py-ispyb : current status

Ivars Karpičs

EMBL

Content

- Motivation
- py-ispyb
  - Libraries
  - Structure: plug-in extension, modules and routes
  - Site specific configuration
  - Security: authentication, tokens and authorization
  - Data models and input validation
  - Endpoint design
  - Documentation: doc strings and swagger ui
  - Code style and quality, continuous integration and deployment
- Summary
  - Roadmap
  - How to contribute and collaborate

Motivation

Enquiry done by Alex and presented during the last ISPyB meeting in Hamburg.

		ESRF	DLS	SOLEIL	MAXIV	ALBA	EMBL
Man power							
	FTEs	1.5	3.5+	0.5	0.5	0.5	0.5
Current Installation							
	Frontend [ISPyB | EXI | SynchWeb]	EXI ISPyB EXI2	SynchWeb	ISPyB	ISPyB EXI	ISPyB	ISPYB, EXI, SynchWeb
	BackEND [ISPyB | SynchWeb]	ISPyB	SynchWeb	ISPyB	ISPyB	ISPyB	ISPYB, Synchweb, ispyb-api
Functionality			ispyb-api (python & java)				
	Techniques to support	MX, TR-MX, BioSAXS EM	MX EM XPDF Powder +	MX, BioSAXS	MX	MX	MX, TR-MX, XRI
New start							
	Preferred list of programming languages	Python NodeJS JAVA	Python	NodeJS, Java	Python, NodeJS	Python, NodeJS, Java	any
	Preferred list of DB engines	Mongo Maria Oracle	MariaDB, Postgres?	MariaDB, Oracle, Postgres	Maria, Postgres	Maria, Mongo	any
	Preferred list of API	REST GraphQL	REST, GraphQL	REST, GraphQL	GraphQL, REST	REST, GraphQL	REST

Motivation

Some bullet points from the meeting minutes:

- The immediate target is to develop a shared backend based on the existing database.
- The developers are tasked with making a working prototype of a shared back-end API for one precisely defined domain. In the process they will pilot a framework for collaboration and a set of technology choices, design rules, good practices, and specifications. This will demonstrate that the process works, that the result is useful, and establish a model that can be used (possibly after some tuning) to proceed to a complete back-end.
- The prototype should be demonstrated at the next ISPyB/MXCuBE at ALBA in June 2020.
- The technology choices should be based on agreement between the participants. Based on the poll of participating groups (and in the absence of major new arguments) that would mean MariaDB, Python, and REST / web services.

py-ispyb

- Initial project *ispyb_backend_prototype* on my personal github account.
- Prototype was several times presented during the developers web meetings and suggestion was to continue the development.
- Moved to ISPyB account: https://github.com/ispyb/py-ispyb

Key features

- Allows to use *existing* database. Also if it is slightly different from the “official” db.
- Focuses on one specific domain: *Proposals*.
- *Python* and *REST* api.

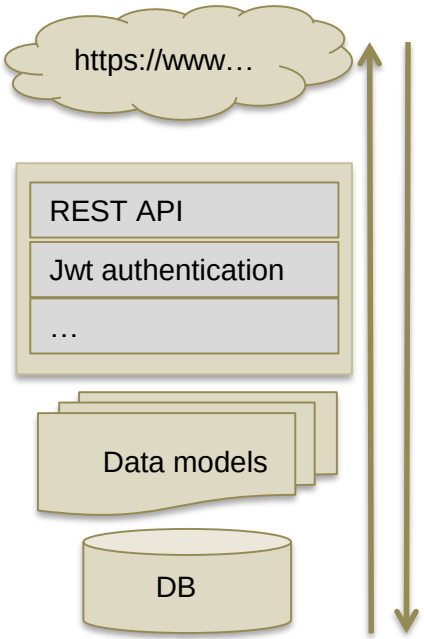
py-ispyb : libraries

- **Python 3.x**
- **Flask**>=1.1,<2
- **flask-restx**
- Flask-Cors>=3.0.8,<4

- **SQLAlchemy**>=1.3.0,<2
- Flask-SQLAlchemy>=2.4,<3

- **marshmallow**>=2.13.5,<3
- flask-marshmallow>=0.7,<0.8
- marshmallow-sqlalchemy>=0.12,<0.13
- marshmallow_jsonschema

- **Pyjwt**
- gunicorn
- mysqlclient

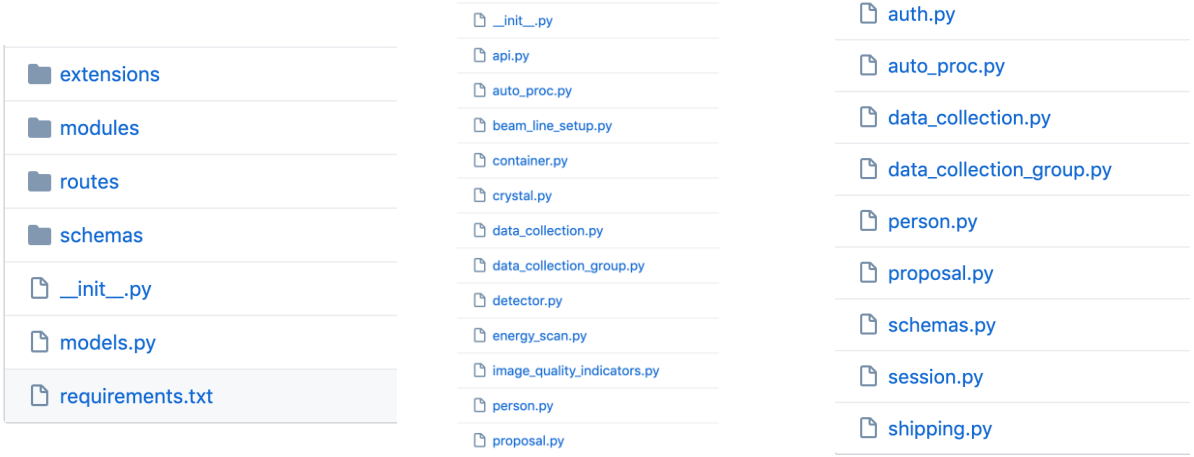


py-ispyb : structure

app	Added abstract base class for authentication	5 days ago
clients	Proposal routes with tests	15 days ago
deploy	code format	last month
docs	Fixed file header	24 days ago
flask_restx_patched	Cleaned imports	19 days ago
scripts	changed schemas names	15 days ago
tests	Proposal routes with tests	15 days ago
.coveragerc	added coveragerc	last month
.gitignore	renamed config.py to config-template.py	last month
.pylintrc	added ignored modules 2	last month
.travis.yml	renamed config.py to config-template.py	last month
CONTRIBUTING.md	added test	3 months ago
Dockerfile	move to modular structure	2 months ago
LICENSE.md	more tests	last month
README.md	readme	19 days ago
config-template.py	Added pagination limit property to config.py	16 days ago
enabled_db_modules.csv	fixed pylint	19 days ago
py_file_header.txt	Update py_file_header.txt	last month
requirements.txt	code format	last month
wsgi.py	Fixed file header	24 days ago

py-ispyb : structure

- system related **extensions**: api, auth, db, logging.
- **models.py**: autogenerated sqlalchemy classes.
- **schemas**: autogenerated flask, marshmallow and json schemas. Currently one file per db table. Join in one file?
- **modules**: data handling modules.
- **routes**: flask-restx resource classes defines end points.



py-ispyb : site specific configuration

- Site specific configuration defined in `config.py` (ignored in git). `config-template.py` available in github.
- Modules and routes dynamically loaded at the startup.
- No branching in the code. Lets avoid `isEMBL()`, `isESRF()`, etc.
- Site specific authentication module and class defined in the `config.py`
- `AbstractAuth` class provided to have a common interface.

config.py

```
AUTH_MODULE = "app.extensions.auth.DummyAuth"
AUTH_CLASS = "DummyAuth"
```

AbstractAuth.py

```
class AbstractAuth(object):

    __metaclass__ = abc.ABCMeta

    @abc.abstractmethod
    def get_roles(self, username, password):
        """Returns roles associated to the user
        Args:
            username (str): username
            password (str): password
```

DummyAuth.py

```
from app.extensions.auth.AbstractAuth import AbstractAuth


class DummyAuth(AbstractAuth):

    def get_roles(self, username, password):
        result = []
        if username.startswith("user"):
            result.append("user")
        if username.startswith("manager"):
            result.append("manager")
        return result
```

py-ispyb : security

- For authentication Json web tokens (jwt) are used.
- Secret key, coding algorithm and expiration time defined in the `config.py`.
- Route *ispyb/api/v1/auth/login* allows to request authentication token. Site specific code determinates user role.
- No role means the user is not authorized.
- Token passed in the Header as authorization “Bearer: Token”.
- For development master token configured in `config.py` is available.
- Python and Java script client examples are available.

- Authorization done by roles: user, manager and admin (more to add if needed).
- When a resource is requested token is used to identify role and provide data. For example: [GET] *ispyb/api/v1/proposals* for user returns proposals associated to the user and for manager all proposals are returned.

py-ispyb : security

- **@token_required** decorator checks if the token is valid. If it is valid then executes the method otherwise returns “Not authorized” message.
- **@write_permission_required** based on the token checks if the user has admin role and if it has then executes the method. Otherwise 401 Unauthorized is returned.

```
@api.expect(proposal_schemas.proposal_f_schema)
@api.marshal_with(proposal_schemas.proposal_f_schema, code=HTTPStatus.CREATED)
@token_required
@write_permission_required
def put(self, proposal_id):
    """Updates proposal with id proposal_id

    Args:
        proposal_id (int): corresponds to proposalId in db
    """
    log.info("Update proposal %d" % proposal_id)
    proposal.update_proposal(**api.payload)
```

py-ispyb : data models

- **models.py**: *flask-sqlacodegen* auto generated sqlalchemy classes.
- **schemas**: auto generated flask, marshmallow and json schemas generated with handwritten python script.
- Allows to use data bases with changes. For example CI with travis generates models.py and schemas.
- No handwritten models and schemas!

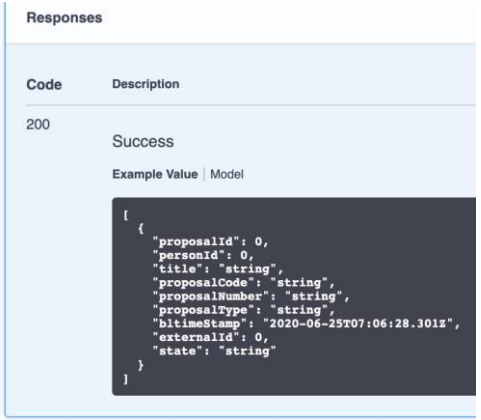
```
class Proposal(db.Model):
    __tablename__ = "Proposal"
    __table_args__ = (
        db.Index("Proposal_FKIndexCodeNumber", '
    )

    proposalId = db.Column(db.Integer, primary_l
    personId = db.Column(
        db.Integer, nullable=False, index=True,
    )
    title = db.Column(db.String(200, "utf8mb4_u
    proposalCode = db.Column(db.String(45))
    proposalNumber = db.Column(db.String(45))
```

```
class ProposalSchema(Schema):
    """Marshmallows schema class representing

    proposalId = ma_fields.Integer()
    personId = ma_fields.Integer()
    title = ma_fields.String()
    proposalCode = ma_fields.String()
    proposalNumber = ma_fields.String()
    proposalType = ma_fields.String()
    bltimeStamp = ma_fields.DateTime()
    externalId = ma_fields.Integer()
    state = ma_fields.String()
```

py-ispyb : input validation and response marshaling

- Input validation currently based on Flask schemas.
- The **@api.expect** decorator allows to specify the expected input fields.
- The **@api.marshal_with** decorator allows to specify response. For example dict with null values, empty dict or null. Preferably empty dict?
- Documentation in Swagger ui.

```python
@api.expect(proposal_schemas.proposal_f_schema)
@api.marshal_with(proposal_schemas.proposal_f_schema, code=HTTPStatus.CREATED)
@token_required
@write_permission_required
def put(self, proposal_id):
    """Updates proposal with id proposal_id

    Args:
        proposal_id (int): corresponds to proposalId in db
    """
    log.info("Update proposal %d" % proposal_id)
    proposal.update_proposal(**api.payload)
```



EMBL

py-ispyb : endpoint design demo

[GET]
http://localhost:5000/ispyb/api/v1/auth/login
http://localhost:5000/ispyb/api/v1/proposals
http://localhost:5000/ispyb/api/v1/proposals/5
http://localhost:5000/ispyb/api/v1/proposals/params?proposalType=MB
http://localhost:5000/ispyb/api/v1/schemas
[POST]
http://localhost:5000/ispyb/api/v1/proposals
[DELETE]
http://localhost:5000/ispyb/api/v1/proposals/5

One end point with various methods and query parameters provides needed functionality.

py-ispyb : endpoint design

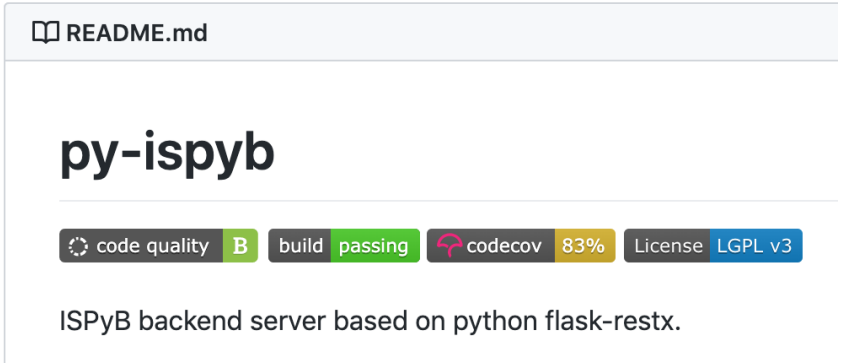
- Follow Rest API guidelines,
- Keep the amount of end-points limited.
- Clear, simple and short syntax (data_collection or dc to be decided).
- List and sort out existing end points, webservices used by exi(2) and synchweb and agree on a common api.

py-ispyb : documentation

- Google style doc strings also available in the swagger ui.
- Files contain header with LGPL3 license. `py_file_header.text`
- Info in git: license, contributing guidelines.
- Swagger ui: REST API documentation tool available at /ispyb/api/v1/doc. Disable in the production mode.

py-ispyb : Continues integration

- Script to format code with autopep and black.
- Code style and quality evaluated with pylint.
- Unit (checks the data models with mockup data) and functional tests (runs test instance and executes requests) with pytest.
- Travis for continues integration.
- At some point bump release and publish package with pypi.
- Codecov to generate coverage info.
- Build status icons in git.



Microservice architecture

- Although data base contains hundreds of tables there is no need to split data base in several data bases and provide distributed microservices.
- One could consider a new data base and a microservice for other methods (ssx, tomography, etc) or services (reprocessing, tomography construction, etc) if they do not fit in the current data base model and require more than x additional tables.
- With slight adjustments current prototype could be used to run ispyb as collection of several microservices: py-ispyb-core, py-ispyb-ssx, py-ispyb-tomo, etc.
- In the case of microservice architectura clearly defined api is even more important as in the single application case.

Road map

- User authorization based on roles.
- A generic HTTP Exception handling to avoid status code 500.
- Site specific authentication classes.
- Full end point definition.
- Release v1.0.0.

How to contribute and collaborate

- Checkout the git repository and run python3 wsgi.py.
- Check the contributing guidelines.
- Developers code camp.
- Use more agile methods: sprints, pair programming and review.
- Lets collaborate…

Summary

- py-ispyb contains main concepts of a web backend server (data models, input validation, response marshalling, authentication, authorization and others) and is sufficient for the scale of ispyb database and api.
- If necessary after slight adjustments prototype could be used to create a microservice architecture.
- As it is in a very early stage it is a good chance to gather opinions, needs and wishes and come up with a well defined api.
- It may serve as a good starting point for a new way of collaboration.

Thank you for your attention!