Documento: BookEasy – “Antico Granaio” – Architettura

Revisione: 0.3

UNIVERSITY OF TRENTO

Dipartimento di Ingegneria e Scienza dell’Informazione

Progetto:

BookEasy – “Antico Granaio”

Titolo del documento:

Architettura

Informazioni Documento

Nome Doc.	D3-BookEasy-AnticoGranaio_Architettura	**Numero Doc.**	A3
Descrizione	Il documento include i diagrammi delle varie classi e il relativo codice in OCL.		

Sommario

1. Scopo del documento

Il presente documento delinea l’architettura del progetto BookEasy – “Antico Granaio” utilizzando i diagrammi delle classi in UML (“Unified Modeling Language”) e il relativo codice in OCL (“Object Constraint Language”).

Nel documento precedente, è stato già presentato il diagramma relativo agli Use Case, il diagramma di contesto e quello dei componenti. Tenendo conto della struttura precedentemente illustrata, si procede nella definizione dell’architettura del sistema.

Verranno dettagliate le classi da implementare a livello di codice e logica che regola il comportamento del software.

Le classi fondamentali per la costruzione del sistema sono rappresentate attraverso un diagramma delle classi in linguaggio UML, fornendo una visione chiara e strutturata degli elementi chiave del progetto. Contemporaneamente, la logica sottostante che guida il comportamento del software viene descritta utilizzano il linguaggio OCL.

La scelta di OCL è giustificata dalla necessità di esprimere concetti che non possono essere rappresentati in modo formale ne contesto UML.

Questo approccio integrato tra diagramma UML e OCL si propone di offrire una comprensione completa dell’architettura del sistema BookEasy – “Antico Granaio”.

2. Diagramma delle classi

In questo capitolo, vengono presentate le classi previste nell’ambito del progetto BookEasy – “Antico Granaio”.

I componenti presenti nel diagramma dei componenti si configurano come delle classi.

Ogni classe individuata è caratterizzata da un nome, un elenco di attributi che specificano i dati gestiti dalla classe e un elenco di metodi che delineano le operazioni previste all’interno di essa.

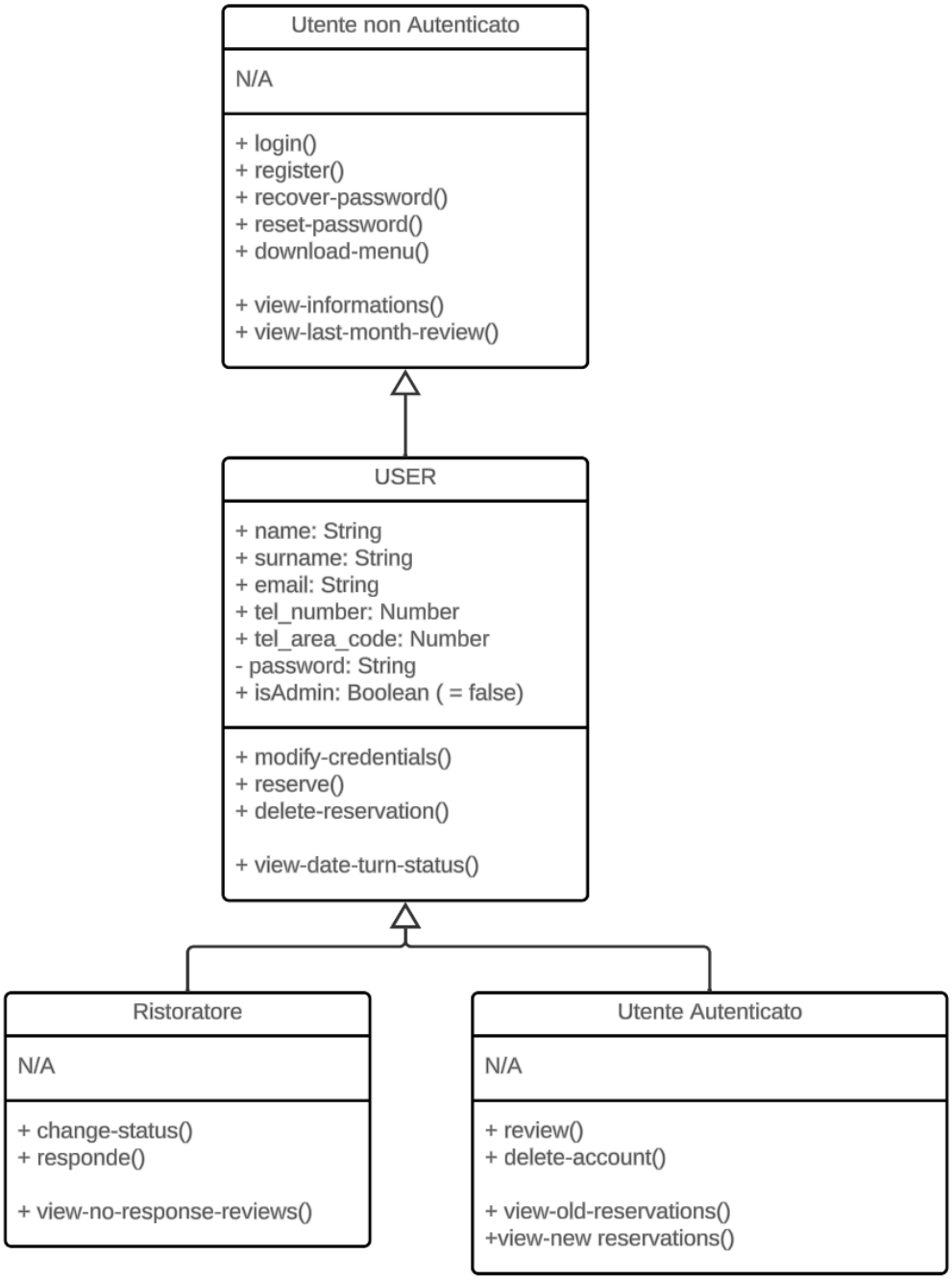
La possibilità di associare una classe ad altre consente di fornire dettagli sulle relazioni tra le diverse categorie (le associazioni sono state pensate come numero di istanze concorrenti che possono essere istanziate all’interno della WebApp).

Riportiamo di seguito le classi individuate a partire dai diagrammi di contesto e dei componenti. Durante questo processo, è stata data particolare importanza alla massimizzazione della coesione.

a. Utenti

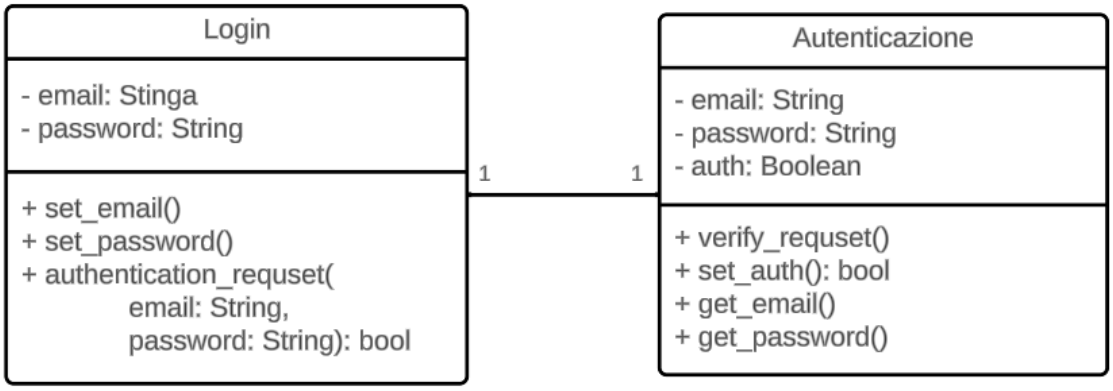
Analizzando i requisiti funzionali e il diagramma dei componenti, si possono individuare tre tipologie di utente:

- **Utente non autenticato**: colui che può effettuare login, registrazione, recuperare la password, ripristinare la password, scaricare il menù e visualizzare tutte le informazioni relative al ristorante (ma non alle prenotazioni).
- **User:** colui che, oltre a poter fare le stesse code dell’utente non autenticato, può prenotare, cancellare le sue prenotazioni, modificare le sue credenziali, e lasciare recensioni.
- **Utente Autenticato** e **Ristoratore:** questi due possono fare le stesse cose, e hanno gli stessi attributi, dello User. Ma si differenziano tra loro grazie alle differenze tra le loro attività.


Utente non Autenticato
N/A
+ login()
+ register()
+ recover-password()
+ reset-password()
+ download-menu()
+ view-informations()
+ view-last-month-review()
USER
+ name: String
+ surname: String
+ email: String
+ tel_number: Number
+ tel_area_code: Number
- password: String
+ isAdmin: Boolean (= false)
+ modify-credentials()
+ reserve()
+ delete-reservation()
+ view-date-turn-status()
Ristoratore
N/A
+ change-status()
+ responde()
+ view-no-response-reviews()
Utente Autenticato
N/A
+ review()
+ delete-account()
+ view-old-reservations()
+view-new reservations()

b. Login e Autenticazione

Analizzando il diagramma dei componenti, sono state identificate le classi **Login** e **Autenticazione** per dare la possibilità all’utente non autenticato di effettuare il login al sistema. Egli dovrà quindi compilare un form di login che prende in input indirizzo e-mail e password che dovranno essere verificati nel processo di autenticazione.



c. Gestione Utente

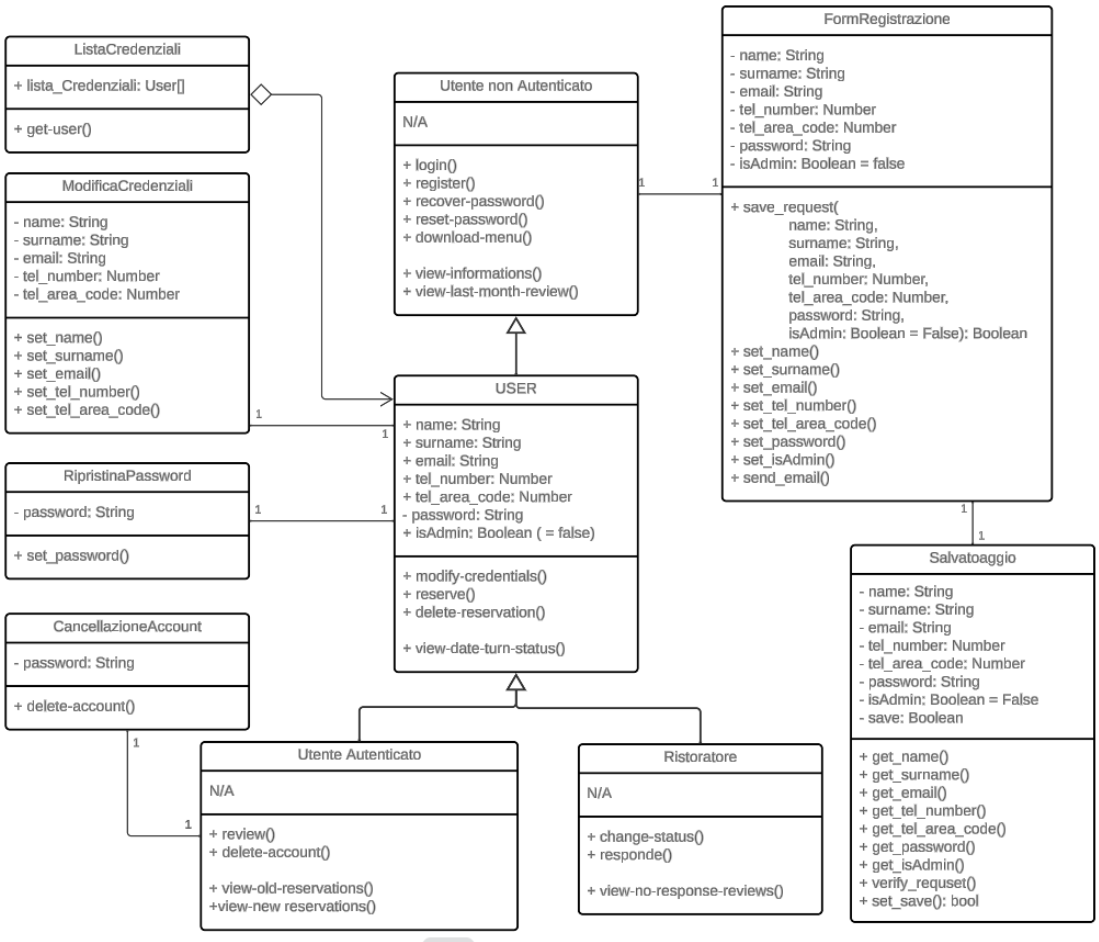
Analizzando i requisiti funzionali e il diagramma di contesto, è emersa la necessità di creare le seguenti classi che sono in grado di aiutare nella gestione dell’utente.

ListaCredenziali serve per facilitare il recupero delle informazioni riguardanti gli utenti.

FormRegistrazione consiste in un form che va compilato dall’utente non autenticato per registrarsi, ciò si appoggia alla classe **Salvataggio** per la verifica dei dati.

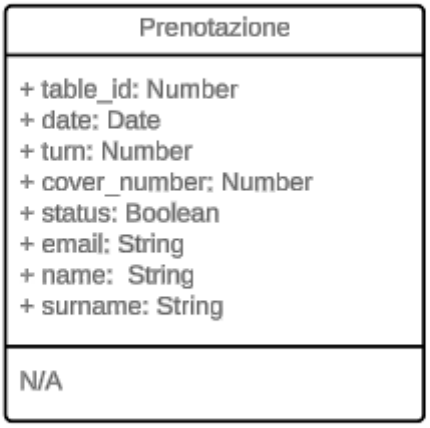
RipristinaPassword si occupa di aggiornare la password nel caso l’utente lo richieda, **ModificaCredenziali** si occupa, invece, di modificare gli attributi.

CancellazioneAccount è la classe che si occupa dell’eliminazione delle utenze.



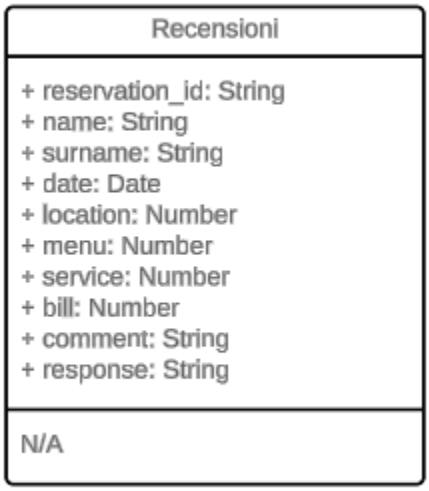
d. Prenotazione

Analizzando i requisiti funzionali e il diagramma di contesto, si è individuata la classe **Prenotazione,** la quale aiuta nella memorizzazione delle prenotazioni.



e. Recensioni

Analizzando i requisiti funzionali e il diagramma di contesto, si è individuata la classe **Recensioni,** la quale memorizza le recensioni relative alle prenotazioni.



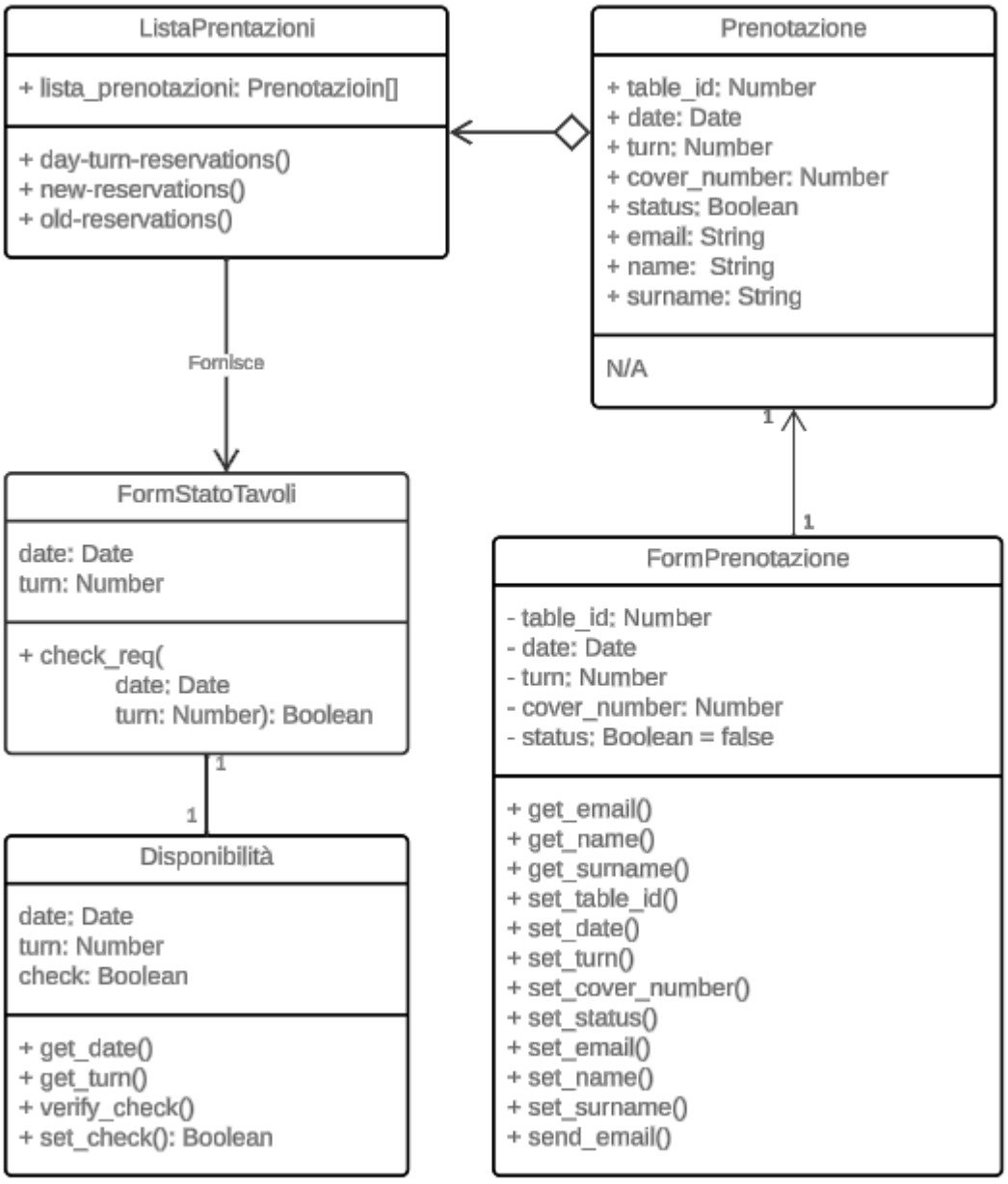
f. Gestione Prenotazioni

Analizzando i requisiti funzionali e il diagramma di contesto, sono emerse le seguenti classi che hanno il compito di gestire le prenotazioni.

ListaPrenotazioni serve per facilitare il recupero delle informazioni riguardanti gli utenti.

FormStatoTavoli consiste in un form che va compilato dallo user per visualizzare lo stato dei tavoli, ciò si appoggia alla classe **Disponibilità** per la verifica dei dati.

Per effettuare la prenotazione è stata individuata la classe **FormPrenotazione** che si occupa della raccolta dei dati e salvataggio della prenotazione.



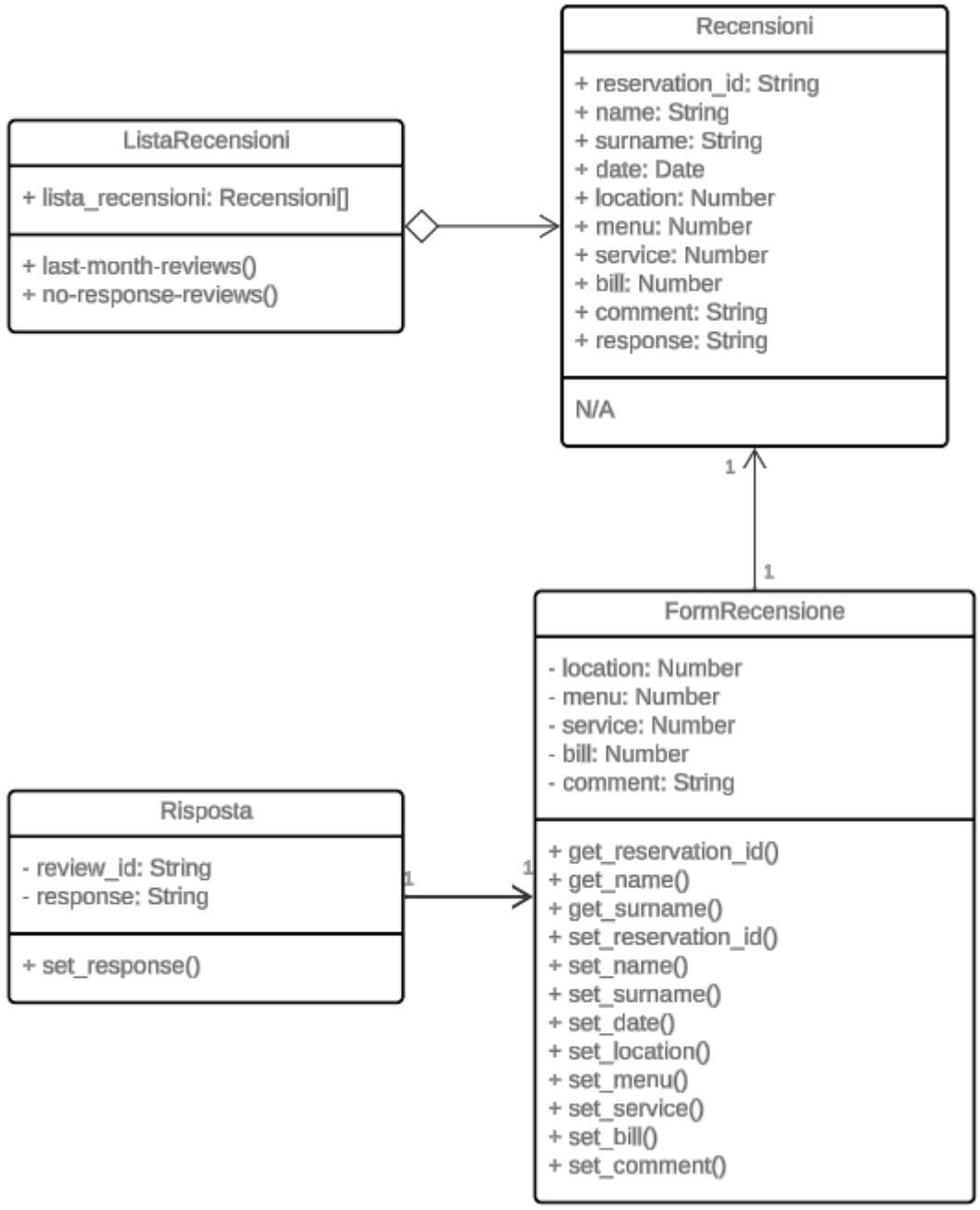
g. Gestione Recensioni

Analizzando i requisiti funzionali e il diagramma di contesto, sono emerse le seguenti classi che hanno il compito di gestire le recensioni.

FormRecensione consiste in un form che va compilato dall’utente autenticato per lasciare una recensione.

ListaRecensioni serve per facilitare il recupero delle informazioni riguardanti le recensioni.

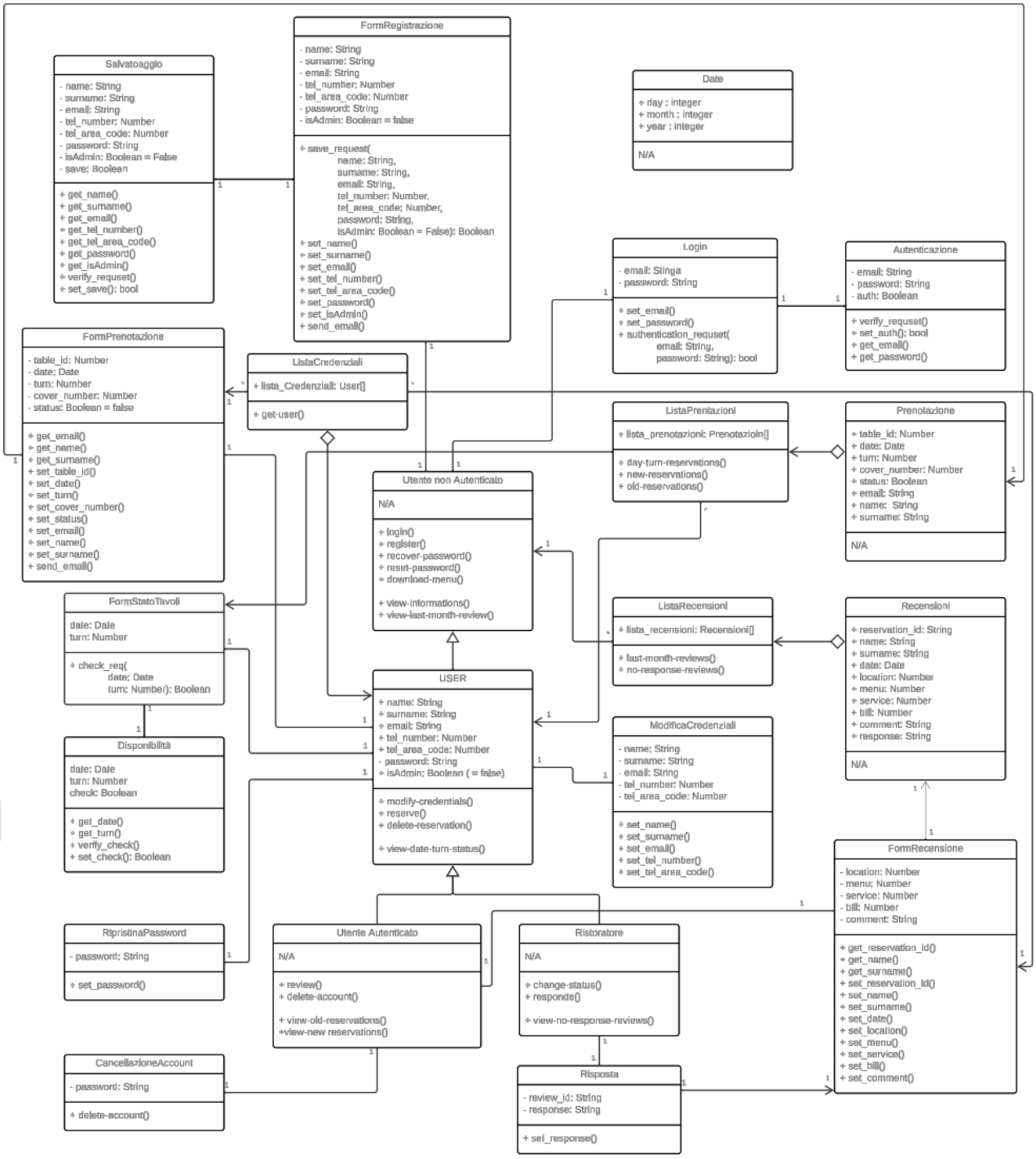
Risposta, invece, si occupa di impostare la risposta fatta dal ristoratore alle rispettive recensioni.



h. Diagramma delle classi complessivo

Di seguito viene riportato il diagramma delle classi, che include tutte le classi fino ad ora presentate.

Oltre alle classi precedentemente descritte, si è fatto uso di una classe ausiliaria, ossia **Date**. Questa classe è stata aggiunta al fine di delineare tipi strutturati eventualmente impiegati, ad esempio, negli attributi delle classi.

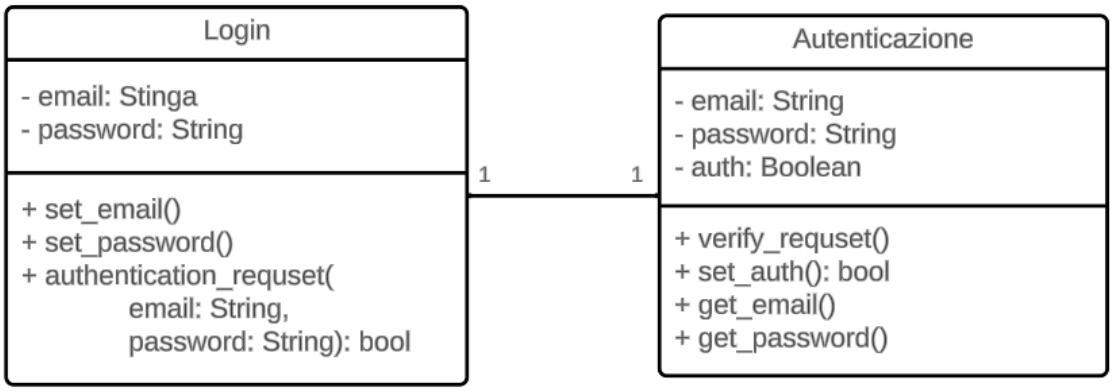


3. Codice in Object Constraint Language

In questo capitolo, viene fornita una formalizzazione della logica associata a specifiche operazioni di determinate classi in un contesto altamente formale. Questa logica è espressa attraverso l'Object Constraint Language (OCL) poiché i concetti coinvolti risultano inesprimibili in qualsiasi altra forma formale all'interno del contesto di UML.

a. Login

Affinché l’operazione di login avvenga con successo, vi sono le seguenti condizioni da rispettare nel contesto delle classi di **Login** e **Autenticazione**.



Prima di richiedere l’autenticazione bisogna compilare il campo *email* e il campo *password*, e ciò viene espresso in OCL tramite una precondizione con questo codice:

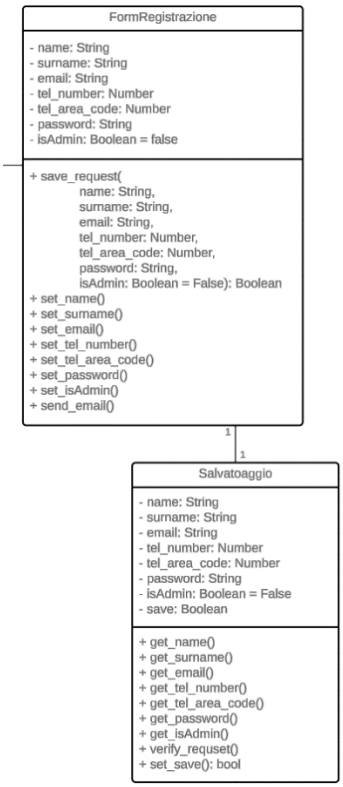
```
context Login::authentication_request()
pre: (self.email != null) AND (self.password != null)
```

Inoltre, dopo che l’utente viene autorizzato, lo stato *auth* viene messo a true, e cio viene rappresentato in OCL tramite una postcondizione con questo codice:

```
context Autenticazione::set_auth()
post: (self.auth = true)
```

b. Registrazione

Affinché l’operazione di registrazione avvenga con successo, vi sono le seguenti condizioni da rispettare, nel contesto delle classi di **FormRegistrazione** e **Salvataggio**.



Prima di richiedere il salvataggio bisogna compilare il *name*, *surname*, *email*, *tel_number*, *tel_area_code* e *password*, e ciò viene espresso in OCL tramite una precondizione con questo codice:

```
context FormRegistrazione::save_request()
pre: (self.name != null) AND (self.surname != null) AND
(self.email != null) AND (self.tel_number != null) AND
(self.tel_area_code != null) AND (self.password != null)
```

Inoltre, dopo che l’utente viene autorizzato, lo stato *save* viene messo a true, e ciò viene rappresentato in OCL tramite una postcondizione con questo codice:

```
context Salvatoaggio::set_save()
post: (self.save = true)
```

c. Ripristina Password

Nel contesto della classe **RipristinaPassword**, la nuova password inserita deve essere compilata e deve anche risultare differente da quella attuale, ciò viene rappresentato in OCL tramite un’invariante con questo codice:

```
context RipristinaPassword::set_password()
inv: (self.password != null) AND (self.password !=
USER.password)
```

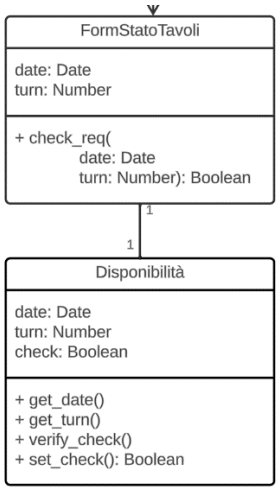
d. Cancellazione Account

Nel contesto della classe **CancellazioneAccount**, la password inserita deve essere compilata e corrispondere a quella presente nel database, ciò viene rappresentato in OCL tramite un’invariante con questo codice:

```
context Ristoratore::delete-account()
inv: (self.password != null) AND (self.password =
USER.password)
```

e. Disponibilità Tavoli

Affinché l’operazione di controllo dello stato dei tavoli venga svolta con successo, vi sono le seguenti condizioni da rispettare, nel contesto delle classi di **FormStatoTavoli** e **Disponibilità**.



Prima di richiedere il salvataggio bisogna compilare *date* e *turn*, e ciò viene espresso in OCL tramite una precondizione con questo codice:

```
context FormStatoTavoli::check_req()
pre: (self.date != null) AND (self.turn != null)
```

Inoltre, dopo che la disponibiltà viene verificata, lo stato *check* viene messo a true, e ciò viene rappresentato in OCL tramite una postcondizione con questo codice:

```
context Disponibilità::set_check()
post: (self.check = true)
```

f. Prenotazione

Nel contesto della classe **FormPrenotazione**, bisogna compilare la *table_id*, *date*, *turn*, e *cover_number*, ciò viene rappresentato in OCL tramite un’invariante con questo codice:

```
context User::reserve()
inv: (self.table_id != null) AND (self.date != null) AND
(self.turn != null) AND  (self.cover_number != null)
```

g. Recensione

Nel contesto della classe **FormRecensione**, è necessario che ci si riferisca ad una prenotazione, ciò viene rappresentato in OCL tramite un’invariante con questo codice:

```
context UtenteAutenticato::review()
inv: (self.get_reservation_id() != null)
```

h. Risposta

Nel contesto della classe **Risposta**, è necessario compilare la risposta ed è anche necessario che essa corrisponda ad una recensione, ciò viene rappresentato in OCL tramite un’invariante con questo codice:

```
context Risposta::set_response()
inv: (self.review_id != null) AND (self.response != null)
```

4. Diagramma delle classi con codice OCL

Infine, viene qui riportato il diagramma delle classi con tutte le classi fino ad ora presentate ed il codice OCL individuato.