Knowledge v2.0

A general approach to write a 2.0 version of a module

- ✓ From Legacy to Module v2.0
- ✓ From SQL to SQLAlchemy Model.
- ✓ How to write upgrade's recipes.
- ✓ Refactoring API: re-implement and deprecate.
- ✓ Write REST API.
- ✓ Write forms with WTForms.
- ✓ The Admin interface with Flask-Admin.
- ✓ User's web interface with Jinja templates.
- ✓ Write documentations.
- ✓ Test the new code.
- ✓ Tips & Tricks

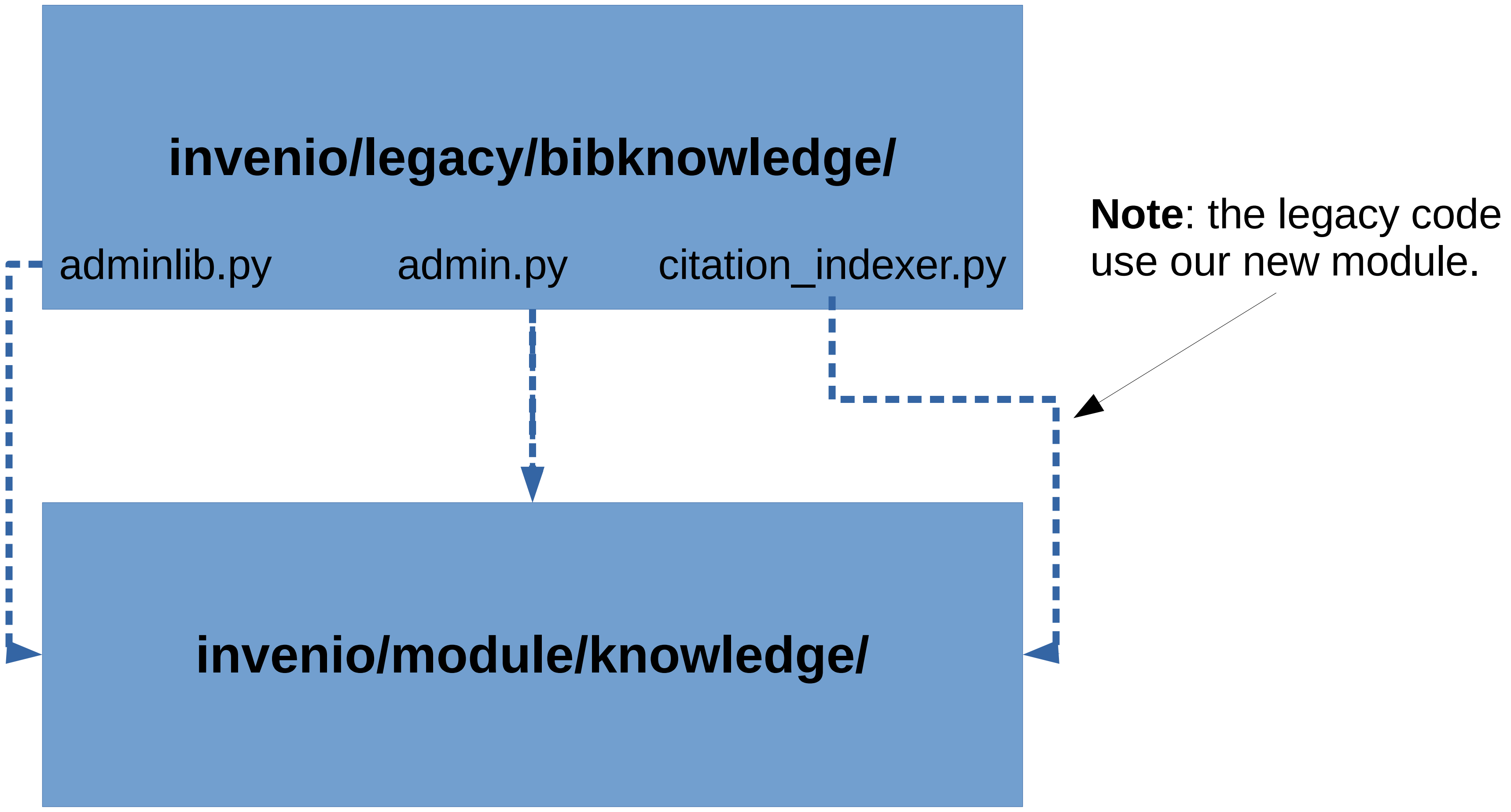
From Legacy to Module v2.0

In the beginning was the legacy code

invenio/legacy/bibknowledge/

From Legacy to Module v2.0

Then, Legacy transition to “module v2.0”


invenio/legacy/bibknowledge/
adminlib.py
admin.py
citation_indexer.py
Note: the legacy code
use our new module.
invenio/module/knowledge/

From Legacy to Module v2.0

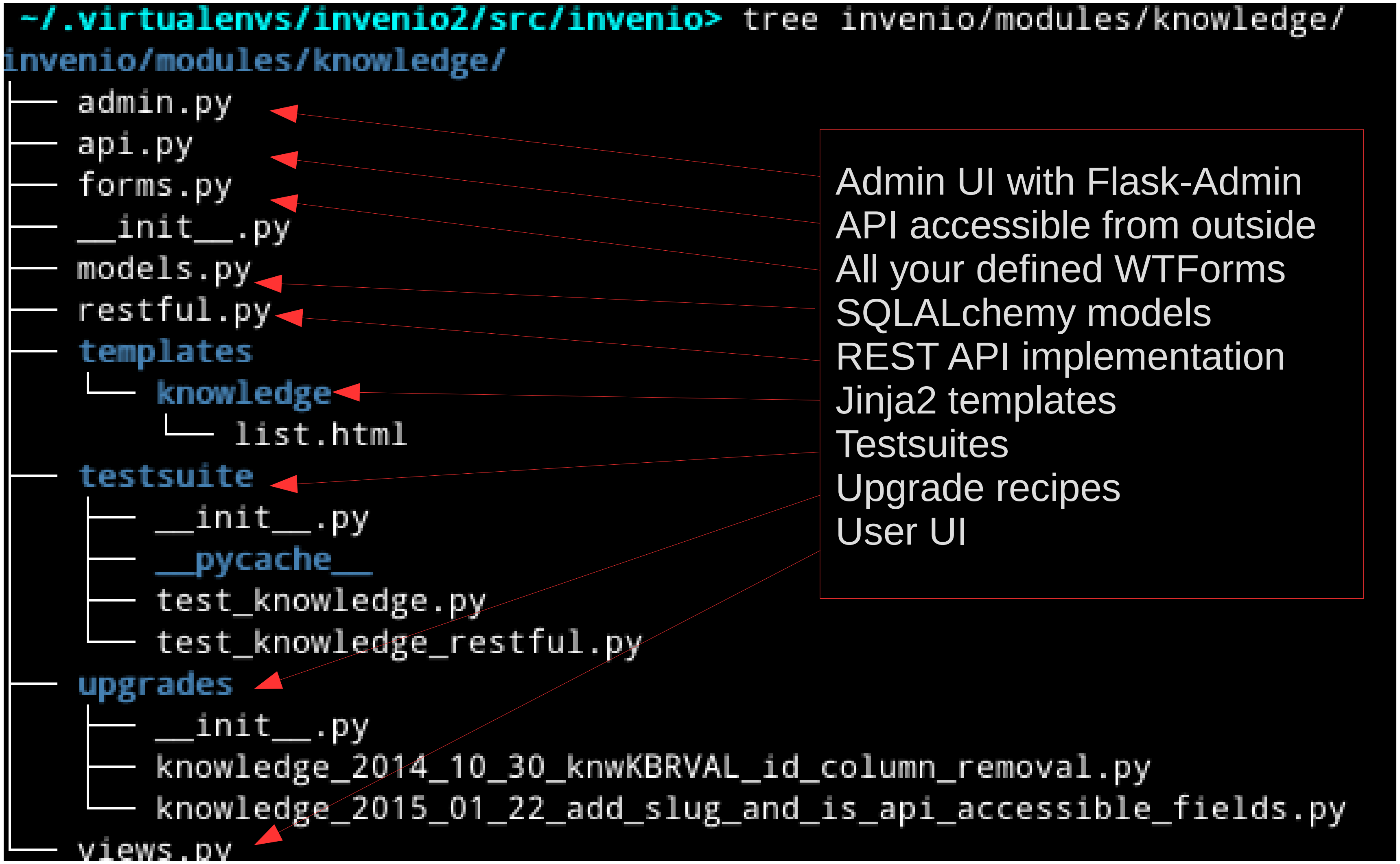
In the end, only “module v2.0” survive

~~invenio/legacy/bibknowledge/~~

invenio/module/knowledge/

From Legacy to Module v2.0

How a module is composed?

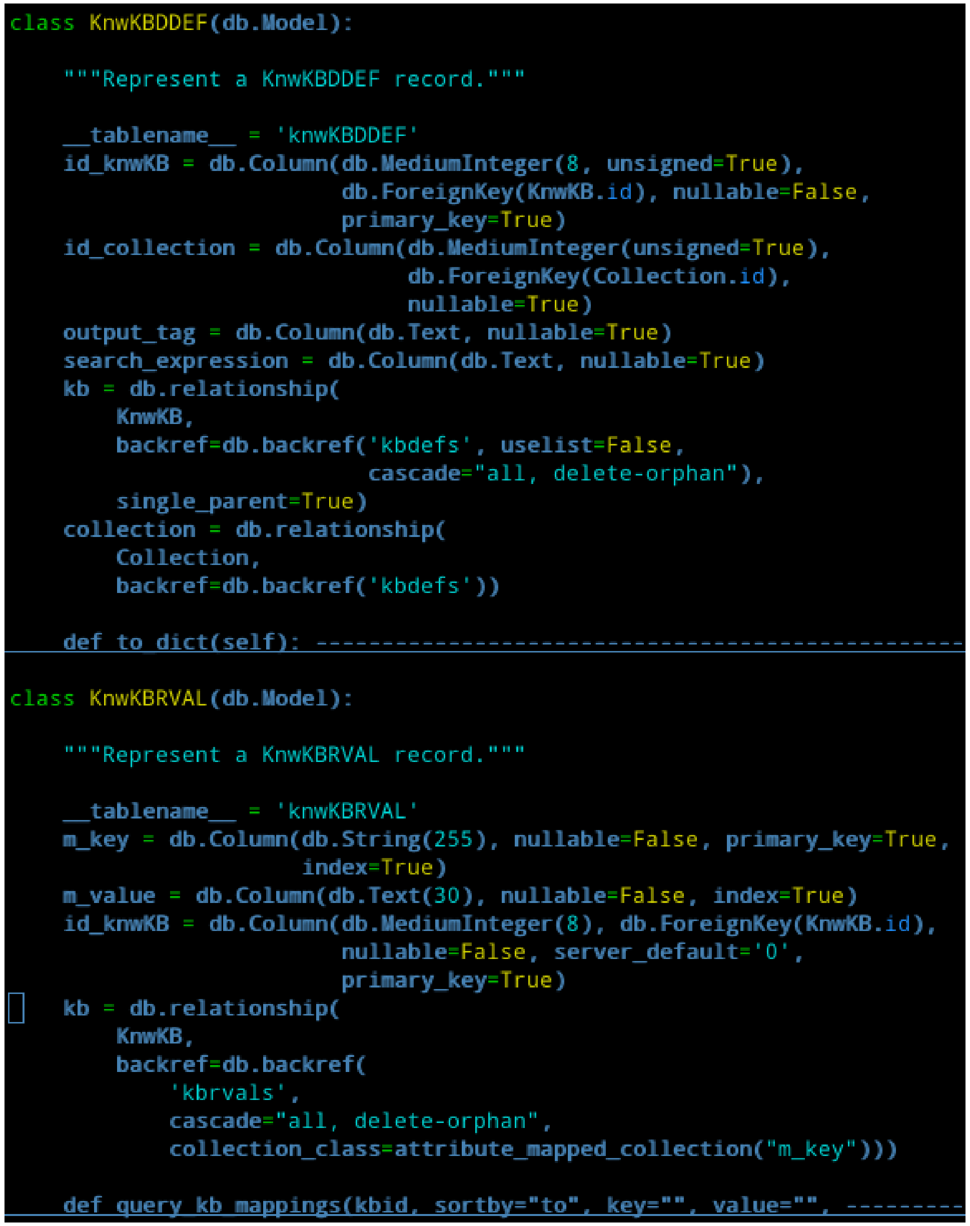


```
~/.virtualenvs/invenio2/src/invenio> tree invenio/modules/knowledge/
invenio/modules/knowledge/
├── admin.py
├── api.py
├── forms.py
├── __init__.py
├── models.py
├── restful.py
├── templates
│   └── knowledge
│       └── list.html
├── testsuite
│   ├── __init__.py
│   ├── __pycache__
│   ├── test_knowledge.py
│   └── test_knowledge_restful.py
├── upgrades
│   ├── __init__.py
│   ├── knowledge_2014_10_30_knwKBRVAL_id_column_removal.py
│   └── knowledge_2015_01_22_add_slug_and_is_api_accessible_fields.py
└── views.py
```

From SQL to SQLAlchemy Model

From
run_sql()
To
models.py

Where in legacy code is used **run_sql()**,
now it'll be replaced with the usage of **SQLAlachemy**
(better if incapsulate the use of the model inside your api)



```
class KnwKBDDEF(db.Model):

    """Represent a KnwKBDDEF record."""

    __tablename__ = 'knwKBDDEF'
    id_knwKB = db.Column(db.MediumInteger(8, unsigned=True),
                         db.ForeignKey(KnwKB.id), nullable=False,
                         primary_key=True)
    id_collection = db.Column(db.MediumInteger(unsigned=True),
                              db.ForeignKey(Collection.id),
                              nullable=True)
    output_tag = db.Column(db.Text, nullable=True)
    search_expression = db.Column(db.Text, nullable=True)
    kb = db.relationship(
        KnwKB,
        backref=db.backref('kbdefs', uselist=False,
                           cascade="all, delete-orphan"),
        single_parent=True)
    collection = db.relationship(
        Collection,
        backref=db.backref('kbdefs'))

    def to_dict(self): ------------------------------------------------

class KnwKBRVAL(db.Model):

    """Represent a KnwKBRVAL record."""

    __tablename__ = 'knwKBRVAL'
    m_key = db.Column(db.String(255), nullable=False, primary_key=True,
                      index=True)
    m_value = db.Column(db.Text(30), nullable=False, index=True)
    id_knwKB = db.Column(db.MediumInteger(8), db.ForeignKey(KnwKB.id),
                         nullable=False, server_default='0',
                         primary_key=True)
    kb = db.relationship(
        KnwKB,
        backref=db.backref(
            'kbrvals',
            cascade="all, delete-orphan",
            collection_class=attribute_mapped_collection("m_key")))

    def query_kb_mappings(kbid, sortby="to", key="", value="", ---------
```

How to write upgrade's recipes (1/2)

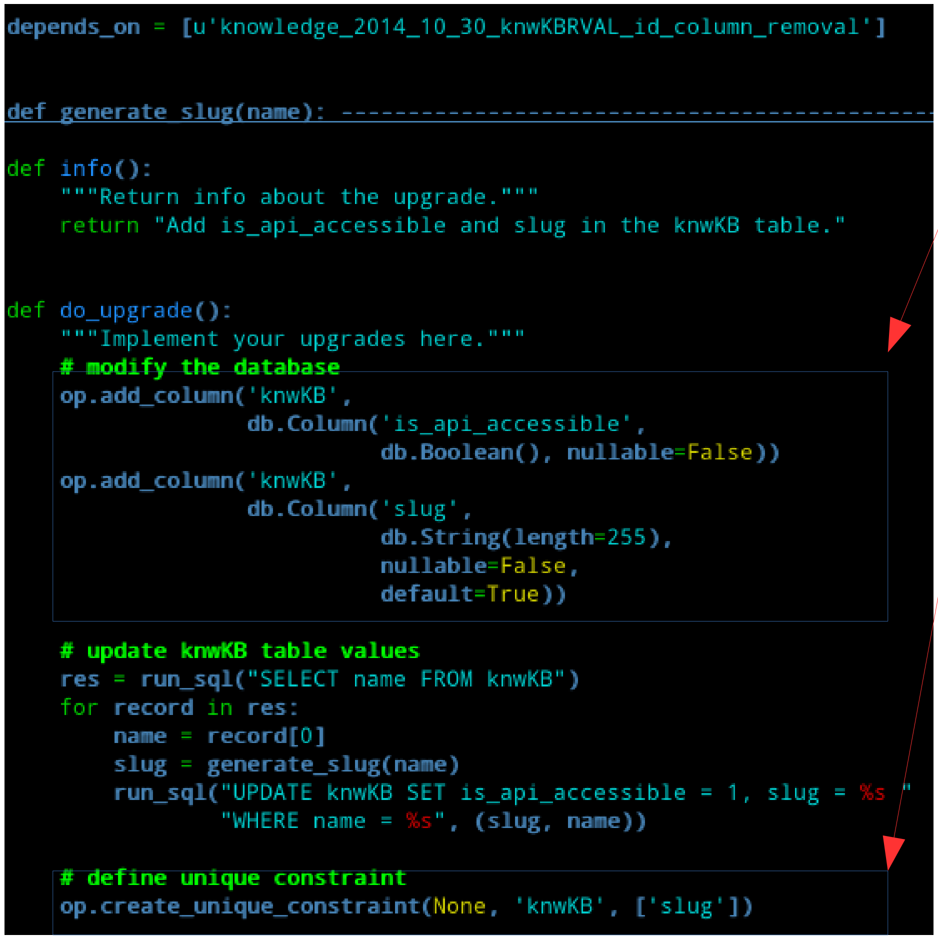
Old data model vs New data model:
how to upgrade your database in production environment without break all?

```
# Use the old Database
$> git checkout pu
$> inveniomanage database recreate –yes-i-know
# With the new Codebase
$> git checkout mybranch
$> inveniomanage upgrader create recipe -a -p invenio.modules.knowledge
# Finish to prepare your recipe
$> vim invenio/modules/knowledge/upgrades/knowledge_2015_02_16_rename_me.py
```

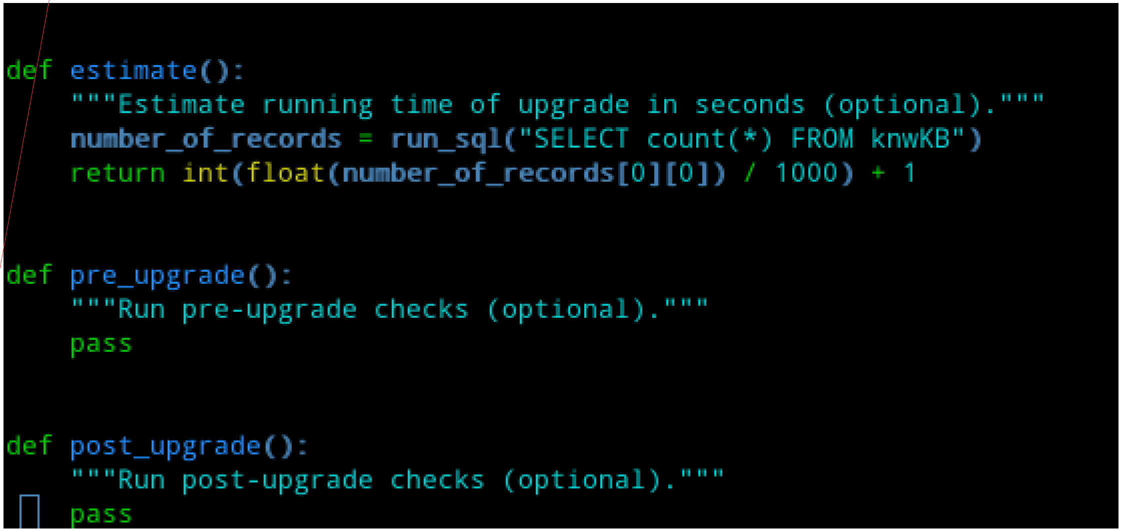
How to write upgrade's recipes (2/2)

Alembic is a database migrations tool written

It help you to write python code to upgrade the database



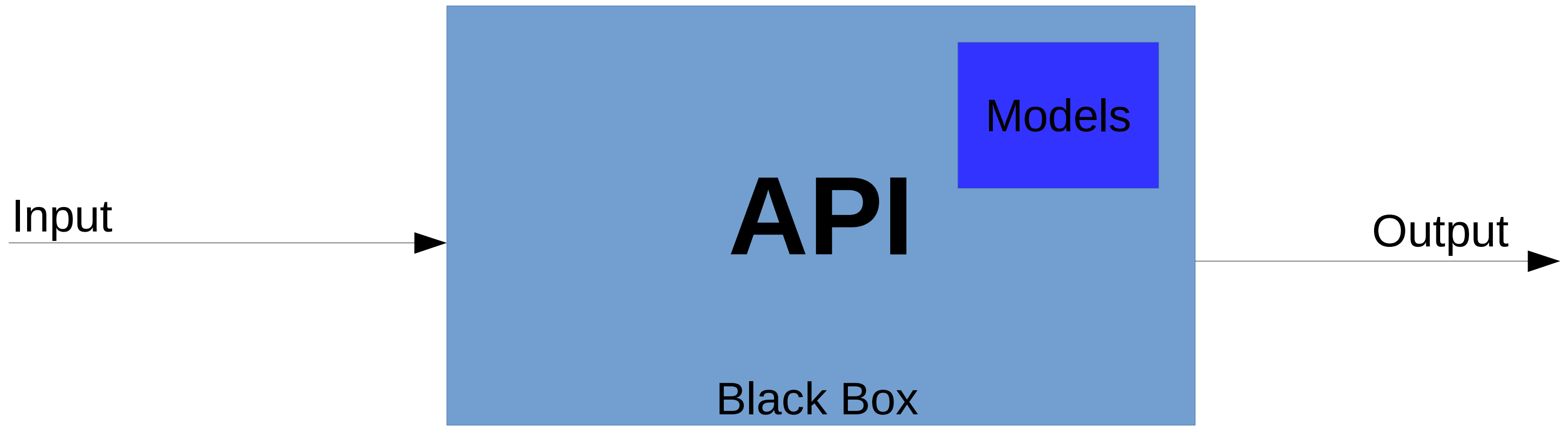
```
depends_on = [u'knowledge_2014_10_30_knwKBRVAL_id_column_removal']


def generate_slug(name): ------------------------------------------

def info():
    """Return info about the upgrade."""
    return "Add is_api_accessible and slug in the knwKB table."


def do_upgrade():
    """Implement your upgrades here."""
    # modify the database
    op.add_column('knwKB',
                  db.Column('is_api_accessible',
                            db.Boolean(), nullable=False))
    op.add_column('knwKB',
                  db.Column('slug',
                            db.String(length=255),
                            nullable=False,
                            default=True))

    # update knwKB table values
    res = run_sql("SELECT name FROM knwKB")
    for record in res:
        name = record[0]
        slug = generate_slug(name)
        run_sql("UPDATE knwKB SET is_api_accessible = 1, slug = %s "
                "WHERE name = %s", (slug, name))

    # define unique constraint
    op.create_unique_constraint(None, 'knwKB', ['slug'])
```



```
def estimate():
    """Estimate running time of upgrade in seconds (optional)."""
    number_of_records = run_sql("SELECT count(*) FROM knwKB")
    return int(float(number_of_records[0][0]) / 1000) + 1


def pre_upgrade():
    """Run pre-upgrade checks (optional)."""
    pass


def post_upgrade():
    """Run post-upgrade checks (optional)."""
    pass
```

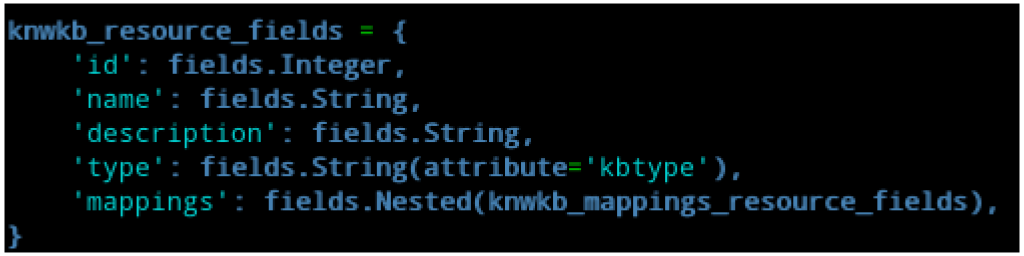
Refactoring API: re-implement and deprecate.

- Each request have to pass from api (no direct access to models).
- Respect the old API (backward compatibility), meanwhile deprecate not useful code and implement a new cool API.



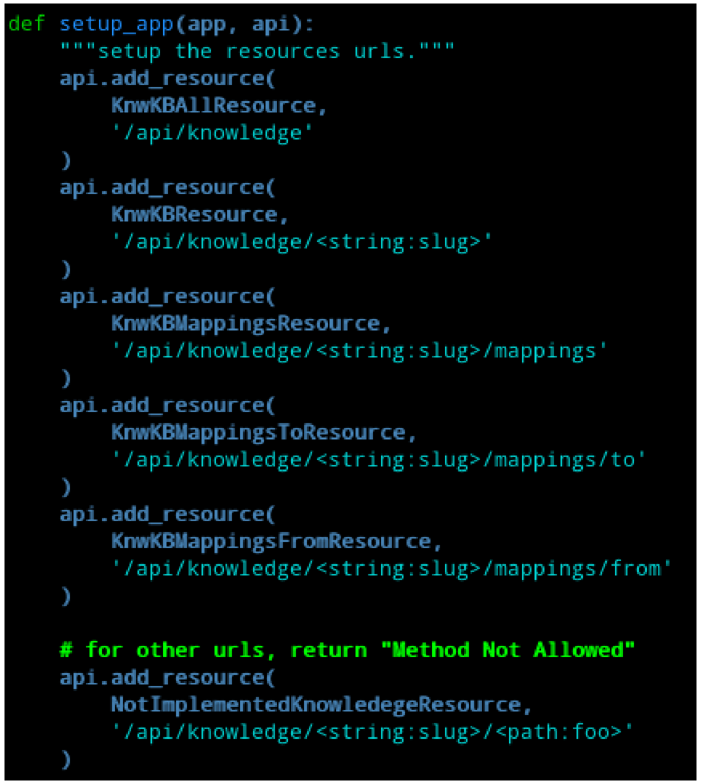
Write REST API

Define the object structure



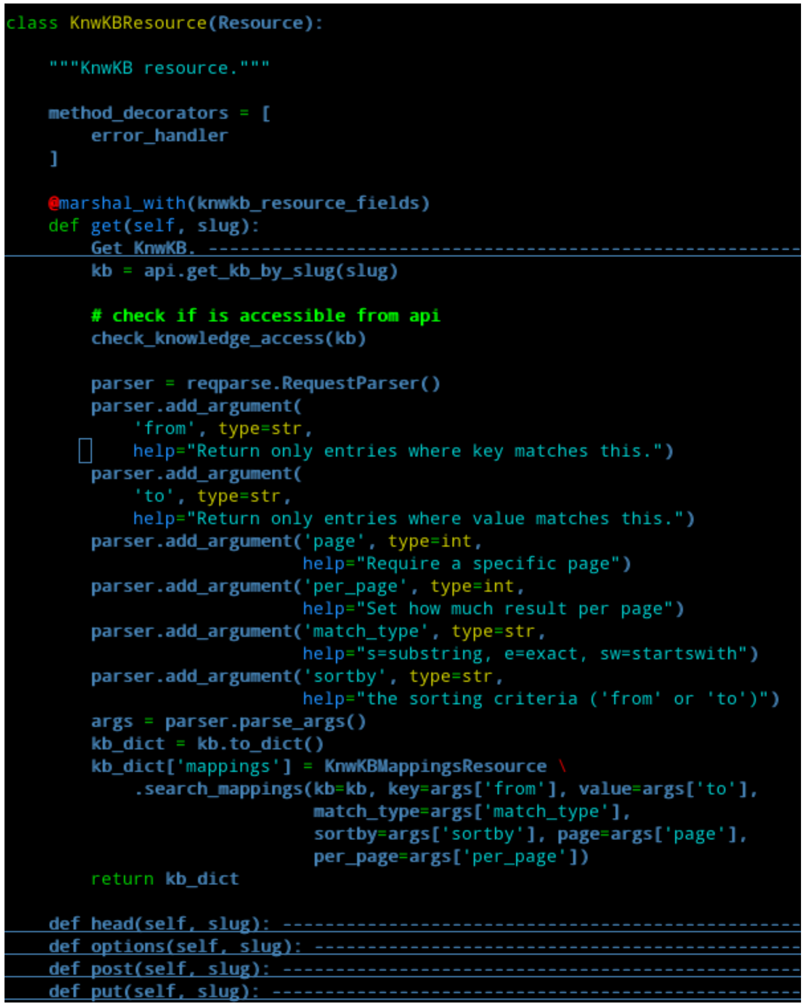
```
knwkb_resource_fields = {
    'id': fields.Integer,
    'name': fields.String,
    'description': fields.String,
    'type': fields.String(attribute='kbtype'),
    'mappings': fields.Nested(knwkb_mappings_resource_fields),
}
```

Define the endpoints



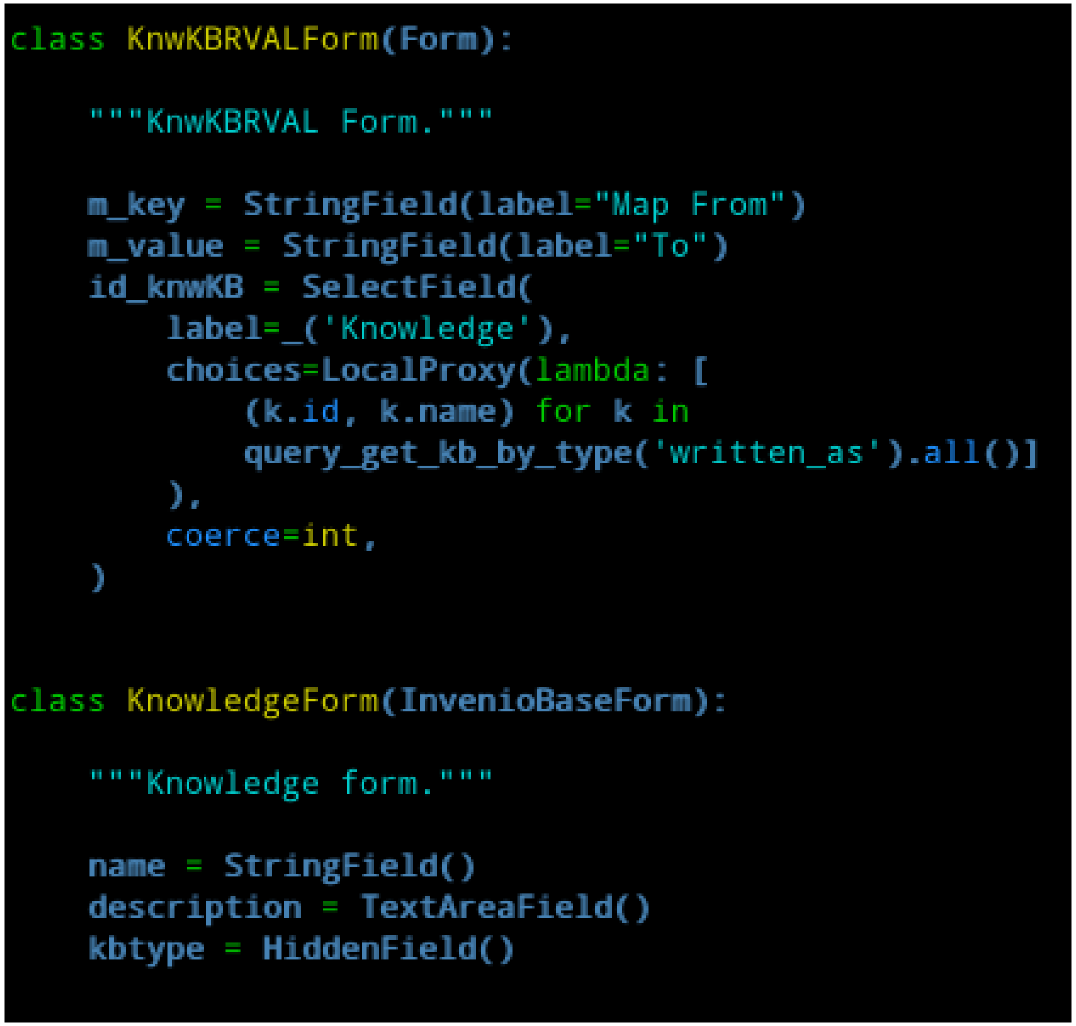
```
def setup_app(app, api):
    """setup the resources urls."""
    api.add_resource(
        KnwKBAllResource,
        '/api/knowledge'
    )
    api.add_resource(
        KnwKBResource,
        '/api/knowledge/<string:slug>'
    )
    api.add_resource(
        KnwKBMappingsResource,
        '/api/knowledge/<string:slug>/mappings'
    )
    api.add_resource(
        KnwKBMappingsToResource,
        '/api/knowledge/<string:slug>/mappings/to'
    )
    api.add_resource(
        KnwKBMappingsFromResource,
        '/api/knowledge/<string:slug>/mappings/from'
    )

    # for other urls, return "Method Not Allowed"
    api.add_resource(
        NotImplementedKnowledegeResource,
        '/api/knowledge/<string:slug>/<path:foo>'
    )
```

Define a resource



```
class KnwKBResource(Resource):

    """KnwKB resource."""

    method_decorators = [
        error_handler
    ]

    @marshal_with(knwkb_resource_fields)
    def get(self, slug):
        Get KnwKB. -------------------------------------------------
        kb = api.get_kb_by_slug(slug)

        # check if is accessible from api
        check_knowledge_access(kb)

        parser = reqparse.RequestParser()
        parser.add_argument(
            'from', type=str,
            help="Return only entries where key matches this.")
        parser.add_argument(
            'to', type=str,
            help="Return only entries where value matches this.")
        parser.add_argument('page', type=int,
                            help="Require a specific page")
        parser.add_argument('per_page', type=int,
                            help="Set how much result per page")
        parser.add_argument('match_type', type=str,
                            help="s=substring, e=exact, sw=startswith")
        parser.add_argument('sortby', type=str,
                            help="the sorting criteria ('from' or 'to')")
        args = parser.parse_args()
        kb_dict = kb.to_dict()
        kb_dict['mappings'] = KnwKBMappingsResource \
            .search_mappings(kb=kb, key=args['from'], value=args['to'],
                             match_type=args['match_type'],
                             sortby=args['sortby'], page=args['page'],
                             per_page=args['per_page'])
        return kb_dict

    def head(self, slug): ------------------------------------------------
    def options(self, slug): ---------------------------------------------
    def post(self, slug): ------------------------------------------------
    def put(self, slug): -------------------------------------------------
```

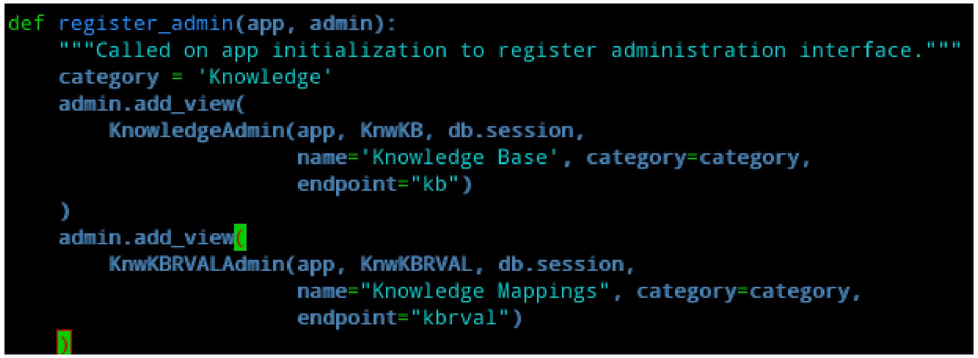
Write forms with WTForms



```
class KnwKBRVALForm(Form):

    """KnwKBRVAL Form."""

    m_key = StringField(label="Map From")
    m_value = StringField(label="To")
    id_knwKB = SelectField(
        label=_('Knowledge'),
        choices=LocalProxy(lambda: [
            (k.id, k.name) for k in
            query_get_kb_by_type('written_as').all()]
        ),
        coerce=int,
    )


class KnowledgeForm(InvenioBaseForm):

    """Knowledge form."""

    name = StringField()
    description = TextAreaField()
    kbtype = HiddenField()
```

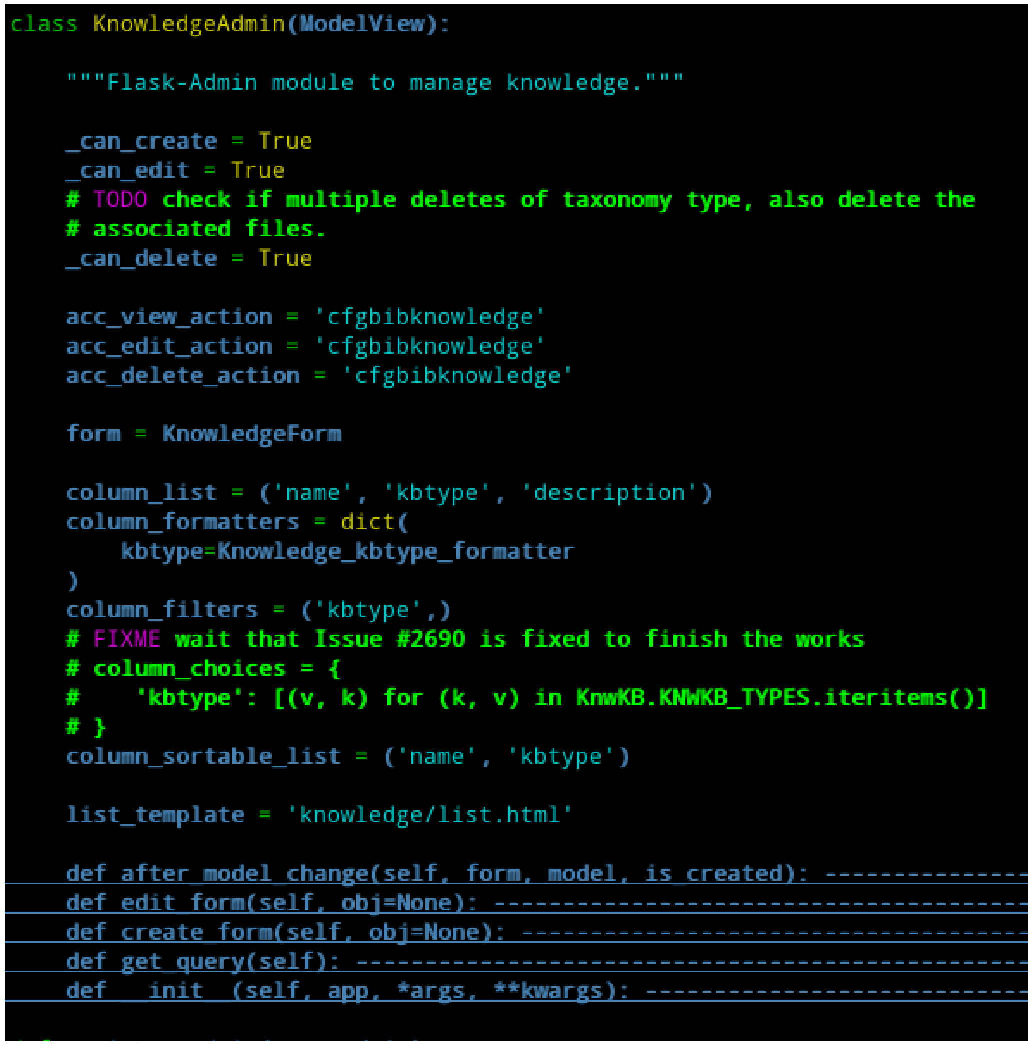
The Admin interface with Flask-Admin

Define endpoints



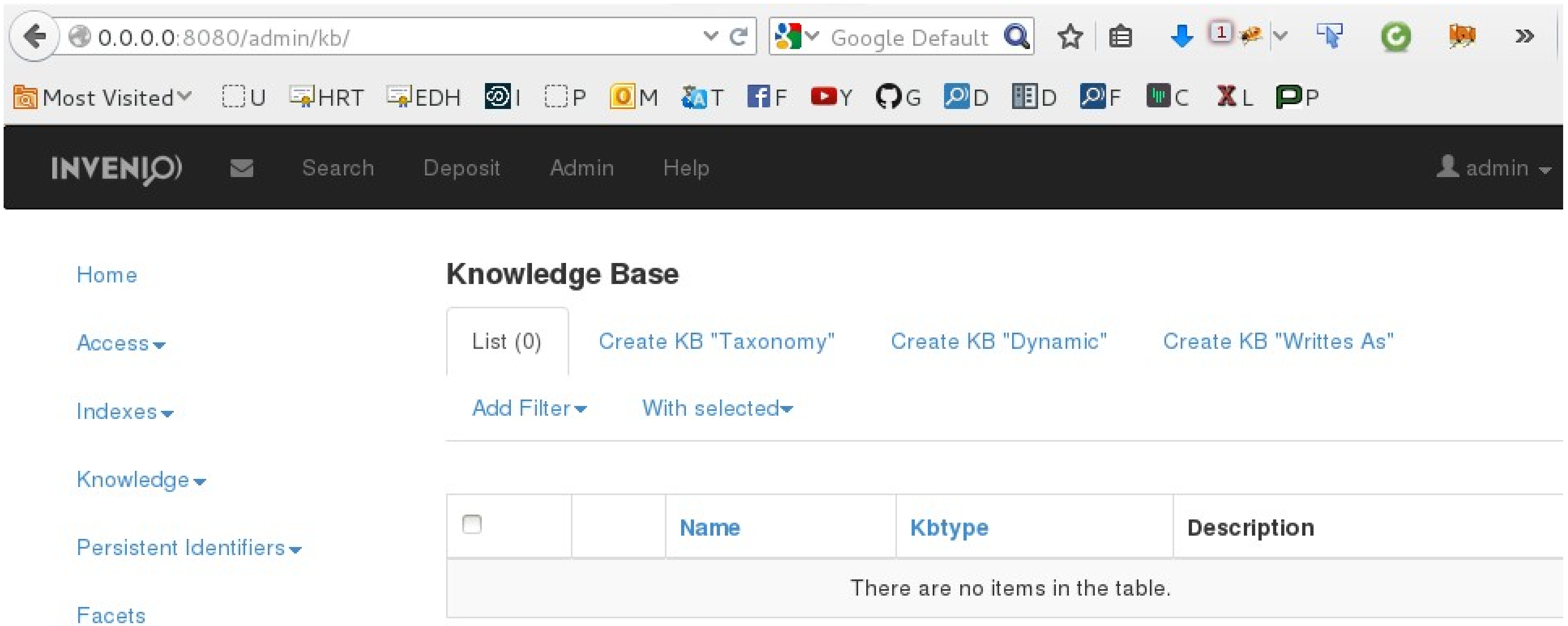
```
def register_admin(app, admin):
    """Called on app initialization to register administration interface."""
    category = 'Knowledge'
    admin.add_view(
        KnowledgeAdmin(app, KnwKB, db.session,
                       name='Knowledge Base', category=category,
                       endpoint="kb")
    )
    admin.add_view(
        KnwKBRVALAdmin(app, KnwKBRVAL, db.session,
                       name="Knowledge Mappings", category=category,
                       endpoint="kbrval")
    )
```

Admin interface for Knowledge



```
class KnowledgeAdmin(ModelView):

    """Flask-Admin module to manage knowledge."""

    _can_create = True
    _can_edit = True
    # TODO check if multiple deletes of taxonomy type, also delete the
    # associated files.
    _can_delete = True

    acc_view_action = 'cfgbibknowledge'
    acc_edit_action = 'cfgbibknowledge'
    acc_delete_action = 'cfgbibknowledge'

    form = KnowledgeForm

    column_list = ('name', 'kbtype', 'description')
    column_formatters = dict(
        kbtype=Knowledge_kbtype_formatter
    )
    column_filters = ('kbtype',)
    # FIXME wait that Issue #2690 is fixed to finish the works
    # column_choices = {
    #     'kbtype': [(v, k) for (k, v) in KnwKB.KNWKB_TYPES.iteritems()]
    # }
    column_sortable_list = ('name', 'kbtype')

    list_template = 'knowledge/list.html'

    def after_model_change(self, form, model, is_created): --------------
    def edit_form(self, obj=None): --------------------------------------
    def create_form(self, obj=None): ------------------------------------
    def get_query(self): ------------------------------------------------
    def __init__(self, app, *args, **kwargs): ---------------------------
```

The Admin interface with Flask-Admin



Knowledge Base

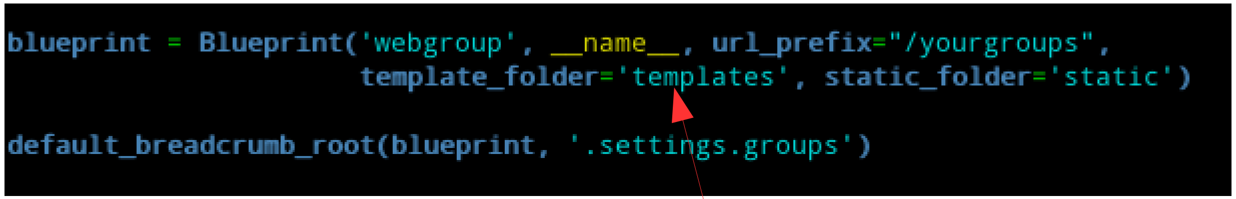
List (0) | Create KB "Taxonomy" | Create KB "Dynamic" | Create KB "Writtes As"

Add Filter | With selected

☐		Name	Kbtype	Description

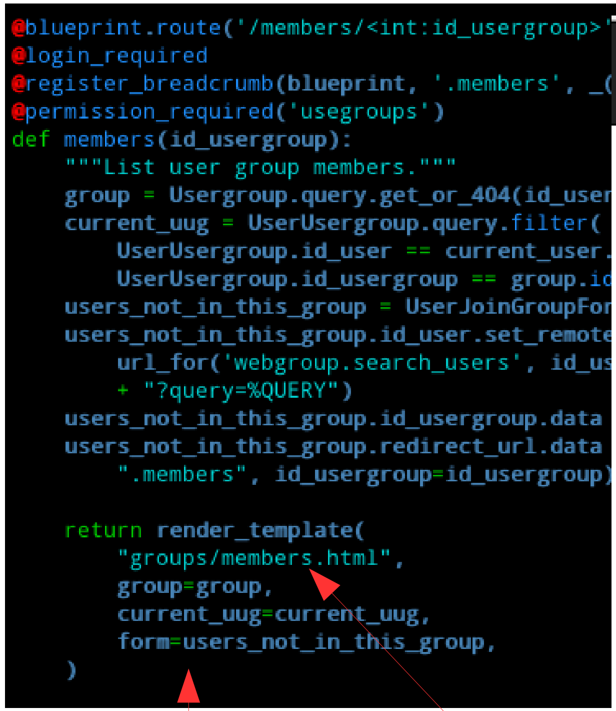
There are no items in the table.

User's web interface with Jinja templates

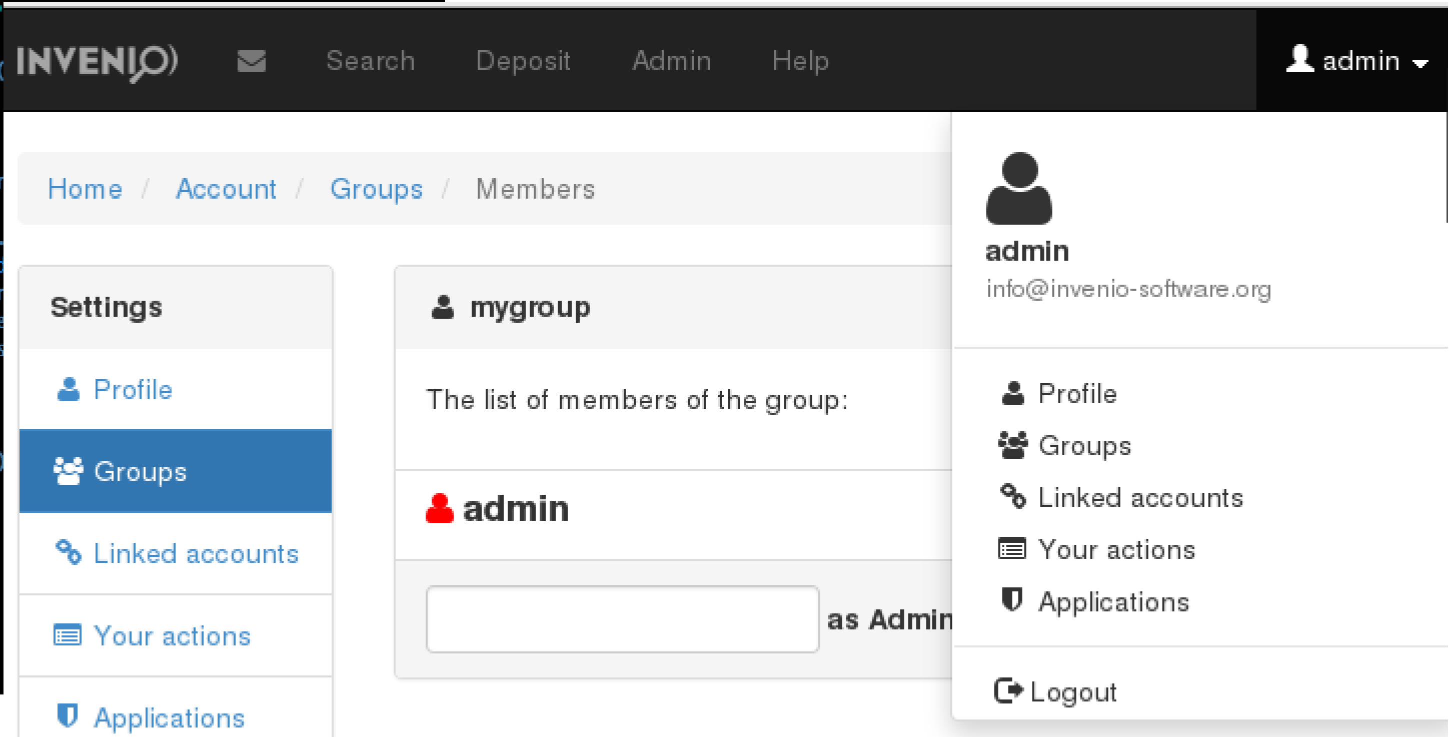


```
blueprint = Blueprint('webgroup', __name__, url_prefix="/yourgroups",
                      template_folder='templates', static_folder='static')

default_breadcrumb_root(blueprint, '.settings.groups')
```

Define where load the templates
(in this case invenio/modules/groups/**templates**)



```
@blueprint.route('/members/<int:id_usergroup>'
@login_required
@register_breadcrumb(blueprint, '.members', _(
@permission_required('usegroups')
def members(id_usergroup):
    """List user group members."""
    group = Usergroup.query.get_or_404(id_user
    current_uug = UserUsergroup.query.filter(
        UserUsergroup.id_user == current_user.
        UserUsergroup.id_usergroup == group.id
    users_not_in_this_group = UserJoinGroupFor
    users_not_in_this_group.id_user.set_remote
        url_for('webgroup.search_users', id_us
        + "?query=%QUERY")
    users_not_in_this_group.id_usergroup.data
    users_not_in_this_group.redirect_url.data
        ".members", id_usergroup=id_usergroup)

    return render_template(
        "groups/members.html",
        group=group,
        current_uug=current_uug,
        form=users_not_in_this_group,
    )
```



The list of members of the group:

admin

as Admin

Your template

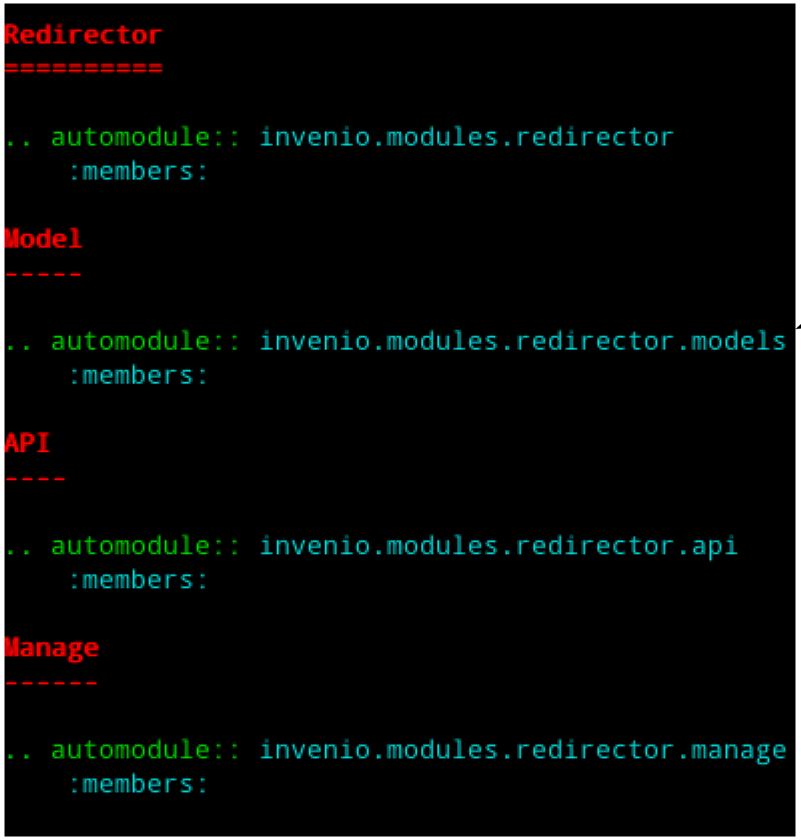
The objects passed to the template

Write documentations

E.g. docs/modules/redirector.rst

Generate documentation

$> sphinx-build -qnNW docs docs/_build/html



```
Redirector
==========

.. automodule:: invenio.modules.redirector
    :members:

Model
-----

.. automodule:: invenio.modules.redirector.models
    :members:

API
---

.. automodule:: invenio.modules.redirector.api
    :members:

Manage
------

.. automodule:: invenio.modules.redirector.manage
    :members:
```

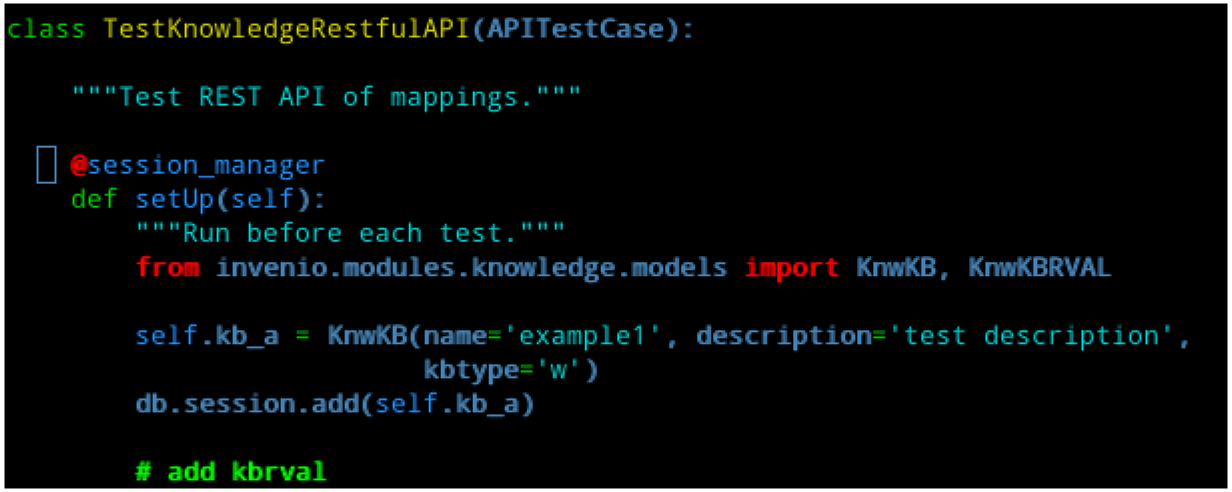
E.g: Define where find the documentation for the section "Model"

Test the new code

Load db interface

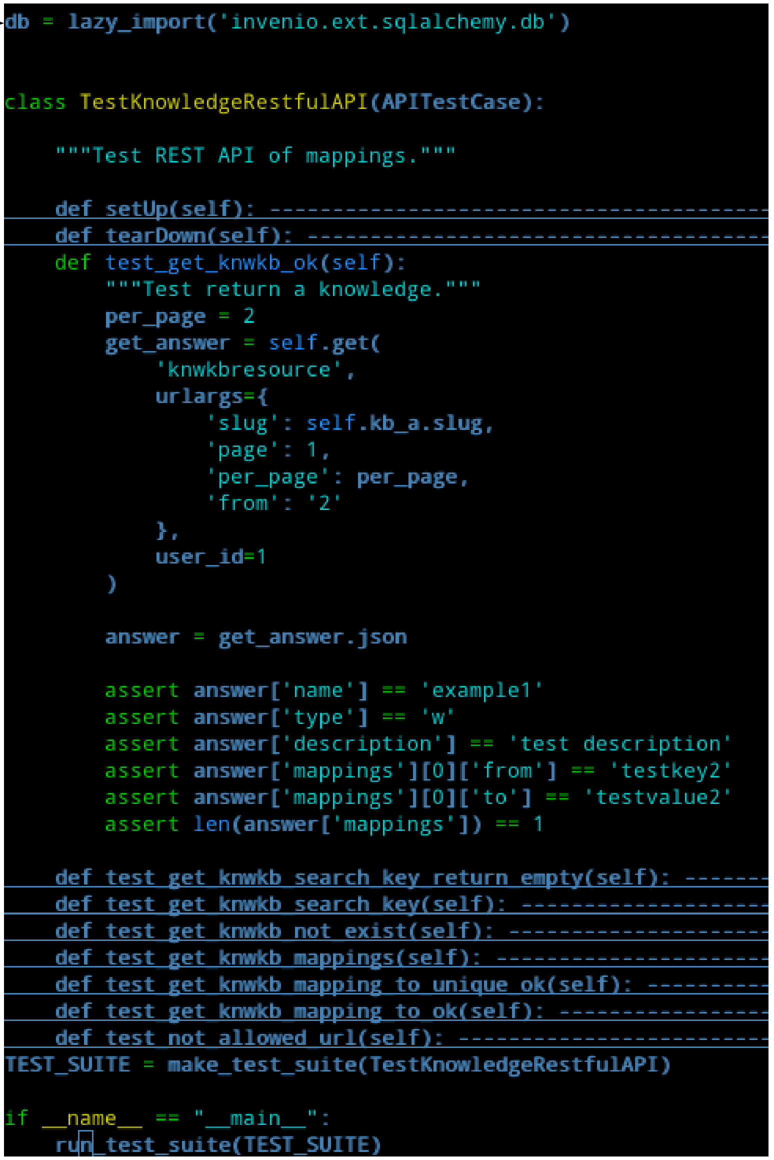
- Start your test:

  $> cd invenio/modules/knowledge/testsuite
  $> py.test test_knowledge_restful.py



```
class TestKnowledgeRestfulAPI(APITestCase):

    """Test REST API of mappings."""

    @session_manager
    def setUp(self):
        """Run before each test."""
        from invenio.modules.knowledge.models import KnwKB, KnwKBRVAL

        self.kb_a = KnwKB(name='example1', description='test description',
                          kbtype='w')
        db.session.add(self.kb_a)

        # add kbrval
```

Note: import only when you need it (inside the function where you use it).



```
db = lazy_import('invenio.ext.sqlalchemy.db')

class TestKnowledgeRestfulAPI(APITestCase):

    """Test REST API of mappings."""

    def setUp(self): ----------------------------------------
    def tearDown(self): -------------------------------------
    def test_get_knwkb_ok(self):
        """Test return a knowledge."""
        per_page = 2
        get_answer = self.get(
            'knwkbresource',
            urlargs={
                'slug': self.kb_a.slug,
                'page': 1,
                'per_page': per_page,
                'from': '2'
            },
            user_id=1
        )

        answer = get_answer.json

        assert answer['name'] == 'example1'
        assert answer['type'] == 'w'
        assert answer['description'] == 'test description'
        assert answer['mappings'][0]['from'] == 'testkey2'
        assert answer['mappings'][0]['to'] == 'testvalue2'
        assert len(answer['mappings']) == 1

    def test_get_knwkb_search_key_return_empty(self): -------
    def test_get_knwkb_search_key(self): --------------------
    def test_get_knwkb_not_exist(self): ---------------------
    def test_get_knwkb_mappings(self): ----------------------
    def test_get_knwkb_mapping_to_unique_ok(self): ----------
    def test_get_knwkb_mapping_to_ok(self): -----------------
    def test_not_allowed_url(self): -------------------------
TEST_SUITE = make_test_suite(TestKnowledgeRestfulAPI)

if __name__ == "__main__":
    run_test_suite(TEST_SUITE)
```

Tips & Tricks

- **Use devscrips to install invenio2:**
  https://github.com/tiborsimko/invenio-devscripts
- **Database create/drop**:
  inveniomanage database create
  inveniomanage database drop –yes-i-know
  inveniomanage database recreate –yes-i-know
- **Start server**:
  inveniomanage runserver
- **Install all dependency**:
  pip install -e . --process-dependency-links
- **Install Kwalitee**:
  pip install kwalitee
  kwalitee githooks install
- **Debugging configuration**:
  SQLALCHEMY_ECHO=True
  ASSETS_DEBUG = True
  DEBUG = True
  TESTING = True
- **Configuration file**:
  cdvirtualenv var/invenio.base-instance/
  vim invenio.cfg
- **Jasmine tests**:
  http://0.0.0.0:8080/jasmine/specrunner
- **Python tests**:
  python setup.py test