pytest-invenio Documentation

Release 2.1.4

CERN

Jun 02, 2023

CONTENTS

Pytest fixtures for Invenio.

The package offers a number of features to help test Invenio based applications:

- Less boilerplate: Using the fixtures you can keep your `conftest.py` short and focused.
- Database re-use: database tests are running inside a transaction which is rolled back after the test.
- End-to-end testing: Selenium tests can easily be switched on/off, and in case of test failures a screenshot is taken (with possibility to output in the console in base64-encoding - useful on e.g. TravisCI).
- Application configuration for testing (e.g. disable CSRF protection in forms and HTTPS requirement).
- JSON decoding support in Flask test client for easier API testing.
- Batteries included: further fixtures help with e.g. mail sending and CLI tests.

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

CHAPTER

ONE

USER'S GUIDE

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

1.1 Installation

pytest-invenio is on PyPI so all you need is:

```
$ pip install pytest-invenio
```

Normally, you would add it to your package's `setup.py` to have it automatically installed:

```
setup(
    # ...
    setup_requires=[
        'pytest-runner>=3.0.0,<5',
    ],
    tests_require=[
        'pytest-invenio>=1.0.0,<1.1.0',
    ]
)
```

Tip: Add the following alias to your `setup.cfg`:

```
[aliases]
test = pytest
```

In this way, the standard Python way of executing test still works:

```
$ python setup.py test
```

1.2 Usage

Pytest fixtures for Invenio.

1.2.1 Quick start

1. Define a module-scoped fixture named `create_app` that returns an application factory for your Invenio installation. If you are using Invenio-App, it's as simple as:

```
# conftest.py
from invenio_app.factory import create_ui

@pytest.fixture(scope='module')
def create_app():
    return create_ui
```

2. Write tests:

```
# test_something.py

def test_e2e(live_server, browser):
    browser.get(url_for('index', _external=True))

def test_testclient(client):
    res = client.get('/api/')
    res.json == {'test-client': 'with-json-decoder'}

def test_db(base_app, db):
    # Database with rollback

def test_cli(cli_runner):
    result = cli_runner(mycmd)
    assert result.exit_code == 0

def test_mailbox(appctx, mailbox):
    # ...
    assert len(mailbox) == 1
```

1.2.2 Running tests

Running tests with py.test is pretty simple. Your package might support the standard way of running tests:

```
$ python setup.py test
```

Alternatively you can use the `pytest` command to run all or specific test cases:

```
$ pytest
$ pytest tests/test_something.py
$ pytest tests/test_something.py::test_acase
```

1.2.3 Fixtures

All available fixtures are documented in the API documentation (see *Fixtures*).

In addition to the ones provided by pytest-invenio, there are further fixtures defined pytest-flask (see documentation for details).

1.2.4 Structuring tests

The pytest fixtures in pytest-invenio all work on *one* Flask application, however most Invenio instances usually consits of *two* Flask applications: UI and API. Thus, to use the pytest-invenio fixtures it's important to understand how to structure your tests, and know exactly which application you are dealing with.

Scope

Most of pytest-invenio fixtures are either *module* scoped or *function* scoped.

- *Module* scoped fixtures are created/destroyed once per Python test file.
- *Function* scoped fixtures are created/destroyed per test.

The fixtures which creates the database and applications are module scoped, hence, all tests in a Python file run against either the UI or the API application, but not both.

Note: All tests in a single file, run against the one and only one application (e.g. UI or REST).

Thus, in a single test file you cannot mix both UI and API tests, which is normally not an issue.

Overriding fixtures

Pytest provides rich support for overriding fixtures at various level and combined with module/function-scoped we can easily override fixtures. Also, you can use `conftest.py` to define per-directory fixtures.

Following is an example of how fixtures overriding works:

```
# conftest.py:
@pytest.fixture()
def myfix():
    return 'root'

# test_root.py
def test_a(myfix):
    print(myfix)
    # will output "root"

# a/conftest.py
@pytest.fixture()
def myfix(myfix):
    return myfix + '-a'

# a/test_subdir.py
def test_a(myfix):
```

(continues on next page)

(continued from previous page)

```
    print(myfix)
    # will output "root-a"
```

Notice that:

- **Overriding:** In `a/test_subdir.py` the fixture `myfix` is coming from `a/conftest.py` which is overriding the fixture from `conftest.py`. In `test_root.py` it's however the `myfix` fixture from `conftest.py` being used.
- **Parent fixture:** In `a/conftest.py`, the fixture `myfix` has access to the parent fixture from `conftest.py`.

Recommend layout

If you are using Invenio-App (recommended), then the following layout is recommended:

```
# ### tests/conftest.py ###
# Common application configuration goes here
@pytest.fixture(scope='module')
def app_config(app_config):
    app_config['MYCONF'] = True
    return app_config

# ### tests/ui/conftest.py ###
# UI tests goes in tests/ui/ folder.
from invenio_app.factory import create_ui

@pytest.fixture(scope='module')
def create_app():
    return create_ui

# ### tests/api/confest.py ###
# API tests goes in tests/api/ folder.
from invenio_app.factory import create_api

@pytest.fixture(scope='module')
def create_app():
    return create_api

# ### tests/e2e/conftest.py ###
# E2E tests (requring both UI/API) goes in tests/e2e/ folder.
from invenio_app.factory import create_app as create_ui_api

@pytest.fixture(scope='module')
def create_app():
    return create_ui_api
```

Using above layout you essentially split your tests into three folders:

```
tests/ui/
tests/api/
tests/e2e/
```

Each subfolder holds tests related to a specific application (UI or API). The `e2e` folder holds tests that need both UI and API application (which is typically the case for end-to-end tests). The E2E tests works by creating both the UI and

API applications and using a special WSGI middleware to dispatch requests between both applications. Having two applications at the same time, can however cause quite a lot of confusion so it is only recommended for E2E tests.

Note, also in above example how all three applications are sharing the same *app_config* fixture.

Note: You shouldn't feel bound to above structure. If you site grows large, you'll likely split tests into further subfolders. The important message from the recommended layout, is that you need **one folder per application**.

1.2.5 Application fixtures

The package provides three different application fixtures:

- *base_app*: Basic application fixture which creates the Flask application.
- *appctx*: Same as the basic application fixture, but pushes an application context onto the stack (i.e. makes `current_app` work).
- *app*: Same as the basic application, but in addition it initializes the database and search indices.

All three fixtures depend on the same user-provided (i.e. you must define it) fixture named `create_app` which must return an application factory (see *Quick start*).

Customizing configuration

The application fixtures rely on fixtures such as *instance_path*, *app_config*, *celery_config_ext*, *db_uri*, *broker_uri* to inject configuration into the application.

You can overwrite each of these fixtures at many different levels:

- **Global**: Override one or more of these fixtures in your global `conftest.py` to inject the same configuration in all applications.
- **Per-directory**: Override fixtures for a specific subdirectory by putting a `conftest.py` in the directory.
- **Per-file**: Fixtures can also be overwritten in specific modules. For instance you may want to customize the celery configuration only for a specific Python test file.

Injecting entry points

Invenio relies heavily upon entry points for constructing a Flask application, and it can be rather cumbersome to try to manually register database models, mappings and other features afterwards.

You can therefore inject extra entry points if needed during testing via the *extra_entry_points* fixture and use it in your custom `create_app()` fixture:

```
@pytest.fixture(scope="module")
def extra_entry_points():
    return {
        'invenio_db.models': [
            'mock_module = mock_module.models',
        ]
    }

@pytest.fixture(scope="module")
```

(continues on next page)

(continued from previous page)

```
def create_app(entry_points):
    return _create_api
```

Note that `create_app()` depends on the *entry_points* fixture not the `extra_entry_points()`.

1.2.6 Views testing

Views can easily be testing using the Flask test clients. Two test clients are provided for convenience: `base_client` and `client`. The only difference is which application fixture they depend on:

```
def test_view1(base_client):
    # Depends on 'base_app' fixture
    base_client.get(url_for(..))

def test_view2(client):
    # Depends on 'app' fixture
    client.get(url_for(..))
```

JSON responses

The default Flask test client does not have built-in support for decoding JSON responses, which can make API testing a bit cumbersome. The test clients are therefore patched to add a JSON property:

```
def test_api(base_client):
    res = base_client.get(...)
    assert res.json == { ... }
```

1.2.7 Database re-use

The default database is an SQLite database located in the application's instance folder. This can easily be overwritten by setting the environment variable `SQLALCHEMY_DATABASE_URI` (useful e.g. in CI systems to test multiple databases).

Tests that make changes to the database should explicitly use the function scoped *db* fixture. This fixture wraps the changes in a transaction and rollback any changes by the end of the test. For instance:

```
def test_db1(db):
    db.session.add(User(username='alice'))
    db.session.commit()
    assert User.query.count() == 1 # i.e. independent of test_db2

def test_db2(db):
    db.session.add(User(username='bob'))
    db.session.commit()
    assert User.query.count() == 1  # i.e. independent of test_db1
```

Note: Take care! The *db* fixture does not rollback other changes. If data, in addition to being added to the database, is also indexed in the search cluster then you should clear the index explicitly using e.g. *search_clear*.

Performance considerations

The database is recreated (all tables dropped and recreated) for each test file, because the database is a module scoped fixture. This adds a performance overhead, thus be careful not to indirectly depend on the database fixtures in a file unless it is really necessary (e.g. via the *app* fixture).

1.2.8 Search testing

Pytest-Invenio depends on Invenio-Search and any mappings registered on Invenio-Search will be created if you depend on the *search* fixture. The fixture is module scoped, meaning that any fixture you write to e.g. load test data should likely also be module scoped.

Clearing changes

Unlike the database fixture, which automatically rollback changes, you must explicitly depend on the *search_clear* fixture if you makes changes to the indexes. This ensures that you leave the indexes in a clean state for the next test. The *search_clear* fixture will however delete and recreate the indexes, and thus comes with a performance penalty if used.

```
def test_search1(search_clear):
    # ...
```

Performance considerations

As for the database fixtures, search indices are deleted and recreated for each test file (due to module scoped fixture). Thus be careful not to indirectly depend on the database fixtures in a file unless it is really necessary (e.g. via the *app* fixture).

1.2.9 CLI testing

Pytest-Invenio provides two quick short cuts for easier testing Click-based commands that require an application context (i.e. most commands).

The shortest version is to use the *cli_runner* fixture:

```
def test_cmd(cli_runner):
    result = cli_runner(mycmd)
    assert result.exit_code == 0
```

The downside is that the Click CLIRunner is recreated for each call. This is not necessary, so an alternative is to use the *script_info* fixture, which however is more verbose:

```
def test_cmd(script_info):
    runner = CliRunner()
    result = runner.invoke(mycmd, obj=script_info)
    assert result.exit_code == 0
```

1.2.10 Mail testing

If you have Invenio-Mail installed on your application, you can use the *mailbox* fixture to test email sending. Any message sent by the application during the test will be captured and is inspectable in via the fixture:

```
def test_mailbox(appctx, mailbox):
    assert len(mailbox) == 0
    appctx.extensions['mail'].send_message(
        sender='no-reply@localhost',
        subject='testing',
        body='test',
        recipients=['no-reply@localhost'],)
    assert len(mailbox) == 1
```

1.2.11 End-to-end testing

In addition to using the Flask test client for testing views (see *Views testing*), you can use a real browser via the Selenium integration for fully end-to-end testing. The tests works by starting the Flask application in a separate process, and using Selenium to drive your favorite browser. Writing the tests are very easy, simply depend on the `live_server` fixture (defined by pytest-flask) and the *browser* fixture:

```
def test_browser(live_server, browser):
    # Note the use of '_external=True'
    browser.get(url_for('index', _external=True))
```

Running E2E tests

By default, tests using the *browser* fixture are skipped. In order to run these tests, you must set an environment variable:

```
$ export E2E=yes
```

Also, by default Chrome is used. If you'd like to use Firefox, Safari or another browser you must set another environment variable:

```
$ export E2E_WEBDRIVER_BROWSERS="Firefox"
```

Note: You must have Selenium Client and the Chrome Webdriver installed on your system in order to run the E2E tests.

Screenshots

The *browser* fixture will take a screenshot of in case of test failures and store it in a folder `.e2e_screenshots`. On CI systems you can also have screenshot printed to the console by setting an environment variable:

```
$ export E2E_OUTPUT=base64
```

TravisCI integration

Following is an example of the needed changes (at time of writing) to your `.travis.yml` in case want to run E2E tests on Travis. Travis is likely to evolve, so please refer to the Travis CI documentation for the latest information.

```
# Install Chrome
# - see https://docs.travis-ci.com/user/chrome
addons:
  chrome: stable

# Chrome driver fails if not trusty dist
dist: trusty

# Selenium webdriver for Chrome fails if not on sudo
# - see https://github.com/travis-ci/travis-ci/issues/8836
sudo: true

# Define environment variables to enable E2E tests and outputing
# screenshots to the console.
env:
  global:
    # Print screenshots to console output
    - E2E_OUTPUT=base64
    # Enable end-to-end tests
    - E2E=yes

# Install Chrome webdriver for Selenium
before_install:
  - "PATH=$PATH:$HOME/webdrivers"
  - "if [ ! -f $HOME/webdrivers/chromedriver ]; then wget https://chromedriver.storage.
↪googleapis.com/2.31/chromedriver_linux64.zip -P   $HOME/webdrivers; unzip -d $HOME/
↪webdrivers $HOME/webdrivers/chromedriver_linux64.zip; fi"  # noqa

# Start a virtual display
# - https://docs.travis-ci.com/user/gui-and-headless-browsers/
before_script:
  - "export DISPLAY=:99.0"
  - "sh -e /etc/init.d/xvfb start"
  - sleep 3 # give xvfb some time to start
```

CHAPTER

TWO

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 Fixtures

Pytest fixtures for Invenio.

class pytest_invenio.fixtures.**MockDistribution**(*extra_entry_points*)

A mocked distribution that we can inject entry points with.

Initialise the extra entry point.

class pytest_invenio.fixtures.**MockImportlibDistribution**(*extra_entry_points*)

A mocked distribution where we can inject entry points.

Entry points for the distribution.

property entry_points

Iterate over entry points.

property name

Return the 'Name' metadata for the distribution package.

pytest_invenio.fixtures.**UserFixture**()

Fixture to help create user fixtures.

Scope: session

```
@pytest.fixture()
def myuser(UserFixture, app, db):
    u = UserFixture(
        email="myuser@inveniosoftware.org",
        password="auser",
    )
    u.create(app, db)
    return u

def test_with_user(service, myuser):
    service.dosomething(myuser.identity)
```

pytest_invenio.fixtures.**app**(*base_app*, *search*, *database*)

Invenio application with database and search.

Scope: module

See also *base_app* for an Invenio application fixture that does not initialize database and search.

pytest_invenio.fixtures.**app_config**(*db_uri*, *broker_uri*, *celery_config_ext*)

Application configuration fixture.

Scope: module

This fixture sets default configuration for an Invenio application to make it suitable for testing. The database and broker URL are injected into the config, CSRF-protection in forms disabled, HTTP secure headers is disabled, mail sending is output to console.

The fixture can easily be customized in your `conftest.py` or specific test module:

```
# conftest.py
import pytest

pytest.fixture(scope='module')
def app_config(app_config):
    app_config['MYVAR'] = 'test'
    return app_config
```

pytest_invenio.fixtures.**appctx**(*base_app*)

Application context for the current base application.

Scope: module

This fixture pushes an application context on the stack, so that `current_app` is defined and e.g `url_for` will also work.

pytest_invenio.fixtures.**base_app**(*create_app*, *app_config*, *request*, *default_handler*)

Base application fixture (without database, search and cache).

Scope: module.

This fixture is responsible for creating the Invenio application. It depends on an application factory fixture that must be defined by the user.

```
# confest.py
import pytest

@pytest.fixture(scope='module)
def create_app():
    from invenio_app.factory import create_api
    return create_api
```

It is possible to overide the application factory for a specific test module, either by defining a fixture like above example, or simply setting the `create_app` property on the module:

```
# test_something.py

from invenio_app.factory import create_api
create_app = create_api
```

(continues on next page)

(continued from previous page)

```
def test_acase(base_app):
    # ...
```

pytest_invenio.fixtures.**base_client**(*base_app*)

Test client for the base application fixture.

Scope: function

If you need the database and search indexes initialized, simply use the Pytest-Flask fixture `client` instead. This fixture is mainly useful if you need a test client without needing to initialize both the database and search indexes.

pytest_invenio.fixtures.**broker_uri**()

Broker URI (defaults to an RabbitMQ on localhost).

Scope: module

The broker can be overwritten by setting the `BROKER_URL` environment variable.

pytest_invenio.fixtures.**browser**(*request*)

Selenium webdriver fixture.

Scope: session

The fixture initializes a Selenium webdriver which can be used for end-to-end testing of your application:

```
from flask import url_for

def test_browser(live_server, browser):
    browser.get(url_for('index', _external=True))
```

The `live_server` fixture is provided by Pytest-Flask and uses the *app* fixture to determine which application to start.

Note: End-to-end test are only executed if the environment variable `E2E` is set to yes:

```
$ export E2E=yes
```

This allows you to easily switch on/off end-to-end tests.

By default, a Chrome webdriver client will be created. However, you can customize which browsers to test via the `E2E_WEBDRIVER_BROWSERS` environment variable:

```
$ export E2E_WEBDRIVER_BROWSERS="Chrome Firefox"
```

If multiple browsers are requested, each test case using the *browser* fixture will be parameterized with the list of browsers.

In case the test fail, a screenshot will be taken and saved in folder `.e2e_screenshots`.

pytest_invenio.fixtures.**bucket_from_dir**(*db*, *location*)

Creates a bucket from the specified directory.

Scope: function

Use this fixture if your test requires a files bucket. The `bucket_from_dir` fixture returns a function with the following signature:

```
def create_bucket_from_dir(source_dir, location_obj=None):
    """Create bucket from the specified source directory.

    :param source_dir: The directory to create the bucket from.
    :param location_obj: Optional location object to use. If None
        is specified, get the current default location.
    :returns: The new bucket object.
    """
```

Below is an example of how to use the bucket_from_dir fixture:

```
def test_with_bucket(bucket_from_dir):
    bucket = bucket_from_dir('/my/directory/path')
    # ... use the bucket for your test
```

pytest_invenio.fixtures.**celery_config**()

Empty celery config.

pytest_invenio.fixtures.**celery_config_ext**(*celery_config*)

Celery configuration (defaults to eager tasks).

Scope: module

This fixture provides the default Celery configuration (eager tasks, in-memory result backend and exception propagation). It can easily be overwritten in a specific test module:

```
# test_something.py
import pytest

pytest.fixture(scope='module')
def celery_config_ext(celery_config_ext):
    celery_config_ext['CELERY_TASK_ALWAYS_EAGER'] = False
    return celery_config_ext
```

pytest_invenio.fixtures.**cli_runner**(*base_app*)

Create a CLI runner for testing a CLI command.

Scope: module

```
def test_cmd(cli_runner):
    result = cli_runner(mycmd)
    assert result.exit_code == 0
```

pytest_invenio.fixtures.**database**(*appctx*)

Setup database.

Scope: module

Normally, tests should use the function-scoped *db* fixture instead. This fixture takes care of creating the database/tables and removing the tables once tests are done.

pytest_invenio.fixtures.**db**(*database*)

Creates a new database session for a test.

Scope: function

You must use this fixture if your test connects to the database. The fixture will set a save point and rollback all changes performed during the test (this is much faster than recreating the entire database).

pytest_invenio.fixtures.**db_uri**(*instance_path*)

Database URI (defaults to an SQLite datbase in the instance path).

Scope: module

The database can be overwritten by setting the `SQLALCHEMY_DATABASE_URI` environment variable to a SQLAlchemy database URI.

pytest_invenio.fixtures.**default_handler**()

Flask default logging handler.

Flask 0.13/1.0 changed logging to not add the default handler in case a handler is already installed. pytest automatically adds a handler to the root logger, causing Flask not to add a handler. This is an issue when testing Click output which uses the logger to output to the console.

pytest_invenio.fixtures.**entry_points**(*extra_entry_points*)

Entry points fixture.

Scope: module

Invenio relies heavily on Python entry points for constructing an application and it can be rather cumbersome to try to register database models, search mappings etc yourself afterwards.

This fixture allows you to inject extra entry points into the application loading, so that you can load e.g. a testing module or test mapping.

To use the fixture simply define the `extra_entry_points()` fixture, and then depend on the `entry_points()` fixture in your `create_app` fixture:

```
@pytest.fixture(scope="module")
def extra_entry_points():
    return {
        'invenio_db.models': [
            'mock_module = mock_module.models',
        ]
    }

@pytest.fixture(scope="module")
def create_app(instance_path, entry_points):
    return _create_api
```

pytest_invenio.fixtures.**es**(*search*)

Alias for search fixture.

pytest_invenio.fixtures.**es_clear**(*search_clear*)

Alias for search_clear fixture.

pytest_invenio.fixtures.**extra_entry_points**()

Extra entry points.

Overwrite this fixture to define extra entry points.

pytest_invenio.fixtures.**instance_path**()

Temporary instance path.

Scope: module

This fixture creates a temporary directory and sets the `INSTANCE_PATH` environment variable to this directory. The directory is automatically removed.

pytest_invenio.fixtures.**location**(*database*)

Creates a simple default location for a test.

Scope: function

Use this fixture if your test requires a files location. The location will be a default location with the name `pytest-location`.

pytest_invenio.fixtures.**mailbox**(*base_app*)

Mailbox fixture.

Scope: function

This fixture provides a mailbox that captures all outgoing emails and thus easily allow you to test mail sending in your app:

```
def test_mailbox(appctx, mailbox):
    appctx.extensions['mail'].send_message(
        sender='no-reply@localhost',
        subject='Testing',
        body='Test',
        recipients=['no-reply@localhost'])
    assert len(mailbox) == 1
```

pytest_invenio.fixtures.**script_info**(*base_app*)

Get ScriptInfo object for testing a CLI command (DEPRECATED).

Scope: module

Use the `cli_runner` runner fixture directly, or use the base_app:

pytest_invenio.fixtures.**search**(*appctx*)

Setup and teardown all registered search indices.

Scope: module

This fixture will create all registered indexes in search and remove once done. Fixtures that perform changes (e.g. index or remove documents), should used the function-scoped *search_clear* fixture to leave the indexes clean for the following tests.

pytest_invenio.fixtures.**search_clear**(*search*)

Clear search indices after test finishes (function scope).

Scope: function

This fixture rollback any changes performed to the indexes during a test, in order to leave search in a clean state for the next test.

2.1.2 Plugin

Pytest plugin for Invenio.

The plugin adds fixtures to help with creation of applications as well as configuring and initializing the database and search engine.

Additional the plugin helps with configuring end-to-end tests with selenium and taking screenshots of failed selenium tests (useful for inspecting why the test failed on CI systems).

pytest_invenio.plugin.**pytest_generate_tests**(*metafunc*)

Skip end-to-end tests unless requested via E2E env variable.

A screenshot is taken in case of test failures. Set the environment variable E2E_OUTPUT to base64 to have the base64 encoded screenshot printed to stdout (useful in e.g. CI systems). Screenshots are saved to an .e2e_screenshots folder.

Overrides pytest's default test collection function to skip tests using the browser fixture, unless the environment variable E2E is set to yes.

Each test using the browser fixture is parameterized with the list of browsers declared by the E2E_WEBDRIVER_BROWSERS environment variable. By default only Chrome is tested.

pytest_invenio.plugin.**pytest_runtest_makereport**(*item*, *call*)

Add hook to track if the test passed or failed.

CHAPTER

THREE

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/pytest-invenio/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

pytest-invenio could always use more documentation, whether as part of the official pytest-invenio 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/pytest-invenio/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 *pytest-invenio* for local development.

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

```
$ git clone git@github.com:your_name_here/pytest-invenio.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 pytest-invenio
$ cd pytest-invenio/
$ 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 3.7, 3.8, and 3.9. Check https://github.com/inveniosoftware/pytest-invenio/actions?query=event%3Apull_request and make sure that the tests pass for all supported Python versions.

3.2 Changes

Version 2.1.4 (released 2023-06-02)

- user fixture: use identity ID as int

Version 2.1.3 (released 2023-04-13)

- yanked, because of an incompatibility with Flask-SQLAlchemy v3.

Version 2.1.2 (released 2023-03-20)

- disable request rate-limiting

Version 2.1.1 (released 2022-10-25)

- pin pytest version

Version 1.4.15 (released 2022-10-04)

- Pin docker-services-cli<0.5.0, which drops Elasticsearch v6.

Version 1.4.14 (yanked)

Version 2.1.0 (released 2022-10-03)

- Adds support for OpenSearch v2

Version 2.0.0 (released 2022-09-23)

- Use invenio-search v2 and replaces Elasticsearch with OpenSearch, including fixture names.
- Deprecate previous fixtures named with *es* prefix.
- Remove upper pin of pytest.

Version 1.4.13 (released 2022-08-09)

- Fix pycodestyle dependency

Version 1.4.12 (released 2022-08-08)

- Fix flask-login dependency

Version 1.4.11 (released 2022-05-05)

- Upper pin Selenium dependency, v4 drops support for Python 3.7.

Version 1.4.10 (released 2022-05-04)

- Fixes an issue with the user id in the UserFixture being None before the db session is flushed.

Version 1.4.9 (released 2022-05-02)

- Mark users as changed and commit through datastore (outside of context manager).

Version 1.4.8 (yanked 2022-05-02 due to UserFixture session close issues)

- Commit users through the datastore in the UserFixture.

Version 1.4.7 (released 2022-04-04)

- Adds support for Flask v2.1

Version 1.4.6 (released 2022-02-29)

- Adds support for Invenio-Accounts 2.0 in the UserFixture.

Version 1.4.5 (released 2022-02-23)

- Fixes an import so that pytest-invenio is now usable without Invenio-Accounts installed.

Version 1.4.4 (released 2022-02-21)

- Adds new UserFixture for easier test user creation.

Version 1.4.3 (released 2022-02-18)

- Adds support for using importlib_metadata to read the patched entry points.

Version 1.4.2 (released 2021-05-11)

- Add APP_THEME and THEME_ICONS in default app config, often needed when testing invenio packages that will render templates.

Version 1.4.1 (released 2020-12-17)

- Remove pytest-celery because it's still an alpha release.

Version 1.4.0 (released 2020-09-16)

- BACKWARD INCOMPATIBLE: Changes to use isort, pycodestyle and pydocstyle via pytest plugins. You need to update *pytest.ini* and remove the `--pep8` from the addopts and instead add `--isort --pydocstyle --pycodestyle`:

  ```
  addopts = --isort --pydocstyle --pycodestyle ...
  ```

  In *./run-tests.sh* script you should also remove calls to pydocstyle and isort as both are now integrated with pytest.

- BACKWARD INCOMPATIBLE: Upgrade dependencies: coverage, pytest-flask, check-manifest, pytest. You need to set the pytest-flask live server fixture scope in your pytest config:

  ```
  [pytest]
  live_server_scope = function
  ```

- Decommission pytest-pep8 (last release in 2014) in favour of pycodestyle.

Version 1.3.4 (released 2020-09-15)

- Add *entrypoints* fixture to allow injecting extra entry points during testing so that you avoid manual registration of e.g. mappings and schemas.

Version 1.3.3 (released 2020-08-27)

- Add *docker-services-cli* as dependency to enable Invenio modules to perform reproducible tests.

Version 1.3.2 (released 2020-05-19)

- Move check-manifest, coverage, isort, pydocstyle, pytest-flask and pytest-pep8 from test to install requirements to provide them as centrally managed dependencies.

Version 1.3.1 (released 2020-05-12)

- Uninstalls numpy in Travis due to incompatibilities with elasticsearch-py.

Version 1.3.0 (released 2020-03-19)

- Removes support for Python 2.7.

Version 1.2.2 (released 2020-05-07)

- Uninstalls numpy in Travis due to incompatibilities with elasticsearch-py.
- Deprecated Python versions lower than 3.6.0. Now supporting 3.6.0.
- Set maximum version of Werkzeug to 1.0.0 due to incompatible imports.
- Set maximum version of Flask to 1.1.0 due to incompatible imports.
- Set maximum version of Pytest-Flask to 1.0.0 due to breaking changes.
- Set minimum version of Invenio-Search to 1.2.3 and maximum to 1.3.0.

Version 1.2.1 (released 2019-11-13)

- Fixes instance path fixture to also set the static folder.

Version 1.2.0 (released 2019-07-31)

- Adds fixture for creating default Location.
- Adds fixture for creating Bucket from directory with files.

Version 1.1.1 (released 2019-05-21)

- Adds pytest-cov as install dependency.

Version 1.1.0 (released 2019-02-15)

- Changes name of fixture from celery_config to celery_config_ext due to unreliable overwriting of celery_config fixture name.

Version 1.0.6 (released 2018-12-03)

- Fixes overwriting of celery_config fixture

Version 1.0.5 (released 2018-10-08)

- Adds default Content Security Policy header to the app configuration.
- Fixes issue with default tests scope.

Version 1.0.4 (released 2018-08-14)

- Bumps pytest minimun version to 3.8.0.

Version 1.0.3 (released 2018-09-05)

- Moves module dependent imports inside the fixture functions in order to decouple dependencies for Invenio apps or modules that might not be using them.

Version 1.0.2 (released 2018-05-25)

Version 1.0.1 (released 2018-04-17)

Version 1.0.0 (released 2018-03-22)

3.3 License

3.4 Contributors

- Alexander Ioannidis
- Chiara Bigarella
- Esteban J. G. Gabancho
- Lars Holm Nielsen

PYTHON MODULE INDEX

p

INDEX