My Project

Generated by Doxygen 1.8.15

Contents

Chapter 1

micado-credential-manager

Version: v1.0

File structures:

- my_script.py : the main script
- resource.csv : containing definition for notification messages, error messages,...
- config.py : configuring the admin's email, database filename and file location
- app :
  - _init_py : initialize log handler and database
  - routes.py : add URL rule for all rest APIs
  - dbmodels.py : implementation of all rest APIs (HTTP return codes are defined by constants in this file)
- template:
  - reset_pwd_mail.html: template mail to notify the user about password reset
- test_script.rst : the Robot framework test script that can be used for automatic acceptance tests
- lib :
  - LoginLibrary.py : the library that contains all API used in the test scripts

How to use Rest API:

Assuming that the following command lines are called inside a docker container in the master node, and the rest APIs are provided by the credential manager, i.e. "credman" container.

- Add a new user (the user's role will be 'USER' as default):

```
curl -d "username=user01&password=123" credman:5001/v1.1/adduser
```

- Verify a user:

```
curl -d "username=user01&password=123" credman:5001/v1.1/verify
```

- Change a user's password:

curl -d "username=user01&oldpasswd=123&newpasswd=456" -X PUT credman:5001/v1.0/changepwd

- Reset a user's password:

curl -d "username=user01" credman:5001/v1.1/resetpwd

- Delete a user:

curl -d "username=user01" credman:5001/v1.1/deleteuser

- Retrieve a user's role

curl -d "username=user01" credman:5001/v1.1/getrole

- Change a user's role (There are 2 roles: user or admin)

curl -d "username=user01&newrole=user" -X PUT credman:5001/v1.1/changerole

curl -d "username=user01&newrole=admin" -X PUT credman:5001/v1.1/changerole

How to use the test_script.rst:

Assuming that you installed Robot framework successfully (Please follow this link if you has not installed the Robot framework yet: https://github.com/robotframework/QuickStartGuide/blob/master/QuickStart.rst#demo-application)

Change the following values defined in the section Variables with approriate information if you wish to test the feature of sending email for resetting password:

- receiverEmail@mail.com
- receverMailPassword
- senderEmail
- imap.gmail.com

You may need to change the settings in your mail account to let less secure apps access your account if the mail server requires. For instance, gmail requires that.

Run the following command line

- robot test_script.rst

Chapter 2

Namespace Index

2.1 Namespace List

Here is a list of all namespaces with brief descriptions:

Chapter 3

Hierarchical Index

3.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 4

Class Index

4.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 5

File Index

5.1 File List

Here is a list of all files with brief descriptions:

Chapter 6

Namespace Documentation

6.1 app Namespace Reference

Namespaces

- dbmodels
- dbmodels_linebr
- routes
- routes_linebr

Variables

- app = Flask(__name__)
- db = SQLAlchemy(app)
- reader = csv.DictReader(f)
- logHandler = RotatingFileHandler('info.log', maxBytes=1000, backupCount=1)
- formatter = logging.Formatter('%(asctime)s - %(name)s - %(module)s - %(funcName)s - %(lineno)d- %(levelname)s - %(message)s')
- auth = None
- secure = None
- mail_handler

6.1.1 Variable Documentation

6.1.1.1 app

```
app.app = Flask(__name__)
```

6.1.1.2 auth

```
tuple app.auth = None
```

6.1.1.3 db

```
app.db = SQLAlchemy(app)
```

6.1.1.4 formatter

```
app.formatter = logging.Formatter('%(asctime)s - %(name)s - %(module)s - %(funcName)s - %(lineno)d-
%(levelname)s - %(message)s')
```

6.1.1.5 logHandler

```
app.logHandler = RotatingFileHandler('info.log', maxBytes=1000, backupCount=1)
```

6.1.1.6 mail_handler

```
app.mail_handler
```

Initial value:

```
1 =  SMTPHandler(
2         mailhost=(app.config['MAIL_SERVER'], app.config['MAIL_PORT']),
3         fromaddr='no-reply@' + app.config['MAIL_SERVER'],
4         toaddrs=app.config['ADMINS'], subject='System Failure',
5         credentials=auth, secure=secure)
```

6.1.1.7 reader

```
app.reader = csv.DictReader(f)
```

6.1.1.8 secure

```
tuple app.secure = None
```

6.2 app.dbmodels Namespace Reference

Classes

- class AccessLog
- class User

Functions

- def hash_passwd (passwd)
- def generate_passwd ()
- def add_user (uname, passwd, email='')
- def create_user_api ()
- def reset_passwd_api ()
- def verify_user_api ()
- def delete_user_api ()
- def read_template (filename)
- def send_reset_pwd_mail (receiver_name, new_pwd, from_addr, to_addr, template_file)
- def change_password_api ()
- def change_role_api ()
- def retrieve_role_api ()

Variables

- int PASSWD_LEN = 256
- int PASSWD_MIN_LEN = 8
- int PASSWD_MAX_LEN = 16
- int MAX_FAILS = 5
- bool LOCKED = False
- bool UNLOCKED = True
- int LOCK_DURATION = 6
- int USER_ROLE = 0
- int ADMIN_ROLE = 2
- list USER_ROLE_LIST = ['user','admin']
- int HTTP_CODE_OK = 200
- int HTTP_CODE_BAD_REQUEST = 400
- int HTTP_CODE_UNAUTHORIZED = 401
- int HTTP_CODE_LOCKED = 423
- int HTTP_CODE_SERVER_ERR = 500
- string REG_EXP_USER_NAME = "^[a-zA-Z0-9_.-]+$"
- string REG_EXP_PASSWD = "^[a-zA-Z0-9]+$"
- FROM_ADDRESS = app.config['MAIL_USERNAME']
- MAIL_PWD = app.config['MAIL_PASSWORD']
- SEND_MAIL_RESET_PWD = app.config['MAIL_SEND_RESET_PWD']
- reader = csv.DictReader(open('resource.csv', 'r'))
- dictionary msg_dict = {}

6.2.1 Function Documentation

6.2.1.1 add_user()

```
def app.dbmodels.add_user (
              uname,
              passwd,
              email = '' )
```

```
[summary]
The function adds a user in the table User.
[description]
This function adds a user into the table User.

Arguments:
    uname {[type: String]} -- [description: user name]
    passwd {[type: String]} -- [description: password]
```

```
184 def add_user(uname, passwd, email=''):
185     """[summary]
186     The function adds a user in the table User.
187     [description]
188     This function adds a user into the table User.
189 
190     Arguments:
191         uname {[type: String]} -- [description: user name]
192         passwd {[type: String]} -- [description: password]
193     """
194     try:
195         # create new user
196         passwd_hash = hash_passwd(passwd)
197         new_user = User(uname, passwd_hash, email)
198 
199         # add new user to database
200         db.session.add(new_user)
201         db.session.commit()
202     # Catch the exception
203     except exc.IntegrityError as e: # existed user
204         db.session.rollback()
205         raise
206     except exc.SQLAlchemyError as e:
207         # Roll back any change if something goes wrong
208         db.session.rollback()
209         raise # Raise error again so that it will be caught in create_user_api()'''
210     except Exception as e:
211         # Roll back any change if something goes wrong
212         db.session.rollback()
213         app.logger.error(e)
214         raise
215     finally:
216         # Close the db connection
217         db.session.close()
218 
219 
```

6.2.1.2 change_password_api()

```
def app.dbmodels.change_password_api ( )
```

```
[summary]
This function is for changing password of a user.
[description]
Only current user is allowed to use this function for himself
Returns:
    [type: json] -- [description: ]
```

```
580 def change_password_api():
581     """[summary]
582     This function is for changing password of a user.
583     [description]
584     Only current user is allowed to use this function for himself
585     Returns:
586         [type: json] -- [description: ]
587     """
588     uname = request.values.get("username")
589     old_passwd = request.values.get("oldpasswd")
590     new_passwd = request.values.get("newpasswd")
591 
592     if(uname==None or old_passwd==None or new_passwd==None or uname=='' or old_passwd=='' or new_passwd==''
    ): # Verify parameters
593         data = {
594             'code' : HTTP_CODE_BAD_REQUEST,
595             'user message'  : msg_dict['lack_of_input'],
596             'result' : msg_dict['lack_of_input'] # User name does not exist
597         }
598         js = json.dumps(data)
599         resp = Response(js, status=HTTP_CODE_BAD_REQUEST, mimetype='application/json')
600         return resp
601 
602     app.logger.info(msg_dict['change_pwd_progress'])#Reset password of user"
603     try:
604         user = db.session.query(User.password_hash).filter_by(username=uname).first()
605         if(user == None):
606             data = {
607                 'code' : HTTP_CODE_BAD_REQUEST,
608                 'developer message' : msg_dict['uname_notexist'], # User name does not exist
609             }
610             js = json.dumps(data)
611             resp = Response(js, status=HTTP_CODE_BAD_REQUEST, mimetype='application/json')
612         else:
613             stored_passwd = user[0]
614             result = check_password_hash(stored_passwd,old_passwd)
615             if (result == True): # password matched
616                 passwd_hash = generate_password_hash(new_passwd)
617                 db.session.query(User).filter_by(username=uname).update({User.password_hash: passwd_hash})
618                 db.session.commit()
619                 data = {
620                     'code' : HTTP_CODE_OK,
621                     'developer message' : msg_dict['change_pwd_success'], # Reset password successfully
622                     # 'new password' : new_passwd
623                 }
624                 js = json.dumps(data)
625                 resp = Response(js, status=HTTP_CODE_OK, mimetype='application/json')
626             else:
627                 data = {
628                     'code' : HTTP_CODE_BAD_REQUEST,
629                     'developer message' : msg_dict['change_pwd_fail'],
630                 }
631                 js = json.dumps(data)
632                 resp = Response(js, status=HTTP_CODE_BAD_REQUEST, mimetype='application/json')
633         return resp
634     except exc.SQLAlchemyError as e:
635         db.session.rollback()
636         app.logger.error(e)
637         abort(HTTP_CODE_SERVER_ERR,msg_dict['sqlalchemy_error'])
638     except Exception as e:
639         db.session.rollback()
640         app.logger.error(e)
641         abort(HTTP_CODE_SERVER_ERR,msg_dict['error_undefined'])
642     finally:
643         db.session.close()
644 
```

6.2.1.3 change_role_api()

```
def app.dbmodels.change_role_api ( )
```

```
645 def change_role_api():
646     try:
647         uname = request.values.get("username")
648         new_role_meaning = request.values.get("newrole")
649 
650         if(uname==None or uname=='' or new_role_meaning==None or new_role_meaning==''): # Verify parameters
```

```
651             data = {
652                 'code' : HTTP_CODE_BAD_REQUEST,
653                 'user message'  : msg_dict['lack_of_input'],
654                 'result' : msg_dict['lack_of_input'] # Lack of user name or password
655             }
656             js = json.dumps(data)
657             resp = Response(js, status=HTTP_CODE_BAD_REQUEST, mimetype='application/json')
658             return resp
659 
660         app.logger.info("Change the user's role")
661 
662         if(new_role_meaning not in USER_ROLE_LIST):
663             data = {
664                 'code' : HTTP_CODE_BAD_REQUEST,
665                 'user message': msg_dict['role_notexist'],
666                 'developer message' : msg_dict['role_notexist'], # User name does not exist
667             }
668             js = json.dumps(data)
669             resp = Response(js, status=HTTP_CODE_BAD_REQUEST, mimetype='application/json')
670             return resp
671 
672         user = db.session.query(User).filter_by(username=uname).first()
673         if(user == None):
674             data = {
675                 'code' : HTTP_CODE_BAD_REQUEST,
676                 'user message': msg_dict['uname_notexist'],
677                 'developer message' : msg_dict['uname_notexist'], # User name does not exist
678             }
679             js = json.dumps(data)
680             resp = Response(js, status=HTTP_CODE_BAD_REQUEST, mimetype='application/json')
681         else:
682             current_role = user.role
683 
684             if(new_role_meaning=='admin'):
685                 new_role = ADMIN_ROLE
686             else:
687                 new_role = USER_ROLE
688 
689             if (current_role == new_role):
690                 data = {
691                     'code' : HTTP_CODE_OK,
692                     'user message' : msg_dict['same_user_role'],
693                     'developer message' : msg_dict['same_user_role'],
694                 }
695             else:
696                 db.session.query(User).filter_by(username=uname).update({User.role: new_role})
697                 db.session.commit()
698                 data = {
699                     'code' : HTTP_CODE_OK,
700                     'user message' : msg_dict['change_user_role_success'],
701                     'developer message' : msg_dict['change_user_role_success'],
702                 }
703             js = json.dumps(data)
704             resp = Response(js, status=HTTP_CODE_OK, mimetype='application/json')
705         return resp
706     except exc.SQLAlchemyError as e:
707         db.session.rollback()
708         app.logger.error(e)
709         abort(HTTP_CODE_SERVER_ERR,msg_dict['sqlalchemy_error'])
710     # Catch the exception
711     except Exception as e:
712         db.session.rollback()
713         db.session.close()
714         app.logger.error(e)
715         abort(HTTP_CODE_SERVER_ERR,msg_dict['error_undefined'])
716 
```

6.2.1.4 create_user_api()

```
def app.dbmodels.create_user_api ( )
```

[summary]

[description]
The function add_user() inserts a user into the database
Input: username in URL request argument

```
Output: password (inputted or randomized), or a raised error
Body: the user's password is randomized or inputted from URL request

Returns:
    [type: json] -- [description: code, message for the user, message for the developer]
```

```
220 def create_user_api():
221     """[summary]
222 
223     [description]
224     The function add_user() inserts a user into the database
225     Input: username in URL request argument
226     Output: password (inputted or randomized), or a raised error
227     Body: the user's password is randomized or inputted from URL request
228 
229     Returns:
230         [type: json] -- [description: code, message for the user, message for the developer]
231     """
232     # parse parameters from http request. Use request.values instead of request.args to indicate parameters
      possibly come from argument or form
233     uname = request.values.get("username")
234     passwd = request.values.get("password")
235     email = request.values.get("email")
236     if(uname is None or uname==''): # verify parameters
237         data = {
238             'code' : HTTP_CODE_BAD_REQUEST,
239             'user message'  : msg_dict['lack_of_input'],#'Add user successfully',
240             'developer message' : msg_dict['lack_of_input']
241         }
242         js = json.dumps(data)
243         resp = Response(js, status=HTTP_CODE_BAD_REQUEST, mimetype='application/json')
244         return resp
245 
246     if(re.match(REG_EXP_USER_NAME,uname)==None): # if name does not follow the rule (only contains a-z,
      A-Z, 0-9)
247         data = {
248             'code' : HTTP_CODE_BAD_REQUEST,
249             'user message'  : msg_dict['wrong_user_name_format'],#'Add user successfully',
250             'developer message' : msg_dict['wrong_user_name_format']
251         }
252         js = json.dumps(data)
253         resp = Response(js, status=HTTP_CODE_BAD_REQUEST, mimetype='application/json')
254         return resp
255 
256     if(email is None):
257         email=''
258 
259     msg_passwd = msg_dict['pwd_setby_user'] # Password is set by inputted value from the user
260     if(passwd is None):
261         # random a default password
262         passwd = generate_passwd()
263         msg_passwd = msg_dict['pwd_generated'] + passwd # Password is auto-generated. Its value is:
264 
265 
266     if(re.match(REG_EXP_PASSWD,passwd)==None): # if password does not follow the rule
267         data = {
268             'code' : HTTP_CODE_BAD_REQUEST,
269             'user message'  : msg_dict['wrong_password_rule'],#'Add user successfully',
270             'developer message' : msg_dict['wrong_password_rule']
271         }
272         js = json.dumps(data)
273         resp = Response(js, status=HTTP_CODE_BAD_REQUEST, mimetype='application/json')
274         return resp
275 
276     # create a user
277     app.logger.info(msg_dict['add_user_progress']) # Trying to add a user to database
278 
279     try:
280         add_user(uname, passwd, email)
281         data = {
282             'code' : HTTP_CODE_OK,
283             'user message'  : msg_dict['add_user_success'],#'Add user successfully',
284             'developer message' : msg_passwd
285         }
286         js = json.dumps(data)
287         resp = Response(js, status=HTTP_CODE_OK, mimetype='application/json')
288         return resp
289     except exc.IntegrityError as e: # existed user
290         data = {
291             'code' : HTTP_CODE_BAD_REQUEST,
292             'user message'  : msg_dict['add_existed_user'], #Add existed user
293             'developer message' : msg_dict['add_existed_user']
294         }
295         js = json.dumps(data)
296         resp = Response(js, status=HTTP_CODE_BAD_REQUEST, mimetype='application/json')
```

```
297         return resp
298     except exc.SQLAlchemyError as e:
299         app.logger.error(e)
300         abort(HTTP_CODE_SERVER_ERR,msg_dict['sqlalchemy_error']) # SQLAlchemyError
301     except Exception as e:
302         app.logger.error(e)
303         abort(HTTP_CODE_BAD_REQUEST,msg_dict['error_undefined'])
304
```

6.2.1.5 delete_user_api()

```
def app.dbmodels.delete_user_api ( )
```

```
[summary]
This function is for deleting a user from the table User
[description]
The function retrieves the user name and password from the request, then delete it if it exists in the databas
Only administrator can call this API.
Returns:
    [type: json] -- [description: code, message for the user, message for the developer]
```

```
482 def delete_user_api():
483     """[summary]
484     This function is for deleting a user from the table User
485     [description]
486     The function retrieves the user name and password from the request, then delete it if it exists in the
        database.
487     Only administrator can call this API.
488     Returns:
489         [type: json] -- [description: code, message for the user, message for the developer]
490     """
491     try:
492         uname = request.values.get("username")
493         if(uname==None or uname==''): # Verify parameters
494             data = {
495                 'code' : HTTP_CODE_BAD_REQUEST,
496                 'user message'  : msg_dict['lack_of_input'],
497                 'result' : msg_dict['lack_of_input'] # Lack of user name or password
498             }
499             js = json.dumps(data)
500             resp = Response(js, status=HTTP_CODE_BAD_REQUEST, mimetype='application/json')
501             return resp
502
503         app.logger.info("Delete a user")
504
505         user = db.session.query(User).filter_by(username=uname).first()
506         if(user == None):
507             data = {
508                 'code' : HTTP_CODE_BAD_REQUEST,
509                 'user message': msg_dict['uname_notexist'],
510                 'developer message' : msg_dict['uname_notexist'], # User name does not exist
511             }
512             js = json.dumps(data)
513             resp = Response(js, status=HTTP_CODE_BAD_REQUEST, mimetype='application/json')
514         else:
515             db.session.delete(user)
516             db.session.commit()
517             data = {
518                 'code' : HTTP_CODE_OK,
519                 'user message' : msg_dict['del_user_success'],
520                 'developer message' : msg_dict['del_user_success'],
521             }
522             js = json.dumps(data)
523             resp = Response(js, status=HTTP_CODE_OK, mimetype='application/json')
524         return resp
525     except exc.SQLAlchemyError as e:
526         db.session.rollback()
527         app.logger.error(e)
528         abort(HTTP_CODE_SERVER_ERR,msg_dict['sqlalchemy_error'])
529     # Catch the exception
530     except Exception as e:
531         db.session.rollback()
532         db.session.close()
533         app.logger.error(e)
534         abort(HTTP_CODE_SERVER_ERR,msg_dict['error_undefined'])
535
```

6.2.1.6 generate_passwd()

```
def app.dbmodels.generate_passwd ( )
```

```
[summary]
The function randomly generates a password.
[description]
This function generates randomly a password from ascii letters and digits. The length of password is limitted

Returns:
    [type: String] -- [description: a generated password]
```

```
171 def generate_passwd():
172     """[summary]
173     The function randomly generates a password.
174     [description]
175     This function generates randomly a password from ascii letters and digits. The length of password is
        limitted from PASSWD_MIN_LEN to PASSWD_MAX_LEN
176
177     Returns:
178         [type: String] -- [description: a generated password]
179     """
180     characters = string.ascii_letters + string.digits # + string.punctuation
181     passwd =  "".join(choice(characters) for x in range(randint(PASSWD_MIN_LEN, PASSWD_MAX_LEN)))
182     return passwd
183
```

6.2.1.7 hash_passwd()

```
def app.dbmodels.hash_passwd (
              passwd )
```

```
[summary]
The function hashes a password.
[description]
This function creates a hash value with salf for an inputted password.
Arguments:
    passwd {[type : string]} -- [description : Hashing using SHA 256 with salt of size 8 bits]

Returns:
    [type : string] -- [description : hash value of size 256 bits]
```

```
157 def hash_passwd(passwd):
158     """[summary]
159     The function hashes a password.
160     [description]
161     This function creates a hash value with salf for an inputted password.
162     Arguments:
163         passwd {[type : string]} -- [description : Hashing using SHA 256 with salt of size 8 bits]
164
165     Returns:
166         [type : string] -- [description : hash value of size 256 bits]
167     """
168     key = generate_password_hash(passwd) # default method='pbkdf2:sha256', default salt_length=8
169     return key
170
```

6.2.1.8 read_template()

```
def app.dbmodels.read_template (
              filename )
```

```
[summary]
This function is for reading a template from a file
[description]

Arguments:
    filename {[type]} -- [description]

Returns:
    [type] -- [description]
```

```
536 def read_template(filename):
537     """[summary]
538     This function is for reading a template from a file
539     [description]
540 
541     Arguments:
542         filename {[type]} -- [description]
543 
544     Returns:
545         [type] -- [description]
546     """
547     with io.open(filename, encoding = 'utf-8') as template_file:
548         content = template_file.read()
549     return Template(content)
550 
551 
```

6.2.1.9 reset_passwd_api()

```
def app.dbmodels.reset_passwd_api ( )
```

```
[summary]
This function is for resetting password of a user.
[description]
Only administrator is allowed to call this API.
Returns:
    [type: json] -- [description: code, message for the developer, new password if resetting successfully]
```

```
305 def reset_passwd_api():
306     """[summary]
307     This function is for resetting password of a user.
308     [description]
309     Only administrator is allowed to call this API.
310     Returns:
311         [type: json] -- [description: code, message for the developer, new password if resetting
      successfully]
312     """
313     uname = request.values.get("username")
314 
315     if(uname==None or uname==''): # Verify parameters
316         data = {
317             'code' : HTTP_CODE_BAD_REQUEST,
318             'user message'  : msg_dict['lack_of_input'],
319             'result' : msg_dict['lack_of_input'] # Lack of user name or password
320         }
321         js = json.dumps(data)
322         resp = Response(js, status=HTTP_CODE_BAD_REQUEST, mimetype='application/json')
323         return resp
324 
325 
326     app.logger.info(msg_dict['reset_pwd_progress'])#Reset password of user"
327     try:
```

```
328         user = db.session.query(User.email).filter_by(username=uname).first()
329         if(user == None):
330             data = {
331                 'code' : HTTP_CODE_BAD_REQUEST,
332                 'developer message' : msg_dict['uname_notexist'], # User name does not exist
333             }
334             js = json.dumps(data)
335             resp = Response(js, status=HTTP_CODE_BAD_REQUEST, mimetype='application/json')
336         else:
337             passwd =  generate_passwd()
338             passwd_hash = generate_password_hash(passwd)
339             db.session.query(User).filter_by(username=uname).update({User.password_hash: passwd_hash})
340             db.session.commit()
341             data = {
342                 'code' : HTTP_CODE_OK,
343                 'developer message' : msg_dict['reset_pwd_success'], # Reset password successfully
344                 'new password' : passwd
345             }
346             js = json.dumps(data)
347             resp = Response(js, status=HTTP_CODE_OK, mimetype='application/json')
348             if(SEND_MAIL_RESET_PWD == True): # NOTE: Add try...catch error from sending email
349                 to_addr = user[0]
350                 try:
351                     if(to_addr!=''): # send email only email address exists
352                         send_reset_pwd_mail(uname,passwd, FROM_ADDRESS,to_addr,'
    template/reset_pwd_mail.html');
353                 except Exception as e:
354                     app.logger.error(e)
355         return resp
356     except exc.SQLAlchemyError as e:
357         db.session.rollback()
358         app.logger.error(e)
359         abort(HTTP_CODE_SERVER_ERR,msg_dict['sqlalchemy_error'])
360     except Exception as e:
361         db.session.rollback()
362         app.logger.error(e)
363         abort(HTTP_CODE_SERVER_ERR,msg_dict['error_undefined'])
364     finally:
365         db.session.close()
366
367 # API to verify a user
```

6.2.1.10 retrieve_role_api()

```
def app.dbmodels.retrieve_role_api ( )
```

```
717 def retrieve_role_api():
718     try:
719         uname = request.values.get("username")
720
721         if(uname==None or uname==''): # Verify parameters
722             data = {
723                 'code' : HTTP_CODE_BAD_REQUEST,
724                 'user message'  : msg_dict['lack_of_input'],
725                 'result' : msg_dict['lack_of_input'] # Lack of user name or password
726             }
727             js = json.dumps(data)
728             resp = Response(js, status=HTTP_CODE_BAD_REQUEST, mimetype='application/json')
729             return resp
730
731         app.logger.info("Retrieve the user's role")
732
733         user = db.session.query(User).filter_by(username=uname).first()
734         if(user == None):
735             data = {
736                 'code' : HTTP_CODE_BAD_REQUEST,
737                 'user message': msg_dict['uname_notexist'],
738                 'developer message' : msg_dict['uname_notexist'], # User name does not exist
739             }
740             js = json.dumps(data)
741             resp = Response(js, status=HTTP_CODE_BAD_REQUEST, mimetype='application/json')
742         else:
743             if(user.role==USER_ROLE):
744                 user_role_meaning = 'user'
745             if (user.role==ADMIN_ROLE):
746                 user_role_meaning = 'admin'
747             data = {
```

```
748                 'code' : HTTP_CODE_OK,
749                 'role' : user_role_meaning,
750             }
751             js = json.dumps(data)
752             resp = Response(js, status=HTTP_CODE_OK, mimetype='application/json')
753         return resp
754     except exc.SQLAlchemyError as e:
755         db.session.rollback()
756         app.logger.error(e)
757         abort(HTTP_CODE_SERVER_ERR,msg_dict['sqlalchemy_error'])
758     # Catch the exception
759     except Exception as e:
760         db.session.rollback()
761         db.session.close()
762         app.logger.error(e)
763         abort(HTTP_CODE_SERVER_ERR,msg_dict['error_undefined'])
```

6.2.1.11 send_reset_pwd_mail()

```
def app.dbmodels.send_reset_pwd_mail (
                receiver_name,
                new_pwd,
                from_addr,
                to_addr,
                template_file )
```

```
552 def send_reset_pwd_mail(receiver_name, new_pwd, from_addr, to_addr, template_file):
553     # set up the SMTP server
554     mail_server = smtplib.SMTP(host='smtp.gmail.com', port=587)
555     if mail_server.starttls()[0] != 220: # start using tls
556         return False # cancel if connection is not encrypted
557     mail_server.login(FROM_ADDRESS, MAIL_PWD) # log in email of the admin
558 
559     message_template = read_template(template_file)
560 
561     msg = MIMEMultipart() # create a message
562 
563     # add in the actual person name to the message template
564     message = message_template.substitute(PERSON_NAME=receiver_name.title(), PWD = new_pwd)
565 
566     # setup the parameters of the message
567     msg['From']= from_addr
568     msg['To']= to_addr
569     msg['Subject']="[MiCADO] Reset your password"
570 
571     msg.attach(MIMEText(message, 'html'))
572 
573     mail_server.sendmail(msg['From'], msg['To'], msg.as_string())
574 
575     del message # delete the message
576 
577     mail_server.quit()
578     return True
579 
```

6.2.1.12 verify_user_api()

```
def app.dbmodels.verify_user_api ( )
```

```
[summary]
This function is for verifying a user.
[description]
The function retrieves the user name and password from the request, then check if they exists in the database
Returns:
    [type: json] -- [description: code, message for the user, authentication result]
```

```
368 def verify_user_api():
369     """[summary]
370     This function is for verifying a user.
371     [description]
372     The function retrieves the user name and password from the request, then check if they exists in the
      database or not.
373     Returns:
374         [type: json] -- [description: code, message for the user, authentication result]
375     """
376     try:
377         uname = request.values.get("username")
378         passwd = request.values.get("password")
379 
380         if(uname==None or passwd==None or uname=='' or passwd==''): # Verify parameters
381             data = {
382                 'code' : HTTP_CODE_BAD_REQUEST,
383                 'user message'  : msg_dict['lack_of_input'],
384                 'result' : msg_dict['lack_of_input'] # Lack of user name or password
385             }
386             js = json.dumps(data)
387             resp = Response(js, status=HTTP_CODE_BAD_REQUEST, mimetype='application/json')
388             return resp
389 
390         # Check password
391         uid_pwd = db.session.query(User.id, User.password_hash, User.email).filter_by(username=uname).first
    ()
392 
393         # If there does not exist the inputted user name 'uname'
394         if(uid_pwd == None):
395             data = {
396                 'code' : HTTP_CODE_UNAUTHORIZED,
397                 'user message'  : msg_dict['uname_pwd_wrong'],
398                 'result' : msg_dict['uname_notexist'] # User name does not exist
399             }
400         else: # If the user exists
401             # Check lock status
402             uid = uid_pwd[0]
403             latest_log = db.session.query(AccessLog.lock_status, AccessLog.no_fails,
    AccessLog.lock_start_time).filter_by(user_id=uid).first()
404             if(latest_log == None): # if there's no log
405                 lock_status = 'Not locked'
406                 number_fails = 0
407             else:
408                 if (latest_log[0] == UNLOCKED): # check lock status
409                     lock_status = 'Not locked'
410                 else:
411                     lock_status = 'Locked'
412                 number_fails = latest_log[1]
413 
414             # If user is locked, check if time is over
415             current_time = datetime.now()
416             if(lock_status == 'Locked'):
417                 if((current_time - latest_log[2]).total_seconds()>LOCK_DURATION): # if time is over
418                     db.session.query(AccessLog).filter_by(user_id=uid).update({AccessLog.lock_status:
    UNLOCKED, AccessLog.no_fails: '0'}) # unlock the user and reset the number of fails to 0
419                     lock_status = 'Not locked'
420                     number_fails = 0
421 
422             if(lock_status == 'Locked'): # Announce that user is being locked
423                 lock_until = latest_log[2] + timedelta(seconds=LOCK_DURATION)
424                 data = {
425                     'code' : HTTP_CODE_LOCKED,
426                     'user message'  : msg_dict['being_locked_user'],
427                     'result' : msg_dict['being_locked_user'] + lock_until.strftime('%m/%d/%Y %H:%M:%S'),
    #http://strftime.org/
428                     'lock status': lock_status,
429                 }
430             else: # verify the password
431                 stored_passwd = uid_pwd[1]
432                 result = check_password_hash(stored_passwd,passwd)
433                 if (result == True): # password matched
434                     data = {
435                         'code' : HTTP_CODE_OK,
436                         'user message'  : msg_dict['auth_success'],
437                         'result' : msg_dict['auth_success'],
438                         'lock status': lock_status,
439                         'role': "user" # 2. modify to other roles later
440                     }
441                     # Unlock
442                     if(latest_log != None):
443                         db.session.query(AccessLog).filter_by(user_id=uid).update({AccessLog.lock_status:
    UNLOCKED, AccessLog.no_fails: 0}) # unlock the user and reset the number of fails to 0
444                 else: # password does not match
445                     number_fails = number_fails + 1
446                     message = msg_dict['pwd_notmatch']
447                     # add log of failed attempts to log-in
448                     if(latest_log == None): # It has not been logged before
```

```
449                         new_log = AccessLog(uid)
450                         db.session.add(new_log)
451                         db.session.commit()
452                     else:
453                         lock_time = datetime.now()
454                         if(number_fails > MAX_FAILS): # Locked if fails more then MAX_FAILS times
455                             db.session.query(AccessLog).filter_by(user_id=uid).update({AccessLog.no_fails:
    number_fails, AccessLog.lock_status: LOCKED, AccessLog.lock_start_time: lock_time}) # update the number of
     fails and lock status
456                             lock_status = 'Locked'
457                             lock_until = lock_time + timedelta(seconds=LOCK_DURATION)
458                             message = message + msg_dict['lock_user_now'] + lock_until.strftime('%m/%d/%Y
     %H:%M:%S')
459                         else:
460                             db.session.query(AccessLog).filter_by(user_id=uid).update({AccessLog.no_fails:
    number_fails}) # update the number of fails only
461                             lock_status = 'Not locked' # check if omitting this is possible
462                         db.session.commit()
463                     if(lock_status=='Locked'):
464                         http_code = HTTP_CODE_LOCKED
465                     else:
466                         http_code = HTTP_CODE_UNAUTHORIZED
467                     data = {
468                         'code' : http_code,
469                         'user message'  : msg_dict['auth_fail'],
470                         'result' : message, # password does not match
471                         'number of fails': number_fails,
472                         'lock status': lock_status
473                     }
474         js = json.dumps(data)
475         resp = Response(js, status=HTTP_CODE_OK, mimetype='application/json')
476         return resp
477     # Catch the exception
478     except Exception as e:
479         app.logger.error(e)
480         abort(HTTP_CODE_SERVER_ERR,msg_dict['error_undefined'])
481
```

6.2.2 Variable Documentation

6.2.2.1 ADMIN_ROLE

```
int app.dbmodels.ADMIN_ROLE = 2
```

6.2.2.2 FROM_ADDRESS

```
app.dbmodels.FROM_ADDRESS = app.config['MAIL_USERNAME']
```

6.2.2.3 HTTP_CODE_BAD_REQUEST

```
int app.dbmodels.HTTP_CODE_BAD_REQUEST = 400
```

6.2.2.4 HTTP_CODE_LOCKED

```
int app.dbmodels.HTTP_CODE_LOCKED = 423
```

6.2.2.5 HTTP_CODE_OK

```
int app.dbmodels.HTTP_CODE_OK = 200
```

6.2.2.6 HTTP_CODE_SERVER_ERR

```
int app.dbmodels.HTTP_CODE_SERVER_ERR = 500
```

6.2.2.7 HTTP_CODE_UNAUTHORIZED

```
int app.dbmodels.HTTP_CODE_UNAUTHORIZED = 401
```

6.2.2.8 LOCK_DURATION

```
int app.dbmodels.LOCK_DURATION = 6
```

6.2.2.9 LOCKED

```
bool app.dbmodels.LOCKED = False
```

6.2.2.10 MAIL_PWD

```
app.dbmodels.MAIL_PWD = app.config['MAIL_PASSWORD']
```

6.2.2.11 MAX_FAILS

```
int app.dbmodels.MAX_FAILS = 5
```

6.2.2.12 msg_dict

```
dictionary app.dbmodels.msg_dict = {}
```

6.2.2.13 PASSWD_LEN

```
int app.dbmodels.PASSWD_LEN = 256
```

6.2.2.14 PASSWD_MAX_LEN

```
int app.dbmodels.PASSWD_MAX_LEN = 16
```

6.2.2.15 PASSWD_MIN_LEN

```
int app.dbmodels.PASSWD_MIN_LEN = 8
```

6.2.2.16 reader

```
app.dbmodels.reader = csv.DictReader(open('resource.csv', 'r'))
```

6.2.2.17 REG_EXP_PASSWD

```
string app.dbmodels.REG_EXP_PASSWD = "^[a-zA-Z0-9]+$"
```

6.2.2.18 REG_EXP_USER_NAME

```
string app.dbmodels.REG_EXP_USER_NAME = "^[a-zA-Z0-9_.-]+$"
```

6.2.2.19 SEND_MAIL_RESET_PWD

```
app.dbmodels.SEND_MAIL_RESET_PWD = app.config['MAIL_SEND_RESET_PWD']
```

6.2.2.20 UNLOCKED

```
bool app.dbmodels.UNLOCKED = True
```

6.2.2.21 USER_ROLE

```
int app.dbmodels.USER_ROLE = 0
```

6.2.2.22 USER_ROLE_LIST

```
list app.dbmodels.USER_ROLE_LIST = ['user','admin']
```

6.3 app.dbmodels_linebr Namespace Reference

6.3.1 Detailed Description

```
[summary]
Database models module
[description]
The dbmodels defines models for the databases for i.e. table User and AccessLog. Later, corresponding database
```

6.4 app.routes Namespace Reference

Functions

- def index ()

6.4.1 Function Documentation

6.4.1.1 index()

```
def app.routes.index ( )
```

```
[summary]
Hello world function
[description]
This function is only for testing if the web service is in operating
```

```
17 def index():
18     """[summary]

19     Hello world function

20     [description]

21     This function is only for testing if the web service is in operating

22     """
23     return "Hello, this is the Credential Manager component!"
24
25 app.add_url_rule('/v1.0/','index',index)
26 # app.add_url_rule('/getuser','get_user', dbmodels.get_user,methods=['GET'])

27 # endpoint to create new user

28 app.add_url_rule('/v1.1/adduser','create_user_api', dbmodels.create_user_api, methods=['POST'])
29 app.add_url_rule('/v1.1/verify','verify_user_api', dbmodels.verify_user_api, methods=['POST'])
30 app.add_url_rule('/v1.1/resetpwd','reset_passwd_api', dbmodels.reset_passwd_api, methods=['PUT'])
31 app.add_url_rule('/v1.1/deleteuser','delete_user_api', dbmodels.delete_user_api, methods=['PUT'])
32 app.add_url_rule('/v1.1/changepwd','change_password_api', dbmodels.change_password_api, methods=['PUT'])
33 app.add_url_rule('/v1.1/changerole','change_role_api', dbmodels.change_role_api, methods=['PUT'])
34 app.add_url_rule('/v1.1/getrole','retrieve_role_api', dbmodels.retrieve_role_api, methods=['GET'])
```

6.5 app.routes_linebr Namespace Reference

6.5.1 Detailed Description

```
[summary]
This modules contains URL rules for all APIs
[description]

Variables:
    app.add_url_rule('/','index',index) {[type]} -- [description]
    app.add_url_rule('/v1.0/adduser','create_user_api', dbmodels.create_user_api, methods {list} -- [descripti
    app.add_url_rule('/v1.0/verify','verify_user_api', dbmodels.verify_user_api, methods {list} -- [descriptio
    app.add_url_rule('/v1.0/resetpwd','reset_passwd_api', dbmodels.reset_passwd_api, methods {list} -- [descri
    app.add_url_rule('/v1.0/deleteuser','delete_user_api', dbmodels.delete_user_api, methods {list} -- [descri
```

6.6 app_linebr Namespace Reference

6.6.1 Detailed Description

```
[summary]
Init module
[description]
The init module creates Flask object, databases, and logging handler
```

6.7 config Namespace Reference

Classes

- class Config

Variables

- basedir = os.path.abspath(os.path.dirname(__file__))

6.7.1 Variable Documentation

6.7.1.1 basedir

```
config.basedir = os.path.abspath(os.path.dirname(__file__))
```

6.8 config_linebr Namespace Reference

6.8.1 Detailed Description

```
[summary]

[description]
This module defines configuration for the project

Variables:
basedir {[type]} -- [description]
```

6.9 LoginLibrary Namespace Reference

Classes

- class LoginLibrary

6.10 my_script Namespace Reference

Variables

- host
- port

6.10.1 Variable Documentation

6.10.1.1 host

```
my_script.host
```

6.10.1.2 port

```
my_script.port
```

6.11 my_script_linebr Namespace Reference

6.11.1 Detailed Description

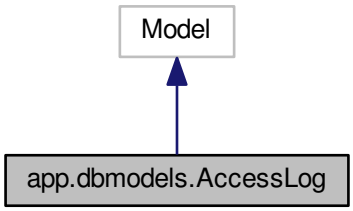
```
[summary]
Main module.
[description]
The main module starts the web service
```

Chapter 7

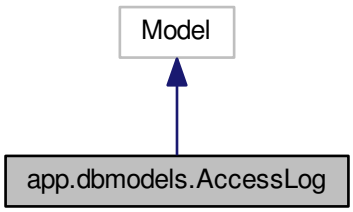
Class Documentation

7.1 app.dbmodels.AccessLog Class Reference

Inheritance diagram for app.dbmodels.AccessLog:



Collaboration diagram for app.dbmodels.AccessLog:



Public Member Functions

- def __repr__ (self)
- def __init__ (self, uid)

Static Public Attributes

- id = db.Column(db.Integer, primary_key=True)
- user_id = db.Column(db.Integer, db.ForeignKey('user.id'))
- lock_status = db.Column(db.Boolean, default = UNLOCKED)
- lock_start_time = db.Column(db.DateTime)
- no_fails = db.Column(db.Integer, default = 0)

7.1.1 Detailed Description

```
[summary]
The class AccessLog defines the model for table AccessLog.
[description]
This class defines all fields of the table AccessLog, for i.e. id, start_time, etc.
Extends:
    db.Model

Variables:
    id {[type: Integer]} -- [description: identity]
    user_id {[type: Integer]} -- [description: Identity of the user. This is the foreign key to the table User
    start_time {[type: Datetime]} -- [description: Datetime of log-in]
```

7.1.2 Constructor & Destructor Documentation

7.1.2.1 __init__()

```
def app.dbmodels.AccessLog.__init__ (
              self,
              uid )
```

```
[summary]
Constructor
[description]
This constructor initalizes a user object with username and password
Arguments:
    username {[type: string]} -- [description: user name]
    password {[type: string]} -- [description: password]
```

```
145     def __init__(self, uid):
146         """[summary]
147         Constructor
148         [description]
149         This constructor initalizes a user object with username and password
150         Arguments:
151             username {[type: string]} -- [description: user name]
152             password {[type: string]} -- [description: password]
153         """
154         self.user_id = uid
155         self.no_fails = 1
156
```

7.1.3 Member Function Documentation

7.1.3.1 __repr__()

```
def app.dbmodels.AccessLog.__repr__ (
                self )
```

```
143     def __repr__(self):
144         return '<AccessLog {}>'.format(self.body)
```

7.1.4 Member Data Documentation

7.1.4.1 id

```
app.dbmodels.AccessLog.id = db.Column(db.Integer, primary_key=True)  [static]
```

7.1.4.2 lock_start_time

```
app.dbmodels.AccessLog.lock_start_time = db.Column(db.DateTime)  [static]
```

7.1.4.3 lock_status

```
app.dbmodels.AccessLog.lock_status = db.Column(db.Boolean, default = UNLOCKED)  [static]
```

7.1.4.4 no_fails

```
app.dbmodels.AccessLog.no_fails = db.Column(db.Integer, default = 0)  [static]
```

7.1.4.5 user_id

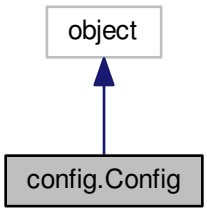
```
app.dbmodels.AccessLog.user_id = db.Column(db.Integer, db.ForeignKey('user.id'))  [static]
```

The documentation for this class was generated from the following file:

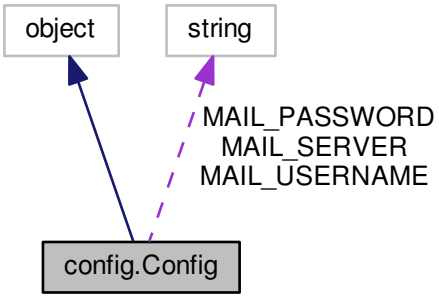
- app/dbmodels.py

7.2 config.Config Class Reference

Inheritance diagram for config.Config:



Collaboration diagram for config.Config:



Static Public Attributes

- SQLALCHEMY_DATABASE_URI = os.environ.get('DATABASE_URL') or \
- bool SQLALCHEMY_TRACK_MODIFICATIONS = False
- PROVISION_FILE = os.environ.get('PROVISION_FILE') or os.path.join(basedir, 'provisioning.csv')
- string MAIL_SERVER = os.environ.get('MAIL_SERVER') or 'smtp.googlemail.com'
- MAIL_PORT = int(os.environ.get('MAIL_PORT') or 587)
- MAIL_USE_TLS = int(os.environ.get('MAIL_USE_TLS') or 1)is not None
- string MAIL_USERNAME = os.environ.get('MAIL_USERNAME') or 'your_email@gmail.com'
- string MAIL_PASSWORD = os.environ.get('MAIL_PASSWORD') or 'your_email_password'
- bool MAIL_SEND_RESET_PWD = os.environ.get('MAIL_SEND_RESET_PWD') or False
- list ADMINS = ['your_email@gmail.com']

7.2.1 Detailed Description

```
[summary]

[description]
This class sets configuration for the project. For instance, database configuration

Variables:
    SQLALCHEMY_DATABASE_URI {[type]} -- [description]
    SQLALCHEMY_TRACK_MODIFICATIONS {bool} -- [description]
```

7.2.2 Member Data Documentation

7.2.2.1 ADMINS

```
list config.Config.ADMINS = ['your_email@gmail.com']  [static]
```

7.2.2.2 MAIL_PASSWORD

```
string config.Config.MAIL_PASSWORD = os.environ.get('MAIL_PASSWORD') or 'your_email_password'
[static]
```

7.2.2.3 MAIL_PORT

```
config.Config.MAIL_PORT = int(os.environ.get('MAIL_PORT') or 587)  [static]
```

7.2.2.4 MAIL_SEND_RESET_PWD

```
bool config.Config.MAIL_SEND_RESET_PWD = os.environ.get('MAIL_SEND_RESET_PWD') or False  [static]
```

7.2.2.5 MAIL_SERVER

```
string config.Config.MAIL_SERVER = os.environ.get('MAIL_SERVER') or 'smtp.googlemail.com'
[static]
```

7.2.2.6 MAIL_USE_TLS

```
config.Config.MAIL_USE_TLS = int(os.environ.get('MAIL_USE_TLS') or 1)is not None  [static]
```

7.2.2.7 MAIL_USERNAME

```
string config.Config.MAIL_USERNAME = os.environ.get('MAIL_USERNAME') or 'your_email@gmail.com'
[static]
```

7.2.2.8 PROVISION_FILE

```
config.Config.PROVISION_FILE = os.environ.get('PROVISION_FILE') or os.path.join(basedir, 'provisioning.↩
csv')  [static]
```

7.2.2.9 SQLALCHEMY_DATABASE_URI

```
config.Config.SQLALCHEMY_DATABASE_URI = os.environ.get('DATABASE_URL') or \  [static]
```

7.2.2.10 SQLALCHEMY_TRACK_MODIFICATIONS

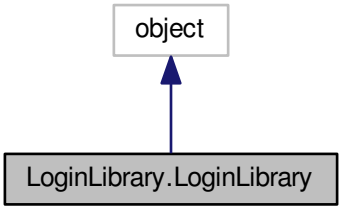
```
bool config.Config.SQLALCHEMY_TRACK_MODIFICATIONS = False  [static]
```

The documentation for this class was generated from the following file:

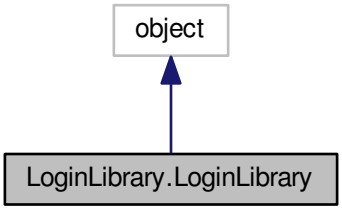
- config.py

7.3 LoginLibrary.LoginLibrary Class Reference

Inheritance diagram for LoginLibrary.LoginLibrary:



Collaboration diagram for LoginLibrary.LoginLibrary:



Public Member Functions

- def __init__ (self)
- def print_hello (self)
- def add_user (self, username=None, password=None, email=None)
- def verify_user (self, username=None, password=None)
- def status_should_be (self, expected_status)
- def reset_passwd (self, username=None)
- def delete_user (self, username=None)
- def change_password (self, username=None, old_passwd=None, new_passwd=None)
- def change_user_role (self, username=None, new_role=None)
- def get_user_role (self, username=None)

7.3.1 Constructor & Destructor Documentation

7.3.1.1 __init__()

```
def LoginLibrary.LoginLibrary.__init__ (
              self )
```

```
10     def __init__(self):
11         #self._sut_path = os.path.join(os.path.dirname(__file__),
12          #                             '..', 'sut', 'login.py')
13         self._status = ''
14
```

7.3.2 Member Function Documentation

7.3.2.1 add_user()

```
def LoginLibrary.LoginLibrary.add_user (
              self,
              username = None,
              password = None,
              email = None )
```

```
20     def add_user(self, username=None, password = None, email = None):
21         url     = 'http://127.0.0.1:5001/v1.1/adduser'
22         payload = {'username': username, 'password': password, 'email': email}
23         res = requests.post(url, data=payload)
24         json_data = json.loads(res.text)
25         '''file = open('test_file.txt','w')
26         file.write(res.url)
27         file.write('  ')
28         file.write(res.content)
29         file.write('  ')
30         file.write(json_data['user message'])
31         file.close()'''
32         self._status = json_data['code']
33
```

7.3.2.2 change_password()

```
def LoginLibrary.LoginLibrary.change_password (
              self,
              username = None,
              old_passwd = None,
              new_passwd = None )
```

```
61     def change_password(self, username=None, old_passwd=None, new_passwd=None):
62         url     = 'http://127.0.0.1:5001/v1.1/changepwd'
63         payload = {'username': username, 'oldpasswd': old_passwd, 'newpasswd': new_passwd}
64         res = requests.put(url, data=payload)
65         json_data = json.loads(res.text)
66         self._status = json_data['code']
67
```

7.3.2.3 change_user_role()

```
def LoginLibrary.LoginLibrary.change_user_role (
              self,
              username = None,
              new_role = None )
```

```
68     def change_user_role(self, username=None, new_role=None):
69         url     = 'http://127.0.0.1:5001/v1.1/changerole'
70         payload = {'username': username, 'newrole': new_role}
71         res = requests.put(url, data=payload)
72         json_data = json.loads(res.text)
73         self._status = json_data['code']
74
```

7.3.2.4 delete_user()

```
def LoginLibrary.LoginLibrary.delete_user (
              self,
              username = None )
```

```
54     def delete_user(self, username=None):
55         url     = 'http://127.0.0.1:5001/v1.1/deleteuser'
56         payload = {'username': username}
57         res = requests.put(url, data=payload)
58         json_data = json.loads(res.text)
59         self._status = json_data['code']
60
```

7.3.2.5 get_user_role()

```
def LoginLibrary.LoginLibrary.get_user_role (
              self,
              username = None )
```

```
75     def get_user_role(self, username=None):
76         url     = 'http://127.0.0.1:5001/v1.1/getrole'
77         payload = {'username': username}
78         res = requests.get(url, data=payload)
79         json_data = json.loads(res.text)
80         self._status = json_data['role']
81
```

7.3.2.6 print_hello()

```
def LoginLibrary.LoginLibrary.print_hello (
              self )
```

```
15     def print_hello(self):
16         url     = 'http://127.0.0.1:5001/v1.0/'
17         res = requests.get(url)
18         return res
19
```

7.3.2.7 reset_passwd()

```
def LoginLibrary.LoginLibrary.reset_passwd (
              self,
              username = None )
```

```
47     def reset_passwd(self, username=None):
48         url     = 'http://127.0.0.1:5001/v1.1/resetpwd'
49         payload = {'username': username}
50         res = requests.put(url, data=payload)
51         json_data = json.loads(res.text)
52         self._status = json_data['code']
53
```

7.3.2.8 status_should_be()

```
def LoginLibrary.LoginLibrary.status_should_be (
              self,
              expected_status )
```

```
42     def status_should_be(self, expected_status):
43         if expected_status != str(self._status):
44             raise AssertionError("Expected status to be '%s' but was '%s'."
45                                  % (expected_status, self._status))
46
```

7.3.2.9 verify_user()

```
def LoginLibrary.LoginLibrary.verify_user (
              self,
              username = None,
              password = None )
```

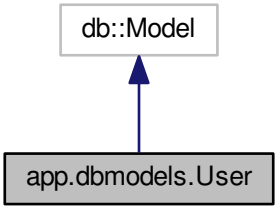
```
34     def verify_user(self, username=None, password=None):
35         url     = 'http://127.0.0.1:5001/v1.1/verify'
36         payload = {'username': username, 'password': password}
37         res = requests.post(url, data=payload)
38         json_data = json.loads(res.text)
39         self._status = json_data['code']
40         #return json_data['code']
41
```

The documentation for this class was generated from the following file:

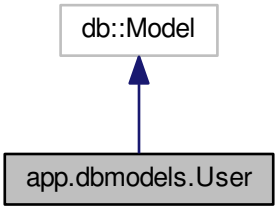
- lib/LoginLibrary.py

7.4 app.dbmodels.User Class Reference

Inheritance diagram for app.dbmodels.User:



Collaboration diagram for app.dbmodels.User:



Public Member Functions

- def __repr__ (self)
- def __init__ (self, username, password, email='', role=USER_ROLE)

Static Public Attributes

- id = db.Column(db.Integer, primary_key=True)
- username = db.Column(db.String(64), index=True, unique=True)
- email = db.Column(db.String(120), index=True)
- password_hash = db.Column(db.String(PASSWD_LEN))
- role = db.Column(db.Integer, default = USER_ROLE)
- accesslog = db.relationship('AccessLog', backref='author', lazy='dynamic')

7.4.1 Detailed Description

```
[summary]
The class User defines the model for table User.
[description]
This class defines all fields of the table User, for i.e. id, username, password, etc.
Extends:
    db.Model

Variables:
    id {[type: Integer]} -- [description: identity]
    username {[type: string]} -- [description: user name]
    email {[type: string]} -- [description: email]
    password_hash {[type: string]} -- [description: hash value of password]
    accesslog {[type: relationship]} -- [description: relationship between the two databases User and AccessLo
```

7.4.2 Constructor & Destructor Documentation

7.4.2.1 __init__()

```
def app.dbmodels.User.__init__ (
              self,
              username,
              password,
              email = '',
              role = USER_ROLE )
```

```
[summary]
Constructor
[description]
This constructor initalizes a user object with username and password
Arguments:
    username {[type: string]} -- [description: user name]
    password {[type: string]} -- [description: password]
```

```
102     def __init__(self, username, password, email='', role=USER_ROLE): # change TO_ADDR to ''
103         """[summary]
104         Constructor
105         [description]
106         This constructor initalizes a user object with username and password
107         Arguments:
108             username {[type: string]} -- [description: user name]
109             password {[type: string]} -- [description: password]
110         """
111         self.username = username
112         self.password_hash = password
113         self.email = email
114         self.role = role
115
116
```

7.4.3 Member Function Documentation

7.4.3.1 __repr__()

```
def app.dbmodels.User.__repr__ (
              self )
```

```
99     def __repr__(self):
100         return '<User {}>'.format(self.username)
101
```

7.4.4 Member Data Documentation

7.4.4.1 accesslog

```
app.dbmodels.User.accesslog = db.relationship('AccessLog', backref='author', lazy='dynamic')
[static]
```

7.4.4.2 email

```
app.dbmodels.User.email = db.Column(db.String(120), index=True)  [static]
```

7.4.4.3 id

```
app.dbmodels.User.id = db.Column(db.Integer, primary_key=True)  [static]
```

7.4.4.4 password_hash

```
app.dbmodels.User.password_hash = db.Column(db.String(PASSWD_LEN))  [static]
```

7.4.4.5 role

```
app.dbmodels.User.role = db.Column(db.Integer, default = USER_ROLE)  [static]
```

7.4.4.6 username

```
app.dbmodels.User.username = db.Column(db.String(64), index=True, unique=True)  [static]
```

The documentation for this class was generated from the following file:

- app/dbmodels.py

Chapter 8

File Documentation

8.1 app/__init__.py File Reference

Namespaces

- app
- app_linebr

Variables

- app.app = Flask(__name__)
- app.db = SQLAlchemy(app)
- app.reader = csv.DictReader(f)
- app.logHandler = RotatingFileHandler('info.log', maxBytes=1000, backupCount=1)
- app.formatter = logging.Formatter('%(asctime)s - %(name)s - %(module)s - %(funcName)s - %(lineno)d-%(levelname)s - %(message)s')
- app.auth = None
- app.secure = None
- app.mail_handler

8.2 app/dbmodels.py File Reference

Classes

- class app.dbmodels.User
- class app.dbmodels.AccessLog

Namespaces

- app.dbmodels
- app.dbmodels_linebr

Functions

- def app.dbmodels.hash_passwd (passwd)
- def app.dbmodels.generate_passwd ()
- def app.dbmodels.add_user (uname, passwd, email='')
- def app.dbmodels.create_user_api ()
- def app.dbmodels.reset_passwd_api ()
- def app.dbmodels.verify_user_api ()
- def app.dbmodels.delete_user_api ()
- def app.dbmodels.read_template (filename)
- def app.dbmodels.send_reset_pwd_mail (receiver_name, new_pwd, from_addr, to_addr, template_file)
- def app.dbmodels.change_password_api ()
- def app.dbmodels.change_role_api ()
- def app.dbmodels.retrieve_role_api ()

Variables

- int app.dbmodels.PASSWD_LEN = 256
- int app.dbmodels.PASSWD_MIN_LEN = 8
- int app.dbmodels.PASSWD_MAX_LEN = 16
- int app.dbmodels.MAX_FAILS = 5
- bool app.dbmodels.LOCKED = False
- bool app.dbmodels.UNLOCKED = True
- int app.dbmodels.LOCK_DURATION = 6
- int app.dbmodels.USER_ROLE = 0
- int app.dbmodels.ADMIN_ROLE = 2
- list app.dbmodels.USER_ROLE_LIST = ['user','admin']
- int app.dbmodels.HTTP_CODE_OK = 200
- int app.dbmodels.HTTP_CODE_BAD_REQUEST = 400
- int app.dbmodels.HTTP_CODE_UNAUTHORIZED = 401
- int app.dbmodels.HTTP_CODE_LOCKED = 423
- int app.dbmodels.HTTP_CODE_SERVER_ERR = 500
- string app.dbmodels.REG_EXP_USER_NAME = "^[a-zA-Z0-9_.-]+$"
- string app.dbmodels.REG_EXP_PASSWD = "^[a-zA-Z0-9]+$"
- app.dbmodels.FROM_ADDRESS = app.config['MAIL_USERNAME']
- app.dbmodels.MAIL_PWD = app.config['MAIL_PASSWORD']
- app.dbmodels.SEND_MAIL_RESET_PWD = app.config['MAIL_SEND_RESET_PWD']
- app.dbmodels.reader = csv.DictReader(open('resource.csv', 'r'))
- dictionary app.dbmodels.msg_dict = {}

8.3 app/routes.py File Reference

Namespaces

- app.routes
- app.routes_linebr

Functions

- def app.routes.index ()

8.4 config.py File Reference

Classes

- class config.Config

Namespaces

- config
- config_linebr

Variables

- config.basedir = os.path.abspath(os.path.dirname(__file__))

8.5 lib/LoginLibrary.py File Reference

Classes

- class LoginLibrary.LoginLibrary

Namespaces

- LoginLibrary

8.6 my_script.py File Reference

Namespaces

- my_script
- my_script_linebr

Variables

- my_script.host
- my_script.port

8.7 readme.md File Reference