
Watson - Auth

Release 4.0.0

Aug 09, 2017

Contents

1	Build Status	3
2	Requirements	5
3	Installation	7
4	Testing	9
5	Contributing	11
6	Table of Contents	13
6.1	Usage	13
6.2	Reference Library	17
	Python Module Index	23

Authorization and authentication library for Watson.

CHAPTER 1

Build Status

CHAPTER 2

Requirements

- watson-db
- bcrypt
- pyjwt

CHAPTER 3

Installation

```
pip install watson-auth
```


CHAPTER 4

Testing

Watson can be tested with `pytest`. 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.

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': {
  'common': {
    'model': {
      'class': 'app.models.User'
    },
  }
}
```

3. Configure the routes and email address you're going to use for forgotten/reset password.

```
'auth': {
  'common': {
    'system_email_from_address': 'you@site.com',
    'reset_password_route': 'auth/reset-password',
    'forgotten_password_route': 'auth/forgotten-password'
  }
}
```

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

```
'auth': {
  'common': {
    'model': {
      'identifier': 'username',
      'email_address': 'username'
    },
    'session': 'default',
    'key': 'watson.user',
    'encoding': 'utf-8',
    'password': {
      'max_length': 30
    },
  },
}
```

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

If you'd like to include authentication information in the toolbar at the bottom of the page, add the `watson.auth.panels.User` panel to the debug configuration.

```
::
debug = { 'enabled': True, 'toolbar': True, 'panels': {
    'watson.auth.panels.User': { 'enabled': True
    }
  }
}
```

Providers

As of `watson.auth 5.0.0` there is now the concept of 'Auth Providers'. These allow you to authenticate users via number of means. Currently `watson.auth` provides session (`watson.auth.providers.Session`) and JWT (`watson.auth.providers.JWT`) authentication, with OAuth2 support coming shortly.

Depending on what your application requirements are, you might want to use a different provider to the default provider that is used. In order to that, modify your auth configuration.

```
'auth': {
  'providers': {
    'watson.auth.providers.ProviderName': {
      'secret': 'APP_SECRET',
    },
  },
}
```

Each provider may require individual configuration settings, and you'll see a nice big error page if you try to access your site without configuring these first.

Authentication

Setting up authentication will differ slightly depending on the provider you've chosen, but only in the decorators that you are using. You still need to configure 2 things:

1. Routes
2. Controllers

We'll assume that for this example we're just going to use the Session provider.

Start by creating the routes that you're going to need:

```
'routes': {
  auth: {
    'children': {
      'login': {
        'path': '/login',
        'options': {'controller': 'app.auth.controllers.Auth'},
        'defaults': {'action': 'login'}
      },
      'logout': {
        'path': '/logout',
        'options': {'controller': 'app.auth.controllers.Auth'},
        'defaults': {'action': 'logout'}
      },
    },
  },
}
```

Now create the controllers that handle these requests:

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

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

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

You'll notice that there is a `form` argument which is not included in your route definition. This is because the decorators will automatically pass through the form that is being used to validate the user input.

If you'd like to override the views (which is highly suggested), you can put your own views in `views/auth/<action>.html`.

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

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

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 via the watson cli (from your application root).

```
./console.py auth add_role [key] [name]
./console.py auth add_permission [key] [name]
# where [key] is the identifier within the application and [name] is the human_
↳ readable name
```

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 via the watson cli (from your application root).

```
:: ./console.py auth create_user [username] [password] ./console.py auth add_role_to_user [username] [key] ./console.py auth add_permission_to_user [username] [key] [value]
```

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).

Authorizing your controllers

Like authentication, authorizing your controllers is done via decorators.

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

class Public(controllers.Action):

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

`@auth` accepts different arguments, but the common ones are:

- roles: A string or tuple containing the roles the user must have
- permissions: A string or tuple containing the permissions the user must have
- requires: A list of `watson.validators.abc.Validator` objects that are used to validate the user.

Check out the `watson.auth.providers.PROVIDER.decorators` module for more information.

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
```

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.

Create the routes you wish to use:

```
'routes': {
    auth: {
        'children': {
            'reset-password': {
                'path': '/reset-password',
                'options': {'controller': 'app.auth.controllers.Auth'},
                'defaults': {'action': 'reset_password'}
            },
            'forgotten-password': {
                'path': '/forgotten-password',
                'options': {'controller': 'app.auth.controllers.Auth'},
                'defaults': {'action': 'forgotten_password'}
            }
        }
    }
}
```

And then create the controllers that will handle these routes:

```
from watson.auth.providers.session.decorators import forgotten, reset
from watson.framework import controllers

class Auth(controllers.Action):
    @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*).

Reference Library

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.

watson.auth.commands

watson.auth.config

watson.auth.crypto

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.

watson.auth.listeners

```
class watson.auth.listeners.Init
```

Bootstraps *watson.auth* into the event system of *watson*.

```
class watson.auth.listeners.Route
```

Listens for a route event and attempts to inject the user into the request if one has been authenticated.

watson.auth.models

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

```

1  # -*- coding: utf-8 -*-
2  from datetime import datetime
3  from sqlalchemy import (Column, Integer, String, DateTime, ForeignKey,
4                          SmallInteger)
5  from sqlalchemy.ext.declarative import declared_attr
6  from sqlalchemy.orm import relationship
7  from watson.common import imports
8  from watson.auth import authorization, crypto
9  from watson.db.models import Model
10 from watson.db.utils import _table_attr
11
12
13 class Permission(Model):
14     id = Column(Integer, primary_key=True)
15     name = Column(String(255))
16     key = Column(String(255))
17     created_date = Column(DateTime, default=datetime.now)
18
19     def __repr__(self):
20         return '<{0} key:{1} name:{2}>'.format(
21             imports.get_qualified_name(self), self.key, self.name)
22
23
24 class Role(Model):
25     id = Column(Integer, primary_key=True)
26     name = Column(String(255))
27     key = Column(String(255))
28     permissions = relationship('RolesHasPermission',
29                               backref='roles')
30     created_date = Column(DateTime, default=datetime.now)
31
32     def add_permission(self, permission, value=1):
33         """Adds a permission to the role.
34
35         Args:
36             Permission permission: The permission to attach
37             int value: The value to give the permission, can be either:
38                 0 - deny
39                 1 - allow
40
41         """
42         role_permission = RolesHasPermission(value=value)
43         role_permission.permission = permission
44         self.permissions.append(role_permission)
45
46     def __repr__(self):
47         return '<{0} key:{1} name:{2}>'.format(
48             imports.get_qualified_name(self), self.key, self.name)
49
50 class UserMixin(object):
51
52     """Common user fields, custom user classes should extend this as well as
53     Model.
54 
```

```

55     Attributes:
56         string id_field: The name of the field to use as the id for the user
57
58     Columns:
59         string _password: The password of the user, aliased by self.password
60         string salt: The salt used to generate the password
61         list roles: The roles associated with the user
62         list permissions: The permissions associated with the user, overrides
63             the permissions associated with the role.
64         date created_date: The time the user was created.
65         date updated_date: The time the user was updated.
66
67     """
68     __tablename__ = 'users'
69     _acl_class = authorization.Acl
70     _acl = None
71     id = Column(Integer, primary_key=True)
72     _password = Column(String(255), name='password')
73     salt = Column(String(255), nullable=False)
74     created_date = Column(DateTime, default=datetime.now)
75     updated_date = Column(DateTime, default=datetime.now)
76
77     @property
78     def acl(self):
79         """Convenience method to access the users ACL.
80
81         See watson.auth.authorization.Acl for more information.
82         """
83         if not self._acl:
84             self._acl = self._acl_class(self)
85         return self._acl
86
87     @declared_attr
88     def permissions(cls):
89         return relationship(UsersHasPermission, backref='user', cascade='all')
90
91     @declared_attr
92     def roles(cls):
93         return relationship(Role,
94             secondary=UsersHasRole.__tablename__,
95             backref='roles', cascade=None)
96
97     @declared_attr
98     def forgotten_password_tokens(cls):
99         return relationship(ForgottenPasswordToken, backref='user', cascade='all')
100
101     @property
102     def password(self):
103         """Return the password.
104         """
105         return self._password
106
107     @password.setter
108     def password(self, password):
109         """Automatically generates the hashed password and salt when set.
110
111         Args:
112             string password: The password to set.
113         """

```



```

113     _pass, salt = crypto.generate_password(password)
114     self._password = _pass
115     self.salt = salt
116
117     def touch(self):
118         """Updates the date the user was modified.
119         """
120         self.updated_date = datetime.now()
121
122     def add_permission(self, permission, value=1):
123         """Adds a permission to the user.
124
125         This overrides any permission given by the associated roles.
126
127         Args:
128             Permission permission: The permission to attach
129             int value: The value to give the permission, can be either:
130                 0 - deny
131                 1 - allow
132         """
133         user_permission = UsersHasPermission(value=value)
134         user_permission.permission = permission
135         self.permissions.append(user_permission)
136
137     def __repr__(self):
138         return '<{0} id:{1}>'.format(imports.get_qualified_name(self), self.id)
139
140
141 class RolesHasPermission(Model):
142     role_id = Column(Integer,
143                     ForeignKey(_table_attr(Role, 'id')),
144                     primary_key=True)
145     permission_id = Column(Integer,
146                           ForeignKey(_table_attr(Permission, 'id')),
147                           primary_key=True)
148     permission = relationship(Permission)
149     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)
```

watson.auth.panels

watson.auth.providers

watson.auth.providers.**JWT**
alias of Provider

watson.auth.providers.**Session**
alias of Provider

watson.auth.validators

watson.auth.**validators**
alias of *watson.auth.validators*

W

`watson.auth.forms`, [18](#)

`watson.auth.listeners`, [18](#)

Symbols

`__init__()` (watson.auth.authorization.Acl method), [18](#)
`_generate_user_permissions()` (watson.auth.authorization.Acl method), [18](#)

A

Acl (class in watson.auth.authorization), [17](#)
`allow_default` (Acl attribute), [18](#)

F

ForgottenPassword (class in watson.auth.forms), [18](#)

H

`has_permission()` (watson.auth.authorization.Acl method), [18](#)
`has_role()` (watson.auth.authorization.Acl method), [18](#)

I

Init (class in watson.auth.listeners), [18](#)

J

JWT (in module watson.auth.providers), [22](#)

L

Login (class in watson.auth.forms), [18](#)

R

ResetPassword (class in watson.auth.forms), [18](#)
Route (class in watson.auth.listeners), [18](#)

S

Session (in module watson.auth.providers), [22](#)

V

validators (in module watson.auth), [22](#)

W

watson.auth.forms (module), [18](#)
watson.auth.listeners (module), [18](#)