# ziggurat_foundations Documentation

***Release 0.7.1***

**Marcin Lulek - Webreactor.eu**

February 19, 2017

Contents


Top layer to make authentication, resource ownership and permission management fast, simple and easy. In summary, Ziggurat Foundations (Zigg), is a set of framework agnostic set of sqlalchemy classes, but most of the documentation refers to using it within pyramid. It is the perfect solution for handling complex login and user management systems, from e-commerce systems, to private intranets or large (and small) CMS systems. It can easily be extended to support any additional features you may need (explained further in the documentation)

**Hint:** Ziggurat Foundations aims to simplify user, permission and resource management, allowing you to concentrate on application development, as opposed to developing your own user and permission based models/code.

**DOCUMENTATION**: http://readthedocs.org/docs/ziggurat-foundations/en/latest/

**BUG TRACKER**: https://github.com/ergo/ziggurat_foundations

**DOCUMENTATION REPO**: https://github.com/ergo/ziggurat_foundations/docs

Contents:

CHAPTER 1

# Overview of functionality

Ziggurat Foundations supplies a set of *sqlalchemy mixins* that can be used to extend Zigguart's base models, within your application. The aim of this project is to supply set of generic models that cover the most common needs in application development when it comes to authorization, user management, permission management and resource management, using flat and tree like data structures. Ziggurat is stable and well tested, meaning it can plug straight in to your application and dramatically reduce development time, as you can concentrate on your application code.

Ziggurat supplies extendable, robust and well tested models that include:

- User - base for user accounts
- Group - container for many users
- Resource - Arbitrary database entity that can represent various object hierarchies - blogs, forums, cms documents, pages etc.
- Resource trees management

Ziggurat provides standard functions that let you:

- Assign arbitrary permissions directly to users (ie. access certain views)
- Assign users to groups
- Assign arbitrary permissions to groups
- Assign arbitrary resource permissions to users (ie. only user X can access private forum)
- Assign arbitrary resource permissions to groups
- Manage nested resources with tree service
- Assign a user o an external identity (such as facebook/twitter)
- Change users password and generate security codes
- Manage the sign in/sign out process (pyramid extension)
- Example root factory for assiginging permissions per request (for pyramid)

Functions that we supply between those patterns allow for complex and flexible permission systems that are easly understandable for non-technical users, whilst at the same time providing a stable base for systems coping with millions of users.

Due to the fact that we supply all models as mixins, the base of Ziggurat can be very easily extended, for example if you wanted to extend the user model to include a column such as "customer_number", you can simply do the following:

```
class User(UserMixin, Base):
    customer_number = Column(Integer)
```

**Warning:** **NEVER** create your ziggurat database models directly without the use of alembic. we provide alembic migrations for all models supplied (and will continue to do so for future releases). This ensures a smooth and documented upgrade of the database design.

CHAPTER 2

# Configuring Ziggurat Foundations

## Installation and initial migration

Install the package:

```
$ pip install ziggurat_foundations
```

You will also need to install one of supported bcrypt backends like *bcrypt* or *py-bcrypt* (this is default solution ziggurat_foundations expects).

```
$ pip install bcrypt
```

Now it's time to initialize your model structure with alembic.

You will first need to install alembic:

```
$ pip install alembic>=0.7.0
```

After you obtain recent alembic you can now run your migrations against database of your choice.

**Warning:** It is *critical* that you use alembic for migrations, if you perform normal table creation like metadata.create_all() with sqlalchemy you will not be able to perform migrations if database schema changes for ziggurat and some constraints *will* be missing from your database even if things will appear to work fine for you.

First you will need to create alembic.ini file with following contents:

```
[alembic]
script_location = ziggurat_foundations:migrations
sqlalchemy.url = driver://user:pass@host/dbname

[loggers]
keys = root,sqlalchemy,alembic

[handlers]
keys = console

[formatters]
keys = generic

[logger_root]
level = WARN
handlers = console
qualname =
```

```
[logger_sqlalchemy]
level = WARN
handlers =
qualname = sqlalchemy.engine

[logger_alembic]
level = INFO
handlers =
qualname = alembic

[handler_console]
class = StreamHandler
args = (sys.stderr,)
level = NOTSET
formatter = generic

[formatter_generic]
format = %(levelname)-5.5s [%(name)s] %(message)s
datefmt = %H:%M:%S
```

then you can run migration command:

```
$ alembic upgrade head
```

At this point all your database structure should be prepared for usage.

## Implementing ziggurat_foundations within your application

**Warning:** class names like User inside ziggurat_foundations.models namespace CAN NOT be changed because they are reused in various queries - unless you reimplement ziggurat_model_init

We need to *include ALL mixins inside our application* and map classes together so internal methods can function properly.

In order to use the mixins inside your application, you need to include the follwing code inside your models file, to extend your existing models (if following the basic pyramid tutorial):

```
# ... your DBSession and base gets created in your favourite framework ...

import ziggurat_foundations.models
from ziggurat_foundations.models.base import BaseModel
from ziggurat_foundations.models.external_identity import ExternalIdentityMixin
from ziggurat_foundations.models.group import GroupMixin
from ziggurat_foundations.models.group_permission import GroupPermissionMixin
from ziggurat_foundations.models.group_resource_permission import GroupResourcePermissionMixin
from ziggurat_foundations.models.resource import ResourceMixin
from ziggurat_foundations.models.user import UserMixin
from ziggurat_foundations.models.user_group import UserGroupMixin
from ziggurat_foundations.models.user_permission import UserPermissionMixin
from ziggurat_foundations.models.user_resource_permission import UserResourcePermissionMixin
from ziggurat_foundations import ziggurat_model_init

# this is needed for pylons 1.0 / akhet approach to db session
ziggurat_foundations.models.DBSession = DBSession
# optional for folks who pass request.db to model methods
```

```
# Base is sqlalchemy's Base = declarative_base() from your project
class Group(GroupMixin, Base):
    pass

class GroupPermission(GroupPermissionMixin, Base):
    pass

class UserGroup(UserGroupMixin, Base):
    pass

class GroupResourcePermission(GroupResourcePermissionMixin, Base):
    pass

class Resource(ResourceMixin, Base):
    # ... your own properties....

    # example implementation of ACLS for pyramid application
    @property
    def __acl__(self):
        acls = []

        if self.owner_user_id:
            acls.extend([(Allow, self.owner_user_id, ALL_PERMISSIONS,), ])

        if self.owner_group_id:
            acls.extend([(Allow, "group:%s" % self.owner_group_id,
                          ALL_PERMISSIONS,), ])
        return acls

class UserPermission(UserPermissionMixin, Base):
    pass

class UserResourcePermission(UserResourcePermissionMixin, Base):
    pass

class User(UserMixin, Base):
    # ... your own properties....
    pass

class ExternalIdentity(ExternalIdentityMixin, Base):
    pass

# you can define multiple resource derived models to build a complex
# application like CMS, forum or other permission based solution

class Entry(Resource):
    """
    Resource of `entry` type
    """

    __tablename__ = 'entries'
    __mapper_args__ = {'polymorphic_identity': 'entry'}

    resource_id = sa.Column(sa.Integer(),
                            sa.ForeignKey('resources.resource_id',
                                          onupdate='CASCADE',
                                          ondelete='CASCADE', ),
                            primary_key=True, )
```

```
    # ... your own properties....
    some_property = sa.Column(sa.UnicodeText())


ziggurat_model_init(User, Group, UserGroup, GroupPermission, UserPermission,
                UserResourcePermission, GroupResourcePermission, Resource,
                ExternalIdentity, passwordmanager=None)
```

**Hint:** Because some systems can't utilize bcypt password manager you can pass your own passlib compatible password manager to ziggurat_model_init, it will be used instead of creating default one.

CHAPTER 3

# Usage examples

## Basics

Create a user called “Joe”

```
new_user = User(user_name="joe")
DBSession.add(new_user)
# At this point new_user becomes a User object, which contains special
# functions that we will describe in this documentation.
```

Now we have the user Joe, we can generate a security code for the user (useful for sending email validation links). Whilst we are doing this, we can assign him a password

```
new_user.set_password("secretpassword")
new_user.regenerate_security_code()
```

Now we have a user, lets create a group to store users in, lets say Joe is an admin user and so we will create a group called “admins” and add Joe to this group:

```
new_group = Group(group_name="admins")
DBSession.add(new_group)
# Now new_group becomes a Group object we can access
group_entry = UserGroup(group_id=new_group.id, user_id=new_user.id)
DBSession.add(group_entry)
```

So now we have the user Joe as part of the group “admins”, but this on its own does nothing, we now want to let give all members of the admin group permission to access a certain view, for example the view below would only be accessible to users (and groups) who had the permission “delete”:

```
@view_config(route_name='delete_users',
    renderer='templates/delete_users.jinja2',
    permission='delete')
def delete_users(request):
    # do some stuff
    return
```

So we can do this one of two ways, we can either add the “delete” permission directly to the user, or assign the delete permission to a group (that the user is part of)

```
# assign the permission to a group
new_group_permission = GroupPermission(perm_name="delete", group_id=new_group.id)
DBSession.add(new_group_permission)
# or assign the permssion directly to a user
```

```
new_user_permission = UserPermission(perm_name="delete", user_id=new_user.id)
DBSession.add(new_user_permission)
```

Now we move on to resource permissions, adding a resource that the user will own

```
resource = SomeResouce()
DBSession.add(resource)
# Assuming "user" is a User() object
user.resources.append(resource)
```

Here we show a demo fo how to add a custom "read" permission for user "foo" for a given resource:

```
permission = UserResourcePermission()
permission.perm_name = "read"
permission.user_name = "foo"
resource.user_permissions.append(permission)
```

We can now fetch all resources with permissions "edit", "vote":

```
# assuming "user" is a User() object as described as above
user.resources_with_perms(["edit","vote"])
```

If we have a user object, we can fetch all non-resource based permissions for user:

```
user.permissions
```

Given a resource fetching all permissions for user, both direct and inherited from groups user belongs to:

```
resource.perms_for_user(user_instance)
```

Checking "resourceless" permission like "user can access admin panel:

```
request.user.permissions
for perm_user, perm_name in request.user.permissions:
    print(perm_user, perm_name)
```

Checking all permissions user has to specific resource:

```
resource = Resource.by_resource_id(rid)
for perm in resource.perms_for_user(user):
    print(perm.user, perm.perm_name, perm.type, perm.group, perm.resource, perm.owner)
    .... list acls ....
```

List all **direct** permissions that users have for specific resource

```
from ziggurat_foundations.permissions import ANY_PERMISSION
permissions = ResourceService.users_for_perm(
    resource, perm_name=ANY_PERMISSION, skip_group_perms=True)
```

Here is an example of how to connect a user to an external identity provider like twitter:

```
ex_identity = ExternalIdentity()
ex_identity.external_id = XXX
ex_identity.external_user_name = XXX
ex_identity.provider_name = 'twitter.com'
ex_identity.access_token = XXX
ex_identity.token_secret = XXX
new_user.external_identities.append(ex_identity)
```

## Tree Structures

**Warning:** When using *populate_instance* or any other means to set values on resources remember to **NOT** modify *ordering* and *parent_id* values on the resource rows - always perform tree operations via tree service. Otherwise it will confuse the service and it might perform incorrect operations.

Create a tree structure manager:

```
from ziggurat_foundations.models.services.resource_tree import ResourceTreeService
from ziggurat_foundations.models.services.resource_tree_postgres import \
    ResourceTreeServicePostgreSQL

TreeService = ResourceTreeService(ResourceTreeServicePostgreSQL)
```

Create a new resource and place it somewhere:

```
resource = Resource(...)

# this accounts for the newly inserted row so the total_children
# will be max+1 position for new row
total_children = tree_service.count_children(
    resource.parent_id, db_session=self.request.dbsession)

tree_service.set_position(
    resource_id=resource.resource_id, to_position=total_children,
    db_session=self.request.dbsession)
```

Fetch all resources that are parent of resource:

```
tree_service.path_upper(resource.resource_id, db_session=db_session)
```

Fetch all children of a resource limiting the amount of levels to go down, then build a nested dictionary structure out of it:

```
result = tree_service.from_resource_deeper(
    resource_id, limit_depth=2, db_session=db_session)
tree_struct = tree_service.build_subtree_strut(result)
```

Delete some resource and all its descendants:

```
tree_service.delete_branch(resource.resource_id)
```

Move node to some other location in tree:

```
tree_service.move_to_position(
    resource_id=resource.resource_id, new_parent_id=X,
    to_position=Y, db_session=request.dbsession)
```

CHAPTER 4

# Configure Ziggurat with Pyramid Framework

## Examples of permission system building

Root context factories for pyramid provide customizable permissions for specific views inside your appplication. It is a good idea to keep the root factory inside your models file (if following the basic pyramid tutorial). This root factory can be used to allow only authenticated users to view:

```
from ziggurat_foundations.permissions import permission_to_pyramid_acls

class RootFactory(object):
    def __init__(self, request):
        self.__acl__ = [(Allow, Authenticated, u'view'), ]
        # general page factory - append custom non resource permissions
        # request.user object from cookbook recipie
        if request.user:
            # for most trivial implementation

            # for perm in request.user.permissions:
            #     self.__acl__.append((Allow, perm.user.id, perm.perm_name,))

            # or alternatively a better way that handles both user
            # and group inherited permissions via `permission_to_pyramid_acls`

            for outcome, perm_user, perm_name in permission_to_pyramid_acls(
                    request.user.permissions):
                self.__acl__.append((outcome, perm_user, perm_name))
```

This example covers the case where every view is secured with a default “view” permission, and some pages require other permissions like “view_admin_panel”, “create_objects” etc. Those permissions are appended dynamicly if authenticated user is present, and has additional custom permissions.

## Example resource based pyramid context factory that can be used with url dispatch

This example shows how to protect and authorize users to perform actions on resources, you can configure your view to expect “edit” or “delete” permissions:

```
from ziggurat_foundations.permissions import permission_to_pyramid_acls

class ResourceFactory(object):
```

```
    def __init__(self, request):
        self.__acl__ = []
        rid = request.matchdict.get("resource_id")

        if not rid:
            raise HTTPNotFound()
        self.resource = Resource.by_resource_id(rid)
        if not self.resource:
            raise HTTPNotFound()
        if self.resource and request.user:
            # append basic resource acl that gives all permissions to owner
            self.__acl__ = self.resource.__acl__
            # append permissions that current user may have for this context resource
            permissions = self.resource.perms_for_user(request.user)
            for outcome, perm_user, perm_name in permission_to_pyramid_acls(
                    permissions):
                self.__acl__.append((outcome, perm_user, perm_name,))
```

Ziggurat Foundations can provide some shortcuts that help build pyramid applications faster.

**Hint:** This approach will also work properly for all models inheriting from *Resource* class.

## Automatic user sign in/sign out

**ziggurat_foundations.ext.pyramid.sign_in**

This extension registers basic views for user authentication using **AuthTktAuthenticationPolicy**, and can fetch user object and verify it against supplied password.

**Extension setup**

To enable this extension it needs to be included via pyramid include mechanism for example in your ini configuration file:

```
pyramid.includes = pyramid_tm
                   ziggurat_foundations.ext.pyramid.sign_in
```

or by adding the following to your applications __init__.py configurator file (both methods yeild the same result):

```
config.include('ziggurat_foundations.ext.pyramid.sign_in')
```

this will register 2 routes:

- ziggurat.routes.sign_in with pattern */sign_in*
- ziggurat.routes.sign_out with pattern */sign_out*

**Tip:** those patterns can be configured to match your app route patterns via following config keys:

- ziggurat_foundations.sign_in.sign_in_pattern = /custom_pattern
- ziggurat_foundations.sign_in.sign_out_pattern = /custom_pattern

In order to use this extension we need to tell the Ziggurat where User model is located in your application for example in your ini file:

```
ziggurat_foundations.model_locations.User = yourapp.models:User
```

Additional config options for extensions to include in your ini configuration file:

```
# name of the POST key that will be used to supply user name
ziggurat_foundations.sign_in.username_key = login

# name of the POST key that will be used to supply user password
ziggurat_foundations.sign_in.password_key = password

# name of the POST key that will be used to provide additional value that can be used to redirect
# user back to area that required authentication/authorization)
ziggurat_foundations.sign_in.came_from_key = came_from

# If you do not use a global DBSession variable, and you bundle DBSession insde the request
# you need to tell Ziggurat its naming convention, do this by providing a function that
# returns the correct request variable
ziggurat_foundations.session_provider_callable = yourapp.model:get_session_callable
```

If you are using a db_session inside the request, you need to provide a basic function to tell Ziggurat where DBSession is inside the request, you can add the following to your models file (yourapp.model):

```
def get_session_callable(request):
    # if DBSession is located at "request.db_session"
    return request.db_session
    # or if DBSession was located at "request.db"
    # return request.db
```

**Configuring your application views**

Here would be a working form/template used for user authentication and to send info to one of the new views registered by extension (sign_in), you can put this code inside any template, as the action is posted directly to pre-registered Ziggurat views/contexts:

```
<form action="{{request.route_url('ziggurat.routes.sign_in')}}" method="post">
    <!-- "came_from", "password" and "login" can all be overwritten -->
    <input type="hidden" value="OPTIONAL" name="came_from" id="came_from">
    <!-- in the example above we changed the value of "login" to "username" -->
    <input type="text" value="" name="login" <!-- change to name="username" if required --> >
    <input type="password" value="" name="password">
    <input type="submit" value="Sign In" name="submit" id="submit">
</form>
```

In next step it is required to register 3 views that will listen for specific context objects that the extension can return upon form sign_in/sign_out requests:

- **ZigguratSignInSuccess - user and password were matched**
  - contains headers that set cookie to persist user identity, fetched user object, “came from” value
- **ZigguratSignInBadAuth - there were no positive matches for user and password**
  - contains headers used to unauthenticate any current user identity
- **ZigguratSignOut - user signed out of application**
  - contains headers used to unauthenticate any current user identity

**Required imports for all 3 views**

So inside the file you will be using for your Ziggurat views, we need to perform some base imports:

```
from pyramid.security import NO_PERMISSION_REQUIRED
from ziggurat_foundations.ext.pyramid.sign_in import ZigguratSignInSuccess
from ziggurat_foundations.ext.pyramid.sign_in import ZigguratSignInBadAuth
from ziggurat_foundations.ext.pyramid.sign_in import ZigguratSignOut
```

**ZigguratSignInSuccess context view example**

Now we can provide a fuction, based off of the ZigguratSignInSuccess context

```
@view_config(context=ZigguratSignInSuccess, permission=NO_PERMISSION_REQUIRED)
def sign_in(request):
    # get the user
    user = request.context.user
    # actions performed on sucessful logon, flash message/new csrf token
    # user status validation etc.
    if request.context.came_from != '/':
        return HTTPFound(location=request.context.came_from,
                         headers=request.context.headers)
    else:
        return HTTPFound(location=request.route_url('some_route'),
                         headers=request.context.headers)
```

**ZigguratSignInBadAuth context view example**

The view below would deal with handling a failed login

```
@view_config(context=ZigguratSignInBadAuth, permission=NO_PERMISSION_REQUIRED)
def bad_auth(request):
    # The user is here if they have failed login, this example
    # would return the user back to "/" (site root)
    return HTTPFound(location=request.route_url('/'),
                     headers=request.context.headers)
    # This view would return the user back to a custom view
    return HTTPFound(location=request.route_url('declined_view'),
                  headers=request.context.headers)
```

**ZigguratSignOut context view example**

This is a basic view that performs X task once the user has navigated to "/sign_out" (if using the default location provided by Ziggurat), like the view above it can be overwritten/modified to do what ever else you would like.

```
@view_config(context=ZigguratSignOut, permission=NO_PERMISSION_REQUIRED)
def sign_out(request):
    return HTTPFound(location=request.route_url('/'),
                     headers=request.context.headers)
```

## Cofiguring groupfinder and session factories

Now before we can actually use the login system, we need to import and include the groupfinder and session factory inside our application configuration, first off in our ini file we need to add a session secret:

```
# replace "sUpersecret" with  a secure secret
session.secret = sUpersecret
```

Now, we need to configure the groupdiner and authn and authz policy inside the main __init__.py file of our application, like so:

```
from ziggurat_foundations.models import groupfinder

def main(global_config, **settings):

    # Set the session secret as per out ini file
    session_factory = SignedCookieSessionFactory(
        settings['session.secret'],
    )

    authn_policy = AuthTktAuthenticationPolicy(settings['session.secret'],
        callback=groupfinder)
    authz_policy = ACLAuthorizationPolicy()

    # Tie it all together
    config = Configurator(settings=settings,
              root_factory='yourapp.models.RootFactory',
                          authentication_policy=authn_policy,
                          authorization_policy=authz_policy)
```

## Modify request to return Ziggurat User() Object

We provide a method to modify the pyramid request and return a Ziggurat User() object (if present) in each request. E.g. once a user is logged in, their details are held in the request (in the form of a userid), if we enable the below function, we can easily access all user attributes in our code, to include this feature, enable it by adding the following to your applications __init__.py configurator file:

```
config.include('ziggurat_foundations.ext.pyramid.get_user')
```

Or in your ini configuration file (both methods yeild the same result):

```
pyramid.includes = pyramid_tm
                   ziggurat_foundations.ext.pyramid.get_user
```

Then inside each pyramid view that contains a request, you can access user information with (the code behind this is as described in the offical pyramid cookbook, but we include in within Ziggurat to make your life easier):

```
@view_config(route_name='edit_note', renderer='templates/edit_note.jinja2',
    permission='edit')
def edit_note(request):
    user = request.user
    # user is now a Ziggurat/SQLAlchemy object that you can access
    # Example for user Joe
    print (user.user_name)
    "Joe"
```

**Tip:** Congratulations, your application is now fully configured to use Ziggurat Foundations, take a look at the Usage Examples for a guide (next page) on how to start taking advantage of all the features that Ziggurat has to offer!

CHAPTER 5

# API Documentation

Contents:

## Models

### UserMixin

**class** ziggurat_foundations.models.user.**UserMixin**
Base mixin for User object representation. It supplies all the basic functionality from password hash generation and matching to utility methods used for querying database for users and their permissions or resources they have access to. It is meant to be extended with other application specific properties

### ExternalIdentityMixin

**class** ziggurat_foundations.models.external_identity.**ExternalIdentityMixin**
Mixin for External Identity model - it represents oAuth(or other) accounts attached to your user object

### GroupMixin

**class** ziggurat_foundations.models.group.**GroupMixin**
Mixin for Group model

**validate_permission**(*key*, *permission*)
validates if group can get assigned with permission

### GroupPermissionMixin

**class** ziggurat_foundations.models.group_permission.**GroupPermissionMixin**
Mixin for GroupPermission model

### UserPermissionMixin

**class** ziggurat_foundations.models.user_permission.**UserPermissionMixin**
Mixin for UserPermission model

## UserGroupMixin

**class** ziggurat_foundations.models.user_group.**UserGroupMixin**
Mixin for UserGroup model

## GroupResourcePermissionMixin

**class** ziggurat_foundations.models.group_resource_permission.**GroupResourcePermissionMixin**
Mixin for GroupResourcePermission model

## UserResourcePermissionMixin

**class** ziggurat_foundations.models.user_resource_permission.**UserResourcePermissionMixin**
Mixin for UserResourcePermission model

## ResourceMixin

**class** ziggurat_foundations.models.resource.**ResourceMixin**
Mixin for Resource model

**validate_permission**(*key*, *permission*)
validate if resouce can have specific permission

## get_db_session

ziggurat_foundations.models.base.**get_db_session**(*session=None*, *obj=None*)
utility function that attempts to return sqlalchemy session that could have been created/passed in one of few ways:

- It first tries to read session attached to instance if object argument was passed
- then it tries to return session passed as argument
- finally tries to read pylons-like threadlocal called DBSession
- if this fails exception is thrown

**Parameters**

- **session** –
- **obj** –

**Returns**

## BaseModel

**class** ziggurat_foundations.models.base.**BaseModel**
Basic class that all other classes inherit from that supplies some basic methods useful for interaction with packages like: deform, colander or wtforms

**delete**(*db_session=None*)
Deletes the object via session, this will permanently delete the object from storage on commit

**Parameters** **db_session** –

**Returns**

**get_appstruct**()
return list of tuples keys and values corresponding to this model's data

**get_db_session**(*session=None*)
Attempts to return session via get_db_session utility function `get_db_session()`

**Parameters** **session** –

**Returns**

**get_dict**(*exclude_keys=None*, *include_keys=None*)
return dictionary of keys and values corresponding to this model's data - if include_keys is null the function will return all keys

**Parameters** **exclude_keys** – (optional) is a list of columns from model that

should not be returned by this function :param include_keys: (optional) is a list of columns from model that should be returned by this function :return:

**persist**(*flush=False*, *db_session=None*)
Adds object to session, if the object was freshly created this will persist the object in the storage on commit

**Parameters**

- **flush** – boolean - if true then the session will be flushed instantly
- **db_session** –

**Returns**

**populate_obj**(*appstruct*, *exclude_keys=None*, *include_keys=None*)
updates instance properties *for column names that exist* for this model and are keys present in passed dictionary

**Parameters**

- **appstruct** – (dictionary)
- **exclude_keys** – (optional) is a list of columns from model that

should not be updated by this function :param include_keys: (optional) is a list of columns from model that should be updated by this function :return:

**populate_obj_from_obj**(*instance*, *exclude_keys=None*, *include_keys=None*)
updates instance properties *for column names that exist* for this model and are properties present in passed dictionary

**Parameters**

- **instance** –
- **exclude_keys** – (optional) is a list of columns from model that

should not be updated by this function :param include_keys: (optional) is a list of columns from model that should be updated by this function :return:

# Services

## UserService

**class** ziggurat_foundations.models.services.user.**UserService**

**classmethod by_email**(*email*, *db_session=None*)
fetch user object by email

**Parameters**

- **email** –
- **db_session** –

**Returns**

**classmethod by_email_and_username**(*email*, *user_name*, *db_session=None*)
fetch user object by email and username

**Parameters**

- **email** –
- **user_name** –
- **db_session** –

**Returns**

**classmethod by_id**(*user_id*, *db_session=None*)
fetch user by user id

**Parameters**

- **user_id** –
- **db_session** –

**Returns**

**classmethod by_user_name**(*user_name*, *db_session=None*)
fetch user by user name

**Parameters**

- **user_name** –
- **db_session** –

**Returns**

**classmethod by_user_name_and_security_code**(*user_name*, *security_code*, *db_session=None*)
fetch user objects by user name and security code

**Parameters**

- **user_name** –
- **security_code** –
- **db_session** –

**Returns**

**classmethod** **by_user_names**(*user_names*, *db_session=None*)
fetch user objects by user names

**Parameters**

- **user_names** –
- **db_session** –

**Returns**

**classmethod** **check_password**(*instance*, *raw_password*)
checks string with users password hash using password manager

**Parameters**

- **instance** –
- **raw_password** –

**Returns**

**classmethod** **generate_random_pass**(*chars=7*)
generates random string of fixed length

**Parameters** **chars** –

**Returns**

**static** **generate_random_string**(*chars=7*)

**Parameters** **chars** –

**Returns**

**classmethod** **get**(*user_id*, *db_session=None*)
Fetch row using primary key - will use existing object in session if already present

**Parameters**

- **user_id** –
- **db_session** –

**Returns**

**classmethod** **gravatar_url**(*instance*, *default=u'mm'*, ***kwargs*)
returns user gravatar url

**Parameters**

- **instance** –
- **default** –
- **kwargs** –

**Returns**

**classmethod** **groups_with_resources**(*instance*)
Returns a list of groups users belongs to with eager loaded resources owned by those groups

**Parameters** **instance** –

**Returns**

**classmethod** **permissions**(*instance*, *db_session=None*)

**returns all non-resource permissions based on what groups user** belongs and directly set ones for this user

**Parameters**

- **instance** –
- **db_session** –

**Returns**

**classmethod regenerate_security_code**(*instance*)
generates new security code

**Parameters** **instance** –

**Returns**

**classmethod resources_with_perms**(*instance*, *perms*, *resource_ids=None*, *resource_types=None*, *db_session=None*)

**returns all resources that user has perms for** (note that at least one perm needs to be met)

**Parameters**

- **instance** –
- **perms** –
- **resource_ids** – restricts the search to specific resources
- **resource_types** –
- **db_session** –

**Returns**

**classmethod resources_with_possible_perms**(*instance*, *resource_ids=None*, *resource_types=None*, *db_session=None*)
returns list of permissions and resources for this user

**Parameters**

- **instance** –
- **resource_ids** – restricts the search to specific resources
- **resource_types** – restricts the search to specific resource types
- **db_session** –

**Returns**

**classmethod set_password**(*instance*, *raw_password*)
sets new password on a user using password manager

**Parameters**

- **instance** –
- **raw_password** –

**Returns**

**classmethod user_names_like**(*user_name*, *db_session=None*)
fetch users with similar names using LIKE clause

**Parameters**

- **user_name** –
- **db_session** –

**Returns**

**classmethod users_for_perms**(*perm_names*, *db_session=None*)
return users hat have one of given permissions

**Parameters**

- **perm_names** –
- **db_session** –

**Returns**

## ExternalIdentityService

**class** ziggurat_foundations.models.services.external_identity.**ExternalIdentityService**

**classmethod by_external_id_and_provider**(*external_id*, *provider_name*, *db_session=None*)
Returns ExternalIdentity instance based on search params

**Parameters**

- **external_id** –
- **provider_name** –
- **db_session** –

**Returns** ExternalIdentity

**classmethod get**(*external_id*, *local_user_id*, *provider_name*, *db_session=None*)
Fetch row using primary key - will use existing object in session if already present

**Parameters**

- **external_id** –
- **local_user_id** –
- **provider_name** –
- **db_session** –

**Returns**

**classmethod user_by_external_id_and_provider**(*external_id*, *provider_name*, *db_session=None*)
Returns User instance based on search params

**Parameters**

- **external_id** –
- **provider_name** –
- **db_session** –

**Returns** User

## GroupService

**class** ziggurat_foundations.models.services.group.**GroupService**

**classmethod by_group_name**(*group_name*, *db_session=None*)
fetch group by name

**Parameters**

- **group_name** –
- **db_session** –

**Returns**

**classmethod get**(*group_id*, *db_session=None*)
Fetch row using primary key - will use existing object in session if already present

**Parameters**

- **group_id** –
- **db_session** –

**Returns**

**classmethod get_user_paginator**(*instance*, *page=1*, *item_count=None*, *items_per_page=50*, *user_ids=None*, *GET_params=None*)
returns paginator over users belonging to the group

**Parameters**

- **instance** –
- **page** –
- **item_count** –
- **items_per_page** –
- **user_ids** –
- **GET_params** –

**Returns**

**classmethod resources_with_possible_perms**(*instance*, *perm_names=None*, *resource_ids=None*, *resource_types=None*, *db_session=None*)
**returns list of permissions and resources for this group,** resource_ids restricts the search to specific resources

**Parameters**

- **instance** –
- **perm_names** –
- **resource_ids** –
- **resource_types** –
- **db_session** –

**Returns**

## GroupPermissionService

**class** ziggurat_foundations.models.services.group_permission.**GroupPermissionService**

**classmethod by_group_and_perm**(*group_id*, *perm_name*, *db_session=None*)
return by by_user_and_perm and permission name

**Parameters**

- **group_id** –
- **perm_name** –
- **db_session** –

**Returns**

**classmethod get**(*group_id*, *perm_name*, *db_session=None*)
Fetch row using primary key - will use existing object in session if already present

**Parameters**

- **group_id** –
- **perm_name** –
- **db_session** –

**Returns**

## UserPermissionService

**class** ziggurat_foundations.models.services.user_permission.**UserPermissionService**

**classmethod by_user_and_perm**(*user_id*, *perm_name*, *db_session=None*)
return by user and permission name

**Parameters**

- **user_id** –
- **perm_name** –
- **db_session** –

**Returns**

**classmethod get**(*user_id*, *perm_name*, *db_session=None*)
Fetch row using primary key - will use existing object in session if already present

**Parameters**

- **user_id** –
- **perm_name** –
- **db_session** –

**Returns**

## UserResourcePermissionService

**class** ziggurat_foundations.models.services.user_resource_permission.**UserResourcePermissionSer**

**classmethod by_resource_user_and_perm**(*user_id*, *perm_name*, *resource_id*, *db_session=None*)
return all instances by user name, perm name and resource id

**Parameters**

- **user_id** –
- **perm_name** –
- **resource_id** –
- **db_session** –

**Returns**

**classmethod get**(*user_id*, *resource_id*, *perm_name*, *db_session=None*)
Fetch row using primary key - will use existing object in session if already present

**Parameters**

- **user_id** –
- **resource_id** –
- **perm_name** –
- **db_session** –

**Returns**

## ResourceService

**class** ziggurat_foundations.models.services.resource.**ResourceService**

**classmethod by_resource_id**(*resource_id*, *db_session=None*)
fetch the resouce by id

**Parameters**

- **resource_id** –
- **db_session** –

**Returns**

**classmethod direct_perms_for_user**(*instance*, *user*, *db_session=None*)

**returns permissions that given user has for this resource** without ones inherited from groups that user belongs to

**Parameters**

- **instance** –
- **user** –
- **db_session** –

**Returns**

**classmethod** **get**(*resource_id*, *db_session=None*)
Fetch row using primary key - will use existing object in session if already present

**Parameters**

- **resource_id** –
- **db_session** –

**Returns**

**classmethod** **group_perms_for_user**(*instance*, *user*, *db_session=None*)

**returns permissions that given user has for this resource** that are inherited from groups

**Parameters**

- **instance** –
- **user** –
- **db_session** –

**Returns**

**classmethod** **groups_for_perm**(*instance*, *perm_name*, *group_ids=None*, *limit_group_permissions=False*, *db_session=None*)
return PermissionTuples for groups that have given permission for the resource, perm_name is __any_permission__ then users with any permission will be listed

**Parameters**

- **instance** –
- **perm_name** –
- **group_ids** – limits the permissions to specific group ids
- **limit_group_permissions** – should be used if we do not want to have

user objects returned for group permissions, this might cause performance issues for big groups :param db_session: :return:

**classmethod** **lock_resource_for_update**(*resource_id*, *db_session*)
Selects resource for update - locking access for other transactions

**Parameters**

- **resource_id** –
- **db_session** –

**Returns**

**classmethod** **perm_by_group_and_perm_name**(*resource_id*, *group_id*, *perm_name*, *db_session=None*)
fetch permissions by group and permission name

**Parameters**

- **resource_id** –
- **group_id** –
- **perm_name** –
- **db_session** –

**Returns**

**classmethod** **perms_for_user**(*instance*, *user*, *db_session=None*)

**returns all permissions that given user has for this resource** from groups and directly set ones too

**Parameters**

- **instance** –
- **user** –
- **db_session** –

**Returns**

**classmethod** **users_for_perm**(*instance*, *perm_name*, *user_ids=None*, *group_ids=None*, *limit_group_permissions=False*, *skip_group_perms=False*, *db_session=None*)

return PermissionTuples for users AND groups that have given permission for the resource, perm_name is __any_permission__ then users with any permission will be listed

**Parameters**

- **instance** –
- **perm_name** –
- **user_ids** – limits the permissions to specific user ids
- **group_ids** – limits the permissions to specific group ids
- **limit_group_permissions** – should be used if we do not want to have

user objects returned for group permissions, this might cause performance issues for big groups :param skip_group_perms: do not attach group permissions to the resultset :param db_session: :return:

## ResourceTreeService

**class** ziggurat_foundations.models.services.resource_tree.**ResourceTreeService**(*service_cls*)

**build_subtree_strut**(*result*, **args*, ***kwargs*)

Returns a dictionary in form of {node:Resource, children:{node_id: Resource}}

**Parameters** **result** –

**Returns**

**check_node_parent**(*resource_id*, *new_parent_id*, *db_session=None*, **args*, ***kwargs*)

Checks if parent destination is valid for node

**Parameters**

- **resource_id** –
- **new_parent_id** –
- **db_session** –

**Returns**

**check_node_position**(*parent_id*, *position*, *on_same_branch*, *db_session=None*, **args*, ***kwargs*)

Checks if node position for given parent is valid, raises exception if this is not the case

**Parameters**

- **parent_id** –
- **position** –
- **on_same_branch** – indicates that we are checking same branch
- **db_session** –

**Returns**

**count_children**(*resource_id*, *db_session=None*, **args*, ***kwargs*)
Counts children of resource node

**Parameters**

- **resource_id** –
- **db_session** –

**Returns**

**delete_branch**(*resource_id=None*, *db_session=None*, **args*, ***kwargs*)
This deletes whole branch with children starting from resource_id

**Parameters**

- **resource_id** –
- **db_session** –

**Returns**

**from_parent_deeper**(*parent_id=None*, *limit_depth=1000000*, *db_session=None*, **args*, ***kwargs*)
This returns you subtree of ordered objects relative to the start parent_id (currently only implemented in postgresql)

**Parameters**

- **resource_id** –
- **limit_depth** –
- **db_session** –

**Returns**

**from_resource_deeper**(*resource_id=None*, *limit_depth=1000000*, *db_session=None*, **args*, ***kwargs*)
This returns you subtree of ordered objects relative to the start resource_id (currently only implemented in postgresql)

**Parameters**

- **resource_id** –
- **limit_depth** –
- **db_session** –

**Returns**

**move_to_position**(*resource_id*, *to_position*, *new_parent_id=<ziggurat_foundations.NOOP object>*, *db_session=None*, **args*, ***kwargs*)
Moves node to new location in the tree

**Parameters**

- **resource_id** – resource to move
- **to_position** – new position
- **new_parent_id** – new parent id
- **db_session** –

**Returns**

**path_upper**(*object_id*, *limit_depth=1000000*, *db_session=None*, **args*, ***kwargs*)

**This returns you path to root node starting from object_id** currently only for postgresql

**Parameters**

- **object_id** –
- **limit_depth** –
- **db_session** –

**Returns**

**set_position**(*resource_id*, *to_position*, *db_session=None*, **args*, ***kwargs*)
Sets node position for new node in the tree

**Parameters**

- **resource_id** – resource to move
- **to_position** – new position
- **db_session** –

:return:def count_children(cls, resource_id, db_session=None):

**shift_ordering_down**(*parent_id*, *position*, *db_session=None*, **args*, ***kwargs*)
Shifts ordering to "close gaps" after node deletion or being moved to another branch, begins the shift from given position

**Parameters**

- **parent_id** –
- **position** –
- **db_session** –

**Returns**

**shift_ordering_up**(*parent_id*, *position*, *db_session=None*, **args*, ***kwargs*)
Shifts ordering to "open a gap" for node insertion, begins the shift from given position

**Parameters**

- **parent_id** –
- **position** –
- **db_session** –

**Returns**

## ResourceTreeServicePostgreSQL

**class** ziggurat_foundations.models.services.resource_tree_postgres.**ResourceTreeServicePostgreS**

**classmethod** **build_subtree_strut**(*result*, **args*, ***kwargs*)
Returns a dictionary in form of {node:Resource, children:{node_id: Resource}}

**Parameters** **result** –

**Returns**

**classmethod** **check_node_parent**(*resource_id*, *new_parent_id*, *db_session=None*, **args*, ***kwargs*)
Checks if parent destination is valid for node

**Parameters**

- **resource_id** –
- **new_parent_id** –
- **db_session** –

**Returns**

**classmethod** **check_node_position**(*parent_id*, *position*, *on_same_branch*, *db_session=None*, **args*, ***kwargs*)
Checks if node position for given parent is valid, raises exception if this is not the case

**Parameters**

- **parent_id** –
- **position** –
- **on_same_branch** – indicates that we are checking same branch
- **db_session** –

**Returns**

**classmethod** **count_children**(*resource_id*, *db_session=None*, **args*, ***kwargs*)
Counts children of resource node

**Parameters**

- **resource_id** –
- **db_session** –

**Returns**

**classmethod** **delete_branch**(*resource_id=None*, *db_session=None*, **args*, ***kwargs*)
This deletes whole branch with children starting from resource_id

**Parameters**

- **resource_id** –
- **db_session** –

**Returns**

**classmethod** **from_parent_deeper**(*parent_id=None*, *limit_depth=1000000*, *db_session=None*, **args*, ***kwargs*)
This returns you subtree of ordered objects relative to the start parent_id (currently only implemented in postgresql)

**Parameters**

- **resource_id** –
- **limit_depth** –
- **db_session** –

**Returns**

**classmethod from_resource_deeper**(*resource_id=None*, *limit_depth=1000000*, *db_session=None*, **args*, ***kwargs*)

This returns you subtree of ordered objects relative to the start resource_id (currently only implemented in postgresql)

**Parameters**

- **resource_id** –
- **limit_depth** –
- **db_session** –

**Returns**

**classmethod move_to_position**(*resource_id*, *to_position*, *new_parent_id=<ziggurat_foundations.NOOP object>*, *db_session=None*, **args*, ***kwargs*)

Moves node to new location in the tree

**Parameters**

- **resource_id** – resource to move
- **to_position** – new position
- **new_parent_id** – new parent id
- **db_session** –

**Returns**

**classmethod path_upper**(*object_id*, *limit_depth=1000000*, *db_session=None*, **args*, ***kwargs*)

**This returns you path to root node starting from object_id** currently only for postgresql

**Parameters**

- **object_id** –
- **limit_depth** –
- **db_session** –

**Returns**

**classmethod set_position**(*resource_id*, *to_position*, *db_session=None*, **args*, ***kwargs*)

Sets node position for new node in the tree

**Parameters**

- **resource_id** – resource to move
- **to_position** – new position
- **db_session** –

:return:def count_children(cls, resource_id, db_session=None):

**classmethod** **shift_ordering_down**(*parent_id*, *position*, *db_session=None*, **args*, ***kwargs*)
Shifts ordering to "close gaps" after node deletion or being moved to another branch, begins the shift from given position

**Parameters**

- **parent_id** –
- **position** –
- **db_session** –

**Returns**

**classmethod** **shift_ordering_up**(*parent_id*, *position*, *db_session=None*, **args*, ***kwargs*)
Shifts ordering to "open a gap" for node insertion, begins the shift from given position

**Parameters**

- **parent_id** –
- **position** –
- **db_session** –

**Returns**

CHAPTER 6

# Changelog

## 2016-11-25

- Release 0.7.0
- Introduced ResourceTreeService for nested resource management (currently only PostgreSQL support is implemented)
- added deprecation warnings

**BACKWARDS INCOMPATIBLE CHANGES**

- big code refactoring that *might* possibly introduce minor breaking changes
- _ziggurat_services is now a list

## 2016-07-05

- Release: 0.6.8
- use importlib for imports to avoid issues with unicode passed to __import__()

## 2016-05-05

- Release: 0.6.6/0.6.7
- increased field sizes for external identity tokens, permission names and username
- perm name is checked to be lowercase in all databases, not only for postgresql

## 2016-04-27

- Release: 0.6.5
- got rid of all unicode warnings generated by sqlalchemy and start to use unicode for all strings
- user.resources_with_possible_perms() passes resource_types properly

## 2016-04-19

- Release: 0.6.3/0.6.4
- extended functionality of *populate_obj* function of Base class
- minor bugfix for mysql

**BACKWARDS INCOMPATIBLE CHANGES**

- external_identities.local_user_name column is dropped and replaced with local_user_id

## 2015-11-13

- Release: 0.6.2
- replace cryptacular with passlib

## 2015-09-19

- Release: 0.6.1
- some fixes for ext.pyramid not-working completly with service related changes

## 2015-08-03

- Release: 0.6.0
- models are decoupled from services that interact with models (all model querying methods now live in services)

**BACKWARDS INCOMPATIBLE CHANGES**

import related changes:

- ziggurat_foundations.models doesn't import all the models anymore every model now lives in separate file
- permission related functions now live in permissions module

## 2015-06-15

- Release: 0.5.6
- user model gains security date column
- minor bugfixes

## 2015-06-07

- Release: 0.5.5
- added persist(), delete(), base_query() methods to Base model

## 2015-04-27

- Release: 0.5.3
- resource.groups_for_perm() returns groups/permissions for single resource

## 2015-04-24

- Release: 0.5.2
- resource.users_for_perm() now accepts *limit_group_permissions* param that that makes it return just group with perm name instead tuples including every user/perm pairs

## 2015-04-23

- Release: 0.5.1
- Group.resources_with_possible_perms() added

## 2015-04-17 Release: 0.5

- Now uses detailed permissions
- Methods also return "owned" permissions where applicable
- **BACKWARDS INCOMPATIBLE API CHANGES**
  - ResourceMixin.users_for_perm() accepts additional parameters group_ids, and user_ids to limit the amount of results if needed
  - User.permissions, Resource.perms_for_user, Resource.direct_perms_for_user, Resource.group_perms_for_user, Resource.users_for_perm_detailed, Resource.users_for_perm now return list of detailed PermissionTuple's instead simple [id, perm_name] pairs this will break your application You can use ziggurat_foundations.utils.permission_to_04_acls() to convert the new tuples to pre 0.5 format

## 2015-02-18

- Release: 0.4.3
- Added a way to filter on resource types in UserMixin.resources_with_perms()
- Made User.resources dynamic relationship

## 2014-08-25 Second Alpha Release 0.4

- Move to paginate from webhelpers.paginate
- Users can now log in via username or email address
- Python 3 compatibale after moving away from webhelpers.paginate

## 2012-11-28 version 0.3 First Alpha release

- This release should have a fairly stable API
- Hundreds of small and big changes - based on all great feedback we are now using surrogate pkeys instead of natural pkeys for most models. As result of this few methods started accepting id's instead usernames, so consider yourself warned that this release might be bw. incompatible a bit with your application
- At this point all tests should pass on mysql, postgresql, sqlite

## 2012-05-27 version 0.2 First public release

- added proper alembic(pre 0.3.3 trunk) support for multiple alembic migrations via separate versioning table
- please do manual stamp for CURRENT revision ID: 54d08f9adc8c
- changes for first public pypi release
- Possible backwards incompatibility: Remove cache keyword cruft

## 2012-05-25

- Possible backwards incompatibility: Remove invalidate keyword cruft

## 2012-03-10

- Add registration date to user model, changed last_login_date to no-timezone type (this seem trivial enough to not faciliate schema change)
- previous revision ID: 2d472fe79b95

## 2012-02-19

- Made external identity fields bigger
- previous revision ID: 264049f80948

## 2012-02-13

- Bumped alembic machinery to 0.2
- Enabled developers to set their own custom password managers
- added ordering column for resources in tree
- Stubs for tree traversal
- previous revision ID: 46a9c4fb9560

## 2011-12-20

- Made hash fields bigger
- previous revision ID: 5c84d7260c5

## 2011-11-15

- Added ExternalIdentityMixin - for storing information about user profiles connected to 3rd party identites like facebook/twitter/google/github etc.
- previous revision ID: 24ab8d11f014

## 2011-11-03

- added alembic migration support
- previous revision ID: 2bb1ba973f0b

## 2011-08-14

- resource.users_for_perm(), resource.direct_perms_for_user() and resource.group_perms_for_user() return tuple (user/group_name,perm_name) now

---

**Note:** By default ziggurat aims at **postgresql 8.4+** (CTE support) as main RDBMS system, but currently *everything* except recursive queries(for **optional** resource tree structures) is tested using sqlite, and will run on other popular database systems including mysql. **For other database systems that don't support CTE's fallbacks will be supplied.**

---

CHAPTER 7

# Indices and tables

- genindex
- modindex
- search

## B



## C



## D

## E



## F



## G



## L



## M



## P

## R



## S



## U



## V