Watson - Auth

Release 3.3.0

January 22, 2017

Contents

Authorization and authentication library for Watson.

CHAPTER 1

Build Status

CHAPTER 2

Requirements

- watson-db
- py3k-bcrypt

CHAPTER 3

Installation

```
pip install watson-auth
```

CHAPTER 4

Testing

Watson can be tested with py.test. Simply activate your virtualenv and run `python setup.py test`.

CHAPTER 5

Contributing

If you would like to contribute to Watson, please feel free to issue a pull request via Github with the associated tests for your code. Your name will be added to the AUTHORS file under contributors.

CHAPTER 6

Table of Contents

6.1 Usage

A few things need to be configured from within the IocContainer of your application before beginning.

1. An INIT application event listener must be added to your applications events. This injects some default configuration into your application and creates a new dependency in the IocContainer.

```
    'events': {
        events.INIT: [
            ('watson.auth.listeners.Init', 1, True)
        ],
    }
```

2. Configure the user model that you're going to use. Make sure that it subclasses `watson.auth.models.UserMixin`.

```
    'auth': {
        'model': {
            'class': 'tests.watson.auth.support.TestUser'
        }
    }
```

The default configuration (below) can be overridden in your applications config if required.

```
'auth': {
    'model': {
        'columns': {
            'username': 'username',
            'email': 'email'
        }
    },
    'authenticator': {
        'session': 'default',
        'password': {
            'max_length': 30,
            'encoding': 'utf-8'
        },
        'urls': {
            'unauthenticated': 'login',
            'unauthorized': 'unauthorized',
        }
    },
    'login': {
```

```
        'redirect_to_unauthenticated': False,
        'urls': {
            'success': '/',
            'route': 'login'
        },
        'form': {
            'class': 'watson.auth.forms.Login',
            'username_field': 'username',
            'password': 'password',
            'messages': {
                'invalid': 'Invalid username and/or password.'
            }
        }
    },
    'logout': {
        'urls': {
            'success': '/',
            'route': 'logout'
        }
    },
    'forgotten_password': {
        'urls': {
            'route': 'forgotten-password'
        },
        'template': 'auth/emails/forgotten-password',
        'subject_line': 'A password reset request has been made',
        'form': {
            'class': 'watson.auth.forms.ForgottenPassword',
            'username_field': 'username',
            'messages': {
                'success': 'A password reset request has been sent to your email.',
                'invalid': 'Could not find your account in the system, please try again.',
            }
        }
    },
    'reset_password': {
        'template': 'auth/emails/reset-password',
        'authenticate_on_reset': False,
        'subject_line': 'Your password has been reset',
        'urls': {
            'route': 'reset-password',
            'success': '/',
            'invalid': '/'
        },
        'form': {
            'class': 'watson.auth.forms.ResetPassword',
            'username_field': 'username',
            'messages': {
                'success': 'Your password has been changed successfully.',
                'invalid': 'Could not find your account in the system, please try again.',
            }
        }
    },
    'session': {
        'key': 'watson.user'
    },
}
```

Note that any of the url's above can also be named routes.

6.1.1 Authentication

There are several steps to authentication, the first being logging in a user. To do this, add the `login` decorator to the action in your controller that renders the login view.

```
from watson.auth.decorators import login, logout
from watson.auth import forms
from watson.framework import controllers

class Public(controllers.Action):

    @login
    def login_action(self, form):
        # handle the displaying of the form in the view
        # form is automatically injected by the decorator.
        return {'form': form}
```

`@login` also accepts the following arguments:

- method: Can be any valid HTTP method
- form_class: The fully qualified class name of the form being used
- auto_redirect: Whether or not to redirect to config['auth']['url']['login_success']

You'll also want to be able to logout a user, so add the `logout` decorator to the logout action as well.

```
@logout(redirect_url='/')
def logout_action(self):
    pass
```

Make sure you add some routes to your application configuration as well to point to these actions.

```
'routes': {
    'login': {
        'path': '/login',
        'options': {'controller': 'controllers.Public'},
        'defaults': {'action': 'login'}
    },
    'logout': {
        'path': '/logout',
        'options': {'controller': 'controllers.Public'},
        'defaults': {'action': 'logout'}
    }
}
```

Anytime a user visits the **/login**, if the request is a POST (this can be overridden if required) then the user with be authenticated. If they visit **/logout** they they will be logged out and redirected to `redirect_url`. If `redirect_url` is omitted, then the logout view will be rendered.

Once the user has been autheticated, you can retrieve the user within the controller by using 'self.request.

6.1.2 Authorization

watson-auth provides a strongly customizable authorization system. It allows you to configure both roles, and permissions for users. The management of these however is not controlled by watson-auth, so it will be up to you to create the necessary UI to create/delete/update roles.

Please note that some of these actions can also be done via the command *./console.py auth*.

Defining the roles and permissions

First, define some roles for the system and add them to the session:

```
from watson.auth import models

role_regular = models.Role(name='Regular', key='regular')
role_admin = models.Role(name='Admin', key='admin')

session.add(role_regular)
session.add(role_admin)
```

Next, define some permissions:

```
permission_create = models.Permission(name='Create', key='create')
permission_delete = models.Permission(name='Delete', key='delete')
permission_read = models.Permission(name='Read', key='read')

session.add(permission_create)
session.add(permission_delete)
session.add(permission_read)
```

Associate the permissions with the roles:

```
role_admin.add_permission(permission_create)
role_admin.add_permission(permission_read)
role_admin.add_permission(permission_delete)

role_regular.add_permission(permission_create)
role_regular.add_permission(permission_read)
```

Finally, commit them to the database:

```
session.commit()
```

Creating a new user

watson-auth provides a base user mixin that has some common fields, and should be subclassed. watson.auth.models.Model will be the declarative base of whatever session you have configured in config['auth']['model']['session'].

```
from watson.auth import models
from watson.form import fields

class User(models.UserMixin, models.Model):
    __tablename__ = 'users'
    username = Column(String(255), unique=True)
```

Next, create the user and give them some roles and permissions:

```
user = User(username='username', password='some password')
session.add(user)

user.roles.append(role_admin)

session.commit()
```

If no permissions are specified, then the user will receive inherited permissions from that role. Permissions can be given either allow (1) or deny (0).

```
user.add_permission(permission_create, value=0)
```

Authorizing your controllers

Like authentication, authorizing your controllers is done via decorators.

```
from watson.auth.decorators import auth
from watson.framework import controllers

class Public(controllers.Action):

    @auth
    def protected_action(self):
        # some sensitive page
```

`@auth` also accepts the following arguments:

- roles: A string or tuple containing the roles the user must have
- permissions: A string or tuple containing the permissions the user must have
- unauthenticated_url: The url (or named route) to redirect to if the user isn't authenticated. By default this will be config['auth']['authenticator']['urls']['unauthenticated']
- unauthorized_url: The url (or named route) to redirect to if the user isn't authorized. By default this will be config['auth']['authenticator']['urls']['unauthorized']
- should_404: Boolean whether or not to raise a 404 instead of redirecting.

6.1.3 Accessing the user

At any time within your controller you can access the user that's currently authenticated through the request.

```
class MyController(controllers.Action):
    def index_action(self):
        user = self.request.user
```

6.1.4 Resetting a password

As of v3.0.0, the user can now reset their password via the forgotten password functionality.

Several options are also configurable such as automatically logging the user in once they have successfully reset their password. See the configuration settings above for more information.

```
from watson.auth.decorators import login, logout, reset, forgotten
from watson.framework import controllers

class Auth(controllers.Action):
    @login
    def login_action(self, form):
        return {
            'form': form
        }
```

```
    @logout
    def logout_action(self):
        pass

    @forgotten
    def forgotten_password_action(self, form):
        return {
            'form': form
        }

    @reset
    def reset_password_action(self, form):
        return {
            'form': form
        }
```

The user will be emailed a link to be able to reset their password. This template uses whatever renderer is the default set in your project configuration, and can therefore be overridden by creating a new template file in your views directory (*auth/emails/forgotten-password.html* and *auth/emails/reset-password.html*).

The following configuration settings must also be set in order for this to function correctly.

```
'auth': {
    'forgotten_password': {
        'from': 'email@from.com',
    }
}
```

6.2 Reference Library

6.2.1 watson.auth.authentication

class watson.auth.authentication.**Authenticator**(*config*, *session*, *user_model*, *user_id_field*, *email_field*)

Authenticates a user against the database.

session
The SqlAlchemy session to use.

user_model
string – The class name of the user model.

user_id_field
string – The field that is to be used as the username.

__init__(*config*, *session*, *user_model*, *user_id_field*, *email_field*)

assign_user_to_session(*user*, *request*, *session_key*)
Assign a user to the session.

authenticate(*username*, *password*)
Validate a user against a supplied username and password.

Parameters

- **username** (*string*) – The username of the user.
- **password** (*string*) – The password of the user.

get_user(*username*)
Retrieves a user from the database based on their username.

Parameters **username** (*string*) – The username of the user to find.

get_user_by_id(*id*)
Retrieves a user from the database based on their id.

Parameters **id** (*int*) – The id of the user to find.

get_user_by_username_or_email(*data*)
Retrieves a user from the database based on their user or email.

Parameters **data** (*string*) – The username/email of the user to find.

watson.auth.authentication.**check_password**(*password*, *existing_password*, *salt*, *encoding='utf-8'*)
Validate a password against an existing password and the salt used to generate it.

Parameters

- **password** (*string*) – The password to validate
- **existing_password** (*string*) – The password to validate against
- **salt** (*string*) – The salt used to generate the existing_password

Returns True/False if valid or invalid

Return type boolean

watson.auth.authentication.**generate_password**(*password*, *rounds=10*, *encoding='utf-8'*)
Generate a new password based on a random salt.

Parameters

- **password** (*string*) – The password to generate the hash off
- **rounds** (*int*) – The complexity of the hashing

Returns The generated password and the salt used

Return type mixed

6.2.2 watson.auth.authorization

class watson.auth.authorization.**Acl**(*user*)
Access Control List functionality for managing users' roles and permissions.

By default, the user model contains an *acl* attribute, which allows access to the Acl object.

allow_default
boolean – Whether or not to allow/deny access if the permission has not been set on that role.

__init__(*user*)
Initializes the Acl.

Parameters **user** (*watson.auth.models.UserMixin*) – The user to validate against

_generate_user_permissions()
Internal method to generate the permissions for the user.

Retrieve all the permissions associated with the users roles, and then merge the users individual permissions to overwrite the inherited role permissions.

has_permission(*permission*)
Check to see if a user has a specific permission.

If the permission has not been set, then it access will be granted based on the allow_default attribute.

Parameters **permission** (*string*) – The permission to find.

has_role(*role_key*)
Validates a role against the associated roles on a user.

Parameters **role_key** (*string|tuple|list*) – The role(s) to validate against.

6.2.3 watson.auth.decorators

watson.auth.decorators.**auth**(*func=None*, *roles=None*, *permissions=None*, *requires=None*, *unauthenticated_url=None*, *unauthorized_url=None*, *should_404=False*, *redirect=False*)
Makes a controller action require an authenticated user.

By setting additional roles and permissions, finer control can be given to the resource.

Parameters

- **func** (*callable*) – the function that is being wrapped
- **roles** (*string|iterable*) – The roles that the user must have
- **permissions** (*string|iterable*) – The permissions the user must have
- **requires** (*iterable*) – An iterable of watson.validator.Validator objects that allows for custom validations outside of standard roles/permissions.
- **unauthenticated_url** (*string*) – The url to redirect to if the user is not logged in.
- **unauthorized_url** (*string*) – The url to redirect to if the user does not have permission.
- **should_404** (*boolean*) – Raise a 404 instead of redirecting.
- **redirect** (*boolean*) – Remember the url to redirect to after login.

Returns The controller response.

Example:

```
class MyController(controllers.Action):
    def index_action(self):
        return 'Index'

    @auth(roles='admin', permissions='view')
    def protected_action(self):
        return 'Authenticated users only'
```

watson.auth.decorators.**forgotten**(*func=None*, *method='POST'*, *form_class=None*)
Finds a user and sends them a reset password request email.

Parameters

- **func** (*callable*) – the function that is being wrapped.
- **method** (*string*) – The HTTP method that request must match.
- **form_class** (*string*) – The qualified class name of the form.

watson.auth.decorators.**login**(*func=None*, *method='POST'*, *form_class=None*, *auto_redirect=True*)

Attempts to authenticate a user if the required fields have been posted.

By setting auto_redirect to False, the user roles and permissions can be checked within the login route and redirected from there.

Parameters

- **func** (*callable*) – the function that is being wrapped
- **method** (*string*) – The HTTP method that authentication will be performed against.
- **form_class** (*string*) – The qualified class name of the form.
- **auto_redirect** (*boolean*) – Whether or not to automatically redirect to a different url on successful login.

Example:

```
class MyController(controllers.Action):
@login(auto_redirect=False)
def login_action(self):
    return 'Logged In'
```

watson.auth.decorators.**logout**(*func=None*, *redirect_url=None*)

Attempts to log a user out of the application.

Parameters **redirect_url** (*string*) – The url to redirect to.

Example:

```
class MyController(controllers.Action):
    def index_action(self):
        return 'Index'

    @logout(redirect_url='home')
    def logout_action(self):
        pass
```

watson.auth.decorators.**reset**(*func=None*, *method='POST'*, *form_class=None*, *authenticate_on_reset=False*)

Resets a users password if the token matches.

If a token is not matched the user is redirected to the specified route/url.

Parameters

- **func** (*callable*) – the function that is being wrapped.
- **method** (*string*) – The HTTP method that request must match.
- **form_class** (*string*) – The qualified class name of the form.
- **authenticate_on_reset** (*bool*) – Automatically log the user in on success.

6.2.4 watson.auth.forms

class watson.auth.forms.**ForgottenPassword**(*name=None*, *method='post'*, *action=None*, *detect_multipart=True*, *validators=None*, *values_provider=None*, ***kwargs*)

A standard forgotten password form.

class watson.auth.forms.**Login**(*name=None*, *method='post'*, *action=None*, *detect_multipart=True*, *validators=None*, *values_provider=None*, ***kwargs*)
A standard login form containing username and password fields.

class watson.auth.forms.**ResetPassword**(*name=None*, *method='post'*, *action=None*, *detect_multipart=True*, *validators=None*, *values_provider=None*, ***kwargs*)
A standard reset password form.

6.2.5 watson.auth.listeners

class watson.auth.listeners.**Init**
Bootstraps watson.auth into the event system of watson.

load_default_commands(*config*)
Load some existing

class watson.auth.listeners.**Route**
Listens for a route event and attempts to inject the user into the request if one has been authenticated.

6.2.6 watson.auth.models

Sphinx cannot automatically generate these docs. The source has been included instead:

```
# -*- coding: utf-8 -*-
from datetime import datetime
from sqlalchemy import (Column, Integer, String, DateTime, ForeignKey,
                        SmallInteger)
from sqlalchemy.ext.declarative import declared_attr
from sqlalchemy.orm import relationship
from watson.common import imports
from watson.auth import authentication, authorization
from watson.db.models import Model
from watson.db.utils import _table_attr


class Permission(Model):
    id = Column(Integer, primary_key=True)
    name = Column(String(255))
    key = Column(String(255))
    created_date = Column(DateTime, default=datetime.now)

    def __repr__(self):
        return '<{0} key:{1} name:{2}>'.format(
            imports.get_qualified_name(self), self.key, self.name)


class Role(Model):
    id = Column(Integer, primary_key=True)
    name = Column(String(255))
    key = Column(String(255))
    permissions = relationship('RolesHasPermission',
                               backref='roles')
    created_date = Column(DateTime, default=datetime.now)

    def add_permission(self, permission, value=1):
        """Adds a permission to the role.
```

```
        Args:
            Permission permission: The permission to attach
            int value: The value to give the permission, can be either:
                       0 - deny
                       1 - allow
        """
        role_permission = RolesHasPermission(value=value)
        role_permission.permission = permission
        self.permissions.append(role_permission)

    def __repr__(self):
        return '<{0} key:{1} name:{2}>'.format(
            imports.get_qualified_name(self), self.key, self.name)


class UserMixin(object):

    """Common user fields, custom user classes should extend this as well as
    Model.

    Attributes:
        string id_field: The name of the field to use as the id for the user

    Columns:
        string _password: The password of the user, aliased by self.password
        string salt: The salt used to generate the password
        list roles: The roles associated with the user
        list permissions: The permissions associated with the user, overrides
                          the permissions associated with the role.
        date created_date: The time the user was created.
        date updated_date: The time the user was updated.
    """
    __tablename__ = 'users'
    _acl_class = authorization.Acl
    _acl = None
    id = Column(Integer, primary_key=True)
    _password = Column(String(255), name='password')
    salt = Column(String(255), nullable=False)
    created_date = Column(DateTime, default=datetime.now)
    updated_date = Column(DateTime, default=datetime.now)

    @property
    def acl(self):
        """Convenience method to access the users ACL.

        See watson.auth.authorization.Acl for more information.
        """
        if not self._acl:
            self._acl = self._acl_class(self)
        return self._acl

    @declared_attr
    def permissions(cls):
        return relationship(UsersHasPermission, backref='user', cascade='all')

    @declared_attr
    def roles(cls):
```

```
        return relationship(Role,
                            secondary=UsersHasRole.__tablename__,
                            backref='roles', cascade=None)

    @declared_attr
    def forgotten_password_tokens(cls):
        return relationship(ForgottenPasswordToken, backref='user', cascade='all')

    @property
    def password(self):
        """Return the password.
        """
        return self._password

    @password.setter
    def password(self, password):
        """Automatically generates the hashed password and salt when set.

        Args:
            string password: The password to set.
        """
        _pass, salt = authentication.generate_password(password)
        self._password = _pass
        self.salt = salt

    def touch(self):
        """Updates the date the user was modified.
        """
        self.updated_date = datetime.now()

    def add_permission(self, permission, value=1):
        """Adds a permission to the user.

        This overrides any permission given by the associated roles.

        Args:
            Permission permission: The permission to attach
            int value: The value to give the permission, can be either:
                        0 - deny
                        1 - allow
        """
        user_permission = UsersHasPermission(value=value)
        user_permission.permission = permission
        self.permissions.append(user_permission)

    def __repr__(self):
        return '<{0} id:{1}>'.format(imports.get_qualified_name(self), self.id)


class RolesHasPermission(Model):
    role_id = Column(Integer,
                     ForeignKey(_table_attr(Role, 'id')),
                     primary_key=True)
    permission_id = Column(Integer,
                           ForeignKey(_table_attr(Permission, 'id')),
                           primary_key=True)
    permission = relationship(Permission)
    value = Column(SmallInteger, default=0)
```

```
150     created_date = Column(DateTime, default=datetime.now)
151
152
153 class UsersHasPermission(Model):
154     user_id = Column(Integer,
155                      ForeignKey(_table_attr(UserMixin, 'id')),
156                      primary_key=True)
157     permission_id = Column(Integer,
158                            ForeignKey(_table_attr(Permission, 'id')),
159                            primary_key=True)
160     permission = relationship(Permission)
161     value = Column(SmallInteger, default=0)
162     created_date = Column(DateTime, default=datetime.now)
163
164
165 class UsersHasRole(Model):
166     user_id = Column(Integer,
167                      ForeignKey(_table_attr(UserMixin, 'id')),
168                      primary_key=True)
169     role_id = Column(Integer,
170                      ForeignKey(_table_attr(Role, 'id')),
171                      primary_key=True)
172
173
174 class ForgottenPasswordToken(Model):
175     id = Column(Integer, primary_key=True)
176     token = Column(String(255))
177     user_id = Column(Integer,
178                      ForeignKey(_table_attr(UserMixin, 'id')))
179     created_date = Column(DateTime, default=datetime.now)
180
181     def __repr__(self):
182         return '<{0} user id:{1}>'.format(
183             imports.get_qualified_name(self), self.user.id)
```

W

Symbols

A

C

F

G

H

I

L

R

S

U

W