Baccalà

UNIVERSITÀ DI TRENTO Dipartimento di Ingegneria e Scienza dell'Informazione

Titolo del documento

Architettura del progetto

Scopo del documento

Il presente documento illustra e specifica l'architettura del sistema tramite diagrammi delle classi in UML e codice OCL.

Enrico Dalla Croce
Andrea Ervetti
Matteo Possamai

Revisione: 0.1

Indice

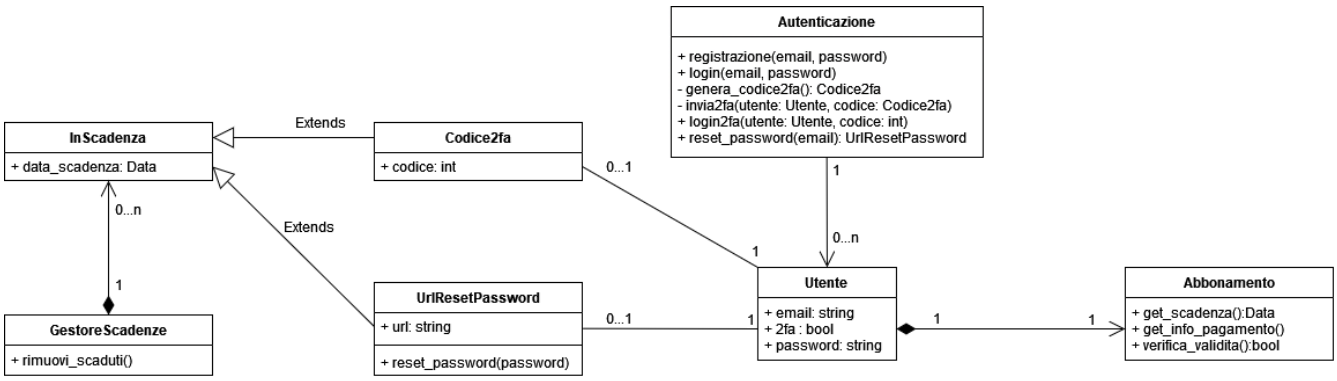
Diagramma delle classi (UML)

Di seguito viene analizzato in piccole porzioni lo schema UML del progetto.
L'obiettivo è quello di approfondire singolarmente parti distinte del software.

Precisazione: la gestione dell'interfaccia multilingua sarà affidata alla libreria i18n. Gli attributi *nome* rappresentano la chiave univoca per identificare un elemento di testo e visualizzarne la versione relativa alla lingua scelta.

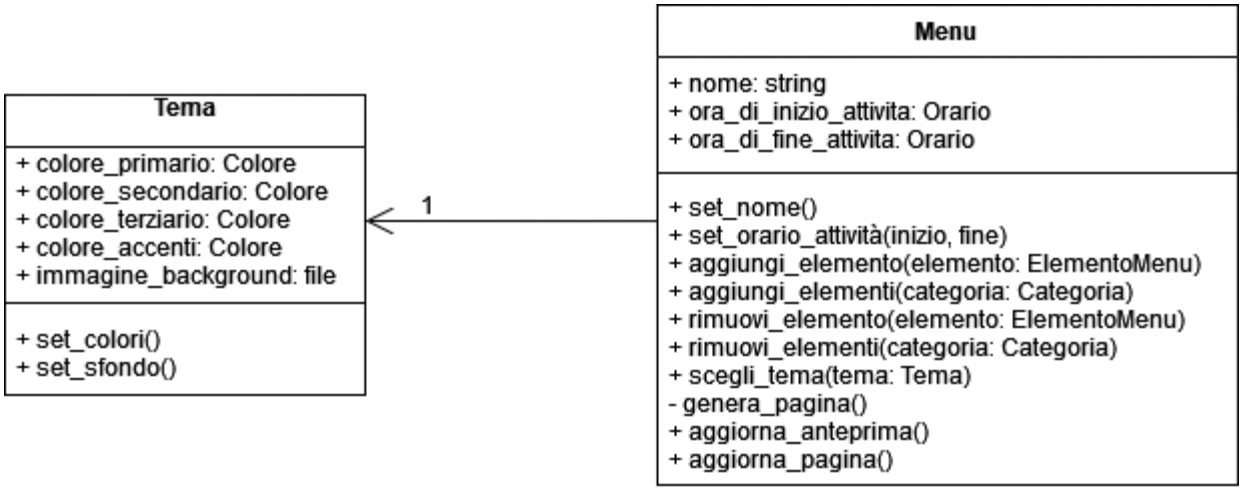
Utente

Analizzando la classe *Abbonamento* si può notare che il nostro software si appoggia a sistemi esterni per gestire il pagamento. Infatti non salviamo mai dati bancari e non ci occupiamo dell'elaborazione delle transazioni. Avendo deciso di appoggiarci al servizio di Stripe Il software deve occuparsi solo di redirigere l'utente alla pagina di pagamento e contattare le API per conoscere lo stato dei pagamenti e mostrare all'utente informazioni utili come la data di scadenza del suo abbonamento.
Riguardo l'accesso dell'utente al sistema abbiamo preso la decisione di permettere l'autenticazione a due fattori e il reset della password in modo autonomo. Questa scelta ci obbliga a dover gestire link e codici temporanei. Abbiamo pensato di creare un servizio che si occupi di gestire tutti gli elementi che possono scadere eliminandoli quando necessario.



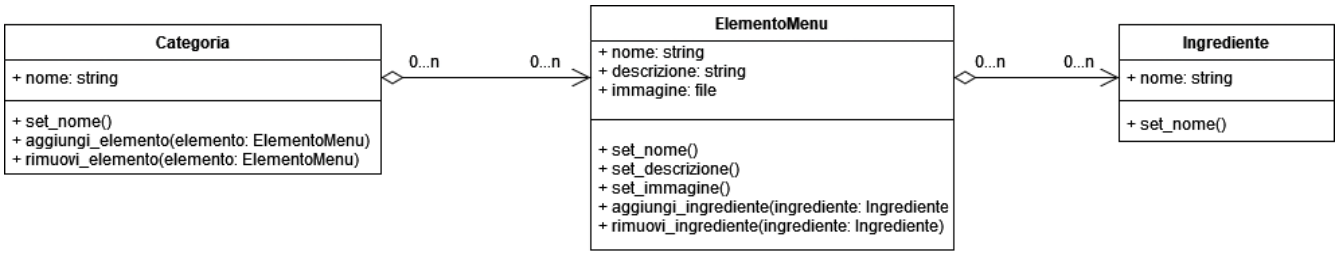
Menu

La classe *Tema* ha lo scopo di permettere all'utente di personalizzare l'aspetto del proprio menù modificandone i colori e lo sfondo ma non il layout.
La classe *Menu* come dice il nome stesso ha lo scopo di gestire le operazioni sul menu al quale si vogliono apportare dei cambiamenti.



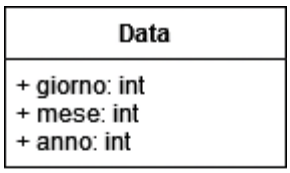
ElementoMenu

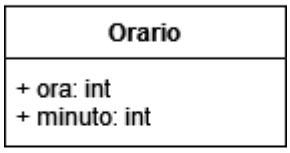
La classe *ElementoMenu* rappresenta gli elementi del menu con tutte le loro informazioni.
È importante notare la scelta di creare classi separate per gli elementi, gli ingredienti e le categorie. Questo perché un utente potrebbe avere gli stessi elementi in menù differenti. Così facendo si risparmia spazio e si evita all'utente l'inutile operazione di reinserire lo stesso elemento o gli stessi ingredienti che invece può riutilizzare.
Le categorie sono elementi separati per i motivi sopra riportati ed anche perchè fungono da filtri per suddividere le portate in sezioni diverse del menù e/o facilitare la ricerca di piatti agli utenti con particolari necessità: per esempio un cliente vegetariano potrebbe voler visualizzare solo piatti categorizzati come vegetariani.

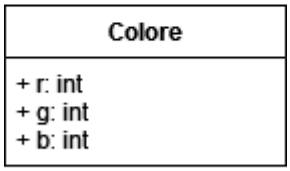


Tipi personalizzati

Per gestire alcuni tipi di informazione è stato necessario creare dei nuovi tipi attraverso delle Classi molto semplici e auto esplicative.
I tipi non primitivi che abbiamo creato sono: Data, Orario, Colore e Soldi.







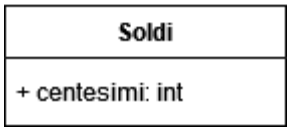
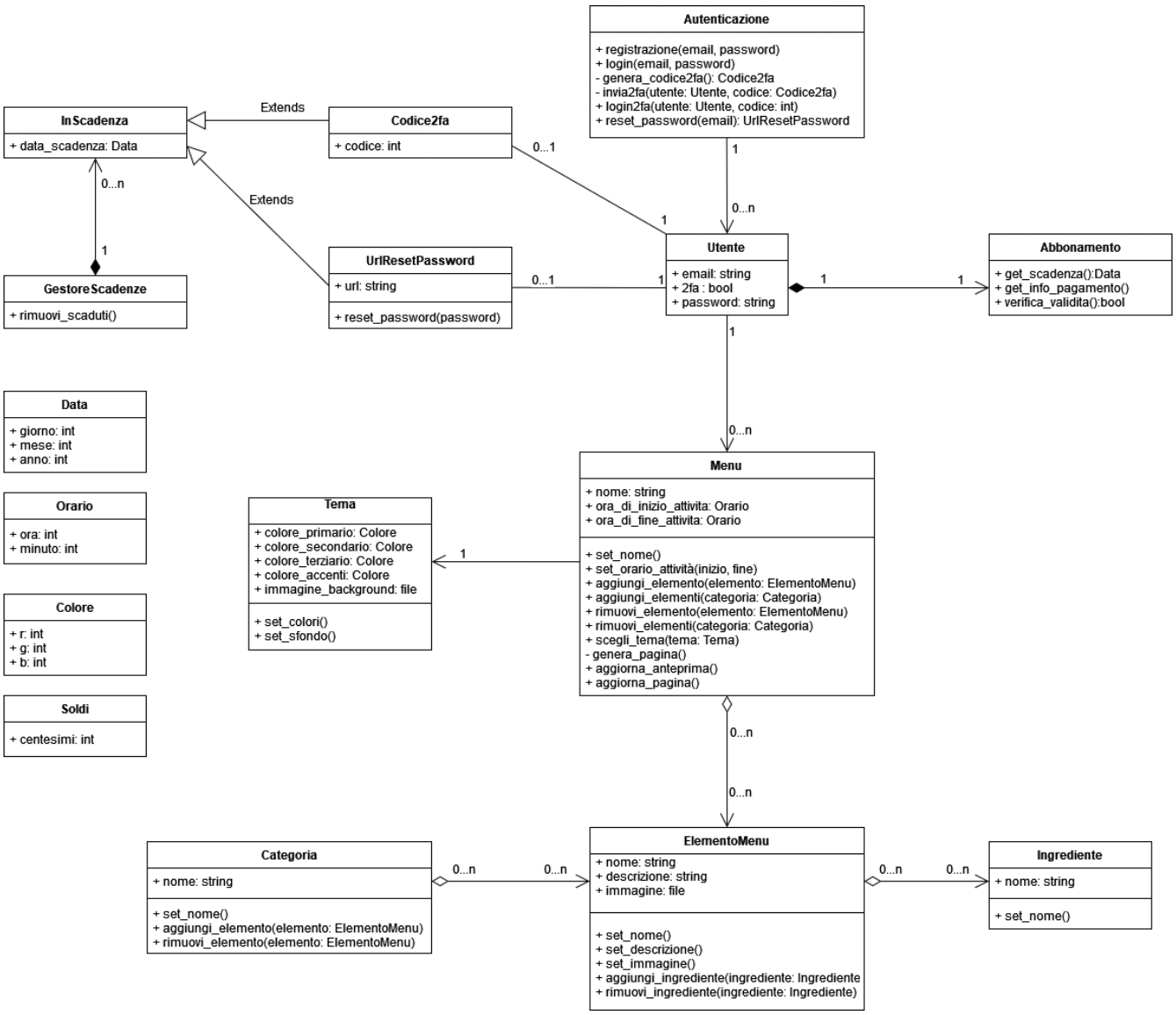


Diagramma Completo

Questo è il diagramma complessivo del nostro applicativo dove si possono vedere tutte le cardinalità e le connessioni tra gli elementi.



Object Constraint Language (OCL)

Di seguito vengono analizzate le logiche degli elementi non esprimibili attraverso lo schema UML.

Orario di attività del menù

Il seguente obbligo è necessario per evitare casi particolari che l'utente potrebbe creare.
In assenza non è possibile identificare univocamente l'orario di inizio attività di un menù con tutti i problemi che ne conseguono.

```
context Menu inv:
ora_di_inizio_attivita < ora_di_fine_attivita
```

Invio del codice per l'autenticazione a due fattori (2FA)

Prima di inviare e generare il codice per l'autenticazione a due fattori è bene controllare che l'utente abbia attivato tale funzione. Così facendo si evita di sprecare risorse inutilmente.

```
context Autenticazione::invia2fa(utente: Utente, codice: Codice2fa) pre:
utente.2fa = true
```

Login verificando la validità del codice inserito (2FA)

Un utente che non ha attivato l'autenticazione a due fattori non dovrebbe poter inserire un codice per completare l'accesso.
Lasciare attivo questo servizio anche se inutilizzabile singolarmente non è una buona pratica. Rappresenta infatti una superficie di attacco.
Un malintenzionato potrebbe sfruttare questa svista per tentare di sovraccaricare l'infrastruttura rendendola inutilizzabile o analizzare i protocolli di sicurezza implementati fino a trovare qualche falla.

```
context Autenticazione::login2fa(utente: Utente, codice: Codice2fa) pre:
utente.2fa = true
```

Diagramma delle classi (UML) completato con codice OCL

Il seguente diagramma rappresenta la struttura completa del nostro applicativo integrata con il codice OCL.
Questo è lo schema integrale, utilizzabile come documentazione, su cui si basa l'implementazione