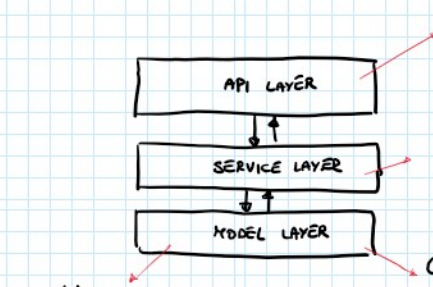
DropBoxAnalyzer

giovedì 9 dicembre 2021 16:48



API:
- /listfolder -> PostListFolder (@R.P. (name="token") String token)
- /get_metadata ->
- /list_revision ->
- /list_file_member ->

FileService() //Interface implementata da => FileServiceImpl() //chiamate HTTP, restituisce JSON
UserService => UserServiceImpl()

CONTENT() //Classe madre
- USERS()
  - OWNER()
  - EDITOR()
- FOLDER()
- FILE()

//Sottoclassi

Funzionalità

1) /get_Metadata

PARAMETRI:

String - path (del file o del folder) ← Unico parametro utile
Boolean - include_media_info
Boolean - include_deleted
Boolean - include_has_explicit_shared_members

RETURN:
- tag (File / Folder)] implementare ereditarietà da qui
- nome
- id
- client_modified] Data ultima modifica
- server_modified] codice utile per statistiche
- rev
- size

NB: con get_Metadata ho solo l'ULTIMA modifica. Se voglio la history delle modifiche, devo chiamare l'API "list_revision" (vedi dopo)

2) /list_folder

PARAMETRI:

String - path (percorso da cui partire per listare i file)
Boolean - recursive (true/false) -> Se abilitato apre anche sottocartelle e lista i file che ci sono dentro.
Boolean - include_media_info
Boolean - include_deleted
Boolean - include_has_explicit_shared_members
Boolean - include_mounted_foulders
Boolean - include_non_downloadable_files

RETURN:

Sottoforma di JSONArray, ritorna la lista di contenuti (file o folder) presenti nel percorso specificato (o anche nei sottopercorsi se "recursive" è "true")

-> Struttura:

```
{ "entries": [ JSONArray ] }
```

↳ Gli elementi del JSONArray sono JSONObject:

```
{
  "tag": folder
  "name": ...
  "path_lower": ...
  "path_display": ...
  "id": ...
}
```

Infine ritorna il "cursor" quando ha letto tutto

3) /list_revisions

PARAMETERS:

String - path
String - mode (Default = "path", ma può anche essere "id")
int - limit (Quante revisioni mostrare)

RETURNS:
- is_deleted ← Può essere anche "true" se il file è stato eliminato. Se "true" dice anche data eliminazione
- entries ← JSONARRAY

↳ Contiene la lista di revisioni, sottoforma di JSONObject: ogni JSONObject dentro "entries" contiene in ordine cronologico tutte le versioni del file specificato con "path". Se il file è stato spostato, si può cercare attraverso l'"id" del file.

- name
- path_lower
- path_display
- id

] tutto uguale a get_Metadata

- name
- path_lower
- path_display
- id

tutto uguale a get_Metadata

!! NB:
- client_modified → Può essere usato per avere uno storico delle modifiche del file e fare statistiche su ogni quanto i file vengono modificati
- server_modified
- rev → Codice univoco associato a ciascuna versione del file
- size
- is_downloadable

IMPORTANTE

[...]

→ Struttura di "entries"

```
"entries" : [
    {
      (versione più recente)
      //Qua c'è il JSONObject che rappresenta la revisione
        più recente
    },
    ...,
    {
      (versione meno recente)
      //Il numero di revisioni mostrate può essere modificato cambiando
    }  l'attributo "limit"
```

4) /list_file_members

Serve per ottenere quali utenti hanno interagito con il file

PARAMETERS

String · file : "id"
Boolean · include_inherited
int · limit (quanti utenti mostrare)

RETURNS

Restituisce 3 JSONARRAY:

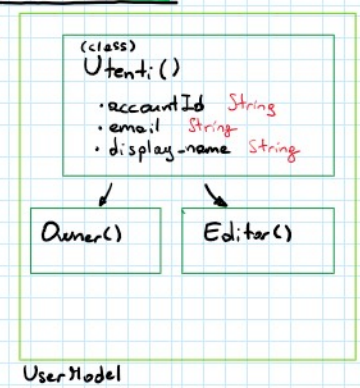
1) "users" [] → Contiene la liste degli utenti → NB.!: Bisogna modellare classe Utenti
2) "groups" []
3) "invitees" []

Siamo interessati solo agli utenti (users).
Ogni utente mette a disposizione le seguenti informazioni:

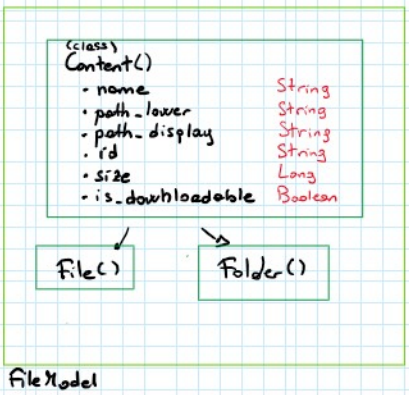
- "access_type" : {"tag" : "owner"} → Può essere usato per ereditarietà su classe Utente
- "user" :
  { "account_id":...,
    "email":...,
    "display_name":...,
    "same_team":...,
  }
- "time_last_seen": ← Ultima modifica dell'utente su quel file
- platform_type :

MODELLAZIONE CLASSI

Models:

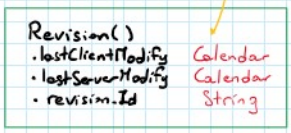


UserModel



FileModel

java.util.Calendar



RevisionModel

SERVICES:

HTTPRequest()
·rootCall(int root, Configuration config, String token) JSONObject

Interface: FileService() extends HTTPRequest()
·getListRevisions(JSONObject JsonObj) <Vector> Revision

//Qua ci vanno le

Interface: FileService() **extends** HTTPRequest()
- getListRevisions(JSONObject JsonObj) <Vector> Revision
- getListFolder(JSONObject JsonObj) <Vector> Content
- getMetadata(JSONObject jsonObj) Content

// Qua ci vanno le implementazioni

Interface: UserService() **extends** HTTPRequest()
- getListUsers(JSONObject JsonObj) <Vector> Users

CONFIGURAZIONE

Configuration()
.getJsonBody() String

// Superclasse [Possibilità di usare overloading x costruttore]

ListFolderConfiguration()

String	· path
Boolean	· recursive
Boolean	· include_media_info
Boolean	· include_deleted
Boolean	· include_has_explicit_shared_members
Boolean	· include_mounted_foulders
Boolean	· include_non_downloadable_files

GetMetadataConfiguration()

String	· path (del file o del folder)
Boolean	· include_media_info
Boolean	· include_deleted
Boolean	· include_has_explicit_shared_members

ListFileMembersConfiguration()

String	· file : "id"
Boolean	· include_inherited
int	· limit

ListRevisionsConfiguration()

String	· path
String	· mode
int	· limit

// Sottoclassi (tutte allo stesso livello)

STATISTICHE

→ Le statistiche sono da effettuare sui file di una cartella DropBox.

Esempio: rotta che restituisce statistiche fatte sututti i file di una cartella

→ Il JSONObject che dovrà essere visualizzato sarà del tipo:

```
{
  "stats" : [
    {
      "name" : "Appunti. paper",
      "number_of_revisions" : 10,
      "time_info" : {
        "average_time_between_each_revision" : "10 days, 1 hour",   | TIME_STAT
        "time_period" : "30 days",
      }
      "size_info" : {
        "average_size_increment" : "4.73 %",                        | SIZE_STAT
        "total_size_increment" : "10%"
      }, "time_period" : 30 days
    }                                                               FILE 1
    {
      ....                                                          FILE 2
    }
  ]
}
```

NOTE
- Cambiare tipo deltaHour da Long a Double

→ Come parametri è possibile specificare

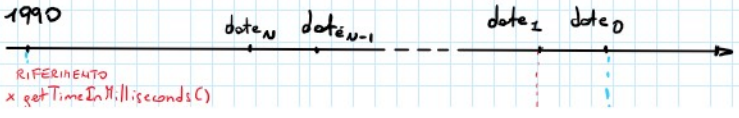
a) cartella
b) periodo di tempo su cui effettuare statistiche
c) se cercare nelle sottocartelle
d) **FILTRI**:
   i) x tipo (.png, .docx, ...)
   ii) x dimensione (soglia)
   iii) x is_downloadable

STATISTICHE SU SINGOLO FILE: effettuate sul singolo file

a) Ogni quanto si effettua una revisione (MEDIA, MAX, MIN)

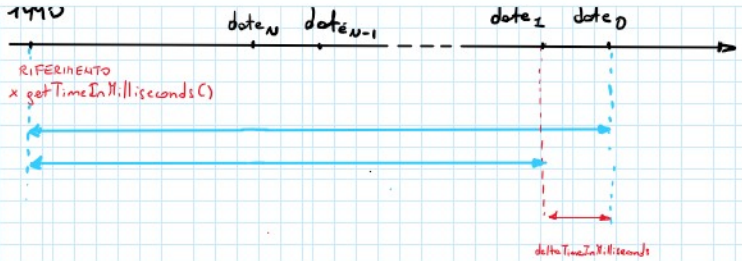
Esempio:

```
REVISIONS r;
r.get(0).date = "2021:12:13T19:02"
r.get(1).date = "2021:12:12T17:45"
```



```
r.get(0).date = "2021:12:13T19:02"
r.get(1).date = "2021:12:12T17:45"
r.get(2).date = "2021:12:11T14.45"

Calendar prevDate = null;
Calendar thisDate = null;
for(Revision revision: revisions)
{
 thisDate = revision.getDate();
 if(revisions.indexOf(revision) != 0)
 {
  deltaTimeInMilliseconds = prevDate.getTime...() - thisDate.getTim...();
  deltaTimeHour = deltaTimeIn... /1000/60/60;
  somma += deltaTimeHour;
 }
 prevDate = thisDate
}
return somma/revisions.length;
```



#

b) numero totale di revisioni di quel file

(FORSE) c) incremento percentuale della dimensione dalla creazione all'ultima revisione

Esempio:

```
Revision r;
r.get(0).size = 220Mb
r.get(1).size = 210Mb
r.get(2).size = 205Mb
r.get(3).size = 200Mb
```

-> Il file è aumentato di 20Mb

delta = dim_finale - dim_iniziale = 20Mb

incr_rel = delta/dim_iniziale = 0.1

incremento % = incr_rel * 100 = 10

-> C'è stato un incremento del 10% della dimensione iniziale del file

200Mb —(+10%)→ 220Mb

#

(SICURAMENTE SÌ) d) Incremento percentuale dimensione file ad ogni revisione (MEDIA, MAX, MIN)

Esempio:

```
Revision r;
r.get(0).size = 220Mb
r.get(1).size = 210Mb
r.get(2).size = 205Mb
r.get(3).size = 200Mb
Long prevSize = r.get(0).size
Long thisSize = 0;

for(Revision revision : revisions)
{
 thisSize = revision.size
 if(revisions.indexOf(revision) != 0)
 {
  deltaSize = prevSize - thisSize;
  incrementoRelativo = deltaSiz/thisSize
  incrementoPercentuale = incrementoRelativo * 100;
  sommaPercentuali += incrementoPercentuale;
 }
 prevSize = thisSize;
}
return sommaPercentuali/revision.length() - 1;
```

delta = 10Mb => 10/210 * 100 = +4.76%

delta = 5Mb => 5/205 * 100 = +2.43%

delta = 5Mb => 5/200 * 100 = +2.5%

SOMMA = 9.71% => SOMMA / N.REV - 1 = +3,24%

-> In media il file è cresciuto del 3.24% ad ogni nuova revisione

-> Il massimo è stato: +4.76%

-> Il minimo è stato: +2.45%