UNIVERSITÀ DI TRENTO

Dipartimento di Ingegneria e Sceienza dell'Informazione

Sleep Code

Progetto per il Corso di Ingegneria del Software
Anno Accademico 2023-2024

Documento di Architettura

Descrizione: descrizione dell'architettura di sistema sotto forma di diagrammi delle classi e codice OCL.

Numero documento: D3 *Versione documento:* 1.0

Membri del gruppo:
Raffaele Castagna
Alberto Rovesti
Zeno Saletti

Numero gruppo: G17

Ultima revisione: 1 dicembre 2023

Indice

Scopo del documento

In questo documento viene riportata la definizione dell'architettura del progetto *SleepCode* impiegando diagrammi delle classi, realizzati secondo gli standard di Unified Modeling Language (UML), e codice Object Constraint Language (OCL). Nel documento precedente (D2, *Specifica dei Requisiti*) sono stati presentati il diagramma degli use case, quello di contesto e infine il diagramma dei componenti. Considerando tale progettazione, viene ora definita l'architettura del sistema specificando in modo più dettagliato le classi che dovranno essere implementate sotto forma di codice, insieme alla logica che regola il comportamento del software che si intende realizzare.

Il linguaggio UML, utilizzato per descrivere le classi, è supportato da codice OCL, impiegato invece per specificare gli aspetti logici citati sopra che non sarebbero esprimibili formalmente mediante i soli diagrammi delle classi.

1 Diagramma delle classi

Nella presente sezione vengono illustrate in linguaggio UML le classi previste dal progetto *SleepCode*. Ogni componente del diagramma dei componenti, presente nel documento D2, viene qui rappresentato in forma di una o più classi. Le classi individuate sono costituite da un nome, un insieme di attributi che identificano i dati gestiti dalla classe stessa e una lista di metodi che definiscono le operazioni eseguibili da quella classe. Eventuali relazioni tra classi sono evidenziate da alcune associazioni.

1.1 Utenti

L'attore che nel diagramma di contesto usufruisce delle funzionalità offerte dal servizio è l'utente. Come descritto nei documenti di Analisi dei Requisiti (D1) e Specifica dei Requisiti (D2), esistono tre categorie di utenti: anonimo, autenticato e amministratore.

Dal momento che tutte queste categorie condividono alcune funzionalità, è stata creata una classe `Utente`, che logicamente corrisponde alla categoria degli utenti anonimi, ovvero utenti che non hanno effettuato il login. Di fatto, non è prevista la memorizzazione di alcuna informazione riguardo a questa categoria di utenti (non sono previsti attributi), e le uniche operazioni disponibili sono:

- `login()`: richiesta di accesso.
- `signUp()`: richiesta di registrazione.
- `recuperaAccount()`: richiesta di avvio della procedura di recupero account.

Da questa prima classe viene definita, mediante generalizzazione, la classe `UtenteAutenticato`, che aggiunge metodi e attributi accessibili previo login:

- `user_id`: dal momento che email e password dell'account di sistema possono essere modificate dall'utente autenticato, la classe contiene questo attributo con ruolo di identificatore univoco.
- `username, email, password`: i dati richiesti per creare un account. In caso di autenticazione con Google, è richiesto che la classe memorizzi solo l'indirizzo email impiegato.
- `isGoogle`: parametro che distingue gli account collegati con Google da account registrati con il meccanismo di credenziali interne.
- `risolti[*]`: insieme che raccoglie gli identificatori dei problemi risolti dall'utente.

- `preferiti[*]`: insieme degli identificatori dei problemi contrassegnati come preferiti, da parte dell'utente.
- `logout()`: richiesta di logout.
- `setEmail, setPassword`: procedure utili a modificare le credenziali dell'account. Oltre alle nuove credenziali, viene richiesta, per questioni di sicurezza, la password attualmente in uso. Entrambi i metodi restituiscono una risposta di tipo booleana per segnalare il successo o l'insuccesso delle rispettive operazioni.
- `setPreferito`: questo metodo provvede ad aggiungere o eliminare dalla lista dei preferiti dell'utente il problema fornito: se presente, il suo identificatore viene rimosso dalla lista, altrimenti viene aggiunto.
- `getPreferiti`: affinché l'utente possa vedere i preferiti della sua lista, questo metodo permette di restituire questo dato.
- `getNumeroRisolti, getRisoltiPerDifficolta`: questi metodi permettono di fornire informazioni riguardo ai progressi dell'utente.

Infine, un'ulteriore generalizzazione specifica le funzionalità aggiuntive dell'utente amministratore, ovvero metodi utili alla gestione dei problemi disponibili nel catalogo:

- `aggiungiProblema`
- `modificaProblema`
- `eliminaProblema`

Nella Figura 1 sono mostrate in linguaggio UML le classi sopra descritte.

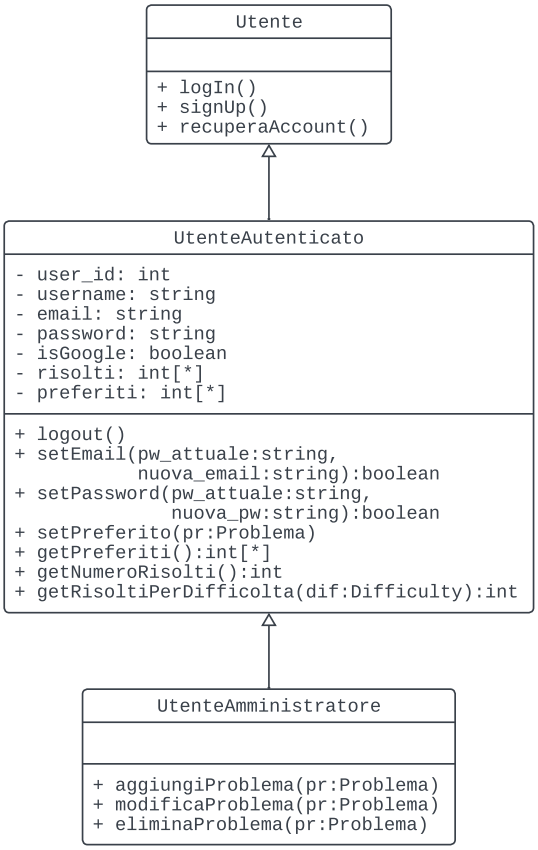

Figura 1: Definizione della classe `Utente`

1.2 Gestione autenticazione

Nel diagramma di contesto, l'autenticazione si avvale di due sistemi subordinati: *Firebase DB*, che si occupa di memorizzare gli account registrati con sistema di autenticazione interno, e *Google SignIn*, con il quale è possibile, per un utente, effettuare la registrazione o l'autenticazione mediante un account Google.

Il componente *Gestione Autenticazione* del rispettivo diagramma si occupa di interagire con tali sistemi esterni in modo da convalidare le operazioni di registrazione e autenticazione. Per questo motivo, nel diagramma delle classi viene individuata la classe `Autenticazione`:

- `autorizzato`: questo attributo mantiene lo stato dell'autorizzazione dell'utente
- `isGoogle`:
- `registraAccount`: questo metodo esegue la registrazione comunicando con il servizio di database.
- `autentica`: l'autenticazione mediante credenziali interne avviene grazie a questo metodo.
- `registraGoogle`
- `autenticaGoogle`

Tutti i metodi comunicano all'esterno l'esito delle loro operazioni mediante un valore booleano di ritorno.

2 Codice in Object Constraint Language

Nella sezione che segue viene descritta formalmente la logica prevista nel comportamento di alcune classi, in relazione alle loro operazioni possibili. Il codice OCL impiegato consente di esprimere tale logica, non descrivibile con i soli diagrammi delle classi in UML.

3 Diagramma delle classi con codice OCL

In Figura 2 è riportato un diagramma che mostra nell'insieme sia le classi individuate che il relativo codice OCL.

Figura 2: Diagramma delle classi arricchito dal codice OCL