Invenio tutorial

pu branch
This tutorial assumes that you have an Invenio 2.0 demosite running installed on your computer looking like http://invenio-demo-next.cern.ch

If that is not the case, follow these instructions to get started:
Your Invenio demosite should now be installed in a virtualenv\(^1\)

```
$ workon <your_venv_name>  # Activate your venv to get started!
```

```
$ inveniomanage runserver
```

* Restarting with reloader

Now head to that URL in your browser to have a first look at your running site

---


2) The command `inveniomanage` is part of Invenio and can be found in `$ cdvirtualenv bin/$`
Question

- Look at the terminal where your dev server is running, what is the meaning of the output you see?

Click on any link on the site and see the output in your terminal
Let’s personalise this. Go to your virtualenv root folder:

$ cd virtualenv

And open var/invenio.base-instance/invenio.cfg

Here is where you overwrite all the configuration variables available in Invenio.

* Read a bit more about Flask configuration [http://flask.pocoo.org/docs/config/]
Now add to your invenio.cfg the line:

```
CFG_SITE_SUPPORT_EMAIL = "your.username@cern.ch"
```

Stop the dev server with Ctrl+C and run it again

After refreshing the page you should see the new email address on the footer.

**NOTE:** Invenio uses Redis ([http://redis.io/](http://redis.io/)) to cache the pages. If your changes do not show up, try running:

```
$ redis-cli flushdb
```
Take a look at Invenio’s source code:

1) There is a global config file in invenio/base/config.py - find there the variable you overwrote before

2) Go to invenio/modules/messages and check the file config.py

There you can see only configuration related to the messages module (which allows to send messages in between users of Invenio)

! Every module’s configuration should be on its file config.py, when the server is restarted all the configuration variables are made available in current_app.config
To finish with the configuration, let’s do 2 more things:

1) Open your invenio.cfg and add the lines:

DEBUG = True
CFG_DEVEL_TOOLS = [u'debug-toolbar']

Now restart the dev server, and refresh the page.

A new tab FDT should be on the right of the page. Click on it and check the information available.
And finally let’s see how it looks in the Python shell.

$ inveniomanage shell  (starts a Python shell with Invenio app context)

And try:
In [1]: app.config.keys()

NOTE: We recommend to **pip install ipython** if you don’t have ipython already installed
Invenio uses the widely adopted Jinja2 templating language.

Go ahead, and read the manual at http://jinja.pocoo.org/
Open the file `base/templates/footer_base.html`

And add the following snippet:

```html
<h1>Hello World!</h1>
```

in the beginning of the file.

Refresh the page and see the result!
Now let’s see where are the templates in the Invenio source code.

Go to invenio/modules and open, for example, messages/templates

Have a look at the available templates and check the code inside.
To read some best practices on templates, let’s first build the Invenio documentation (work in progress in pu branch)

```
$ inveniomanage documentation build
```

And then go to http://localhost:4000/documentation/developers/templates/

NOTE: if building fails, the documentation is in the source code `invenio/docs/developers/*.rst`
Adding a record

Let’s add a dummy new record to our installation.

We will use the MarcXML format, which is the one accepted by bibupload command line tool.

In your favourite editor, create a new file with the content in the next slide
Adding a record (II)

For reference:

Field 035 is an identifier for the record
Field 100 is the main author
Field 245 is the title
Field 520 is the abstract
Field 980 is the collection the record belongs
Adding a record (III)

Save the file, for example, as test.xml and then run:

```
$ bibupload -i test.xml  (-i means insert, for other options, see bibupload -h)
```

2014-02-26 17:02:19 --> Task #11 submitted.

Legacy Invenio uses a task manager called BibSched where all the jobs are scheduled (in the new Invenio: **Celery** will take over this role).

```
$ bibsched
```
Adding a record (IV)

You should see your upload job waiting.

Before you run it, check your demosite front page and you will see that there are 32 articles. We are adding a new one (due to 980 ARTICLE in our xml file)

To run it, press the \texttt{r} key

Refresh the front page, did the article count go up?

NOTE: The bottom of BibSched shows some possible shortcuts
Adding a record (V)

The reason why it is not there is due to another command line tool: webcoll.

This tool allocates new records into the appropriate collection so they are not visible until it runs.

Schedule it with:

$ webcoll -s 5m  # -s 5m means to run it periodically every 5 minutes

Open bibsched and run the task. After a page refresh the Article count should have increased.
Adding a record (VI)

Click on the articles collection and then on the search button.

If you don’t specify a query, Invenio returns all results.

Your record should be the first result.
Searching

Let’s see if we can find the record we just added.

From the main page do the query: Bond

Did you record come up?
Searching (II)

If not, the record is missing being indexed, so the search engine cannot find it.

As you might expect, there is another command line tool for this.

You can run:

$ bibindex

Run the task in bibsched and search for ‘Bond’ again, better?
Record API

Hopefully you found your newly created record!

Click on the title and you will get into the “Detailed view” page of that record which contains some extra information and export options.

Check the URL and you will see there is a number at the end, that is what we call record id, a unique number for the record.

Take note of that number and let’s go to the next slide, where we’ll play with the python interpreter.
You should have already seen how Invenio’s master branch works.

As a reminder, open an interpreter:

$ inveniomanage shell

And import the legacy (same as in master branch) `get_record` function:

In [10]: from invenio.legacy.search_engine import get_record

In [11]: my_record = get_record(142)

In [12]: my_record
Record API (III)

The structure is a Python dictionary where the keys are MARC\(^1\) tags (each one has a standard meaning)

Dealing with the marc tags directly can be quite painful for a programmer, this is one of the reasons why, in pu branch, the default is to work with a different record API that we’ll explore in the next slide.

1) [http://www.loc.gov/marc/bibliographic/](http://www.loc.gov/marc/bibliographic/)
Let’s do the following inside of the Invenio shell:

1. `from invenio.modules.records.api import get_record`
2. `record = get_record(142)`
3. `record["authors"]`
4. `record["title"]`
5. `record["abstract"]`

A bit better?
Let’s see a real life example, if you search for “Bond” you should see your record.

How is the search result displayed so that dynamically information from the record is presented?

Check the file `src/invenio/invenio/modules/formatter/templates/format/record/Default_HTML_brief.tpl`

Which is the Jinja template for the brief format display (used in search results)
When you work with Jinja templates, you will be using this new record API.

To have a better understanding of how the record model is defined (meaning which fields are available in a record) check the file:

```
src/demosite/invenio_demosite/recordext/fields/atlantis.cfg
```

Which defines the model of the Invenio demo site you have installed.

*No need to dig into details, just have a look and try to guess what happens in that file.*
For a (more or less) updated overview of the Invenio modules check:
http://invenio-demo.cern.ch/help/hacking/modules-overview

Also worth reading other material related to Hacking Invenio:
http://invenio-demo.cern.ch/help/hacking/

And for the INSPIRE project specifics go to:
https://twiki.cern.ch/twiki/bin/view/Inspire/WebHome
Congratulations!

You finished this tutorial!