Django CBTools Documentation

Release 1.2.0

Viacheslav Iutin

August 05, 2016

Contents

Contents:

CHAPTER 1

Installation

1.1 Pre-requisite

- working Couchbase server / cluster
- working Sync-Gateway server

1.2 Requirements

- `couchbase==2.0.8`
- `django-extensions==1.6.1`
- `django-tastypie==0.12.2`
- `requests==2.9.1`
- `shortuuid==0.4.3`

`couchbase` package installation can be tricky. A recipe for Ubuntu 12:

```
sudo wget -O/etc/apt/sources.list.d/couchbase.list http://packages.couchbase.com/ubuntu/couchbase-ubu

wget -O- http://packages.couchbase.com/ubuntu/couchbase.key | sudo apt-key add -

sudo apt-get update

sudo apt-get install libcouchbase-dev libcouchbase2-libevent
```

1.3 Quick Install

Install package:

```
pip install django-cbtools
```

The following configuration settings are used for the package (you can use the set below for the fast installation):

```
COUCHBASE_BUCKET = 'default'
COUCHBASE_HOSTS = ['127.0.0.1']
COUCHBASE_PASSWORD = None
SYNC_GATEWAY_BUCKET = 'default'
```

```
SYNC_GATEWAY_URL = 'http://127.0.0.1:4984'
SYNC_GATEWAY_ADMIN_URL = 'http://127.0.0.1:4985'
SYNC_GATEWAY_USER = "demo_admin"
SYNC_GATEWAY_PASSWORD = "demo_admin_password"
SYNC_GATEWAY_GUEST_USER = "demo_guest"
SYNC_GATEWAY_GUEST_PASSWORD = "demo_guest_password"
```

Add `django_cbtools` to `INSTALLED_APPS`:

```
INSTALLED_APPS = (
    # ...
    'django_cbtools',
)
```

Create folder `couchbase_views` in the project root.

1.4 Testing

You should create a testing couchbase bucket to run the package tests (and further your apps tests). For example `default_test`.

The testing bucket must contain `test` in the name. Otherwise some helper functions will raise exception.

Certianly SyncGateway configuration must also have to be configurated properly to take in account additional bucket, for example:

```
{
    "adminInterface":"0.0.0.0:4985",
    "databases": {
        "default": {
            "server": "http://127.0.0.1:8091",
            "bucket": "default"
        },
        "default_test": {
            "server": "http://127.0.0.1:8091",
            "bucket": "default_test"
        }
    }
}
```

Also you need an alternative `settings.py` to run tests. Probably you already have similar file to run your own tests. If you don't it's time to create it now. The following settings should be changed in order to run Couchbase-related tests properly:

1. `COUCHBASE_BUCKET` is targetted to test bucket
2. `SYNC_GATEWAY_BUCKET` is targetted to test bucket
3. `COUCHBASE_STALE` is set to disable Couchbase caching

Like that, in file `settings_test.py`:

```
# ...
COUCHBASE_BUCKET = 'default_test'
COUCHBASE_STALE = False
SYNC_GATEWAY_BUCKET = 'default_test'
# ...
```

Now run tests as usual for django:

```
python manage.py test --settings=<your-project>.settings_test django_cbtools
```

CHAPTER 2

Getting Started with Django CBTools

Django Couchbase is a wrapper around couchbase python library plus several hook to Sync-Gateway API.

The document search is perfomred using `couchbase` library (directly) connection to couchbase server, but saving and retrieving of the document is done using Sync-Gateway HTTP API. This is done in order to have documents available for mobile clients, which can get all benefits of `couchbase` library only through Sync-Gateway.

The essential part of the package is models. They are inherited from django models with almost all the benefits they have: can be validated with django forms and have fields all sort of field you are used to have.

2.1 Creating Model

Typical couchbase model class looks like that:

```
from django_cbtools.models import CouchbaseModel
from django.db import models

class CBArticle(CouchbaseModel):
    class Meta:
        abstract = True

    doc_type = 'article'
    uid_prefix = 'atl'

    title = models.CharField(max_length=45, null=True, blank=True)
    year_published = models.IntegerField(default=2014)
    is_draft = models.BooleanField(default=True)

    author_uid = models.TextField()
```

Certainly you can use all the rest types of fields. Let’s review the code above.

- The class has a prefix `CB`. It is optional. But you will probably have models related to your relational database. So to distinguish them we find it’s useful to have this small prefix.
- `abstract = True` this is to avoid django migration tool to take care about changes in the couchbase models.
- `doc_type = 'article'` is the field you have to define. This is the way Django CBtools stores the type of the objects. This value is stored in the database.
- `uid_prefix = 'atl'` this is an optional prefix for the `uid` of the document. Having prefix for the `uid` help a lot to debug the application. For example you can easily define type of the document having just its `uid`. Very useful.

2.1.1 Getting Documents

The document creation is a stright forward process:

```
article = CBArticle()
article.title = 'You & Couchbase'
```

Or alternatively:

```
article = CBArticle(title='You & Couchbase')
```

2.1.2 Saving Documents / Channels

Ideally it should be as simple as that:

```
article = CBArticle(title='Couchbase & You')
article.save()
```

But if you do that you get exception:

```
CouchbaseModelError: Empty channels list can not be saved
```

Channels. This is how Sync-Gateway limit access to the documents for different mobile clients. The server side framework uses an admin user to create and save documents, so it has access to all of them, but we mind mobile clients also. So:

```
article = CBArticle(title='Couchbase & You', channels=['channel_name'])
article.save()
```

or:

```
article = CBArticle(title='Couchbase & You')
article.append_channel('channel_name')
article.save()
```

`channel_name` is given here as an example. In real world it will probably somehow related to your users. For example, somewhere in a view:

```
article = CBArticle(title='Couchbase & You')
article.append_channel(self.request.user.username)
article.save()
```

You can / should read some more about the concept of channels for Sync-Gateway here.

2.1.3 Load Documents

You usually load document if you have its `UID`:

```
article = CBArticle('atl_0a1cf319ae4e8b3d5f8249fef9d1bb2c')
print article
```

2.1.4 Load Related Documents

This is how the model supports relations. Just a small helper method to load related object. In our example above it's an `author` document:

```
from django_cbtools.model import load_related_objects

article = CBArticle('atl_0a1cf319ae4e8b3d5f8249fef9d1bb2c')
load_related_objects([article], 'author', CBAuthor)
print article.author
```

The function above just create another instance variable `author` with loaded `CBAuthor` document. By default it will check for `UID` in a filed with name `author_uid`.

Please note, the function will make only one request to couchbase to load all the related documents for the given documents.

2.1.5 Removing Documents

The package implements **soft** deletion of the documents. It means it just set `st_deleted` field of the document to `True`.

A periodic process has to be setup in order to really delete the documents when you really don't need them.

There are two important points about `st_deleted` field:

- `st_deleted` field is defined in every document you create within the package. You don't have to define it explicitely.
- You should take in account this fields when you create your views. Probably you don't want to index the deleted documents.

So to set `st_deleted` to `True` you use `delete()` method:

```
article.delete()
```

2.1.6 Document signals

The package includes a "signal dispatcher" this means you can get notified when actions such as a document is saved, updated or deleted.

The django_cbtools.signals module defines a set of signals sent by the couchbase model system.

cb_pre_save

This is sent at the beginning of a couchbase model's save() method.

Arguments sent with this signal:

sender: The model class.

instance: The actual instance being saved.

cb_post_save

Like pre_save, but sent at the end of the save() method this signal also notifys if a document was saved.

Arguments sent with this signal:

sender: The model class.

instance: The actual instance being saved.

created: A boolean; True if a new document was created else False if existing document is saved.

cb_pre_delete

Sent at the beginning of a document model's delete().

Arguments sent with this signal:

sender: The model class.

instance: The actual instance being deleted.

cb_post_delete

Like pre_delete, but sent at the end of a model's delete() method

Arguments sent with this signal:

sender: The model class.

instance: The actual instance being deleted.

Listening to signals

Here's how you listen to a signal ::

```
from django_cbtools.signals import cb_pre_save, cb_post_save, cb_pre_delete, cb_post_delete

# Listen to cb_pre_save signal
cb_pre_save.connect(pre_save_handler, CBArticle)

# Listen to cb_post_save signal
cb_post_save.connect(post_save_handler, CBArticle)

# Listen to cb_pre_delete signal
cb_pre_delete.connect(pre_delete_handler, CBArticle)

# Listen to cb_post_delete signal
cb_post_delete.connect(post_delete_handler, CBArticle)
```

2.2 Couchbase Views

Views in coachbase are JavaScript functions. You can read some more about it in couchbase documentation as it's out of the scope of this document.

This package goes with two views in: `by_channel` (the view which allows you to find documents by channel name and document type) and `by_type` which can be used to get documents of particular type.

You can see the files of the views in folder `couchbase_views/` of the project. Those files are optional and if you don't need them, just don't copy them to your project.

2.2.1 Creating Views

Firstly, create folder `couchbase_views/` in your project. Then create a `js`-file with your view, for example to find all articles of by the author `couchbase_views/by_author.js`:

```
function (doc, meta) {
    if (doc.st_deleted) {
        // the document is deleted, nothing to index
        return;
    }
    if (doc.doc_type != 'article') {
        // it's not an article document, not for this index
    }
    emit(doc.author_uid, null)
}
```

You also may want to create `reduce` function for your view. Then create yet another file with name `by_author_reduce.js`:

```
_count
```

Now your view has both `map` and `reduce` parts. The last one is optional.

2.2.2 Deploying Views

Your couchbase can not be used until they are not in couchbase server. To deploy them from command line you use command `deploy_cb_views`:

```
python manage.py create_cb_views
```

2.2.3 Views Helper Functions

`get_stale`

`get_stale`()

Short hand for

`settings.COUCHBASE_STALE if hasattr(settings, 'COUCHBASE_STALE') else STALE_OK`

It means it just getter for your `COUCHBASE_STALE` option. Please read more about it in the couchbase docs.

`query_view`

`query_view`(*view_name*, *query_key*, *query=None*)

Search for `query_key` in a view `view_name`. Return list of document `uid` s. Example:

```
import django_cbtools.models import query_view

uids = query_views('by_author', 'aut_5f8249fef9d1bb2c0a1cf319ae4e8b3d')
# uids now is list of articles
```

Internally it builds a quiry for the view, but you can build a generic view and pass it to perform more complicated view query:

```
from couchbase.views.params import Query
import django_cbtools.models import query_view

# get all articles of these two authors
query = Query(
    keys=['aut_8b3d5f8249fef9d1b', 'aut_f8249fef9d1b8b3d5'],
    stale=get_stale()
)
uids = query_views(
    'by_author',
    query_key=None,  # will be ignored anyway
    query=query
)
```

`query_objects`

`query_objects`(*view_name*, *query_key*, *class_name*, *query=None*)

Very similar to `query_view`, but it returns list of object of given `class_name` instead just keys:

```
import django_cbtools.models import query_objects
objects = query_objects('by_author', 'aut_f8249fef9d1b8b3d5', CBAuthor)
```

2.3 Sync-Gateway

2.3.1 Sync-Gateway Users

Django-cbtools package needs at least one Sync-Gateway user. The one which has full access to database:

```
SYNC_GATEWAY_USER = "django_cbtools_admin"
SYNC_GATEWAY_PASSWORD = "django_cbtools_admin_password"
```

The library will access the database using the credentials from the settings above.

If you are also working on mobile app creation you may want to have a *guest* user, the one which has access to a *public* documents (the documents in *public* channel). The *guest* user can be set like that:

```
SYNC_GATEWAY_GUEST_USER = "django_cbtools_guest"
SYNC_GATEWAY_GUEST_PASSWORD = "django_cbtools_guest_password"
```

Sync-Gateway has a concept of a *GUEST* user, but we don't use it by many reasons. So your mobile client will create pull / push processes using the credentials above to access *public* documents. The library by itself does not use these credentials. But it has a management command to create this users in Sync-Gateway:

```
python manage.py create_sg_users
```

The command above will create admin and guest user in Sync-Gateway.

If you want to create a *public* document on server side you can do that:

```
from django_cbtools.models import CHANNEL_PUBLIC

article = CBArticle()
article.append_channel(CHANNEL_PUBLIC)
article.save()
```

2.3.2 `SyncGateway` Class

At the moment Sync-Gateway does not have any "native" library to access it, but it provides awesome REST HTTP interface. `SyncGateway` class is just a simple wrapper to access this HTTP interface. Internally it uses requests package.

put_user

`SyncGateway.`**`put_user`**`(`*username*, *email*, *password*, *admin_channels*, *disabled=False*`)`

A statis method to add a user to Sync-Gateway.

Usage:

```
from django_cbtools.sync_gateway import SyncGateway

SyncGateway.put_user('username', 'some@email.com', 'pass', ['user_channel'])
```

`get_user`

`SyncGateway.`**`get_user`**`(`*username*`)`

A static method to get information about Sync-Gateway user.

Usage:

```
from django_cbtools.sync_gateway import SyncGateway

print SyncGateway.get_user('username')
```

`change_username`

`SyncGateway.`**`change_username`**`(`*old_username*, *new_username*, *password*`)`

A static method to change the username of the user.

`delete_user`

`SyncGateway.`**`delete_user`**`(`*username*`)`

A static method to delete the username of the user.

`create_session`

`SyncGateway.`**`create_session`**`(`*username*, *ttl*`)`

A static method to create session for specified username. Returns response object where you can find content with session cookie and session id.

2.4 Testing

There are several helper functions which you could find useful in your unit / intergration tests.

When you write you tests you don't have to deploy the view to test database every time. Instead you deploy them in `setUp` function of your test classes.

Your tests coulc look like that:

```
from django.test import TestCase

from django_cbtools.sync_gateway import SyncGateway
from django_cbtools.tests import clean_buckets

from dashboard.management.commands.create_cb_views import Command


class ArticleTest(TestCase):
    def setUp(self):
        super(ArticleTest, self).setUp()
        SyncGateway.put_admin_user()
        clean_buckets()
        command = Command()
        command.handle()
```

CHAPTER 3

Django-Cbtools Settings

Full list of setting Django-Cbtools recognizes.

Warning: Please don't use username and password from the lines below!

3.1 COUCHBASE_BUCKET

An example:

```
COUCHBASE_BUCKET = 'default'
```

3.2 COUCHBASE_HOSTS

Couchbase server hosts.

An example:

```
COUCHBASE_HOSTS = ['127.0.0.1']
```

3.3 COUCHBASE_PASSWORD

Couchbase password.

An example:

```
COUCHBASE_PASSWORD = None
```

OR:

```
COUCHBASE_PASSWORD = 'password'
```

3.4 COUCHBASE_STALE

Caching option for views. It's used as a query parameter. Read about values in couchbase documentation.

An example:

```
COUCHBASE_STALE = False
```

OR:

```
from couchbase.views.params import STALE_OK
COUCHBASE_STALE = STALE_OK
```

3.5 SYNC_GATEWAY_BUCKET

Basicly it's probably the same as `COUCHBASE_BUCKET`. But Sync-Gateway can have different settings.

An example:

```
SYNC_GATEWAY_BUCKET = 'default'
```

3.6 SYNC_GATEWAY_URL

An URL for Sync-Gateway server.

An example:

```
SYNC_GATEWAY_URL = 'http://127.0.0.1:4984'
```

3.7 SYNC_GATEWAY_ADMIN_URL

It's probably the same as `SYNC_GATEWAY_URL` but with different port number.

An example:

```
SYNC_GATEWAY_ADMIN_URL = 'http://127.0.0.1:4985'
```

3.8 SYNC_GATEWAY_USER

The user which will be used to access the couchbase by your application.

An example:

```
SYNC_GATEWAY_USER = "admin"
```

3.9 SYNC_GATEWAY_PASSWORD

Password for the user above.

An example:

```
SYNC_GATEWAY_PASSWORD = "admin_password"
```

3.10 SYNC_GATEWAY_GUEST_USER

Guest user.

An example:

```
SYNC_GATEWAY_GUEST_USER = "guest"
```

3.11 SYNC_GATEWAY_GUEST_PASSWORD

Password for the user above.

An example:

```
SYNC_GATEWAY_GUEST_PASSWORD = "guest_password"
```

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

Index