pyramid_fullauthDocumentation

Release 0.6.0

Grzegorz Śliwiński

Nov 25, 2018

Contents

Pyramid fullauth's provides full user registration and management functionality for pyramid based web applications.

CHAPTER 1

Package status

CHAPTER 2

Package resources

- Bug tracker: https://github.com/fizyk/pyramid_fullauth/issues
- Documentation: http://pyramid-fullauth.readthedocs.org/
- PyPI: https://pypi.python.org/pypi/pyramid_fullauth

CHAPTER 3

Installation

```
pip install pyramid_fullauth
```

or add **pyramid_fullauth** to your **setup.py** requirements.

CHAPTER 4

Tests

You'll need: packages defined in extra_requires[tests] to run tests, and then:

```
py.test
```

CHAPTER 5

Contents

5.1 Basics

`pyramid_fullauth` provides models and actions that allows to register and log in user as well as reset password functionality. It does not provide ability to send appropriate emails, that have to be covered by subscribing to appropriate events emitted by plugin.

Note: By default, all actions are unrestricted (have permissions set to pyramid.security.NO_PERMISSION_REQUIRED, that way setting default permission in your pyramid app would allow the user to log in, register without the need to being logged in to the system

5.1.1 Simple usage

If You have a `sqlalchemy.url` key in the config file In Your pyramid application configuration section just add those two lines:

```
config.include('pyramid_basemodel')
config.include('pyramid_fullauth')
```

And that's it, this is the most simple usage of this plugin. To register just go to the **/register** url and You will see the form with which You can register. Login in is performed on **/login** page

pyramid_fullauth uses under the hood **pyramid_yml** to include configuration defaults defined in yaml file, and to override them, you'd have to employ pyramid_yml on your own into the project.

5.1.2 Events and event interfaces

Plugin emits events while handling requests:

```
BeforeRegister
AfterRegister
AfterActivate
AfterResetRequest
AfterReset
AlreadyLoggedIn
BeforeLogIn
AfterLogIn
```

Events can be found in the *pyramid_fullauth.events* package.

Read the Using Events chapter of Pyramid's documentation to see how to add an event subscriber to Your application and handle those events.

5.1.3 Configuration

Note: Plugins uses tzf.pyramid_yml for its configuration settings

Plugin, by default works on these assumptions:

```
# Copyright (c) 2013 - 2016 by pyramid_fullauth authors and contributors <see AUTHORS file>
#
# This module is part of pyramid_fullauth and is released under
# the MIT License (MIT): http://opensource.org/licenses/MIT

# pyramid_fullauth's default configuration
fullauth:
    check_csrf: True    # Whether login processes should work with csrf token, or without
    register:
        password:
            require: True   # set to false to not read password during register, and generate random one
            length_min: 6   # will be used either to check password's length or generate this length random password
            confirm: True   # by default there will be a field for password confirmation on the registration form
    AuthTkt:
        secret: fullauth_psst   # default secret used to hash auth_tk cookie
        hashalg: sha512         # default authentication policy hash algorithm
    login:
        cookie_max_age: 2592000     # 30 days
    redirects: # route name, to redirect to. If False, then redirects just to /
        logout: False
    session: # session factory settings
        # factory key is responsible for providing full path to factory class (module.submodule.SessionFactory)
        factory: pyramid.session.UnencryptedCookieSessionFactoryConfig
        # settings are key: value pairs of all factory initialize attributes
        settings:
            secret: THATS_NOT_SECRET_ITS_A_SECRET
```

Note: For alternative values of the settings above look at config.{env}.yml configurations found in tests.config

directory.

5.1.4 Fullauth data models

pyramid_fullauth comes with SQLAlchemy models to maintain the user data.

- *pyramid_fullauth.models.User* - is the base model with most relevant user data
- *pyramid_fullauth.models.Group* - allows grouping user in permission groups
- *pyramid_fullauth.models.AuthenticationProvider* - is where the 3rd party authentication system identifiers for different providers lives

Fullauth models are based on declarative_base defined in pyramid_basemodel and functionality uses Session object provided by basemodel.

To connect fullauth's models to your database, it is required to base your own models on the same declarative_base. It can be achieved by either using those provided by pyramid_basemodel or patching them with your own.

Last thing is updating the database. If you're using alembic for that, remember to import fullauth models in alembic's env,py or in common place for your model. If models won't be imported while running alembic commands, they won't be seen by alembic.

5.1.5 Request object additional methods

Request object gets these methods:

- *login_perform()* - performs login action
- `user()` - returns logged in user or None
- *logout()* - logs user out

5.1.6 CSRF Check

CSRF can be turned on/off for fullauth views by modifying *fullauth.check_csrf* key. It's turned on by default.

pyramid_fullauth extends pyrmid's check_csrf predicate in that way, that you can turn it on and off, and when check fails, it raises `HTTPUnauthorized` exception instead of returning False, which gives usually 404 Not Found error

5.2 Advanced configuration

5.2.1 Authentication policy

registerlogin uses **AuthTktAuthenticationPolicy**. It's default settings are stored within config as:

```
registerlogin:
    AuthTkt:
        secret: fullauth_psst # default secret used to hash auth_tk cookie
        hashalg: sha512             # default authentication policy hash algorithm
        timeout: 2              # (optional) number of seconds for which an auth␣
↪ticket will be valid
        reissue_time: 0.2  # (optional) number of seconds that must pass before an␣
↪authentication token cookie is automatically reissued as the result of a request␣
↪which requires authentication
```

(continues on next page)

(continued from previous page)

Note: **timeout** and **reissue_time** settings indicate after which period of time user will be logged out in case of inactivity. If not included in your AuthTktAuthenticationPolicy config, default value for them will be None. To get a better insight on how they work when they are set, look at **tests.test_login**: **test_automatic_logout** and **test_automatic_logout_not_expired** test cases.

See also:

For more information about additional settings that could be included in your AuthTktAuthenticationPolicy as well as how to set optimal values for timeout and reissue_time please see `AuthTktAuthenticationPolicy`.

Warning: **callback** setting is already defined by registerlogin as *pyramid_fullauth.auth.groupfinder()*.

Note: To restrict subdomain applications from using the same cookie, use **registerlogin.AuthTkt.wild_domain setting**, and set it to False. This will restrict emitted cookies to current domain only. You can also change settings as **registerlogin.AuthTkt.cookie_name** and **registerlogin.AuthTkt.secret** to make sure, your apps will use different cookie names and salts.

5.2.2 Authentication Providers

Might happen, that the project needs to identify what authentication providers is user using (Might use e.g facebook, google, email, some OpenID). That's what the user.providers relation is for.

It stores data needed to authenticate with different providers for each user, but the exception is email, where user is identified by id in a system. Each of the social providers entry gets added by connecting user account with given social network, and the email entry during standard registration or during reset password.

5.2.3 ACL

`pyramid_fullauth` package provides also a basic ACL Mixin for your RootFactory. it contains basic acl definition as well as init method.

See *pyramid_fullauth.auth.BaseACLRootFactoryMixin*

5.2.4 Events

Plugin emits several events throughout the registration process, login and several other actions.

All of them, along with details description can be found in the *pyramid_fullauth.events* package.

Read the Using Events chapter of Pyramid's documentation to see how to add an event subscriber to Your application and handle those events.

5.2.5 Session Factory

pyramid_fullauth allows you to connect custom session factory within application, by default, it uses pyramid's `UnencryptedCookieSessionFactoryConfig()`, but using different session factory is just a matter of appropriate settings in **fullauth.session**. See *Configuration* section on how to configure.

More on sessions and session factory can be read in _ Sessions chapter of Pyramid's documentation

5.3 Social register and login

Module provides register and login funcion using social sites such as facebook or twitter. Those actions use velruse library for unification of different authentication providers. Now only facebook and twitter were tested, but this library offers many more.

5.3.1 Configuration

Plugin creates login/register views for auth providers provided in config file. Below is example for facebook, you can check required configuration for other providers here: https://velruse.readthedocs.org/en/latest/

```
fullauth:
    social:
        facebook:
            consumer_key : some_key
            consumer_secret : some_secret
            scope : email,offline_access # some providers requires additional␣
↪information about user data our application wants from provider
```

5.3.2 Usage

In default views for register and login there are links in **Social login** section. If you want to show those buttons somewhere else you can do this by importing function for creating urls:

```
<%namespace name="social" file="pyramid_fullauth:resources/helpers/social.mako"␣
↪import="social_auth_uri"/>
```

and puting one of those links in your templates:

```
<a href="${social.social_auth_uri('facebook', scope=request.config.fullauth.social.
↪facebook.scope)}">Connect with facebook</a>
<a href="${social.social_auth_uri('twitter')}">Connect with twitter</a>
```

5.3.3 Events

Plugin emits three events only for social login: *events*

5.4 Narrative

5.4.1 Social login process

When user clicks on 'Connect to facebook' link from any page he already is, he is redirect to social page to grant us access to his social data. If this succeed, page redirects user to view with context 'velruse.AuthenticationComplete' and provides data from social page. View register_social is fired and few scenarios are available. When user is already logged in, system connects this user's account with social account. Same happens when user with email provided by social site exists. When there is no user to connect him with social site, we need to create new user. When social site do not provide email of user create user with fake email based on social user id and domain of social site, otherwise create user with data provived by social site.

When user is created or retrieved form session or database, then log him in and redirect to page he came from (or index). Result of this action is always an activated, logged in user.

5.5 Customisation

To overwrite templates provided in `pyramid_fullauth` just add this line to your project's init script:

```
config.override_asset(
    to_override='pyramid_fullauth:resources/templates/layout.mako',
    override_with='mypackage:path/to/template/layout.html')
```

You can overwrite all templates separately, all by a group, for more information, read: http://docs.pylonsproject.org/projects/pyramid/en/latest/narr/assets.html#overriding-assets

5.5.1 Form inclusion

You might want to just include form on some pages. all form templates that can be included are prefixed with '_form':

```
<%include file="pyramid_fullauth:resources/templates/_form.login.mako"/>
```

5.6 Api

Here's the api for pyramid_fullauth

5.6.1 models

Models needed for registration, and user servicing.

class pyramid_fullauth.models.**User**(**args*, ***kwargs*)

Bases: pyramid_fullauth.models.mixins.password.UserPasswordMixin, pyramid_fullauth.models.mixins.email.UserEmailMixin, sqlalchemy.ext.declarative.api.Base

User object.

Switch possible email and new_email kwarg into new column attribute names.

delete()

Perform soft delete action. along with checking if it's super admin, or not.

Rises pyramid_fullauth.exceptions.DeleteException if you try to delete last super admin.

Note: You should use this method to delete users

is_active
Check if user is active.

Returns Returns False if user account is not active (or deleted).

Return type bool

provider_id(*provider*)
Return provider identification for give user.

Parameters **provider** (*str*) – provider name

Returns provider identification

Return type str

validate_is_admin(*_*, *value*)
Validate is_admin value, we forbid the deletion of the last superadmin.

Note: More about simple validators: http://docs.sqlalchemy.org/en/latest/orm/mapper_config.html#simple-validators

Raises **AttributeError** – Information about an error

class pyramid_fullauth.models.**Group**(***kwargs*)
User group object.

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

class pyramid_fullauth.models.**AuthenticationProvider**(***kwargs*)
Model to store authentication methods for different providers.

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

models.mixins

Mixin's main module.

class pyramid_fullauth.models.mixins.**UserPasswordMixin**
Authentication field definition along with appropriate methods.

password = Column(None, Unicode(length=128), table=None, nullable=False)
password field

_hash_algorithm = Column('hash_algorithm', Enum('md5', 'sha1', 'sha224', 'sha256', 'sha
hash_algorithm field

_salt = Column('salt', Unicode(length=128), table=None, nullable=False)
salt field

reset_key = Column(None, String(length=255), table=None)
reset key field

check_password(*password*)
Check if password correspond to the saved one.

Parameters **password** (*str*) – password to compare

Returns True, if password is same, False if not

Return type bool

classmethod hash_password(*password*, *salt*, *hash_method*)
Produce hash out of a password.

Parameters

- **password** (*str*) – password string, not hashed
- **salt** (*str*) – salt
- **hash_method** (*callable*) – a hash method which will be used to generate hash

Returns hashed password

Return type str

password_validator(_, *password*)
Validate password.

Password validator keeps new password hashed. Rises Value error on empty password

Parameters

- **key** (*str*) – field key
- **password** (*str*) – new password

Returns hashed and salted password

Return type str

Raises pyramid_fullauth.exceptions.EmptyError

Note: If you're using this Mixin on your own User object, don't forget to add a listener as well, like that:

```
from sqlalchemy.event import listen

listen(User.password, 'set', User.password_listener, retval=True)
```

Note: For more information on Attribute Events in sqlalchemy see:

sqlalchemy.orm.events.AttributeEvents.set()

---

class pyramid_fullauth.models.mixins.**UserEmailMixin**(**args*, ***kwargs*)
User email fields and functionality.

Switch possible email and new_email kwarg into new column attribute names.

email
Email field comparator.

new_email
Email field comparator.

email_change_key = Column(None, String(length=255), table=None)

change_email()
Change email after activation.

We don't clear new email field because of validator of email which won't allow to None value.

set_new_email(*email_new*)
Set new email and generate new email change hash.

Parameters **email_new** (*str*) – email address

Returns generated email_change_key

Trype str

validate_email(*_*, *address*)
Validate email addresses.

Note: See pyramid docs about simple validators

Parameters

- **key** (*str*) – field key
- **address** (*str*) – email address

Raises

- ***EmailValidationError*** –
- ***EmptyError*** –

models.extensions

CaseInsensitive comparator for sqlalchemy models.

class pyramid_fullauth.models.extensions.**CaseInsensitive**(*word*)
Hybrid value representing a lower case representation.

Initialise comparator object.

5.6.2 auth

auth module provides basic pyramid auth helpers

Auth related methods and classes.

class pyramid_fullauth.auth.**BaseACLRootFactoryMixin**(*request*)
ACL list factory Mixin.

__acl__ is the attribute which stores the list.

Returns tuple (Allow|Deny, Group name, Permission)

Return type list

Note: Can be converted later to database stored (sqlalchemy session is accessible through request.db)

Assing request as instance attribute.

pyramid_fullauth.auth.**groupfinder**(*userid*, *request*)
Read all user's groups.

Note: Adds **s:inactive** group to users who has not activated their account, and **s:user** group to those, who did. If user has is_admin flag, he gets **s:superadmin** group set

Might be useful, when you want restrict access to some parts of your application, but still allow log in, and access to some other parts.

Parameters

- **userid** (*int*) – user identity
- **request** (*pyramid.request.Request*) – request object

Returns list of groups

Return type list

5.6.3 events

pyramid_fullauth emits these events during whole cycle.

Registration

class pyramid_fullauth.events.**BeforeRegister**(*request*, *user*, *errors*)
Execute custom code at the start of registration process.

Note: User object is not yet in session.

Initialize event.

Parameters

- **request** (*pyramid.request.Request*) – request object
- **user** (pyramid_fullauth.models.User) – user object
- **errors** (*dict*) – a dictionary with wrong/not submitted fields with format - fields for which error occured: error message

class pyramid_fullauth.events.**AfterRegister**(*request*, *user*, *response_values*)
Add custom post-processing code in registration process.

Can be used to add e.g. e-mail sending with registration links.

Note: User object is already in a session.

Note: Action emitting this event, should catch all HTTPRedirection that might be risen in event listener.

Warning: If HTTPRedirection is risen from event listener, then response_values will not be used!

Initialize event.

Parameters

- **request** (*pyramid.request.Request*) – request object
- **user** (pyramid_fullauth.models.User) – user object
- **response_values** (*dict*) – a dictionary with response values

Account activation

class pyramid_fullauth.events.**AfterActivate**(*request*, *user*)
Add custom post-processing logic after user gets activated.

Note: Action emitting this event, should catch all HTTPRedirection that might be risen in event listener.

Initialize event.

Parameters

- **request** (*pyramid.request.Request*) – request object
- **user** (pyramid_fullauth.models.User) – user object

Password reset

class pyramid_fullauth.events.**AfterResetRequest**(*request*, *user*)
Add custom post-processing after user sends request to reset password.

Note: Action emitting this event, should catch all HTTPRedirection that might be risen in event listener.

Initialize event.

Parameters

- **request** (*pyramid.request.Request*) – request object
- **user** (pyramid_fullauth.models.User) – user object

class pyramid_fullauth.events.**AfterReset**(*request*, *user*)
Add custom post-processing after the actual reset-password process.

Note: Action emitting this event, should catch all HTTPRedirection that might be risen in event listener.

Initialize event.

Parameters

- **request** (*pyramid.request.Request*) – request object
- **user** (pyramid_fullauth.models.User) – user object

Login

class pyramid_fullauth.events.**AlreadyLoggedIn**(*request*)
Allow execute custom logic, when logged in user tries to log in again.

Note: Action emitting this event, should catch all HTTPRedirection that might be risen in event listener.

Initialize event.

Parameters **request** (*pyramid.request.Request*) – request object

class pyramid_fullauth.events.**BeforeLogIn**(*request*, *user*)
Add custom logic before user gets logged in.

Note: Action emitting this event, should catch all AttributeError that might be risen in event listener. User param set to None when user is not found or request method is GET.

Initialize event.

Parameters

- **request** (*pyramid.request.Request*) – request object
- **user** (pyramid_fullauth.models.User) – user object

class pyramid_fullauth.events.**AfterLogIn**(*request*, *user*)
Add custom logic after user logs in.

Initialize event.

Parameters

- **request** (*pyramid.request.Request*) – request object
- **user** (pyramid_fullauth.models.User) – user object

Social registration/login

class pyramid_fullauth.events.**BeforeSocialRegister**(*request*, *user*, *profile*)
Adds custom logic before the social login process start.

Initialize base events.

Parameters

- **request** (*pyramid.request.Request*) – request object
- **user** (pyramid_fullauth.models.User) – user object
- **profile** (*dict*) – a dictionary with profile data

class pyramid_fullauth.events.**AfterSocialRegister**(*request*, *user*, *profile*, *response_values*)

Add custom logic after user registers through social network.

Note: Action emitting this event, should catch all HTTPRedirection that might be risen in event listener.

Initialize event.

Parameters

- **request** (*pyramid.request.Request*) – request object
- **user** (pyramid_fullauth.models.User) – user object
- **profile** (*dict*) – a dictionary with profile data
- **response_values** (*dict*) – a dictionary with response values

class pyramid_fullauth.events.**AfterSocialLogIn**(*request*, *user*, *profile*)

Custom logic after user logs in through social network.

Note: Action emitting this event, should catch all HTTPRedirection that might be risen in event listener.

Initialize base events.

Parameters

- **request** (*pyramid.request.Request*) – request object
- **user** (pyramid_fullauth.models.User) – user object
- **profile** (*dict*) – a dictionary with profile data

class pyramid_fullauth.events.**SocialAccountAlreadyConnected**(*request*, *user*, *profile*, *response_values*)

Event raised when social account is already connected to some other user.

Allow to add custom logic, when someone tries to connect social account to second user in application.

Note: Action emitting this event, should catch all HTTPRedirection that might be risen in event listener.

Initialize event.

Parameters

- **request** (*pyramid.request.Request*) – request object
- **user** (pyramid_fullauth.models.User) – user object
- **profile** (*dict*) – a dictionary with profile data
- **response_values** (*dict*) – a dictionary with response values

5.6.4 exceptions

Fullauth's exceptions definition.

exception pyramid_fullauth.exceptions.**DeleteException**

Exception risen when the user can't be deleted.

exception pyramid_fullauth.exceptions.**EmailValidationError**
Exception thrown, when there's incorrect email provided.

exception pyramid_fullauth.exceptions.**EmptyError**
Thrown whenever user is trying to set empty value.

exception pyramid_fullauth.exceptions.**PasswordConfirmMismatchError**
Thrown when there's a mismatch with cpassword_confirm.

exception pyramid_fullauth.exceptions.**ShortPasswordError**
Thrown when password doesn't meet the length requirement.

exception pyramid_fullauth.exceptions.**ValidateError**
Base of every validate error in pyramid_fullauth.

5.6.5 routing.predicates

Routing predicate definitions.

class pyramid_fullauth.routing.predicates.**CSRFCheckPredicate**(*val*, *config*)
Bases: pyramid.predicates.CheckCSRFTokenPredicate

Run csrf check dependant on configuration.

Note: Raises HTTPUnauthorized exception if check fails.

Raises pyramid.httpexceptions.HTTPUnauthorized

Returns True if check succeeds or turned off.

Return type bool

pyramid_fullauth.routing.predicates.**change_email_hash**(*info*, *_*)
Check whether change email hash is correct.

Parameters

- **info** (*dict*) – pyramid info dict with path fragments and info
- **_** (*pyramid.request.Request*) – request object

Returns whether change email hash exists or not

Return type bool

pyramid_fullauth.routing.predicates.**reset_hash**(*info*, *_*)
Check whether reset hash is correct.

Parameters

- **info** (*dict*) – pyramid info dict with path fragments and info
- **_** (*pyramid.request.Request*) – request object

Returns whether reset hash exists or not

Return type bool

5.6.6 request

These method gets added to each `pyramid.request.Request` object.

`pyramid_fullauth.request.`**`login_perform`**(*request*, *user*, *location=None*, *remember_me=False*)

Perform login action.

Parameters

- **request** (*pyramid.request.Request*) – a request object
- **user** (`pyramid_fullauth.models.User`) – a user object
- **location** (*str*) – where user should be redirected after login
- **remember_me** (*bool*) – if True set cookie max_age to one month (60 * 60 * 24 * 30 seconds)

Returns redirect exception

Return type pyramid.httpexceptions.HTTPSeeOther

`pyramid_fullauth.request.`**`logout`**(*request*)

Log user out.

Parameters **request** (*pyramid.request.Request*) – a request object

`pyramid_fullauth.request.`**`request_user`**(*request*)

Return user object.

When called for the first time, it queries for user, which is later available as a pure property overriding this method. See `pyramid_fullauth.includeme()` for logic behind property.

Returns logged in user object, or None

Return type *pyramid_fullauth.models.User*

5.6.7 tools

Additional tools used in pyramid_fullauth.

`pyramid_fullauth.tools.`**`password_generator`**(*length*, *chars='abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQR -./:;<=>?@[\\]^_`{|}~'*)

Generate random password.

> **Warning:** TODO: tests!

Parameters

- **length** (*int*) – length of a password to Generates
- **chars** (*list*) – list of characters from which to choose

Returns password

Return type str

`pyramid_fullauth.tools.`**`validate_passsword`**(*request*, *password*, *user=None*)

Validate password properly.

Note: If no user provided, password is just validated

Parameters

- **request** (*pyramid.request.Request*) – request object
- **password** (*str*) – password to be set
- **user** (pyramid_fullauth.models.User) – user object

Raises pyramid_fullauth.exceptions.ValidateError

5.7 Contribute to pyramid_fullauth

Thank you for taking time to contribute to pyramid_fullauth!

The following is a set of guidelines for contributing to pyramid_fullauth. These are just guidelines, not rules, use your best judgment and feel free to propose changes to this document in a pull request.

5.7.1 Bug Reports

1. Use a clear and descriptive title for the issue - it'll be much easier to identify the problem.
2. Describe the steps to reproduce the problems in as many details as possible.
3. If possible, provide a code snippet to reproduce the issue.

5.7.2 Feature requests/proposals

1. Use a clear and descriptive title for the proposal
2. **Provide as detailed description as possible**
    - Use case is great to have
3. There'll be a bit of discussion for the feature. Don't worry, if it is to be accepted, we'd like to support it, so we need to understand it thoroughly.

5.7.3 Pull requests

1. Start with a bug report or feature request
2. Use a clear and descriptive title
3. Provide a description - which issue does it refers to, and what part of the issue is being solved
4. Be ready for code review :)

5.7.4 Commits

1. Make sure commits are atomic, and each atomic change is being followed by test.
2. If the commit solves part of the issue reported, include *refs #[Issue number]* in a commit message.
3. If the commit solves whole issue reported, please refer to Closing issues via commit messages for ways to close issues when commits will be merged.

5.7.5 Coding style

1. All python coding style are being enforced by Pylama and configured in pylama.ini file.
2. Additional, not always mandatory checks are being performed by QuantifiedCode

5.8 CHANGELOG

5.8.1 unreleased

- properly lint code through pylint an fix found issues

5.8.2 0.6.0

- increased the size of password and salt fields to 128 characters each
- default password hashing algorithm is sha256

5.8.3 0.5.0

- full python3 compatibility, since velruse migrated to py3 enabled requests-oauth
- require velruse 1.1.1
- run tests with sqlalchemy 1.0.x
- small updates to conform with new linters versions embedded in pylama

5.8.4 0.4.1

- fixed spelling for error message when user does not exist while trying to reset password.
- require pyramid_basemodel at least version 0.3

5.8.5 0.4.0

- python 3 compatibility (without oauth2 though)
- cleared use of deprecated function *pyramid.security.authenticated_userid* in favour of *pyramid.request.Request.authenticated_userid* attribute.
- make email fields case insensitive by using hybrid properties and CaseInsensitive comparator for model.

5.8.6 0.3.3

- Fix issue where groupfined was returning empty list instead of None when user did not existed

5.8.7 0.3.2

- catch all HTTPRedirect instead of just HTTPFound.
- redirect with HTTPSeeOther instead of HTTPFound where applicable.

5.8.8 0.3.1

- fixes MANIFEST.in to include yaml files - fixes #33.

5.8.9 0.3.0

Features

- configure root factory if it hasn't been already done
- configure session factory only if it hasn't been configured before
- configure authorization policy only if it hasn't been configured before
- configure authentication policy only if it hasn't been configured before
- logged in user will be redirected always away from login page
- views reorganisation - grouping by their function
- replaced force_logout decorator with logout request method
- small login view simplification

tests

- rewritten tests to use pytest_pyramid
- unified session with pyramid_basemodel's
- parametrize tests against two most recent pyramid versions and sqlalchemy
- **turned on pylama to check code with linters:**
  - pep8
  - pep257
  - pyflakes
  - mccabe
- add pytest-dbfixtures, and run tests against postgresql and mysql as well
- drop python 2.6 from tests
- 100% test coverage

5.8.10 0.2.3

- weaker pyramid_yml requirements. Use `registry['config']` instead of `request.config` which gets added only when explicitly including tzf.pyramid_yml package.
- remove default_config with permission set for forbidden views. Throwning errors in pyramid 1.5a3
- remove lazy='load' for relationship between AuthenticationProvider and User models as it was incorrect. Fixes error while using with sqlalchemy 0.9

5.8.11 0.2.2

- copy all headers when login user. fixes issue, when headers set in AfterLogin event would not get passed

5.8.12 0.2.1

- fixed csrf_check in password:reset:continue action
- updated translation files

5.8.13 0.2.0

- migrated tests to py.test
- removed nose and lxml from test requirements
- extracted UserEmailMixin from User model
- validation exception improvements
- set licensing to MIT License
- fixed general error message for register_POST processing
- activate action no longer gives 404 error after first use. Default is message about token being invalid or used [veronicazgirvaci]
- **extending csrf_check predicate:**
  - Can be turned on/off in settings.
  - Failed check rises 401 Unauthorised error

Backwards Incompatibilities

- token variable is changed into csrf_token in fullatuh views
- view no longer returns error messages on failed csrf token. Rises 401 Unauthorised error instead.

5.8.14 0.1.0

- add localize to requirements. Ability to translate registerlogin communicates
- ability to set custom session factory [with Veronica Zgirvaci help]
- moved password validation to one place

CHAPTER 6

Python 3

Python 3 support is limited to python 3.3 and 3.4. Unfortunately, python 3.x support does not include social authentication.

CHAPTER 7

License

Python Module Index

p

Index

Symbols

A

B

C

D

E

G

H

I

L

N

P

R

S

U

V