**Trillium**

Documento di architettura - D3

## **Indice**

[**Indice 1**](#_68zc6of3li4d)

[**Scopo del documento 1**](#_n1fn5knb2cje)

[**Diagramma delle classi 2**](#_l63nwp7xsfwy)

[1. Utenti e Opzioni 2](#_700bxthpsroy)

[2. Gestione dei post e Ricerca 2](#_h33cp39p7js8)

[3. Registrazione ed Autenticazione 3](#_ky9hkhqqsnih)

[**Codice in Object Constraint Language 4**](#_ixrolimn2o9r)

[1. Utenti e Opzioni 4](#_tbndgmyj297s)

[2. Gestione dei post e ricerca 6](#_d7rw7jvhrw5l)

[Post 6](#_lpgxl91j4gu4)

[PostPreferences 6](#_yj2xzn20e3f0)

[PostComment 7](#_iytcdmqlzi07)

[SearchPage 8](#_gv5ddzsigln6)

[**Diagramma delle classi con codice OCL 9**](#_v7e93qwf8ar2)

## 

## **Scopo del documento**

Il presente documento riporta l’analisi dell’architettura del progetto “**Trillium**” tramite i diagrammi delle classi in **UML** (Unified Model Language) e codice **OCL** (Object Constraint Language). L’analisi prende in considerazione il documento precedente, contenente la definizione dei componenti, diagramma use-case e quello di contesto. Le classi verranno rappresentate tramite UML, mentre la logica e tutti i concetti che ne conseguono saranno rappresentati tramite OCL.

## **Diagramma delle classi**

Nella presente sezione verranno analizzate le classi.

Ogni componente del documento precedente verrà utilizzato per formare una o più classi ed ogni classe sarà rappresentata da un nome, degli attributi (dati gestiti dalla classe) e dei metodi, inoltre ogni classe potrà essere relazionata con altre classi.

Il diagramma delle classi può essere suddiviso in tre componenti princiali:

### **Utenti e Opzioni**

La classe **User** contiene tutte le informazioni che un utente può inserire nella piattaforma più altre informazioni utili per la gestione dello stesso, ad esempio le date di creazione, dell’ultima modifica e dell’eliminazione dell’account.

Inoltre **User** contiene una lista di amici e di altri utenti che segue e una lista dei propri post e commenti per facilitarne la gestione.

La classe **UserPage** permette di creare uno **User** inserendo tutte le informazioni necessarie oppure di modificarne uno già esistente.

Tale classe non memorizza permanentemente i dati sensibili inseriti dall’utente ma funge solo da interfaccia alla classe **User**.

Un’ulteriore classe **UserSettings** è stata aggiunta per gestire le varie opzioni dell’account dell’utente e per facilitarne l’ampliamento.

### **Gestione dei post e Ricerca**

La classe **Post** contiene tutte le informazioni presenti in un post creato da un utente.

In **Post** vengono memorizzate la data di creazione, dell’ultima modifica e della cancellazione se avvenuta e deve essere salvato anche un ID che permetta di identificare i post tra di loro.

Come per lo **User** è stata creata una classe **PostPreferences** per gestire le opzioni del post come la loro visibilità.

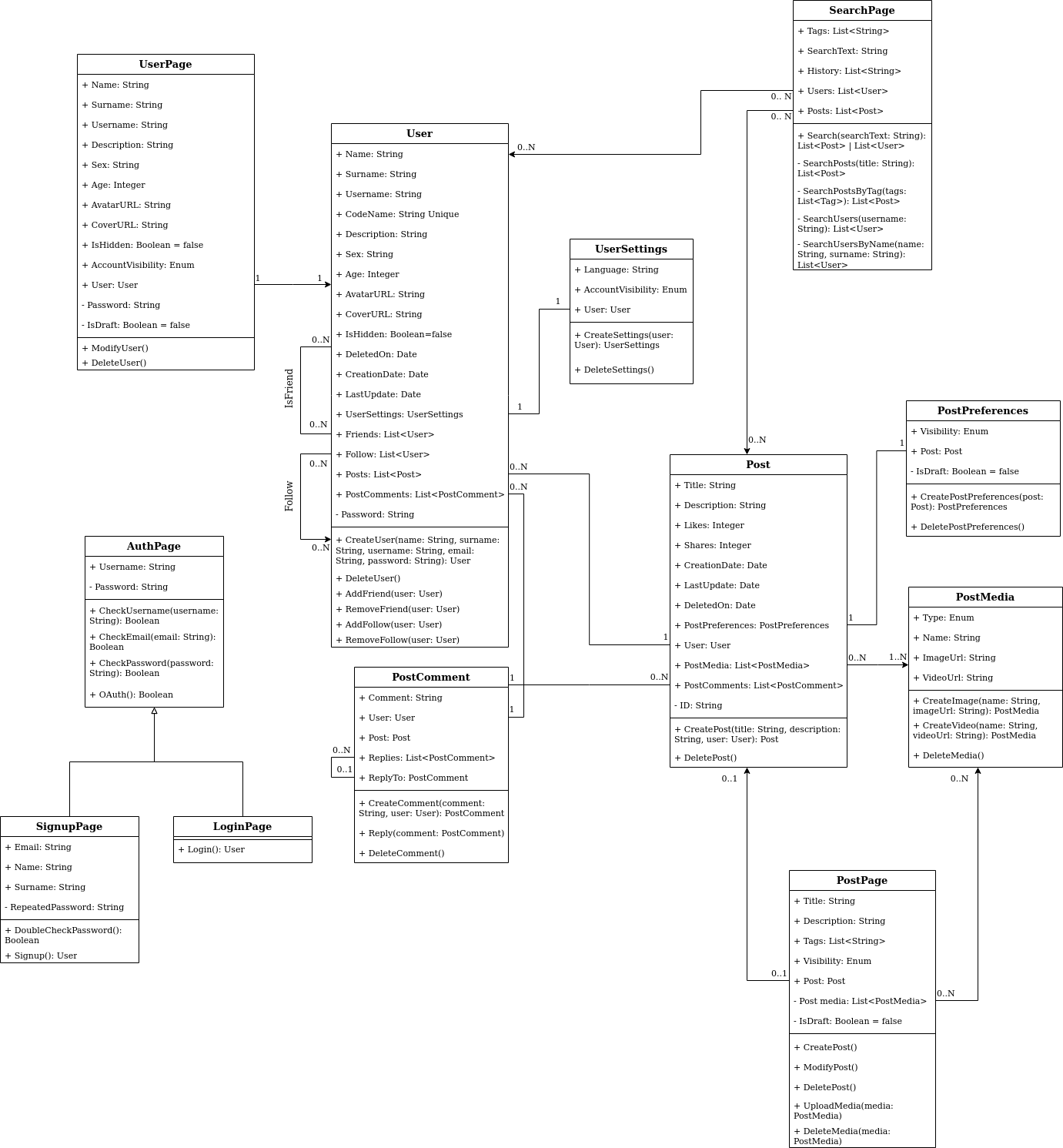
La classe **PostMedia** serve a gestire tutti i media, come immagini, video, ecc…, che vengono aggiunti al post, similmente è stata creata **PostComment** per gestire tutti i commenti che gli utenti possono aggiungere al post.

La classe **PostPage** permette di creare, modificare ed eliminare un post inserendo tutte le informazioni necessarie.

Infine è stata introdotta la classe **SearchPage** che permette ad un utente di cercare dei post tramite il loro titolo, eventualmente filtrandoli per tag, oppure trovare utenti tramite il loro usename o nome e cognome

### **Registrazione ed Autenticazione**

La registrazione di un account alla piattaforma avviene tramite la classe **SignupPage** mentre l’accesso all’account tramite **LoginPage**, entrambe le pagine sono estensioni della classe **AuthPage** che contiene tutte le operazioni che permettono la verifica dei dati e l’autenticazione.



## **Codice in Object Constraint Language**

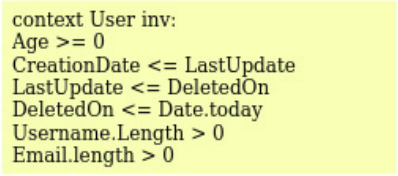
In questa sezione verrà illustrata in modo formale la logica delle classi. Tale logica verrà descritta tramite OCL.

### **Utenti e Opzioni**

Nella classe **User** l’anno dell’utente non può essere un numero negativo e lo username e l’email non possono essere vuoti.

Inoltre la data di creazione dell’utente deve essere minore o uguale a quella dell’ultima modifica che a sua volta dev’essere minore o uguale a quella della cancellazione.

Questi concetti sono espressi tramite il seguente codice OCL:



Ad ogni aggiunta o rimozione di un’amicizia o di un “*follow*” deve essere rimosso il corrispettivo amico (*seguito*) nella lista **Friends** o **Follow** rispettivamente.

Espresso in OCL con questo codice:



Nella classe **UserPage** l’età non può essere negativa.

Rappresentata tramite in una variante nel seguente codice OCL:

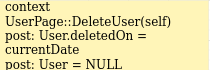


Inoltre, se viene chiamato il metodo **ModifyUser(newUserSettings)** la variabile **IsDraft** deve essere settata a *false* una volta terminata la modifica:



Se l’utente viene cancellato deve essere aggiornata la deletedOn (data rimozione profilo) e rimosso il riferimento a **User**.

Espresso tramite il seguente codice OCL:



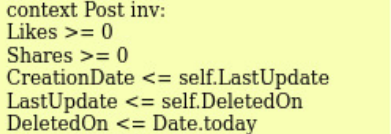
### **Gestione dei post e ricerca**

### Post

Nella classe **Post**, il numero di like e shares non può essere negativo.

Le date devono essere valide e coerenti: la **creationDate** non puo’ essere posteriore al **lastUpdate**, **lastUpdate** non puo’ essere poteriore a **deletedOn** (data eliminazione) e **deletedOn** deve essere precedente o uguale alla **currentDate**.

I vincoli sopra illustrati sono sviluppati secondo il seguente codice OCL:



Alla creazione del post deve seguire l’aggiunta del post alla lista dell’utente. Quindi deve essere chiamata la funzione **CreatePost(self)**, con user definito e accessibile (per poter chiamare **User.Posts.Add(self)**):



Allo stesso modo deve venire utilizzata la funzione **DeletePost(self)**, che al suo interno chiamera’ **User.Posts.Remove(self)**:

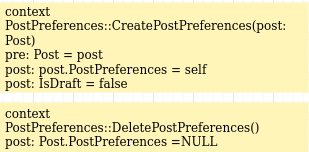


### PostPreferences

Nella classe **PostPreferences** abbiamo solamente la definizione di 2 funzioni:

1. **CreatePostPreferences()**: Chiamata ogni volta che un post viene creato. Setta il post “padre”, aggiunge il collegamento al post a se stessa e setta **isDraft** a false.
2. **DeletePostPreferences()**: elimina se stessa e toglie il collegamento nel post.

Rappresentato tramite il seguente codice OCL:



### PostComment

La classe PostComment introduce delle funzioni per creare, cancellare e rispondere ad un commento.

Rispettivamente: **CreateComment(params)**, **DeleteComment()** e **Reply()**.

**CreateComment(params)** richiede che sia definito un User, colui che pubblica il commento, e un Post, per collegare il commento al Post.

Rappresentazione OCL:



**DeleteComment(params)** richiede che l’utente che vuole eliminare il commento sia lo stesso che lo ha pubblicato precedentemente. Una volta validata questa condizione il commento deve essere rimosso dal post in questione e poi eliminato.

Rappresentazione OCL:



### SearchPage

La classe SearchPage introduce la funzione per ricercare post caricati sulla piattaforma.

La funzione **Search(searchText: String)** restituisce i post che corrispondono al testo cercato e successivamente salva nella history la stringa cercata (per essere facilmente ricercabile nel caso l’utente voglia cercare lo stesso testo).

Il codice OCL e’ il seguente:



## **Diagramma delle classi con codice OCL**

Questa sezione contiene il diagramma **completo** delle classi individuate fino ad ora con in aggiunta il codice OCL.

