INGEGNERIA DEL SOFTWARE

SuperF®igo

Documento di Architettura

Isacco Cenacchi, Enrico Dal Bianco, Pietro De Angeli

Università di Trento

INDICE DEL DOCUMENTO

1. SCOPO DEL DOCUMENTO

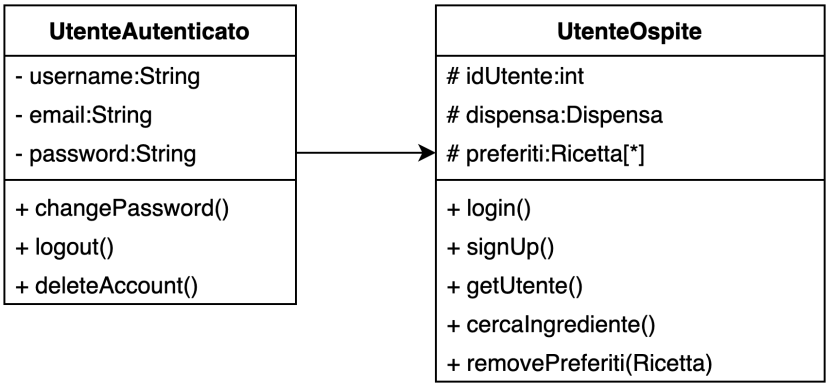
Il presente documento riporta la definizione dell'architettura del progetto SuperF®igo usando diagrammi delle classi in Unified Modeling Language (UML) e codice in Object Constraint Language (OCL). Nel precedente documento è stato presentato il diagramma degli use case, il diagramma di contesto e quello 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.

2. DIAGRAMMA DELLE CLASSI

In questo capitolo vengono riportate le classi previste all'interno del progetto SuperF®igo. Le classi sono state individuate attraverso lo studio dei requisiti funzionali del D1 e dei componenti del diagramma dei componenti del D2. Ogni classe viene caratterizzata da un nome, una lista di attributi (che identificano i dati gestiti dalla classe) e una lista di metodi (che descrivono tutte le operazioni previste all'interno della classe). Le classi possono essere associate ad altre classi, per questo motivo all'interno del diagramma saranno presenti cardinalità e altre informazioni su come le classi interagiscono tra loro. Riportiamo di seguito le classi individuate a partire dai diagrammi di contesto e dei componenti.

2.1 Utenti

Osservando il diagramma di contesto, si nota la presenza di due attori: utente ospite e utente autenticato, che eredita tutte le funzionalità dell'utente ospite ma viene registrato all'interno del database tramite la sua mail e password per evitare di perdere i dati al cambio browser o cancellamento dei cookie. Per ognuno di questi due attori introduciamo una classe.

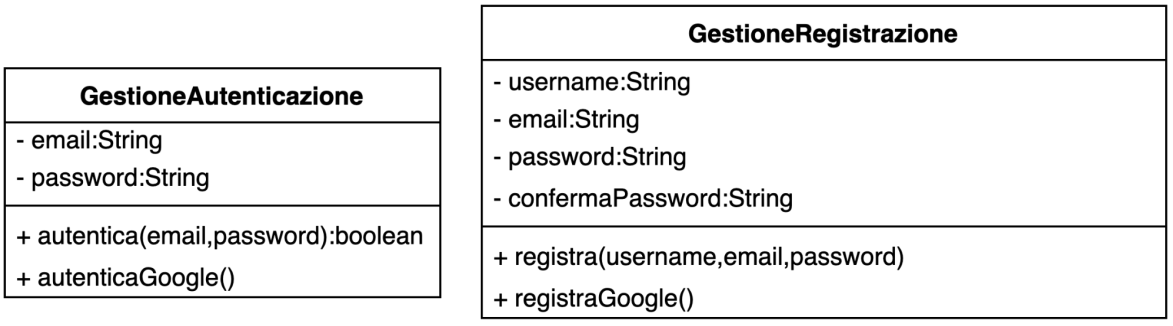


La classe UtenteOspite rappresenta un qualsiasi utente non autenticato che usufruisce del sito. Come si osserva dai metodi, l'UtenteOspite può già usufruire di tutte le funzionalità del sito.

La classe UtenteAutenticato indica un qualsiasi utente dopo aver eseguito il login. In questa classe sono state aggiunte le operazioni come il logout, il cambio password e l'eliminazione dell'account.

2.2 Autenticazione e Registrazione

Il diagramma dei componenti include diversi componenti per la gestione degli account a partire dalla creazione di un nuovo account, passando dall'autenticazione e permettendo la gestione del proprio account. Definiamo quindi le classi GestioneRegistrazione e GestoreAutenticazione.

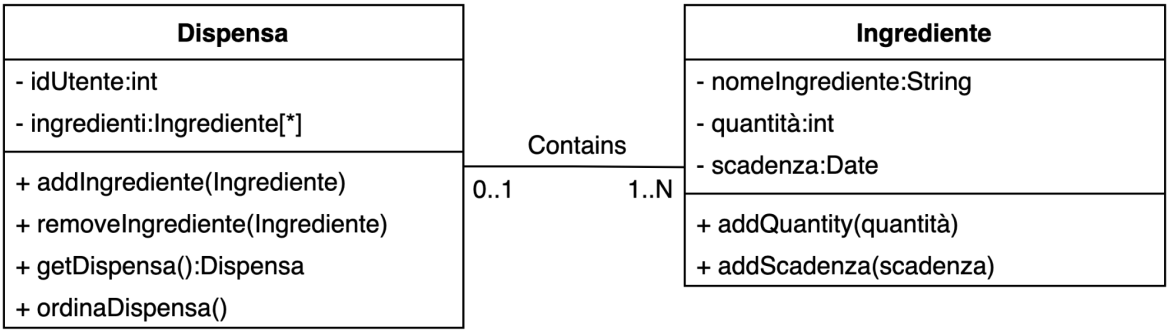


La classe GestioneRegistrazione viene utilizzata unicamente dall'utente non ancora registrato, permettendo poi all'utente di autenticarsi, con la possibilità di registrarsi attraverso le API di Google.

La classe GestioneAutenticazione ha il compito di verificare password e email di un utente già iscritto a SuperF®igo o, alternativamente, gestisce l'accesso con Google se esso è stato scelto come metodo di registrazione.

2.3 Dispensa e Ingredienti

Analizzando il diagramma dei componenti realizzato per il progetto SuperF®igo, si nota
la presenza di una dispensa contenente gli ingredienti che un utente ha in casa, si è perciò provveduto a creare due classi:

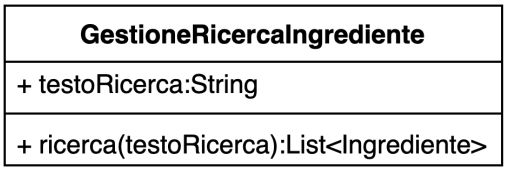


La classe Dispensa si occupa di tenere traccia di tutti gli ingredienti che ogni utente ha, e con i suoi metodi permette di modificarla aggiungendo o rimuovendo ingredienti. É possibile anche ordinare la dispensa in ordine alfabetico o per data di scadenza attraverso il metodo ordinaDispensa(). È presente una e una sola Dispensa per ogni Utente.

La classe Ingredienti rappresenta gli ingredienti che sono all'interno del database che vengono aggiunti dall'utente alla propria dispensa, e ai quali si deve dichiarare una quantità e la data di scadenza.

2.4 Ricerca Ingredienti

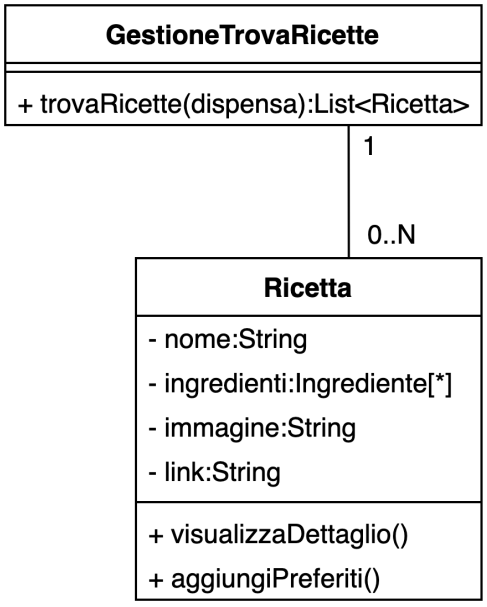
Per il componente GestioneIngredienti, del diagramma dei componenti, abbiamo sentito la necessità di creare una classe GestioneRicercaIngrediente per meglio definire come avviene l'aggiunta di un nuovo ingrediente alla dispensa.



L'utente ha infatti la possibilità di effettuare una ricerca tramite barra di testo di un ingrediente per poi poterlo aggiungere.
Questa classe ha proprio il compito di mostrare i vari possibili ingredienti presenti nel database. Infatti, a partire da una stringa inserita dall'utente si produce come risultato una lista di ingredienti affini.

2.5 Trova Ricette

Il diagramma dei componenti identifica un componente GestioneRicette di cui si è creata una classe, più una classe di supporto Ricetta.

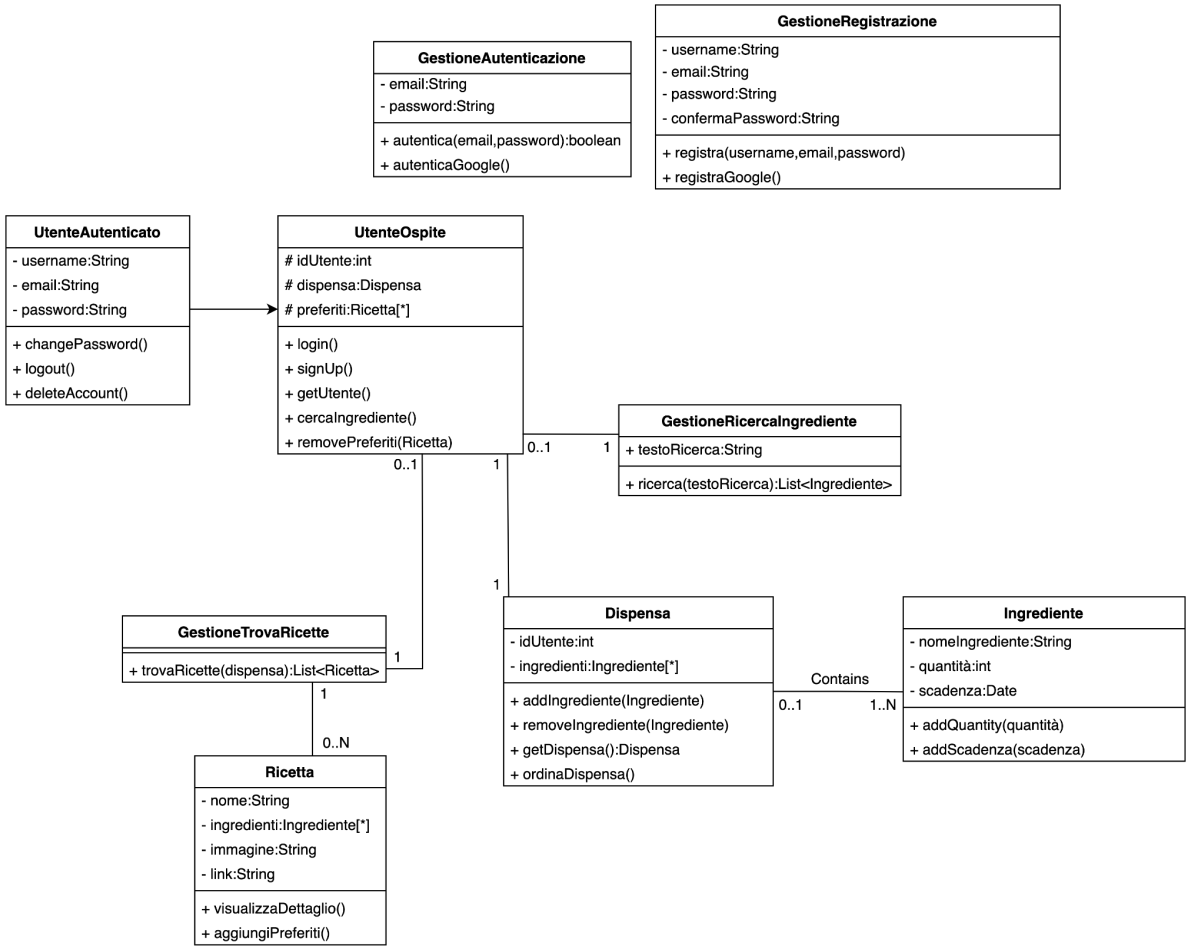


La classe GestioneTrovaRicette serve per mostrare all'utente le 10 possibili ricette (presenti nel database) che possono essere cucinate con gli ingredienti che l'utente ha nella propria dispensa. Le 10 ricette saranno ordinate per minor numero di ingredienti mancanti per completare la ricetta; se ci sono più ricette che l'utente può fare senza che gli manchino ingredienti, esse saranno ordinate in ordine alfabetico.

La classe Ricetta rappresenta le ricette all'interno del database, le quali possiedono oltre al nome, ingredienti necessari e anche un'immagine (presa da GialloZafferano) e il link che rimanda al sito della ricetta di GialloZafferano. È possibile inoltre aggiungere ricette nei preferiti.

2.6 Diagramma delle classi complessivo

Riportiamo di seguito il diagramma delle classi con tutte le classi fino ad ora presentate.



3. CODICE IN OBJECT CONSTRAINT LANGUAGE

In questo capitolo è descritta in modo formale la logica prevista nell'ambito di alcune operazioni di alcune classi. Tale logica viene descritta in Object Constraint Language (OCL) perché tali concetti non sono esprimibili in nessun altro modo formale nel contesto di UML.

3.1 UtenteAutenticato, Registrazione e Autenticazione

Per la classe UtenteAutenticato abbiamo inserito delle invarianti per la password, username e email. Ciò è espresso dal seguente codice OCL da un invariante:

```
context UtenteAutenticato
inv:  self.username.size() > 0 and
      self.password.size() > 8 and
      self.email.matches('^[A-Za-z0-9+_.-]+@(.+)$')
```

Questa invariante serve per assicurarsi che l'username non sia vuoto, la password sia minimo di 8 caratteri e che la mail sia in un formato valido.

Che si tramutano in ulteriori condizioni per le classi GestioneRegistrazione e GestioneAutenticazione. Ciò è espresso dal seguente codice OCL da queste pre e post condizioni:

```
context GestioneRegistrazione::registra(username,email,password)
pre:  self.username.size() > 0 and
      self.password.size() > 8 and
      self.password = self.confermaPassword and
      self.email.matches('^[A-Za-z0-9+_.-]+@(.+)$')
post: self.UtenteAutenticato.username = self.username and
       self.UtenteAutenticato.password = self.password and
       self.UtenteAutenticato.email = self.email
```

```
context GestioneAutenticazione::autentica(email,password)
pre:  self.email.size() > 0 and
      self.password.size() > 8
post: self.UtenteAutenticato->notEmpty()
       implies   self.UtenteAutenticato.email = self.email and
                 self.UtenteAutenticato.password = self.password
```

3.2 UtenteOspite

Nella classe UtenteOspite sono presenti i metodi signUp() e login(). L'esecuzione di questi
metodi comporta rispettivamente l'esecuzione dei metodi registra(username,email,password) presente nella classe GestioneRegistrazione, e autentica(email,password) presente della classe GestioneAutenticazione. Questo metodo è espresso in OCL tramite postcondizione con questo codice:

```
context UtenteOspite::singUp()
post: GestioneRegistrazione::registra(username,email,password)
context UtenteOspite::login()
post: GestioneAutenticazione::autentica(email,password)
```

3.3 Ricette Preferite

Nella classe Ricetta è presente un metodo aggiungiPreferiti() che serve ad aggiungere la ricetta nei preferiti dell'utente. Prima di aggiungerla deve essere verificato che la ricetta in questione non sia già presente tra i preferiti, ed in seguito all'esecuzione la ricetta dovrà essere presente all'interno dei preferiti. Questo metodo è espresso in OCl tramite precondizione e postcondizione con questo codice:

```
context Ricetta::aggiungiPreferiti()
pre: UtenteOspite.preferiti.contains(Ricetta)==false
post: UtenteOspite.preferiti.contains(Ricetta)==true
```

Nella classe UtenteOspite è poi presente presente il metodo removePreferiti(Ricetta) che serve a rimuovere la ricetta dai preferiti, e dissimile da quanto detto prima il metodo avrà queste precondizione e postcondizione:

```
context UtenteOspite::removePreferiti(Ricetta)
pre:  self.preferiti.contains(Ricetta)==true
post: self.preferiti.contains(Ricetta)==false
```

3.4 Dispensa

Nella classe Dispensa sono metodi per la sua gestione come addIngrediente(Ingrediente) e removeIngrediente(Ingrediente). Per questi metodi, come per le ricette preferite, bisogna verificare che l'ingrediente sia presente prima di rimuoverlo. In particolare poi per aggiungere un nuovo ingrediente bisogna prima ricercarlo attraverso il metodo ricerca(testoRicerca) della classe GestioneRicercaIngrediente. Questo metodo è espresso in OCl tramite precondizione e postcondizione con questo codice:

```
context Dispensa::addIngrediente(Ingrediente)
pre: GestioneRicercaIngrediente::ricerca(testoRicerca)
post: self.ingredienti.contains(Ingrediente)==true
context Dispensa::removeIngrediente(Ingrediente)
pre: self.ingredienti.contains(Ingrediente)==true
post: self.ingredienti.contains(Ingrediente)==false
```

3.5 Ricerca Ingrediente

Nella classe GestioneRicercaIngrediente è necessario verificare che il testo della ricerca non sia un campo vuoto, per evitare di fare ricerche senza un input. Ciò è espresso dal seguente codice OCL da un invariante

```
context GestioneRicercaIngrediente
inv: self.testoRicerca.size() > 0
```

3.6 Trova Ricette

La funzione Trova Ricette, definita dalla classe GestioneTrovaRicette, deve trovare le ricette solo se la dispensa ha almeno un ingrediente, per evitare di fare ricerche senza input. Ciò è espresso dal seguente codice OCL da un invariante:

```
context GestioneTrovaRicette
inv: UtenteOspite.dispensa.size() > 0
```

3.7 Ingrediente

Per la classe Ingrediente sono state definite delle invarianti per verificare che la quantità aggiunta non sia < 0, che non avrebbe senso, e che la data di scadenza deve essere maggiore o uguale della data odierna. Ciò è espresso dal seguente codice OCL da un invariante:

```
context Ingrediente
inv:  self.quantita > 0 and
      self.scadenza >= Date.today()
```

4. Diagramma delle classi con OCL