FastAPI Contrib Documentation

Release 0.2.11

Lev Rubel

Jun 03, 2021

CONTENTS:

1 FastAPI Contrib **1**
- 1.1 Features . . . 1
- 1.2 Roadmap . . . 1
- 1.3 Installation . . . 2
- 1.4 Usage . . . 2
- 1.5 Auto-creation of MongoDB indexes . . . 6
- 1.6 Credits . . . 7

2 Installation **9**
- 2.1 Stable release . . . 9
- 2.2 From sources . . . 9

3 Auto-creation of MongoDB indexes **15**

4 fastapi_contrib **17**
- 4.1 fastapi_contrib package . . . 17

5 Contributing **37**
- 5.1 Types of Contributions . . . 37
- 5.2 Get Started! . . . 38
- 5.3 Pull Request Guidelines . . . 39
- 5.4 Tips . . . 39
- 5.5 Deploying . . . 39

6 Credits **41**
- 6.1 Development Lead . . . 41
- 6.2 Contributors . . . 41

7 Indices and tables **43**

Python Module Index **45**

Index **47**

CHAPTER

ONE

FASTAPI CONTRIB

Opinionated set of utilities on top of FastAPI

- Free software: MIT license
- Documentation: https://fastapi-contrib.readthedocs.io.

1.1 Features

- Auth Backend & Middleware (User or None in every request object)
- Permissions: reusable class permissions, specify multiple as FastAPI Dependency
- ModelSerializers: serialize (pydantic) incoming request, connect data with DB model and save
- UJSONResponse: correctly show slashes in fields with URLs
- Limit-Offset Pagination: use it as FastAPI Dependency (works only with ModelSerializers for now)
- MongoDB integration: Use models as if it was Django (based on pydantic models)
- MongoDB indices verification on startup of the app
- Custom Exceptions and Custom Exception Handlers
- Opentracing middleware & setup utility with Jaeger tracer + root span available in every Request's state
- StateRequestIDMiddleware: receives configurable header and saves it in request state

1.2 Roadmap

See GitHub Project Roadmap.

1.3 Installation

To install just Contrib (without mongodb, pytz, ujson):

```
$ pip install fastapi_contrib
```

To install contrib with mongodb support:

```
$ pip install fastapi_contrib[mongo]
```

To install contrib with ujson support:

```
$ pip install fastapi_contrib[ujson]
```

To install contrib with pytz support:

```
$ pip install fastapi_contrib[pytz]
```

To install contrib with opentracing & Jaeger tracer:

```
$ pip install fastapi_contrib[jaegertracing]
```

To install everything:

```
$ pip install fastapi_contrib[all]
```

1.4 Usage

To use Limit-Offset pagination:

```
from fastapi import FastAPI
from fastapi_contrib.pagination import Pagination
from fastapi_contrib.serializers.common import ModelSerializer
from yourapp.models import SomeModel

app = FastAPI()

class SomeSerializer(ModelSerializer):
    class Meta:
        model = SomeModel

@app.get("/")
async def list(pagination: Pagination = Depends()):
    filter_kwargs = {}
    return await pagination.paginate(
        serializer_class=SomeSerializer, **filter_kwargs
    )
```

Subclass this pagination to define custom default & maximum values for offset & limit:

```
class CustomPagination(Pagination):
    default_offset = 90
```

(continues on next page)

(continued from previous page)

```
    default_limit = 1
    max_offset = 100
    max_limit = 2000
```

To use State Request ID Middleware:

```
from fastapi import FastAPI
from fastapi_contrib.common.middlewares import StateRequestIDMiddleware

app = FastAPI()

@app.on_event('startup')
async def startup():
    app.add_middleware(StateRequestIDMiddleware)
```

To use Authentication Middleware:

```
from fastapi import FastAPI
from fastapi_contrib.auth.backends import AuthBackend
from fastapi_contrib.auth.middlewares import AuthenticationMiddleware

app = FastAPI()

@app.on_event('startup')
async def startup():
    app.add_middleware(AuthenticationMiddleware, backend=AuthBackend())
```

Define & use custom permissions based on FastAPI Dependency framework:

```
from fastapi import FastAPI
from fastapi_contrib.permissions import BasePermission, PermissionsDependency

class TeapotUserAgentPermission(BasePermission):

    def has_required_permissions(self, request: Request) -> bool:
        return request.headers.get('User-Agent') == "Teapot v1.0"

app = FastAPI()

@app.get(
    "/teapot/",
    dependencies=[Depends(
        PermissionsDependency([TeapotUserAgentPermission]))]
)
async def teapot() -> dict:
    return {"teapot": True}
```

Setup uniform exception-handling:

```
from fastapi import FastAPI
from fastapi_contrib.exception_handlers import setup_exception_handlers

app = FastAPI()
```

(continues on next page)

(continued from previous page)

```
@app.on_event('startup')
async def startup():
    setup_exception_handlers(app)
```

If you want to correctly handle scenario when request is an empty body (IMPORTANT: non-multipart):

```
from fastapi import FastAPI
from fastapi_contrib.routes import ValidationErrorLoggingRoute

app = FastAPI()
app.router.route_class = ValidationErrorLoggingRoute
```

Or if you use multiple routes for handling different namespaces (IMPORTANT: non-multipart):

```
from fastapi import APIRouter, FastAPI
from fastapi_contrib.routes import ValidationErrorLoggingRoute

app = FastAPI()

my_router = APIRouter(route_class=ValidationErrorLoggingRoute)
```

To correctly show slashes in fields with URLs + ascii locking:

```
from fastapi import FastAPI
from fastapi_contrib.common.responses import UJSONResponse

app = FastAPI()

@app.get("/", response_class=UJSONResponse)
async def root():
    return {"a": "b"}
```

Or specify it as default response class for the whole app (FastAPI >= 0.39.0):

```
from fastapi import FastAPI
from fastapi_contrib.common.responses import UJSONResponse

app = FastAPI(default_response_class=UJSONResponse)
```

To setup Jaeger tracer and enable Middleware that captures every request in opentracing span:

```
from fastapi import FastAPI
from fastapi_contrib.tracing.middlewares import OpentracingMiddleware
from fastapi_contrib.tracing.utils import setup_opentracing

app = FastAPI()

@app.on_event('startup')
async def startup():
    setup_opentracing(app)
    app.add_middleware(OpentracingMiddleware)
```

To setup mongodb connection at startup and never worry about it again:

```
from fastapi import FastAPI
from fastapi_contrib.db.utils import setup_mongodb

app = FastAPI()

@app.on_event('startup')
async def startup():
    setup_mongodb(app)
```

Use models to map data to MongoDB:

```
from fastapi_contrib.db.models import MongoDBModel

class MyModel(MongoDBModel):
    additional_field1: str
    optional_field2: int = 42

    class Meta:
        collection = "mymodel_collection"


mymodel = MyModel(additional_field1="value")
mymodel.save()

assert mymodel.additional_field1 == "value"
assert mymodel.optional_field2 == 42
assert isinstance(mymodel.id, int)
```

Or use TimeStamped model with creation datetime:

```
from fastapi_contrib.db.models import MongoDBTimeStampedModel

class MyTimeStampedModel(MongoDBTimeStampedModel):

    class Meta:
        collection = "timestamped_collection"


mymodel = MyTimeStampedModel()
mymodel.save()

assert isinstance(mymodel.id, int)
assert isinstance(mymodel.created, datetime)
```

Use serializers and their response models to correctly show Schemas and convert from JSON/dict to models and back:

```
from fastapi import FastAPI
from fastapi_contrib.db.models import MongoDBModel
from fastapi_contrib.serializers import openapi
from fastapi_contrib.serializers.common import Serializer

from yourapp.models import SomeModel
```

(continues on next page)

(continued from previous page)

```
app = FastAPI()


class SomeModel(MongoDBModel):
    field1: str


@openapi.patch
class SomeSerializer(Serializer):
    read_only1: str = "const"
    write_only2: int
    not_visible: str = "42"

    class Meta:
        model = SomeModel
        exclude = {"not_visible"}
        write_only_fields = {"write_only2"}
        read_only_fields = {"read_only1"}


@app.get("/", response_model=SomeSerializer.response_model)
async def root(serializer: SomeSerializer):
    model_instance = await serializer.save()
    return model_instance.dict()
```

POST-ing to this route following JSON:

```
{"read_only1": "a", "write_only2": 123, "field1": "b"}
```

Should return following response:

```
{"id": 1, "field1": "b", "read_only1": "const"}
```

1.5 Auto-creation of MongoDB indexes

Suppose we have this directory structure:

```
-- project_root/
    -- apps/
        -- app1/
            -- models.py (with MongoDBModel inside with indices declared)
        -- app2/
            -- models.py (with MongoDBModel inside with indices declared)
```

Based on this, your name of the folder with all the apps would be "apps". This is the default name for fastapi_contrib package to pick up your structure automatically. You can change that by setting ENV variable *CONTRIB_APPS_FOLDER_NAME* (by the way, all the setting of this package are overridable via ENV vars with *CONTRIB_* prefix before them).

You also need to tell fastapi_contrib which apps to look into for your models. This is controlled by *CONTRIB_APPS* ENV variable, which is list of str names of the apps with models. In the example above, this would be *CONTRIB_APPS=["app1","app2"]*.

Just use create_indexes function after setting up mongodb:

```
from fastapi import FastAPI
from fastapi_contrib.db.utils import setup_mongodb, create_indexes

app = FastAPI()

@app.on_event("startup")
async def startup():
    setup_mongodb(app)
    await create_indexes()
```

This will scan all the specified *CONTRIB_APPS* in the *CONTRIB_APPS_FOLDER_NAME* for models, that are subclassed from either MongoDBModel or MongoDBTimeStampedModel and create indices for any of them that has Meta class with indexes attribute:

models.py:

```
import pymongo
from fastapi_contrib.db.models import MongoDBTimeStampedModel


class MyModel(MongoDBTimeStampedModel):

    class Meta:
        collection = "mymodel"
        indexes = [
            pymongo.IndexModel(...),
            pymongo.IndexModel(...),
        ]
```

This would not create duplicate indices because it relies on pymongo and motor to do all the job.

1.6 Credits

This package was created with Cookiecutter and the audreyr/cookiecutter-pypackage project template.

CHAPTER

TWO

INSTALLATION

2.1 Stable release

To install just Contrib (without mongodb, pytz, ujson):

```
$ pip install fastapi_contrib
```

To install contrib with mongodb support:

```
$ pip install fastapi_contrib[mongo]
```

To install contrib with ujson support:

```
$ pip install fastapi_contrib[ujson]
```

To install contrib with pytz support:

```
$ pip install fastapi_contrib[pytz]
```

To install contrib with opentracing & Jaeger tracer:

```
$ pip install fastapi_contrib[jaegertracing]
```

To install everything:

```
$ pip install fastapi_contrib[all]
```

This is the preferred method to install FastAPI Contrib, as it will always install the most recent stable release.

If you don't have pip installed, this Python installation guide can guide you through the process.

2.2 From sources

The sources for FastAPI Contrib can be downloaded from the Github repo.

You can either clone the public repository:

```
$ git clone git://github.com/l@datacorp.ee/fastapi_contrib
```

Or download the tarball:

```
$ curl  -OL https://github.com/l@datacorp.ee/fastapi_contrib/tarball/master
```

Once you have a copy of the source, you can install it with:

```
$ python setup.py install
```

To use Limit-Offset pagination:

```
from fastapi import FastAPI
from fastapi_contrib.pagination import Pagination
from fastapi_contrib.serializers.common import ModelSerializer
from yourapp.models import SomeModel

app = FastAPI()

class SomeSerializer(ModelSerializer):
    class Meta:
        model = SomeModel

@app.get("/")
async def list(pagination: Pagination = Depends()):
    filter_kwargs = {}
    return await pagination.paginate(
        serializer_class=SomeSerializer, **filter_kwargs
    )
```

Subclass this pagination to define custom default & maximum values for offset & limit:

```
class CustomPagination(Pagination):
    default_offset = 90
    default_limit = 1
    max_offset = 100
    max_limit = 2000
```

To use State Request ID Middleware:

```
from fastapi import FastAPI
from fastapi_contrib.common.middlewares import StateRequestIDMiddleware

app = FastAPI()

@app.on_event('startup')
async def startup():
    app.add_middleware(StateRequestIDMiddleware)
```

To use Authentication Middleware:

```
from fastapi import FastAPI
from fastapi_contrib.auth.backends import AuthBackend
from fastapi_contrib.auth.middlewares import AuthenticationMiddleware

app = FastAPI()

@app.on_event('startup')
```

(continues on next page)

(continued from previous page)

```
async def startup():
    app.add_middleware(AuthenticationMiddleware, backend=AuthBackend())
```

Define & use custom permissions based on FastAPI Dependency framework:

```
from fastapi import FastAPI
from fastapi_contrib.permissions import BasePermission, PermissionsDependency

class TeapotUserAgentPermission(BasePermission):

    def has_required_permissions(self, request: Request) -> bool:
        return request.headers.get('User-Agent') == "Teapot v1.0"

app = FastAPI()

@app.get(
    "/teapot/",
    dependencies=[Depends(
        PermissionsDependency([TeapotUserAgentPermission]))]
)
async def teapot() -> dict:
    return {"teapot": True}
```

Setup uniform exception-handling:

```
from fastapi import FastAPI
from fastapi_contrib.exception_handlers import setup_exception_handlers

app = FastAPI()

@app.on_event('startup')
async def startup():
    setup_exception_handlers(app)
```

If you want to correctly handle scenario when request is an empty body (IMPORTANT: non-multipart):

```
from fastapi import FastAPI
from fastapi_contrib.routes import ValidationErrorLoggingRoute

app = FastAPI()
app.router.route_class = ValidationErrorLoggingRoute
```

Or if you use multiple routes for handling different namespaces (IMPORTANT: non-multipart):

```
from fastapi import APIRouter, FastAPI
from fastapi_contrib.routes import ValidationErrorLoggingRoute

app = FastAPI()

my_router = APIRouter(route_class=ValidationErrorLoggingRoute)
```

To correctly show slashes in fields with URLs + ascii locking:

```
from fastapi import FastAPI
from fastapi_contrib.common.responses import UJSONResponse

app = FastAPI()

@app.get("/", response_class=UJSONResponse)
async def root():
    return {"a": "b"}
```

Or specify it as default response class for the whole app (FastAPI >= 0.39.0):

```
from fastapi import FastAPI
from fastapi_contrib.common.responses import UJSONResponse

app = FastAPI(default_response_class=UJSONResponse)
```

To setup Jaeger tracer and enable Middleware that captures every request in opentracing span:

```
from fastapi import FastAPI
from fastapi_contrib.tracing.middlewares import OpentracingMiddleware

app = FastAPI()

@app.on_event('startup')
async def startup():
    setup_opentracing(app)
    app.add_middleware(AuthenticationMiddleware)
```

To setup mongodb connection at startup and never worry about it again:

```
from fastapi import FastAPI
from fastapi_contrib.db.utils import setup_mongodb

app = FastAPI()

@app.on_event('startup')
async def startup():
    setup_mongodb(app)
```

Use models to map data to MongoDB:

```
from fastapi_contrib.db.models import MongoDBModel

class MyModel(MongoDBModel):
    additional_field1: str
    optional_field2: int = 42

    class Meta:
        collection = "mymodel_collection"


mymodel = MyModel(additional_field1="value")
mymodel.save()
```

(continues on next page)

(continued from previous page)

```
assert mymodel.additional_field1 == "value"
assert mymodel.optional_field2 == 42
assert isinstance(mymodel.id, int)
```

Or use TimeStamped model with creation datetime:

```
from fastapi_contrib.db.models import MongoDBTimeStampedModel

class MyTimeStampedModel(MongoDBTimeStampedModel):

    class Meta:
        collection = "timestamped_collection"


mymodel = MyTimeStampedModel()
mymodel.save()

assert isinstance(mymodel.id, int)
assert isinstance(mymodel.created, datetime)
```

Use serializers and their response models to correctly show Schemas and convert from JSON/dict to models and back:

```
from fastapi import FastAPI
from fastapi_contrib.db.models import MongoDBModel
from fastapi_contrib.serializers import openapi
from fastapi_contrib.serializers.common import Serializer

from yourapp.models import SomeModel

app = FastAPI()


class SomeModel(MongoDBModel):
    field1: str


@openapi.patch
class SomeSerializer(Serializer):
    read_only1: str = "const"
    write_only2: int
    not_visible: str = "42"

    class Meta:
        model = SomeModel
        exclude = {"not_visible"}
        write_only_fields = {"write_only2"}
        read_only_fields = {"read_only1"}


@app.get("/", response_model=SomeSerializer.response_model)
async def root(serializer: SomeSerializer):
```

(continues on next page)

(continued from previous page)

```
    model_instance = await serializer.save()
    return model_instance.dict()
```

POST-ing to this route following JSON:

```
{"read_only1": "a", "write_only2": 123, "field1": "b"}
```

Should return following response:

```
{"id": 1, "field1": "b", "read_only1": "const"}
```

CHAPTER

THREE

AUTO-CREATION OF MONGODB INDEXES

Suppose we have this directory structure:

```
-- project_root/
     -- apps/
          -- app1/
               -- models.py (with MongoDBModel inside with indices declared)
          -- app2/
               -- models.py (with MongoDBModel inside with indices declared)
```

Based on this, your name of the folder with all the apps would be "apps". This is the default name for fastapi_contrib package to pick up your structure automatically. You can change that by setting ENV variable *CONTRIB_APPS_FOLDER_NAME* (by the way, all the setting of this package are overridable via ENV vars with *CONTRIB_* prefix before them).

You also need to tell fastapi_contrib which apps to look into for your models. This is controlled by *CONTRIB_APPS* ENV variable, which is list of str names of the apps with models. In the example above, this would be *CONTRIB_APPS=["app1","app2"]*.

Just use create_indexes function after setting up mongodb:

```
from fastapi import FastAPI
from fastapi_contrib.db.utils import setup_mongodb, create_indexes

app = FastAPI()

@app.on_event("startup")
async def startup():
    setup_mongodb(app)
    await create_indexes()
```

This will scan all the specified *CONTRIB_APPS* in the *CONTRIB_APPS_FOLDER_NAME* for models, that are subclassed from either MongoDBModel or MongoDBTimeStampedModel and create indices for any of them that has Meta class with indexes attribute:

models.py:

```
import pymongo
from fastapi_contrib.db.models import MongoDBTimeStampedModel


class MyModel(MongoDBTimeStampedModel):

```

(continues on next page)

(continued from previous page)

```
    class Meta:
        collection = "mymodel"
        indexes = [
            pymongo.IndexModel(...),
            pymongo.IndexModel(...),
        ]
```

This would not create duplicate indices because it relies on pymongo and motor to do all the job.

CHAPTER

FOUR

FASTAPI_CONTRIB

4.1 fastapi_contrib package

4.1.1 Subpackages

fastapi_contrib.auth package

Submodules

fastapi_contrib.auth.backends module

class fastapi_contrib.auth.backends.**AuthBackend**

Bases: starlette.authentication.AuthenticationBackend

Own Auth Backend based on Starlette's AuthenticationBackend.

Use instance of this class as *backend* argument to *add_middleware* func:

```
app = FastAPI()

@app.on_event('startup')
async def startup():
    app.add_middleware(AuthenticationMiddleware, backend=AuthBackend())
```

async authenticate(*conn: starlette.requests.HTTPConnection*) → Tuple[bool, Optional[*fastapi_contrib.auth.models.User*]]

Main function that AuthenticationMiddleware uses from this backend. Should return whether request is authenticated based on credentials and if it was, return also user instance.

Parameters **conn** – HTTPConnection of the current request-response cycle

Returns 2-tuple: is authenticated & user instance if exists

fastapi_contrib.auth.middlewares module

class fastapi_contrib.auth.middlewares.**AuthenticationMiddleware**(*app: Callable[[MutableMapping[str, Any], Callable[], Awaitable[MutableMapping[str, Any]]], Callable[[MutableMapping[str, Any]], Awaitable[None]]], Awaitable[None]], backend: starlette.authentication.AuthenticationBackend, on_error: Optional[Callable[[starlette.requests.HTTPConnectio starlette.authentication.AuthenticationError], starlette.responses.Response]] = None*)

Bases: starlette.middleware.authentication.AuthenticationMiddleware

Own Authentication Middleware based on Starlette's default one.

Use instance of this class as a first argument to *add_middleware* func:

```
app = FastAPI()

@app.on_event('startup')
async def startup():
    app.add_middleware(AuthenticationMiddleware, backend=AuthBackend())
```

static default_on_error(*conn: starlette.requests.HTTPConnection*, *exc: Exception*) → *fastapi_contrib.common.responses.UJSONResponse*

Overriden method just to make sure we return response in our format.

Parameters

- **conn** – HTTPConnection of the current request-response cycle
- **exc** – Any exception that could have been raised

Returns UJSONResponse with error data as dict and 403 status code

fastapi_contrib.auth.models module

class fastapi_contrib.auth.models.**Token**(**, id: int = None, created: datetime.datetime = None, key: fastapi_contrib.auth.models.ConstrainedStrValue = None, user_id: int = None, expires: datetime.datetime = None, is_active: bool = True*)

Bases: *fastapi_contrib.db.models.MongoDBTimeStampedModel*

Default Token model with several fields implemented as a default:

- id - inherited from *MongoDBTimeStampedModel*
- created - inherited from *MongoDBTimeStampedModel*
- key - string against which user will be authenticated

- user_id - id of *User*, who owns this token
- expires - datetime when this token no longer active
- is_active - defines whether this token can be used

class Meta
Bases: object

collection = 'tokens'

indexes = [<pymongo.operations.IndexModel object>]

expires: datetime.datetime

is_active: bool

key: fastapi_contrib.auth.models.ConstrainedStrValue

classmethod set_key(*v*, *values*, ***kwargs*) → str
If key is supplied (ex. from DB) then use it, otherwise generate new.

user_id: int

class fastapi_contrib.auth.models.**User**(*, *id: int = None*, *created: datetime.datetime = None*, *username: str*)
Bases: *fastapi_contrib.db.models.MongoDBTimeStampedModel*

Default User model that has only *username* field on top of default (id, created) pair from *MongoDBTimeStamped-Model*

class Meta
Bases: object

collection = 'users'

username: str

fastapi_contrib.auth.permissions module

class fastapi_contrib.auth.permissions.**IsAuthenticated**(*request: starlette.requests.Request*)
Bases: *fastapi_contrib.permissions.BasePermission*

Permission that checks if the user has been authenticated (by middleware)

Use it as an argument to *PermissionsDependency* as follows:

```
app = FastAPI()

@app.get(
    "/user/",
    dependencies=[Depends(PermissionsDependency([IsAuthenticated]))]
)
async def user(request: Request) -> dict:
    return request.scope["user"].dict()
```

error_code = 401

error_msg = 'Not authenticated.'

has_required_permissions(*request: starlette.requests.Request*) → bool

status_code = 401

fastapi_contrib.auth.serializers module

class fastapi_contrib.auth.serializers.**TokenSerializer**

Bases: *fastapi_contrib.serializers.common.ModelSerializer*

Serializer for the default *Token* model. Use it if you use default model.

class Meta

Bases: object

exclude = {'user_id'}

model

alias of *fastapi_contrib.auth.models.Token*

Module contents

fastapi_contrib.common package

Submodules

fastapi_contrib.common.middlewares module

class fastapi_contrib.common.middlewares.**StateRequestIDMiddleware**(*app: Callable[[MutableMapping[str, Any], Callable[], Awaitable[MutableMapping[str, Any]]], Callable[[MutableMapping[str, Any]], Awaitable[None]]], Awaitable[None]], dispatch: Optional[Callable[[starlette.requests.Request, Callable[[starlette.requests.Request], Awaitable[starlette.responses.Response]]], Awaitable[starlette.responses.Response]]] = None*)

Bases: starlette.middleware.base.BaseHTTPMiddleware

Middleware to store Request ID headers value inside request's state object.

Use this class as a first argument to *add_middleware* func:

```
app = FastAPI()

@app.on_event('startup')
async def startup():
    app.add_middleware(StateRequestIDMiddleware)
```

async dispatch(*request: starlette.requests.Request*, *call_next: Any*) → starlette.responses.Response

Get header from request and save it in request's state for future use. :param request: current Request instance :param call_next: next callable in list :return: response

property request_id_header_name: str
Gets the name of Request ID header from the project settings. :return: string with Request ID header name

fastapi_contrib.common.responses module

class fastapi_contrib.common.responses.**UJSONResponse**(*content: Optional[Any] = None*, *status_code: int = 200*, *headers: Optional[dict] = None*, *media_type: Optional[str] = None*, *background: Optional[starlette.background.BackgroundTask] = None*)

Bases: starlette.responses.JSONResponse

Custom Response, based on default UJSONResponse, but with differences:

- Allows to have forward slashes inside strings of JSON
- Limits output to ASCII and escapes all extended characters above 127.

Should be used as *response_class* argument to routes of your app:

```
app = FastAPI()


@app.get("/", response_class=UJSONResponse)
async def root():
    return {"a": "b"}
```

render(*content: Any*) → bytes

fastapi_contrib.common.utils module

fastapi_contrib.common.utils.**async_timing**(*func*)
Decorator for logging timing of async functions. Used in this library internally for tracking DB functions performance.

Parameters **func** – function to be decorated

Returns wrapped function

fastapi_contrib.common.utils.**get_current_app**() → fastapi.applications.FastAPI
Retrieves FastAPI app instance from the path, specified in project's conf. :return: FastAPI app

fastapi_contrib.common.utils.**get_logger**() → Any
Gets logger that will be used throughout this whole library. First it finds and imports the logger, then if it can be configured using loguru-compatible config, it does so.

Returns desired logger (pre-configured if loguru)

fastapi_contrib.common.utils.**get_now**() → datetime.datetime
Retrieves *now* function from the path, specified in project's conf. :return: datetime of "now"

fastapi_contrib.common.utils.**get_timezone**()
Retrieves timezone name from settings and tries to create tzinfo from it. :return: tzinfo object

fastapi_contrib.common.utils.**resolve_dotted_path**(*path: str*) → Any
Retrieves attribute (var, function, class, etc.) from module by dotted path

```
from datetime.datetime import utcnow as default_utcnow
utcnow = resolve_dotted_path('datetime.datetime.utcnow')
assert utcnow == default_utcnow
```

Parameters **path** – dotted path to the attribute in module

Returns desired attribute or None

Module contents

fastapi_contrib.db package

Submodules

fastapi_contrib.db.client module

class fastapi_contrib.db.client.**MongoDBClient**

Bases: object

Singleton client for interacting with MongoDB. Operates mostly using models, specified when making DB queries.

Implements only part of internal *motor* methods, but can be populated more

Please don't use it directly, use *fastapi_contrib.db.utils.get_db_client*.

async count(*model:* fastapi_contrib.db.models.MongoDBModel, *session: Optional[pymongo.client_session.ClientSession] = None*, ***kwargs*) → int

async delete(*model:* fastapi_contrib.db.models.MongoDBModel, *session: Optional[pymongo.client_session.ClientSession] = None*, ***kwargs*) → pymongo.results.DeleteResult

async get(*model:* fastapi_contrib.db.models.MongoDBModel, *session: Optional[pymongo.client_session.ClientSession] = None*, ***kwargs*) → dict

get_collection(*collection_name: str*) → pymongo.collection.Collection

async insert(*model:* fastapi_contrib.db.models.MongoDBModel, *session: Optional[pymongo.client_session.ClientSession] = None*, *include=None*, *exclude=None*) → pymongo.results.InsertOneResult

list(*model:* fastapi_contrib.db.models.MongoDBModel, *session: Optional[pymongo.client_session.ClientSession] = None*, *_offset: int = 0*, *_limit: int = 0*, *_sort: Optional[list] = None*, ***kwargs*) → pymongo.cursor.Cursor

async update_many(*model:* fastapi_contrib.db.models.MongoDBModel, *filter_kwargs: dict*, *session: Optional[pymongo.client_session.ClientSession] = None*, ***kwargs*) → pymongo.results.UpdateResult

async update_one(*model:* fastapi_contrib.db.models.MongoDBModel, *filter_kwargs: dict*, *session: Optional[pymongo.client_session.ClientSession] = None*, ***kwargs*) → pymongo.results.UpdateResult

fastapi_contrib.db.models module

class fastapi_contrib.db.models.**MongoDBModel**(*, *id: int = None*)

Bases: pydantic.main.BaseModel

Base Model to use for any information saving in MongoDB. Provides *id* field as a base, populated by id-generator. Use it as follows:

```
class MyModel(MongoDBModel):
    additional_field1: str
    optional_field2: int = 42

    class Meta:
        collection = "mymodel_collection"


mymodel = MyModel(additional_field1="value")
mymodel.save()

assert mymodel.additional_field1 == "value"
assert mymodel.optional_field2 == 42
assert isinstance(mymodel.id, int)
```

class Config

Bases: object

anystr_strip_whitespace = True

async classmethod count(***kwargs*) → int

async classmethod create_indexes() → Optional[List[str]]

async classmethod delete(***kwargs*) → pymongo.results.DeleteResult

async classmethod get(***kwargs*) → Optional[*fastapi_contrib.db.models.MongoDBModel*]

classmethod get_db_collection() → str

id: int

async classmethod list(*raw=True*, *_limit=0*, *_offset=0*, *_sort=None*, ***kwargs*)

async save(*include: set = None*, *exclude: set = None*, *rewrite_fields: dict = None*) → int

classmethod set_id(*v*, *values*, ***kwargs*) → int

If id is supplied (ex. from DB) then use it, otherwise generate new.

async classmethod update_many(*filter_kwargs: dict*, ***kwargs*) → pymongo.results.UpdateResult

async classmethod update_one(*filter_kwargs: dict*, ***kwargs*) → pymongo.results.UpdateResult

class fastapi_contrib.db.models.**MongoDBTimeStampedModel**(*, *id: int = None*, *created: datetime.datetime = None*)

Bases: *fastapi_contrib.db.models.MongoDBModel*

TimeStampedModel to use when you need to have *created* field, populated at your model creation time.

Use it as follows:

```
class MyTimeStampedModel(MongoDBTimeStampedModel):
```

(continues on next page)

(continued from previous page)

```
    class Meta:
        collection = "timestamped_collection"


mymodel = MyTimeStampedModel()
mymodel.save()

assert isinstance(mymodel.id, int)
assert isinstance(mymodel.created, datetime)
```

created: datetime.datetime

classmethod set_created_now(*v: datetime.datetime*) → datetime.datetime
If created is supplied (ex. from DB) -> use it, otherwise generate new.

class fastapi_contrib.db.models.**NotSet**
Bases: object

fastapi_contrib.db.serializers module

fastapi_contrib.db.utils module

async fastapi_contrib.db.utils.**create_indexes**() → List[str]
Gets all models in project and then creates indexes for each one of them. :return: list of indexes that has been invoked to create

(could've been created earlier, it doesn't raise in this case)

fastapi_contrib.db.utils.**default_id_generator**(*bit_size: int = 32*) → int
Generator of IDs for newly created MongoDB rows.

Returns *bit_size* long int

fastapi_contrib.db.utils.**get_db_client**()
Gets instance of MongoDB client for you to make DB queries. :return: MongoDBClient

fastapi_contrib.db.utils.**get_models**() → list
Scans *settings.apps_folder_name*. Find *models* modules in each of them and get all attributes there. Last step is to filter attributes to return only those, subclassed from MongoDBModel (or timestamped version).

Used internally only by *create_indexes* function.

Returns list of user-defined models (subclassed from MongoDBModel) in apps

fastapi_contrib.db.utils.**get_next_id**() → int
Retrieves ID generator function from the path, specified in project's conf. :return: newly generated ID

fastapi_contrib.db.utils.**setup_mongodb**(*app: fastapi.applications.FastAPI*) → None
Helper function to setup MongoDB connection & *motor* client during setup. Use during app startup as follows:

```
app = FastAPI()

@app.on_event('startup')
async def startup():
    setup_mongodb(app)
```

Parameters **app** – app object, instance of FastAPI

Returns None

Module contents

fastapi_contrib.serializers package

Submodules

fastapi_contrib.serializers.common module

class fastapi_contrib.serializers.common.**AbstractMeta**
Bases: abc.ABC

exclude: set = {}

model: *fastapi_contrib.db.models.MongoDBModel* = None

read_only_fields: set = {}

write_only_fields: set = {}

class fastapi_contrib.serializers.common.**ModelSerializer**
Bases: *fastapi_contrib.serializers.common.Serializer*

Left as a proxy for correct naming until we figure out how to inherit all the specific to model-handling methods and fields directly in here.

class fastapi_contrib.serializers.common.**Serializer**
Bases: pydantic.main.BaseModel

Base Serializer class.

Almost ALWAYS should be used in conjunction with *fastapi_contrib.serializers.openapi.patch* decorator to correctly handle inherited model fields and OpenAPI Schema generation with *response_model*.

Responsible for sanitizing data & converting JSON to & from MongoDBModel.

Contains supplemental function, related to MongoDBModel, mostly proxied to corresponding functions inside model (ex. save, update)

Heavily uses *Meta* class for fine-tuning input & output. Main fields are:

- **exclude - set of fields that are excluded when serializing to dict** and sanitizing list of dicts
- model - class of the MongoDBModel to use, inherits fields from it
- **write_only_fields - set of fields that can be accepted in request,** but excluded when serializing to dict
- **read_only_fields - set of fields that cannot be accepted in request,** but included when serializing to dict

Example usage:

```
app = FastAPI()


class SomeModel(MongoDBModel):
    field1: str
```

(continues on next page)

(continued from previous page)

```
@openapi.patch
class SomeSerializer(Serializer):
    read_only1: str = "const"
    write_only2: int
    not_visible: str = "42"

    class Meta:
        model = SomeModel
        exclude = {"not_visible"}
        write_only_fields = {"write_only2"}
        read_only_fields = {"read_only1"}


@app.get("/", response_model=SomeSerializer.response_model)
async def root(serializer: SomeSerializer):
    model_instance = await serializer.save()
    return model_instance.dict()
```

POST-ing to this route following JSON:

```
{"read_only1": "a", "write_only2": 123, "field1": "b"}
```

Should return following response:

```
{"id": 1, "field1": "b", "read_only1": "const"}
```

class Meta

Bases: *fastapi_contrib.serializers.common.AbstractMeta*

dict(**args*, ***kwargs*) → *dict*

Removes excluded fields based on *Meta* and *kwargs* :return: dict of serializer data fields

classmethod sanitize_list(*iterable: Iterable*) → List[*dict*]

Sanitize list of rows that comes from DB to not include *exclude* set.

Parameters **iterable** – sequence of dicts with model fields (from rows in DB)

Returns list of cleaned, without *excluded*, dicts with model rows

async save(*include: Optional[set] = None*, *exclude: Optional[set] = None*, *rewrite_fields: Optional[*dict*] = None*) → *fastapi_contrib.db.models.MongoDBModel*

If we have *model* attribute in Meta, it populates model with data and saves it in DB, returning instance of model.

Parameters

- **rewrite_fields** – dict of fields with values that override any other values for these fields right before inserting into DB. This is useful when you need to set some value explicitly based on request (e.g. user or token).
- **include** – fields to include from model in DB insert command
- **exclude** – fields to exclude from model in DB insert command

Returns model (MongoDBModel) that was saved

async update_many(*filter_kwargs:* dict, *skip_defaults: bool = True*, *array_fields: Optional[list] = None*) → pymongo.results.UpdateResult

If we have *model* attribute in Meta, it proxies filters & update data and after that returns actual result of update operation.

Returns result of update many operation

async update_one(*filter_kwargs:* dict, *skip_defaults: bool = True*, *array_fields: Optional[list] = None*) → pymongo.results.UpdateResult

If we have *model* attribute in Meta, it proxies filters & update data and after that returns actual result of update operation.

Returns result of update operation

fastapi_contrib.serializers.openapi module

fastapi_contrib.serializers.openapi.**patch**(*cls: Type*) → Type

Decorator for *Serializer* classes to handle inheritance from models, read- and write-only fields, combining `Meta`s.

For more info see *gen_model* method. :param cls: serializer class (model or regular) :return: wrapped class, which is newly generated pydantic's *BaseModel*

fastapi_contrib.serializers.utils module

class fastapi_contrib.serializers.utils.**FieldGenerationMode**(*value*)

Bases: int, enum.Enum

Defines modes in which fields of decorated serializer should be generated.

REQUEST = 1

RESPONSE = 2

fastapi_contrib.serializers.utils.**gen_model**(*cls: Type*, *mode:* fastapi_contrib.serializers.utils.FieldGenerationMode)

Generate *pydantic.BaseModel* based on fields in Serializer class, its Meta class and possible Model class.

Parameters

- **cls** – serializer class (could be modelserializer or regular one)
- **mode** – field generation mode

Returns newly generated *BaseModel* from fields in Model & Serializer

Module contents

fastapi_contrib.tracing package

Submodules

fastapi_contrib.tracing.middlewares module

class fastapi_contrib.tracing.middlewares.**OpentracingMiddleware**(*app: Callable[[MutableMapping[str, Any], Callable[], Awaitable[MutableMapping[str, Any]]], Callable[[MutableMapping[str, Any]], Awaitable[None]]]*, *Awaitable[None]]*, *dispatch: Optional[Callable[[starlette.requests.Request, Callable[[starlette.requests.Request], Awaitable[starlette.responses.Response]]], Awaitable[starlette.responses.Response]]] = None*)

Bases: starlette.middleware.base.BaseHTTPMiddleware

static before_request(*request: starlette.requests.Request*, *tracer*)
Gather various info about the request and start new span with the data.

async dispatch(*request: starlette.requests.Request*, *call_next: Any*) → starlette.responses.Response
Store span in some request.state storage using Tracer.scope_manager, using the returned *Scope* as Context Manager to ensure *Span* will be cleared and (in this case) *Span.finish()* be called.

Parameters

- **request** – Starlette's Request object
- **call_next** – Next callable Middleware in chain or final view

Returns Starlette's Response object

fastapi_contrib.tracing.utils module

fastapi_contrib.tracing.utils.**setup_opentracing**(*app*)
Helper function to setup opentracing with Jaeger client during setup. Use during app startup as follows:

```
app = FastAPI()

@app.on_event('startup')
async def startup():
    setup_opentracing(app)
```

Parameters **app** – app object, instance of FastAPI

Returns None

Module contents

4.1.2 Submodules

4.1.3 fastapi_contrib.conf module

class fastapi_contrib.conf.**Settings**(*_env_file: Optional[Union[pathlib.Path, str]] = '<object object>'*, *_env_file_encoding: Optional[str] = None*, *_secrets_dir: Optional[Union[pathlib.Path, str]] = None*, ***, *logger: str = 'logging'*, *log_level: str = 'INFO'*, *debug_timing: bool = False*, *request_id_header: str = 'Request-ID'*, *service_name: str = 'fastapi_contrib'*, *trace_id_header: str = 'X-TRACE-ID'*, *jaeger_host: str = 'jaeger'*, *jaeger_port: int = 5775*, *jaeger_sampler_type: str = 'probabilistic'*, *jaeger_sampler_rate: float = 1.0*, *mongodb_dsn: str = 'mongodb://example:pwd@localhost:27017'*, *mongodb_dbname: str = 'default'*, *mongodb_min_pool_size: int = 0*, *mongodb_max_pool_size: int = 100*, *mongodb_id_generator: str = 'fastapi_contrib.db.utils.default_id_generator'*, *now_function: str = None*, *TZ: str = 'UTC'*, *fastapi_app: str = None*, *user_model: str = 'fastapi_contrib.auth.models.User'*, *token_model: str = 'fastapi_contrib.auth.models.Token'*, *token_generator: str = 'fastapi_contrib.auth.utils.default_token_generator'*, *apps: List[str] = []*, *apps_folder_name: str = 'apps'*)

Bases: pydantic.env_settings.BaseSettings

Configuration settings for this library.

For now you could only change the settings via CONTRIB_<ATTRIBUTE_NAME> environment variables.

Parameters

- **logger** – Dotted path to the logger (using this attribute, standard logging methods will be used: logging.debug(), .info(), etc.
- **log_level** – Standard LEVEL for logging (DEBUG/INFO/WARNING/etc.)
- **debug_timing** – Whether to enable time logging for decorated functions
- **request_id_header** – String name for header, that is expected to have unique request id for tracing purposes. Might go away when we add opentracing here.
- **mongodb_dsn** – DSN connection string to MongoDB
- **mongodb_dbname** – String name of a database to connect to in MongoDB
- **mongodb_id_generator** – Dotted path to the function, which will be used when assigning IDs for MongoDB records
- **now_function** – Dotted path to the function, which will be used when assigning *created* field for MongoDB records. Should be used throughout the code for consistency.
- **fastapi_app** – Dotted path to the instance of *FastAPI* main app.
- **user_model** – Dotted path to the class, which will be used as the main user model in a project.
- **token_model** – Dotted path to the class, which will be used as the main token model in a project.
- **token_generator** – Dotted path to the function, which will be used when assigning *key* attribute of a token model.

- **apps** – List of app names. For now only needed to detect models inside them and generate indexes upon startup (see: *create_indexes*)
- **apps_folder_name** – Name of the folders which contains dirs with apps.

class Config
Bases: object

env_prefix = 'CONTRIB_'

secrets_dir = None

TZ: str

apps: List[str]

apps_folder_name: str

debug_timing: bool

fastapi_app: str

jaeger_host: str

jaeger_port: int

jaeger_sampler_rate: float

jaeger_sampler_type: str

log_level: str

logger: str

mongodb_dbname: str

mongodb_dsn: str

mongodb_id_generator: str

mongodb_max_pool_size: int

mongodb_min_pool_size: int

now_function: str

request_id_header: str

service_name: str

token_generator: str

token_model: str

trace_id_header: str

user_model: str

4.1.4 fastapi_contrib.exception_handlers module

async fastapi_contrib.exception_handlers.**http_exception_handler**(*request: starlette.requests.Request*, *exc: starlette.exceptions.HTTPException*) → *fastapi_contrib.common.responses.UJSONResponse*

Handles StarletteHTTPException, translating it into flat dict error data:

- code - unique code of the error in the system
- detail - general description of the error
- fields - list of dicts with description of the error in each field

Parameters

- **request** – Starlette Request instance
- **exc** – StarletteHTTPException instance

Returns UJSONResponse with newly formatted error data

async fastapi_contrib.exception_handlers.**internal_server_error_handler**(*request: starlette.requests.Request*, *exc: fastapi.exceptions.RequestValidationError* → *fastapi_contrib.common.responses.UJSON*

async fastapi_contrib.exception_handlers.**not_found_error_handler**(*request: starlette.requests.Request*, *exc: fastapi.exceptions.RequestValidationError*) → *fastapi_contrib.common.responses.UJSONRespon*

fastapi_contrib.exception_handlers.**parse_error**(*err: Any*, *field_names: List*, *raw: bool = True*) → Optional[dict]

Parse single error object (such as pydantic-based or fastapi-based) to dict

Parameters

- **err** – Error object
- **field_names** – List of names of the field that are already processed
- **raw** – Whether this is a raw error or wrapped pydantic error

Returns dict with name of the field (or "__all__") and actual message

fastapi_contrib.exception_handlers.**raw_errors_to_fields**(*raw_errors: List*) → List[dict]

Translates list of raw errors (instances) into list of dicts with name/msg

Parameters **raw_errors** – List with instances of raw error

Returns List of dicts (1 dict for every raw error)

fastapi_contrib.exception_handlers.**setup_exception_handlers**(*app: fastapi.applications.FastAPI*) → None

Helper function to setup exception handlers for app. Use during app startup as follows:

```
app = FastAPI()

@app.on_event('startup')
async def startup():
    setup_exception_handlers(app)
```

Parameters **app** – app object, instance of FastAPI

Returns None

async fastapi_contrib.exception_handlers.**validation_exception_handler**(*request: starlette.requests.Request*, *exc: fastapi.exceptions.RequestValidationError*) → *fastapi_contrib.common.responses.UJSONI*

Handles ValidationError, translating it into flat dict error data:

- code - unique code of the error in the system
- detail - general description of the error
- fields - list of dicts with description of the error in each field

Parameters

- **request** – Starlette Request instance
- **exc** – StarletteHTTPException instance

Returns UJSONResponse with newly formatted error data

4.1.5 fastapi_contrib.exceptions module

exception fastapi_contrib.exceptions.**BadRequestError**(*error_code: int*, *detail: Any*, *fields: Optional[List[Dict]] = None*)

Bases: *fastapi_contrib.exceptions.HTTPException*

exception fastapi_contrib.exceptions.**ForbiddenError**(*error_code: int = 403*, *detail: Any = 'Forbidden.'*, *fields: Optional[List[Dict]] = None*)

Bases: *fastapi_contrib.exceptions.HTTPException*

exception fastapi_contrib.exceptions.**HTTPException**(*status_code: int*, *error_code: int*, *detail: Optional[Any] = None*, *fields: Optional[List[Dict]] = None*)

Bases: starlette.exceptions.HTTPException

exception fastapi_contrib.exceptions.**InternalServerError**(*error_code: int = 500*, *detail: Any = 'Internal Server Error.'*, *fields: Optional[List[Dict]] = None*)

Bases: *fastapi_contrib.exceptions.HTTPException*

exception fastapi_contrib.exceptions.**NotFoundError**(*error_code: int = 404*, *detail: Any = 'Not found.'*, *fields: Optional[List[Dict]] = None*)

Bases: *fastapi_contrib.exceptions.HTTPException*

exception fastapi_contrib.exceptions.**UnauthorizedError**(*error_code: int = 401*, *detail: Any = 'Unauthorized.'*, *fields: Optional[List[Dict]] = None*)

Bases: *fastapi_contrib.exceptions.HTTPException*

4.1.6 fastapi_contrib.pagination module

class fastapi_contrib.pagination.**Pagination**(*request: starlette.requests.Request*, *offset: int = Query(0)*, *limit: int = Query(100)*)

Bases: `object`

Query params parser and db collection paginator in one.

Use it as dependency in route, then invoke *paginate* with serializer:

```
app = FastAPI()

class SomeSerializer(ModelSerializer):
    class Meta:
        model = SomeModel

@app.get("/")
async def list(pagination: Pagination = Depends()):
    filter_kwargs = {}
    return await pagination.paginate(
        serializer_class=SomeSerializer, **filter_kwargs
    )
```

Subclass this pagination to define custom default & maximum values for offset & limit:

```
class CustomPagination(Pagination):
    default_offset = 90
    default_limit = 1
    max_offset = 100`
    max_limit = 2000
```

Parameters

- **request** – starlette Request object
- **offset** – query param of how many records to skip
- **limit** – query param of how many records to show

default_limit = 100

default_offset = 0

async get_count(***kwargs*) → int

Retrieves counts for query list, filtered by kwargs.

Parameters **kwargs** – filters that are proxied in db query

Returns number of found records

async get_list(*_sort=None*, ***kwargs*) → list

Retrieves actual list of records. It comes raw, which means it retrieves dict from DB, instead of making conversion for every object in list into Model.

Parameters **kwargs** – filters that are proxied in db query

Returns list of dicts from DB, filtered by kwargs

get_next_url() → str
Constructs *next* parameter in resulting JSON, produces URL for next "page" of paginated results.

Returns URL for next "page" of paginated results.

get_previous_url() → str
Constructs *previous* parameter in resulting JSON, produces URL for previous "page" of paginated results.

Returns URL for previous "page" of paginated results.

max_limit = 1000

max_offset = None

async paginate(*serializer_class:* fastapi_contrib.serializers.common.Serializer, *_sort=None*, ***kwargs*) → dict
Actual pagination function, takes serializer class, filter options as kwargs and returns dict with the following fields:

- count - counts for query list, filtered by kwargs
- next - URL for next "page" of paginated results
- previous - URL for previous "page" of paginated results
- result - actual list of records (dicts)

Parameters

- **serializer_class** – needed to get Model & sanitize list from DB
- **kwargs** – filters that are proxied in db query

Returns dict that should be returned as a response

class fastapi_contrib.pagination.**PaginationMeta**(*name*, *bases*, *namespace*, **args*, ***kwargs*)
Bases: type

4.1.7 fastapi_contrib.permissions module

class fastapi_contrib.permissions.**BasePermission**(*request: starlette.requests.Request*)
Bases: abc.ABC

Abstract permission that all other Permissions must be inherited from.

Defines basic error message, status & error codes.

Upon initialization, calls abstract method *has_required_permissions* which will be specific to concrete implementation of Permission class.

You would write your permissions like this:

```
class TeapotUserAgentPermission(BasePermission):

    def has_required_permissions(self, request: Request) -> bool:
        return request.headers.get('User-Agent') == "Teapot v1.0"
```

error_code = 403

error_msg = 'Forbidden.'

abstract has_required_permissions(*request: starlette.requests.Request*) → bool

status_code = 403

class fastapi_contrib.permissions.**PermissionsDependency**(*permissions_classes: list*)

Bases: object

Permission dependency that is used to define and check all the permission classes from one place inside route definition.

Use it as an argument to FastAPI's *Depends* as follows:

```
app = FastAPI()

@app.get(
    "/teapot/",
    dependencies=[Depends(
        PermissionsDependency([TeapotUserAgentPermission]))]
)
async def teapot() -> dict:
    return {"teapot": True}
```

4.1.8 fastapi_contrib.routes module

class fastapi_contrib.routes.**ValidationErrorLoggingRoute**(*path: str, endpoint: Callable[[...], Any], *, response_model: Optional[Type[Any]] = None, status_code: int = 200, tags: Optional[List[str]] = None, dependencies: Optional[Sequence[fastapi.params.Depends]] = None, summary: Optional[str] = None, description: Optional[str] = None, response_description: str = 'Successful Response', responses: Optional[Dict[Union[int, str], Dict[str, Any]]] = None, deprecated: Optional[bool] = None, name: Optional[str] = None, methods: Optional[Union[Set[str], List[str]]] = None, operation_id: Optional[str] = None, response_model_include: Optional[Union[Set[Union[int, str]], Dict[Union[int, str], Any]]] = None, response_model_exclude: Optional[Union[Set[Union[int, str]], Dict[Union[int, str], Any]]] = None, response_model_by_alias: bool = True, response_model_exclude_unset: bool = False, response_model_exclude_defaults: bool = False, response_model_exclude_none: bool = False, include_in_schema: bool = True, response_class: Union[Type[starlette.responses.Response], fastapi.datastructures.DefaultPlaceholder] = <fastapi.datastructures.DefaultPlaceholder object>, dependency_overrides_provider: Optional[Any] = None, callbacks: Optional[List[starlette.routing.BaseRoute]] = None*)

Bases: fastapi.routing.APIRoute

get_route_handler() → Callable

4.1.9 Module contents

Top-level package for FastAPI Contrib.

CHAPTER

FIVE

CONTRIBUTING

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

5.1 Types of Contributions

5.1.1 Report Bugs

Report bugs at https://github.com/identixone/fastapi_contrib/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

5.1.2 Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" and "help wanted" is open to whoever wants to implement it.

5.1.3 Implement Features

Look through the GitHub issues for features. Anything tagged with "enhancement" and "help wanted" is open to whoever wants to implement it.

5.1.4 Write Documentation

FastAPI Contrib could always use more documentation, whether as part of the official FastAPI Contrib docs, in docstrings, or even on the web in blog posts, articles, and such.

5.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/identixone/fastapi_contrib/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

5.2 Get Started!

Ready to contribute? Here's how to set up *fastapi_contrib* for local development.

1. Fork the *fastapi_contrib* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/fastapi_contrib.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ mkvirtualenv fastapi_contrib
$ cd fastapi_contrib/
$ python setup.py develop
```

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

```
$ flake8 fastapi_contrib tests
$ python setup.py test or py.test
$ tox
```

   To get flake8 and tox, just pip install them into your virtualenv.

6. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

5.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 3.7, 3.8, 3.9. Check https://travis-ci.org/identixone/fastapi_contrib/pull_requests and make sure that the tests pass for all supported Python versions.

5.4 Tips

To run a subset of tests:

```
$ py.test tests.test_fastapi_contrib
```

5.5 Deploying

A reminder for the maintainers on how to deploy. Make sure all your changes are committed. Then run:

```
$ bumpversion patch # possible: major / minor / patch
$ git push
$ git push --tags
```

Travis will then deploy to PyPI if tests pass.

CHAPTER

SIX

CREDITS

6.1 Development Lead

- Lev Rubel <l@datacorp.ee>

6.2 Contributors

- @yarara
- @BumagniyPacket
- @aCLr
- @rsommerard
- @mumtozvalijonov
- @danield137
- @Haider8

CHAPTER

SEVEN

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

f

fastapi_contrib, 36
fastapi_contrib.auth, 20
fastapi_contrib.auth.backends, 17
fastapi_contrib.auth.middlewares, 18
fastapi_contrib.auth.models, 18
fastapi_contrib.auth.permissions, 19
fastapi_contrib.auth.serializers, 20
fastapi_contrib.common, 22
fastapi_contrib.common.middlewares, 20
fastapi_contrib.common.responses, 21
fastapi_contrib.common.utils, 21
fastapi_contrib.conf, 29
fastapi_contrib.db, 25
fastapi_contrib.db.client, 22
fastapi_contrib.db.models, 23
fastapi_contrib.db.utils, 24
fastapi_contrib.exception_handlers, 31
fastapi_contrib.exceptions, 32
fastapi_contrib.pagination, 33
fastapi_contrib.permissions, 34
fastapi_contrib.routes, 36
fastapi_contrib.serializers, 27
fastapi_contrib.serializers.common, 25
fastapi_contrib.serializers.openapi, 27
fastapi_contrib.serializers.utils, 27
fastapi_contrib.tracing, 29
fastapi_contrib.tracing.middlewares, 28
fastapi_contrib.tracing.utils, 28

INDEX

A

AbstractMeta (*class in fastapi_contrib.serializers.common*), 25
anystr_strip_whitespace (*fastapi_contrib.db.models.MongoDBModel.Config attribute*), 23
apps (*fastapi_contrib.conf.Settings attribute*), 30
apps_folder_name (*fastapi_contrib.conf.Settings attribute*), 30
async_timing() (*in module fastapi_contrib.common.utils*), 21
AuthBackend (*class in fastapi_contrib.auth.backends*), 17
authenticate() (*fastapi_contrib.auth.backends.AuthBackend method*), 17
AuthenticationMiddleware (*class in fastapi_contrib.auth.middlewares*), 18

B

BadRequestError, 32
BasePermission (*class in fastapi_contrib.permissions*), 34
before_request() (*fastapi_contrib.tracing.middlewares.OpentracingMiddleware static method*), 28

C

collection (*fastapi_contrib.auth.models.Token.Meta attribute*), 19
collection (*fastapi_contrib.auth.models.User.Meta attribute*), 19
count() (*fastapi_contrib.db.client.MongoDBClient method*), 22
count() (*fastapi_contrib.db.models.MongoDBModel class method*), 23
create_indexes() (*fastapi_contrib.db.models.MongoDBModel class method*), 23
create_indexes() (*in module fastapi_contrib.db.utils*), 24
created (*fastapi_contrib.db.models.MongoDBTimeStampedModel attribute*), 24

D

debug_timing (*fastapi_contrib.conf.Settings attribute*), 30
default_id_generator() (*in module fastapi_contrib.db.utils*), 24
default_limit (*fastapi_contrib.pagination.Pagination attribute*), 33
default_offset (*fastapi_contrib.pagination.Pagination attribute*), 33
default_on_error() (*fastapi_contrib.auth.middlewares.AuthenticationM static method*), 18
delete() (*fastapi_contrib.db.client.MongoDBClient method*), 22
delete() (*fastapi_contrib.db.models.MongoDBModel class method*), 23
dict() (*fastapi_contrib.serializers.common.Serializer method*), 26
dispatch() (*fastapi_contrib.common.middlewares.StateRequestIDMiddlev method*), 20
dispatch() (*fastapi_contrib.tracing.middlewares.OpentracingMiddleware method*), 28

E

env_prefix (*fastapi_contrib.conf.Settings.Config attribute*), 30
error_code (*fastapi_contrib.auth.permissions.IsAuthenticated attribute*), 19
error_code (*fastapi_contrib.permissions.BasePermission attribute*), 34
error_msg (*fastapi_contrib.auth.permissions.IsAuthenticated attribute*), 19
error_msg (*fastapi_contrib.permissions.BasePermission attribute*), 34
exclude (*fastapi_contrib.auth.serializers.TokenSerializer.Meta attribute*), 20
exclude (*fastapi_contrib.serializers.common.AbstractMeta attribute*), 25
expires (*fastapi_contrib.auth.models.Token attribute*), 19

F

fastapi_app (*fastapi_contrib.conf.Settings attribute*),

30
fastapi_contrib
module, 36
fastapi_contrib.auth
module, 20
fastapi_contrib.auth.backends
module, 17
fastapi_contrib.auth.middlewares
module, 18
fastapi_contrib.auth.models
module, 18
fastapi_contrib.auth.permissions
module, 19
fastapi_contrib.auth.serializers
module, 20
fastapi_contrib.common
module, 22
fastapi_contrib.common.middlewares
module, 20
fastapi_contrib.common.responses
module, 21
fastapi_contrib.common.utils
module, 21
fastapi_contrib.conf
module, 29
fastapi_contrib.db
module, 25
fastapi_contrib.db.client
module, 22
fastapi_contrib.db.models
module, 23
fastapi_contrib.db.utils
module, 24
fastapi_contrib.exception_handlers
module, 31
fastapi_contrib.exceptions
module, 32
fastapi_contrib.pagination
module, 33
fastapi_contrib.permissions
module, 34
fastapi_contrib.routes
module, 36
fastapi_contrib.serializers
module, 27
fastapi_contrib.serializers.common
module, 25
fastapi_contrib.serializers.openapi
module, 27
fastapi_contrib.serializers.utils
module, 27
fastapi_contrib.tracing
module, 29
fastapi_contrib.tracing.middlewares
module, 28
fastapi_contrib.tracing.utils
module, 28
FieldGenerationMode (*class in fastapi_contrib.serializers.utils*), 27
ForbiddenError, 32

G

gen_model() (*in module fastapi_contrib.serializers.utils*), 27
get() (*fastapi_contrib.db.client.MongoDBClient method*), 22
get() (*fastapi_contrib.db.models.MongoDBModel class method*), 23
get_collection() (*fastapi_contrib.db.client.MongoDBClient method*), 22
get_count() (*fastapi_contrib.pagination.Pagination method*), 33
get_current_app() (*in module fastapi_contrib.common.utils*), 21
get_db_client() (*in module fastapi_contrib.db.utils*), 24
get_db_collection() (*fastapi_contrib.db.models.MongoDBModel class method*), 23
get_list() (*fastapi_contrib.pagination.Pagination method*), 33
get_logger() (*in module fastapi_contrib.common.utils*), 21
get_models() (*in module fastapi_contrib.db.utils*), 24
get_next_id() (*in module fastapi_contrib.db.utils*), 24
get_next_url() (*fastapi_contrib.pagination.Pagination method*), 34
get_now() (*in module fastapi_contrib.common.utils*), 21
get_previous_url() (*fastapi_contrib.pagination.Pagination method*), 34
get_route_handler() (*fastapi_contrib.routes.ValidationErrorLoggingRoute method*), 36
get_timezone() (*in module fastapi_contrib.common.utils*), 21

H

has_required_permissions() (*fastapi_contrib.auth.permissions.IsAuthenticated method*), 19
has_required_permissions() (*fastapi_contrib.permissions.BasePermission method*), 35
http_exception_handler() (*in module fastapi_contrib.exception_handlers*), 31
HTTPException, 32

I

id (*fastapi_contrib.db.models.MongoDBModel attribute*), 23
indexes (*fastapi_contrib.auth.models.Token.Meta attribute*), 19
insert() (*fastapi_contrib.db.client.MongoDBClient method*), 22
internal_server_error_handler() (*in module fastapi_contrib.exception_handlers*), 31
InternalServerError, 32
is_active (*fastapi_contrib.auth.models.Token attribute*), 19
IsAuthenticated (*class in fastapi_contrib.auth.permissions*), 19

J

jaeger_host (*fastapi_contrib.conf.Settings attribute*), 30
jaeger_port (*fastapi_contrib.conf.Settings attribute*), 30
jaeger_sampler_rate (*fastapi_contrib.conf.Settings attribute*), 30
jaeger_sampler_type (*fastapi_contrib.conf.Settings attribute*), 30

K

key (*fastapi_contrib.auth.models.Token attribute*), 19

L

list() (*fastapi_contrib.db.client.MongoDBClient method*), 22
list() (*fastapi_contrib.db.models.MongoDBModel class method*), 23
log_level (*fastapi_contrib.conf.Settings attribute*), 30
logger (*fastapi_contrib.conf.Settings attribute*), 30

M

max_limit (*fastapi_contrib.pagination.Pagination attribute*), 34
max_offset (*fastapi_contrib.pagination.Pagination attribute*), 34
model (*fastapi_contrib.auth.serializers.TokenSerializer.Meta attribute*), 20
model (*fastapi_contrib.serializers.common.AbstractMeta attribute*), 25
ModelSerializer (*class in fastapi_contrib.serializers.common*), 25
module
    fastapi_contrib, 36
    fastapi_contrib.auth, 20
    fastapi_contrib.auth.backends, 17
    fastapi_contrib.auth.middlewares, 18
    fastapi_contrib.auth.models, 18
    fastapi_contrib.auth.permissions, 19
    fastapi_contrib.auth.serializers, 20
    fastapi_contrib.common, 22
    fastapi_contrib.common.middlewares, 20
    fastapi_contrib.common.responses, 21
    fastapi_contrib.common.utils, 21
    fastapi_contrib.conf, 29
    fastapi_contrib.db, 25
    fastapi_contrib.db.client, 22
    fastapi_contrib.db.models, 23
    fastapi_contrib.db.utils, 24
    fastapi_contrib.exception_handlers, 31
    fastapi_contrib.exceptions, 32
    fastapi_contrib.pagination, 33
    fastapi_contrib.permissions, 34
    fastapi_contrib.routes, 36
    fastapi_contrib.serializers, 27
    fastapi_contrib.serializers.common, 25
    fastapi_contrib.serializers.openapi, 27
    fastapi_contrib.serializers.utils, 27
    fastapi_contrib.tracing, 29
    fastapi_contrib.tracing.middlewares, 28
    fastapi_contrib.tracing.utils, 28
mongodb_dbname (*fastapi_contrib.conf.Settings attribute*), 30
mongodb_dsn (*fastapi_contrib.conf.Settings attribute*), 30
mongodb_id_generator (*fastapi_contrib.conf.Settings attribute*), 30
mongodb_max_pool_size (*fastapi_contrib.conf.Settings attribute*), 30
mongodb_min_pool_size (*fastapi_contrib.conf.Settings attribute*), 30
MongoDBClient (*class in fastapi_contrib.db.client*), 22
MongoDBModel (*class in fastapi_contrib.db.models*), 23
MongoDBModel.Config (*class in fastapi_contrib.db.models*), 23
MongoDBTimeStampedModel (*class in fastapi_contrib.db.models*), 23

N

not_found_error_handler() (*in module fastapi_contrib.exception_handlers*), 31
NotFoundError, 32
NotSet (*class in fastapi_contrib.db.models*), 24
now_function (*fastapi_contrib.conf.Settings attribute*), 30

O

OpentracingMiddleware (*class in fastapi_contrib.tracing.middlewares*), 28

P

paginate() (*fastapi_contrib.pagination.Pagination*

method), 34
Pagination (*class in fastapi_contrib.pagination*), 33
PaginationMeta (*class in fastapi_contrib.pagination*), 34
parse_error() (*in module fastapi_contrib.exception_handlers*), 31
patch() (*in module fastapi_contrib.serializers.openapi*), 27
PermissionsDependency (*class in fastapi_contrib.permissions*), 35

R

raw_errors_to_fields() (*in module fastapi_contrib.exception_handlers*), 31
read_only_fields (*fastapi_contrib.serializers.common.AbstractMeta attribute*), 25
render() (*fastapi_contrib.common.responses.UJSONResponse method*), 21
REQUEST (*fastapi_contrib.serializers.utils.FieldGenerationMode attribute*), 27
request_id_header (*fastapi_contrib.conf.Settings attribute*), 30
request_id_header_name (*fastapi_contrib.common.middlewares.StateRequestIDMiddleware property*), 20
resolve_dotted_path() (*in module fastapi_contrib.common.utils*), 21
RESPONSE (*fastapi_contrib.serializers.utils.FieldGenerationMode attribute*), 27

S

sanitize_list() (*fastapi_contrib.serializers.common.Serializer class method*), 26
save() (*fastapi_contrib.db.models.MongoDBModel method*), 23
save() (*fastapi_contrib.serializers.common.Serializer method*), 26
secrets_dir (*fastapi_contrib.conf.Settings.Config attribute*), 30
Serializer (*class in fastapi_contrib.serializers.common*), 25
Serializer.Meta (*class in fastapi_contrib.serializers.common*), 26
service_name (*fastapi_contrib.conf.Settings attribute*), 30
set_created_now() (*fastapi_contrib.db.models.MongoDBModel class method*), 24
set_id() (*fastapi_contrib.db.models.MongoDBModel class method*), 23
set_key() (*fastapi_contrib.auth.models.Token class method*), 19
Settings (*class in fastapi_contrib.conf*), 29
Settings.Config (*class in fastapi_contrib.conf*), 30
setup_exception_handlers() (*in module fastapi_contrib.exception_handlers*), 31
setup_mongodb() (*in module fastapi_contrib.db.utils*), 24
setup_opentracing() (*in module fastapi_contrib.tracing.utils*), 28
StateRequestIDMiddleware (*class in fastapi_contrib.common.middlewares*), 20
status_code (*fastapi_contrib.auth.permissions.IsAuthenticated attribute*), 19
status_code (*fastapi_contrib.permissions.BasePermission attribute*), 35

T

Token (*class in fastapi_contrib.auth.models*), 18
Token.Meta (*class in fastapi_contrib.auth.models*), 19
token_generator (*fastapi_contrib.conf.Settings attribute*), 30
token_model (*fastapi_contrib.conf.Settings attribute*), 30
TokenSerializer (*class in fastapi_contrib.auth.serializers*), 20
TokenSerializer.Meta (*class in fastapi_contrib.auth.serializers*), 20
trace_id_header (*fastapi_contrib.conf.Settings attribute*), 30
TZ (*fastapi_contrib.conf.Settings attribute*), 30

U

UJSONResponse (*class in fastapi_contrib.common.responses*), 21
UnauthorizedError, 32
update_many() (*fastapi_contrib.db.client.MongoDBClient method*), 22
update_many() (*fastapi_contrib.db.models.MongoDBModel class method*), 23
update_many() (*fastapi_contrib.serializers.common.Serializer method*), 26
update_one() (*fastapi_contrib.db.client.MongoDBClient method*), 22
update_one() (*fastapi_contrib.db.models.MongoDBModel class method*), 23
update_one() (*fastapi_contrib.serializers.common.Serializer method*), 27
User (*class in fastapi_contrib.auth.models*), 19
User.Meta (*class in fastapi_contrib.auth.models*), 19
user_id (*fastapi_contrib.auth.models.Token attribute*), 19
user_model (*fastapi_contrib.conf.Settings attribute*), 30
username (*fastapi_contrib.auth.models.User attribute*), 19

V

validation_exception_handler() (*in module*

fastapi_contrib.exception_handlers), 32
ValidationErrorLoggingRoute (*class in fastapi_contrib.routes*), 36

W

write_only_fields (*fastapi_contrib.serializers.common.AbstractMeta attribute*), 25