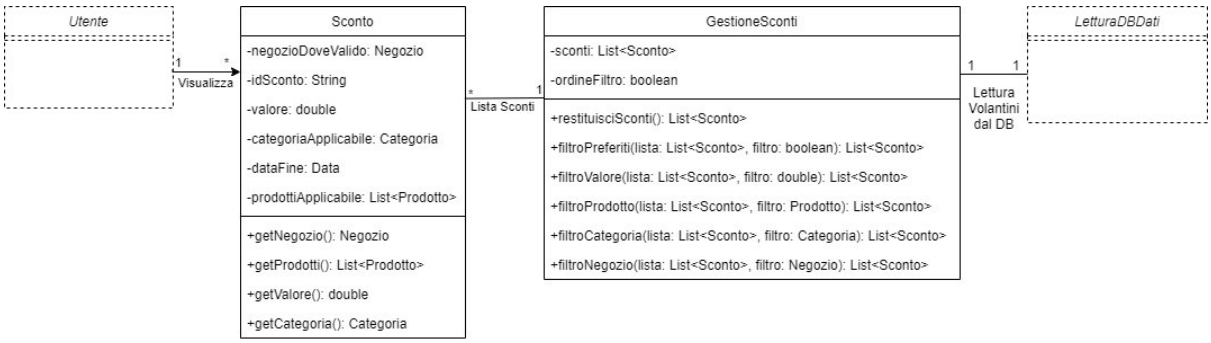


Figura 6. Diagramma delle classi, con relative associazioni, coinvolte nella Visualizzazione degli Sconti.

1.7 Aggiornamento Sito

La classe **AggiornamentoSito** si occupa di leggere dal DB i dati precedenti relativi al prezzo dei prodotti, di chiamare le API dei vari negozi e di scrivere i nuovi dati nel database dei dati iSpesa.

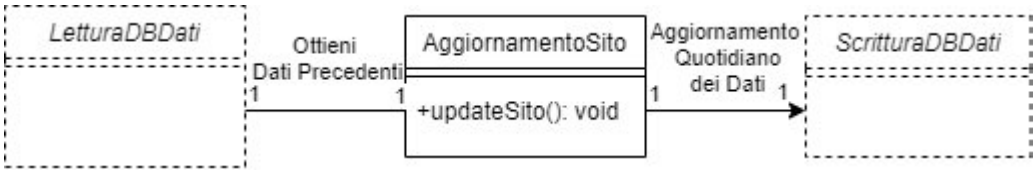


Figura 7. Diagramma delle classi, con relative associazioni, coinvolte nell'aggiornamento del sito.

1.8 Recensioni

Procediamo ora a descrivere le classi relative alle recensioni. La classe **Recensione** comprende tutte i dati, nonché un id univico, relativi ad una recensione coni relativi getter (*getTitolo*, *getTesto*, *getNumeroStelle*, *getUtente*, *getdata*, *getID*). Quando un utente scrive una recensione questa viene semplicemente creata ed inserita nel database mediante la classe **ScritturaDBDati**. L'amministratore, dal canto suo, si interfaccia invece con la classe **GestoreRecensioni** che permette allo stesso di modificare il testo di una recensione (mediante *oscuraTesto* e *oscuraParte*) o di eliminare una recensione. Questa classe ottiene tutti i dati necessari dalla classe **LetturaDBDati** e salve le modifiche nella classe **ScritturaDBDati**. La classe **GestoreRecensioni** permette di trovare le recensioni relative ad un solo negozio.

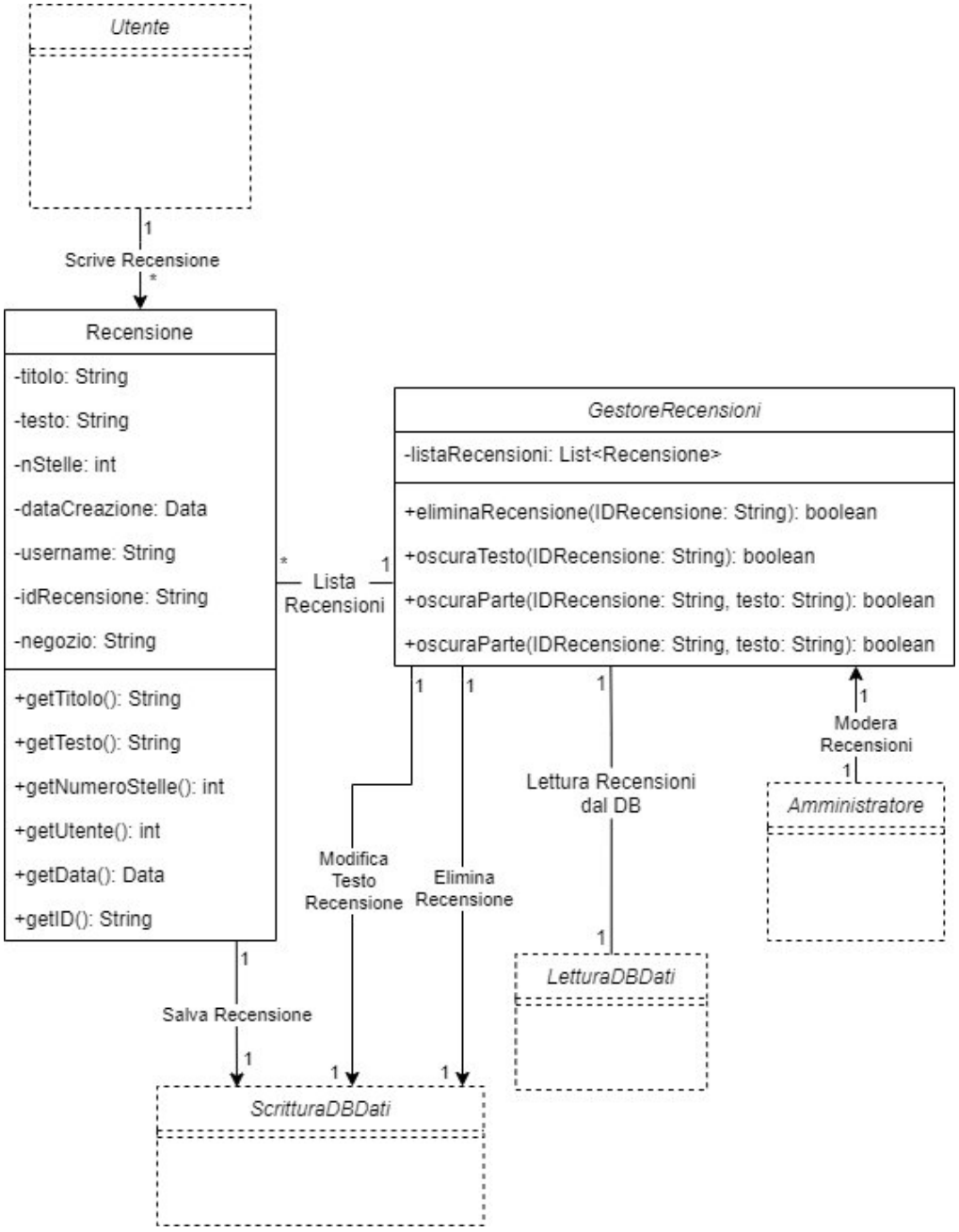


Figura 8. Diagramma delle classi, con relative associazioni, coinvolte nelle recensioni e nella loro gestione.

1.9 Moderazione Utenti

Abbiamo poi le classi relative alla moderazione degli utenti. Ciò viene gestito interamente dalla classe **GestioneUtenti**, di cui viene creata un'istanza ogni volta che un amministratore apre la pagina utenti. La classe **GestioneUtenti** permette di eliminare un utente (*eliminaUtente*), modificarne la password (*modificaPassword*) o il numero di telefono(*modificaNumeroUtente*). Per far ciò la classe si interfaccia con un'istanza della classe **GestoreAutenticazione** e con la classe **ScritturaDBUtenti**. La classe, inoltre, si interfaccia con la classe **InterfacciaGmail** per inviare tutte le notifiche del caso all'utente. Ovviamente i dati vengono ottenuti mediante la classe **LetturaDBUtenti**.

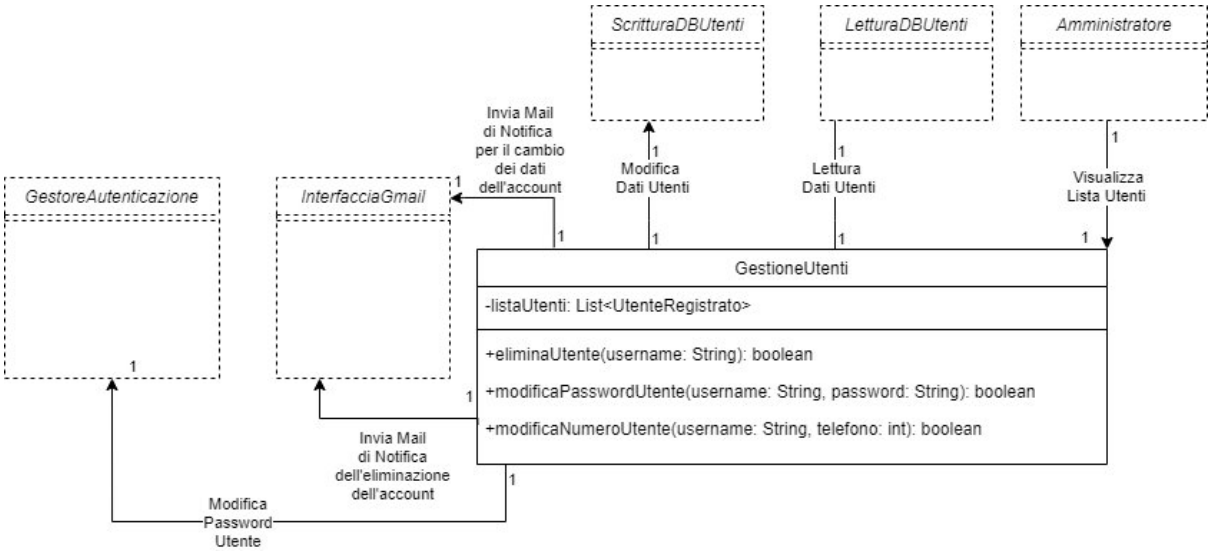


Figura 9. Diagramma delle classi, con relative associazioni, coinvolte nella moderazione.

1.10 Interfaccia Gmail

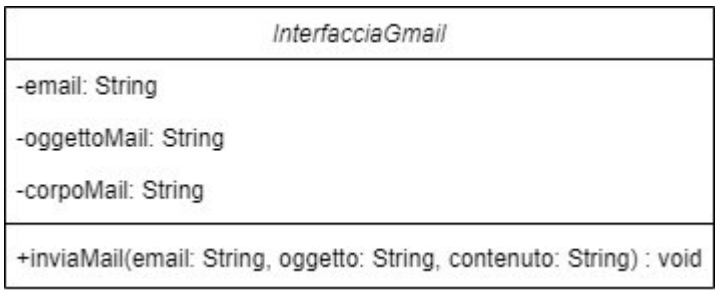


Figura 10. Classe InterfacciaGmail.

La classe **InterfacciaGmail** ha il compito di gestire l'invio delle mail di comunicazione per gli utenti ed anche di inviare le segnalazioni degli utenti all'apposito indirizzo. Il suo unico metodo, *inviaMail*, comunica con l'API Gmail per inviare suddette mail

1.11 Classi di supporto

Le suddette classi di supporto hanno lo scopo di fornire un modo per conservare in maniera apposita gli indirizzi dei negozi (questo vale per la classe Indirizzo), le date (ciò vale per la classe date) e di avere un set di categorie per i prodotti (ciò vale per l'enumerativo Categoria). La classe Data prevede un metodo passata che permette di controllare se la data in questione è già passata oppure no.

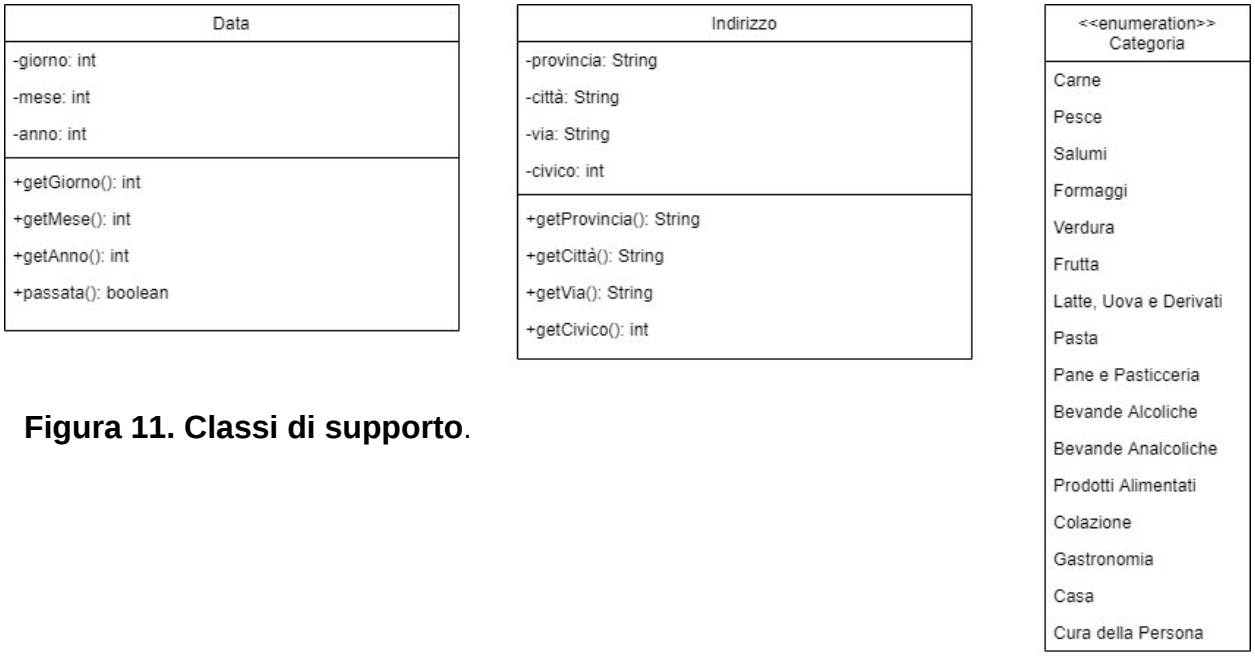


Figura 11. Classi di supporto.

1.12 Classe di Lettura/ScritturaDBUtenti

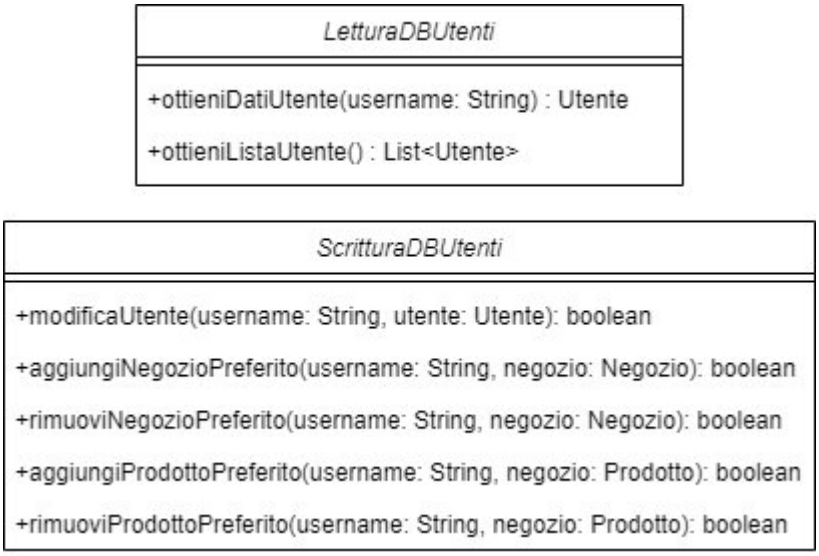


Figura 12. Classi di Lettura/ScritturaDBUtenti.

Le classi **Lettura**/**ScritturaDBUtenti** hanno lo scopo di gestire l'interfacciamento con le tabelle del database iSpesa che hanno il compito di conservare i dati degli utenti. La classe **LetturaDBUtenti** permette di ottenere i dati di un utente (con il metodo *ottieniDatiUtente*) o tutta la lista utenti (con il metodo *ottieniListaUtente*). La classe **ScritturaDBUtenti**, invece, permette di modificare i dati di un utente ed aggiungere/rimuovere un negozio/prodotto preferito, rispettivamente con i metodi *modificaUtente*, *aggiungi*/*rimuoviNegozioPreferito* e *aggiungi*/*rimuoviProdottoPreferito*.

1.13 Classe di Lettura/ScritturaDBDati

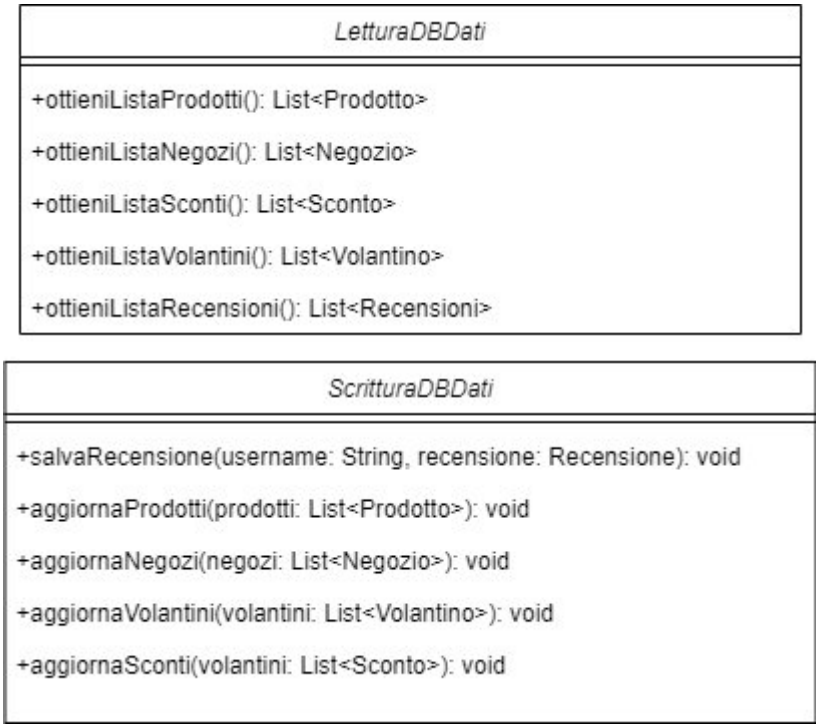


Figura 13. Classi di Lettura/ScritturaDBDati.

Le classi **Lettura**/**ScritturaDBDati** hanno lo scopo di gestire l'interfacciamento con le tabelle del database iSpesa che hanno il compito di conservare i dati relativi ai prodotti/negozi/sconti/volantini/ recensioni. La classe **LetturaDBDati** permette di ottenere la lista dei prodotti/negozi/sconti/volantini mediante i metodi *OttieniListaProdotti/Negozi/Sconti/ Volantini/Recensioni*. La classe **ScritturaDBDati**, invece, permette di salvare le recensioni (mediante il metodo *salvaRecensione*) e di aggiornare i dati del database mediante i metodi *aggiornaProdotti/Negozi/Volantini/ Sconti*.

1.14 Diagramma delle classi completo

Di seguito l'intero diagramma delle classi del sistema iSpesa.

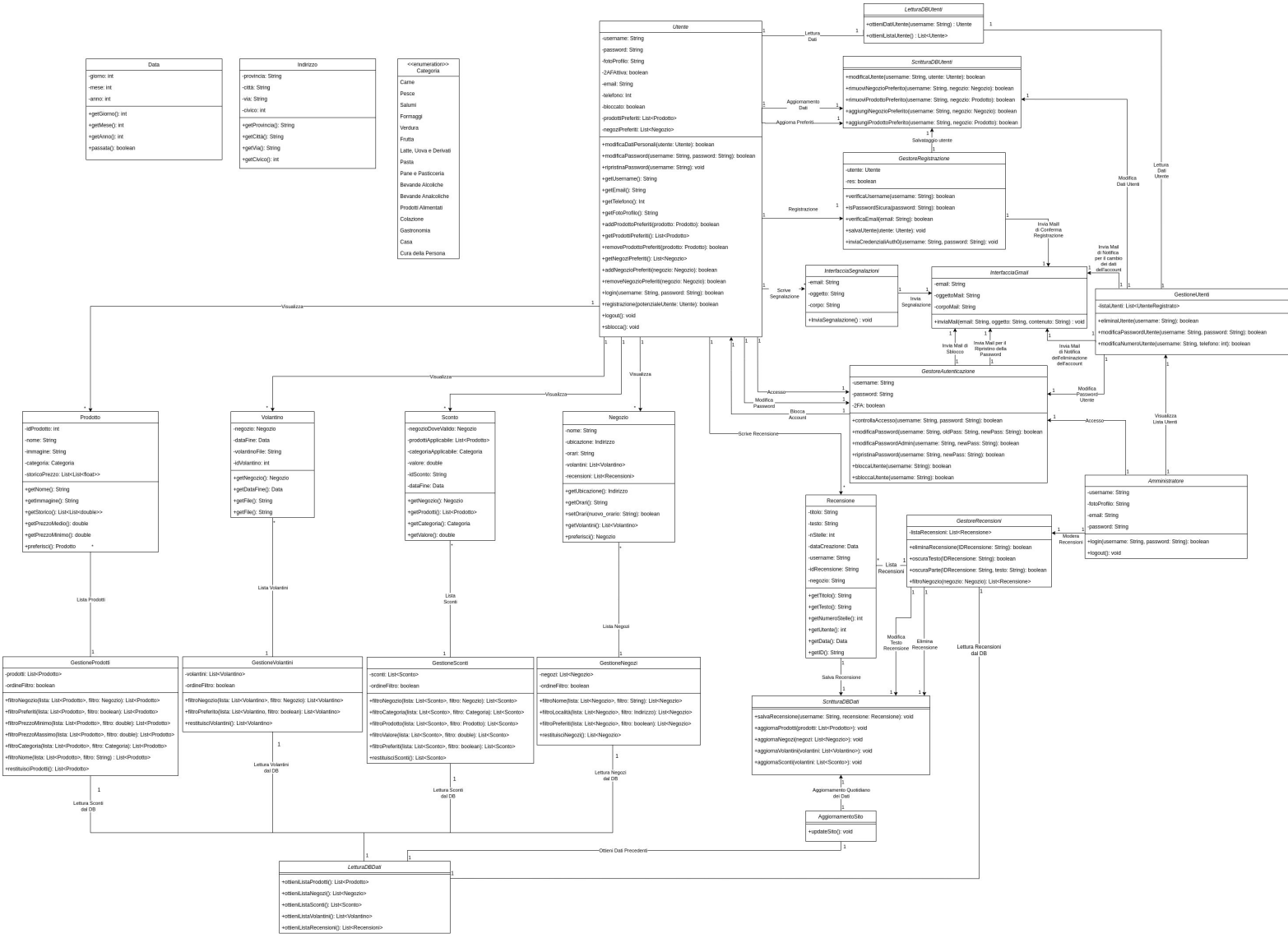


Figura 14. Classi delle classi del sistema iSpesa.

2. Codice in OCL

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).

2.1 Vincoli OCL classe GestoreRegistrazione

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

context GestoreRegistrazione::salvaUtente(utente: Utente)
pre: verificaEmail(email: String) = true
AND isPasswordSicura(password: String) = true
AND verificaUsername(username: String) = true
post: res = true

2.2 Vincoli OCL classe GestoreAutenticazione

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

context GestoreAutenticazione::modificaPassword(username: String, oldPass:String, newPass: String): boolean
pre: oldpass = password
AND isPasswordSicura(newPass: String) = true

2.3 Vincoli OCL classe Utente

Questi vincoli specificato che per effettuare il logout, e visualizzare il proprio profilo personale è necessario aver effettuato l'accesso al sistema (username deve essere non nullo), mentre username deve essere nullo (cioè è necessario non aver fatto l'accesso al sistema) prima di fare la registrazione e il login. Mentre per sbloccare l'account è necessario che sia stato bloccato.

context Utente::logout()
pre: username != NULL

context Utente::login(username: String, password: String): boolean
pre: username = NULL
AND bloccato = false
post: username != NULL

context Utente::registrazione(potenzialeUtente: Utente)

pre: username = NULL

context Utente::sblocca()
pre: bloccato = true
post: bloccato = false

context Utente::getUsername(): String
pre: username != NULL

context Utente::getEmail(): String
pre: username != NULL

context Utente::getTelefono(): int
pre: username != NULL

context Utente::getFotoProfilo(): String
pre: username != NULL

2.4 Vincoli OCL classe Amministratore

Questi vincoli specificato che per effettuare il logout è necessario aver effettuato l'accesso al sistema (username deve essere non nullo), mentre username deve essere nullo (cioè è necessario non aver fatto l'accesso al sistema) prima di fare il login.

context Amministratore::logout()
pre: username != NULL

context Amministratore::login(username: String, password: String): boolean
pre: username = NULL
post: username != NULL

2.5 Vincoli OCL classe Recensione

Questi vincoli specificano invarianti per alcuni attributi tipici di una recensione.

context Recensione **inv**: nStelle >= 1 AND nStelle <=5
context Recensione **inv**: testo->length() <= 1024

2.6 Vincoli OCL classe AggiornamentoSito

Questo vincolo rappresenta ciò che deve avvenire prima della chiamata del metodo updateSito sia ciò che deve avvenire dopo.

context AggiornamentoSito::updateSito()
pre: LetturaDBDati.ottieniListaProdotti()
AND LetturaDBDati.ottieniListaSconti()
AND LetturaDBDati.ottieniListaVolantini()
AND LetturaDBDati.ottieniListaNegozi()
post: ScritturaDBDati.aggiornaProdotti(prodotti: List<Prodotto>)
AND ScritturaDBDati.aggiornaSconti(sconti: List<Sconto>)
AND ScritturaDBDati.aggiornaVolantini(volantini: List<Volantino>)
AND ScritturaDBDati.aggiornaNegozi(sconti: List<Negozio>)

2.7 Vincoli OCL classe Prodotto

Questo vincolo specifica un invariante per un attributo di un prodotto

context Prodotto **inv**: categoria != NULL

2.8 Vincoli OCL classe Volantino

Questo vincolo specifica un invariante per un attributo di un volantino

context Prodotto **inv**: dataFine.passata() != false

2.9 Vincoli OCL classe Sconto

Questo vincolo specifica degli invarianti per alcuni attributi di uno sconto.

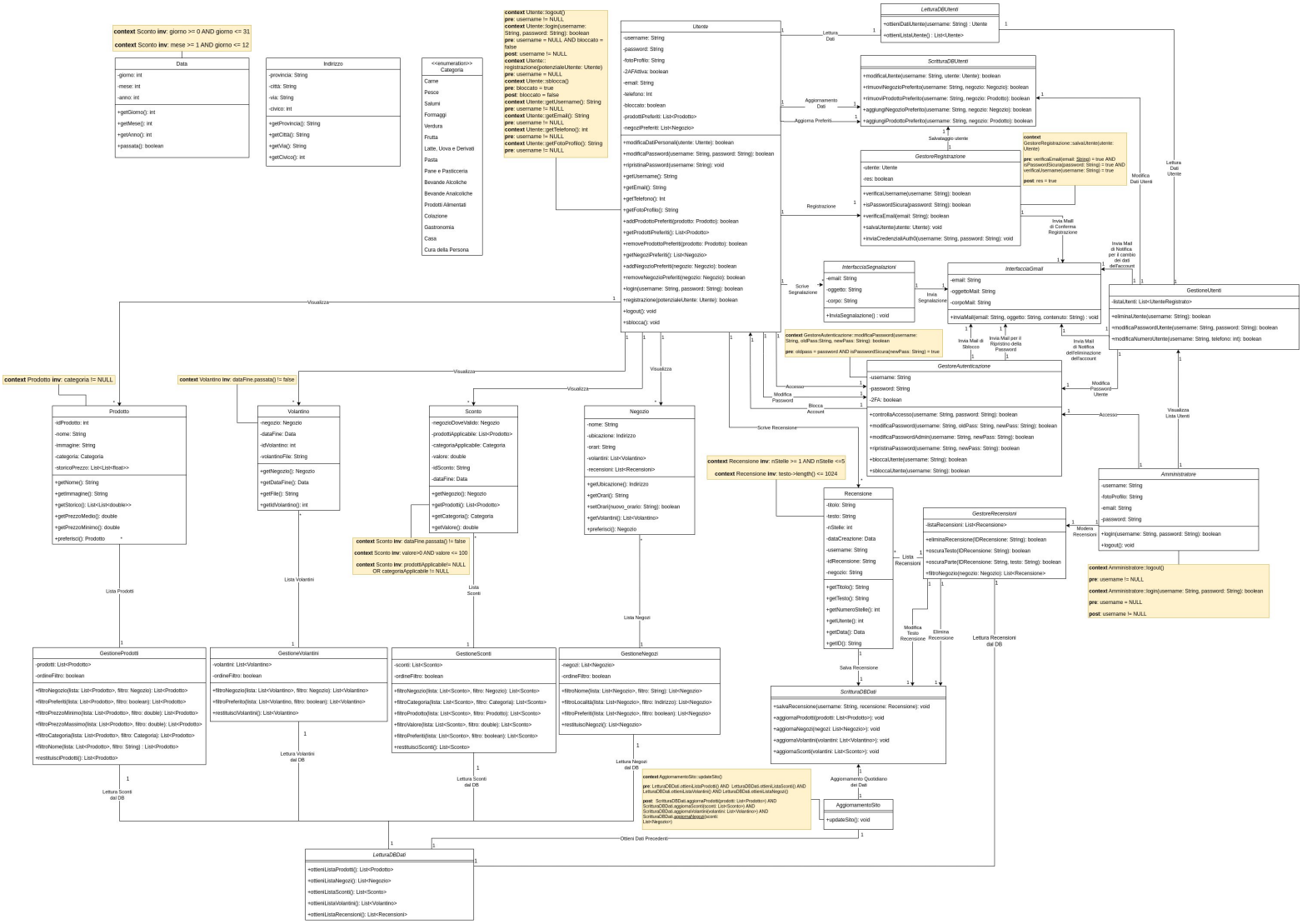
context Sconto **inv**: dataFine.passata() != false
context Sconto **inv**: valore >0 AND valore <= 100
context Sconto **inv**: prodottiApplicabile != NULL OR categoriaApplicabile != NULL

2.10 Vincoli OCL classe Data

Questo vincolo specifica degli invarianti per alcuni attributi di una data.

context Sconto **inv**: giorno >= 0 AND giorno <= 31
context Sconto **inv**: mese >= 1 AND giorno <= 12

2.11 Diagramma delle classi con codice OCL associato



[Figura 15. Classi delle classi del sistema iSpesa con codice OCL.]