Dynamic Form

Release 0.0.1

Rafael S. Müller <rafa.molitoris@gmail.com>

Oct 18, 2022

CONTENTS:

Dynamic Form is a tool to store forms in a data store and dynamically create them during runtime. The current implementation supports loading the forms directly from MongoDB.

CHAPTER

ONE

GETTING STARTED

1.1 Workflow

To load a form, the Form Manager is instructed to create a form. The Form Manager passes the request to the Data Store. The Data Store establishes a connection to the database and downloads the form template and returns it to the Form Manager. The Form Parser is requested to convert the form template into a form. The converted form is returned via the Form Manager.

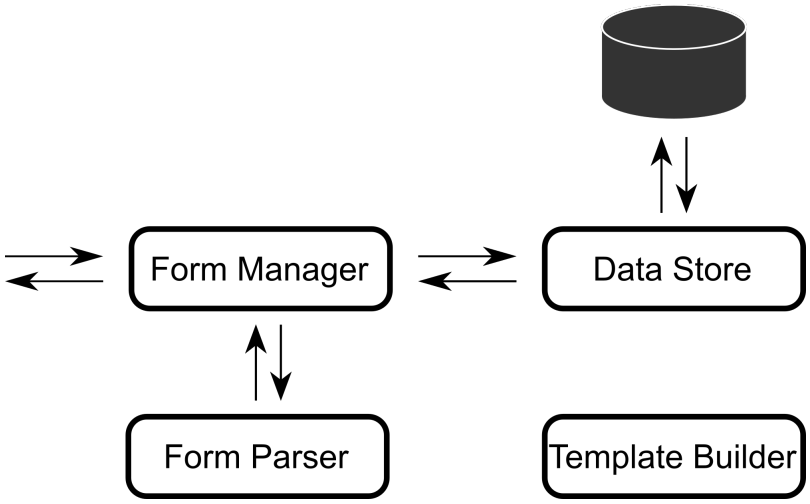

Fig. 1: The workflow to load a form from a data store.

1.2 Components

Dynamic form consists of four components:

1. Form Manager
2. Data Store
3. Form Parser
4. Template Builder

There are interfaces for Data Store and Form Parser. This allows any combination of data sources and form types. The corresponding implementations are passed to the Form Manager. All concrete implementation of the data store

and the form parser have to inherit from their abstract base classes: *dynamic_form.interfaces.IDataStore* and *dynamic_form.interfaces.IFormParser*, respectively.

1.2.1 Form Manager

The `FormManager` is the controller of this package. It is his job to load and save forms from and to a data store. It can be found in *dynamic_form.form_manager.FormManager*. The constructor of the `FormManager` expects two arguments: a data store and a form parser. If the data store contains a form, we load it through the form manager by the form's name.

```
from pymongo import MongoClient
from dynamic_form import FormManager, MongoDataStore, JsonFlaskParser

client = MongoClient()
form_manager = FormManager(
                    data_store=MongoDataStore(client["database"]["collection"]),
                    input_parser=JsonFlaskParser())

form_manager.get_form("signup")
```

1.2.2 Data Store

The data store serves as the interface to the underlying data store. It is responsible to fetch and dump forms in the data store representation.

1.2.3 Form Parser

The form parser is responsible to convert form templates from the data store into forms.

CHAPTER

TWO

DYNAMICFORM

2.1 dynamic_form package

2.1.1 Submodules

2.1.2 dynamic_form.datastore_mongodb module

class dynamic_form.datastore_mongodb.**MongoDataStore**(*db_collection*)
Bases: *dynamic_form.interfaces.IDataStore*

Data store implementation for Mongo database

deprecate_form(*identifier*)
Deprecate form (forms should not be deleted)

find_form(*search_filter*, **args*, ***kwargs*)
Search query to find forms

insert_form(*form_template*)
Push new form to the database

load_form(*identifier*)
Load form based on unique identifier

load_form_by_name(*form_name*)
load form based on form_name

load_forms()
Load all forms from database

2.1.3 dynamic_form.errors module

exception dynamic_form.errors.**DataStoreException**
Bases: *dynamic_form.errors.DynamicFormException*

exception dynamic_form.errors.**DynamicFormException**
Bases: Exception

Base exception for dynamic forms

exception dynamic_form.errors.**FormManagerException**
Bases: *dynamic_form.errors.DynamicFormException*

exception dynamic_form.errors.**FormParserException**
Bases: *dynamic_form.errors.DynamicFormException*

2.1.4 dynamic_form.form_manager module

class dynamic_form.form_manager.**FormManager**(*data_store=None*, *format_parser=<dynamic_form.parser_json.JsonFlaskParser object>*, *initial_load=True*, *max_age_seconds=60*)

Bases: object

A controller which fetches form templates from a data store and converts them into Forms.

The forms are cached in an expiring dict (default 60 seconds, 100 items). This reduces traffic to the data store.

get_cached_form_names()
Return names of all form currently in the cache

get_form_by_name(*form_name*, *use_cache=True*)
Return form based on form_name

First, local cache is examined for the form. If unsuccessful, it tries to locate the form in the database.

Parameters

- **form_name** (*str*) – the name of the form
- **use_cache** – If false, always load from data store

Raises FormManagerException: If neither cache nor database contains form with passed name

Returns A form class as defined in the :class:FormParser

insert_form(*form_template*)
Add form to data store

Before the form template is inserted into the data store, it is parsed into the form format. This ensures that only valid form templates are added to the data store. The parsed form is added to the cache.

Parameters **form_template** –

Raises *FormParserException* – If the form_template is not parsable

Returns unique identifier of inserted form

set_max_cache_age(*seconds*)
Change the expiration time of the form cache

update_form_cache()
Update local cache with forms from database

2.1.5 dynamic_form.interfaces module

class dynamic_form.interfaces.**IDataStore**

Bases: abc.ABC

interface to load form from data store.

abstract deprecate_form(*identifier*)
Deprecate form in data store

abstract find_form(**args*, ***kwargs*)
Find form in data store based on search query

abstract insert_form(*form_template*)
Insert form into data store

abstract load_form(*identifier*)
Load form from data store based on unique identifier

abstract load_form_by_name(*name*)
Load form from data store based on name

abstract load_forms()
Load all forms from data store

class dynamic_form.interfaces.**IFormParser**
Bases: abc.ABC

Interface for form parser

abstract to_form(*template_form*)
Convert from data store format to form format

abstract to_template(*form*, ***kwargs*)
Convert from form format to data store format

2.1.6 dynamic_form.parser_json module

class dynamic_form.parser_json.**JsonFlaskParser**(*form_type=<class 'flask_wtf.form.FlaskForm'>*)
Bases: *dynamic_form.interfaces.IFormParser*

Class to build a FlaskForm from a json object

>>> from dynamic_form.template_builder import FormTemplate, FieldTemplate # … form_template = FormTemplate("") # …JsonFlaskParser.to_form()

classmethod get_choice(*field_template*, *allow_synonyms*)

to_form(*template_form*)
Convert from data store format to form format

to_template(*form*, ***kwargs*)
Convert from form format to data store format

2.1.7 dynamic_form.template_builder module

class dynamic_form.template_builder.**ArgsTemplate**(*_cls*, *objs=None*)
Bases: *dynamic_form.template_builder.BaseTemplate*

to_dict() → dict

class dynamic_form.template_builder.**BaseTemplate**
Bases: abc.ABC

abstract to_dict() → dict

class dynamic_form.template_builder.**ControlledVocabularyTemplate**(*label*, *name*, *description*, *cv_origin*, *cv_external_id=''*)
Bases: *dynamic_form.template_builder.BaseTemplate*

add_item(*item:* dynamic_form.template_builder.ItemTemplate)

to_dict() → dict

class dynamic_form.template_builder.**FieldTemplate**(*class_name*, *property*, *args: Optional[*dynamic_form.template_builder.ArgsTemplate*]* = *None*, ***kwargs*)

Bases: *dynamic_form.template_builder.BaseTemplate*

to_dict() → dict

class dynamic_form.template_builder.**FormFieldTemplate**(*prop*, *class_name='FormField'*, *args=None*, ***kwargs*)

Bases: *dynamic_form.template_builder.FieldTemplate*

add_field(*field:* dynamic_form.template_builder.FieldTemplate)

to_dict() → dict

class dynamic_form.template_builder.**FormTemplate**(*label*, *name*, *description*)

Bases: *dynamic_form.template_builder.BaseTemplate*

add_field(*field:* dynamic_form.template_builder.FieldTemplate)

to_dict() → dict

class dynamic_form.template_builder.**ItemTemplate**(*label*, *name*, *description*)

Bases: *dynamic_form.template_builder.BaseTemplate*

add_synonym(*synonym*)

to_dict() → dict

class dynamic_form.template_builder.**LNDTemplate**(*label*, *name*, *description*, *deprecated=False*)

Bases: *dynamic_form.template_builder.BaseTemplate*

Label, Name, Description template

to_dict() → dict

class dynamic_form.template_builder.**ObjectTemplate**(*class_name*, *property:* dynamic_form.template_builder.PropertyTemplate)

Bases: *dynamic_form.template_builder.BaseTemplate*

to_dict() → dict

class dynamic_form.template_builder.**ObjectsTemplate**(*objs*)

Bases: *dynamic_form.template_builder.BaseTemplate*

to_dict() → dict

class dynamic_form.template_builder.**PropertyTemplate**(*label*, *name*, *level*, *description*, *value_type: dynamic_form.template_builder.ValueTypeTemplate* = *<dynamic_form.template_builder.ValueTypeTemplate object>*, *synonyms=None*, ***kwargs*)

Bases: *dynamic_form.template_builder.LNDTemplate*

to_dict() → dict

class dynamic_form.template_builder.**UserTemplate**(*firstname*, *lastname*, *email*, *password*)

Bases: *dynamic_form.template_builder.BaseTemplate*

to_dict() → dict

class dynamic_form.template_builder.**ValueTypeTemplate**(*data_type='text'*, *controlled_vocabulary: Optional[*dynamic_form.template_builder.ControlledVocabularyTe*]* = *None*)

Bases: *dynamic_form.template_builder.BaseTemplate*

to_dict() → dict

2.1.8 Module contents

class dynamic_form.**FormManager**(*data_store=None*, *format_parser=<dynamic_form.parser_json.JsonFlaskParser object>*, *initial_load=True*, *max_age_seconds=60*)

Bases: object

A controller which fetches form templates from a data store and converts them into Forms.

The forms are cached in an expiring dict (default 60 seconds, 100 items). This reduces traffic to the data store.

get_cached_form_names()
Return names of all form currently in the cache

get_form_by_name(*form_name*, *use_cache=True*)
Return form based on form_name

First, local cache is examined for the form. If unsuccessful, it tries to locate the form in the database.

Parameters

- **form_name** (*str*) – the name of the form
- **use_cache** – If false, always load from data store

Raises FormManagerException: If neither cache nor database contains form with passed name

Returns A form class as defined in the :class:FormParser

insert_form(*form_template*)
Add form to data store

Before the form template is inserted into the data store, it is parsed into the form format. This ensures that only valid form templates are added to the data store. The parsed form is added to the cache.

Parameters **form_template** –

Raises *FormParserException* – If the form_template is not parsable

Returns unique identifier of inserted form

set_max_cache_age(*seconds*)
Change the expiration time of the form cache

update_form_cache()
Update local cache with forms from database

class dynamic_form.**IDataStore**

Bases: abc.ABC

interface to load form from data store.

abstract deprecate_form(*identifier*)
Deprecate form in data store

abstract find_form(**args*, ***kwargs*)
Find form in data store based on search query

abstract insert_form(*form_template*)
Insert form into data store

abstract load_form(*identifier*)
Load form from data store based on unique identifier

abstract load_form_by_name(*name*)
Load form from data store based on name

abstract load_forms()
Load all forms from data store

class dynamic_form.**IFormParser**
Bases: abc.ABC

Interface for form parser

abstract to_form(*template_form*)
Convert from data store format to form format

abstract to_template(*form*, ***kwargs*)
Convert from form format to data store format

class dynamic_form.**JsonFlaskParser**(*form_type=<class 'flask_wtf.form.FlaskForm'>*)
Bases: *dynamic_form.interfaces.IFormParser*

Class to build a FlaskForm from a json object

>>> from dynamic_form.template_builder import FormTemplate, FieldTemplate # … form_template = FormTemplate(“”) # …JsonFlaskParser.to_form()

classmethod get_choice(*field_template*, *allow_synonyms*)

to_form(*template_form*)
Convert from data store format to form format

to_template(*form*, ***kwargs*)
Convert from form format to data store format

class dynamic_form.**MongoDataStore**(*db_collection*)
Bases: *dynamic_form.interfaces.IDataStore*

Data store implementation for Mongo database

deprecate_form(*identifier*)
Deprecate form (forms should not be deleted)

find_form(*search_filter*, **args*, ***kwargs*)
Search query to find forms

insert_form(*form_template*)
Push new form to the database

load_form(*identifier*)
Load form based on unique identifier

load_form_by_name(*form_name*)
load form based on form_name

load_forms()
Load all forms from database

2.2 test package

2.2.1 Submodules

2.2.2 test.test_datastore_mongo module

class test.test_datastore_mongo.**TestMongoDataStore**(*methodName='runTest'*)
Bases: unittest.case.TestCase

setUp() → None
Hook method for setting up the test fixture before exercising it.

classmethod setUpClass() → None

classmethod tearDownClass() → None

test_load_forms_empty()

test_wrong_collection_cls()

class test.test_datastore_mongo.**TestMongoDataStoreWithEntry**(*methodName='runTest'*)
Bases: unittest.case.TestCase

setUp() → None
Hook method for setting up the test fixture before exercising it.

classmethod setUpClass() → None

classmethod tearDownClass() → None

test_deprecate_form()

test_load_forms()

test_load_login_form_by_id()

test_load_login_form_by_name()

test_load_nonexisting_login_form_by_name()

test_search_form()

2.2.3 test.test_form_manager module

class test.test_form_manager.**TestFormManagerInit**(*methodName='runTest'*)
Bases: unittest.case.TestCase

Test Initialization of Form Manager

test_init_wrong_data_store()

test_init_wrong_form_parser()

class test.test_form_manager.**TestFormManagerInsert**(*methodName='runTest'*)
Bases: unittest.case.TestCase

classmethod setUp() → None

classmethod setUpClass() → None

classmethod tearDownClass() → None

test_init_without_initial_load()

test_insert_form()

class test.test_form_manager.**TestFormManagerManyForms**(*methodName='runTest'*)
Bases: unittest.case.TestCase

setUp() → None
Hook method for setting up the test fixture before exercising it.

classmethod setUpClass() → None

test_many_forms()

class test.test_form_manager.**TestFromFormManagerFetch**(*methodName='runTest'*)
Bases: unittest.case.TestCase

setUp() → None
Hook method for setting up the test fixture before exercising it.

classmethod setUpClass() → None

test_cached_form_names()

test_load_form_by_name_expired_cache()
Manually reduce expiration time

test_load_form_by_name_from_cache()

test_load_form_by_nonexisting_name()

test_update_form_cache()

2.2.4 test.test_json_parser module

class test.test_json_parser.**TestJsonFormParser**(*methodName='runTest'*)
Bases: unittest.case.TestCase

The following test check the helper functions of parser json.

classmethod setUpClass() → None

test_create_field_validation()

test_create_selectfield_choice()

test_form_with_fieldList()
Create form which contains a FieldList

test_form_with_fieldList_and_FormField()

test_form_with_formfield()

test_obj_validators_widgets()
Test validator and widget parsing with and without attributes

test_simple_field_attributes_in_field()

test_simple_field_attributes_in_kwargs()

test_simple_field_attributes_in_property()

test_tuple_diff_length()
Test conversion of tuples

2.2.5 test.test_main module

test.test_main.**suite**()

2.2.6 test.test_utils module

test.test_utils.**get_login_form**()

test.test_utils.**get_many_login_forms**(*num=3*)

2.2.7 Module contents

2.3 setup module

CHAPTER

THREE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

d

t

INDEX

T

U

V