Anuket Documentation

Release 0.5.2

Bertrand Lecervoisier

August 15, 2012

CONTENTS

Author LDPL - Laboratoire Départemental de Préhistoire du Lazaret

Version 0.5.2, released 2012-08-15

PyPI http://pypi.python.org/pypi/anuket

License Expat license (MIT license)

Docs http://anuket.readthedocs.org/

Source https://github.com/lazaret/anuket (Git)

Bugs https://github.com/lazaret/anuket/issues

CHAPTER

ONE

INTRODUCTION

Anuket is an opiniated Python web framework based on Pyramid. It is intended to be used by other Pyramid applications as a base for common choices.

Choices done for you by Anuket:

- Pyramid: Core web framework
- URLDispatch: Resources location mecanism
- SQLAlchemy: SQL toolkit and ORM
- FormEncode: Form validator
- Pyramid_simpleform: Form generator
- Mako: Templating engine
- Twitter Bootstrap: Default templates

The project also integrate:

- Alembic: Database migration tool for SQLAlchemy
- Babel: Internationalization and Localization tools
- Cracklib: Password checking library
- Cryptacular: Password hashing framework

CHAPTER

TWO

USAGE

Anuket is writed so he can be extended by other Pyramid applications. Normaly, it is not necessary to fork Anuket. Just use the extensibility mecanism. For details please read the Pyramid documentation

The main objective of Anuket, is to be used for database related applications. We will use it at the Lazaret laboratory mostly for filling and quering relational databases with web forms. If your application is like this, Anuket may be suited for you.

CHAPTER

THREE

WARNING

The developement is still at an early stage and other choices have to be made before the 1.0 version. In particulary take care of the facts than:

- The database schema is subject to change
- WTForms may be choised over FormEncode and Pyramid_simpleform
- We may add a 'plugin' system to allow optional features

CHAPTER

FOUR

ALTERNATIVES

There are already other web frameworks and CMS based on Pyramid. Anuket have take inspiration from them but the choices made are sometime different. They may be best suited to your needs. Have a look on Akhet, Apex, Cone.app, Kotti, Khufu, Ptah, PyCK, Pyrone and more.

CHAPTER

FIVE

NARATIVE DOCUMENTATION

5.1 Install

This install process, explain how to install Anuket and how to quicky test it in an isolated environemnt.

5.1.1 Prerequistes

Anuket need **Python 2.7** to work. It is not tested yet for Python 3 but we are planing to do so.

Anuket need the following Python packages to work :

- pyramid
- SQLAlchemy
- pyramid_beaker
- pyramid_debugtoolbar
- pyramid_exclog
- pyramid_simpleform
- pyramid_tm
- alembic
- Babel
- formencode
- cracklib
- cryptacular
- Mako

You can install Python and the packages prerequistes with your OS package manager, from PyPI or directly from source.

Note: If you just want to play with Anuket and quicky see what he offer then we advice you to read the Anuket tutorial bundled in the documentation.

5.1.2 Install Anuket from PyPI

You can simply install Anuket from PyPI by using pip or easy_install:

```
$ pip install anuket
```

or:

```
$ easy_install anuket
```

This will install Anuket and all the necessary Python packages dependencies.

Alternaly, if you fell more risky you can install the last development version from the the Git repository:

```
$ git clone git@github.com:lazaret/anuket.git
$ cd anuket/
$ python setup.py develop
```

Keep in mind than the development version may not work perfectly!

5.1.3 Creating your application with Anuket

Anuket is not intended to be used alone. Anuket is intended to be used by other Pyramid based aplication by exenting it. Please read the Anuket tutorial and the pyramid documentation to know how to do this.

5.2 Tutorial

This tutorial will explain the course to create a simple *Hello world* application with Anuket.

5.2.1 Introduction

Anuket is a python application based on the Pyramid web framework. Anuket is intended to provide features not bundled by Pyramid. The main features provided by Anuket are:

- Twitter Bootstrap template
- Form management
- Flash messages
- Database based users & groups
- Database migrations
- Admin tools backend

The main objective of Anuket, is to be used for relational database driven applications. For example, application with web forms, or wikis.

During this tutorial, we will create a very simple *Hello world* application. You can browse the code of this example application in our Git repository: https://github.com/lazaret/anuket-example

5.2.2 Install Anuket and the prerequistes

Anuket require the Python language (2.7 version), and a SQL database engine. In this tutorial we will use SQLite.

For this tutorial we will asume than they are already installed on your computer. If it's not the case, please install them first.

Prepare the isolated environment

To avoid messing with your working Python environment during this tutorial, we will create first an an isolated environment:

```
$ easy_install virtualenv
$ virtualenv --no-site-packages tutorial
$ source tutorial/bin/activate
```

This have:

- installed the virtualenv and pip packages
- activated the *tutorial* isolated environment

When you will have finished the tutorial, you can get rid of everything we done by just deleting the */tutorial* directory.

Install Anuket

You can simply install Anuket from PyPI by using *pip*, a Python packages installer witch have been installed with *virtualenv* :

```
(tutorial)$ pip install anuket
```

This will install Anuket and all the required Python packages (*Pyramid*, *SQLAlchemy*, *Mako*, *alembic*, etc.). You can display the list of all the installed packages with *pip*:

```
(tutorial)$ pip freeze
```

Note: As today, Anuket require the *cracklib* module with serve to test the security of the user password. If you have probelm during the install it's probably because you need to install the cracklib developement libraries. (probably crackib-devel or libcrack2-dev depending on your OS)

5.2.3 Create the example application

Now we have a working environment with Anuket, Pyramid ans all the other prerequistes installed. It's time now to create ou example application.

Create the application

We need first to create a Pyramid application with the *starter* scafold:

```
(tutorial)$ pcreate -t starter anuket-example
```

This create a minimalistic Pyramid application with all the default files. We will edit this file to create our example application. In a future release we will add ou own *anuket* scafold to start with.

Configure the application

We need now to edit three files to tell the application to use Anuket as a base:

- setup.py
- development.ini
- anuketexample/__init__.py

If your are lazy you can download them directly from http://github.com/lazaret/anuket-example

First we need to tell to the `setup.py` file than `anuket` is a prerequiste for our application:

```
requires = [
    'anuket',
    ]
```

Secondly we have to edit the `development.ini` file to add the options required by Anyket:

```
[app:main]
use = egg:anuket-example

pyramid.reload_templates = true
pyramid.debug_authorization = false
pyramid.debug_notfound = false
pyramid.debug_routematch = false
pyramid.default_locale_name = en
pyramid.available_languages = en fr
pyramid.includes =
    pyramid_debugtoolbar
    pyramid_tm

# mako template settings
mako.directories =
    anuket:templates
    anuketexample:templates
mako.module_directory = %(here)s/var/templates
mako.imports = from markupsafe import escape_silent
mako.default_filters = escape_silent

# pyramid_beaker settings
session.type = file
session.data_dir = %(here)s/var/sessions/data
session.lock_dir = %(here)s/var/sessions/lock
session.key = anuketkey
session.secret = anuketexamplesecret
session.timeout = 3600

# database connection string
sqlalchemy.url = sqlite:///%(here)s/anuket-example.db

anuket.backup_directory = %(here)s/var/backups
anuket.brand_name = Example
```

Most notabily we have setup this options:

- The SQLAlchemy database type and name
- The Mako template engine options
- The Beaker session options

- Anuket options

Finaly we need to tell imperatively to our application than we will use Anuket. For this will have to edit the `__init__.py` file inside the `anuketexample` directory.

```
from pyramid.config import Configurator
from pyramid_beaker import session_factory_from_settings

from sqlalchemy import engine_from_config

import anuket
from anuket.models import DBSession
from anuket.models.rootfactory import RootFactory
from anuket.security import get_auth_user


def main(global_config, **settings):
    """ This function returns a Pyramid WSGI application.
    """
    # configure SQLAlchemy
    engine = engine_from_config(settings, 'sqlalchemy.')
    DBSession.configure(bind=engine)

    config = Configurator(settings=settings)
    # configure the root factory (used for Auth & Auth)
    root_factory = RootFactory
    config.set_root_factory(root_factory)
    # configure session
    session_factory = session_factory_from_settings(settings)
    config.set_session_factory(session_factory)
    # configure auth & auth
    config.include(anuket.add_authorization)
    # set an auth_user object
    config.set_request_property(get_auth_user, 'auth_user', reify=True)
    # configure subscribers
    config.include(anuket.subscribers)
    # configure static views
    config.include(anuket.add_static_views)
    # configure routes
    config.include(anuket.views.root)
    config.include(anuket.views.tools)
    config.include(anuket.views.user)
    from anuketexample import views
    config.include(views)
    # configure views
    config.scan('anuket')
    config.scan()

    config.add_translation_dirs('anuket:locale')
    config.set_locale_negotiator('anuket.lib.i18n.locale_negotiator')

    return config.make_wsgi_app()
```

In this file we have configured the database, the authentification, the session, the routes, the view and even the translation system. And as you can see, most of them comme from Anuket.

Initialize the application

We need now to initialize the database for our application. For this we will use the *initialize_anuket_db* script.

```
(tutorial)$ initialize_anuket_db development.ini
```

The script read the *sqlalchemy.url* option and our database model, then create the database, and finaly fill it with default values.

As we use SQLite the script have normaly created a anuket-example.db file witch is our database.

Serve the application

At this point we have now a working application than we can serve:

```
(tutorial)$ pserve develpment.ini
```

You can access to the application with a web browser at http://0.0.0.0:6543/

For now, the application only offer the base application from Anuket. You can already login to the aplication with the default admin credentials: *admin/admin*.

Add the *hello_world* views

Our starter application is ready. We now need to add features by extending the application. We will add a wonderfull *Hello World* feature. For this you have to edit the `views.py` file inside the `anuketexample` directory.

```
from pyramid.view import view_config


def includeme(config):
    """ Configure the Hello World routes."""
    config.add_route('hello', '/hello')
    config.add_route('hello_admin', '/hello/admin')


@view_config(route_name='hello', renderer='hello.mako')
def hello_world(request):
    """ Render the `Hello world` view."""
    hello = u"Hello World!"
    return dict(hello=hello)


@view_config(route_name='hello_admin', permission='admin',
             renderer='hello.mako')
def hello_admin(request):
    """ Render the `Hello admin` view."""
    hello = u"Hello Admin!"
    return dict(hello=hello)
```

This will create:

- The *include* function, witch register the routes of the views when you lauch the application with the *config.scan()* call.
- The *Hello World* view available at http://0.0.0.0:6543/hello
- The *Hello admin* view available at http://0.0.0.0:6543/hello/admin

Add the *hello.mako* template

Our two views need a template to be rendered:

```
## -*- coding:utf-8 -*-
##
<%inherit file="anuket:templates/base.mako" />

<h2>${hello}</h2>


## Page title
<%def name="page_title()">
${_(u"Hello")}
</%def>
```

We use here a Mako template witch inherit from the default templates of Anuket. The template itsef just display the *hello* variable witch is returned by the views.

Connect to the views

Now the application can serve the Anuket views and our two new views:

```
(tutorial)$ pserve develpment.ini
```

You can for example connect with your browser to:

Adress	View name	Application
http://0.0.0.0:6543/	*home*	anuket
http://0.0.0.0:6543/hello	*hello*	anuket-example
http://0.0.0.0:6543/login	*login*	anuket
http://0.0.0.0:6543/hello/admin	*hello_admin*	anuket-example

Note than the *hello_admin* require an user with admin permission. If you try to access to it without login first the application will redirect you to the *login* view.

5.2.4 Further reading

- Creating your first Pyramid application
- Extending an existing Pyramid application

5.3 Console scripts

Anuket provide console scripts:

- `initialize_anuket_db`
- `backup_anuket_db`
- `upgrade_anuket_db`

5.3.1 Initializing the database

`initialize_anuket_db` is a script used to initialize your SQLAlchemy database.

```
$ initialize_anuket_db development.ini
```

The script will create a database by using the `sqlalchemy.url` option from the .ini file, and will fill it with default values.

5.3.2 Backup the database

`backup_anuket_db` is a script used to backup your database.

```
$ backup_anuket_db development.ini
```

The script will create a bziped SQL dump of your database using the `anuket.backup_directory` option from the .ini file (/var/backups by default). The filename will include the date of the backup.

5.3.3 Upgrade the database

`upgrade_anuket_db` is a script used to upgrade your database in case of database schema change in future Anuket versions. The database schemas changes are maitained by using Alembic.

```
$ upgrade_anuket_db development.ini
```

The script will check first if there is an up-to-date database backup, and if it's the case, it will perform the upgrade of the database by using Alembic.

See Also:

Database migrations

5.4 Database migrations

During the process of writing an application it's often necessary to modify and extend the database structure. To allow this, Anuket integrate the Alembic package witch work with SQLAlchemy.

To migrate your database, you can use the `upgrade_anuket_db` script provided with Anuket:

```
$ upgrade_anuket_db development.ini
```

or the alembic command:

```
$ alembic -c development.ini -n app:main upgrade head
```

The only direrence is than the first one will check if you have a database backup before performing the update.

> **Warning:** SQLite have limitaions with the ALTER statement and so Alembic do not support well tables and columns alteration of SQLite databases.

5.5 Flash messages

To add flash messages from the views you can use the Pyramid flash messages mecanisms.

For example:

```
request.session.flash(u"info message", 'info')
request.session.flash(u"success message", 'success')
request.session.flash(u"warning message", 'warn')
request.session.flash(u"error message", 'error')
```

5.6 Running tests

To run the tests, you need first to install the test packages used to build the tests : `nose`, `WebTest`

```
$ pip install nose WebTest
```

Optionaly you can install the coverage package.

```
$ pip install coverage
```

Then run the tests:

```
$ python setup.py nostetests
```

Or simply:

```
$ nostetests
```

5.7 Authors

Bertrand Lecervoisier - LDPL - Laboratoire Départemental de Préhistoire du Lazaret

5.8 Licenses

5.8.1 Anuket License

The code in Anuket is supplied under this license:

OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

5.8.2 Additional Licenses

Anuket use the Twitter Bootstrap for his templates. Twitter Boostrap is licencied under the Apache 2 license. See http://twitter.github.com/bootstrap/

Anuket use the Raphaël Icon-Set via @font-face for his templates. The Raphaël Icon-Set via @font-face is licencied under the MIT license (Expat license). See http://icons.marekventur.de/

5.9 Changelog

5.9.1 0.5 released 2012-08-15

- Fix documentation and packaging issues

5.9.2 0.5 released 2012-08-14

- Add database migration support with alembic
- Add database console scripts
- Add Beaker sessions
- Add a tutorial in the documentation
- Improve the security of the deletion of users

5.9.3 0.4 released 2012-04-29

- Change the auth_user request property to an AuthUser object
- Extend the tests suite
- Extend the documentation

5.9.4 0.3 released 2012-03-30

- Change package name to Anuket
- Extend the user management tool
- Add cracklib password checker
- Add pyramid_exclog for production environment

5.9.5 0.2 released 2012-03-20

- Add 100% tests coverage
- Add minimalistic documentation

5.9.6 0.1 released 2012-03-12

- Base templates with mako and Twitter Bootstrap
- Basic Auth&Auth mechanism and model with SQLAlchemy

5.10 Todo

5.10.1 0.6

- Decide if we use WTForms instead of pyramid_simpleform
- Improve the SecurePassword validator
- Make cracklib an optional dependency

5.10.2 0.7

- Use form template widgets
- Add a form for users to change their datas
- Add a came_from mecanism
- Add a paster template for child applications

5.10.3 Future

- Add data versioning in the database
- Add a 'plugin' mecanism
- Add security and sanity check tools
- Add a logs display tool
- Add database tools in the admin part
- Add an options table for the application
- Add an image manipulation library. Maybe PIL or Wand as a Tween

5.10.4 After version 1.0

- Add Python3 support

CHAPTER

SIX

API DOCUMENTATION

6.1 API Documentation

Anuket is an opiniated Python web framework based on Pyramid.

6.1.1 `anuket.models` – SQLAlchemy models

Models for the Anuket application.

`anuket.models.auth` – Auth model

`SQLAlchemy` model definition for authentification.

class `anuket.models.auth.`**`AuthGroup`**(***kwargs*)
AuthGroup table and model definition.

Define the database *auth_group* table for the users groups and a method to query the table. This table is used used for ACLs principals.

classmethod `get_by_id`(*group_id*)
Query the *auth_group* table by *group_id*.

Parameters **group_id** (*integer*) – the group id

Returns a `sqlalchemy.orm.query.Query` object

class `anuket.models.auth.`**`AuthUser`**(***kwargs*)
AuthUser table and model definition.

Define the database *auth_user* table for the authenticated users and the methods for querring the table or check the validity of the password.

classmethod `get_by_id`(*user_id=None*)
Query the *auth_user* table by *user_id*.

Parameters **user_id** – the user id

Returns a `sqlalchemy.orm.query.Query` object

classmethod `get_by_username`(*username=None*)
Query the *auth_user* table by username.

Parameters **username** (*unicode*) – the user username

Returns a `sqlalchemy.orm.query.Query` object

classmethod **get_by_email**(*email=None*)
Query the auth_user table by email.

Parameters **username** (*unicode*) – the user email

Returns a `sqlalchemy.orm.query.Query` object

classmethod **check_password**(*username*, *password*)
Check the user password.

Check if the submited password for username is the same than the encrypted one recorded in the database. Return None if the username did not exist.

Parameters

- **username** (*unicode*) – the user username
- **username** – the submited password

Returns True if the password is correct. false if incorect

Return type boolean

anuket.models.migration – Migration model

`SQLAlchemy` model definition for database migration with `Alembic`.

class anuket.models.migration.**Migration**
Migration table and model definition.

Reflect the default version table used by Alembic. This table is used for tracking database migrations.

anuket.models.rootfactory – RootFactory model

`Pyramid` *root factory* model.

class anuket.models.rootfactory.**RootFactory**(*request*)
Add ACLs to the default route factory.

6.1.2 anuket.views – Pyramid views

Views callables for the Anuket application.

anuket.views.root – Root views

Main views for the application.

anuket.views.root.**root_view**(*request*)
Render the root pages.

Render the home page, the login page and 404 not found page.

Parameters **request** – a `pyramid.request` object

anuket.views.root.**forbiden_view**(*request*)
Redirect the 403 forbiden view.

Authenticated user with not enought permission are redirected to the home page. Non-autenthicaded users are redirected to the login page. A corresponding flash message is also added to the error message queue.

Parameters **request** – a `pyramid.request` object

`anuket.views.root.`**`login_view`**(*request*)
Render the login form.

Display an empty login form or check the submited credentials with the ones from the database. Add a success flash message, an userid in the cookies and redirect to the home page if the credentials are goods. Add an error flash message and display again the login form if the credentials are wrong.

Parameters **request** – a `pyramid.request` object

`anuket.views.root.`**`logout_view`**(*request*)
Logout authenticated user.

Clear the credentials of the connected user if any. Then, redirect to the home page and add a info flash message.

Parameters **request** – a `pyramid.request` object

`anuket.views.tools` – Tools views

Admin tools views for the application.

`anuket.views.tools.`**`tools_index_view`**(*request*)
Render the tools home page.

Parameters **request** – a `pyramid.request` object

`anuket.views.user` – User views

Admin tools for user management.

`anuket.views.user.`**`user_add_view`**(*request*)
Render the add user form page.

Display an empty user form or validate the user submited form. If the form is validated then add the user datas to the database and a success flash message. If the form is not valid, then display again the form with validation errors. Return also a list of groups to use in the group select form.

Parameters **request** – a `pyramid.request` object

`anuket.views.user.`**`user_delete_view`**(*request*)
Delete an user.

Seek the database for the user datas based on user_id used in the route. If the user did not exist then add an error flash message and redirect to the user list. If the user exist then delete the user in the database, add a warning flash message and then redirect to the user list.

Parameters **request** – a `pyramid.request` object

`anuket.views.user.`**`user_edit_view`**(*request*)
Render the edit user form page.

Seek the database for the user datas based on user_id used in the route. If the user did not exist then add an error flash message and redirect to the user list. If the user exist then render the user form filled with the user datas. If the form is validated then change the user datas to the database and add success flash message. If the form is not valid, then display again the form with validation errors. Return also a list of groups to use in the group select form.

Parameters **request** – a `pyramid.request` object

anuket.views.user.**user_list_view**(*request*)
Render the user list page.

Return a paged user list from the database. The paged list can be ordered by username, first name or last name. Add an error flash message if the list is empty. Return also basic users statistics.

Parameters **request** – a `pyramid.request` object

anuket.views.user.**user_show_view**(*request*)
Render the show user datas page.

Seek the database for the user datas based on user_id used in the route. If the user did not exist then add an error flash message and redirect to the user list. If the user exist then return his datas.

Parameters **request** – a `pyramid.request` object

anuket.views.user.**password_edit_view**(*request*)
Render the change password form page.

Seek the database for the user datas based on user_id used in the route. If the user did not exist then add an error flash message and redirect to the user list. If the user exist then render an empty password form. If the form is validated then change the user password in the database and add success flash message. If the form is not valid, then display again the form with validation errors.

Parameters **request** – a `pyramid.request` object

6.1.3 anuket.scripts – Console scripts

Command line scripts for the Anuket application.

anuket.scripts.initializedb – Database initialization

Script to initialize the Anuket database.

class anuket.scripts.initializedb.**InitializeDBCommand**(*argv*)
Create the database using the configuration from the .ini file passed as a positional argument.

run()
Run the `initialize_db` method or display the parser help message if the *config_uri* argument is missing.

Returns `initialize_db` method or 2 (missing argument error)

initialize_db()
Initialize the database schema and insert default values.

Returns 0 (OK) or 1 (abnormal termination error)

anuket.scripts.backupdb – Database backup

Script to backup the Anuket database.

class anuket.scripts.backupdb.**BackupDBCommand**(*argv*)
Dump the database for backup purpose.

Supported database: SQLite.

run()
Run the `backup_db` method or display the parser help message if the *config_uri* argument is missing.

Returns `backup_db` method or 2 (missing argument error)

backup_db()
Dump the database and then compress and save the file.

Returns 0 (OK) or 1 (abnormal termination error)

anuket.scripts.upgradedb – Database upgrade

Script for upgrading the Anuket database with Alembic.

class anuket.scripts.upgradedb.**UpgradeDBCommand**(*argv*)
Upgrade the database using the configuration from the .ini file passed as a positional argument.

run()
Run the `upgrade_db` method or display the parser help message if the *config_uri* argument is missing.

Returns `upgrade_db` method or 2 (missing argument error)

upgrade_db()
Upgrade the database to the head revision with Alembic.

Returns 0 (OK) or 1 (abnormal termination error)

6.1.4 anuket.lib – Internal libraries

Internal library for the Anuket application.

anuket.lib.alembic_utils – Alembic utilities

Alembic utilities to use with database scripts.

anuket.lib.alembic_utils.**get_alembic_settings**(*config_uri*)
Get alembic settings from the config file.

Parameters **config_uri** – an .ini config file

Returns an `alembic.config.Config` object

anuket.lib.alembic_utils.**get_alembic_revision**(*config_uri*)
Check the existence of an alembic revision in the database. If the database is versioned, then return the current revision.

Parameters **config_uri** – an .ini file

Returns the alembic revision value or None

anuket.lib.i18n – Translation library

Define a message factory and a locale negotiator.

anuket.lib.i18n.**locale_negotiator**(*request*)
Return a locale name by looking at the `Accept-Language` HTTP header.

Parameters **request** – a `pyramid.request` object

Returns the language code

anuket.lib.validators – FormEncode Validators

FormEncode validators.

class anuket.lib.validators.**FirstNameString**(**args*, ***kw*)

Expand the validators.String class tu return a capitalised value with excessives inner whitespaces removed.

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)

tooLong: Enter a value not more than %(max)i characters long

tooShort: Enter a value %(min)i characters long or more

class anuket.lib.validators.**LastNameString**(**args*, ***kw*)

Expand the validators.String class tu return a value with excessives inner whitespaces removed.

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)

tooLong: Enter a value not more than %(max)i characters long

tooShort: Enter a value %(min)i characters long or more

class anuket.lib.validators.**UsernamePlainText**(**args*, ***kw*)

Expand the validators.PlainText class tu return a lowercased value with all whitespaces removed.

Messages

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

empty: Please enter a value

invalid: Enter only letters, numbers, or _ (underscore)

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

class anuket.lib.validators.**UniqueAuthUsername**(**args*, ***kw*)

Unique username validator.

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)

not_unique_username: This username is already used

validate_python(*values*, *state*)

Check for the uniqueness of *username*.

class anuket.lib.validators.**UniqueAuthEmail**(**args*, ***kw*)
Unique email validator.

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)

not_unique_email: This email is already used

validate_python(*values*, *state*)
Check for the uniqueness of *email*.

class anuket.lib.validators.**SecurePassword**(**args*, ***kw*)
Secure password validator.

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)

not_secure: This password is not secure

tooLong: Enter a value not more than %(max)i characters long

tooShort: Enter a value %(min)i characters long or more

validate_python(*value*, *state*)
Use cracklib to check the strenght of passwords.

6.1.5 Others

anuket.security – Security

Authentification related utilities.

anuket.security.**groupfinder**(*user_id*, *request*)
Groupfinder callback for authentification policy.

Return the groupname (principal) of an authenticated user form the database. Return None if the user do not exist.

Parameters

- **user_id** (*integer*) – the id of the authenticated user
- **request** – a pyramid.request object

Returns the user groupname or None

anuket.subscribers – Subscribers

Pyramid event subscribers.

anuket.subscribers.**add_renderer_globals**(*event*)
Renderers globals event subscriber.

Add globals to the renderer. Add _, *localizer* and *brand_name* globals.

Parameters **event** – a `pyramid.event.BeforeRender` object

`anuket.subscribers.`**`add_localizer`**(*event*)
Localization event subscriber.

Automaticaly translate strings in the templates.

Parameters **event** – a `pyramid.event.NewRequest` object

`anuket.subscribers.`**`add_csrf_validation`**(*event*)
CSRF validation event subscriber.

If the POST forms do not have a CSRF token, or an invalid one then user is logged out and the forbident view is called.

Parameters **event** – a `pyramid.event.NewRequest` object

Raises **HTTPForbidden** if the CSRF token is None or invalid

CHAPTER

SEVEN

INDEX AND GLOSSARY

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

PYTHON MODULE INDEX

a