identity.model Documentation

Release 0.5

Thomas G. Willis

December 25, 2009

CONTENTS

identity.model is a python library that attempts to provide a complete user registration/authentication solution for use in wsgi web applications.

Since registration and authentication are so tightly coupled, identity.model also provides wsgi applications for authentication, registration, and password reset.

They are provided as separate applications to provide the most flexibility. Some may need authentication, and have something different in mind for user registration and so on.

CHAPTER

ONE

USE CASES

You will likely find this useful if you have the following use cases for your website.

1. As a user I want to login/logout
2. As a user I want to create an account
3. As a user I want to reset my password
4. As an administrator I want to assign users to groups
5. As an administrator I want to assign permissions to groups
6. As an administrator I want to set a users password
7. As an administrator I want to delete a user
8. As an administrator I want to delete a group
9. As an administrator I want to create a user
10. As an administrator I want to create a group

CHAPTER

TWO

CONTENTS

2.1 Installing identity.model

installation is easy, just download and untar the source, then install in the typical python way.

```
user@computer:~/identity.model$ python setup.py install
```

There's probably other things you might want to know, and I'll get to it eventually.

2.2 Mini Applications

Rather than having identity.model get involved in the request pipeline via middleware, I thought it might be better to have wsgi apps mounted in the URL space that could be consulted by the bigger application when necessary. This may not be the right approach but it was worth a try.

The reason there are 4 distinct applications is so that the site has the most flexibility. Maybe you only need authentication, or maybe you only need registration, maybe you don't want to offer the ability to reset passwords.

Each application can be configured via PasteDeploy ini files. And can be used in larger applications by configuring them in a URLMap.

2.2.1 Authentication

This application provides a very basic login form, and urls for logging in, and logging out. It also provides a method that can be queried by other apps to determine whether a request has a given user associated with it.

Configuration

All that's really required is the database connection information. This is true of all of the mini apps and so, if you are using more than one, you should probably put it in the global configuration.

```
sqlalchemy.url = sqlite:///%(here)s/mydatabase
```

Whatever values prefixed with "sqlalchemy." will be passed to initialize the engine. See sqlalchemy for more details. It can be setup in a PasteDeploy ini file like so...

```
[app:auth]
paste.app_factory = identity.model.apps.auth:app
```

Usage

Functions

See

- identity.model.apps.auth.logout()
- identity.model.apps.auth.get_current_user()
- identity.model.apps.auth.login_url()
- identity.model.apps.auth.logout_url()

2.2.2 Registration

This application provides a user registration routine. The workflow goes like this...

> new user fills in user_name, email_address, password, password_confirmation.
>
> The application will verify a number of things like the uniqueness of the user_name, the uniqueness of the email_address, and that password and password confirmation match.
>
> If the form data is valid, a confirmation link will be generated and sent to the users email address.
>
> When the user clicks on the link in their email, their account will be activated, and they will be able to login.

Configuration

Both registration and password reset require an smtp config along with the database config

```
smtp.server = smtp.mail.com
smtp.port = 25
smtp.from = webmaster@example.com
smtp.user = webmaster
smtp.password = s3cr3t
```

Because this is config info shared between registration and password reset, you may find it convenient to put it in the global section.

Registration Config

You need to also configure a template for the email that gets generated when a user registers.

```
email.activate_subject = test register
email.activate_body = %(here)s/activate.txt
```

email.activate_body refers to a genshi template that constitutes the body of the email. The registration application will populate the template with the a "user_name" value and a "activation_link" value.

```
Hi ${user_name},

You are almost registered and there is one thing left to do to
activate your account, then you can login and use the site. Click
the link below...
```

```
${activation_link}

Sincerely,

Happy Site Master
```

Optionally, you can specify a list of `identity.model.group.Group` and `identity.model.group.Permission` names for your application.

```
groups = users, admins, subscribers
permissions = read, write, delete
```

Usage

Functions

See `identity.model.apps.register.register_url()`

2.2.3 Password Reset

Similar to the registration application. When a user request that their password be reset the workflow goes as follows.

> First the user enters their email address that they registered with. If the email address is valid, and known, the user will be sent a generated link that can be used to reset their password.
>
> The user clicks on the link in the email and arrives at a form to enter their new password.
>
> If the password and password confirmation match, the users password is reset, and if they are logged into the system, they are logged out.
>
> The user can then login with their new password.

Password Reset Config

You need to also configure a template for the email that gets generated when a user requests their password is reset.

```
email.reset_subject = test reset password
email.reset_body = %(here)s/reset.txt
```

email.reset_body refers to a genshi template that constitutes the body of the email. The password reset application will populate the template with the a "user_name" value and a "reset_link" value.

```
Hi ${user_name},

In order to reset your password follow the link below...

${reset_link}

Sincerely,

Happy Site Master
```

Usage

Functions

See `identity.model.apps.password.reset_url()`

2.2.4 Manager

This application provides a way to manage identity.model data including users, groups and permissions. With the manager application you can...

1. Create new users
2. Set users passwords
3. Set the groups a user belongs to
4. Delete users
5. Create groups
6. Set permissions for a group
7. Delete groups

By default, users in the "user-admin" group are the only ones permitted to access the application. The first user to successfully register is granted access.

2.2.5 Full configuration Example

Here's a complete example configuration to put this all in context.

```
[DEFAULT]
debug = true
smtp.server = smtp.mail.com
smtp.port = 25
smtp.from = webmaster@mail.com
smtp.user = webmaster
smtp.password = secret
sqlalchemy.url = sqlite:///%(here)s/_t.db
sqlalchemy.echo = true

[filter:widgets]
use = egg:ToscaWidgets#middleware
toscawidgets.framework.default_view = genshi
toscawidgets.middleware.inject_resources = true

[filter:registry]
use = egg:Paste#registry

[composite:site]
use = egg:Paste#urlmap
/register = register
/auth = auth
/reset = reset
/manage = manager
```

```
[pipeline:auth_pipeline]
pipeline = egg:repoze.tm2#tm
       auth

[app:auth]
paste.app_factory = identity.model.apps.auth:app

[app:manager]
paste.app_factory = identity.model.apps.manager:app


[app:register]
paste.app_factory = identity.model.apps.register:app
reload_templates = true
email.activate_subject =  register
email.activate_body = %(here)s/activate.txt
groups = users, admins, subscribers
permissions = read, write, delete

[app:reset]
paste.app_factory = identity.model.apps.password:app
reload_templates = true
email.reset_subject = reset password
email.reset_body = %(here)s/reset.txt

[pipeline:main]
pipeline =
  registry
  widgets
  egg:repoze.tm2#tm
  site

[server:main]
use = egg:Paste#http
host = 0.0.0.0
port = 6543
```

2.3 identity.model API

If you need to do a lot of customization, then this section is probably useful.

2.3.1 Contents

identity.model.user

exception **PasswordsDontMatchError**
raised when passwords don't match

class **User** (*user_name, email, password*)
User Model Object *Attributes*

- id - id for the user
- user_name - name for user used to login

- email - email address user used to register
- active - boolean indicating whether user has completed registration
- created - date/time the user was created

active
Public-facing descriptor, placed in the mapped class dictionary.

check_password(*password*)
returns true if password results in matching hash

created
Public-facing descriptor, placed in the mapped class dictionary.

email
Public-facing descriptor, placed in the mapped class dictionary.

id
Public-facing descriptor, placed in the mapped class dictionary.

user_name
Public-facing descriptor, placed in the mapped class dictionary.

activate_user(*user_id*)
activate `User` identified by id

deactivate_user(*user_id*)
deactivate `User` identified by id

find_user(*user_name, password, active=True*)
returns `User` object if user/pass matches

user_exists(*user_name=None, email=None*)
if user_name or email exists return true else false

identity.model.group

Simple model for associating resources with groups. resources can be users or whatever, it's up to the consumer to decide.

class Group(*name*)
Group object is readonly at runtime.

add_permission(*permission*)
add permission to this group,commits to engine on success

class **apply_groups**(*resource, groups*)
will delete all existing groups for resource and then apply the list of groups if they exist

class **get_groups**(*resource_id*)
return list of groups for resource id

get_permissions()
tuple of names belonging to group

get_resources()
tuple of resource names belonging to group

name
read-only getter

class **GroupPermission**(*group, permission*)
mapping class to table_group_permission

group
Public-facing descriptor, placed in the mapped class dictionary.

permission
Public-facing descriptor, placed in the mapped class dictionary.

class **GroupResource**(*group, resource*)
mapping class to table_group_resource

group
Public-facing descriptor, placed in the mapped class dictionary.

resource
Public-facing descriptor, placed in the mapped class dictionary.

class **Permission**(*name*)
Permission object is readonly at runtime.

class **apply_permissions**(*group, permissions*)
will delete all existing groups for resource and then apply the list of groups if they exist

get_groups()
tuple of names belonging to permission

class **get_permissions**(*group*)
tuple of names belonging to group

name
read-only getter

identity.model.command

in order to not be passing the user-id around and possibly compromising the system, all pending operations will be stored in a table keyed by id

class **Command**()
Command represents a pending action, generally you want to store data that can be used to do whatever you have to do. It will be given an id that can be referenced to retrieve it later.

This is handy for things like password reset which could be made to require email verification first

data
Public-facing descriptor, placed in the mapped class dictionary.

id
Public-facing descriptor, placed in the mapped class dictionary.

identity.model.common

useful things for db model objects

class **Queryable**()
provides common query operations to keep it dry

class **get**(**key*)
get by primary key

class **list**()
select all

class **qry**(**attrs*)
return a query for the class with or without specified attributes

identity.model.meta

used internally for getting hooked into session provided by some other system.

_s()
returns session object used internally in queries for the model, presumably the session has already been initialized by calling `identity.model.manage.init_model()`

commit()
method to be used to commit, by default calls commit on `_s()`, but can be overridden to work with some other transaction manager

rollback()
method to be used to rollback, by default calls rollback on `_s()`, but can be overridden to work with some other transaction manager

identity.model.manage

For the most part, this should constitute all that is needed for a public api.

create_user(*user_name, email, password*)
initializes a `identity.model.user.User` object and returns it. validates that user_name and email are unique will throw error for violations. Up to the caller to add to the session etc...

get_user_by_id(*user_id*)
given user_id returns `identity.model.user.User`

get_user(*user_name, password*)
if user_name and password are correct, will return `identity.model.user.User` object otherwise return None

activate_user(*user_name, password*)
Activation process, will activate `identityt.model.user.User` and return corresponding object

exception DuplicateUserError
raised duplicate user is found

exception AuthenticationError
raised is user/pass doesn't resolve to a valid user

init_model(*metadata, session, committer=None, rollbacker=None, init_callback=None*)
must be called before using any of the model objects to setup metadata,sessionmaker etc....

init_callback is a callable that will be called with no params once init_model has finished

create_tables()
util method for creating schema once init_model has been called

drop_tables()
util method for dropping schema, call init_model prior

create_command(*command, data*)
commands are used during the registration phase prior to activation. they represent things to do upon activation(when user responds to activation email) command and data need to be json serializable

this function creates and returns a command object that can be saved as part of the registration process.

returns identity.model.command.Command

get_command(*id*)
retrieve command by id

returns identity.model.command.Command

remove_command(*id*)
remove identity.model.command.Command from the database(will be removed when session is committed

init_new_user(*user, cb=None*)

assign user to identity.model.group.Group s and maybe other things cb: callable to call for further processing...

identity.model.repoze_integration

repoze.who integration with user model

class GroupSource()
Group Source uses *identity.model.group*

class PermissionSource()
Permission Source uses *identity.model.permission*

class WhoPlugin()
Authenticator Plugin for repoze.who

add_metadata(*environ, identity*)
slap extra user related info on the identity

authenticate(*environ, identity*)
authenticate according to protocol

Auth

module for the mini application for login/logout functionality Authorization package. Contains wsgi app for providing the login/logout logic

get_current_user(*request*)
Get's current user associated with the request, returns None if anonymous

login_url(*request*)
url for login action

logout(*request*)
performs logout for user associated with this request. headers returned need to be passed back with the response so the users cookies get nuked appropriately

logout_url(*request*)
url for logout action

Models

models package

class **LoginAction**()
Context for Login

form(*action='?POST', user=None, password=None, came_from='/'*)
initialize a paramless callable to render the form when called

validate(*request*)
validate form data and set user or throw error returns `LoginResult`

class **LoginResult**(*user=None, headers=None, destination='/', message=None*)
Result of `LoginAction` success

indicating success or failure

user the `identity.model.user.User` associated with the login

headers the response headers that need to be sent in order f or the user to remain logged in

destination url to redirect to

message optional message to display to the user

class **LogoutAction**()
Logout Action callable nukes cookie headers when called and returns `LogoutResult` to be used in building the subsequent response

class **LogoutResult**(*headers=None, destination='/', message='You have now been logged out'*)
Result of calling `LogoutAction`

headers headers to be used in the response

destination the url to redirect to

message a message to optionally show to the user

response A redirect response that can be returned instead of doing something more specific

success indicating whether the action was successful or not

class **Root**(*parent=None, name=None*)
Container for handling login and logout operations

appmaker(*global_config, **settings*)
initializes application and returns root factory

Views

views for auth app

view_login(*context, request*)
view for `identity.model.apps.auth.models.LoginAction`

view_logout(*context, request*)
view for `identity.model.apps.auth.models.LogoutAction`

view_root(*context, request*)
root model does nothing...

Widgets

ui widgets for auth application

class **LoginForm**(**args, **kw*)
the form for login (user/pass)

show_login_form(*action='?POST', user=None, password=None, came_from=None*)
function to render `LoginForm` with some data

action form post action

user user name to place in user field

password password to place in password field

came_from url where user is coming from

Register

Registration package Contains wsgi app for handling user registration process

register_url(*request*)
url for register action

Models

models for registration app

class **RegisterResult**(*uid=None, destination='/', message=None*)
Returned as result of validation on the `RegisterAction`

class **Root**(*config, parent=None, name=None*)
Handles all things about a user submitting a registration form

activate(*command_id*)
if cmd is valid, activate user...

queue_user(*data, request*)
schedules potential user for activation returns command id that can be used in an activation link

validate(*request*)
called to validate form, if form is valid, create a command for activation and send off the registration email returns `RegisterResult`

Views

views for registration

view_register(*context, request*)
View for `identity.model.apps.register.models.Root`

Widgets

registration related widgets

class **RegisterForm**(**args, **kw*)
Registration Form

show_register_form(*action='?POST'*)
function to render `RegisterForm`

action form post action

Password

Password package Contains wsgi app to handle password reset logic

reset_url(*request*)
url for reset action

Models

Password reset related models

class **ResetReqResult**(*success, result=None, message=None*)
Result of action

success indicates success or failure

form if not successful, this is the form to display with error messages

ResetResult
alias of `ResetReqResult`

class **Root**(*config, parent=None, name=None*)
password reset request

change_password(*data, command_id*)
data = form_data command = command that initiated this request

password_reset_form(*command_id, id*)
Returns callable to generate the password reset form

queue_request(*data, request*)
schedules potential user for activation returns command id that can be used in an activation link

validate_email(*request*)
validates the form data presumably from `password_reset_form()` if validation is sucessful, it will send an email to the user. returns a `ResetReqResult`

validate_reset_password(*request*)
validates the form data for password reset. retuns `ResetResult`

Views

password reset related views

view_reset(*context, request*)
view for `identity.model.apps.password.models.Root`

Widgets

widgets related to password reset

class **EmailReqForm**(**args, **kw*)
Password reset request form

class **EmailReqSchema**(**args, **kw*)
Schema to validate the `EmailReqForm`

email is required, must be a valid email, and belong to an existing account

Messages

badDictType: The input must be dict-like (not a `%(type)s`: `%(value)r`)

badType: The input must be a string (not a `%(type)s`: `%(value)r`)

empty: Please enter a value

missingValue: Missing value

noneType: The input must be a string (not None)

notExpected: The input field `%(name)s` was not expected.

class **ResetPasswordForm**(**args, **kw*)
Reset Password Form

show_email_req_form(*action='?POST'*)
function for rendering `EmailReqForm`

action form post action

show_reset_password_form(*id, action='?POST'*)
function to render `ResetPasswordForm` with parameters

id user id that the password is being changed for

action form post action

validate_email_req_form(*request*)
function that validates POST data against the schema of `EmailReqForm`

request Webob request containing form data

returns result,data: if valid result is True and data is the validated form data, else, result is false and data is the newly rendered form complete with errors included

validate_reset_password_form(*request*)
validates post data in the request

request Webob request containing form data

returns result,data: if valid result is True and data is the validated form data, else, result is false and data is the newly rendered form complete with errors included

Manager

module for the mini application for login/logout functionality Manager package Contains wsgi application for managing users and groups

Models

models for manager mini-app

GroupDeleteResult
alias of UserDeleteResult

class GroupModel(*config, parent*)
GroupModel provides all the facilities for managing identity.model.group.Group and identity.model.group.Permission data on behalf of the view functions

create(*request*)
handles taking post data(presumably from the create form) validates it, creates a identity.model.group.Group with the selected identity.model.group.Permission list

create_form()
reference to a callable that will render the create form. the callable returned takes a dictionary of values that will be used when rendering the form

create_url(*request*)
returns url for creating a new user

delete(*request, group*)
handles taking post data(presumably from the delete form) and deletes the identity.model.group.Group

delete_form()
reference to a callable that will render the delete form. the callable returned takes a dictionary of values that will be used when rendering the form

delete_url(*request*)
returns url for deleting existing user

get_group(*gid*)
returns a identity.model.group.Group for the given gid

permission_url(*request*)
returns url for deleting existing user

permissions_form()
reference to a callable that will render the permissions form. the callable returned takes a dictionary of values that will be used when rendering the form

set_permissions(*request, group*)
handles taking post data(presumably from the permissions form) and applies the identity.model.group.Permission list to the identity.model.group.Group

GroupSaveResult
alias of UserSaveResult

class UserDeleteResult(*success, destination='/'*)
returned by UserModel to indicate success or failure and recommend next steps

success indicates success or failure for the action that was performed

response recommended response to return(usually a redirect destination

class UserModel(*config, parent*)
UserModel provides all the facilities for managing identity.model.user.User data on behalf of the view functions

create(*request*)
handles taking post data(presumably from the create form) validates it, creates an active identity.model.user.User with the selected identity.model.group.Group list

create_form()
reference to a callable that will render the create form. the callable returned takes a dictionary of values that will be used when rendering the form

create_url(*request*)
returns url for creating a new identity.model.user.User

delete(*request, user*)
handles taking post data(presumably from the delete form) and deletes the identity.model.user.User.

delete_form()
reference to a callable that will render the delete form. the callable returned takes a dictionary of values that will be used when rendering the form

delete_url(*request*)
returns url for deleting existing identity.model.user.User

get_user(*uid*)
returns a identity.model.user.User object for the specified ui

groups_form()
reference to a callable that will render the groups form. the callable returned takes a dictionary of values that will be used when rendering the form

groups_url(*request*)
returns url for setting password for existing identity.model.user.User

password_form()
reference to a callable that will render the password form. the callable returned takes a dictionary of values that will be used when rendering the form

password_url(*request*)
returns url for setting password for existing identity.model.user.User

set_groups(*request, user*)
handles taking post data(presumably from the group form) and applies the identity.model.group.Group list to the identity.model.user.User

set_password(*request, user*)
handles taking post data(presumably from the password form) validates it, and sets the password for the identity.model.user.User

class **UserSaveResult**(*data=None, form=None, destination='/', message=None*)
returned by UserModel to indicate success or failure and recommend next steps

success indicates success or failure for the action that was performed

response recommended response to return(usually a redirect destination

appmaker(*global_config, **settings*)
initializes application and returns root factory

Views

views for manager app

form_callable (*form, data*)
basically a closure, returns a paramless callable that will call the form with **data as it's parameters when called

get_group_record (*context, request*)
Determines what the current identity.model.group.Group to perform operations on based on POST or GET params

get_user_record (*context, request*)
Determines what the current identity.model.user.User to perform operations on based on POST or GET params

view_add_group (*context, request*)
Shows identity.model.apps.manager.widgets.NewGroupForm

view_add_user (*context, request*)
Shows identity.model.apps.manager.widgets.NewUserForm to create a new user.

view_delete_group (*context, request*)
Shows identity.model.apps.manager.widgets.DeleteForm for identity.model.group.Group

view_delete_user (*context, request*)
Shows identity.model.apps.manager.widgets.DeleteForm for identity.model.user.User

view_groups (*context, request*)
Shows identity.model.apps.manager.widgets.GroupsForm

view_password (*context, request*)
Shows identity.model.apps.manager.widgets.PasswordForm

view_permissions (*context, request*)
Shows identity.model.apps.manager.widgets.PermissionsForm

view_root (*context, request*)
since this is a fairly complex view, provide the data in a variety of ways to meet a variety of theming needs

returns dictionary with...

display_users callable to render identity.model.apps.manager.widgets.UserTable

display_groups callable to render identity.model.apps.manager.widgets.GroupTable

users list of identity.model.user.User objects

groups list of identity.model.group.Group objects

new_user_url url for creating new identity.model.user.User

new_group_url url for creating new identity.model.group.Group

views dictionary of views (identity.model.apps.manager.widgets.UserTable,:class:*identity.model.apps.m*
keyed by type (identity.model.user.User, identity.model.group.Group,:class:*identity.model.apps.m*
with keys 'users', 'groups'

data dictionary of lists (identity.model.user.User, identity.model.group.Group)
keyed by type (identity.model.user.User, identity.model.group.Group)
with keys 'users', 'groups'

create_urls dictionary of create_urls keyed by type (identity.model.user.User, identity.model.group.Group) with keys 'users', 'groups'

tables dictionary of tablse/views((identity.model.apps.manager.widgets.UserTable,:class:*identity.model.*
keyed by type (identity.model.user.User, identity.model.group.Group)
with keys 'users', 'groups'

Widgets

widgets for manager mini-app

class DeleteForm(**args, **kw*)
Delete confirmation form.

fields

- id: hidden
- yes: button
- no: button

class GroupTable(**args, **kw*)
Table view for displaying groups

class GroupValidator(**args, **kw*)
validates that group name is not empty

Messages

badDictType: The input must be dict-like (not a %(type)s: %(value)r)

badType: The input must be a string (not a %(type)s: %(value)r)

empty: Please enter a value

missingValue: Missing value

noneType: The input must be a string (not None)

notExpected: The input field %(name)s was not expected.

class GroupsForm(**args, **kw*)
Form for setting groups

class NewGroupForm(**args, **kw*)
form for creating new groups

class NewUserForm(**args, **kw*)
form for creating new users

class PasswordForm(**args, **kw*)
Set password form which is identical to identity.model.apps.password.widgets.ResetPasswordForm
except for the submit_text so we should probably refactor this at some point

class PermissionsForm(**args, **kw*)
form for setting permissions

class UniqueGroup(*name_field='name'*)
validates that a group name does not already exist in the database

Messages

badType: The input must be a string (not a %(type)s: %(value)r)

empty: Please enter a value

noneType: The input must be a string (not None)

class UserTable(**args, **kw*)
Table view for displaying users

show_delete_form(***kw*)
function returns renders DeleteForm

show_group_table(*delete_url=None, permission_url=None, permission_provider=<function get_permissions_for_group at 0xadebbfc>*)
function returns a callable for rendering GroupTable

delete_url url base for delete group action

permission_url url base for settings permissions action

permission_provider function to return permissions for groups (passed group name)

show_groups_form(***kw*)
function rendering GroupsForm

show_new_group_form()
function renders NewGroupForm

show_new_user_form()
function renders NewUserForm

show_password_form(***kw*)
function renders PasswordForm

show_permissions_form(***kw*)
function renders PermissionsForm

show_user_table(*delete_url=None, password_url=None, groups_url=None, group_provider=<function get_groups at 0xadebbc4>*)
returns callable for rendering UserTable

delete_url url base for the delete user action

password_url url base for the set password action

groups_url url base for the set groups action

group_provider function to provide groups for a user (passed user_id)

Common

Widgets that are common between 2 or more applications things common amongst all the apps

Widgets

common widgets shared between apps

class EmailExists(*email_field='email'*)
Validator that validates that the supplied email exists in the identity.model database

before we send an email for a password reset confirm the email address is known to us

Messages

badType: The input must be a string (not a %(type)s: %(value)r)

empty: Please enter a value

noneType: The input must be a string (not None)

class NewUserValidator(**args, **kw*)
Schema that validates that required information for a new user has been supplied and is valid

user_name can not already exist in the database email can not already exist in the database

Messages

badDictType: The input must be dict-like (not a `%(type)s`: `%(value)r`)

badType: The input must be a string (not a `%(type)s`: `%(value)r`)

empty: Please enter a value

missingValue: Missing value

noneType: The input must be a string (not None)

notExpected: The input field `%(name)s` was not expected.

class PasswordValidator(**args, **kw*)
Schema that validates that required password information has been supplied and is valid(passwords match).

this is only needed at registration time, or when the password fields are set in the management ui, because we store as hashes it would force the form to have it each time, but we can't pre-populate it.

password can not be empty password2 can not be empty both password and password2 must match

Messages

badDictType: The input must be dict-like (not a `%(type)s`: `%(value)r`)

badType: The input must be a string (not a `%(type)s`: `%(value)r`)

empty: Please enter a value

missingValue: Missing value

noneType: The input must be a string (not None)

notExpected: The input field `%(name)s` was not expected.

class UniqueUserInfo(*user_name_field='user_name', email_field='email'*)
Validator that validates that supplied email and user_name are unique

both user_name and email must not already exist in the database

Messages

badType: The input must be a string (not a `%(type)s`: `%(value)r`)

empty: Please enter a value

noneType: The input must be a string (not None)

class UserValidator(**args, **kw*)
Schema that validates that the required user information has been supplied

user_name can not be empty email can not be empty

Messages

badDictType: The input must be dict-like (not a `%(type)s`: `%(value)r`)

badType: The input must be a string (not a `%(type)s`: `%(value)r`)

empty: Please enter a value

missingValue: Missing value

noneType: The input must be a string (not None)

notExpected: The input field `%(name)s` was not expected.

CHAPTER

THREE

INDICES AND TABLES

- *Index*
- *Module Index*
- *Search Page*

MODULE INDEX

I

INDEX

Symbols

A

C

D

E

F

P

Q

R

S

U

V

W