Marshmallow-Mongoengine Documentation

Release 0.7.7

Emmanuel Leblond

January 30, 2017

Contents

1 Contents **3**
- 1.1 Tutorial . . . 3
- 1.2 API Reference . . . 5

2 Indices and tables **7**

Python Module Index **9**

Mongoengine integration with the marshmallow (de)serialization library.

CHAPTER 1

Contents

Tutorial A quick tutorial to start with the project

API Reference The complete API documentation

1.1 Tutorial

Marshmallow-Mongoengine is about bringing together a Mongoengine Document with a Marshmallow Schema.

1.1.1 Warming up

First we need a Mongoengine Document:

```
import mongoengine as me


class Task(me.EmbeddedDocument):
    content = me.StringField(required=True)
    priority = me.IntField(default=1)


class User(me.Document):
    name = me.StringField()
    password = me.StringField(required=True)
    email = me.StringField()
    tasks = me.ListField(me.EmbeddedDocumentField(Task))
```

Great ! Now it's time for the Marshmallow Schema. To keep things DRY, we use marshmallow-mongoengine to do the mapping:

```
import marshmallow_mongoengine as ma


class UserSchema(ma.ModelSchema):
    class Meta:
        model = User
```

Finally it's time to use our schema to load/dump documents:

```
>>> user_schema = UserSchema()
>>> u, errors = user.schema.load({
```

```
...     "name": "John Doe", "email": "jdoe@example.com", "password": "123456",
...     "tasks": [{"content": "Find a proper password"}]})
>>> u.save()
<User: User object>
>>> u.name
"John Doe"
>>> u.tasks
[<Task: Task object>]
>>> user_schema.dump(u)
UnmarshalResult(data={"name": "John Doe", "email": "jdoe@example.com", "password": "123456", "tasks"
```

If the document already exists, we can update it using *update*

```
>>> u
>>> u2, errors = user.schema.update(u, {"name": "Jacques Faite"})
>>> u2 is u
True
>>> u2.name
"Jacques Faite"
```

Note: *required* argument in the fields is not taken into account when using update

1.1.2 Configuring the schema

Let say we use *user_schema.dump* to send data to a client throught HTTP. In this case, returning the *password* field seems a pretty bad idea !

We could solve this by using marshmallow's *Meta.exclude* list, but this means the field would be also excluded loading.

The solution is to use the *Model.model_fields_kwargs* option to customize the field (de)seriliazers:

```
class UserSchemaNoPassword(ma.ModelSchema):
    class Meta:
        model = User
        model_fields_kwargs = {'password': {'load_only': True}}
```

No consider the loading process: for the moment we directly put the password in the password field. This is not a good idea - this is also a really poor idea to use "123456" as password ;-) - we should first hash and salt it.

To do that, we need to disable the build of the Mongoengine document by specifying *Model.model_build_obj*

```
class UserSchemaJSON(ma.ModelSchema):
    class Meta:
        model = User
        model_build_obj = False # default is True
```

Now the schema will do all the integrity checks, but after that will stop and return a dict:

```
>>> user_schema = UserSchemaJSON()
>>> data, errors = user.schema.load({"name": "John Doe", "email": "jdoe@example.com", "password": "12
>>> data
{"name": "John Doe", "email": "jdoe@example.com", "password": "123456"}
>>> data["password"] = hash_and_salt(data["password"]) # Alter the data
>>> User(**data) # Finally build the Mongoengine document from the data
<User: User object>
```

1.1.3 Customizing the schema

Now let say we want to customize the way the tasks are dumped, for example we want to return the field *priority* in a more understandably way than just a number (1 => “High”, 2 => “Medium”, 3 => “Will see tomorrow”).

Given we can shadow the auto-generated fields by defining our own in the Schema, we only have to redefine the *property* field and we’re done !

```
class UserSchemaCustomPriority(ma.ModelSchema):
    class Meta:
        model = User

    priority = ma.fields.method(serialize="_priority_serializer", deserialize="_priority_deserializer

    def _priority_serializer(self, obj):
        if obj.priority == 1:
            return "High"
        elif obj.priority == 2:
            return "Medium"
        else:
            return "Will do tomorrow"
```

```
>>> user_schema = UserSchemaCustomPriority()
>>> user = User(name="John Doe", email="jdoe@example.com",
...             tasks=[{"content": "Find a proper password"},
...                    {"content": "Learn to cook", "priority": 2},
...                    {"content": "Fix issues", "priority": 3}])
>>> dump, errors = user.schema.dump(user)
>>> dump
{"name": "John Doe", "email": "jdoe@example.com", "tasks": [{"content": "Find a proper password", "pr
```

1.2 API Reference

class marshmallow_mongoengine.**ModelSchema**(*extra=None*, *only=()*, *exclude=()*, *prefix=u''*, *strict=None*, *many=False*, *context=None*, *load_only=()*, *dump_only=()*, *partial=False*)

Base class for Mongoengine model-based Schemas.

Example:

```
from marshmallow_mongoengine import ModelSchema
from mymodels import User

class UserSchema(ModelSchema):
    class Meta:
        model = User
```

OPTIONS_CLASS

alias of *SchemaOpts*

update(*obj*, *data*)

Helper function to update an already existing document instead of creating a new one. :param obj: Mongoengine Document to update :param data: incomming payload to deserialize :return: an :class UnmarshallResult:

Example:

```
from marshmallow_mongoengine import ModelSchema
from mymodels import User

class UserSchema(ModelSchema):
    class Meta:
        model = User

def update_obj(id, payload):
    user = User.objects(id=id).first()
    result = UserSchema().update(user, payload)
    result.data is user # True
```

Note:

Given the update is done on a existing object, the required param on the fields is ignored

class marshmallow_mongoengine.**SchemaOpts**(*meta*)

Options class for *ModelSchema*. Adds the following options:

- **model: The Mongoengine Document model to generate the *Schema*** from (required).
- **model_fields_kwargs: Dict of {field: kwargs} to provide as** additionals argument during fields creation.
- **model_build_obj: If true, :Schema load: returns a :model: objects** instead of a dict (default: True).
- **model_converter: *ModelConverter* class to use for converting the** Mongoengine Document model to marshmallow fields.
- **model_dump_only_pk: If the document autogenerate it primary_key** (default behaviour in Mongoengine), ignore it from the incomming data (default: False)
- **model_skip_values: Skip the field if it contains one of the given** values (default: None, [] and { })

class marshmallow_mongoengine.**ModelConverter**

Class that converts a mongoengine Document into a dictionary of corresponding marshmallow *Fields <marshmallow.fields.Field>*.

exception marshmallow_mongoengine.**ModelConversionError**

Raised when an error occurs in converting a Mongoengine construct to a marshmallow object.

marshmallow_mongoengine.**register_field_builder**(*mongo_field_cls*, *builder*)

Register a :class MetaFieldBuilder: to a given Mongoengine Field :param mongo_field_cls: Mongoengine Field :param build: field_builder to register

marshmallow_mongoengine.**register_field**(*mongo_field_cls*, *marshmallow_field_cls*, *available_params=()*)

Bind a marshmallow field to it corresponding mongoengine field :param mongo_field_cls: Mongoengine Field :param marshmallow_field_cls: Marshmallow Field :param available_params: List of :class marshmallow_mongoengine.cnoversion.params.MetaParam:

instances to import the mongoengine field config to marshmallow

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

m

marshmallow_mongoengine, 5

M

marshmallow_mongoengine (module), 5
ModelConversionError, 6
ModelConverter (class in marshmallow_mongoengine), 6
ModelSchema (class in marshmallow_mongoengine), 5

O

OPTIONS_CLASS (marshmallow_mongoengine.ModelSchema attribute), 5

R

register_field() (in module marshmallow_mongoengine), 6
register_field_builder() (in module marshmallow_mongoengine), 6

S

SchemaOpts (class in marshmallow_mongoengine), 6

U

update() (marshmallow_mongoengine.ModelSchema method), 5