Wagtail

Proposal - Google Summer of Code 2022

Open this proposal in Google Docs: Wagtail

Personal Details

- Name : Vaibhav Shukla
- LinkedIn : https://linkedin.com/in/mrvaibh
- GitHub : https://github.com/mr-vaibh
- Medium : https://mrvaibh.medium.com
- Slack : **@mrvaibh**
- Telephone : +91 7457947897 (WhatsApp available)
- Country of Residence : India
- Timezone : IST (India)
- Typical Working Hours : 11:30-13:30, 15:00-19:00, 22:00-05:00 (IST) 6:00-8:00, 9:30-13:30, 16:30-23:30 (UTC)

Abstract

Project Title

Utility tools for StreamField data migrations in Wagtail

Here is the discussion page

Overview

In Wagtail, there is a field type of semi-freeform data named “StreamField”. It uses JSON representation to store "blocks" of data. While being heavily used by the editors, it is still a real pain for the developers to make migrations whenever changes are made in the block definition.

The motive of this project is to provide a set of reusable utilities to allow Wagtail implementers to easily generate data migrations for changes to StreamField block structure.

A reference to this can be found on this DOC page:
How to use StreamField for mixed content
Though this is just one of the instances where the user needs to imply one extra conversion step in order for data to become accessible again, but there are few others, for example when we’re building a new custom block/field using other nested blocks (which is very likely to happen), it can be a lot of hassle.

Why this project?

This utility driven approach of creating migrations unlocks a handful of avenues for devs to expand the data-types or block-types to store data without worrying about the nesting behaviour of blocks at all.

So, our aim is to create an installable python package updating StreamFields within data-migrations

Previous contributions to Wagtail

- **#7794 (closed)**: Fixing a disabled button styling. (ISSUE STILL OPEN)
- **#8297 (open)**: Fixing non-responsive checkbox ::after text
- **#8300 (merged)**: Clarification in doc of USER and CONTRIBUTOR guide
- **#8312 (open)**: Different lock colour when it's locked by current user (NEEDS TEST)

Contributions to *wagtail.org*

- **#153 (approved):** Getting rid of unused code as per the TODO

Schedule

Milestones

1. Creating Management Utility structure:
    a. Creating a DRY draft to understand and get used to it.
2. Setting up raw templates for migrations
3. Creating command utilities
    a. for eg: `python manage.py stfm`
    b. or maybe anything relevant, we'll discuss with senior devs according to the requirements.
4. Creating Python Package for the same
5. Tweaking tests for StreamField
6. Writing tests for Utility created
7. Writing Documentation for every new or modified feature

Timeline

Before May 20th (BUGS & TODOS):

- Fixing bugs and TODOs in Wagtail.
- Getting familiar with the code.

May 20th - June 13th (UNDERSTANDING FLOW):

- Community Bonding.
- Understanding the structure of the technologies used.
- Getting familiar with the code of the `wagtail` folder i.e. Django portion.

June 13th - 25th July (CREATING UTILITY):

- Modify the existing StreamField in models.py (field app)
- Creating a data migration template as a mould
    i. Reference — django migrations
- Add a functionality that can be executed by the RunPython command.
    i. NOTE: above command is just for temporary use
    ii. Reference — official doc, simple blog

- Create new module in management folder for automating all of this hassle
- Write the features and functionalities as mentioned in the next section.

29st July - 11th August (PACKAGING):

- Bootstrapping the package.
- Convert into an installable app.
- Creating setup.cfg/setup.py.

11th August - 5th September (TESTING):

- Using already existing tests for StreamField to test the feature.
- Write custom tests for robust production level testing.

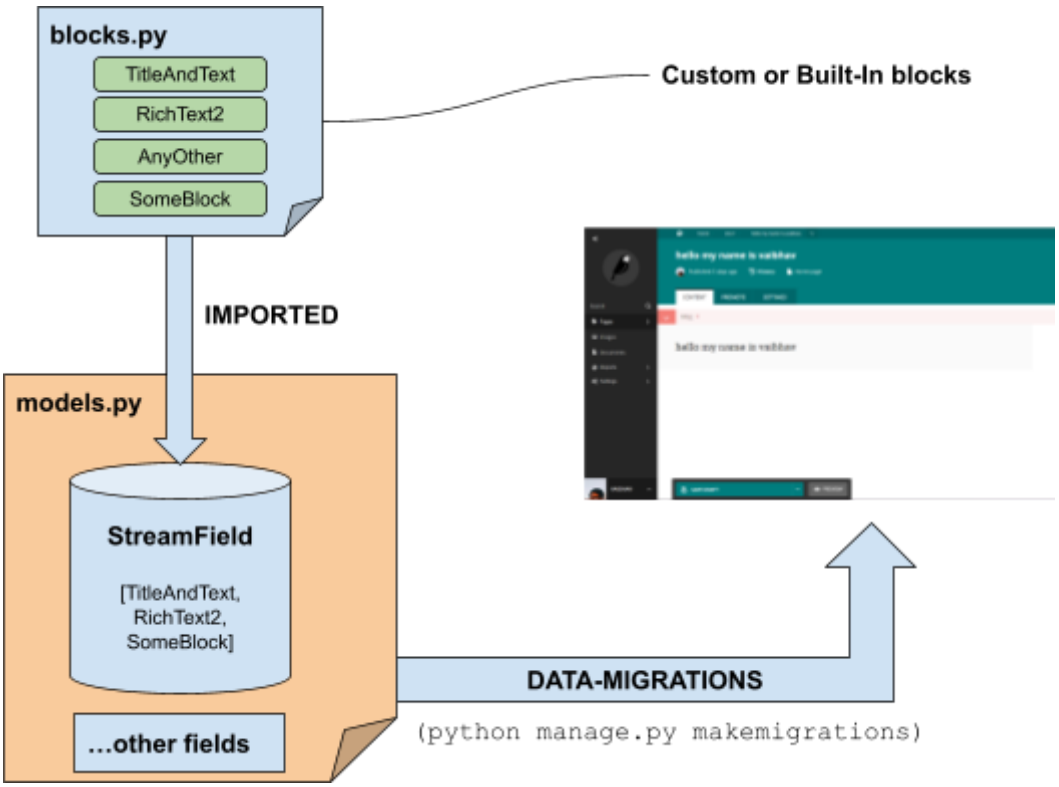
5th September - 12 September (Final Evaluation):

- Making further changes in the code to improve the Functionality, Exception handling, Bug Removal.
- To be in constant touch with the other developers on the feature and to let them know overall progress.
- Creating a crystal clear but simple documentation

The timeline might change depending on the activity of StreamField.
For Eg : If someone made some tweaks in StreamField, then might affect some flow. More details can be found in the next section.

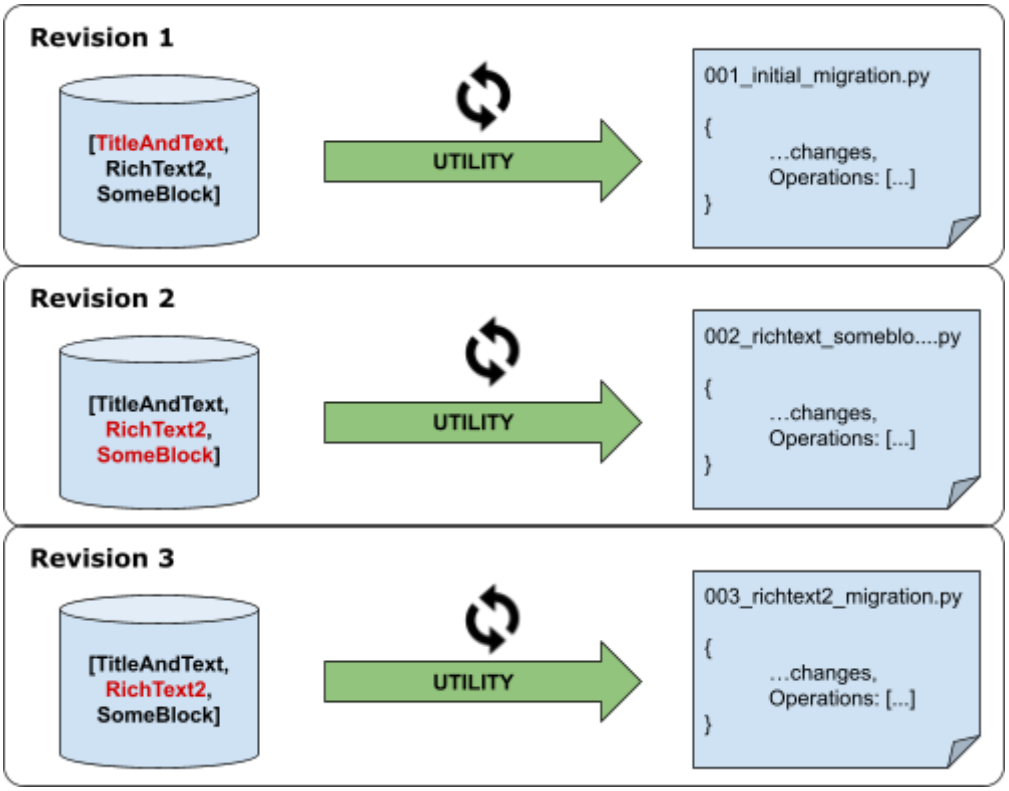
Technical Details

StreamField working structure



Right now, whenever we make any changes in any of the StructBlock, we need to run the migration in order to update the database and application.

Here is the proposed solution



Here is an example explaining how exactly the problem is likely to be approached. We'll be following Django's official documentation as a base.

To start, make an empty migration file you can work from:

```
python manage.py makemigrations --empty wagtail
```

Then, open up the file; it should look something like this:

```
# Generated by Django X.X on YYYY-MM-DD HH:MM
from django.db import migrations

class Migration(migrations.Migration):

    dependencies = [
        (admin, '0001_initial'),
    ]

    operations = [
    ]
```

Now, all you need to do is create a new function and have `RunPython` use it. RunPython expects a callable as its argument which takes two arguments.

- The first is an app registry that has the historical versions of all your models.
- The second is a SchemaEditor, which you can use to manually effect database schema changes

```
# ... This is same file

def combine_richtextfields(apps, schema_editor):
    # We can't import the StreamField model directly as it may be a
newer
    StreamField = apps.get_model('admin', 'StreamField')

    # Right now we're hard-coding the value, but later we'll create a
utility to automate
    block_types = ["some value"]
    use_json_field = None
    # ...other attrs

    for rtf in StreamField.objects.all():
        block_opts = {}
        # any other Field logic here
        rtf.use_json_field = rtf.get_internal_type()
        rtf.save()

# ... rest of the migration data
```

Once that's done, we can run python manage.py migrate as normal and the data migration will run in place alongside other migrations.

To enable accessing the model from other apps, which might be a requirement in the future according to plans I've mentioned in the next section.
Add an operation .RunPython that uses models from apps other than this one.

```
operations = [
    migrations.RunPython(combine_richtextfields),
    # other operations
]
```

Now comes the part for UTILITY.
In the root folder, we can create a new folder named StreamFieldUtility, or anything. Likewise we'll create setup.py, __init__.py and follow other procedures for creating a Python Package.

Create the management command file, let's say `` `stfm.py` `` inside the right folder structure.

```
from django.core.management.base import BaseCommand, CommandError

class Command(BaseCommand):
    help = 'Utilities for easily updating StreamFields within data
migrations'

    def change_block(self, change):
        # Utility Logic here
        call_command("python", "manage.py", "makemigration", "...")

        self.stdout.write(self.style.SUCCESS('Successful'))
```

This is how I'm planning the CLI to be in the upfront

```
$ python manage.py stfm
```

Here is a list of some CLI commands and features that are going to be implemented throughout GSoC.

- Default: when running python manage.py stfm, following operations will be triggered in the respective order:
    - it will loop through all blocks and detect changes.
    - Run makemigrations.
        - Ask for default values if trying to add non-nullable fields
    - Run migrate and update DB
- Revert: when running with `--revert` / `-R` flag, we'll be able to go back in the history and move back to the last migration.
    - By default, only the last migration would be reverted.
    - But you can specify the migration number after the flag for going all the way to one specific migration.
        - For eg: `--revert 0001`
- Delete: when running `--delete` / `-D` flag, the last migration will be reverted and deleted instantly
    - By default it is similar to revert, but the difference comes when you delete a specific migration.
        - For eg: if your current migration is 0004, and you run `--delete 0002`. It will theoretically create 0005 which will contain all the migrations except for the ones in 0002.
        - Whereas in revert it will create another 0005 with all migrations from 0002 to 0004 reverted.
- List Blocks: For listing blocks concerning the StreamField.
    - All blocks
    - Blocks used in the Stream Field
    - Blocks changed and need migrations
- Help: when running `--help` / `-H`, it'll show all the commands, flags and options documented on the CLI interface.

These links can help getting the package published:

How to Write an Installable Django App – Real Python
- Bootstrapping Django Outside of a Project
- Running Management Commands With Your Installable Django App
- Defining Your Installable Package With setup.cfg

FUTURE PLANS FOR WAGTAIL (AFTER GSoC)

Many students STOP their contribution and just leave the organisation as it is after GSoC, whereas, I have some really cool plans I'm very excited about in the long run. Or maybe we can try to get some of the things started this GSoC session only.

I've been working on both Django and React for a very long time now, which makes me competent enough to get into this project.

Reactjs has some trending general practices that are considered ideal. I have plans to UPDATE the older practices which can cause exploits in the application.

Re-writing bad practices, like in this one, where they're trying to directly manipulate the DOM, which is not good.

As of **Django**, some **ANALYTICAL FEATURE** could be added, for eg: view count on the CRM page, or maybe TIME ON PAGE.

Use of **HITCOUNT library** would be a perfect shot on implementing this.
https://django-hitcount.readthedocs.io/en/latest/

This medium article, (published by ME a few months ago), shows how you can accurately get time spent on the page using Vanilla JS.
Algorithm to Calculate "Time On Page" in JavaScript using sendBeacon()

Even extending the idea further, a **PROFESSIONAL STATS ADMIN PAGE**, could be planned to create a professional dashboard allowing users to track the performance or even analytics without the usage or cost of Google Analytics.

Engagements during Summer

I have no commitments in the summer. I'll be staying back home for the most part of it. I have mentioned my typical working hours above and on an average will be able to spend 40 hours per week on the project.

About Me

- I'm a self-taught full-stack web developer and engineer.
- I've been coding since 2016, when I was in 8th standard, for 6+ years now.
- Python, JavaScript, C++, PHP are some of my primary languages I use.
- I primarily work in Django + React stack. MERN stack too.
- I've even published an article on how to integrate above two technologies – Integrate React with Django: The Best Full-Stack Web Integration
- As the above link suggests, I write tech articles on Medium.

- I've created many projects that can be found on my GitHub profile.

- NerdsWay – is one of my favourite projects. It is a blog/newsletter website where you can publish articles. It's top features are:
    - add tags, customise featured section
    - Time spent on page
    - Display article views. (No fake spam views)
    - Subscribe to Newsletter

- I've also mentored dozens of beginner programmers to get started with coding and open-source development.

Open-Source Development Experience

I was introduced to open source two years back. Some of the open source projects that I have worked on includes:

1. **Reactjs.org** : I worked on some documentation improvement and translation.
2. **Exec-Time**: I built an open-source tool useful for python developers to find the code execution time in milliseconds or seconds
   It is available on PYPI and has 1K+ downloads.
3. **Zeus-Uploader-Automation**: I built an open-source tool to automate the manual hassle of downloading the *BIOMETRIC machine data* and uploading it to the cloud server.
4. **Proton**: I built an open-source desktop assistant which can do ALMOST everything that Google Assistant can do. It's very easy to set up and really helpful in day-to-day use. *My friends and I use it everyday on a regular basis.*
5. **Cloud-CV** : I have been an active contributor to CloudCV. I've made changes in frontend and SEO.
   Also did a LOTS and LOTS of code reviews.
6. Apart from that I'm also very active on **StackOverflow**, ready to answer every question under my umbrella.
7. Other than this, my other projects and coding experience can be found on my github page.

Work/Internship/Academic Experience

- ## Code Jam 2022
  I participated in Google Code Jam 2022. I worked on the practice and official round problems.

- ## Zeus Technologies, New Delhi - 2021
  ### Software Engineer Internship
    - Built and maintained various company's websites, including SPA.
    - Work with senior developers to manage cloud API in backend.
    - Complete tasks for front end public and internal websites as well as challenging back-end server code.
    - Automating tasks on the backend using CRON jobs.

- ## Paramount International School, New Delhi
  ### Synergy 2020 - Presenting start-up ideas
  In October 2020, I participated and secured 2nd position in a competition, presenting a start-up idea as per the curricular fest organised by CBSE (National board of education).
    - LiveBit: A social-networking platform live on the intranet of Paramount International School where people can socialise with their friends.
    - Front end was mainly in javascript and CSS and the backend is written in Django. It involves the use of AJAX. The media files were stored on cloud using APIs.