Dipartimento di Ingegneria e Scienza dell’Informazione

Progetto:

BookEasy – “Antico Granaio”

Titolo del documento:

Architettura

Informazioni Documento

Nome Doc.	D2-BookEasy-AnticoGranaio_Architettura	**Numero Doc.**	SR3
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. Adesso, tenendo conto di questa progettazione, si procede nella definizione dell’architettura del sistema.

Verranno dettagliate le classi da implementare a livello di codice e la 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 utilizzando l’Object Constraint Language (OCL).

La scelta di OCL è giustificata dalla necessità di esprimere concetti che non possono essere rappresentati in modo formale nel 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 del 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.

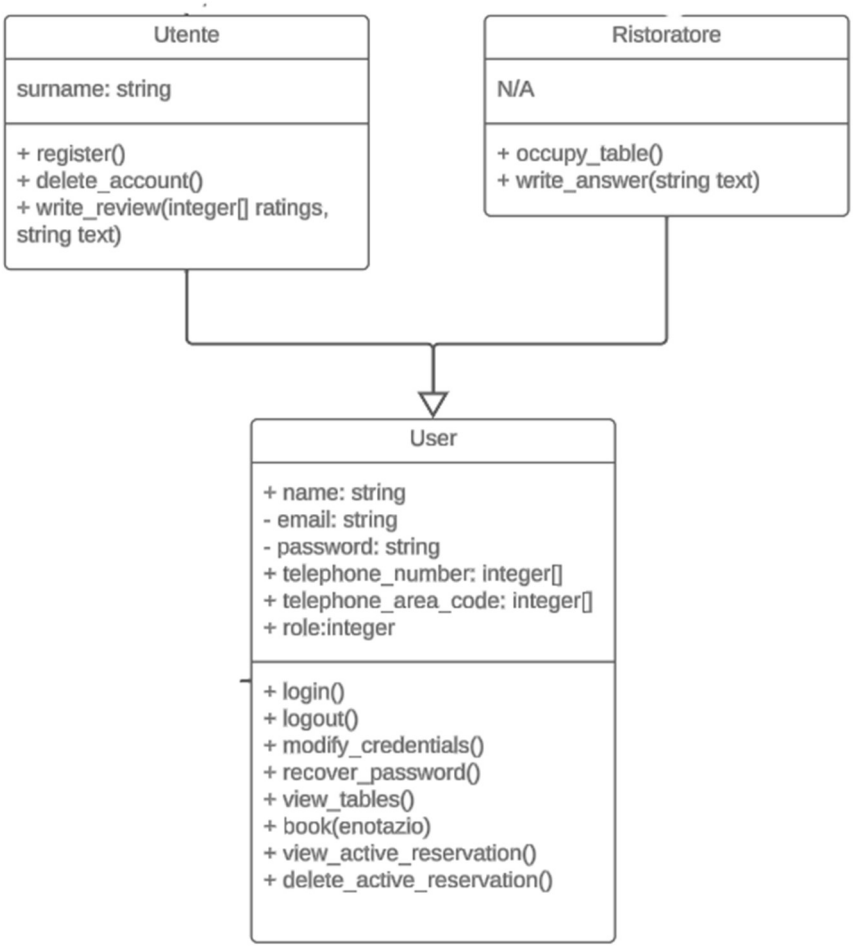
Riportiamo di seguito le classi individuate a partire dai diagrammi di contesto e dei componenti. Durante questo processo, è stata data particolare attenzione alla massimizzazione della coesione e alla minimizzazione dell’accoppiamento tra le categorie.

a. Utenti e sistemi esterni

Analizzando il diagramma di contesto realizzato per il progetto BookEasy – “Antico Granaio”, si nota la presenza di due attori: “user” e “ristoratore”.

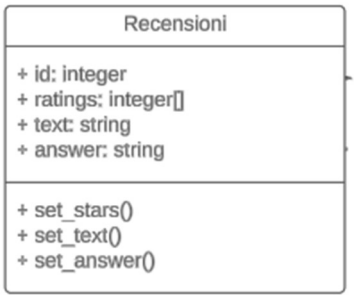
Lo “user” è colui che utilizza l’applicazione per prenotare un tavolo del ristorante, per visualizzare tutte le informazioni legate al locale e recensire le proprie esperienze. Il “ristoratore”, invece, è colui che gestisce il ristorante e tutte le interazioni con i suoi clienti tramite controlli sulle prenotazioni e risposte alle recensioni.

Entrambi questi attori hanno delle funzioni e degli attributi specifici, ma hanno anche molto in comune. Sono state, quindi, individuate due classi `User` e `Ristoratore` con funzioni e attributi specifici, in aggiunta è stata anche definita la classe `Utente` con funzioni e attributi comuni alle due classi appena citate che le collega tramite una generalizzazione.



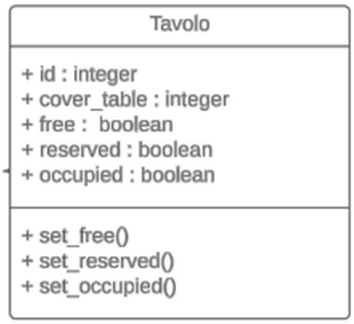
b. Sistema di recensioni

Analizzando il componente “Scrittura recensioni”, il quale dà la possibilità agli utenti autenticati di inserire una recensione relativa ad una prenotazione passata e offre l’opportunità al ristoratore di rispondere, definiamo la classe `Recensione`, la quale memorizzerà i valori e il testo (opzionale) inseriti nel caso fosse un utente a scrivere, mentre memorizzerà il testo inserito nel caso fosse il ristoratore a farlo.



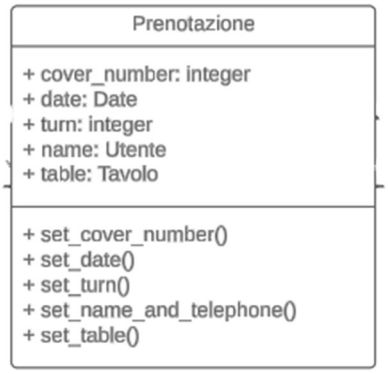
c. Gestione dei Tavoli

Analizzando il componente “Visualizzazione Tavoli”, che prevede la visualizzazione dei tavoli tramite una pianta interattiva del locale e la modifica dello stato degli stessi tramite prenotazione, cancellazione o modifica da parte del ristoratore, definiamo la classe `Tavolo`, che permette all’utente e al ristoratore la visualizzazione e l’interazione con lo stato dei tavoli.



d. Gestione Prenotazioni

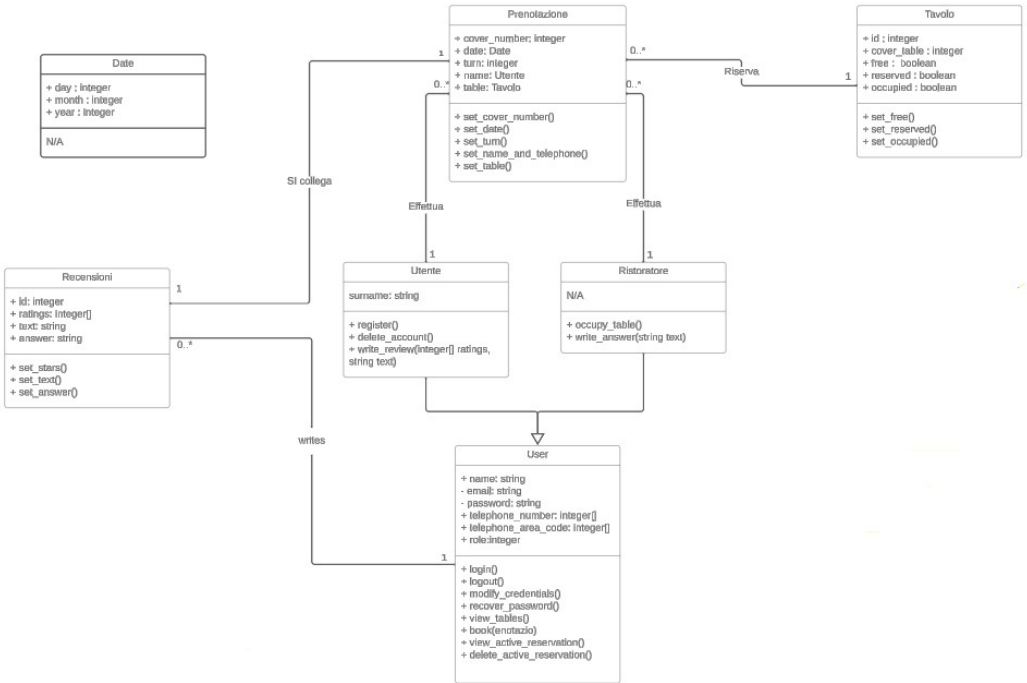
Analizzando i componenti “Gestione prenotazioni utente” e “Gestione prenotazioni ristoratore”, risulta necessario creare la classe `Prenotazione` (che interagisce direttamente con la classe `Tavolo`), la quale memorizzerà le informazioni necessarie per creare o cancellare una prenotazione per poi memorizzarle nel database.



e. Diagramma delle classi complessivo

Di seguito è riportato il diagramma delle classi che include tutte le classi presentate fino a questo punto.

Oltre alle classi precedentemente descritte, è stata introdotta una classe ausiliaria: `Data`. Questa classe è stata aggiunta al fine di delineare tipi strutturati eventualmente impiegati, ad esempio, negli attributi delle classi.



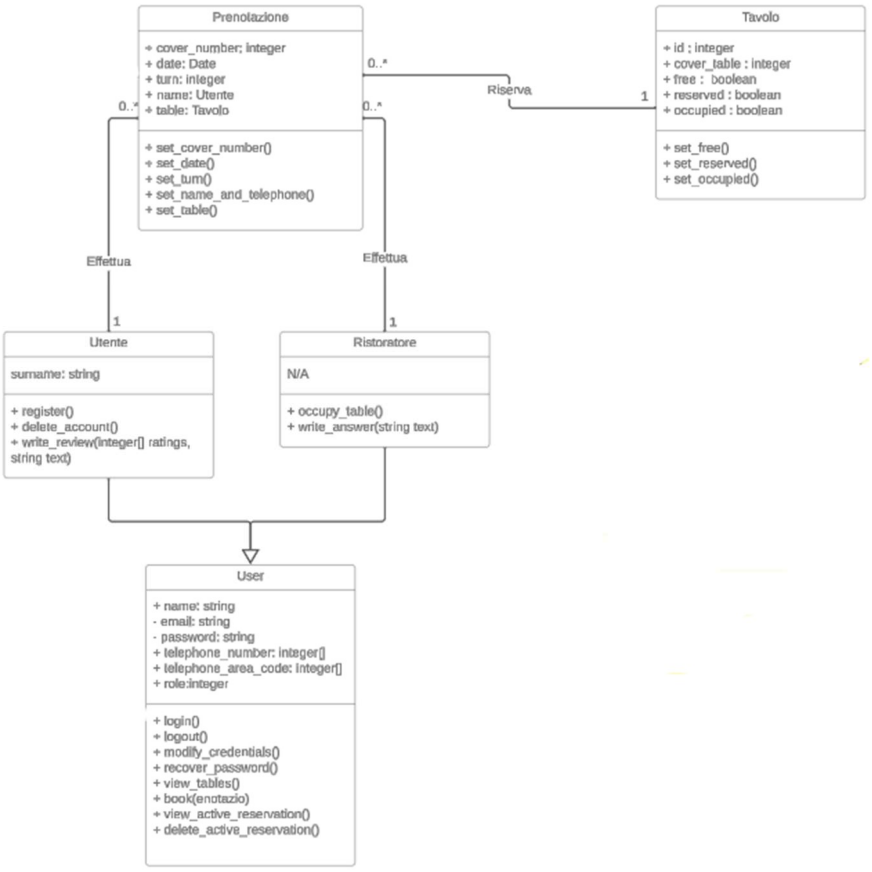
3. Codice in Object Constraint Language

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. Prenotazione

La prenotazione di un tavolo specifico può avvenire solo se il tavolo in questione è libero rispetto alla data e al turno indicato dall’utente o dal ristoratore nel momento della prenotazione.

Questa condizione implica queste classi:



Viene espressa in OCL attraverso una precondizione con questo codice:

```
context User::book()
pre: (Tavolo.free = true) AND (Tavolo.reserved = false)
```

b. Conferma di avvenuta prenotazione

Nel caso in cui il ristoratore o un utente completassero la procedura di prenotazione di un tavolo specifico, sarà necessario aggiornare lo stato dello stesso in modo tale da poter visualizzare una pianta corretta.

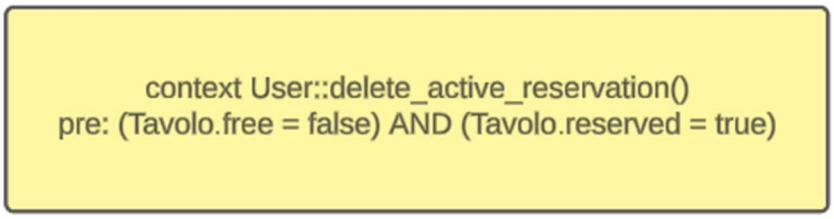
Questa condizione implica le stesse classi indicate al punto **3.a** è espressa in OCL attraverso una post condizione con questo codice:

```
context User::book()
post: (Tavolo.free = false) AND (Tavolo.reserved = true)
```

c. Passaggio allo stato “libero” di un tavolo

Il ristoratore, in caso di necessità, può decidere di cambiare lo stato di un tavolo da “prenotato” a “libero”, ma anche l’utente può decidere di cancellare una sua prenotazione attiva. Ciò può avvenire solo se il tavolo in questione è prenotato rispetto alla data e al turno indicato dal ristoratore o dall’utente nel momento della cancellazione.

Questa condizione implica le stesse classi indicate al punto **3.a** è espressa in OCL attraverso una precondizione con questo codice:



d. Conferma liberazione tavolo

Nel caso in cui il ristoratore o un utente completassero la procedura di cancellazione di una prenotazione attiva, sarà necessario aggiornare lo stato del tavolo legato alla prenotazione in modo tale da poter visualizzare una pianta corretta.

Questa condizione implica le stesse classi indicate al punto **3.a** è espressa in OCL attraverso una post condizione con questo codice:

e. Passaggio allo stato “occupato” di un tavolo

Il ristoratore, in caso di necessità, può decidere di cambiare lo stato di un tavolo da “prenotato” a “occupato”. Ciò può avvenire solo se il tavolo in questione è prenotato rispetto alla data e al turno indicato dal ristoratore nel momento della modifica.

Questa condizione implica le stesse classi indicate al punto **3.a** è espressa in OCL attraverso una precondizione con questo codice:

```
context User::occupy_table()
post: (Tavolo.free = false) AND (Tavolo.occupied = false)
AND (Tavolo.reserved = true)
```

f. Conferma occupazione tavolo

Nel caso in cui il ristoratore completasse la procedura di modifica dello stato di un tavolo, sarà necessario aggiornare la vista della pianta in maniera corretta.

Questa condizione implica le stesse classi indicate al punto **3.a** è espressa in OCL attraverso una post condizione con questo codice:

```
context User::occupy_table()
post: (Tavolo.free = false) AND (Tavolo.occupied = true) AND
(Tavolo.reserved = false)
```

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.