Pràctica 2: Gestió de la Xarxa Social iTICApp

Tecnologia de la Programació — iTIC

Marta I. Tarrés-Puertas

February 5, 2021

Contents

1 Organització

1.1 Objectius

L'objectiu d'aquesta pràctica és prendre contacte amb els programes estructurats en base a classes d'objectes. El context del projecte se sitúa en una xarxa social, en el marc d'un projecte de recerca sobre els seus usuaris i els posts que publiquen a la xarxa.

1.2 Condicions

- La pràctica està calibrada per a ésser treballada en equip.
- El model de desenvolupament que es demana que utilitzeu és test driven programming. Si no en recordeu els detalls del curs passat, pregunteu!

1.3 Lliurables

- Caldrà lliurar el codi resultant del projecte i una documentació escrita amb `Sphinx` que ha d'incloure també una taula de dedicacions de cada persona del grup a les diferents tasques de la pràctica.
- La durada de la pràctica és de 2 setmanes.

2 Introducció

La nostra empresa està treballant en una innovadora aplicació mòbil tipus xarxa social on els usuaris poden publicar missatges de text, imatges i vídeos. En el futur es pretén ampliar els tipus de publicacions (posts). Aquests posts són públics i es poden visualitzar sense estar autenticat al sistema, però només els usuaris autenticats podran publicar.

Es vol que els usuaris es puguin registrar fàcilment només proporcionant el seu correu i paraula clau. També hauran de seleccionar un identificador únic al sistema.

Per començar en la implementació, se us demana el desenvolupament simplificat d'aquesta que permeti la gestió d'usuaris i publicacions (amb hashtags).

Cal destacar és que aquesta pràctica és un exercici acadèmic, l'objectiu del qual és el disseny d'una aplicació d'una certa complexitat. En aquest sentit, cal tenir en compte que la funcionalitat descrita a l'enunciat constitueix una simplificació de la realitat, i en algun aspecte concret, pot diferir de la pràctica habitual de les operacions descrites.

2.1 Arquitectura

L'aplicació consistirà en un conjunt de classes i un programa principal. Aquestes classes es relacionen seguint el diagrama UML que segueix (veure Figura 1).

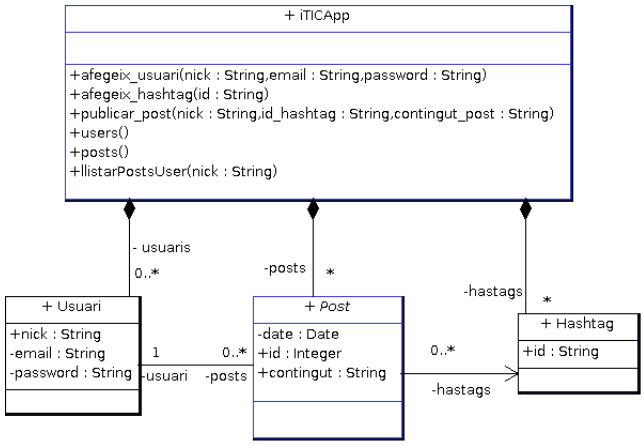

Figure 1: Arquitectura UML d'iTICApp.

Usuari Representa un usuari d'iTICApp. Un Usuari té un conjunt d'usuaris followers que el segueixen i un conjunt d'usuaris als que segueix (following).

Post Representen els Posts de l'aplicatiu. Cada Post és d'un Usuari i un Usuari pot realitzar diferents Post.

Hashtag Representa un hastag usat en els Post. Per simplificar, en aquest disseny inicial, només interessa saber els hastags d'un Post.

iTICApp Una iTICApp representa una col.lecció d'instàncies de Usuari, Post i HashTag interrelacionades. És una *Façana*.

Intèrpret Un Intèrpret representa un intèrpret de comandes simple. Usarem una instància d'aquesta classe com a interfície d'usuari entre el nostre programa i l'usuari.

Més a més, l'aplicació tindrà un mòdul `main.py` que encabirà el programa principal.

Tasca 1 Prepareu l'entorn de treball on desenvolupareu el projecte. A tal efecte:

1. Creeu un directori anomenat `xarxaSocial` i dins d'aquest un subdirectori anomenat `src` en que hi anireu escrivint els fitxers font del projecte.
2. Dins de `xarxaSocial` i usant les comandes de `Sphinx` creeu un directori anomenat `doc` en el que escriureu la documentació usant les eines que ja coneixeu.
3. Creeu un capítol de la documentació en el que començareu a anotar el temps que dediqueu individualment a cada tasca. Feu servir una taula amb un format similar al següent:

Tasca	Persona1 (h)	Persona 2 (h)	Total (h)
T1	1.20	2.10	3.30
T2	—	0.80	0.80
T3	4.13	2.00	6.13
⋮	⋮	⋮	⋮
TOTAL	5.33	4.90	10.23

3 Implementació i test

3.1 Gestió d'usuaris

El mòdul `usuari.py` conté la classe Usuari. La classe ha de tenir els següents atributs:

nick [**str**, Public] És el nick de l'usuari.

email [**str**, Private] És l'email de l'usuari.

password [**str**, Private] És el password de l'usuari.

La classe disposarà d'un constructor amb el nick, email i password com a paràmetres, així com els mètodes addicionals que siguin necessaris per al correcte funcionament de l'aplicatiu. Per simplificar, suposeu que totes les dades introduides són correctes i que no hi haurà usuaris repetits. Fixeu-vos que en el cas del password, per mesures de seguretat, s'espera que el mostreu encriptat. Podeu utilitzar alguna de les llibreries de Python a tal efecte, o bé implementar algun mecanisme d'encriptació (exemple: xifratge Cèsar).

```
if __name__=='__main__':
    p1=Usuari("john24","john24@gmail.com","abracadabra")
    p2=Usuari("johh24","john244@gmail.com","patadecabra")
    print p1
```

```
    print p2
    p3=Usuari("john24","john2444@gmail.com","supercalifra")
    print p3.nick
    p1==p3
#Resultats execució
Usuari: john24 Email: john24@gmail.com Encripted password: cdtcecfcdtc
Usuari: johh24 Email: john244@gmail.com Encripted password: rcvcfgecdtc
john24
```

Tasca 2 Implementeu el mòdul `usuari.py`: primer definiu els doctests corresponents, assegureu-vos que són complets. Després implementeu els mètodes i comproveu que funcionen correctament usant els doctests. Finalment documenteu el codi correctament i annexeu aquesta documentació a la documentació del projecte usant `Sphinx`.

3.2 Gestió de posts

El mòdul `posts.py` conté la classe Post. La classe ha de tenir els següents atributs:

id [**int**,Públic] És un identificador assignat directament pel sistema i que s'incrementa en 1 per cada post.

contingut [**str**, Públic] És el contingut del post.

date [Date, Privat] És la data en què s'ha fet el Post.

La classe disposarà dels mètodes que segueixen, i per a cadascun d'ells s'exemplifica el seu funcionament per mitjà de doctests.

1. __init__(self, contingut)

   Constructor. Crea el Post amb el contingut donat i emmagatzema la data del sistema.

Tasca 3 Implementar la classe Post i els mètodes necessaris per al seu correcte funcionament.

```
if __name__=='__main__':
    post1=Post("Cal realitzar el possible per assolir l'impossible.")
    post2=Post("Tota accio provoca reaccions.")
    print post1
    print post2
    post3=Post("Cal realitzar el possible per assolir l'impossible.")
    print post3.info
    print post1==post3
#Resultats d'execució
Post id: 1 info: Cal realitzar el possible per assolir l'impossible. Date: Tue Feb  6 12:00:21 2018
Post id: 2 info: Tota accio provoca reaccions. Date: Tue Feb  6 12:00:21 2018
Cal realitzar el possible per assolir l'impossible.
True
```

3.3 Gestió de hashtags

El mòdul `posts.py` conté la classe Post. La classe ha de tenir els següents atributs:

id [**str**, Públic] És l'identificador del hashtag.

La classe disposarà dels mètodes necessaris pel correcte funcionament de doctests.

TASCA 4 Implementar la classe Post i els mètodes necessaris per al seu correcte funcionament.

```
if __name__=='__main__':
    h1=Hashtag("adventure")
    h2=Hashtag("winter")
    print h1
    print h1==h2
#Resultats d'execució
#adventure
False
```

3.4 Registre d'usuaris, posts i hashtags

El mòdul `xarxaSocial.py` conté la classe iTICApp. La classe ha de tenir els següents atributs:

usuaris [**dict**, Privat] És un diccionari en el que la clau correspon a cadascun dels nick dels usuaris que intervenen en la xarxaSocial i el valor correspon a l'objecte Usuari corresponent.

posts [**dict**, Privat] És un diccionari en el que la clau correspon a l'info de cadascun dels post del xarxaSocial i el valor correspon a l'objecte Post corresponent.

hashtags [**dict**, Privat] És un diccionari en el que la clau correspon al id de cadascun dels hashtag de la xarxaSocial i el valor correspon a l'objecte Hashtag corresponent.

La classe iTICApp disposarà dels mètodes següents.

1. __init__(self)

   Constructor. Crea un xarxaSocial buit, sense usuaris ni posts ni hashtags.

2. afegeix_usuari(self,nick, email, password)

   Afegeix un usuari de nick n a la xarxaSocial. Si existia l'usuari amb el mateix nick es queixa.

3. afegeix_hastag(self,**id**)

   Afegeix un hashtag de id **id** a la xarxaSocial. Si existia hashtag amb el mateix id es queixa.

4. publicar_post(self,nick,id_hashtag,contingut_post)

   Comprova si el nick d'usuari existeix a l'aplicació. Si és el cas, crea el Post, guarda el nick d'usuari al Post, afegeix l'objecte Hashtag al Post, afegeix l'objecte Post a l'usuari corresponent i afegeix el Post al contenidor de posts.

5. users(self)

   Llista per pantalla la informació completa dels usuaris de l'aplicatiu, incloent la informació dels seus posts.

6. posts(self)

   Llista per pantalla la informació completa de tots els posts en ordre invers a com s'han realitzat.

7. llistarPostsUser(self,nick)

   Obté el llistat d'informació dels posts (contingut) d'un Usuari amb el nick proporcionat

   A continuació segueix uns exemples de funcionament.

```
if __name__==’__main__’:
    i=iTICApp()
    i.afegeix_usuari("pere","pere@gmail.com","gilisticoexpia")
    i.afegeix_hashtag("adventure")
    i.publicar_post("pere","adventure","into the wild")
    i.afegeix_usuari("maria","maria@gmail.com","gilisticoexpia")
    i.afegeix_hashtag("winter")
    i.afegeix_hashtag("blue")
    i.publicar_post("pere","winter","into the wild")
    i.publicar_post("pere","winter","into the wild")
    i.publicar_post("pere","february","fall again, fall better")
    i.users()
    i.posts()
    print i.llistarPostsUser("pere")

#Resultats d’execució
Usuari: pere Email: pere@gmail.com Encripted password: gilisticoexpia
 Published posts:
Post id: 1 info: into the wild Date: Tue Feb  6 15:53:45 2018
Nick user: pere Available hashtags: #adventure #winter

Post id: 2 info: fall again, fall better Date: Tue Feb  6 15:53:45 2018
Nick user: pere Available hashtags: #february

Usuari: maria Email: maria@gmail.com Encripted password: gilisticoexpia
 Published posts:  not available

Post id: 2 info: fall again, fall better Date: Tue Feb  6 15:53:45 2018
Nick user: pere Available hashtags: #february

Post id: 1 info: into the wild Date: Tue Feb  6 15:53:45 2018
Nick user: pere Available hashtags: #adventure #winter

[’into the wild’, ’fall again, fall better’]
```

TASCA 5 Implementeu la classe iTICApp amb els mètodes especificats i modifiqueu les classes Usuari i Post per tal que realitzin el comportament esperat.

En particular us caldrà,

1. A la classe Usuari:
   a) Afegir un contenidor de Posts. Utilitzeu una llista.
   b) Implementar el mètode registra_post(self,post)

      Afegeix un objecte Post a la llista de Posts d’un usuari
   c) Modifiqueu el mètode **str** per tal de que doni la informació completa dels posts d’un usuari.
2. A la classe Post,
   a) Afegir un contenidor de Hashtags. Utilitzeu una llista.
   b) Implementar el mètode registra_usuari(self,nick)

      Crea un nou atribut a Post que contindrà el nick de l’usuari

c) afegeix_hashtag(self,**id**)

Mètode que permet afegir un objecte Hashtag(**id**)

d) Modificar el mètode **str** per tal que doni informació del nick de l'usuari propietari del post i dels hashtag/hashtags del Post.

TASCA 6 Dibuixeu el nou diagramaUML resultant.

TASCA 7 Implementeu el mòdul `xarxaSocial.py` seguint la mateixa pauta que en la tasca anterior. Documenteu-lo usant `Sphinx`. Definiu l'esquelet de la classe, afegint-hi els mètodes que calguin, i afegiu els doctests corresponents. Assegureu-vos que són complets. Aneu implementant els mètodes i comproveu que passen els doctests correctament. Finalment documenteu el codi correctament i annexeu aquesta documentació a la documentació del projecte usant `Sphinx`.

3.5 Classe Intèrpret

Aquesta classe d'objectes abstreu un intèrpret d'ordres senzill. Recordeu que un intèrpret d'ordres és un programa que, de forma interactiva, llegeix ordres de l'usuari i les va executant una a una. Ja coneixeu alguns intèrprets d'ordres, en particular:

1. La shell de UNIX, que us aten en la terminal d'ordres del sistema operatiu.
2. L'intèrpret interactiu de Python, que us permet executar ordres Python interactivament i usar-lo com una calculadora.

Un objecte de la classe Interpret és un interpret d'ordres configurable. Això significa que per usar-lo primer cal configurar-lo. Configurar-lo consisteix a dir-li quines ordres ha de conèixer i què han de fer.

Una sessió de treball típica amb aquesta classe podria ser:

```
>>> def c1(l): print "executo l'ordre 1: {0}".format(l[0])
>>>
>>> def c2(l): print "executo l'ordre 2: {0}".format(l[0])
>>>
>>> i = Interpret()
>>> i.set_prompt("**")
>>> i.afegeix_ordre("llista", c1)
>>> i.afegeix_ordre("bloqueja", c2)
>>>
>>> i.run()
** llista usuaris
executo l'ordre 1: usuaris
** bloqueja pere
executo l'ordre 2: pere
** surt
>>>
```

Fixeu-vos que una instància de l'intèrpret es configura amb el mètode afegeix_ordre. Afegir una ordre implica indicar el nick de la ordre i també la funció que implementa aquesta ordre. També

haureu observat que l'intèrpret s'engega amb el mètode run i acaba quan l'usuari usa l'ordre especial `surt`.

En el mòdul `interpret.py`, la classe Interpret ha de tenir els següents atributs:

prompt [**str**, Privat] Emmagatzema el prompt que usarà l'intèrpret.

dcom [**dict**, Privat] És el diccionari que emmagatzema les ordres conegudes per l'intèrpret. El diccionari emmagatzema una entrada per a cada ordre. Per a una ordre específica, la clau correspon amb el nom de l'ordre i el valor és la funció que implementa l'ordre (vegeu el mètode afegeix_ordre).

La classe disposarà dels següents mètodes:

1. __init__(self)

   Constructor. Crea un interpret buit, sense ordres.

2. set_prompt(self,p)

   Modificador. Fixa l'string p com el prompt de l'intèrpret.

3. afegeix_ordre(self,nomc,ordre)

   Modificador. Afegeix a l'intèrpret una ordre de nom nomc associada a la funció ordre. Si ja existia una ordre amb aquest nom, es queixa. Noteu que el tercer paràmetre del mètode és una funció!

   La funció de nom ordre és una funció que té com a únic paràmetre una llista de strings.

4. run(self)

   Arrenca l'execució d'aquest intèrpret d'ordres. L'intèrpret s'executa indefinidament fins que l'usuari escriu l'ordre `surt`.

   A cada cicle d'interpretació, l'intèrpret escriu el prompt, llegeix un string del teclat, l'analitza separant els mots que el formen. El primer mot considera que és un nom d'ordre i la resta de mots els paràmetres d'aquesta ordre. Finalment executa la funció corresponent a l'ordre i li passa com a paràmetre la resta de mots en una llista.

TASCA 8 Implementeu el mòdul tot i definint els doctests corresponents, assegureu-vos que són complets i funcionen correctament. Finalment documenteu el codi correctament i annexeu aquesta documentació a la documentació del projecte usant `Sphinx`.

3.6 Mòdul main

El mòdul `main.py` conté el programa principal. Aquest programa crea una instància de XarxaSocial i una instància d'Interpret juntament amb les funcions que implementen les ordres necessàries. A continuació, engega l'intèrpret. Aquest es va comunicant interactivament amb l'usuari fins acabar la sessió.

L'intèrpret ha de tenir les següents ordres:

`usuari <nick>`

Afegeix un usuari al xarxaSocial amb nick valor`<nick>`.

`hashtag <id>`

Afegeix un hashtag a la xarxaSocial amb l'id `<id>`.

`publicar <nick> <id> <post>`

Publica el post `<post>` l'usuari amb nick `<nick>` i hashtag `<id>`.

`print <ent> [<nick>]`

Escriu per pantalla segons el valor de `<ent>`. Si `<ent>` és:

`users`

Escriu la informació completa dels usuaris de la xarxaSocial.

`posts`

Llista la informació completa dels posts en ordre invers a com s'han introduït.

`pots-user`

Escriu la llista d'informació del contingut dels posts que ha publicat l'usuari anomenat `<nick>`.

`surt`

Acaba l'execució del programa.

TASCA 9 Implementeu el mòdul principal i comproveu que funciona correctament.

3.7 Funcionalitat addicional

TASCA 10 Doteu al vostre aplicatiu d'una tasca addicional. Dissenyeu inicialment els canvis en el diagrama UML i després passeu a la implementació. Exemples de funcionalitats addicionals poden ser, per exemple,

- Gestió de followers i following. Els usuaris registrats podran seguir a altres usuaris de la xarxa social i veure així més fàcilment les seves publicacions. Quan l'usuari s'identifica en el sistema veurà els missatges publicats per les persones a que segueix ordenats cronològicament (els més nous primer). El llistat d'usuaris seguits i seguidors és públic i no és necessari estar autenticat per a poder veure-la.
- Obtenir la freqüència d'aparició de cadascun dels hashtag.
- Fer un 'Like'. Per marcar que a un usuari li agrada un missatge caldria fer-ho des de la mateixa instància de l'usuari. En aquest cas caldria crear un nou objecte de la classe Like i es relaciona tant amb l'usuari (User) com amb la publicació (Post). Després de crear un nou objecte de tipus Like caldria cridar al mètode 'add' tant de User com de Post per afegir el nou objecte a la llista existent. El mateix passa en esborrar un objecte Like, que caldrà invocar al mètode 'remove' corresponents. És important tenir en compte que un mateix usuari no pot indicar que li agrada un mateix missatge més d'una vegada.