Abilian Core Documentation

Release 0.1

2015-08-07

Contents

Welcome to Abilian Core's documentation.

Abilian Core is an enterprise application development platform based on the Flask micro-framework, the SQLAlchemy ORM, good intentions and best practices (for some value of "best").

The full documentation is available on http://docs.abilian.com/.

It builds on powerful and well documented Python librairies, mainly:

- Flask
- SQLAlchemy
- WTForms

This documentation will assume that a developer already has some knowledge of these librairies.

Part I
CONTENTS

CHAPTER 1

About Abilian Core

Abilian Core is an enterprise application development platform based on the Flask micro-framework, the SQLAlchemy ORM, good intentions and best practices (for some value of "best").

The full documentation is available on http://docs.abilian.com/.

1.1 Goals & principles

- Development must be easy and fun (some some definition of "easy" and "fun", of course)
- The less code (and configuration) we write, the better
- Leverage existing reputable open source libraries and frameworks, such as SQLAlchemy and Flask
- It must lower errors, bugs, project's time to deliver. It's intended to be a rapid application development tool
- It must promote best practices in software development, specially Test-Driven Development (as advocated by the GOOS book)

1.2 Features

Here's a short list of features that you may find appealing in Abilian:

1.2.1 Infrastructure

- Plugin framework
- Asynchronous tasks (using Celery)

- Security model and service

1.2.2 Domain model and services

- Domain object model, based on SQLAlchemy
- Audit

1.2.3 Content management and services

- Indexing service
- Document preview and transformation

1.2.4 Social

- Users, groups and social graph (followers)
- Activity streams

1.2.5 User Interface and API

- Forms (based on WTForms)
- CRUD (Create, Retrieve, Edit/Update, Remove) interface from domain models
- Labels and descriptions for each field
- Various web utilities: view decorators, class-based views, Jinja2 filters, etc.
- A default UI based on Bootstrap 3 and several carefully selected jQuery plugins such as Select2
- REST and AJAX API helpers
- i8n: support for multi-language via Babel, with multiple translation dictionaries

1.2.6 Management and admin

- Initial settings wizard
- Admin and user settings framework
- System monitoring (using Sentry)

1.3 Current status

Abilian Core is currently alpha (or even pre-alpha) software, in terms of API stability.

It is currently used in several applications that have been developped by Abilian over the last two years:

- Abilian SBE (Social Business Engine) - an enterprise 2.0 (social collaboration) platform
- Abilian EMS (Event Management System)
- Abilian CRM (Customer / Contact / Community Relationship Management System)
- Abilian Le MOOC - a MOOC prototype
- Abilian CMS - a Web CMS

In other words, Abilian Core is the foundation for a small, but growing, family of business-critical applications that our customers intend us to support in the coming years.

So while Abilian Core APIs, object model and even architecture, may (and most probably will) change due to various refactorings that are expected as we can't be expected to ship perfect software on the firt release, we also intend to treat it as a valuable business asset and keep maintaining and improving it in the foreseeable future.

1.4 Roadmap & getting involved

We have a roadmap on Pivotal Tracker that we use internally to manage our iterative delivery process.

For features and bug requests (or is it the other way around?), we recommend that you use the GitHub issue tracker.

1.5 Licence

Abilian Core is licensed under the LGPL.

1.6 Credits

Abilian Core has been created by the development team at Abilian (currently: Stefane and Bertrand), with financial support from our wonderful customers, and R&D fundings from the French Government, the Paris Region and the European Union.

We are also specially grateful to:

- Armin Ronacher for his work on Flask.
- Michael Bayer for his work on SQLAlchemy.
- Everyone who has been involved with and produced open source software for the Flask ecosystem (Kiran Jonnalagadda and the HasGeek team, Max Countryman, Matt Wright, Matt Good, Thomas Johansson, James Crasta, and probably many others).
- The creators of Django, Pylons, TurboGears, Pyramid and Zope, for even more inspiration.
- The whole Python community.

CHAPTER 2

Installing Abilian Core

If you are a Python web developer (which is the primary target for this project), you probably already know about:

- Python 2.7
- Virtualenv
- Pip

So, after you have created and activated a virtualenv for the project, just run:

```
pip install -r requirements.txt
```

To use some features of the library, namely document and images transformation, you will need to install the additional native packages, using our operating system's package management tools (`dpkg`, `yum`, `brew`...):

- A few image manipulation libraries (`libpng`, `libjpeg`)
- The `poppler-utils`, `unoconv`, `LibreOffice`, `ImageMagick` utilities
- lesscss:

  For Debian/Ubuntu the package is named *node-less*. If your distribution's package is too old, you may install node-js >= 0.10 and npm. Lesscss can then be installed with:

```
$ sudo npm install -g less
npm http GET https://registry.npmjs.org/less
npm http 200 https://registry.npmjs.org/less
...
$ which lessc
/usr/bin/lessc
```

2.1 Testing

Abilian Core come with a full unit and integration testing suite. You can run it with `make test` (once your virtualenv has been activated).

Alternatively, you can use `tox` to run the full test suite in an isolated environment.

Constributing to Abilian Core

3.1 Project on GitHub

The project is hosted on GitHub at: https://github.com/abilian/abilian-core.

Participation in the development of Abilian is welcome and encouraged, through the various mechanisms provided by GitHub:

- Bug reports and feature requests.
- Forks and pull requests.

3.2 License and copyright

The Abilian code is copyrighted by Abilian SAS, a french company.

It is licenced under the LGPL (Lesser General Public License), which means you can reuse the product as a library

If you contribute to Abilian, we ask you to transfer your rights to your contribution to us.

In case you have questions, you're welcome to contact us.

3.3 Build Status

We give a great deal of care to the quality of our software, and try to use all the tools that are at our disposal to make it rock-solid.

This includes:

- Having an exhaustive test suite.
- Using continuous integration (CI) servers to run the test suite on every commit.

- Running tests.
- Using our products daily.

You can check the build status:

- Our own Jenkins server
- On drone.io
- On Travis CI

You can also check the coverage reports:

- On coveralls.io

3.4 Releasing

We're now using *setuptools_scm* to manage version numbers.

It comes with some conventions on its own when it comes to releasing.

Here's what you should do to make a new release on PyPI:

1. Check that the CHANGES.rst file is correct.
2. Commit.
3. Tag (ex: *git tag 0.3.0*), using numbers that are consistent with semantic versionning.
4. Run *python setup.py sdist upload*.

Coding standard

We recommend using the PEP8 and Google coding standard, with the following exceptions:

- Indentation should be 2 chars, not 4.

4.1 Additional rules

TODO

4.2 Notes

4.2.1 Line length

We stick to the "no lines longer than 80 characters" rule despite the fact that we're living in a post VT-220 world.

Here's some rationale by user "badsector" on Reddit:

> I used to use a 120 character limit or ignore E501 on my pep8 checker (python), but eventually went back to the default 80 character limit. I realized it did more for me than let me fit 4 files side by side on a laptop screen:
>
> - It discouraged me from writing long sprawling if statements and method chains.
> - With less space, I thought more assigning about clear and concise names for things.
> - I would break out deeply nested ifs and other control statements into separate functions. This is probably the biggest win since smaller code pieces are easier to unit test due to lowered cyclomatic complexity.

API

5.1 Package abilian

5.1.1 Module abilian.app

Base Flask application class, used by tests or to be extended in real applications.

class Application(*name=None*, *config=None*, **args*, ***kwargs*)
Base application class. Extend it in your own app.

celery_app_cls
celery app class

alias of FlaskCelery

add_access_controller(*name*, *func*, *endpoint=False*)
Add an access controller.

If *name* is None it is added at application level, else if is considered as a blueprint name. If *endpoint* is True then it is considered as an endpoint.

add_static_url(*url_path*, *directory*, *endpoint=None*, *roles=None*)
Adds a new url rule for static files.

Parameters

- **endpoint** – flask endpoint name for this url rule.
- **url** – subpath from application static url path. No heading or trailing slash.
- **directory** – directory to serve content from.

Example:

```
app.add_static_url('myplugin',
                   '/path/to/myplugin/resources',
                   endpoint='myplugin_static')
```

With default setup it will serve content from directory */path/to/myplugin/resources* from url *http://.../static/myplugin*

add_url_rule(*rule*, *endpoint=None*, *view_func=None*, ***options*)
See `Flask.add_url_rule()`.

If *roles* parameter is present, it must be a `abilian.service.security.models.Role` instance, or a list of Role instances.

check_instance_folder(*create=False*)
Verifies instance folder exists, is a directory, and has necessary permissions.

:param:create: if *True*, creates directory hierarchy

Raises OSError with relevant errno

create_db()

create_jinja_environment()

handle_exception(*e*)

handle_http_error(*code*, *error*)
Helper that renders *error{code}.html*.

Convenient way to use it:

```
from functools import partial
handler = partial(app.handle_http_error, code)
app.errorhandler(code)(handler)
```

handle_user_exception(*e*)

init_breadcrumbs()
Inserts the first element in breadcrumbs.

This happens during *request_started* event, which is triggered before any url_value_preprocessor and *before_request* handlers.

init_debug_toolbar()

init_extensions()
Initializes flask extensions, helpers and services.

init_sentry()
Installs Sentry handler if config defines 'SENTRY_DSN'.

install_default_handler(*http_error_code*)
Installs a default error handler for *http_error_code*.

The default error handler renders a template named error404.html for http_error_code 404.

log_exception(*exc_info*)
Log exception only if sentry is not installed (this avoids getting error twice in sentry).

make_config(*instance_relative=False*)

maybe_register_setup_wizard()

register_asset(*type_*, **assets*)
Registers webassets bundle to be served on all pages.

Parameters

- **type** – *"css"*, *"js-top"* or *"js""*.
- ***asset** – a path to file, a webassets.Bundle instance or a callable that returns a webassets.Bundle instance.

Raises KeyError if *type_* is not supported.

register_i18n_js(**paths*)
register templates path translations files, like *select2/select2_locale_{lang}.js*.

Only existing files are registered.

register_jinja_loaders(**loaders*)
Registers one or many *jinja2.Loader* instances for templates lookup.

During application initialization plugins can register a loader so that their templates are available to jinja2 renderer.

Order of registration matters: last registered is first looked up (after standard Flask lookup in app template folder). This allows a plugin to override templates provided by others, or by base application. The application can override any template from any plugins from its template folder (See *Flask.Application.template_folder*).

Raise *ValueError* if a template has already been rendered

register_plugins()
Loads plugins listed in config variable 'PLUGINS'.

setup_logging()

APP_PLUGINS = ('abilian.web.search', 'abilian.web.tags', 'abilian.web.comments', 'abilian.we
Custom apps may want to always load some plugins: list them here.

CONFIG_ENVVAR = 'ABILIAN_CONFIG'
Environment variable used to locate a config file to load last (after instance config file). Use this if you want to override some settings on a configured instance.

configured
True if application has a config file and can be considered configured for site.

db

default_config = ImmutableDict({'JSON_AS_ASCII': True, 'SENTRY_JS_PLUGINS': ('con

default_view = None
instance of `web.views.registry.Registry`.

jinja_loader
Searches templates in custom app templates dir (default flask behaviour), fallback on abilian templates.

jinja_options

js_api = None
json serializable dict to land in Javascript under Abilian.api

redis

script_manager = '.commands.manager'
`flask.ext.script.Manager` instance for shell commands of this app. defaults to *.commands.manager*, relative to app name.

class ServiceManager
Mixin that provides lifecycle (register/start/stop) support for services.

start_services()

stop_services()

create_app(*config=None*)

5.1.2 Module `abilian.i18n`

I18n.

To mark strings for transalation:

```
from abilian.i18n import _
_(u'message to translate')
```

Use _ for gettext, _l for lazy_gettext, _n for ngettext.

`Babel extension` support multiple translation paths. This allows to add more catalogs to search for translations, in LIFO order. This feature can be used to override some translations in a custom application, be providing a catalog with messages to override:

```
current_app.extensions['babel'].add_translations('abilian.core')
```

See `add_translations`.

To extract messages to build the message catalog template (.pot), use the following "*-k*" parameters:

```
$ pybabel extract -F babel.cfg -k "_n:1,2" -k "_l" -o "msg.pot" "src"
```

This can be made easier by placing in *setup.cfg*:

```
[extract_messages]
mapping_file = babel.cfg
keywords = _n:1,2 _l
output-file = msg.pot
input-dirs = src
```

And just type:

```
$ python setup.py extract_messages
```

_ = <function gettext>
Translates a string with the current locale and passes in the given keyword arguments as mapping to a string formatting string.

```
gettext(u'Hello World!')
gettext(u'Hello %(name)s!', name='World')
```

_l = <function lazy_gettext>
Like `gettext()` but the string returned is lazy which means it will be translated when it is used as an actual string.

Example:

```
hello = lazy_gettext(u'Hello World')

@app.route('/')
def index():
    return unicode(hello)
```

_n = <function ngettext>
Translates a string with the current locale and passes in the given keyword arguments as mapping to a string formatting string. The *num* parameter is used to dispatch between singular and various plural forms of the message. It is available in the format string as `%(num)d` or `%(num)s`. The source language should be English or a similar language which only has one plural form.

```
ngettext(u'%(num)d Apple', u'%(num)d Apples', num=len(apples))
```

class Babel(**args*, ***kwargs*)
Bases: `flask_babel.Babel`

Allow to load translations from other modules

add_translations(*module_name*, *translations_dir='translations'*, *domain='messages'*)
Adds translations from external module. For example:

```
babel.add_translations('abilian.core')
```

Will add translations files from *abilian.core* module.

babel = <abilian.i18n.Babel object>
importable instance of `Babel`

gettext(*string*, ***variables*)
Translates a string with the current locale and passes in the given keyword arguments as mapping to a string formatting string.

```
gettext(u'Hello World!')
gettext(u'Hello %(name)s!', name='World')
```

lazy_gettext(*string*, ***variables*)
Like `gettext()` but the string returned is lazy which means it will be translated when it is used as an actual string.

Example:

```
hello = lazy_gettext(u'Hello World')

@app.route('/')
def index():
    return unicode(hello)
```

localeselector()
Default locale selector used in abilian applications

ngettext(*singular*, *plural*, *num*, ***variables*)
Translates a string with the current locale and passes in the given keyword arguments as mapping to a string formatting string. The *num* parameter is used to dispatch between singular and various plural forms of the message. It is available in the format string as `%(num)d` or `%(num)s`. The source language should be English or a similar language which only has one plural form.

```
ngettext(u'%(num)d Apple', u'%(num)d Apples', num=len(apples))
```

set_locale(**args*, ***kwds*)
Change current locale.

Can be used as a context manager to temporary change locale:

```
with set_locale('fr') as fr_locale:
    ...
```

Parameters **locale** (`babel.core.Locale` or *str*) – locale to use. If it's a string if must be a valid locale specification

Return type `babel.core.Locale`

Returns locale set

timezoneselector()
Default timezone selector used in abilian applications

render_template_i18n(*template_name_or_list*, ***context*)
Try to build an ordered list of template to satisfy the current locale

babel = <abilian.i18n.Babel object>
importable instance of `Babel`

VALID_LANGUAGES_CODE = frozenset(['gv', 'gu', 'gd', 'ga', 'gl', 'lg', 'tn', 'ln', 'lo', 'tr', 'ts', 'lv', 'to', 'lt',
accepted languages codes

5.2 Package abilian.plugin

Starting work on a plugin system. This will probably be refactored heavily in the future.

5.3 Package abilian.core

5.3.1 Module abilian.core.commands

Abilian script commands to be used in a project. See Flask-Script documentation for full documentation.

Here is how a *manage.py* can include these commands:

```
from flask.ext.script import Manager
from abilian.commands import setup_abilian_commands

my_manager = Manager(app)
setup_abilian_commands(my_manager)
```

You can also include abilian commands as sub commands:

```
from abilian.commands import manager as abilian_manager
my_manager.add_command('abilian', abilian_manager)
```

Extensions can add their own commands to manager:

```
from flask.ext.script import Manager
from abilian.commands import manager

@manager.command
def hello():
    print u"hello"

# or install subcommands
sub_manager = Manager(usage='Little extension')
abilian_manager.add_command('special_commands', sub_manager)
```

setup_abilian_commands(*manager*)

Register abilian commands on manager.

Parameters **manager** – flask.ext.script.Manager instance to add commands onto

Usage exemple:

```
from flask.ext.script import Manager
from abilian.commands import setup_abilian_commands
```

```
my_manager = Manager(app)
setup_abilian_commands(my_manager)
```

5.3.2 Module abilian.core.entities

Base class for entities, objects that are managed by the Abilian framwework (unlike SQLAlchemy models which are considered lower-level).

class Entity(**args*, ***kwargs*)

Bases: abilian.core.models.base.Indexable, abilian.core.models.BaseMixin, flask_sqlalchemy.Model

Base class for Abilian entities.

From Sqlalchemy POV Entities use Joined-Table inheritance, thus entities subclasses cannot use inheritance themselves (as of 2013 Sqlalchemy does not support multi-level inheritance)

The metaclass automatically sets up polymorphic inheritance parameters by inserting a mixin class in parent classes. If you need to pass additional parameters to *__mapper_args__*, do it as follow:

```
class MyContent(Entity):

    @sqlalchemy.ext.declarative.declared_attr
    def __mapper_args__(cls):
        # super(Mycontent, cls).__mapper_args__ would be prettier, but
        # `MyContent` is not defined at this stage.
        args = Entity.__dict__['__mapper_args__'].fget(cls)
        args['order_by'] = cls.created_at # for example
        return args
```

__metaclass__

alias of EntityMeta

query_class

alias of EntityQuery

__init__(**args*, ***kwargs*)

display_value(*field_name*, *value=<object object>*)

Return display value for fields having 'choices' mapping (stored value -> human readable value). For other fields it will simply return field value.

display_value should be used instead of directly getting field value.

If *value* is provided it is "tranlated" to a human-readable value. This is useful for obtaining a human readable label from a raw value

SLUG_SEPARATOR = u'-'

__auditable__ = frozenset([])

__default_permissions__ = frozenset([])

Permission to roles mapping to set at object creation time.

Default permissions can be declared as a `dict` on classes, the final datastructure will changed by metaclass to a `frozenset` of `dict.items()`. This is made to garantee the immutability of definition on parent classes.

Exemple definition:

```
__default_permissions__ = {
    READ: {Owner, Authenticated},
    WRITE: {Owner},
}
```

To alter inherited default permissions:

```
class Child(Parent):
    __default_permissions__ = dp = dict(ParentClass.__default_permissions__)
    dp[READ] = dp[READ] - {Authenticated} + {Anonymous}
    del dp
```

__editable__ = frozenset([])

__indexable__ = False

__indexation_args__ = {'index_to': (('object_key', (('object_key', ID(format=Existence(boos

__mapper__ = <Mapper at 0x7f9e9eba30d0; Entity>

__mapper_args__ = {'polymorphic_on': '_entity_type'}

__module__ = 'abilian.core.entities'

__searchable__ = frozenset([])

__table__ = Table('entity', MetaData(bind=None), Column('id', Integer(), table=<entity>, p

auto_slug

This property is used to auto-generate a slug from the name attribute. It can be customized by subclasses.

created_at

creator

creator_id

deleted_at

entity_class

entity_type = None

id

meta

A dictionnary of simple values (JSON-serializable) to conveniently annotate the entity.

It is recommanded to keep it lighwight and not store large objects in it.

name
The name is a string that is shown to the user; it could be a title for document, a folder name, etc.

object_type

owner

owner_id

slug
The slug attribute may be used in URLs to reference the entity, but uniqueness is not enforced, even within same entity type. For example if an entity class represent folders, one could want uniqueness only within same parent folder.

If slug is empty at first creation, its is derived from the name. When name changes the slug is not updated. If name is also empty, the slug will be the friendly entity_type with concatenated with entity's id.

updated_at

exception ValidationError

class Entity(**args*, ***kwargs*)
Base class for Abilian entities.

From Sqlalchemy POV Entities use Joined-Table inheritance, thus entities subclasses cannot use inheritance themselves (as of 2013 Sqlalchemy does not support multi-level inheritance)

The metaclass automatically sets up polymorphic inheritance parameters by inserting a mixin class in parent classes. If you need to pass additional parameters to *__mapper_args__*, do it as follow:

```
class MyContent(Entity):

    @sqlalchemy.ext.declarative.declared_attr
    def __mapper_args__(cls):
        # super(Mycontent, cls).__mapper_args__ would be prettier, but
        # `MyContent` is not defined at this stage.
        args = Entity.__dict__['__mapper_args__'].fget(cls)
        args['order_by'] = cls.created_at # for example
        return args
```

query_class
alias of `EntityQuery`

display_value(*field_name*, *value=<object object>*)
Return display value for fields having 'choices' mapping (stored value -> human readable value). For other fields it will simply return field value.

display_value should be used instead of directly getting field value.

If *value* is provided it is "tranlated" to a human-readable value. This is useful for obtaining a human readable label from a raw value

SLUG_SEPARATOR = u'-'

auto_slug
This property is used to auto-generate a slug from the name attribute. It can be customized by subclasses.

created_at

creator

creator_id

deleted_at

entity_class

entity_type = None

id

meta
A dictionnary of simple values (JSON-serializable) to conveniently annotate the entity.

It is recommanded to keep it lighwight and not store large objects in it.

name
The name is a string that is shown to the user; it could be a title for document, a folder name, etc.

object_type

owner

owner_id

slug
The slug attribute may be used in URLs to reference the entity, but uniqueness is not enforced, even within same entity type. For example if an entity class represent folders, one could want uniqueness only within same parent folder.

If slug is empty at first creation, its is derived from the name. When name changes the slug is not updated. If name is also empty, the slug will be the friendly entity_type with concatenated with entity's id.

updated_at

all_entity_classes
Returns the list of all concrete persistent classes that are subclasses of Entity.

5.3.3 Module `abilian.core.extensions`

Create all standard extensions.

get_extension(*name*)
Get the named extension from the current app, returning None if not found.

5.3.4 Module abilian.core.logging

Special loggers

Changing *patch_logger* logging level must be done very early, because it may emit logging during imports. Ideally, it's should be the very first action in your entry point before anything has been imported:

```
import logging
logging.getLogger('PATCH').setLevel(logging.INFO)
```

patch_logger = <abilian.core.logging.PatchLoggerAdapter object>
logger for monkey patchs. use like this: patch_logger.info(<func>'patched_func')

5.3.5 Module abilian.core.signals

All signals used by Abilian Core.

Signals are the main tools used for decoupling applications components by sending notifications. In short, signals allow certain senders to notify subscribers that something happened.

Cf. http://flask.pocoo.org/docs/signals/ for detailed documentation.

The main signal is currently `activity`.

activity = <blinker.base.NamedSignal object at 0x7f9e9f66bb90; 'activity'>
This signal is used by the activity streams service and its clients.

components_registered = <blinker.base.NamedSignal object at 0x7f9e9f66bb10; 'app:components
Triggered at application initialization when all extensions and plugins have been loaded

register_js_api = <blinker.base.NamedSignal object at 0x7f9e9f66bb50; 'app:register-js-api'>
Trigger when JS api must be registered. At this time `flask.url_for()` is usable

user_loaded = <blinker.base.NamedSignal object at 0x7f9e9f66bbd0; 'user_loaded'>
This signal is sent when user object has been loaded. g.user and current_user are available.

5.3.6 Module abilian.core.sqlalchemy

Additional data types for sqlalchemy

class `AbilianBaseSAExtension`(*app=None*, *use_native_unicode=True*, *session_options=None*)
Base subclass of `flask_sqlalchemy.SQLAlchemy`. Add our custom driver hacks.

`apply_driver_hacks`(*app*, *info*, *options*)

class `JSON`(**args*, ***kwargs*)
Stores any structure serializable with json.

Usage JSON() Takes same parameters as sqlalchemy.types.Text

`impl`
alias of `Text`

`process_bind_param`(*value*, *dialect*)

`process_result_value`(*value*, *dialect*)

class `JSONUniqueListType`(**args*, ***kwargs*)
Store a list in JSON format, with items made unique and sorted.

`process_bind_param`(*value*, *dialect*)

`python_type`

class `Locale`(**args*, ***kwargs*)
Store a `babel.Locale` instance

`impl`
alias of `UnicodeText`

`process_bind_param`(*value*, *dialect*)

`process_result_value`(*value*, *dialect*)

`python_type`

class `MutationDict`
Provides a dictionary type with mutability support.

`clear`()

classmethod `coerce`(*key*, *value*)
Convert plain dictionaries to MutationDict.

`pop`(*key*, **args*)

`popitem`()

`setdefault`(*key*, *failobj=None*)

`update`(*other*)

class `MutationList`
Provides a list type with mutability support.

`append`(*item*)

classmethod `coerce`(*key*, *value*)
Convert list to MutationList.

extend(*other*)

insert(*idx*, *value*)

pop(*i=-1*)

remove(*item*)

reverse()

sort(**args*, ***kwargs*)

SQLAlchemy
alias of `AbilianBaseSAExtension`

class Timezone(**args*, ***kwargs*)
Store a `pytz.tzfile.DstTzInfo` instance

impl
alias of `UnicodeText`

process_bind_param(*value*, *dialect*)

process_result_value(*value*, *dialect*)

python_type

class UUID(**args*, ***kwargs*)
Platform-independent UUID type.

Uses Postgresql's UUID type, otherwise uses CHAR(32), storing as stringified hex values.

From SQLAlchemy documentation.

impl
alias of `CHAR`

load_dialect_impl(*dialect*)

process_bind_param(*value*, *dialect*)

process_result_value(*value*, *dialect*)

JSONDict(**args*, ***kwargs*)
Stores a dict as JSON on database, with mutability support.

JSONList(**args*, ***kwargs*)
Stores a list as JSON on database, with mutability support.

If kwargs has a param *unique_sorted* (which evaluated to True), list values are made unique and sorted.

filter_cols(*model*, **filtered_columns*)
Return columnsnames for a model except named ones. Useful for defer() for example to retain only columns of interest

ping_connection(*dbapi_connection*, *connection_record*, *connection_proxy*)
Ensure connections are valid.

From: *http://docs.sqlalchemy.org/en/rel_0_8/core/pooling.html*

In case db has been restarted pool may return invalid connections.

5.3.7 Module `abilian.core.models`

class `IdMixin`

`id` = Column(None, Integer(), table=None, primary_key=True, nullable=False)

class `Indexable`

Mixin with sensible defaults for indexable objects.

`object_key`

`object_type`

class `Info`(***kw*)

`copy`()

class `Model`(***kwargs*)

Base Model class.

`metadata` = MetaData(bind=None)

`query` = None

class `TimestampedMixin`

`created_at` = Column(None, DateTime(), table=None, default=ColumnDefault(<function <

creation date

`deleted_at` = Column(None, DateTime(), table=None)

`updated_at` = Column(None, DateTime(), table=None, onupdate=ColumnDefault(<functio

last modification date

`SYSTEM` = {'auditable': False, 'editable': False}

SYSTEM properties are properties defined by the system and not supposed to be changed manually.

Subject classes (i.e. people, groups, etc.).

See ICOM-ics-v1.0 "Subject Branch".

TODO: I'm not a big fan of the "subject" name. Could be replaced by something else, like "people" or "principal" ?

class `User`(*password=None*, ***kwargs*)

`query_class`

alias of `UserQuery`

authenticate(*password*)

display_value(*field_name*, *value=<object object>*)
Return display value for fields having 'choices' mapping (stored value -> human readable value). For other fields it will simply return field value.

display_value should be used instead of directly getting field value.

If *value* is provided it is "tranlated" to a human-readable value. This is useful for obtaining a human readable label from a raw value

follow(*followee*)

is_admin_of(*group*)

is_following(*other*)

is_member_of(*group*)

join(*group*)

leave(*group*)

set_password(*password*)
Encrypts and sets password.

unfollow(*followee*)

can_login

created_at

deleted_at

email

entity_type = u'abilian.core.models.subjects.User'

first_name

followers

id

is_online

last_active

last_name

locale

name

password

photo

timezone

updated_at

class `Group`(***kwargs*)

`display_value`(*field_name*, *value=<object object>*)
Return display value for fields having 'choices' mapping (stored value -> human readable value). For other fields it will simply return field value.

display_value should be used instead of directly getting field value.

If *value* is provided it is "tranlated" to a human-readable value. This is useful for obtaining a human readable label from a raw value

`admins`

`created_at`

`deleted_at`

`description`

`entity_type = u'abilian.core.models.subjects.Group'`

`id`

`members`

`name`

`photo`

`public`

`updated_at`

class `Principal`
A principal is either a User or a Group.

`has_role`(*role*)

class `OwnedMixin`(**args*, ***kwargs*)

`creator = <RelationshipProperty at 0x7f9e9cb33290; no key>`

`creator_id = Column(None, NullType(), ForeignKey('user.id'), table=None)`

`creator_name`

`owner = <RelationshipProperty at 0x7f9e9cb33250; no key>`

`owner_id = Column(None, NullType(), ForeignKey('user.id'), table=None)`

`owner_name`

Blob. References to files stored in a on-disk repository

class `Blob`(*value=None*, **args*, ***kwargs*)
Model for storing large file content.

Files are stored on-disk, named after their uuid. Repository is located in instance folder/data/files.

query_class
alias of `BlobQuery`

display_value(*field_name*, *value=<object object>*)
Return display value for fields having 'choices' mapping (stored value -> human readable value). For other fields it will simply return field value.

display_value should be used instead of directly getting field value.

If *value* is provided it is "tranlated" to a human-readable value. This is useful for obtaining a human readable label from a raw value

file
Return `pathlib.Path` object used for storing value

id

md5
Return md5 from meta, or compute it if absent

meta

size
Return size in bytes of value

uuid

value
Binary value content

class BlobQuery(*entities*, *session=None*)
Query class for `Blob` objects

by_uuid(*uuid*)
Like *.get()* but by uuid

Parameters **uuid** – a *string* or an *uuid*.

class SupportTagging

class Tag(***kwargs*)
Tags are text labels that can be attached to `entities`.

They are namespaced, so that independent group of tags can be defined in the application. The default namespace is *"default"*.

display_value(*field_name*, *value=<object object>*)
Return display value for fields having 'choices' mapping (stored value -> human readable value). For other fields it will simply return field value.

display_value should be used instead of directly getting field value.

If *value* is provided it is "tranlated" to a human-readable value. This is useful for obtaining a human readable label from a raw value

entities
`entities` attached to this tag

id

label
Label visible to the user

ns
namespace

is_support_tagging(*obj*)

Parameters **obj** – a class or instance

register(*cls*)
Register an `Entity` as a taggable class.

Can be used as a class decorator:

```
@tag.register
class MyContent(Entity):
    ....
```

TAGS_ATTR = '__tags__'
backref attribute on tagged elements

class Comment(**args*, ***kwargs*)
A Comment related to an `Entity`.

SLUG_SEPARATOR = u'-'

body
comment's main content

created_at

creator

creator_id

deleted_at

entity
Commented entity

entity_id

entity_type = 'abilian.core.models.comment.Comment'

id

meta

name

owner

owner_id

slug

updated_at

class Commentable

for_entity(*obj*, *check_commentable=False*)
Return comments on an entity.

is_commentable(*obj*)

Parameters **obj** – a class or instance

register(*cls*)
Register an `Entity` as a commentable class.

Can be used as a class decorator:

```
@comment.register
class MyContent(Entity):
    ...
```

ATTRIBUTE = '__comments__'
name of backref on target `Entity` object

class Attachment(**args*, ***kwargs*)
An Attachment owned by an `Entity`.

SLUG_SEPARATOR = u'-'

blob
file. Stored in a `Blob`

blob_id

created_at

creator

creator_id

deleted_at

description

entity
owning entity

entity_id

entity_type = 'abilian.core.models.attachment.Attachment'

id

meta

name

owner

owner_id

slug

updated_at

class SupportAttachment

for_entity(*obj*, *check_support_attachments=False*)
Return attachments on an entity.

is_support_attachments(*obj*)

Parameters **obj** – a class or instance

register(*cls*)
Register an `Entity` as a attachmentable class.

Can be used as a class decorator:

```
@attachment.register
class MyContent(Entity):
    ....
```

set_attachment_name(*mapper*, *connection*, *target*)

ATTRIBUTE = '__attachments__'
name of backref on target `Entity` object

class BaseMixin

to_dict()

to_json()

column_names

5.3.8 Module `abilian.core.util`

Various tools that don't belong some place specific.

class BasePresenter(*model*)
A presenter wraps a model an adds specific (often, web-centric) accessors. subclass to make it useful. Presenters are immutable.

classmethod wrap_collection(*models*)

class Pagination(*page*, *per_page*, *total_count*)

iter_pages(*left_edge=2*, *left_current=2*, *right_current=5*, *right_edge=2*)

has_next

has_prev

next

pages

prev

class **memoized**(*func*)
Decorator. Caches a function's return value each time it is called. If called later with the same arguments, the cached value is returned (not reevaluated).

class **timer**(*f*)
Decorator that mesures the time it takes to run a function.

fqcn(*cls*)
Fully Qualified Class Name

friendly_fqcn(*cls_name*)
Friendly name of fully qualified class name. :param cls_name: a string or a class

get_params(*names*)
Returns a dictionary with params from request.

TODO: I think we don't use it anymore and it should be removed before someone gets hurt.

local_dt(*dt*)
Return an aware datetime in system timezone, from a naive or aware datetime. Naive datetime are assumed to be in UTC TZ.

noproxy(*obj*)
Unwrap obj from werkzeug.local.LocalProxy if needed. This is required if one want to test *isinstance(obj, SomeClass)*.

pdb_on_error(*fn*)
Decorator to trigger (i)pdb on exception inside decorated function. Active only in DEBUG mode. Useful to debug POST only views for example.

slugify(*value*, *separator=u'-'*)
Slugify an unicode string, to make it URL friendly.

utc_dt(*dt*)
set UTC timezone on a datetime object. A naive datetime is assumed to be in UTC TZ

5.4 Package abilian.services

5.4.1 Module abilian.services.base

exception **ServiceNotRegistered**

class **Service**(*app=None*)
Base class for services.

AppStateClass
State class to use for this Service

alias of `ServiceState`

static if_running(*meth*)
Decorator for service methods that must be ran only if service is in running state.

init_app(*app*)

start(*ignore_state=False*)
Starts the service.

stop(*ignore_state=False*)
Stops the service.

app_state
Current service state in current application.

:raise:RuntimeError if working outside application context.

name = None
service name in Application.extensions / Application.services

running

Returns *False* if working outside application context, if service is not registered on current application, or if service is halted for current application.

class ServiceState(*service*, *running=False*)
Service state stored in Application.extensions

running = False

service = None
reference to `Service` instance

5.4.2 Module `abilian.services.activity`

class ActivityEntry(***kwargs*)
Main table for all activities.

display_value(*field_name*, *value=<object object>*)
Return display value for fields having 'choices' mapping (stored value -> human readable value). For other fields it will simply return field value.

display_value should be used instead of directly getting field value.

If *value* is provided it is "tranlated" to a human-readable value. This is useful for obtaining a human readable label from a raw value

actor

actor_id

happened_at

id

object

object_id

object_type

target

target_id

target_type

verb

class **ActivityService**(*app=None*)

static **entries_for_actor**(*actor*, *limit=50*)

init_app(*app*)

log_activity(*sender*, *actor*, *verb*, *object*, *target=None*)

start()

stop()

name = 'activity'

5.4.3 Module abilian.services.audit

5.4.4 Module abilian.services.conversion

Conversion service.

Hardcoded to manage only conversion to PDF, to text and to image series.

Includes result caching (on filesystem).

Assumes poppler-utils and LibreOffice are installed.

TODO: rename Converter into ConversionService ?

exception **ConversionError**

class **AbiwordPDFHandler**(**args*, ***kwargs*)

convert(*blob*, ***kw*)

accepts_mime_types = ['application/msword', 'application/vnd.oasis.opendocument.text', 'te

produces_mime_types = ['application/pdf']

class **AbiwordTextHandler**(**args*, ***kwargs*)

convert(*blob*, ***kw*)

accepts_mime_types = ['application/msword']

produces_mime_types = ['text/plain']

class Cache

clear()

get(*key*)

set(*key*, *value*)

CACHE_DIR = None

class CloudoooPdfHandler(**args*, ***kwargs*)
Handles conversion from OOo to PDF.

Highly inefficient since file are serialized in base64 over HTTP.

Deactivated because it's so hard to set up.

FIXME: needs cleanup, or removal.

convert(*key*)

SERVER_URL = 'http://localhost:8011'

accepts_mime_types = ['application/.*']

pivot_format_map = {'pptx': 'odp', 'xlsx': 'ods', 'docx': 'odt', 'doc': 'odt', 'ppt': 'odp', 'xls': 'od

produces_mime_types = ['application/pdf']

class Converter

clear()

static digest(*blob*)

get_image(*digest*, *blob*, *mime_type*, *index*, *size=500*)
Return an image for the given content, only if it already exists in the image cache

get_metadata(*digest*, *content*, *mime_type*)
Gets a dictionary representing the metadata embedded in the given content.

has_image(*digest*, *mime_type*, *index*, *size=500*)
Tell if there is a preview image

init_app(*app*)

init_work_dirs(*cache_dir*, *tmp_dir*)

register_handler(*handler*)

to_image(*digest*, *blob*, *mime_type*, *index*, *size=500*)
Converts a file to a list of images. Returns image at the given index.

to_pdf(*digest*, *blob*, *mime_type*)

to_text(*digest*, *blob*, *mime_type*)
Converts a file to plain text.

Useful for full-text indexing. Returns an unicode string.

class Handler(**args*, ***kwargs*)

accept(*source_mime_type*, *target_mime_type*)
Generic matcher based on patterns.

convert(*key*, ***kw*)

init_app(*app*)

TMP_DIR

accepts_mime_types = []

encoding_sniffer

mime_sniffer

produces_mime_types = []

class ImageMagickHandler(**args*, ***kwargs*)

convert(*blob*, ***kw*)

accepts_mime_types = ['image/.*']

produces_mime_types = ['application/pdf']

class PdfToPpmHandler(**args*, ***kwargs*)

convert(*blob*, *size=500*)
Size is the maximum horizontal size.

accepts_mime_types = ['application/pdf']

produces_mime_types = ['image/jpeg']

class PdfToTextHandler(**args*, ***kwargs*)

convert(*blob*, ***kw*)

accepts_mime_types = ['application/pdf']

produces_mime_types = ['text/plain']

class UnoconvPdfHandler(**args*, ***kwargs*)
Handles conversion from office documents (MS-Office, OOo) to PDF.

Uses unoconv.

convert(*blob*, ***kw*)
Unoconv converter called.

init_app(*app*)

accepts_mime_types = ['application/vnd.oasis.*', 'application/msword', 'application/mspow

produces_mime_types = ['application/pdf']

run_timeout = 60

unoconv = 'unoconv'

unoconv_version

class WvwareTextHandler(**args*, ***kwargs*)

convert(*blob*, ***kw*)

accepts_mime_types = ['application/msword']

produces_mime_types = ['text/plain']

get_tmp_dir()

make_temp_file(**args*, ***kwds*)

5.4.5 Module abilian.services.image

Provides tools (currently: only functions, not a real service) for image processing.

resize(*orig*, *width*, *height*, *mode='fit'*)

SCALE = 'scale'
resize without retaining original proportions

FIT = 'fit'
resize image and retain original proportions. Image width and height will be at most specified width and height, respectively; At least width or height will be equal to specified width and height, respectively.

CROP = 'crop'
crop image and resize so that it matches specified width and height.

5.4.6 Module abilian.services.indexing

service
Index documents using whoosh

class WhooshIndexService(**args*, ***kwargs*)
Index documents using whoosh

AppStateClass
alias of IndexServiceState

after_commit(*session*)
Any db updates go through here. We check if any of these models have

`__searchable__` fields, indicating they need to be indexed. With these we update the whoosh index for the model. If no index exists, it will be created here; this could impose a penalty on the initial commit of a model.

after_flush(*session*, *flush_context*)

clear()
Remove all content from indexes, and unregister all classes.

After clear() the service is stopped. It must be started again to create new indexes and register classes.

clear_update_queue(*app=None*)

get_document(*obj*, *adapter=None*)

index(*name='default'*)

index_objects(*objects*, *index='default'*)
Bulk index a list of objects.

init_app(*app*)

init_indexes()
Create indexes for schemas.

register_class(*cls*, *app_state=None*)
Registers a model class

register_classes()

register_search_filter(*func*)
Register a function that returns a query used for filtering search results. This query is And'ed with other filters.

If no filtering should be performed the function must return None.

register_value_provider(*func*)
Register a function that may alter content of indexable document.

It is used in `get_document()` and called after adapter has built document.

The function must accept (document, obj) as arguments, and return the new document object.

search(*q*, *index='default'*, *fields=None*, *Models=()*, *object_types=()*, *prefix=True*, *facet_by_type=None*, ***search_args*)
Interface to search indexes.

Parameters

- **q** – unparsed search string.
- **index** – name of index to use for search.
- **fields** – optionnal mapping of field names -> boost factor?
- **Models** – list of Model classes to limit search on.
- **object_types** – same as *Models*, but directly the model string.

- **prefix** – enable or disable search by prefix
- **facet_by_type** – if set, returns a dict of object_type: results with a max of *limit* matches for each type.
- **search_args** – any valid parameter for `whoosh.searching.Search.search()`. This includes *limit*, *groupedby* and *sortedby*

search_for_class(*query*, *cls*, *index='default'*, ***search_args*)

searchable_object_types()
List of (object_types, friendly name) present in the index.

start()

default_search_fields
Return default field names and boosts to be used for searching. Can be configured with *SEARCH_DEFAULT_BOOSTS*

name = 'indexing'

indexable_role(*principal*)
Returns a string suitable for query against *allowed_roles_and_users* field.

Parameters **principal** – It can be `Anonymous`, `Authenticated`, or an instance of `User` or `Group`.

5.4.7 Module abilian.services.security

service
Security service, manages roles and permissions.

class SecurityServiceState(*service*, *running=False*)

needs_db_flush = False
True if security has changed

use_cache = True

Anonymous
marker for role assigned to 'Anonymous'

Authenticated
marker for role assigned to 'Authenticated'

5.5 Package abilian.web

5.5.1 Module abilian.web.attachments

register_plugin(*app*)

5.5.2 Module abilian.web.comments

register_plugin(*app*)

5.5.3 Module abilian.web.decorators

Useful decorators for web views.

templated(*template=None*)
The idea of this decorator is that you return a dictionary with the values passed to the template from the view function and the template is automatically rendered.

@deprecated

5.5.4 Module abilian.web.filters

Add a few specific filters to Jinja2.

abbrev(*s*, *max_size*)

age(*dt*, *now=None*)

autoescape(*filter_func*)
Decorator to autoescape result from filters.

babel2datepicker(*pattern*)
Converts date format from babel (http://babel.pocoo.org/docs/dates/#date-fields)) to a format understood by bootstrap-datepicker.

date(*value*, *format='EE, d MMMM y'*)

date_age(*dt*, *now=None*)

filesize(*d*)

init_filters(*env*)

labelize(*s*)

linkify(*eval_ctx*, **args*, ***kwargs*)

nl2br(*eval_ctx*, **args*, ***kwargs*)
Replace newlines with
.

obj_to_url(*obj*)
Find url for obj using `url_for()`, return empty string is not found.

`url_for()` is also provided in jinja context, the filtering version is forgiving when *obj* has no default view set.

paragraphs(*eval_ctx*, **args*, ***kwargs*)
Blank lines delimitates paragraphs.

roughsize(*size*, *above=20*, *mod=10*)
6 -> '6' 15 -> '15' 134 -> '130+'

to_timestamp(*dt*)

5.5.5 Module abilian.web.action

class ActionRegistry
The Action registry.

This is a Flask extension which registers `Action` sets. Actions are grouped by category and are ordered by registering order.

From your application use the instanciated registry `actions`.

The registry is available in jinja2 templates as *actions*.

actions(*context=None*)
Return a mapping of category => actions list.

Actions are filtered according to `Action.available()`.

if *context* is None, then current action context is used (`context`).

for_category(*category*, *context=None*)
Returns actions list for this category in current application.

Actions are filtered according to `Action.available()`.

if *context* is None, then current action context is used (`context`)

init_app(*app*)

installed(*app=None*)
Return *True* if the registry has been installed in current applications

register(**actions*)
Register *actions* in the current application. All *actions* must be an instance of `Action` or one of its subclasses.

If *overwrite* is *True*, then it is allowed to overwrite an existing action with same name and category; else *ValueError* is raised.

context
Return action context (dict type). Applications can modify it to suit their needs.

class Action(*category*, *name*, *title=None*, *description=None*, *icon=None*, *url=None*, *endpoint=None*, *condition=None*, *status=None*, *template=None*, *template_string=None*, *button=None*, *css=None*)
Action interface.

class Endpoint(*name*, **args*, ***kwargs*)

get_kwargs()
Hook for subclasses.

The key and values in the returned dictionnary can be safely changed without side effects on self.kwargs (provided you don't alter mutable values, like calling list.pop()).

Action.**available**(*context*)
Determine if this actions is available in this *context*.

Parameters **context** – a dict whose content is left to application needs; if condition is a callable it receives *context* in parameter.

Action.**get_render_args**(***kwargs*)

Action.**pre_condition**(*context*)
Called by available() before checking condition.

Subclasses may override it to ease creating actions with repetitive check (for example: actions that apply on a given content type only).

Action.**render**(***kwargs*)

Action.**url**(*context=None*)

Action.**CSS_CLASS = u'action action-{category} action-{category}-{name}'**

Action.**category = None**

Action.**condition = None**
A boolean (or something that can be converted to boolean), or a callable which accepts a context dict as parameter. See available().

Action.**description**

Action.**enabled**

Action.**endpoint**
A Endpoint instance, a string for a simple endpoint, a tuple (endpoint_name, kwargs) or a callable which accept a : context dict and returns one of those a valid values.

Action.**icon**

Action.**name = None**

Action.**status**

Action.**template_string = u'<a class="{{ action.css_class }}" href="{{ url }}">{%- if action.ico**

Action.**title**

class ModalActionMixin

template_string = u'<a class="{{ action.css_class }}" href="{{ url }}" data-toggle="modal">{%

5.5.6 Module abilian.web.nav

Navigation elements.

Abilian define theses categories:

***section*:** Used for navigation elements relevant to site section

***user*:** User for element that should appear in user menu

class **BreadcrumbItem**(*label=u''*, *url=u'#'*, *icon=None*, *description=None*)
A breadcrumb element has at least a label or an icon.

render()

description = None
Additional text, can be used as tooltip for example

icon = None
Icon to use.

label = None
Label shown to user. May be an i18n string instance

template_string = u'{%- if url %}<a href="{{ url }}">{%- endif %}{%- if item.icon %}{{ item.i

url

class **NavGroup**(*category*, *name*, *items=()*, **args*, ***kwargs*)
A navigation group renders a list of items.

append(*item*)

get_render_args(***kwargs*)

insert(*pos*, *item*)

status

template_string = '\n <ul class="nav navbar-nav {{ action.css_class }}">\n <li class="dropd

class **NavItem**(*category*, *name*, *divider=False*, **args*, ***kwargs*)
A single navigation item.

divider = False

path

status

5.5.7 Module abilian.web.forms

Extensions to WTForms fields, widgets and validators.

class **Form**(**args*, ***kwargs*)

class **FormPermissions**(*default=Role(u'anonymous')*, *read=None*, *write=None*, *fields_read=None*, *fields_write=None*)
Form role/permission manager

has_permission(*permission*, *field=None*, *obj=None*, *user=None*)

5.5.8 Module abilian.web.views

class **View**
Base class to use for all class based views.

The view instance is accessible in g and is set in actions context.

dispatch_request(**args*, ***kwargs*)

prepare_args(*args*, *kwargs*)
If view arguments need to be prepared it can be done here.

A typical use case is to take an identifier, convert it to an object instance and maybe store it on view instance and/or replace identifier by object in arguments.

redirect(*url*)
Shortcut all call stack and return response.

usage: *self.response(url_for(...))*

class **JSONView**
Base view for JSON GET.

Renders as JSON when requested by Ajax, renders as HTML when requested from browser.

data(**args*, ***kwargs*)
This method should return data to be serialized using JSON

get(**args*, ***kwargs*)

prepare_args(*args*, *kwargs*)

methods = ['GET']

class **Registry**(**args*, ***kwargs*)
Registry for default (canonical) views for entities.

There is one registry per application instance.

register(*entity*, *url_func*)
Associates a *url_func* with entity's type.

:param:entity: an abilian.core.extensions.db.Model class or instance.

:param:url_func: any callable that accepts an entity instance and return an url for it.

url_for(*entity=None*, *object_type=None*, *object_id=None*, ***kwargs*)
Returns canonical view for given entity instance.

If no view has been registered the registry will try to find an endpoint named with entity's class lowercased followed by '.view' and that accepts *object_id=entity.id* to generates an url.

Parameters

- **entity** – a instance of a subclass of `abilian.core.extensions.db.Model`, `whoosh.searching.Hit` or `dict`
- **object_id** – if *entity* is not an instance, this parameter must be set to target id. This is usefull when you know the type and id of an object but don't want to retrieve it from DB.

Raises KeyError if no view can be found for the given entity.

class default_view(*app_or_blueprint*, *entity*, *id_attr='object_id'*, *endpoint=None*, *kw_func=None*)

Decorator to register a view as default view for given entity class.

Parameters

- **id_attr** – url parameter name for object id.
- **endpoint** – endpoint to use, defaults to view function's name.
- **kw_func** – function to process keywords to be passed to url_for. Useful for additional keywords. This function receives: kw, obj, obj_type, obj_id, **kwargs. It must return kw.

class BaseObjectView(*Model=None*, *pk=None*, *base_template=None*, **args*, ***kwargs*)

Base class common to all database objects views

breadcrumb()

Return `nav.BreadcrumbItem` instance for this object.

This method may return a list of BreadcrumbItem instances. Return *None* if nothing.

get(**args*, ***kwargs*)

init_object(*args*, *kwargs*)

This method is reponsible for setting `obj`. It is called during `prepare_args()`.

prepare_args(*args*, *kwargs*)

Model = None

base_template = 'base.html'

methods = ['GET']

obj = None

pk = 'object_id'

template = None

template_kwargs
Template render arguments. You can override *base_template* for instance. Only *view* cannot be overriden.

title = None

class ObjectView(*Model=None*, *pk=None*, *Form=None*, *template=None*, **args*, ***kwargs*)
View objects

get_form_kwargs()

index_url()

prepare_args(*args*, *kwargs*)
`form` is initialized here. See also `View.prepare_args()`.

redirect_to_index()

Form = None

form = None

methods = ['GET']

permission = Permission(u'read')

template = 'default/object_view.html'

template_kwargs
provides `form` to templates

class ObjectEdit(*Model=None*, *pk=None*, *Form=None*, *template=None*, *view_endpoint=None*, *message_success=None*, **args*, ***kwargs*)
Edit objects

after_populate_obj()
Called after *self.obj* values have been updated, and *self.obj* attached to an ORM session.

before_populate_obj()
This method is called after form has been validated and before calling *form.populate_obj()*. Sometimes one may want to remove a field from the form because it's non-sense to store it on edited object, and use it in a specific manner, for example:

```
image = form.image
del form.image
store_image(image)
```

cancel()

commit_success()
Called after object has been successfully saved to database

edit()

form_csrf_invalid()
Called when a form doesn't validate *only* because of csrf token expiration.

This works only if form is an instance of `flask_wtf.form.SecureForm`. Else default CSRF protection (before request) will take place.

It must return a valid `Flask.Response` instance. By default it returns to edit form screen with an informative message.

form_invalid()
When a form doesn't validate this method is called.

It may return a `Flask.Response` instance, to handle specific errors in custom screens.

Else the edit form screen is returned with error(s) highlighted.

This method is useful for detecting edition conflict using hidden fields and show a specific screen to help resolve the conflict.

form_valid()
Save object.

Called when form is validated.

get_form_buttons(**args*, ***kwargs*)

handle_action(*action*)

handle_commit_exception(*exc*)
hook point to handle exception that may happen during commit.

It is the responsability of this method to perform a rollback if it is required for handling *exc*. If the method does not handle *exc* if should do nothing and return None.

Returns

- a valid `Response` if exception is handled.
- *None* if exception is not handled. Default handling happens.

message_success()

post(**args*, ***kwargs*)

prepare_args(*args*, *kwargs*)

put()

redirect_to_view()

validate()

view_url()

action = None

activity_target
Return *target* to use when creating activity.

activity_verb = 'update'

button = None

buttons

data = None

decorators = (<function support_graceful_failure at 0x7f9e9dc6aed8>,)

methods = ['GET', 'POST', 'PUT']

permission = Permission(u'write')

template = 'default/object_edit.html'

class ObjectCreate(**args*, ***kwargs*)

Create a new object

breadcrumb()

cancel()

chain_create()

create()

get_form_buttons(**args*, ***kwargs*)

get_form_kwargs()

init_object(*args*, *kwargs*)

activity_verb = 'post'

chain_create_allowed = False

methods = ['GET', 'POST', 'PUT']

class ObjectDelete(*Model=None*, *pk=None*, *Form=None*, *template=None*, *view_endpoint=None*, *message_success=None*, **args*, ***kwargs*)

Delete object. Supports DELETE verb.

delete()

get_form_buttons(**args*, ***kwargs*)

init_object(*args*, *kwargs*)

This method is reponsible for setting `obj`. It is called during `prepare_args()`.

activity_verb = 'delete'

methods = ['POST']

class JSONBaseSearch(**args*, ***kwargs*)

data(*q*, **args*, ***kwargs*)

get_item(*obj*)

Return a result item

Parameters **obj** – Instance object

Returns a dictionnary with at least *id* and *text* values

get_results(*q*, **args*, ***kwargs*)

prepare_args(*args*, *kwargs*)

Model = None

methods = ['GET']

minimum_input_length = 2

class JSONModelSearch(**args*, ***kwargs*)

Base class for json sqlalchemy model search, as used by select2 widgets for example

filter(*query*, *q*, ***kwargs*)

get_item(*obj*)

Return a result item

Parameters **obj** – Instance object

Returns a dictionnary with at least *id* and *text* values

get_label(*obj*)

get_results(*q*, **args*, ***kwargs*)

options(*query*)

order_by(*query*)

methods = ['GET', 'OPTIONS']

class JSONWhooshSearch(**args*, ***kwargs*)

Base class for JSON Whoosh search, as used by select2 widgets for example

get_item(*hit*)

Return a result item

Parameters **hit** – Hit object from Whoosh

Returns a dictionnary with at least *id* and *text* values

get_results(*q*, **args*, ***kwargs*)

methods = ['GET']

5.5.9 Module abilian.web.frontend

Front-end for a CRM app.

This should eventually allow implementing very custom CRM-style application.

class BaseEntityView(*module*, **args*, ***kwargs*)

breadcrumb()

check_access()

init_object(*args*, *kwargs*)

prepare_args(*args*, *kwargs*)

redirect_to_index()

can_create

can_delete

can_edit

pk = 'entity_id'

single_view

class CRUDApp(*app*, *modules=None*)

add_module(*module*)

create_blueprint(*module*)

class DefaultRelatedView(*label*, *attr*, *column_names*, *options=None*)
Default view used by Module for items directly related to entity

render(*entity*)

class EntityCreate(*module*, **args*, ***kwargs*)

breadcrumb()

check_access()

prepare_args(*args*, *kwargs*)

methods = ['GET', 'POST', 'PUT']

template = 'default/single_view.html'

template_kwargs

class EntityDelete(*module*, **args*, ***kwargs*)

methods = ['POST']

class EntityEdit(*module*, **args*, ***kwargs*)

methods = ['GET', 'POST', 'PUT']

template = 'default/single_view.html'

template_kwargs

class `EntityView`(*module*, **args*, ***kwargs*)

`methods` = ['GET']

`object_actions`

`template` = 'default/single_view.html'

`template_kwargs`

class `ListJson`(*module*, **args*, ***kwargs*)
JSON endpoint, for AJAX-backed table views.

`data`(**args*, ***kwargs*)

`methods` = ['GET']

class `Module`

`create_cls`
alias of `EntityCreate`

`delete_cls`
alias of `EntityDelete`

`edit_cls`
alias of `EntityEdit`

`json_search_cls`
alias of `JSONWhooshSearch`

`view_cls`
alias of `EntityView`

`create_blueprint`(*crud_app*)
Create a Flask blueprint for this module.

`get_grouped_actions`()

`init_related_views`()

`is_current`()

`list_json2`()
Other JSON endpoint, this time used for filling select boxes dynamically.

NB: not used currently.

`list_view`()

`ordered_query`(*request*, *query=None*)
Order query according to request args.

If query is None, the query is generated according to request args with self.query(request)

query(*request*)
Return filtered query based on request args

register_actions()

action_category

base_query
Return a query instance for `managed_class`.

base_template = 'base.html'

blueprint = None

edit_form_class = None

endpoint = None

id = None

label = None

list_view_columns = []

managed_class = None

name = None

read_query
Return a query instance for `managed_class` filtering on *READ* permission

related_views = []

search_criterions = (<abilian.web.search.criterion.TextSearchCriterion object at 0x7f9e9cc

single_view = None

static_folder = None

tableview_options = {}

url = None

view_form_class = None

view_new_save_and_add = False

view_options = None

view_template = None

class ModuleAction(*module*, *group*, *name*, **args*, ***kwargs*)
Base action class for `Module` actions.

Basic condition is simple: `category` must match the string *'module:{module.endpoint}'*

pre_condition(*context*)

class ModuleActionDropDown(*module*, *group*, *name*, **args*, ***kwargs*)

template_string = u'\n \n <button type="button" class="{{ action

class ModuleActionGroup(*module*, *group*, *name*, **args*, ***kwargs*)

template_string = u' {%

class ModuleActionGroupItem(*module*, *group*, *name*, **args*, ***kwargs*)

class ModuleMeta(*classname*, *bases*, *fields*)
Module metaclass.

Does some precalculations (like getting list of view methods from the class) to avoid calculating them for each view class instance.

class ModuleView(*module*, **args*, ***kwargs*)
Mixin for module base views.

Provide `module`.

module = None
`Module` instance

class RelatedView
A base class for related views

render(*entity*)
Return a dict with keys 'label', 'attr_name', 'rendered', 'size'

add_to_recent_items(*entity*, *type='ignored'*)

expose(*url='/'*, *methods=('GET',)*)
Use this decorator to expose views in your view classes.

url Relative URL for the view

methods Allowed HTTP methods. By default only GET is allowed.

labelize(*s*)

make_single_view(*form*, ***options*)

5.5.10 Module `abilian.web.tags`

register_plugin(*app*)

5.5.11 Module `abilian.web.util`

A few utility functions.

See https://docs.djangoproject.com/en/dev/topics/http/shortcuts/ for more ideas of stuff to implement.

capture_stream_errors(*logger*, *msg*)
Decorator that capture and log errors during streamed response.

Decorated function is automatically decorated with :func:<*Flask.stream_with_context*>.

@param logger: a logger name or logger instance @param msg: message to log

get_object_or_404(*cls*, **args*)
Shorthand similar to Django's *get_object_or_404*.

send_file_from_directory(*filename*, *directory*, *app=None*)
Helper to add static rules, like in *abilian.app*.app

Example use:

```
app.add_url_rule(
   app.static_url_path + '/abilian/<path:filename>',
   endpoint='abilian_static',
   view_func=partial(send_file_from_directory,
                     directory='/path/to/static/files/dir'))
```

url_for(*obj*, ***kw*)
Polymorphic variant of Flask's *url_for* function.

Behaves like the original function when the first argument is a string. When it's an object, it

5.6 Package abilian.testing

Elements to build test cases for an abilian.app.Application

class TestConfig
Base class config settings for test cases.

The environment variable SQLALCHEMY_DATABASE_URI can be set to easily test against different databases.

BABEL_DEFAULT_LOCALE = 'en'

BABEL_DEFAULT_TIMEZONE = 'UTC'

CELERY_ALWAYS_EAGER = True

CELERY_EAGER_PROPAGATES_EXCEPTIONS = True

CSRF_ENABLED = False

MAIL_SENDER = 'test@testcase.app.tld'

SECRET_KEY = 'SECRET'

SITE_NAME = u'Abilian Test'

SQLALCHEMY_DATABASE_URI = 'sqlite://'

SQLALCHEMY_ECHO = False

TESTING = True

TRAP_BAD_REQUEST_ERRORS = True

TRAP_HTTP_EXCEPTIONS = False

WTF_CSRF_ENABLED = False

class BaseTestCase(*methodName='runTest'*)
Base test case to test an `abilian.app.Application`.

It will create an instance path that will be used and shared for all tests defined in this test case.

The test case creates a clean database before running each test by calling `abilian.app.Application.create_db()` et destroys it after test.

Additionaly if the database is postgresql a schema is created for each test and the connection role is altered to use this DB schema. This is done to ensure harder test isolation.

application_class
Application class to instantiate.

alias of `Application`

config_class
Config class to use for `application_class` configuration.

alias of `TestConfig`

assert_302(*response*)

assert_valid(*response*)
Validate *response.data* as HTML using validator provided by *config.VALIDATOR_URL*.

client_login(*email*, *password*)
Like `login()` but with a web login request. Can be used as contextmanager.

All subsequent request made with *self.client* will be authentifed until `client_logout()` is called or exit of *with* block.

client_logout()
Like `logout()` but with a web logout

create_app()

get(*url*, *validate=True*)
Validates HTML if asked by the config or the Unix environment.

get_setup_config()
Called by `create_app()` Override this if you want to tweak the config before `application_class` is instanciated.

Returns an instance of `config_class`, or anything that is a valid config object for Flask.

login(*user*, *remember=False*, *force=False*)
Perform user login for *user*, so that code needing a logged-in user can work.

This method can also be used as a context manager, so that logout is performed automatically:

```
with self.login(user):
    self.assertEquals(...)
```

See also:

`logout()`

login_system()
Perform login with SYSTEM user. Can be used as a context manager.

See also:

`login()`, `logout()`

logout()
Perform user logout.

See also:

`login()`

setUp()

classmethod setUpClass()

tearDown()

classmethod tearDownClass()

validate(*url*, *content*, *content_type*, *validator_url*)

CLEAR_PASSWORDS = True
set to False to use cryptographic scheme (standard) for user password. By default the testcase switches to clear text to avoid longer running.

SERVICES = ()
list services names that should be started during setUp. 'repository' and 'session_repository' services are always started, do not list them here.

SQLALCHEMY_WARNINGS_AS_ERROR = True
By default sqlalchemy treats warnings as info. This settings makes sqlalchemy warnings treated as errors (and thus making test fail). The rationale is that it improves code quality (for example most frequent warnings are non-unicode string assigned on a Unicode column; this setting force you to be explicit and ensure unicode where appropriate)

TESTING_BUILD_ASSETS = False
enable assets building during tests. False by default.

TEST_INSTANCE_PATH = None
Path to instance folder. Mostly set for internal use, since you should access the value on the application (see Flask instance folders) This parameter is set by `setUpClass()`

db

Shortcut to the application db object.

CHAPTER 6

Changelog for Abilian Core

6.1 0.4.5 (unreleased)

TODO

6.2 0.4.4 (2015-08-07)

6.2.1 Design / UI

- Navbar is now non-fluid.

6.2.2 Updates

- Upgrade Jinja to 2.8 and Babel to 2.0

6.2.3 Fixes

- Fixed image cropping.

6.3 0.4.3 (2015-07-29)

Another release because there was a version number issue with the previous one.

6.4 0.4.2 (2015-07-29)

6.4.1 Bugfixes / cleanup

- Replace Scribe by CKEditor for better IE compatibility.
- Smaller bug fixes and code cleanups

6.5 0.4.1 (2015-07-21)

6.5.1 Bugfixes / cleanup

- permission: no-op when service not running
- JS fixes
- CSS fixes
- https://github.com/mitsuhiko/flask/issues/1135

6.6 0.4.0 (2015-07-15)

6.6.1 Features

- Object level permissions
- Add “meta” properties to entities
- Attached files to entities
- More flexible search filters
- Avatars
- Tag engine (alpha)

6.6.2 Fixes / cleanup

- JS: Upate ravenjs, requirejs, bootbox, jquery, scribe

6.7 0.3.6 (2015-05-27)

6.7.1 Fixes

- security service: fix exception on has_role()

6.8 0.3.5 (2015-05-27)

6.8.1 Features

- default user avatar is now a circle with their last name initial (#12)
- add PRIVATE_SITE, app, blueprint and endpoint access controller registration
- Better handling of CSRF failures
- add dynamic row widget js
- js: add datatable advanced search

6.8.2 Fixes

- CSS (Bootstrap) fixes
- Permissions fixes

6.8.3 Updates

- Updated Bootstrap to 3.3.4
- Updated flask-login to 0.2.11
- Updated Sentry JS code to 1.1.18

6.9 0.3.4 (2015-04-14)

- updated Select2 to 3.5.2
- enhanced fields and widgets
- set default SQLALCHEMY_POOL_RECYCLE to 30 minutes
- Users admin panel: fix roles not set; fix all assignable roles not listed; fix cannot set password during user creation.

6.10 0.3.3 (2015-03-31)

6.10.1 Features

- Use ravenjs to monitor JS errors with Sentry
- Vocabularies

6.11 0.3.2 (2014-12-23)

- Minor bugfixes

6.12 0.3.1 (2014-12-23)

- Minor bugfixes

6.13 0.3.0 (2014-12-23)

6.13.1 Features

- Added a virus scanner.
- Changed the WYSIWYG editor to Scribe.
- Vocabularies

6.13.2 API changes

- Deprecated the @templated decorator (will be removed in 0.4.0).

6.13.3 Building, tests

- Build: Use pbr to simplify setup.py.
- Dependencies: moved deps to ./requirements.txt + cleanup / update.
- Testing: Tox and Travis config updates.
- Testing: Run tests under Vagrant.
- QA: Fixed many pyflakes warnings.

6.14 0.2.0 (2014-08-07)

- Too long to list.

6.15 0.1.4 (2014-03-27)

- refactored abilian.core.entities, abilian.core.subjects. New module abilian.core.models containing modules: base, subjects, owned.
- Fixed or cleaned up dependencies.
- Fixed setupwizard.
- added config value: BABEL_ACCEPT_LANGUAGES, to limit supported languages and change order during negociation
- Switched CSS to LESS.
- Updated to Bootstrap 3.1.1

6.16 0.1.3 (2014-02-03)

- Update some dependencies
- Added login/logout via JSON api
- Added 'createuser' command

6.17 0.1.2 (2014-01-11)

- added jinja extension to collect JS snippets during page generation and put them at end of document ("deferred")
- added basic javascript to prevent double submission
- Added Flask-Migrate

6.18 0.1.1 (2013-12-26)

- Redesigned indexing:
  - single whoosh index for all objects
  - search results page do not need anymore to fetch actual object from database
  - index security information, used for filtering search results
  - Added "reindex" shell command

6.19 0.1 (2013-12-13)

- Initial release.

CHAPTER 7

Credits

7.1 Design, programming

Abilian development team: 2012-2013.

7.2 Art (images, icons)

See links for copyright and licences (usually Creative Commons).

- Background image(s) for login: Kevin Dooley http://www.flickr.com/photos/pagedooley/.
- File types icons: http://www.splitbrain.org/projects/file_icons.
- Animal pictures: http://en.wikipedia.org/wiki/Wikipedia:Featured_pictures/Animals#Anim
- Group pictures: http://en.wikipedia.org/wiki/File:Estudiante_INTEC.jpg, http://en.wikipedia.org/wiki/File:Salesman_-beach_-_bikini-_sun-27Dec2008.jpg, http://en.wikipedia.org/wiki/File:Cocacola-5cents-1900_edit1.jpg.
- Group Icon (CC Attribution-Noncommercial-No Derivate 3.0): http://www.iconarchive.com/show/soft-scraps-icons-by-deleket/User-Group-icon.html.

Part II

INDICES AND TABLES

- genindex
- modindex
- search

Index

Symbols

A

B

C

D

E

F

G

H

I

J

L

M

N

O

P

Q

R

S

T

U

V

W