



| The European Synchrotron

A quick intro to FastAPI

Stuart FISHER
BCU / Acquisition

- Python
- Graylog?
- Versioning /api/v1
- JWT + One time tokens for resource streaming (file download)
- Extensible and modular framework (API grows quickly)
- Programatical database interactions (sqlalchemy)
 - Allows reuse of common functional units
 - Avoid string concatenation to create queries
 - Return entities
 - Editor column autocompletion based on models
- Automatic documentation (OpenAPI)
- Input / output marshalling (Validation)
- Schema for reuse on client side
- Reusable pagination
- Reusable query parameters (search, filter, etc)
- Follow REST best practices (re naming conventions, http query types (get, post, patch, put))
<https://docs.microsoft.com/en-us/azure/architecture/best-practices/api-design>
- Follow OWASP guidelines (https://cheatsheetseries.owasp.org/cheatsheets/REST_Security_Cheat_Sheet.html)
- Resources not tied to proposal / session, limits ability to create views for example for a beamline
- Use column names as cased in db, no renaming
- Use filters as defined by db naming, i.e. filter should be proteinId, not protein_id, or proteinid
- Inbuilt query debugging (query dump, query time, possibly explain)
- Config in env, options in database

Asynchronous framework for quickly building REST APIs

Marshalling (input / output validation)

Probably the most vital part of building a public API

Primary attack vector

Body, Query, Path

Documentation (OpenAPI v3)

JSON Schema

Consume by client (saves us time on front end dev)

Webservices (consume API without reading code)

Typed based routing

Dependency injection

fastapi

sqlalchemy

fast-restx

sqlalchemy

flask-sqlalchemy(optional)

marshmallow (removable)

marshmallow-jsonschema

flask-marshmallow

marshmallow-sqlalchemy (serialisation)

Type based routing

Helps type the rest of the project

Better code quality, fewer bugs, better dev environment

Dependency injection

Reusable parameters

Makes testing easier

Routing - flask-restx

```
@api.route("/<int:sample_id>", endpoint="sample_by_id")
@api.param("sample_id", "Sample id (integer)")
@api.doc(security="apikey")
@api.response(code=HTTPStatus.NOT_FOUND, description="Sample not found.")
class SampleById(Resource):
    """Allows to get/set/delete a sample item"""

    @authentication_required
    @authorization_required
    @api.doc(description="sample_id should be an integer ")
    @api.marshal_with(sample_schemas.f_schema, skip_none=False, code=HTTPStatus.OK)
    def get(self, sample_id):
        """Returns a sample by sampleId"""
        return sample.get_sample_by_id(sample_id)
```

```
@router.get(
   ("/{blSampleId}",
    response_model=schema.Sample,
    responses={404: {"description": "No such sample"}},
)

def get_sample(
    blSampleId: int = Depends(filters.blSampleId),
) -> models.BLSample:
    """Get a sample"""
    samples = crud.get_samples(blSampleId=blSampleId, skip=0, limit=1)
    try:
        return samples.first
    except IndexError:
        raise HTTPException(status_code=404, detail="Sample not found")
```

```
class AuthenticatedAPIRouter(BaseRouter):  
    def __init__(self, *args, **kwargs):  
        print("AuthenticatedAPIRouter")  
        super().__init__(*args, dependencies=[Depends(JWTBearer)], **kwargs)
```

```
router = AuthenticatedAPIRouter(prefix="/samples", tags=["Samples"])
```

```
class LegacyAPIRouter(BaseRouter):  
    def __init__(self, *args, **kwargs):  
        print("LegacyAPIRouter")  
        super().__init__(*args, dependencies=[Depends(token)], **kwargs)
```

```
router = LegacyAPIRouter(prefix="/legacy", tags=["Legacy"])
```



```
@router.get(
    "/",
    response_model=paginated(schema.Event),
    responses={404: {"description": "Entity not found"}},
)

def get_events(
    page: dict[str, int] = Depends(pagination),
    session: str = Depends(filters.session),
    dataCollectionGroupId: int = Depends(filters.dataCollectionGroupId),
    blSampleId: int = Depends(filters.blSampleId),
    proteinId: int = Depends(filters.proteinId),
) -> Paged[schema.Event]:
    """Get a list of events"""
    ...
```

```
def pagination(
    skip: Optional[int] = Query(0, description="Results to skip"),
    limit: Optional[int] = Query(25, description="Number of results to show"),
) -> dict[str, int]:
    assert skip is not None
    assert limit is not None

    return {"skip": skip, "limit": limit}

def blSampleId(
    blSampleId: Optional[int] = Query(None, description="Sample id to filter
by")
) -> Optional[int]:
    return blSampleId
```

GET /ispyb/api/v1/events/ Get Events

Get a list of events

Parameters

Try it out

Name	Description
skip integer (query)	Results to skip Default value : 0
	<input type="text" value="0"/>
limit integer (query)	Number of results to show Default value : 25
	<input type="text" value="25"/>
session string (query) pattern: ^\w+\d+-\d+\$	Session name to filter by
	<input type="text" value="session"/>
dataCollectionGroupId integer (query)	Data collection group id to filter by
	<input type="text" value="dataCollectionGroupId"/>
blSampleId integer (query)	Sample id to filter by
	<input type="text" value="blSampleId"/>
proteinId integer (query)	Protein id to filter by
	<input type="text" value="proteinId"/>

GET

/ispyb/api/v1/legacy/{token}/session/list

Get Sessions

^

Get all sessions that user is allowed to access.

Parameters

Try it out

Name	Description
token <small>★ required</small> string (path)	<input type="text" value="token"/>

Responses

Code	Description	Links
200	<p>Successful Response</p> <p>Media type <input type="text" value="application/json"/> ▼ <small>Controls Accept header.</small></p> <p>Example Value Schema</p> <pre>"string"</pre>	No links

FastAPI automatically serialises db results

Don't need an additional serializer (marshmallow)

Uses pydantic

Based on dataclasses (python 3.7+)

Define properties with standard python typing

With validation

Easy to nest models

Adapt Ivars' script to create these automatically from db structure

```
@dataclass
class Sample
    id: int
    name: str
    size: int

sample_dict = { "id": 1, "name": "test1", "size": 1 }

sample = Sample(**sample_dict)

sample.id
sample.name
```

No more dictionary syntax blah["key"]

```
class SampleBase(BaseModel):  
    name: str  
    comments: Optional[str] = Field(title="Comments", nullable=True)
```

```
class Sample(SampleBase):  
    blSampleId: int  
  
    Crystal: Crystal  
    metadata: SampleMetaData = Field(alias="_metadata")
```

```
class Config:  
    orm_mode = True
```

Modelling

Code

Description

Links

200

Successful Response

No links

Media type

application/json

Controls Accept header.

Example Value | Schema

```
{
  "total": 0,
  "results": [
    {
      "name": "string",
      "comments": "Unknown Type: null,string",
      "_metadata": {
        "subsamples": 0,
        "datacollections": 0
      },
      "blSampleId": 0,
      "Crystal": {
        "cell_a": "Unknown Type: null,number",
        "cell_b": "Unknown Type: null,number",
        "cell_c": "Unknown Type: null,number",
        "cell_alpha": "Unknown Type: null,number",
        "cell_beta": "Unknown Type: null,number",
        "cell_gamma": "Unknown Type: null,number",
        "Protein": {
          "name": "string",
          "acronym": "string",
          "proteinId": 0
        },
        "crystalId": 0
      }
    }
  ]
}
```



```
@router.post(
    "/",
    response_model=schema.LabContact,
    status_code=status.HTTP_201_CREATED,
)
def create_lab_contact(
    labcontact: schema.LabContactCreate
) -> models.LabContact:
    """Create a new lab contact"""
    return crud.create_labcontact(labcontact=labcontact)
```

POST /ispyb/api/v1/labcontacts/ Create Lab Contact

Create a new lab contact

Parameters Try it out

No parameters

Request body required application/json

Example Value Schema

```
{
  "proposalId": 0,
  "cardName": "string",
  "defaultCourierCompany": "Unknown Type: null,string",
  "courierAccount": "Unknown Type: null,string",
  "billingReference": "Unknown Type: null,string",
  "dewarAvgCustomsValue": 0,
  "dewarAvgTransportValue": 0,
  "Person": {
    "givenName": "string",
    "familyName": "string",
    "emailAddress": "Unknown Type: null,string",
    "phoneNumber": "Unknown Type: null,string",
    "Laboratory": {
      "name": "string",
      "address": "string",
      "city": "string",
      "postcode": "string",
      "country": "string"
    }
  }
}
```

Modelling

Request body **required**

application/json

Example Value | Schema

```
LabContactCreate {
  proposalId* integer
    title: Proposalid
  cardName* string
    title: Card Name

  The name for this lab contact

  defaultCourierCompany nullstring
    title: Courier Company
    nullable: true
  courierAccount nullstring
    title: Account No.
    nullable: true
  billingReference nullstring
    title: Billing Reference
    nullable: true
  dewarAvgCustomsValue integer
    title: Avg Customs Value
    unit: Eur
  dewarAvgTransportValue integer
    title: Avg Transport Value
    unit: Eur

  Person* Contact Person {
    givenName* string
      title: First Name
    familyName* string
      title: Surname
    emailAddress nullstring
      title: Email Address
      nullable: true
    phoneNumber nullstring
      title: Phone Number
      nullable: true
    Laboratory Laboratory {
      name* string
        title: Laboratory Name
        pattern: ^([\\w\\s-])+${
      address* string
        title: Address
      city* string
        title: City
      postcode* string
        title: Post Code
      country* string
        title: Country
    }
  }
}
```

Client Improvements

<https://www.npmjs.com/package/json-schema-to-typescript>

```
export interface Sample {  
  name: Name;  
  comments?: Comments;  
  _metadata: SampleMetadata;  
  blSampleId: Blsampleid;  
  Crystal: Crystal;  
}  
  
export interface SampleMetadata {  
  subsamples: Subsamples;  
  datacollections: Datacollections;  
}
```

```
export interface Crystal {  
  cell_a?: CellA;  
  cell_b?: CellB;  
  cell_c?: CellC;  
  cell_alpha?: CellAlpha;  
  cell_beta?: CellBeta;  
  cell_gamma?: CellGamma;  
  Protein: Protein;  
  crystalId: Crystalid;  
}  
  
export interface Protein {  
  name: Name1;  
  acronym: Acronym;  
  proteinId: Proteinid;  
}
```

Client Improvements

```
export default function CreateLabContact() {  
  const schema = useSchema('LabContactCreate', 'Create Lab Contact');  
  const uiSchema = {  
    proposalId: { classNames: 'hidden-row', 'ui:widget': 'hidden' },  
  };  
  ...  
  return (  
    <Form  
      liveValidate  
      schema={schema} uiSchema={uiSchema} onSubmit={onSubmit} formData={formData}  
    />  
  );  
}
```

Client Improvements

Create Lab Contact

Card Name*

The name for this lab contact

is a required property

Courrier Company

Account No.

Billing Reference

Avg Customs Value

Avg Transport Value

Contact Person

First Name*

is a required property

Surname*

is a required property

Email Address

Maintainability

Modern framework

Simple dependency tree

Building a new API -> Using python standards

***Working in async context**

Existing project basically migrated

Some more tidying to do

Config handling? .env vs. yml

Auth refactor

Check / fix keycloak

Does not exclude any of Mael's existing work