invenio-base Documentation

Release 1.0.2

CERN

Dec 17, 2018

Contents

Base package for building Invenio application factories.

Further documentation is available on https://invenio-base.readthedocs.io/

CHAPTER 1

User's Guide

This part of the documentation will show you how to get started in using Invenio-Base.

1.1 Installation

Invenio-Base is on PyPI so all you need is:

```
$ pip install invenio-base
```

1.2 Usage

Invenio application loader.

1.2.1 Quickstart

Invenio-Base is taking advantage of advanced patterns for building Flask application. It assumes you already have understanding of patterns for Flask.

Dependencies

First we need to install and import dependencies:

```
$ mkvirtualenv example
(example)$ pip install invenio-base
```

Now you can create new file `app.py` with following imports:

```
import os
import sys

from invenio_base.app import create_app_factory, create_cli
from invenio_base.wsgi import create_wsgi_factory
```

Configuration

Tell the application factory how to load configuration by creating `config_loader` function that accepts an application instance:

```
class Config(object):
    """Example configuration."""

    DEBUG = True
    SECRET_KEY = 'CHANGE_ME'


def config_loader(app, **kwargs):
    """Custom config loader."""
    app.config.from_object(Config)
    app.config.update(**kwargs)
```

The recommended way is to use Invenio-Config that provides a default configuration loader `invenio_config.utils.create_config_loader()` which is sufficient for most cases:

```
from invenio_config import create_config_loader
config_loader = create_config_loader(config=Config, env_prefix='APP')
```

In the next step you should set an absolute path for the *instance folder* in order to load configuration files and other data from deployment specific location. The instance folder is also perfect place for dropping static files if you do not serve them from CDN:

```
env_prefix = 'APP'

instance_path = os.getenv(env_prefix + '_INSTANCE_PATH') or \
    os.path.join(sys.prefix, 'var', 'example-instance')
"""Instance path for Invenio.

Defaults to ``<env_prefix>_INSTANCE_PATH`` or if environment variable is not
set ``<sys.prefix>/var/<app_name>-instance``.
"""

static_folder = os.getenv(env_prefix + '_STATIC_FOLDER') or \
    os.path.join(instance_path, 'static')
"""Static folder path.

Defaults to ``<env_prefix>_STATIC_FOLDER`` or if environment variable is not
set ``<sys.prefix>/var/<app_name>-instance/static``.
"""
```

In our example the variables are read from environment variables first with the purpose that they can be easily changed without modifying code for various deployment usecases.

Combining Applications

It is highly recommendended to separate Invenio UI and REST applications then different exception handlers, URL converters and session management can be installed on each application instance. You can even install your own WSGI application side by side with Invenio ones.

Invenio packages provide apps (extensions), blueprints, and URL converters via entry points `invenio_base.[api_]<apps,blueprints,converters>`. You can specify multiple entry point groups for each application factory (e.g. `myservice.blueprints`):

```
create_api = create_app_factory(
    'example',
    config_loader=config_loader,
    blueprint_entry_points=['invenio_base.api_blueprints'],
    extension_entry_points=['invenio_base.api_apps'],
    converter_entry_points=['invenio_base.api_converters'],
    instance_path=instance_path,
)


create_app = create_app_factory(
    'example',
    config_loader=config_loader,
    blueprint_entry_points=['invenio_base.blueprints'],
    extension_entry_points=['invenio_base.apps'],
    converter_entry_points=['invenio_base.converters'],
    wsgi_factory=create_wsgi_factory({'/api': create_api}),
    instance_path=instance_path,
    static_folder=static_folder,
)
```

You provide instances of your own apps, blueprints, or URL converters directly to the factory:

```
from flask import Blueprint

blueprint = Blueprint('example', __name__)

@blueprint.route('/')
def index():
    return 'Hello from Example application.'

 create_app = create_app_factory(
     'example',
     blueprints=[blueprint],
     # other parameters as shown in previous example
 )
```

Running

To run you application you need to first instantiate the application object:

```
app = application = create_app()
"""The application object."""
```

Then you need to tell the **"flask"** command where is your file located by setting environment variable `FLASK_APP=app.py`:

```
$ export FLASK_APP=app.py
$ flask run
```

If you prefer to make your own executable script then you can use following pattern:

```
from invenio_base.app import create_cli

cli = create_cli(create_app=create_app)

if __name__ == '__main__':
    cli()
```

Do not worry, you do not have to write all this by yourself. Follow next steps and use `inveniomanage` command that generates the scaffold code for you.

1.2.2 The `inveniomanage` command

Invenio-Base installs the `inveniomanage` command. By default only three subcommands are available:

```
$ inveniomanage --help
Usage: inveniomanage [OPTIONS] COMMAND [ARGS]...

  Command Line Interface for Invenio.

Options:
  -a, --app TEXT        The application to run.
  --debug / --no-debug  Enable or disable debug mode.
  --help                Show this message and exit.

Commands:
  run              Run development server.
  shell            Run shell in the app context.
```

The `run` and `shell` commands only works if you have specified the `--app` option or the `FLASK_APP` environment variable. See Flask documentation for further information.

Listing all entrypoints of an Invenio instance

The `instance entrypoints` subcommand helps you list all entrypoints of your Invenio application:

```
$ inveniomanage instance entrypoints
```

The output of the command will be in the below format:

```
<entrypoint_group_name>
  <entrypoint>
```

You can also restrict the output of the command to list all entrypoints for a specific entrypoint group by passing the name via the *-e* option:

```
$ inveniomanage instance entrypoints -e <entrypoint_group_name>
```

For further details about the available options run the *help* command:

```
$ inveniomanage instance entrypoints --help
...
```

Migrating the application's old secret key

The `instance migrate_secret_key` subcommand helps you migrate your application's old secret key:

```
$ inveniomanage instance migrate_secret_key --old-key <old_key>
```

The purpose of this command is to provide the administrator the capability to change the Invenio application's secret_key and migrate that change in all database's EncryptedType properties through an entrypoint group called *invenio_base.secret_key'*. There you can specify your migration function that will receive the old secret_key that can be used to decrypt the old properties and encrypt them again with the application's new secret_key.

You can register your migration function as shown below in your package's entrypoints in the setup.py:

```
entrypoints= {
    'invenio_base.secret_key': [
        '<entrypoint_name> = <entrypoint_function>'
    ]
}
```

Also you can see an example of use in invenio_oauthclient package's setup.py.

Note: You should change your application's *secret_key* in the config before calling the migration command.

For further details about the available options run the *help* command:

```
$ inveniomanage instance migrate_secret_key --help
...
```

CHAPTER 2

API Reference

If you are looking for information on a specific function, class or method, this part of the documentation is for you.

2.1 API Docs

2.1.1 Application and CLI factory

Invenio application factory.

invenio_base.app.**app_loader**(*app*, *entry_points=None*, *modules=None*)

Run default application loader.

Parameters

- **entry_points** – List of entry points providing to Flask extensions.
- **modules** – List of Flask extensions.

invenio_base.app.**base_app**(*import_name*, *instance_path=None*, *static_folder=None*, *static_url_path='/static'*, *template_folder='templates'*, *instance_relative_config=True*, *app_class=<class 'flask.app.Flask'>*)

Invenio base application factory.

If the instance folder does not exists, it will be created.

Parameters

- **import_name** – The name of the application package.
- **env_prefix** – Environment variable prefix.
- **instance_path** – Instance path for Flask application.
- **static_folder** – Static folder path.
- **app_class** – Flask application class.

Returns Flask application instance.

invenio_base.app.**blueprint_loader**(*app*, *entry_points=None*, *modules=None*)
Run default blueprint loader.

The value of any entry_point or module passed can be either an instance of `flask.Blueprint` or a callable accepting a `flask.Flask` application instance as a single argument and returning an instance of `flask.Blueprint`.

Parameters

- **entry_points** – List of entry points providing to Blueprints.
- **modules** – List of Blueprints.

invenio_base.app.**configure_warnings**()
Configure warnings by routing warnings to the logging system.

It also unhides `DeprecationWarning`.

invenio_base.app.**converter_loader**(*app*, *entry_points=None*, *modules=None*)
Run default converter loader.

Parameters

- **entry_points** – List of entry points providing to Blue.
- **modules** – Map of coverters.

invenio_base.app.**create_app_factory**(*app_name*, *config_loader=None*, *extension_entry_points=None*, *extensions=None*, *blueprint_entry_points=None*, *blueprints=None*, *converter_entry_points=None*, *converters=None*, *wsgi_factory=None*, ***app_kwargs*)
Create a Flask application factory.

The application factory will load Flask extensions and blueprints specified using both entry points and directly in the arguments. Loading order of entry points are not guaranteed and can happen in any order.

Parameters

- **app_name** – Flask application name.
- **config_loader** – Callable which will be invoked on application creation in order to load the Flask configuration. See example below.
- **extension_entry_points** – List of entry points, which specifies Flask extensions that will be initialized only by passing in the Flask application object
- **extensions** – List of Flask extensions that can be initialized only by passing in the Flask application object.
- **blueprint_entry_points** – List of entry points, which specifies Blueprints that will be registered on the Flask application.
- **blueprints** – List of Blueprints that will be registered on the Flask application.
- **converter_entry_points** – List of entry points, which specifies Werkzeug URL map converters that will be added to `app.url_map.converters`.
- **converters** – Map of Werkzeug URL map converter classes that will be added to `app.url_map.converters`.
- **wsgi_factory** – A callable that will be passed the Flask application object in order to overwrite the default WSGI application (e.g. to install `DispatcherMiddleware`).
- **app_kwargs** – Keyword arguments passed to *base_app()*.

Returns Flask application factory.

Example of a configuration loader:

```
def my_config_loader(app, **kwargs):
    app.config.from_module('mysite.config')
    app.config.update(**kwargs)
```

Note: Invenio-Config provides a factory creating default configuration loader (see `invenio_config.utils.create_config_loader()`) which is sufficient for most cases.

Example of a WSGI factory:

```
def my_wsgi_factory(app):
    return DispatcherMiddleware(app.wsgi_app, {'/api': api_app})
```

invenio_base.app.**create_cli**(*create_app=None*)
Create CLI for `inveniomanage` command.

Parameters **create_app** – Flask application factory.

Returns Click command group.

2.1.2 WSGI factory

WSGI application factory for Invenio.

invenio_base.wsgi.**create_wsgi_factory**(*mounts_factories*)
Create a WSGI application factory.

Usage example:

```
wsgi_factory = create_wsgi_factory({'/api': create_api})
```

Parameters **mounts_factories** – Dictionary of mount points per application factory.

New in version 1.0.0.

invenio_base.wsgi.**wsgi_proxyfix**(*factory=None*)
Fix `REMOTE_ADDR` based on `X-Forwarded-For` headers.

Note: You must set `WSGI_PROXIES` to the correct number of proxies, otherwise you application is susceptible to malicious attacks.

New in version 1.0.0.

2.1.3 Signals

Signals for application creation.

invenio_base.signals.**app_created = <blinker.base.NamedSignal object at 0x7f5c6581c4d0; 'app**
Signal sent when the base Flask application have been created.

Parameters: - `sender` - the application factory function. - `app` - the Flask application instance.

Example receiver:

```
def receiver(sender, app=None, **kwargs):
    # ...
```

invenio_base.signals.**app_loaded = <blinker.base.NamedSignal object at 0x7f5c6581c510; 'app-**

Signal sent when the Flask application have been fully loaded.

Parameters: - `sender` - the application factory function. - `app` - the Flask application instance.

Example receiver:

```
def receiver(sender, app=None, **kwargs):
    # ...
```

CHAPTER 3

Additional Notes

Notes on how to contribute, legal information and changes are here for the interested.

3.1 Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

3.1.1 Types of Contributions

Report Bugs

Report bugs at https://github.com/inveniosoftware/invenio-base/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" is open to whoever wants to implement it.

Implement Features

Look through the GitHub issues for features. Anything tagged with "feature" is open to whoever wants to implement it.

Write Documentation

Invenio-Base could always use more documentation, whether as part of the official Invenio-Base docs, in docstrings, or even on the web in blog posts, articles, and such.

Submit Feedback

The best way to send feedback is to file an issue at https://github.com/inveniosoftware/invenio-base/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

3.1.2 Get Started!

Ready to contribute? Here's how to set up *invenio-base* for local development.

1. Fork the *inveniosoftware/invenio-base* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/invenio-base.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ mkvirtualenv invenio-base
$ cd invenio-base/
$ pip install -e .[all]
```

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass tests:

```
$ ./run-tests.sh
```

   The tests will provide you with test coverage and also check PEP8 (code style), PEP257 (documentation), flake8 as well as build the Sphinx documentation and run doctests.

6. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -s
    -m "component: title without verbs"
    -m "* NEW Adds your new feature."
    -m "* FIX Fixes an existing issue."
    -m "* BETTER Improves and existing feature."
    -m "* Changes something that should not be visible in release notes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

3.1.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests and must not decrease test coverage.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring.
3. The pull request should work for Python 2.7, 3.3, 3.4 and 3.5. Check https://travis-ci.org/inveniosoftware/invenio-base/pull_requests and make sure that the tests pass for all supported Python versions.

3.2 Changes

Version 1.0.2 (released 2018-12-14)

Version 1.0.1 (released 2018-05-25)

- Added support for blueprint factory functions in the `invenio_base.blueprints` and the `invenio_base.api_blueprints` entry point groups. In addition to specifying an import path to an already created blueprint, you can now specify an import path of a blueprint factory function with the signature create_blueprint(app), that will create and return a blueprint. This allows moving dynamic blueprint creation from the extension initialization phase to the blueprint registration phase.

Version 1.0.0 (released 2018-03-23)

- Initial public release.

3.3 License

3.4 Contributors

- Alizee Pace
- Chiara Bigarella
- Dinos Kousidis
- Harris Tzovanakis
- Javier Delgado
- Jiri Kuncar
- Krzysztof Nowak
- Lars Holm Nielsen
- Leonardo Rossi
- Marco Neumann
- Paulina Lach
- Rémi Ducceschi
- Sami Hiltunen
- Sebastian Witowski
- Tibor Simko
- Yoan Blanc
- Zacharias Zacharodimos

Python Module Index

i

Index