From Invenio 1 to Invenio 2

Case Study
of
BibKnowledge

Created by Leonardo Rossi (@hachreak)

From Invenio 1 to Invenio 2

Summary

- ✓ Introduction
- ✓ Use the “*Divide et impera*” concept

  how to split the module in all its aspects:
- ✓ From SQL to Model
- ✓ Upgrade's recipes
- ✓ Refactoring API
- ✓ REST API
- ✓ Forms
- ✓ Admin interface
- ✓ User's web interface
- ✓ Documentation
- ✓ Test the new code

Introduction

Invenio

access	authorlist	circulation	converter	access	deposit
encoder	messages	pages	redirector	statistics	scheduler
access	authorlist	circulation	converter	access	knowledge
upgrader	accounts	author profiles	classifier	dashboard	webhooks
exporter	jsonalchemy	multimedia	pdfchecker	refextract	pidstore
submit	uploader	alerts	authors	cloud connector	baskets

● ● ●

Introduction

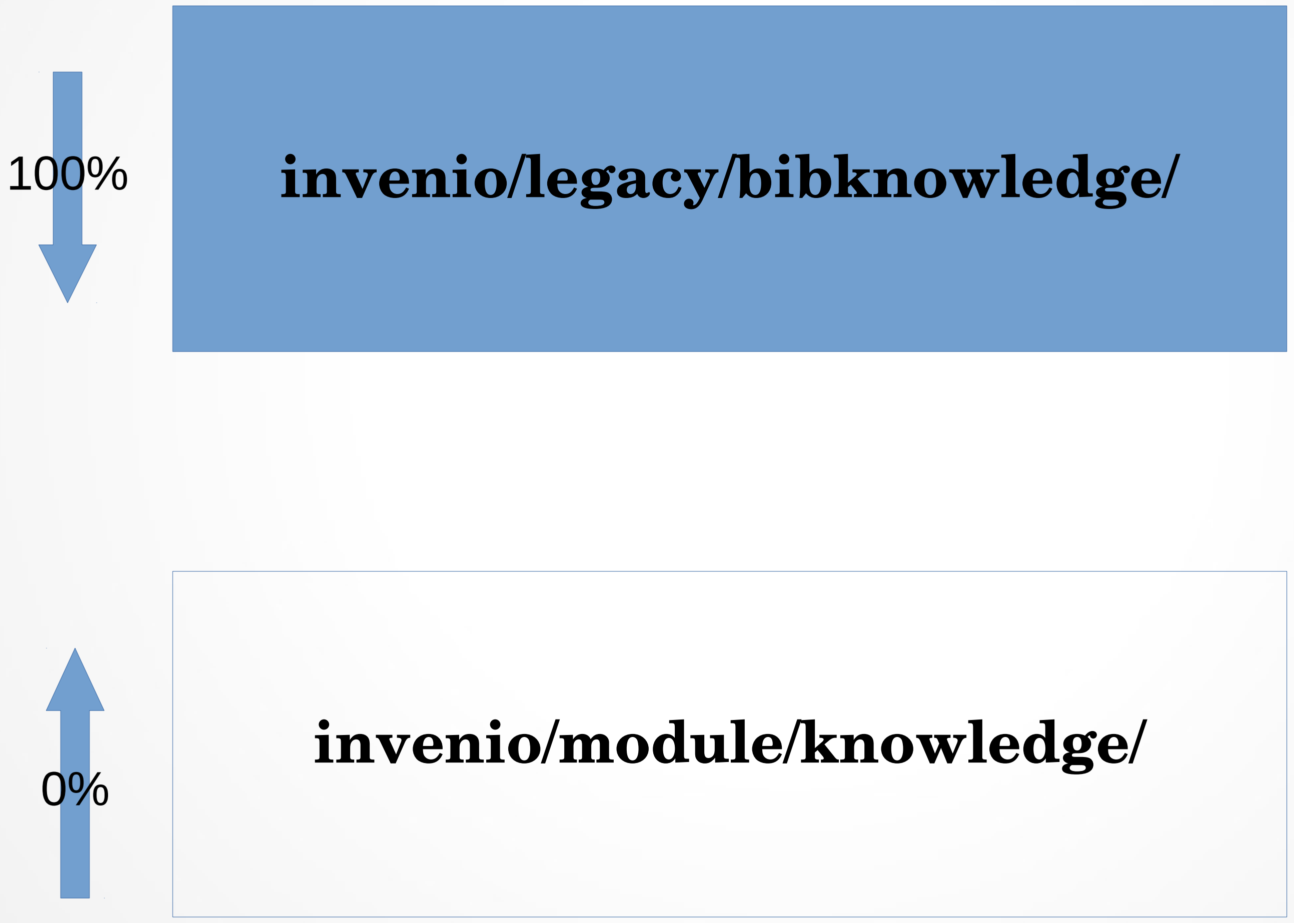
Invenio

access	authorlist	circulation	converter	access	deposit
encoder	messages	pages	redirector	statistics	scheduler
access	authorlist	circulation	converter	access	knowledge
upgrader	accounts	author profiles	classifier	dashboard	webhooks
exporter	jsonalchemy	multimedia	pdfchecker	refextract	pidstore
submit	uploader	alerts	authors	cloud connector	baskets

● ● ●

From Legacy to Module v2.0

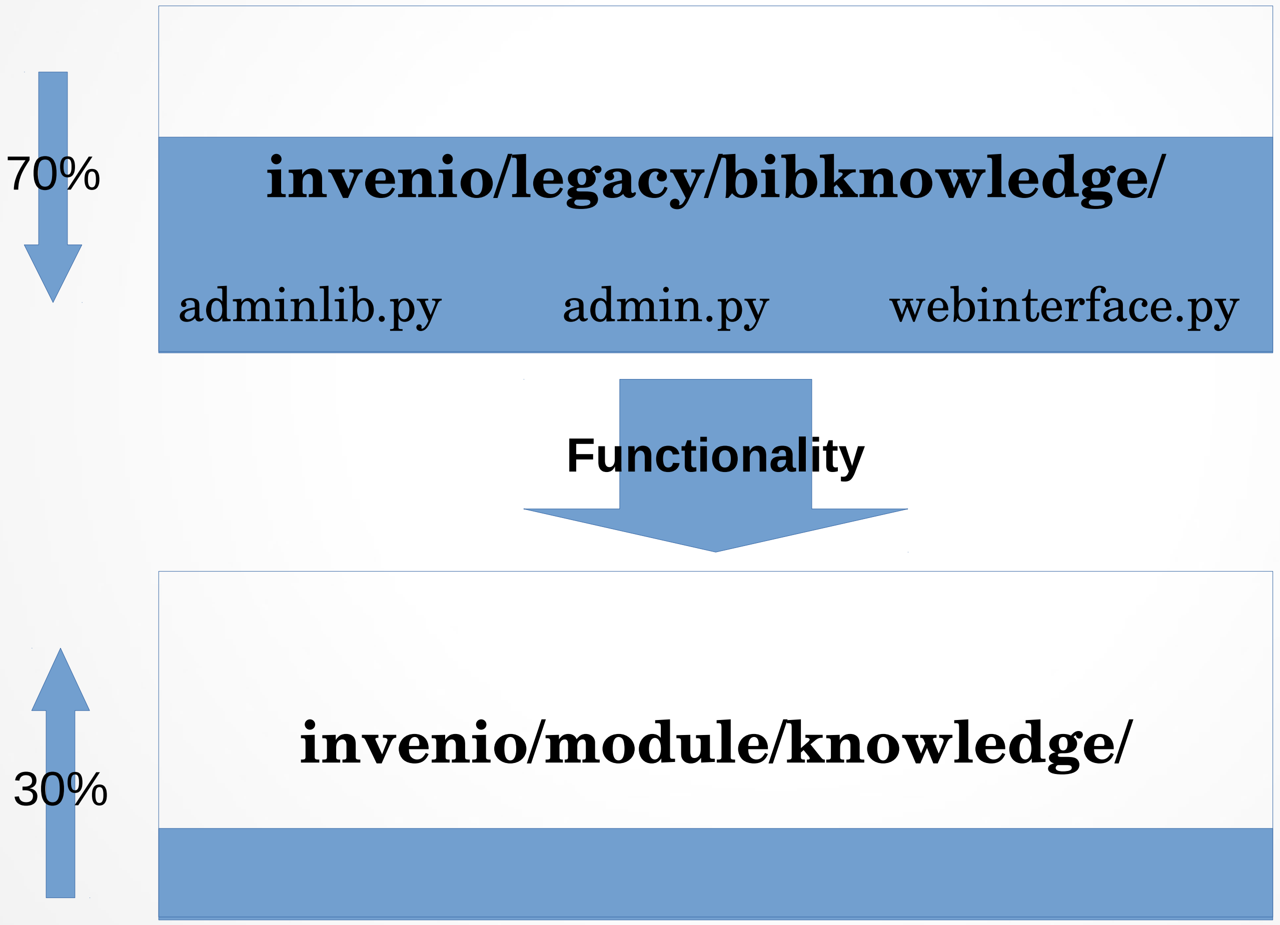
"In the beginning was the legacy code"



Conceptually the process is like "*communicating vessels*"....

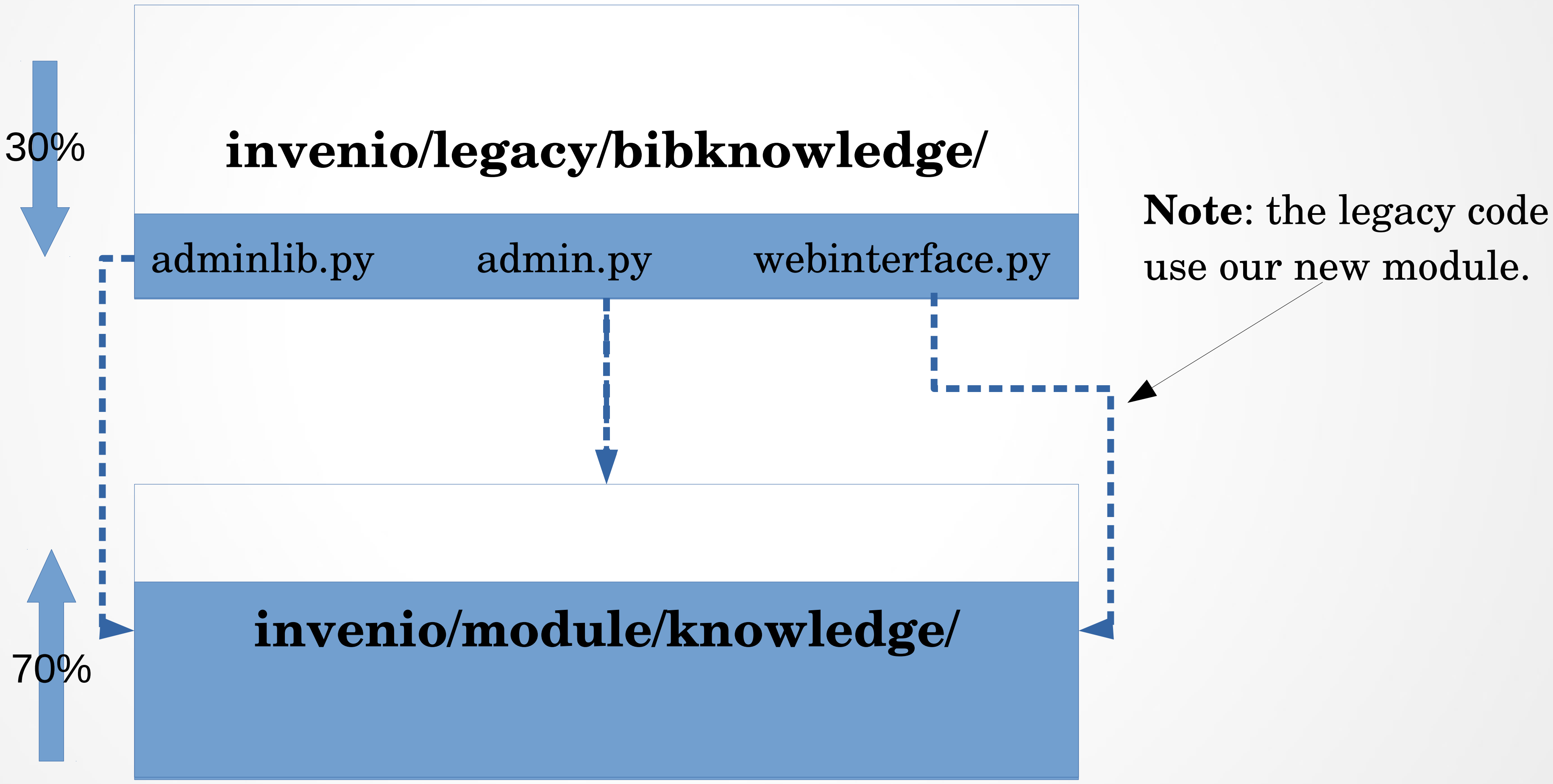
From Legacy to Module v2.0

Then, Legacy transition to “module v2.0”



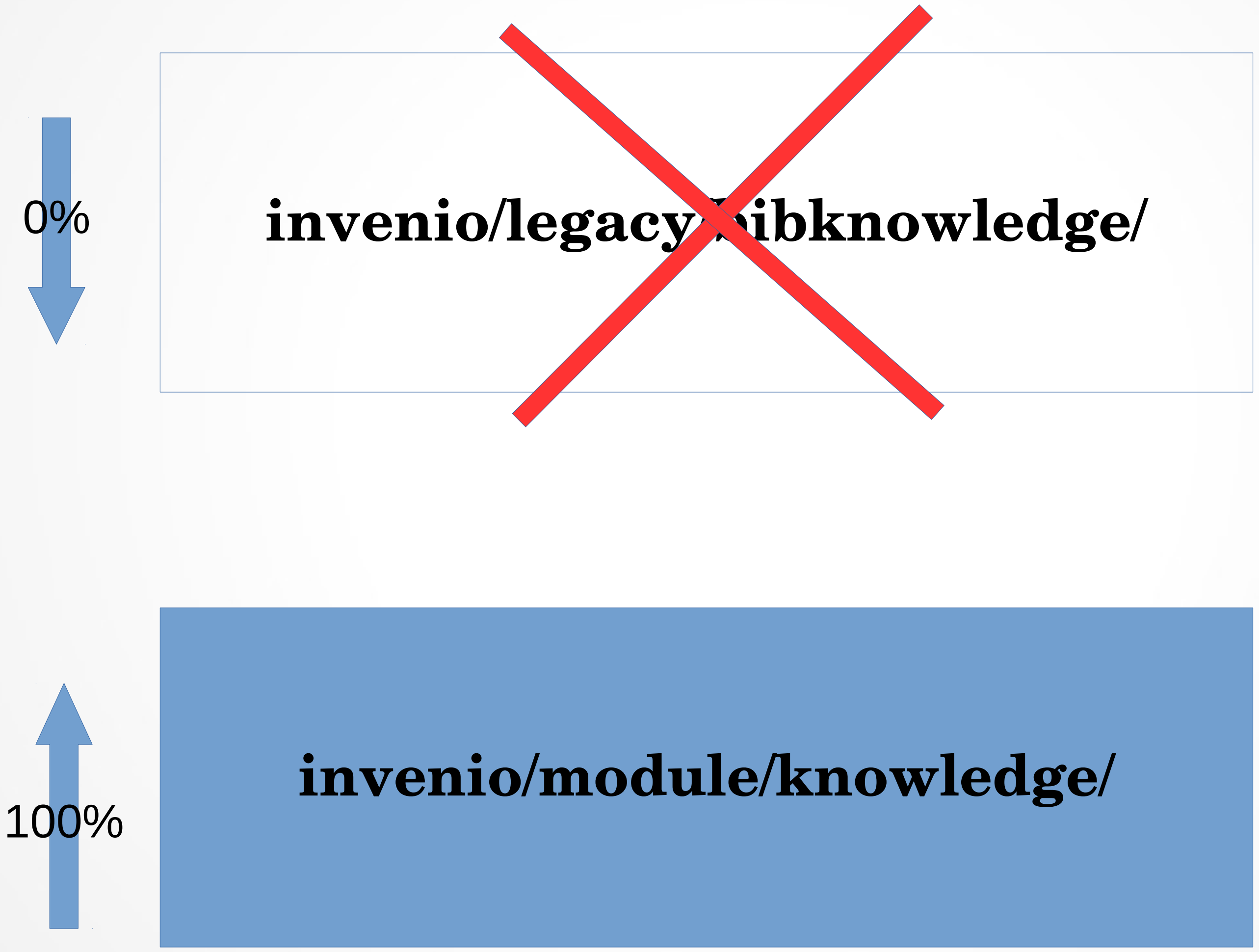
From Legacy to Module v2.0

Then, Legacy transition to “module v2.0”


30%
invenio/legacy/bibknowledge/
adminlib.py
admin.py
webinterface.py
Note: the legacy code use our new module.
70%
invenio/module/knowledge/

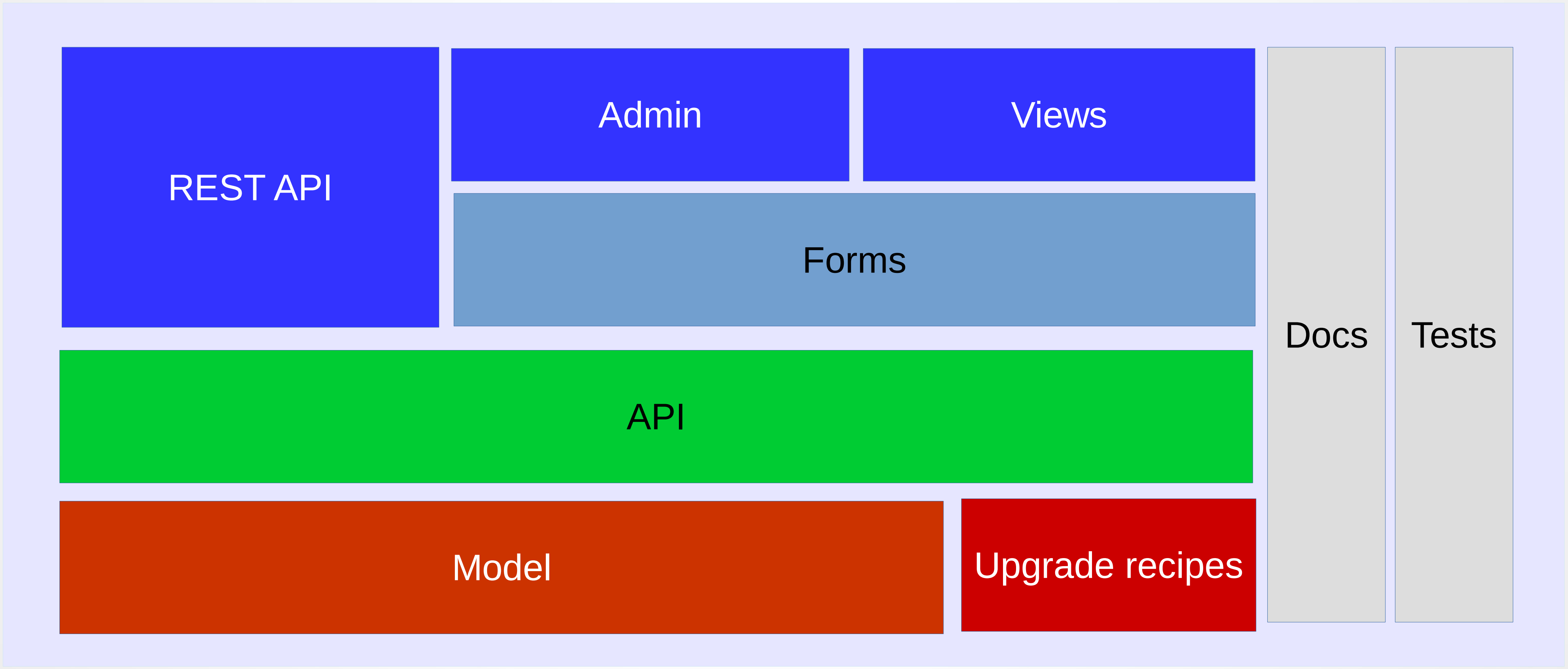
From Legacy to Module v2.0

In the end, only "module v2.0" survive



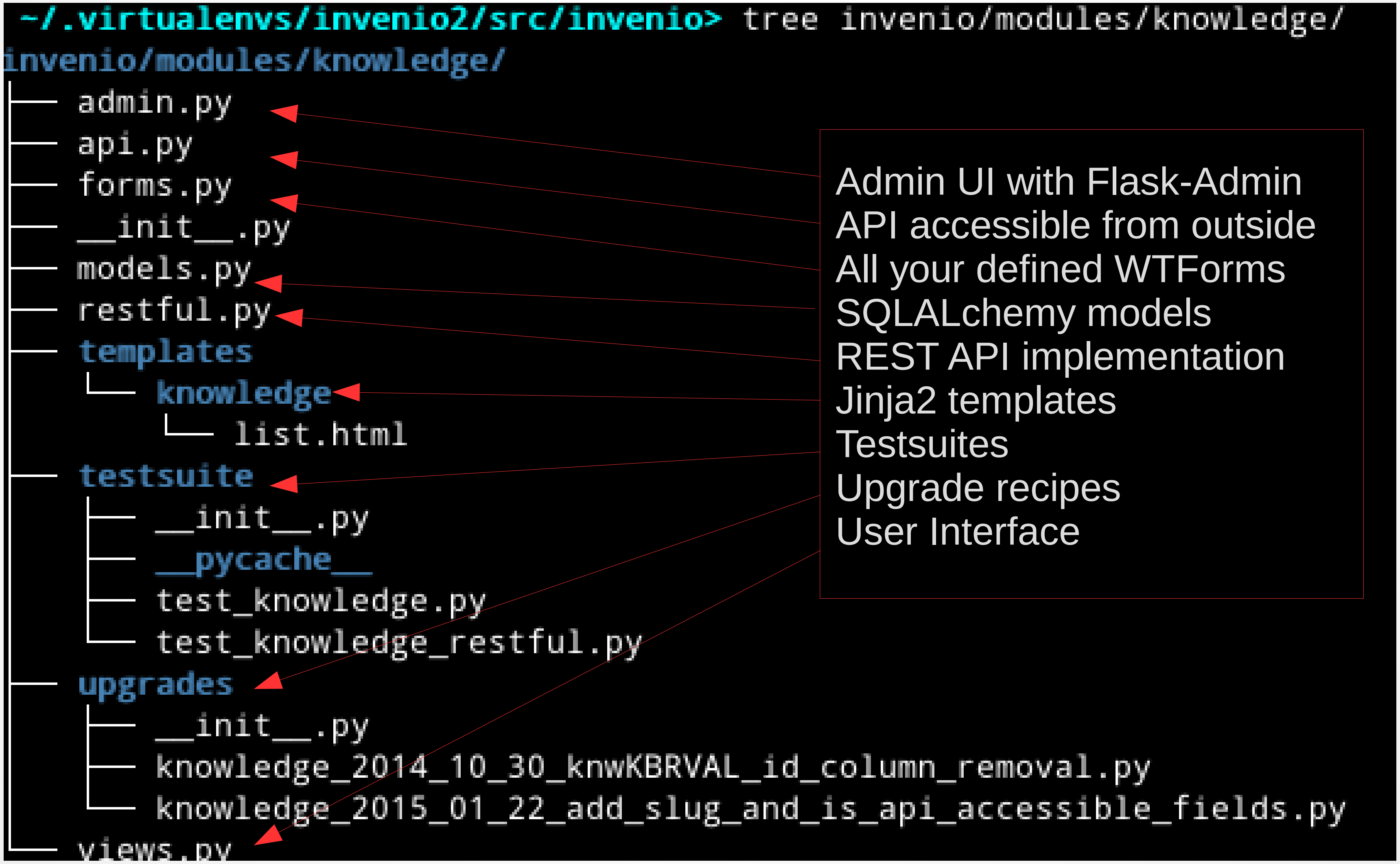
Stack

Visual representation of a module



From Legacy to Module v2.0

How a module is composed?

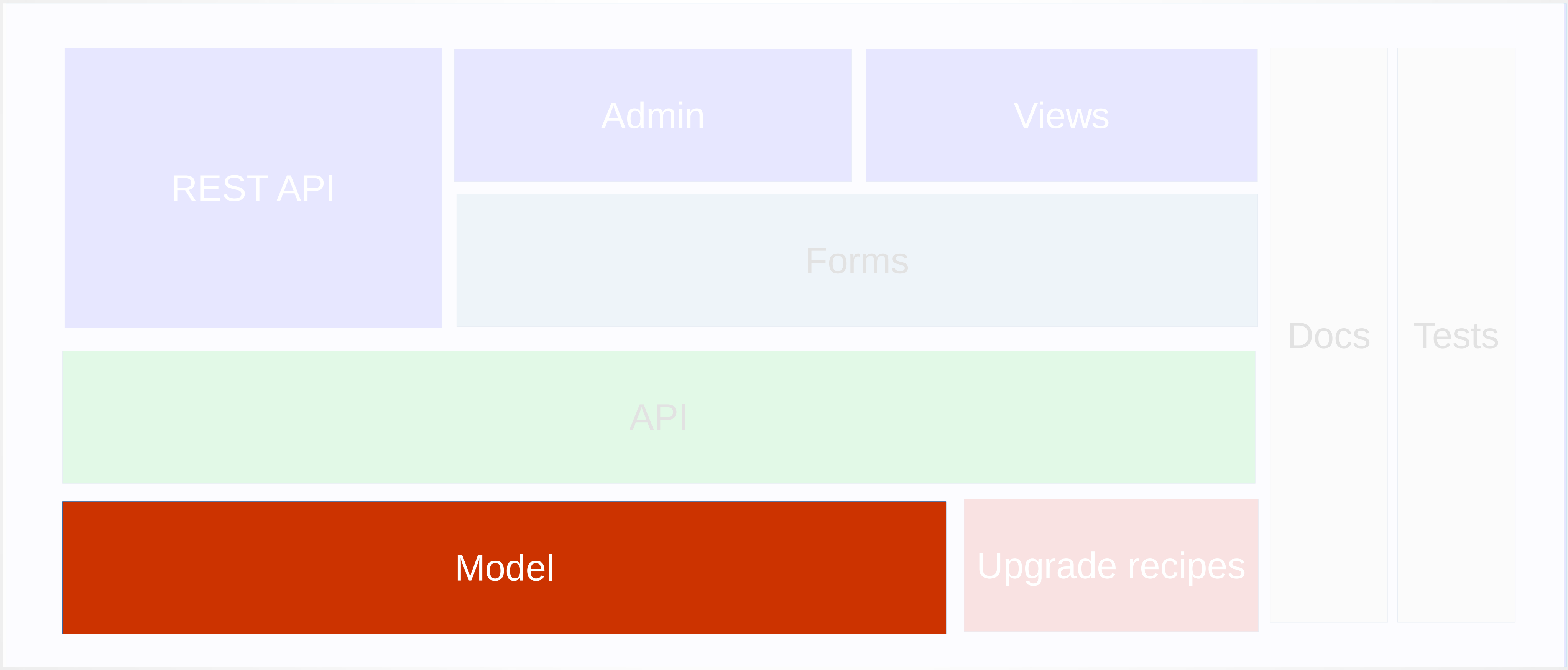


```
 ~/.virtualenvs/invenio2/src/invenio> tree invenio/modules/knowledge/
invenio/modules/knowledge/
├── admin.py
├── api.py
├── forms.py
├── __init__.py
├── models.py
├── restful.py
├── templates
│   └── knowledge
│       └── list.html
├── testsuite
│   ├── __init__.py
│   ├── __pycache__
│   ├── test_knowledge.py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

Admin UI with Flask-Admin
API accessible from outside
All your defined WTForms
SQLALchemy models
REST API implementation
Jinja2 templates
Testsuites
Upgrade recipes
User Interface

Stack

Visual representation of a module



From RAW SQL to SQLAlchemy

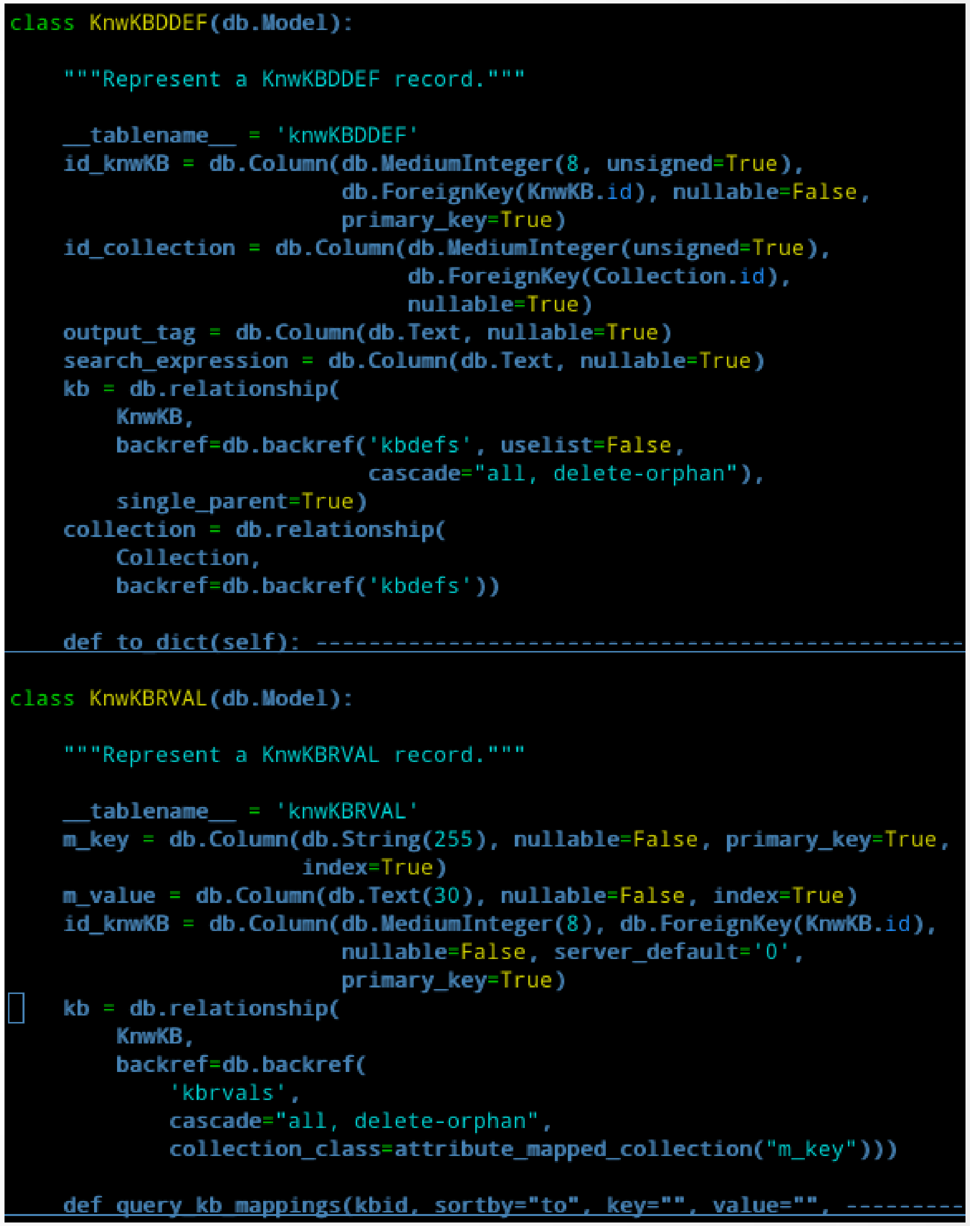
From
run_sql()
To
models.py

Why?

- Independence from the DB server.
- More easy to develop and maintain.

How?

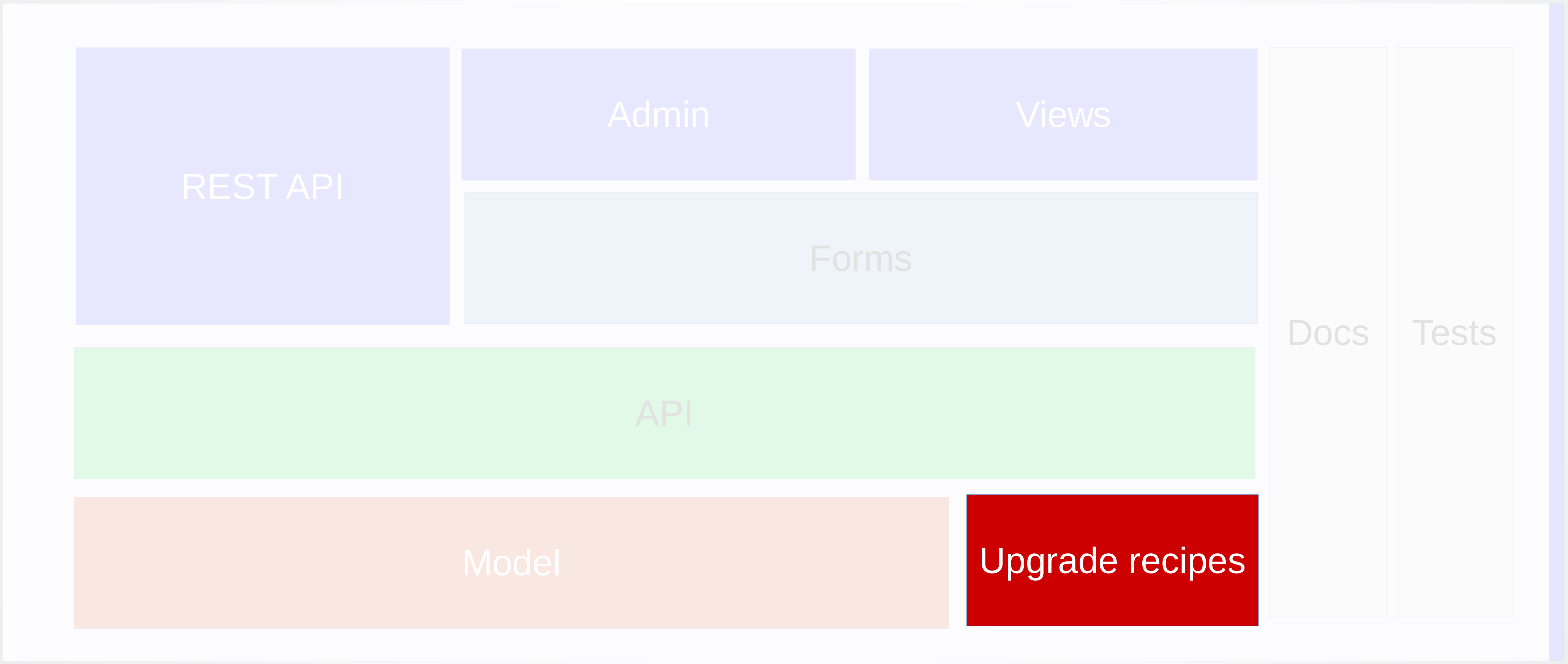
- Generating the model (Alembic can help you) and refine it.
- For any SQL, we need to replace it with SQLAlchemy ORM equivalente.



```
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="", ---------
```

Stack

Visual representation of a module



How to write upgrade's recipes

Old data model vs New model

how to upgrade your database in production environment without break it?

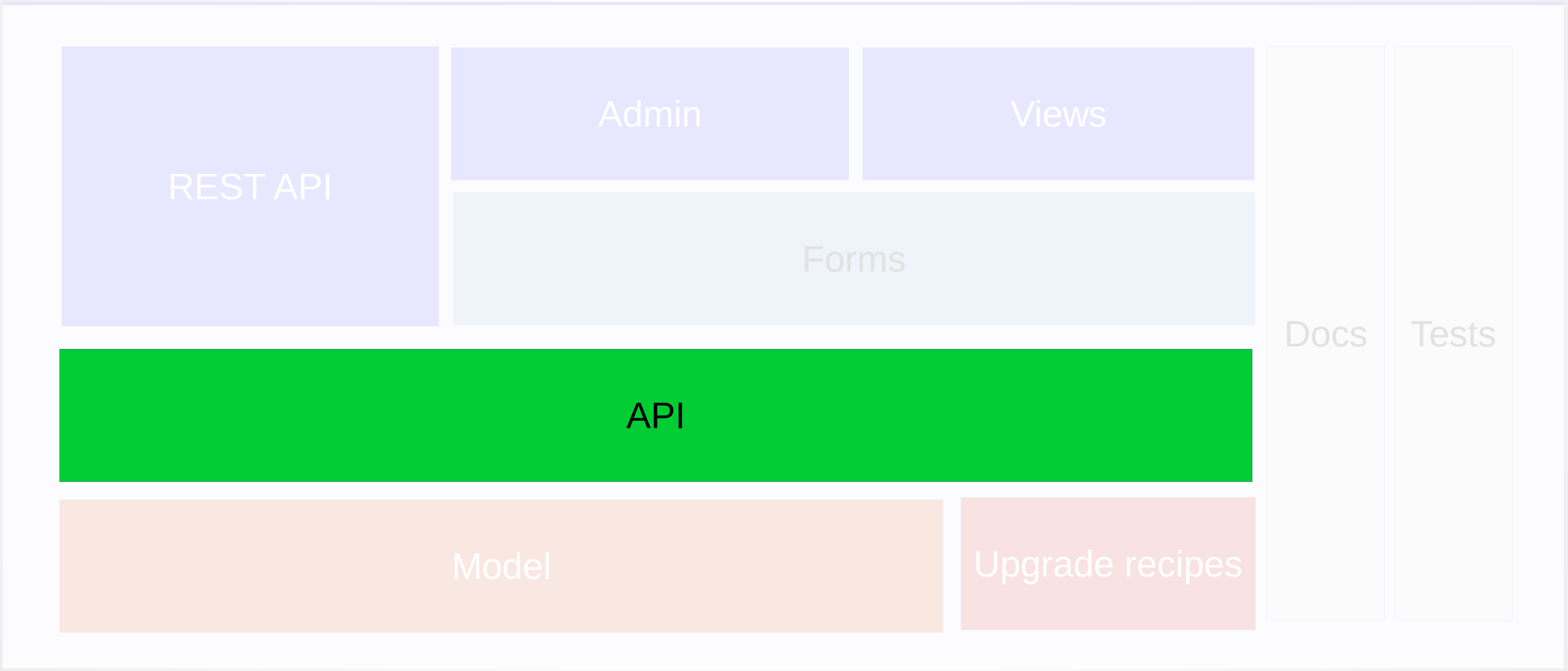
```
-  old   Database
+ new Model
-----------------------
= upgrade recipe
```

<u>Alembic</u> is a database migrations tool

It helps you to write python code to upgrade the database.

Stack

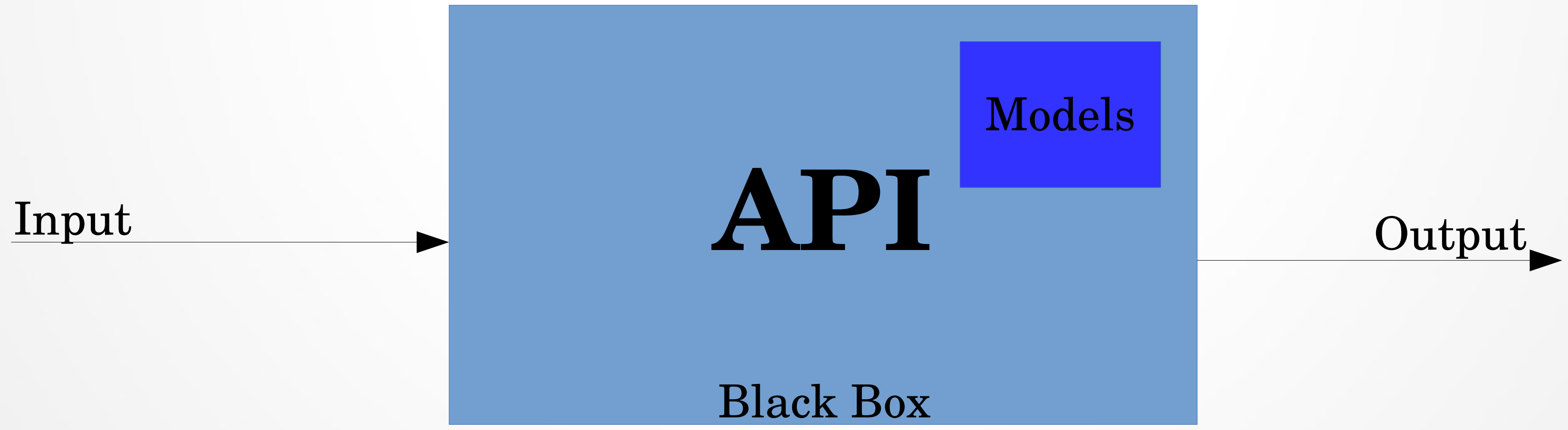
Visual representation of a module



Refactoring API: re-implement and deprecate.

What we want?

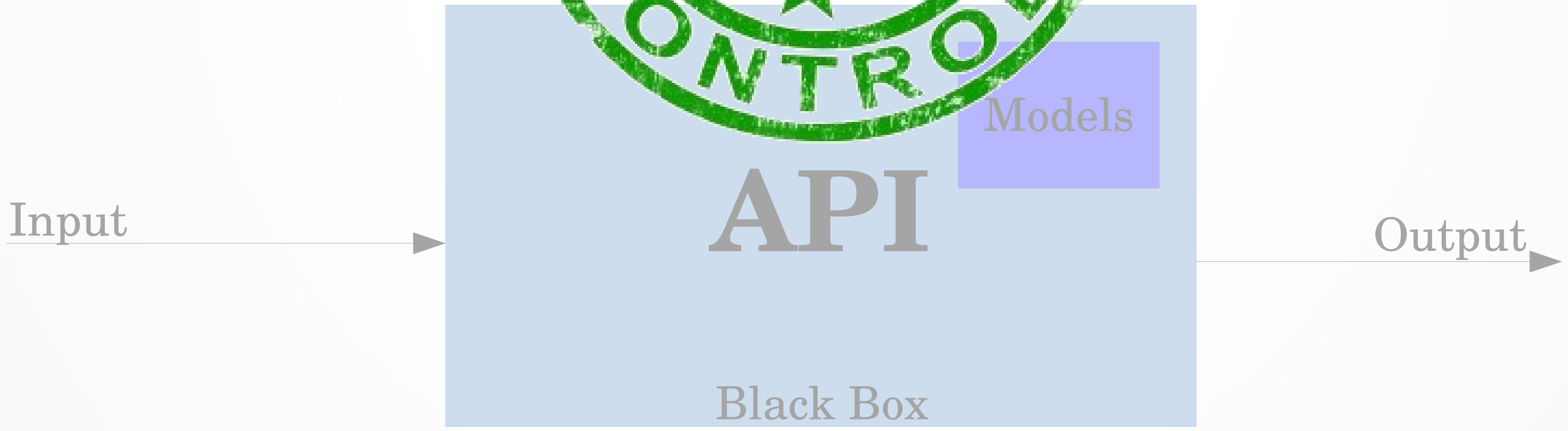
- Each request have to pass from the api (<u>no direct access to models</u>).
- Respect the old API (<u>backward compatibility</u>)
- <u>Meanwhile refactoring</u>: deprecate not useful code and implement a new cool API.



Refactoring API: re-implement and deprecate.

What we want?

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

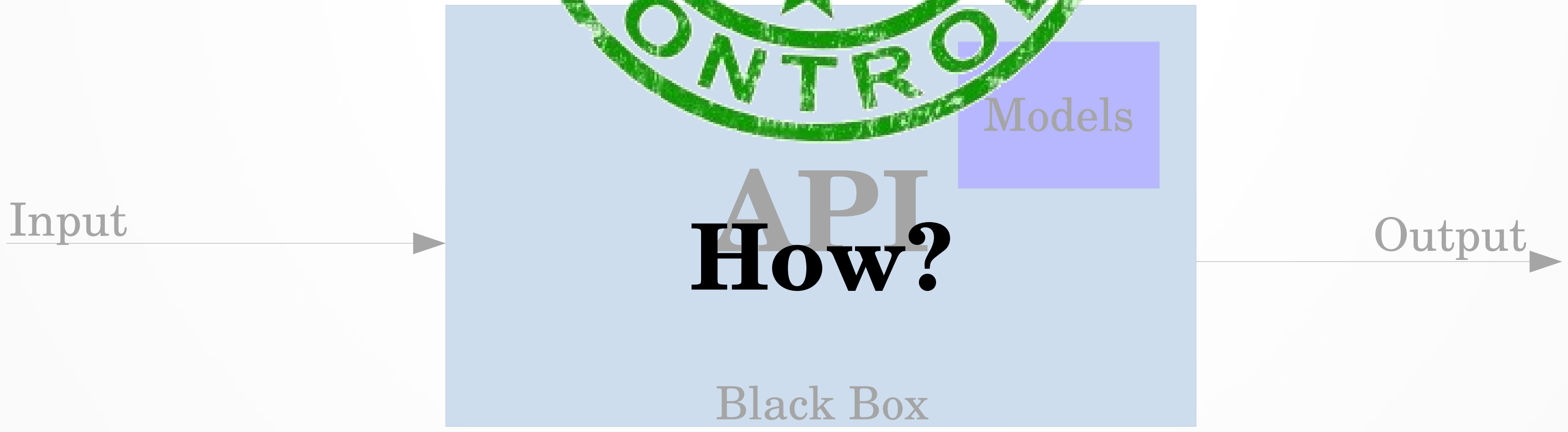


Refactoring API: re-implement and deprecate.

What we want?

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

QUALITY APPROVED CONTROL

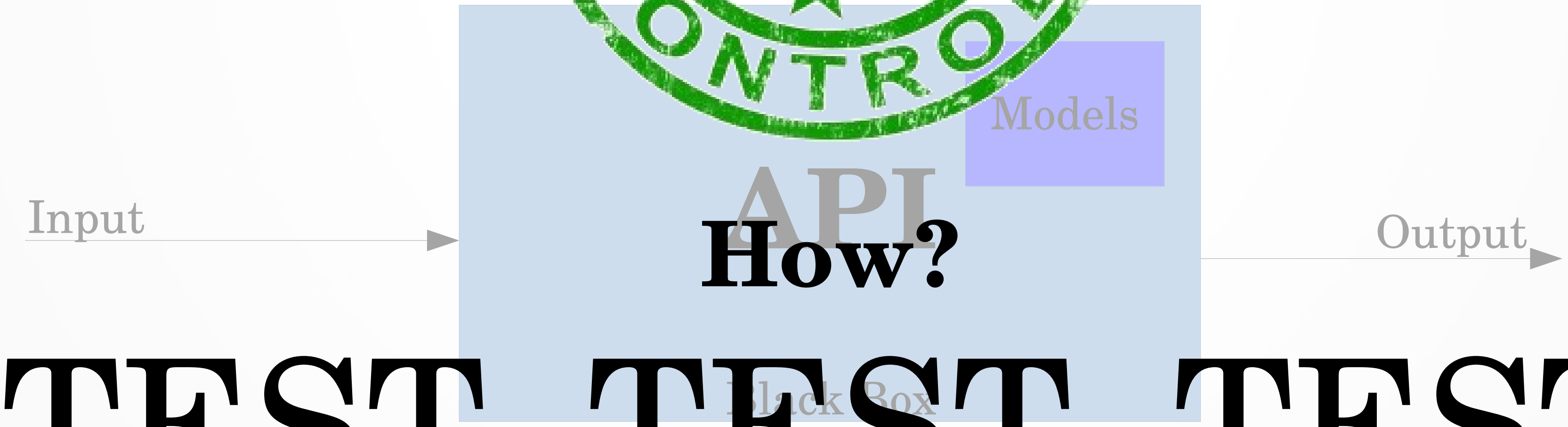


How?

Refactoring API: re-implement and deprecate.

What we want?

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

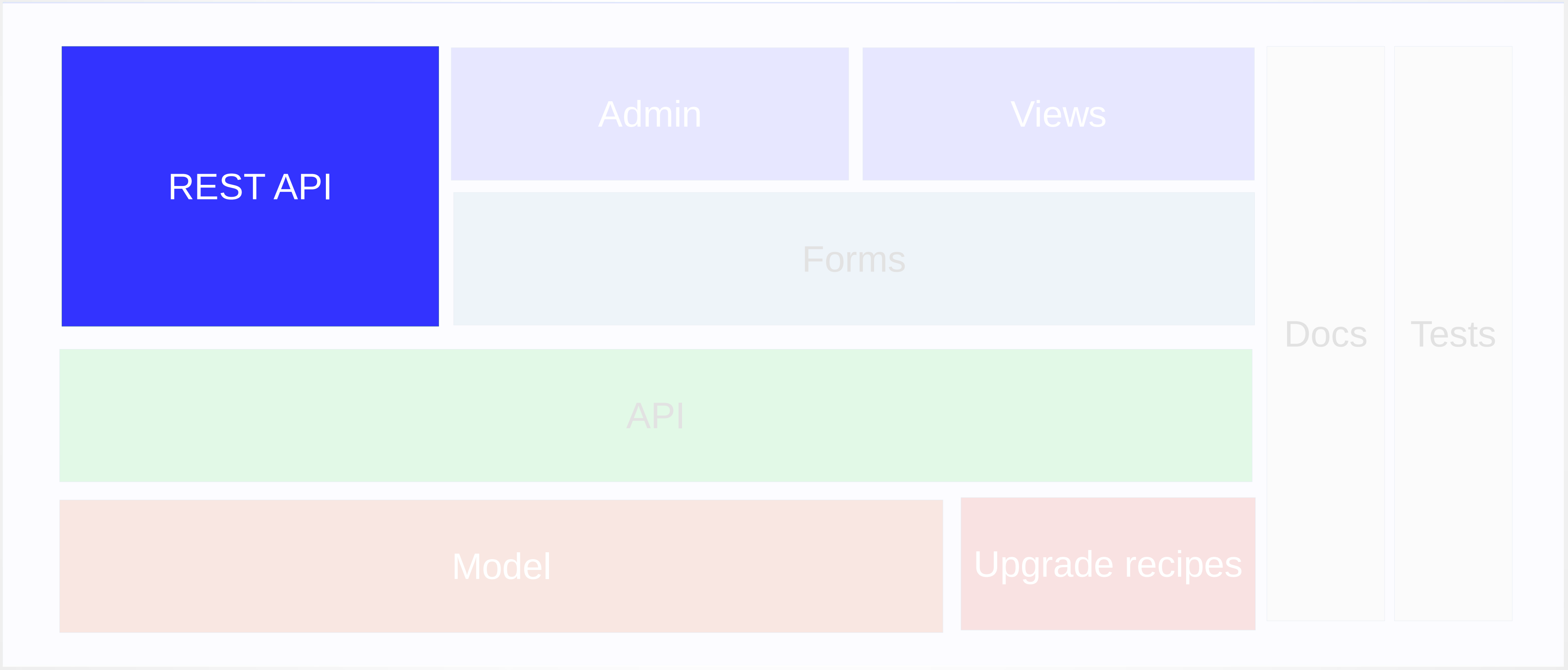


How?

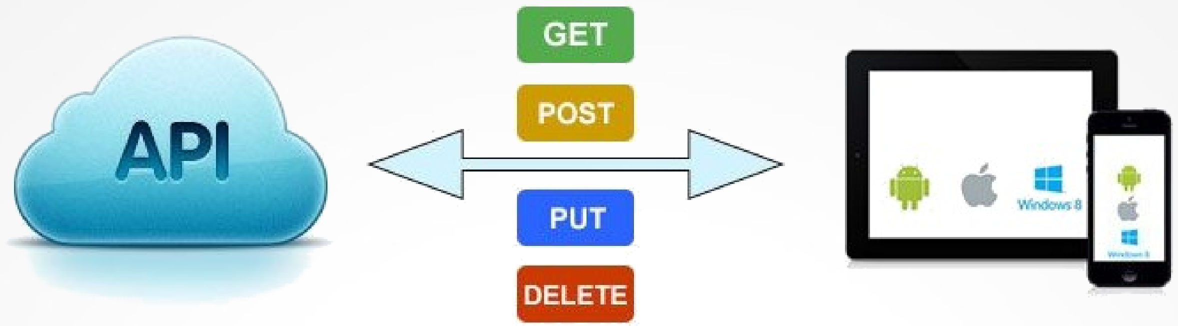
TEST, TEST ,TEST

Stack

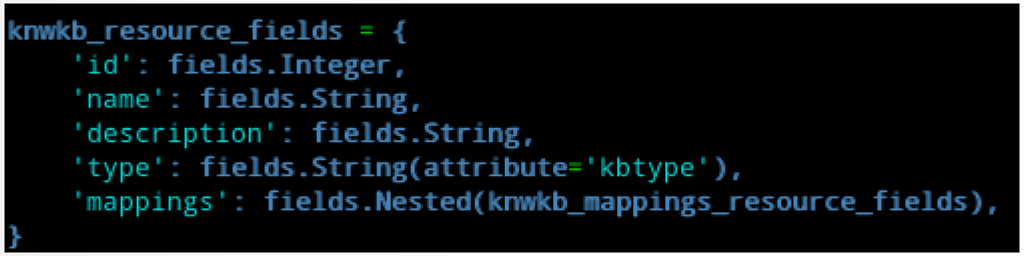
Visual representation of a module



Write REST API

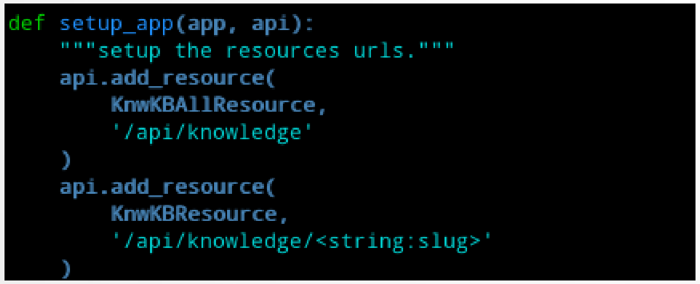


Define the object structure



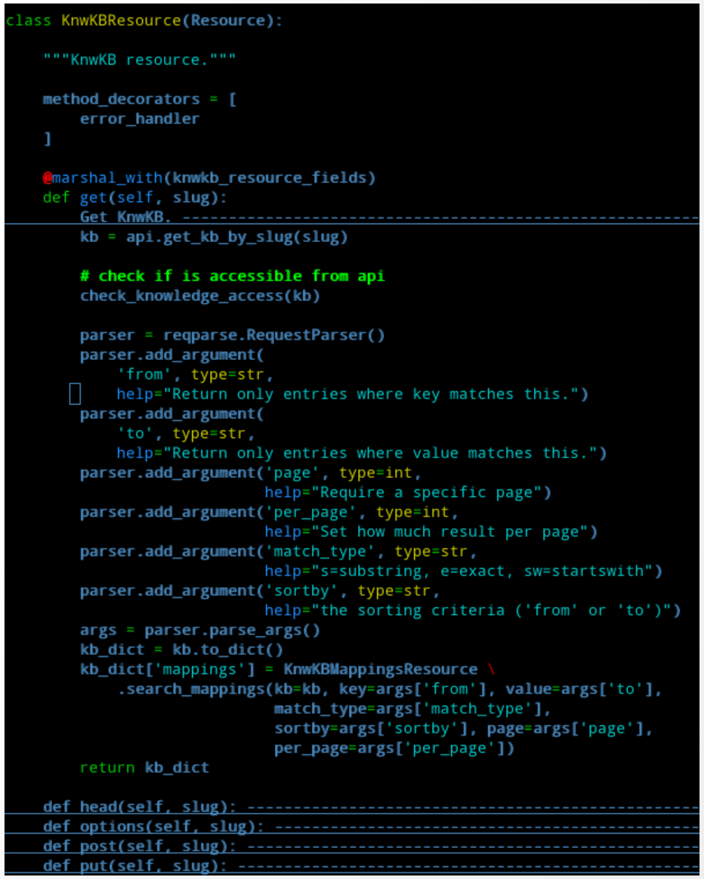
```
knwkb_resource_fields = {
    'id': fields.Integer,
    'name': fields.String,
    'description': fields.String,
    'type': fields.String(attribute='kbtype'),
    'mappings': fields.Nested(knwkb_mappings_resource_fields),
}
```

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>'
    )
```

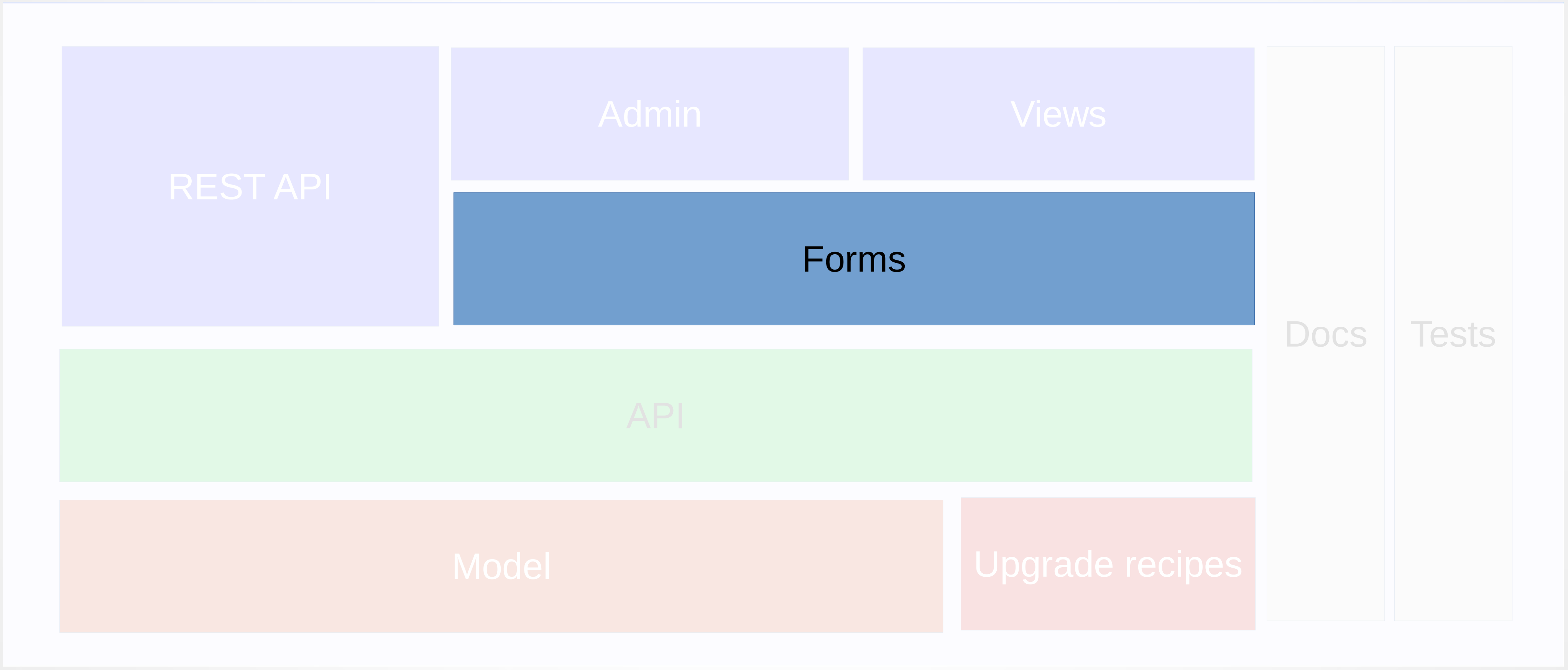
Define a resource



```
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): -------------------------------------------------
```

Stack

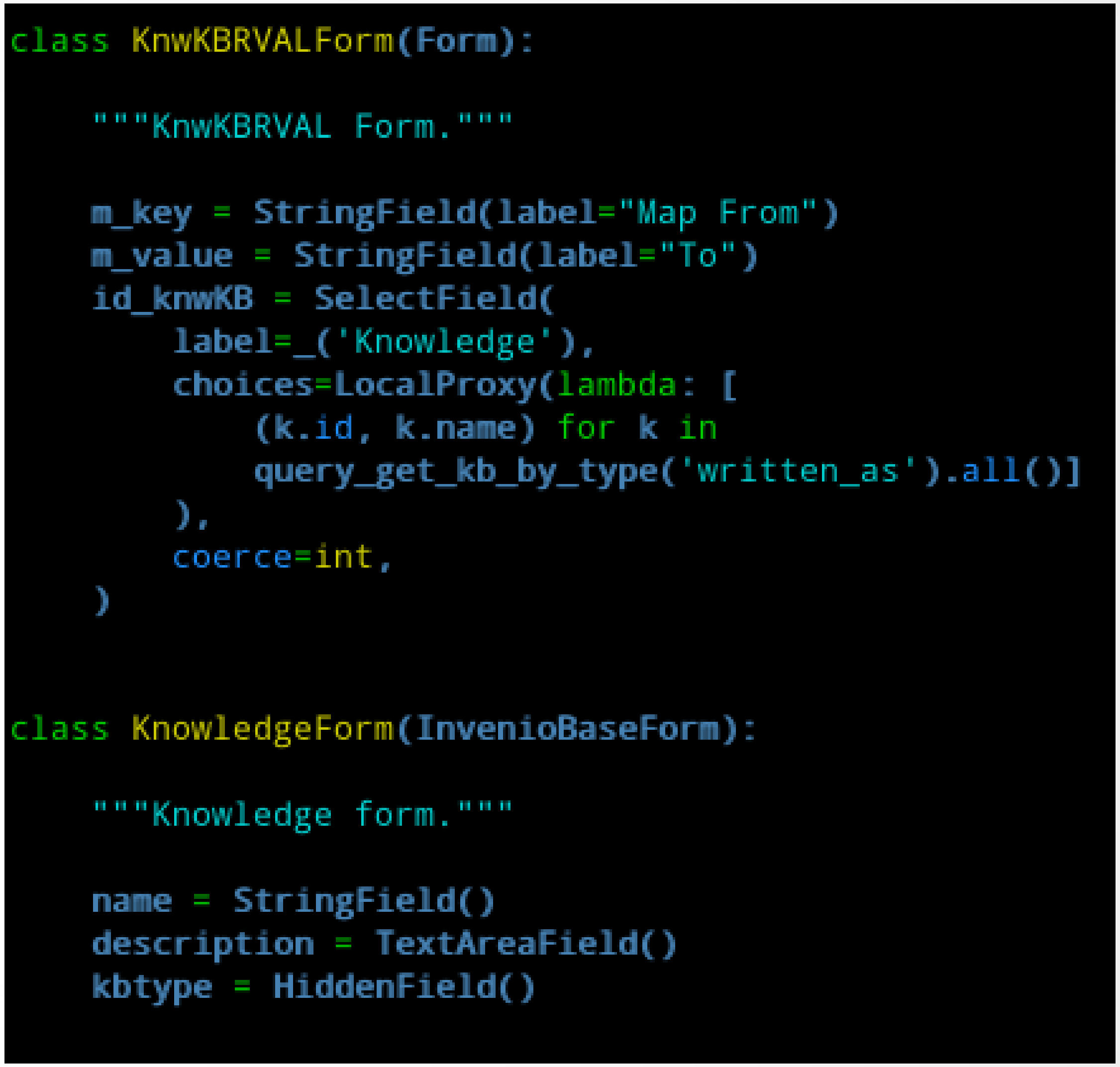
Visual representation of a module



Write forms with WTForms

WTForms

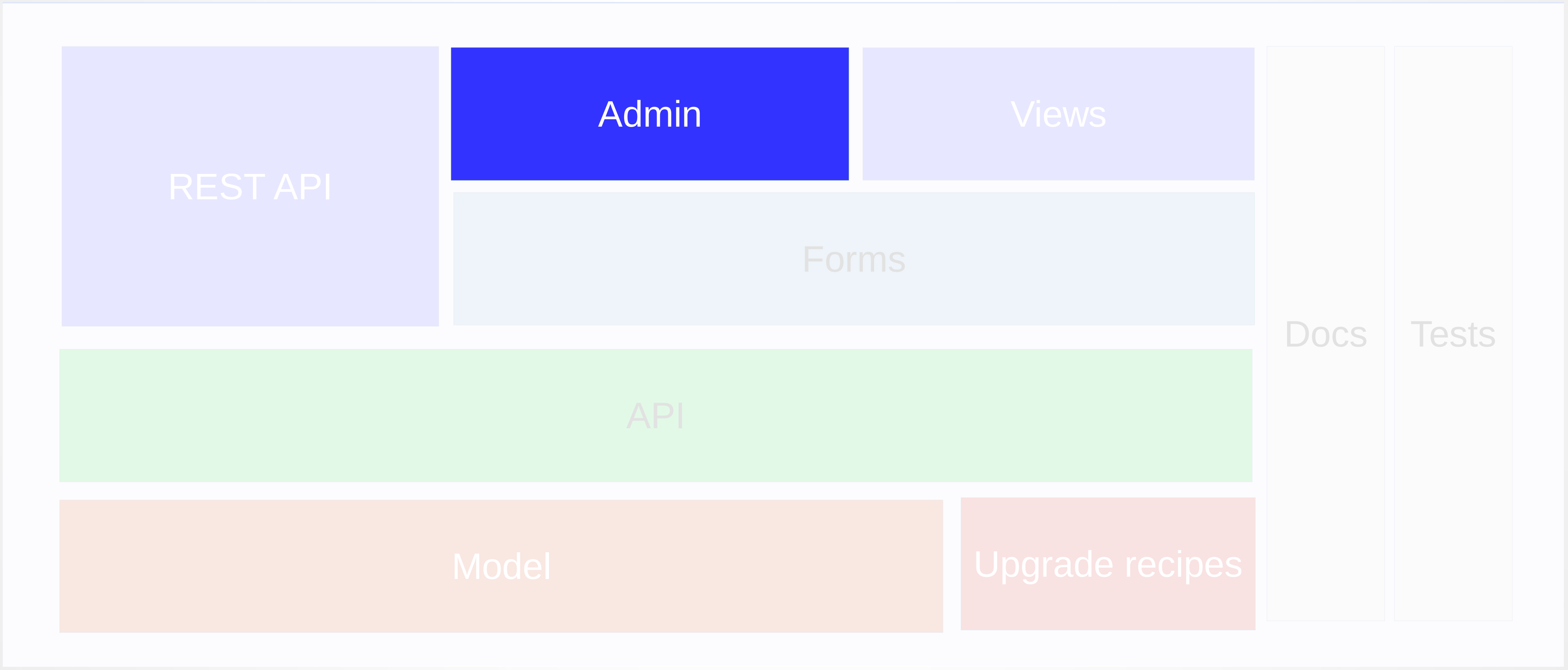
A flexible forms validation and rendering library for Python



```
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()
```

Stack

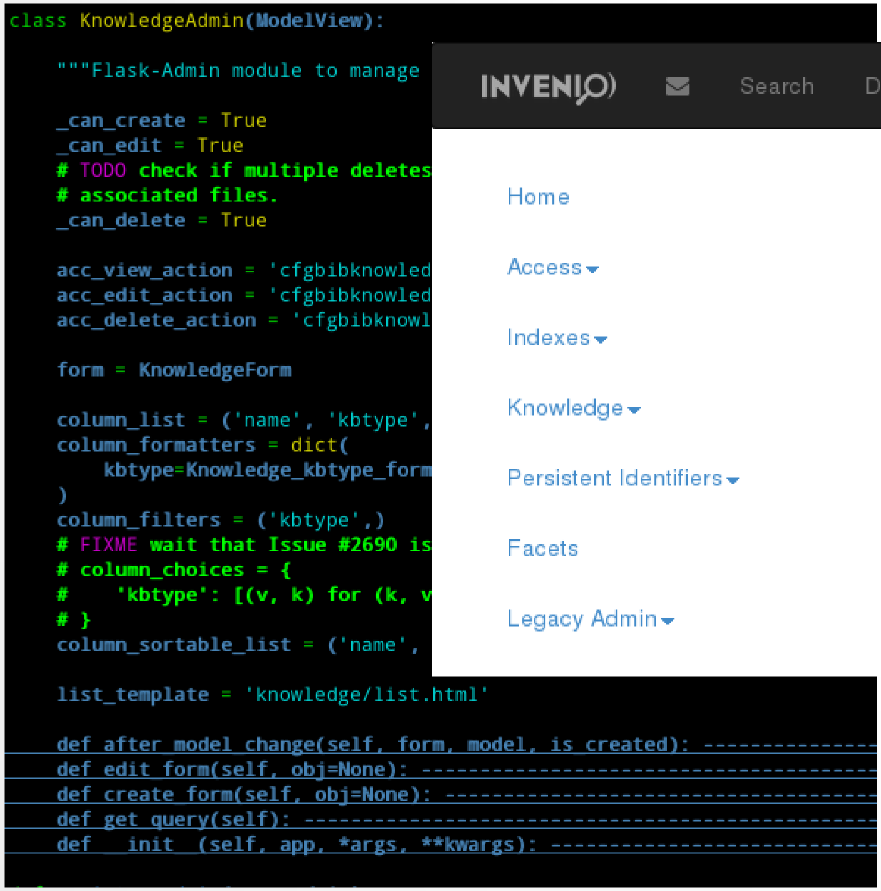
Visual representation of a module



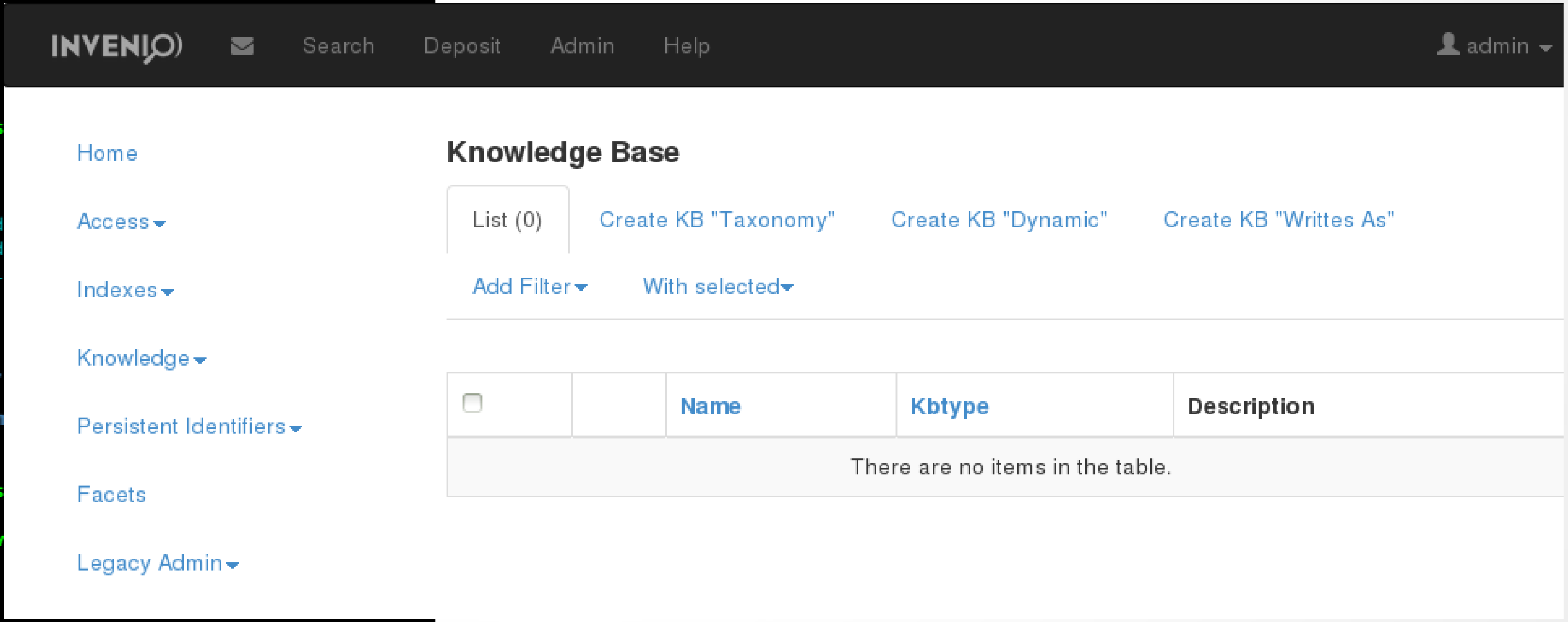
The Admin interface with Flask-Admin

Easy to extend admin interface through “hooks”.

Admin interface for Knowledge



```
class KnowledgeAdmin(ModelView):

    """Flask-Admin module to manage

    _can_create = True
    _can_edit = True
    # TODO check if multiple deletes
    # associated files.
    _can_delete = True

    acc_view_action = 'cfgbibknowled
    acc_edit_action = 'cfgbibknowled
    acc_delete_action = 'cfgbibknowl

    form = KnowledgeForm

    column_list = ('name', 'kbtype',
    column_formatters = dict(
        kbtype=Knowledge_kbtype_form
    )
    column_filters = ('kbtype',)
    # FIXME wait that Issue #2690 is
    # column_choices = {
    #     'kbtype': [(v, k) for (k, v
    # }
    column_sortable_list = ('name',

    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): -------------------------
```



Knowledge Base

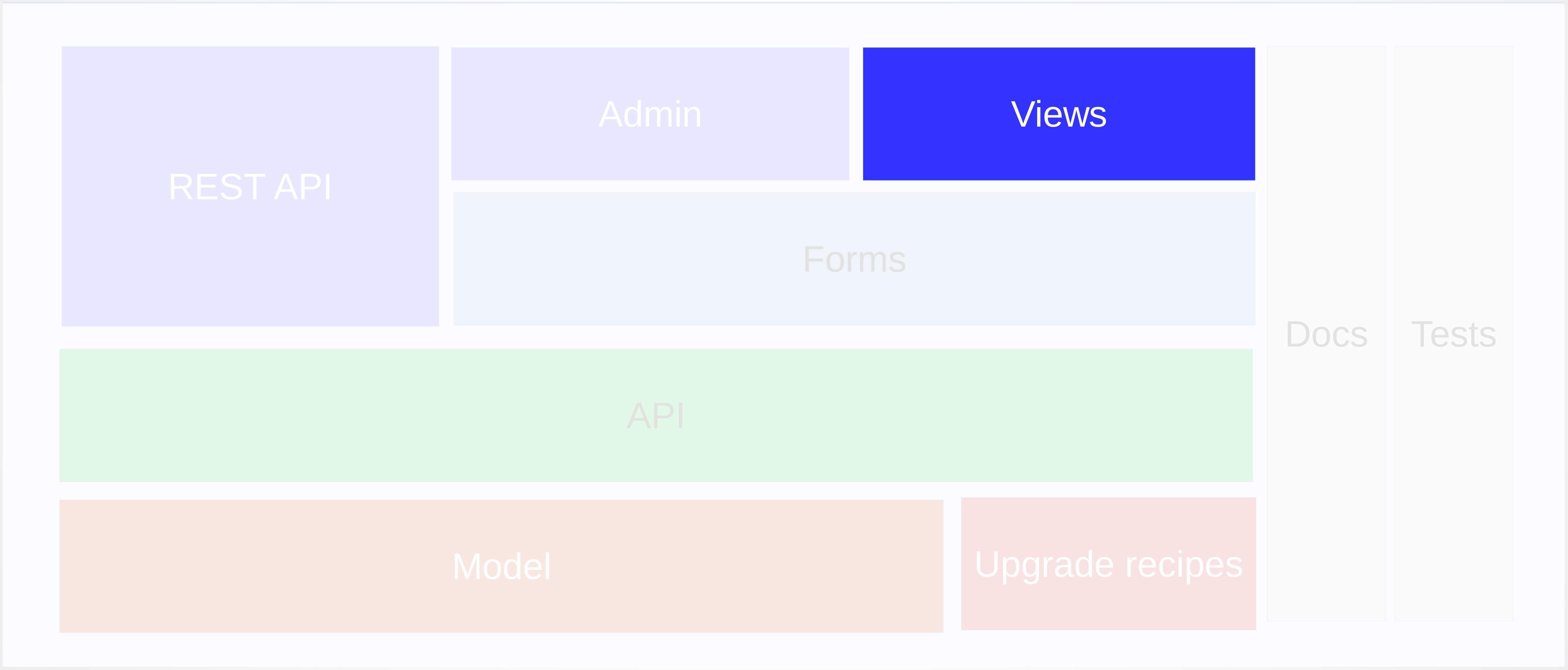
List (0) | Create KB "Taxonomy" | Create KB "Dynamic" | Create KB "Writtes As"

Add Filter | With selected

☐		Name	Kbtype	Description
There are no items in the table.				

Stack

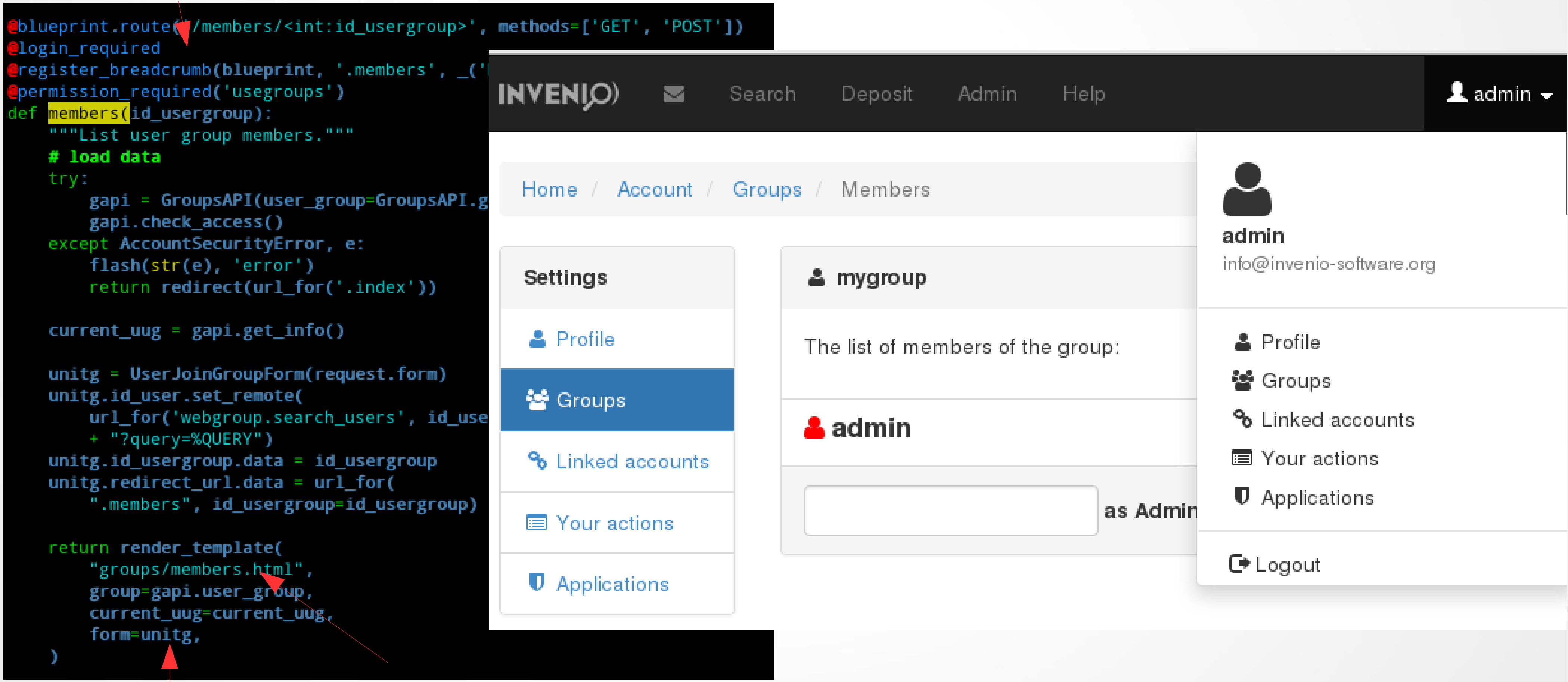
Visual representation of a module



User's web interface with Jinja templates

.. but, if you need also a “normal” web interface?

Endpoint & params



```python
@blueprint.route('/members/<int:id_usergroup>', methods=['GET', 'POST'])
@login_required
@register_breadcrumb(blueprint, '.members', _('
@permission_required('usegroups')
def members(id_usergroup):
    """List user group members."""
    # load data
    try:
        gapi = GroupsAPI(user_group=GroupsAPI.g
        gapi.check_access()
    except AccountSecurityError, e:
        flash(str(e), 'error')
        return redirect(url_for('.index'))

    current_uug = gapi.get_info()

    unitg = UserJoinGroupForm(request.form)
    unitg.id_user.set_remote(
        url_for('webgroup.search_users', id_use
        + "?query=%QUERY")
    unitg.id_usergroup.data = id_usergroup
    unitg.redirect_url.data = url_for(
        ".members", id_usergroup=id_usergroup)

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

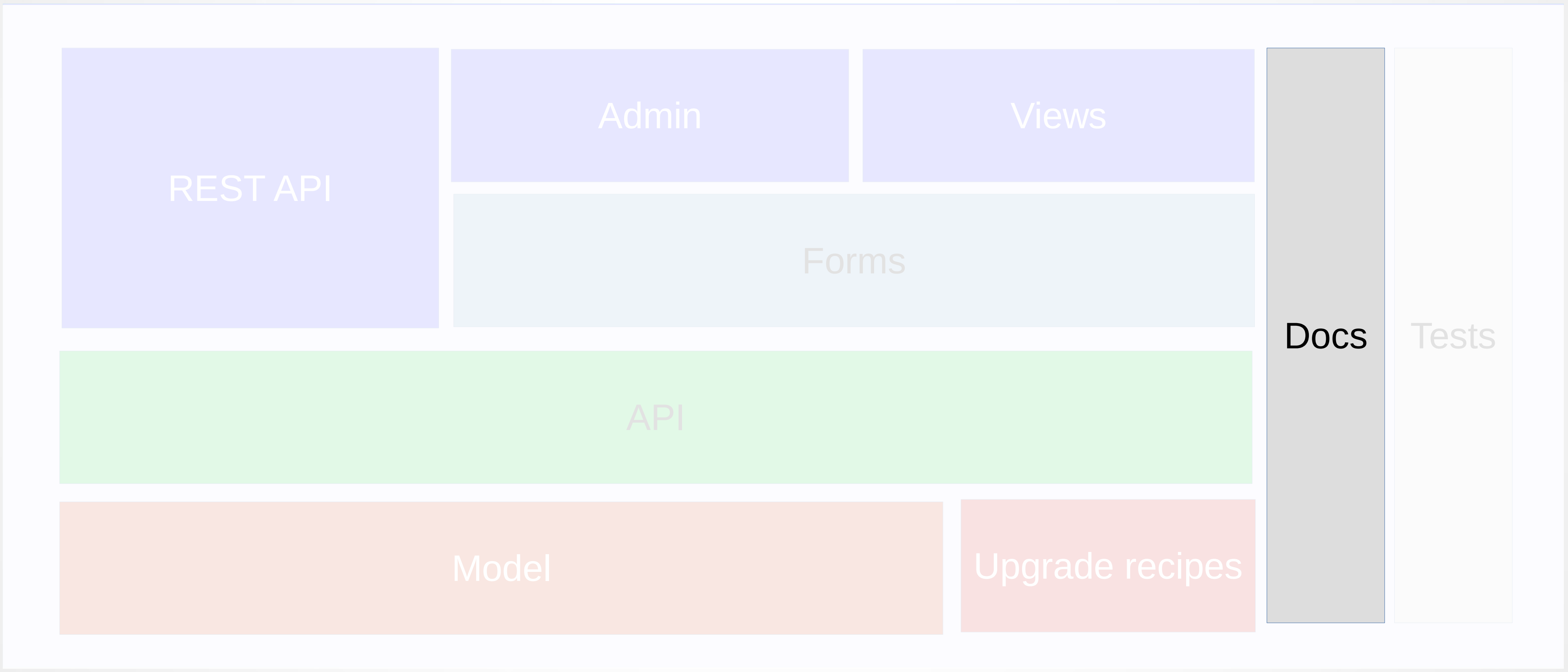
The list of members of the group:

Your template

The objects passed to the template

Stack

Visual representation of a module

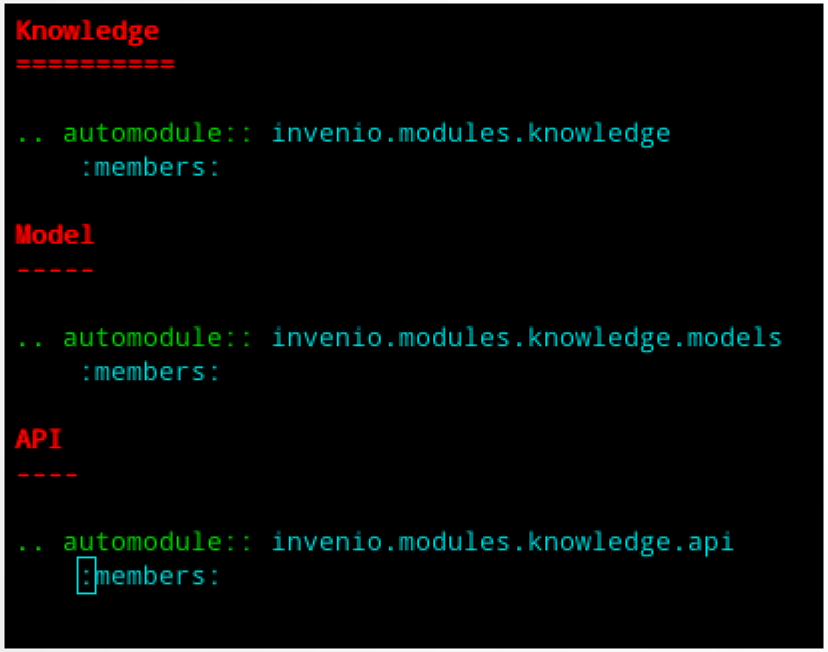


Write documentation

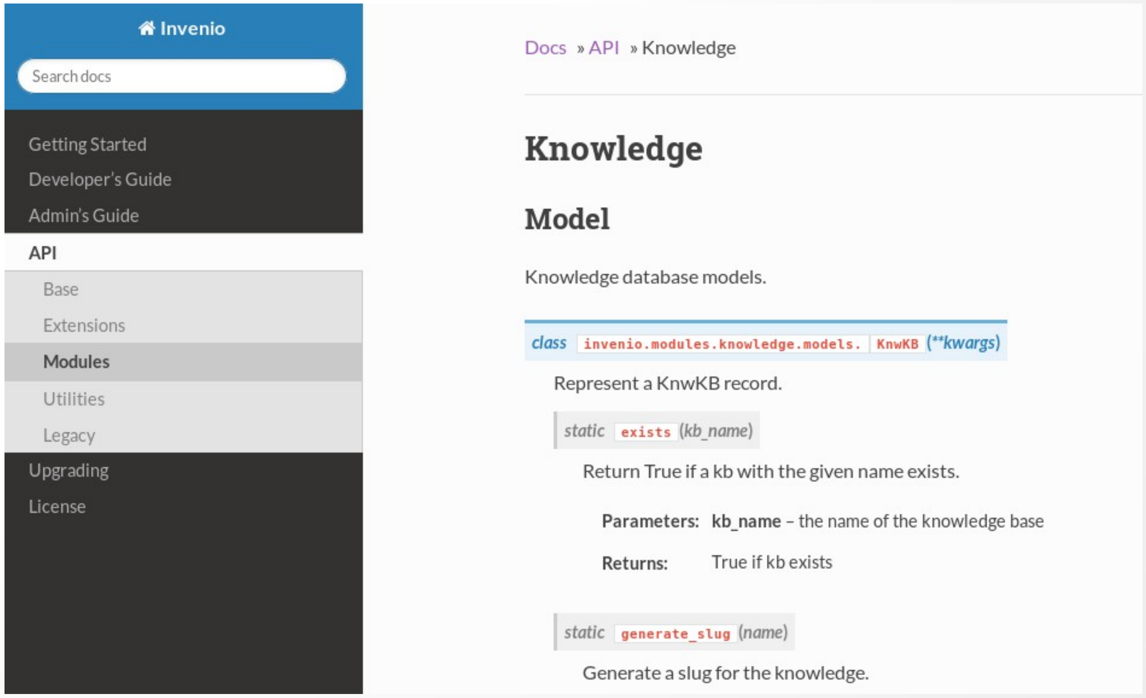
"Write as much documentation as possible!"

Why?

To refactoring the code, you need to know what the code does..



```
Knowledge
==========

.. automodule:: invenio.modules.knowledge
    :members:

Model
-----

.. automodule:: invenio.modules.knowledge.models
    :members:

API
----

.. automodule:: invenio.modules.knowledge.api
    :members:
```



Knowledge

Model

Knowledge database models.

class `invenio.modules.knowledge.models.KnwKB`(***kwargs*)

Represent a KnwKB record.

static `exists`(*kb_name*)

Return True if a kb with the given name exists.

Parameters: **kb_name** – the name of the knowledge base

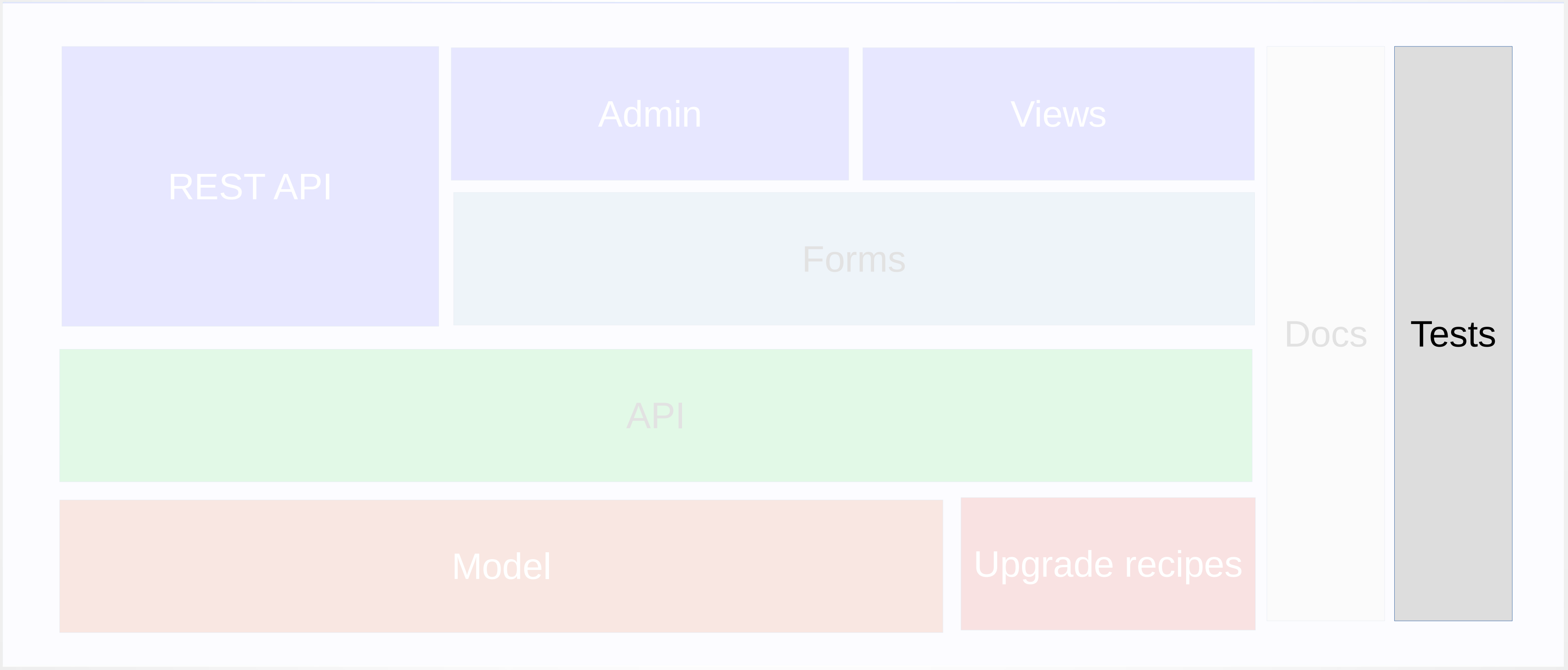
Returns: True if kb exists

static `generate_slug`(*name*)

Generate a slug for the knowledge.

Stack

Visual representation of a module



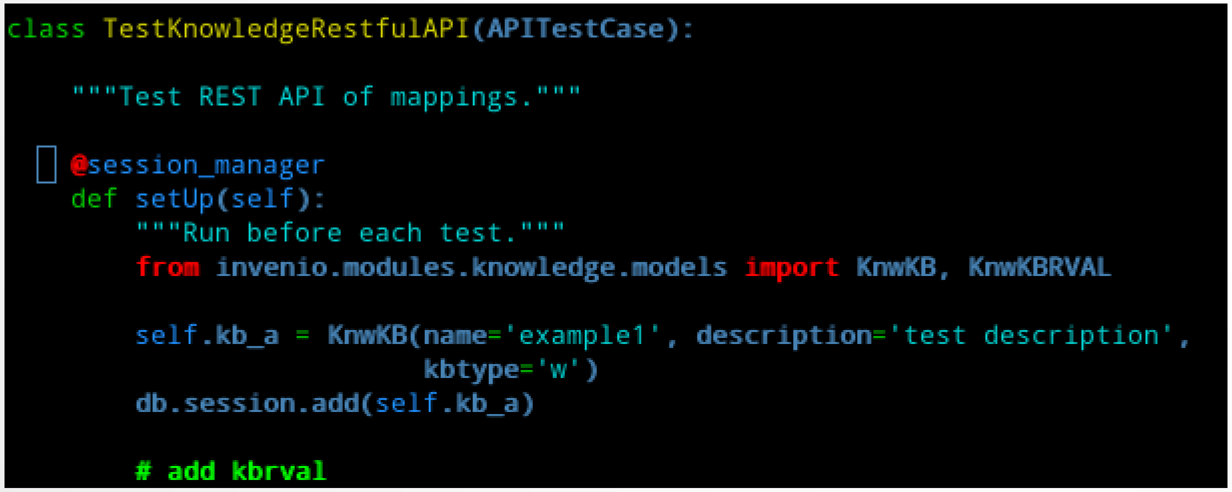
Test the new code

"Test Driven philosophy is your best friend"

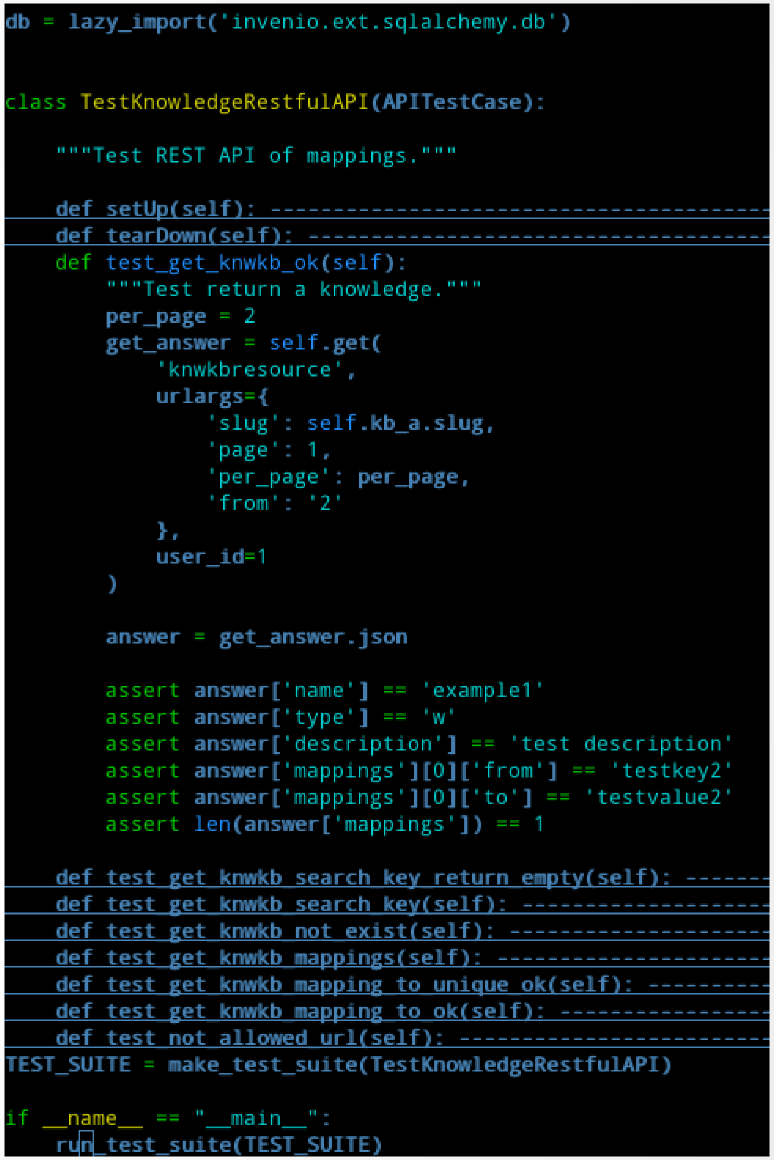
Start your test:

$> **cd** invenio/modules/knowledge/testsuite

$> **py.test** test_knowledge_restful.py

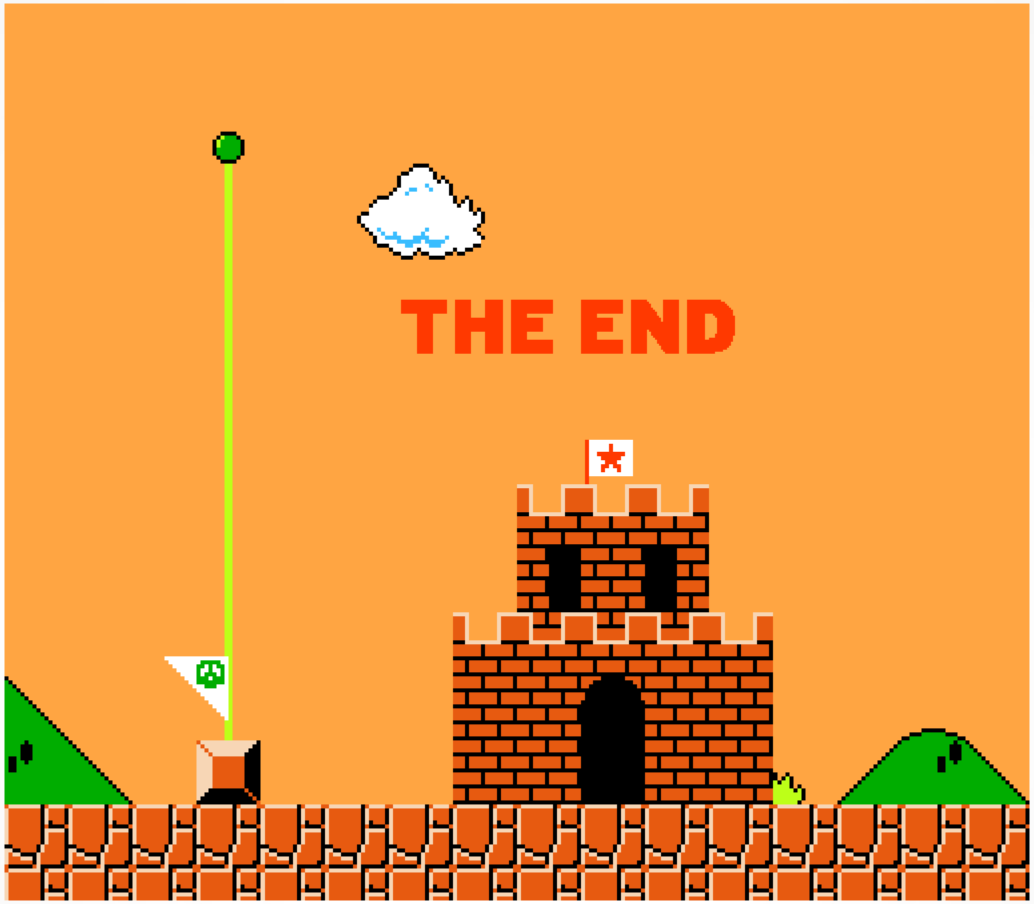


```
class TestKnowledgeRestfulAPI(APITestCase):

    """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
```



```
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)
```

The End



Thank you for your attention