Clortho Documentation

Release 0.0.0

Patrick Lawson (ShopWiki)

June 16, 2014

Contents

Contents:

CHAPTER 1

The Authentication Module

exception cloртho.auth.**ActivationError**

class cloртho.auth.**UserBase**(*email*)

The UserBase class subclasses sqlalchemy's default declarative base to provide a schema for basic user authentication. UserBase provides fields and methods for setting and checking a password, generating and checking an activation code, and disabling the user entirely. The email field is unique.

This class **cannot be used on its own**. In order to use this class in your application, subclass it and define the __tablename__ attribute.

__init__(*email*)

Parameters email – **Required**. The user's email address. Validity as an email address is not enforced; any string will be allowed.

Create a new UserBase object. By default, password_is_set, disabled, and activated are all initialized to False.

activate(*activation_code_plaintext*)

Parameters activation_code_plaintext – The activation code to check. If its hash matches activation_code_hash, the user will be activated.

Take the plaintext activation code that was sent to the user, generally via e-mail. Hash the plaintext and compare it against activation_code_hash. If they are the same, set activated to True. Finally, call generate_activation_code() in order to prevent reactivation of the account using the same code. Since we don't save the new reactivation code, it is useless. If the account becomes inactive in the future for any reason, generate_activation_code() must be called again in order to reactivate.

If activation fails, an exception is raised and no change is made to the model.

check_password(*password_plaintext*)

Parameters password_plaintext – The plaintext of the password to check.

If password_is_set is False or disabled is True, return False. Otherwise, take a plaintext password string and hash it using bcrypt along with the salt for this user. If the resulting hash matches the hash in password_hash, return True. The session can now be considered authenticated for this user.

generate_activation_code()

Generate a 20 character random code from letters and digits. This is the activation code that will be sent to the user (presumably via e-mail) and allow them to activate their account. The actual code is not stored locally; like a password, it is hashed and stored in activation_code_hash. The plaintext activation code is returned to the caller. In order to activate the user with the plaintext activation code, call activate().

set_password(*password_plaintext*)

Parameters **password_plaintext** – The plaintext of the password to set.

Given a plaintext password, generate a salted hash using bcrypt. Set `password_is_set` to `True` and `password_hash` to the generated hash. The user can now be authenticated with the plain text password and `check_password()`.

activated = Column(None, Boolean(), table=None)
`activated` is `True` if the user has completed the e-mail verification process using `activation_code_hash`.

activation_code_hash = Column(None, String(length=80), table=None)
The `activation_code_hash` is set to a hash of the activation code that is generated by `generate_activation_code()` and subsequently e-mailed to the user. This code is immediately forgotten by the application; only the bcrypt hashed version of it is stored in `activation_code_hash`. The activation code can be checked using `activate()`.

disabled = Column(None, Boolean(), table=None)
Prevent check_password from returning `True`, regardless of the password supplied.

email = Column(None, Unicode(length=255), table=None)
The user's email. The uniqueness is enforced at the database level.

id = Column(None, Integer(), table=None, primary_key=True, nullable=False)
The primary key and id which will always be used to reference a user.

password_hash = Column(None, String(length=80), table=None)
The hash of the user's password, with salt, as generated by bcrypt.

password_is_set = Column(None, Boolean(), table=None)
`password_hash` might be empty, so we explicitly record whether or not the user has a password set.

CHAPTER 2

Quick Start

```python
import bcrypt

from sqlalchemy.orm import sessionmaker
from sqlalchemy import create_engine
from sqlalchemy.exc import IntegrityError

from cloртho.auth import UserBase, ActivationError

class SimpleUser(UserBase):
    __tablename__ = 'simple_users'

if __name__ == '__main__':
    engine = create_engine('sqlite:///:memory:')
    session = sessionmaker(bind=engine)()

    SimpleUser.metadata.create_all(engine)

    user1 = SimpleUser(email=u'example@example.com')
    session.add(user1)
    session.commit()

    # Users are unique by email
    try:
        user2 = SimpleUser(email=u'example@example.com')
        session.add(user2)
        session.commit()
    except IntegrityError:
        print "Users are unique by email!"
        session.rollback()

    user1.set_password('pwd')

    assert user1.check_password('pwd')
    assert user1.check_password('pwdd') == False
    assert user1.password_is_set
    assert user1.password_hash == bcrypt.hashpw('pwd', user1.password_hash)


    activation_code = user1.generate_activation_code()
    assert user1.activated == False
    user1.activate(activation_code)
    assert user1.activated

```

```
44      user1.activated = False
45      try:
46          user1.activate(activation_code)
47      except ActivationError:
48          print "Activation codes only work once!"
49
50      new_activation_code = user1.generate_activation_code()
51      user1.activate(new_activation_code)
52      assert user1.activated
53
54      assert user1.disabled == False
55      user1.disabled = True
56      assert user1.disabled
57
58      # Password checks fail no matter what if the user is disabled
59      assert user1.check_password('pwd') == False
```

CHAPTER 3

Indices and tables

- *genindex*
- *modindex*
- *search*

C