

Auditing Content in Wagtail

Chuck Sebian-Lander and Will Barton (he/him)



Disclaimer

This presentation is being made by a Consumer Financial Protection Bureau representative on behalf of the Bureau. It does not constitute legal interpretation, guidance, or advice of the Consumer Financial Protection Bureau. Any opinions or views stated by the presenter are the presenter's own and may not represent the Bureau's views.

We're from the CFPB, a US government agency set up after the 2008 financial crisis to implement and enforce federal consumer financial law and ensure markets are fair, transparent, and competitive. Because of that, we have to give this disclaimer.

We're open source and work in the open

<https://github.com/cfpb/consumerfinance.gov/>



Just as a quick note, all of our code for our Django and Wagtail website is open source on GitHub, as are a series of libraries we maintain. We also work entirely in the open, in these repositories, for better and worse.

Eight+ years of Wagtail

Some history



For a little bit of history, we've been a user of Wagtail for a little over eight years now,

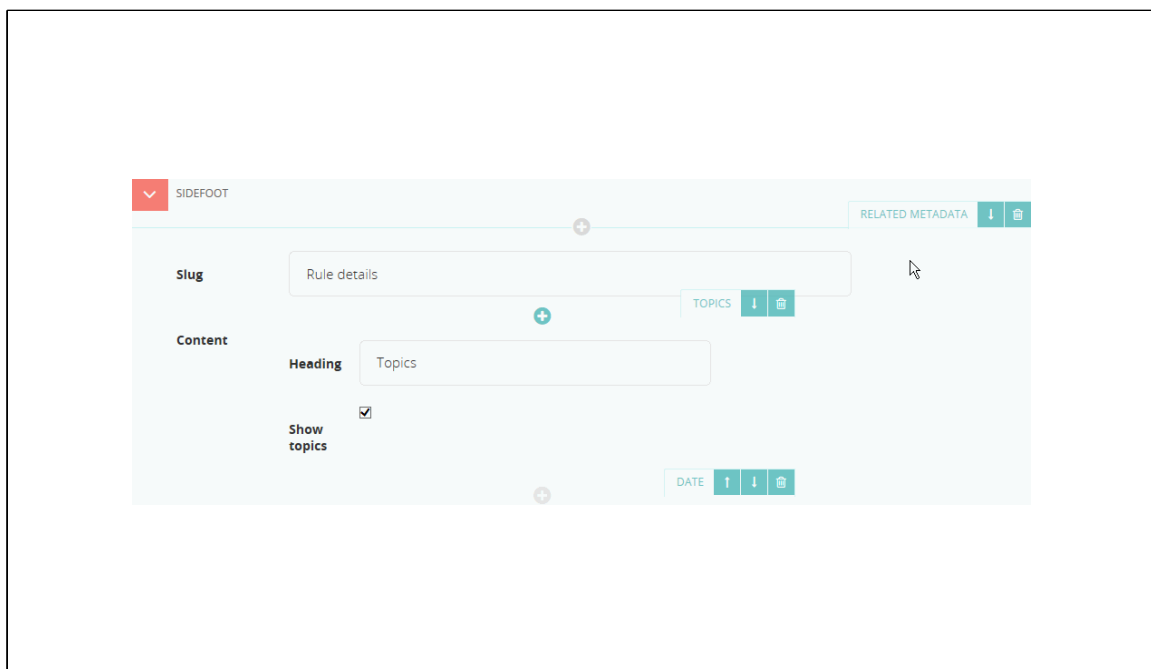
Wagtail #987

 Merged kave merged 3 commits into [cfpb:flapjack](#) from [kurtrwall:wagtail](#)  on Sep 21, 2015

```
requirements/base.txt
@@ -6,3 +6,4 @@ python-dateutil==2.1
6 6 requests==2.7.0
7 7 git+https://github.com/rosskarchner/govdelivery.git#egg=govdelivery
8 8 icalendar==3.9.0
9 + wagtail==1.1
```

We adopted Wagtail at version 1.1 in September of 2015. The site based on Wagtail launched to the public in May 2016.

It had to accommodate data from a prior WordPress site that was hastily assembled when the CFPB was established in 2012, and which grew unwieldy with a huge number of custom fields.



Wagtail let us streamline that kind of page architecture for content managers, where a limited number of page types can potentially meet many needs based on which fields are selected in a StreamField. Content managers have options while also limiting the number of unused fields they have to wade through in the admin.

At least, that was the theory.



With this powerful new CMS and its flexibility, we could do anything.

As designers, developers, content managers, etc, when the bureau identified needs the public had, we helped meet those needs in as many ways as we could.

```

class FullWidthText(blocks.StreamBlock):
    content = v1_blocks.UnescapedRichTextBlock(icon="edit")
    content_with_anchor = molecules.ContentWithAnchor()
    heading = v1_blocks.HeadingBlock(required=False)

class RegulationLandingPage(ShareableRoutablePageMixin, CFGOVPage):
    """Landing page for eregs."""

    cta = molecules.CallToAction()
    related_links = molecules.RelatedLinks()
    reusable_text = v1_blocks.ReusableTextChooserBlock("v1.ReusableText")
    reusable_notification = v1_blocks.ReusableNotificationChooserBlock(
        "v1.ReusableNotification"
    )
    class RegulationsListingFullWidthText(organisms.FullWidthText):
        regulations_list = RegulationsList()
    email_signup = v1_blocks.EmailSignUpChooserBlock()
    well = Well()

class Meta:
    icon = "edit"
    template = "v1/includes/organisms/full-width-text.html"

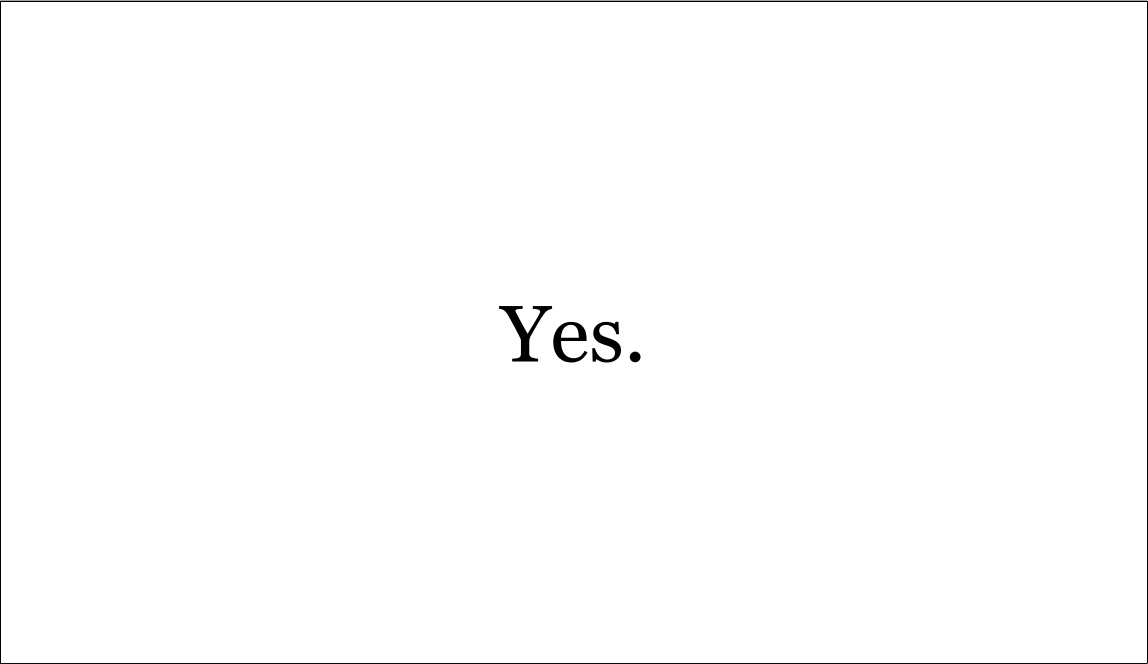
```

Need a new block or five added to a StreamField? Done. (ADVANCE)

Need a new page type for this particular type of content? Done. (ADVANCE)

Need a new stream block that's nearly like another one but not quite for use on a single page type? Done. (ADVANCE)

And if you saw Michael's talk yesterday morning, this probably sounds familiar!



Yes.

Should we have used this power to add all of this? Yes. (ADVANCE)

I want to pause here, because I don't want this talk to devolve into condescension of our past selves. I'm not here to tell you this was a mistake, to learn from it, and limit what you implement.

Trust that the decisions you're making right now are the best ones you can make now. Future-you might make a different decision based on extra context that now-you doesn't have, and that's fine.

But that doesn't mean we didn't end up with a complex pile of content and code.

Complexity doesn't like to hide



Welcome to the consumerfinance.gov Wagtail CMS

Chuck Sebian-Lander



9,180
Pages



2,582
Images



7,546
Documents

Consumerfinance.gov is a very big website! (This doesn't include several non-Wagtail pages, or pages generated dynamically such as certain search result pages or pages generated based on snippets)

Many pages? *Many* page types

- Over 30 page types
 - Frequently used standards (Blog, Newsroom)
 - Minor derivations of other page types
 - One-offs for special pages
- Legacy content that pre-dates Wagtail
- If you don't know, you don't know: institutional knowledge and internal naming conventions accumulated since 2016
- Complexity driven in part by a separation between "content managers" and "content owners" (i.e. the writers are not in the CMS)



Create a page in CFGov

Choose which type of page you'd like to create.

1 Activity top page Right hand column, no left hand column. Use if children should be searchable using standard search filter module.	Pages using Activity top page
2 Answer landing page Right hand column, no left hand column. Use if children should be searchable using standard search filter module.	Pages using Answer landing page
3 Answer page Right hand column, no left hand column. Use if children should be searchable using standard search filter module.	Pages using Answer page
4 Answer results page Right hand column, no left hand column. Use if children should be searchable using standard search filter module.	Pages using Answer results page
5 Blog page Right hand column, no left hand column. Use if children should be searchable using standard search filter module.	Pages using Blog page
6 Browse filterable page Left hand navigation, no right hand column. Use if children should be searchable using standard search filter module.	Pages using Browse filterable page
7 Browse page Left hand navigation, no right hand column. Use if children should be searchable using standard search filter module.	Pages using Browse page
8 College costs page Right hand column, no left hand column. Use if children should be searchable using standard search filter module.	Pages using College costs page
9 Document detail page Right hand column, no left hand column. Use if children should be searchable using standard search filter module.	Pages using Document detail page
10 Enforcement action page Right hand column, no left hand column. Use if children should be searchable using standard search filter module.	Pages using Enforcement action page
11 Enforcement actions filter page Left hand navigation, no right hand column. Use if children should be searchable using standard search filter module.	Pages using Enforcement actions filter page
12 Event archive page Left hand navigation, no right hand column. Use if children should be searchable using standard search filter module.	Pages using Event archive page
13 Event page Right hand column, no left hand column. Use if children should be searchable using standard search filter module.	Pages using Event page
14 Fiq content page Right hand column, no left hand column. Use if children should be searchable using standard search filter module.	Pages using Fiq content page
15 Form explainer page Right hand column, no left hand column. Use if children should be searchable using standard search filter module.	Pages using Form explainer page
16 Funds historic data page Right hand column, no left hand column. Use if children should be searchable using standard search filter module.	Pages using Funds historic data page
17 Home page Right hand column, no left hand column. Use if children should be searchable using standard search filter module.	Pages using Home page
18 Job listing page Right hand column, no left hand column. Use if children should be searchable using standard search filter module.	Pages using Job listing page
19 Low cost/low cost page Right hand column, no left hand column. Use if children should be searchable using standard search filter module.	Pages using Low cost/low cost page
20 Low cost/low cost page Right hand column, no left hand column. Use if children should be searchable using standard search filter module.	Pages using Low cost/low cost page

12

Discrepancies arise with:

- Nomenclature ("Newsroom page" vs. press release, director's remarks, etc)
- Blocks and layout ("well" vs. callout, table vs. info unit group)
- Static vs. dynamic functionality ("related posts" blocks that are sometimes static; filterable search results)

Redundant redundancies department: More content means *more of the same* content

The image shows two overlapping browser windows from the website consumerfinance.gov. The top window displays a page titled "Reverse mortgage" with a sub-header "Reverse mortgage" and a paragraph explaining that a reverse mortgage allows homeowners age 62 or older to borrow against their home equity. The bottom window displays a page titled "What is a reverse mortgage?" with a sub-header "What is a reverse mortgage?" and a paragraph explaining that a Home Equity Conversion Mortgage (HECM) is a special type of home loan only for homeowners who are 62 and older. Both pages include a "LAST REVIEWED: AUG 28, 2023" notice and a "cfpb" logo. The bottom window also includes a link to "Consider before applying for a reverse mortgage."

consumerfinance.gov/consumer-tools/mortgages/answers/key-terms/#reverse-mortgage

Reverse mortgage

A reverse mortgage allows homeowners age 62 or older to borrow against their home equity. It is called a "reverse" mortgage because, instead of making payments to the lender, the lender pays you. The money you receive, and the interest charged on the loan, are added to your loan each month. Most reverse mortgages today are Home Equity Conversion Mortgages.

[Consider before applying for a reverse mortgage.](#)

← → ↺ 🏠 🌐 consumerfinance.gov/ask-cfpb/what-is-a-reverse-mortgage-en-224/

LAST REVIEWED: AUG 28, 2023

What is a reverse mortgage?

English | [Español](#)

A Home Equity Conversion Mortgage (HECM), the most common type of reverse mortgage, is a special type of home loan only for homeowners who are 62 and older.

cfpb Consumer Financial Protection Bureau

13

CF.gov covers a wide range of topics, but also covers many of the same topics with various pieces of content intended for similar audiences

Not identical enough for a snippet, but too close not to want both updated in tandem

For example: Finding and replacing content site-wide

Circa 2022: Update language re: housing counselors, agencies, and the U.S. Department of Housing and Urban Development (HUD)

The screenshot shows the CFPB website with the following elements:

- Header: "An official website of the United States government" and language options (Español, 中文, Tiếng Việt).
- Logo: "cfpb Consumer Financial Protection Bureau".
- Navigation: "Consumer Education", "Rules & Policy", "Enforcement", "Compliance".
- Section: "Find a housing counselor".
- Text: "Housing counselors throughout the country can provide advice on buying a home, renting, defaults, forbearances, foreclosures, and credit issues. This list will show you several approved agencies in your area. The counseling agencies on this list are approved by the U.S. Department of Housing and Urban Development (HUD) and they can offer independent advice, often at little or no cost to you. There is also a [list of nationwide HUD-approved counseling agencies](#).".
- Text: "Using the search box below, you can find one near you. **Not every housing counselor offers all services, so please look at the list of services offered by each agency.**"
- Form: "Search by ZIP code:" with a text input field "Please enter a five-digit ZIP code" and a "Find a counselor" button.
- Footer: "cfpb Consumer Financial Protection Bureau" and page number "14".

HUD approves housing counselor *agencies*, not the counselors themselves, but CF.gov language – in blog posts, on consumer fact sheets, on random sidebar callouts – referred consistently to “HUD-approved housing counselors”

The ask, from Bureau stakeholders: Find everywhere that says “HUD-approved housing counselors” and change it to “HUD-approved housing counselor agency”

Needs to include variations in phrasing (e.g. “HUD-approved counselors”)

Needs to include snippets, plain text fields, rich text fields

Needs to cover every part of every page on the website: Sidebars, call-outs, search descriptions, table cells, alternative text for images

Where do you even start?



This is a daunting problem: you've got complexity built up over years and years, you know it's there, but how do you start interrogating it to know what patterns might exist?

Identifying problem areas and what we need to audit

- We have a lot of page types
 - How many pages do we have of each type?
 - How many pages are only ever children of a specific page?
- We have a lot of blocks
 - How many pages use each block?
 - How many blocks are available on a page type but not used?
- We have some places with raw HTML in content fields
 - Uh...
 - We'll come back to this one.

We identified a few specific problem areas we wanted to look at, and what we needed an audit to tell us how to think about solving them. And that's what the rest of this will be about: how we did the audit, not CFPB-specific solutions to CFPB-specific problems.

(ADVANCE)

We have a lot of page types, and as you can tell from Chuck's screenshot a few slides ago, this clutters up the "Create new page" listing for content managers.

(ADVANCE)

We have a lot of blocks available on those page types

(ADVANCE)

And, we have some places where raw HTML was entered into content fields because those available blocks didn't quite do what someone needed them to do

(ADVANCE) (ADVANCE)

 Page types usage				
Type	App	Pages	Last edited page	Last edit
Answer Page	ask_cfpb.answerpage	1834	¿Cuáles son las señales de advertencia típicas de posibles fraudes y estafas?-es-2094	19 hours ago
Document Detail Page	v1.documentdetailpage	1816	Request for Information regarding Mortgage Closing Costs	38 minutes ago
Newsroom Page	v1.newsroompage	1139	CFPB Launches Inquiry into Junk Fees in Mortgage Closing Costs	38 minutes ago
Blog Page	v1.blogpage	970	Banks' responsibility for scams	23 hours ago
Legacy Newsroom Page	v1.legacynewsroompage	779	Written Testimony of Rohit Chopra before the Committee on the Budget	6 months ago
Legacy Blog Page	v1.legacyblogpage	661	When your child learns self-control, it helps their financial future, too	1 month ago

So, page types. This is somewhat simple, in that Wagtail has a built in Page Types Usage report.

This tells us exactly how many pages we have of each type.

(ADVANCE)

Note our “legacy newsroom” and “legacy blog” pages here — we’ll revisit these two later

Hmda Historic Data Page	hmda.hmdahistoricdatapage	1	Download Historic HMDA Data	1 year ago
College Costs Page	paying_for_college.collegecostspage	1	Your financial path to graduation	1 year ago
Regulation Landing Page	regulations3k.regulationlandingpage	1	Interactive Bureau Regulations	2 years ago
Regulations Search Page	regulations3k.regulationssearchpage	1	Search regulations	1 year ago
Tdp Activity Search Page	teachers_digital_platform.activityindexpage	1	Search for activities	11 months ago
Activity Log Page	v1.activitylogpage	1	Recent updates	1 year ago
Enforcement Actions Filter Page	v1.enforcementactionsfilterpage	1	Enforcement Actions	3 months ago
Event Archive Page	v1.eventarchivepage	1	Archive of Past Events	1 year ago
Newsroom Landing Page	v1.newsroomlandingpage	1	Newsroom	1 year ago
Research Hub Page	v1.researchhubpage	1	Research Hub	1 year ago

18

Another thing this report can show us is which pages are only ever used once.

One thing you'll notice about these one-time use page types is that they are almost all landing pages of some description, which suggests that, for example, our Newsroom Page type could be limited to being a subpage type of Newsroom Landing Page.

Generally too, page types are the easiest thing here to audit — they're Django models and we can easily construct querysets to look into them without a lot of effort.

Identifying problem areas and what we need to audit

- ~~We have a lot of page types~~

- We have a lot of blocks

Ok, so that gives us some basic page type information we can act on, built-in, no problem.

(ADVANCE)

That's great. What's next?



When we start considering which blocks are used on those page types, and in which stream fields... then we get into the weeds of Wagtail StreamFields.

Like I said earlier too, StreamFields were one of the major draws of Wagtail to us, in the way they permit a lot of flexibility for page content without being a UX nightmare in the admin. Let's look into an example of how we're using them.

What to do if you're struggling to pay rent



If you're having trouble paying for rent and utilities, you're not alone.

- [Get help paying rent and bills](#)
- Get year-round help with utility bills by [contacting your local Low Income Home Energy Assistance Program \(LIHEAP\) office](#) or calling the National Energy Assistance Referral Hotline at [\(866\)-674-6327](#)

Talk to your landlord about a repayment agreement

In addition to applying for rental assistance, now may be the time to start a conversation with your landlord if you're struggling to pay your full rent each month or need to repay past due rent.

[Learn about options and how to start the conversation](#)

<https://www.consumerfinance.gov/housing/housing-insecurity/help-for-renters/>

This is a part of a page where we have help for renters who may be struggling to make payments.

Content

+

Info unit group

Format

Choose the number and width of info unit columns.

25/75

+

Info units

Image Upload

165 × 165

cfpb_renters_overdue-rent.png

Body

B I X¹ H2 H3 H4 H5 ☰ ☷ “ — ↶ ⚓ 🔗 📎 🖼️ 💬

If you're having trouble paying for rent and utilities, you're not alone.

- 🔗 [Get help paying rent and bills](#)
- Get year-round help with utility bills by 🔗 [contacting your local Low Income Home Energy Assistance Program \(LIHEAP\) office](#) or calling the National Energy Assistance Referral Hotline at 🔗 [\(866\)-674-6327](#)

🔗 [Talk to your landlord about a repayment agreement](#)

In addition to applying for rental assistance, now may be the time to start a conversation with your landlord if you're struggling to pay your full rent each month or need to repay past due rent.

🔗 [Learn about options and how to start the conversation](#)

In the Units

The V these below

ains Info

reen early

```
class InfoUnitGroup(blocks.StructBlock):
    format = blocks.ChoiceBlock(
        choices=[
            ("50-50", "50/50"),
            ("33-33-33", "33/33/33"),
            ("25-75", "25/75"),
        ],
        default="50-50",
        label="Format",
        help_text="Choose the number and width of info unit columns.",
    )
    info_units = blocks.ListBlock(molecules.InfoUnit(), default=list())
```

Let's look at our Info Unit Group. It's a Struct Block that defines some specific fields, including a list of Info Unit blocks.

```
class InfoUnit(blocks.StructBlock):
    image = atoms.ImageBasic(
        required=False,
    )

    heading = HeadingBlock(required=False, default={"level": "h3"})

    body = blocks.RichTextBlock(blank=True, required=False)
    links = blocks.ListBlock(atoms.Hyperlink(), required=False)

    class Meta:
        icon = "image"
        template = "v1/includes/molecules/info-unit.html"
```

The Info Unit block is also a struct block, and includes our image block, a Rich Text Block, and a list of our hyperlinks. So, that's the hierarchy in code, what does it look like in the database?

```
[
  {
    "id": "1ee4f9cc-1581-4a26-8fe7-aae8e9e93580",
    "type": "info_unit_group",
    "value": {
      "info_units": [
        {
          "id": "0820f545-db19-45b3-a928-8c1f559e9cbd",
          "type": "item",
          "value": {
            "body": "<p data-block-key=\\\"yahat\\\">If you\\u2019re having trouble paying for rent and utilities,",
            "image": {
              "alt": "",
              "upload": 3280
            },
            "links": [],
            "heading": {
              "icon": "",
              "text": "",
              "level": "h3"
            }
          }
        }
      ]
    }
  }
]
```

Because Wagtail stream fields are implemented as JSON columns in the database, when we look into the database at the content field, we see something like this in the “content” column.

Given these data structures then, if we asked, how many pages are using our Image Basic blocks, how would we answer this?

```

class ContentImage(blocks.StructBlock):
    image = atoms.ImageBasic()
    image_width = blocks.ChoiceBlock(
        choices=[
            ("full", "Full width"),
            (470, "470px"),
            (270, "270px"),
            (170, "170px"),
        ],
        default="full",
    )
    image_position = blocks.ChoiceBlock(
        choices=[("right", "right"), ("left", "left")],
        default="right",
        help_text="Does not apply if the image is full-width",
    )
    text = blocks.RichTextBlock(required=False, label="Caption")
    is_bottom_rule = blocks.BooleanBlock(
        required=False,
        default=True,
        label="Has bottom rule line",
        help_text="Check to add a horizontal rule line to bottom of inset.",
    )

```

Our Info Unit struct block is not the only place the image block appears, it also appears in our Featured Content block, for example. And many others, which are then included in other blocks themselves. What if we want to count all of them?

Querying **JSONField** ¶

Lookups implementation is different in **JSONField**, mainly due to the existence of key transformations. To demonstrate, we will use the following example model:

...

Multiple keys can be chained together to form a path lookup:

```
>>> Dog.objects.filter(data__owner__name="Bob")
<QuerySet [<Dog: Rufus>]>
```

If the key is an integer, it will be interpreted as an index transform in an array:

```
>>> Dog.objects.filter(data__owner__other_pets__0__name="Fishy")
<QuerySet [<Dog: Rufus>]>
```

So, both Django itself and PostgreSQL have some really nice JSON support built in. For Django, if we know the JSON path, we can query into JSON fields directly. But the problem with that is, stream fields don't have a concrete schema. The JSON path lookup is what we want to discover, but while we *could theoretically* know all possible permutations of it, that won't help us here.

```

WITH RECURSIVE blocks AS (
    SELECT page_id, index::text, block FROM (
        SELECT
            {primary_key_field} AS page_id,
            {streamfield_name}::jsonb AS data FROM {database_table}
        ) x
    LEFT JOIN LATERAL jsonb_array_elements(x.data)
    WITH ORDINALITY AS a (block, index)
    ON true
    UNION ALL
    SELECT
        page_id,
        index || '.' || COALESCE(obj_key, (arr_key - 1)::text),
        COALESCE(arr_block, obj_block)
    FROM blocks
    LEFT JOIN LATERAL
        jsonb_array_elements(
            CASE jsonb_typeof(block)
                WHEN 'array' THEN block
            END
        )
        WITH ORDINALITY AS a(arr_block, arr_key)
    ON jsonb_typeof(block) = 'array'
    LEFT JOIN LATERAL
        jsonb_each(
            CASE jsonb_typeof(block)
                WHEN 'object' THEN block
            END
        )
        AS o(obj_key, obj_block)
    ON jsonb_typeof(block) = 'object'
    WHERE arr_key IS NOT NULL OR obj_key IS NOT NULL
)

```

PostgreSQL has some really very nice JSON support built in. We can build a temporary table via lateral joins of all matching blocks within the JSON and query that... and then maybe build a Django QuerySet around it.



Unfortunately, we couldn't make this work the way we wanted to in a Django QuerySet. We got stuck on this notion that we had to do this in the database, because that's the only way it could be reasonably performant, right?

Instead, we stepped back and started working with idea that maybe it's okay if this requires some time processing in Python. For a team of Python developers, this also makes more sense, because it'll be more immediately comprehensible by our colleagues as well.

Fundamentally there are two things we want to know: what blocks are available on each page type, and what blocks are in use?

```

# Traverse a stream field and yield back each available block type
def traverse_streamblock(page_model, block, parent=None):
    """Walk model stream block objects to get initial AuditedBlocks"""
    block_name = block.name if block.name != "" else "item"
    audited_block = AuditedBlock(
        page_model=dotted_name(page_model),
        field=None,
        path=(parent + "." + block_name if parent is not None else block_name),
        block=dotted_name(block.__class__),
        pages=[],
    )

    yield audited_block

# If this is a StreamBlock or StructBlock it'll have child blocks
if isinstance(block, (StreamBlock, StructBlock, TypedTableBlock)):
    for child_block_field_name in block.child_blocks:
        yield from traverse_streamblock(
            page_model,
            block.child_blocks[child_block_field_name],
            parent=audited_block.path,
        )

elif isinstance(block, ListBlock):
    yield from traverse_streamblock(
        page_model,
        block.child_block,
        parent=audited_block.path,
    )

else:
    pass

@dataclass
class AuditedBlock:
    page_model: type
    field: str
    path: str
    block: type
    pages: list
    total_occurrences: int = 0
    pages_count: int = 0
    pages_live_count: int = 0
    pages_in_default_site_count: int = 0

# Traverse a stream field's value and yield back each block type in use
def traverse_streamvalue(value, parent=None):
    """Walk model stream value objects to get AuditedBlocks in-use"""

    # This is a block type
    if isinstance(value, BoundBlock):
        block_name = value.block.name if value.block.name != "" else "item"
        path = parent + "." + block_name if parent is not None else block_name
        yield path
        yield from traverse_streamvalue(value.value, parent=path)

    # This is a StructValue
    elif isinstance(value, StructValue):
        for child in value.bound_blocks.values():
            yield from traverse_streamvalue(child, parent=parent)

    elif isinstance(value, ListValue):
        for child in value.bound_blocks:
            yield from traverse_streamvalue(child, parent=parent)

    elif isinstance(value, TypedTable):
        for row_index, row in enumerate(value.rows):
            for column_index, child in enumerate(row):
                yield from traverse_streamvalue(child, parent=parent)

    # This is a sequence of blocks
    elif isinstance(value, StreamValue):
        for child in value:
            yield from traverse_streamvalue(child, parent=parent)

```

We ended up with two recursive functions to walk through first the Stream Block to gather all possible paths to the blocks themselves on each Stream Field on our models, and then to walk through Stream Values on page instances to find which of those are being used.

And we use a dataclass to hold the results.

I know this is small on the screen, don't worry about the following the code too closely right now — this is all open source as a library with management commands to run the audits.

```
./manage.py block_usage > block_usage_audit.csv  
./manage.py block_usage --pagetype myapp.PageWithContent.content
```

<https://github.com/cfpb/wagtail-content-audit>

So we've published a library with the code I showed above and a management command to run it. It outputs CSV. It can be run on everything,

or you can filter for a specific page model and streamfield.

```
./manage.py block_usage > block_usage_audit.csv

./manage.py block_usage --pagetype myapp.PageWithContent.content

from wagtail_content_audit.query import BlockUsageQuerySet

qs = BlockUsageQuerySet().filter(
    page_model="myapp.PageWithContent",
    field="content"
)
```

<https://github.com/cfpb/wagtail-content-audit>

That filtering is possible because we've implemented this as a QuerySet-like object using the <https://github.com/wagtail/queryish> library, which is awesome, and a whole talk in itself. But that means you can also use it as such.

This also means it could be a Wagtail Report — this hasn't happened yet though.



1	Page Type	Field	Path	Block	Occurrence	Pages
269	v1.models.learn_page.LearnPage	content	full_width_text.image.image	v1.atomic_elements.atoms.ImageBasic	4	4
375	v1.models.learn_page.LearnPage	content	info_unit_group.info_units.item.image	v1.atomic_elements.atoms.ImageBasic	70	21
418	v1.models.learn_page.LearnPage	content	expandable_group.expandables.item.content.info_unit_group.info_units.item.image	v1.atomic_elements.atoms.ImageBasic	2	1
562	v1.models.learn_page.LearnPage	content	expandable.content.info_unit_group.info_units.item.image	v1.atomic_elements.atoms.ImageBasic	0	0
688	v1.models.learn_page.LearnPage	content	faq_group.faq_items.item.answer.full_width_text.image.image	v1.atomic_elements.atoms.ImageBasic	0	0
760	v1.models.learn_page.LearnPage	content	faq_group.faq_items.item.answer.info_unit_group.info_units.item.image	v1.atomic_elements.atoms.ImageBasic	32	4
810	v1.models.learn_page.DocumentDetailPage	content	full_width_text.image.image	v1.atomic_elements.atoms.ImageBasic	11	6
988	v1.models.learn_page.DocumentDetailPage	content	expandable.content.info_unit_group.info_units.item.image	v1.atomic_elements.atoms.ImageBasic	0	0
1122	v1.models.learn_page.DocumentDetailPage	content	expandable_group.expandables.item.content.info_unit_group.info_units.item.image	v1.atomic_elements.atoms.ImageBasic	0	0
1178	v1.models.learn_page.EventPage	persistent_block	image.image	v1.atomic_elements.atoms.ImageBasic	4	1
1221	v1.models.blog_page.BlogPage	content	full_width_text.image.image	v1.atomic_elements.atoms.ImageBasic	282	211
	v1.models.blog_page.BlogPage	content	info_unit_group.info_units.item.image	v1.atomic_elements.atoms.ImageBasic	133	46
	v1.models.blog_page.BlogPage	content	expandable.content.info_unit_group.info_units.item.image	v1.atomic_elements.atoms.ImageBasic	0	0

We output the overall result of this to CSV, and do some filtering in Excel, and we can see where this Image Basic block is used!

A couple of useful things to highlight here: it's available in six places in the content field of our "Learn Page". But it's not used at all in two of those. This gives us something to investigate — is that block needed at all in those places? We don't know, but now we can check! And if we can remove them, then that simplifies the options available on those blocks in the admin; we can stop presenting content managers with options they don't use.

The other thing to note here, the "Path" column, we've tried to make sure the path that results here matches the path that Wagtail's new streamfield migrations use, so that hopefully it makes writing data migrations based on this that little bit more straight forward.



(Pause)

Ok, so, how are we doing? This is exciting right?

We were certainly excited. We can pull our useful information about our deeply nested blocks and potentially act on it to make our users lives easier. That's what it's all about right?

CHUCK: This is also a great example of using a large, legacy database of content to your advantage – the realities of usage trends are baked in at this point. It's the most organic user testing you can have. (More on that in a moment...)

Identifying problem areas and what we need to audit

- ~~We have a lot of page types~~
- ~~We have a lot of blocks~~
- We have some places with raw HTML in content fields

Ok, so we've got our page types audited and we have information we can act on

(Advance)

we have our blocks audited and have information we can act on.

(Advance)

And there was this little bullet right here. Yeah.

Raw HTML

Sometimes our CMS doesn't do what our users need it to.



Maybe we don't have a block for that, or maybe the block doesn't have the right options. And maybe we don't have the capacity to add that option, it might be way down the priority list.

What's a user to do?

So, we were able to identify at a glance a few places where raw HTML was being used.

Uses of raw HTML

- Wagtail's own RawHTMLBlock
- Raw HTML in text fields
- Raw HTML in rich text fields



cfpb Consumer Financial
Protection Bureau

(ADVANCE)

We have places where we include Wagtail's RawHTMLBlock where we know we might need to use raw HTML. This has also become a way for us to embed React components, which is also probably not the best approach, and a whole other talk.

(ADVANCE)

We had raw HTML in some of our plain text fields... not great.

(ADVANCE)

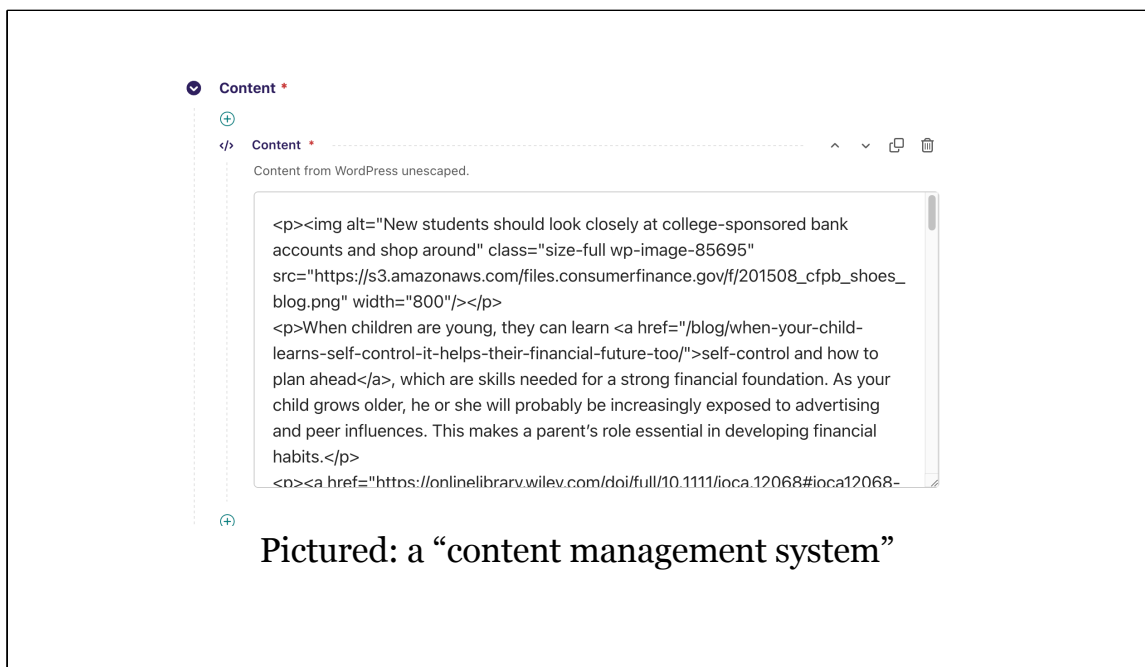
And we had raw HTML in our rich text fields.

(ADVANCE)

(PAUSE)

But what are some examples? After all, we have users who clearly have needs the

CMS isn't meeting, so what are they?



Pictured: a “content management system”

“Legacy” content – imported into Wagtail as raw HTML when we migrated from WordPress in 2016

Not the only example we had of using raw HTML in pages, but certainly the most egregious—and there are nearly 1,500 of these on CF.gov!

We have asked, and have been told we cannot remove them, so: how do we deal with this?

Label

```
<h3 style="padding-top: 15px;" >Mortgages</h3>
```

Heading

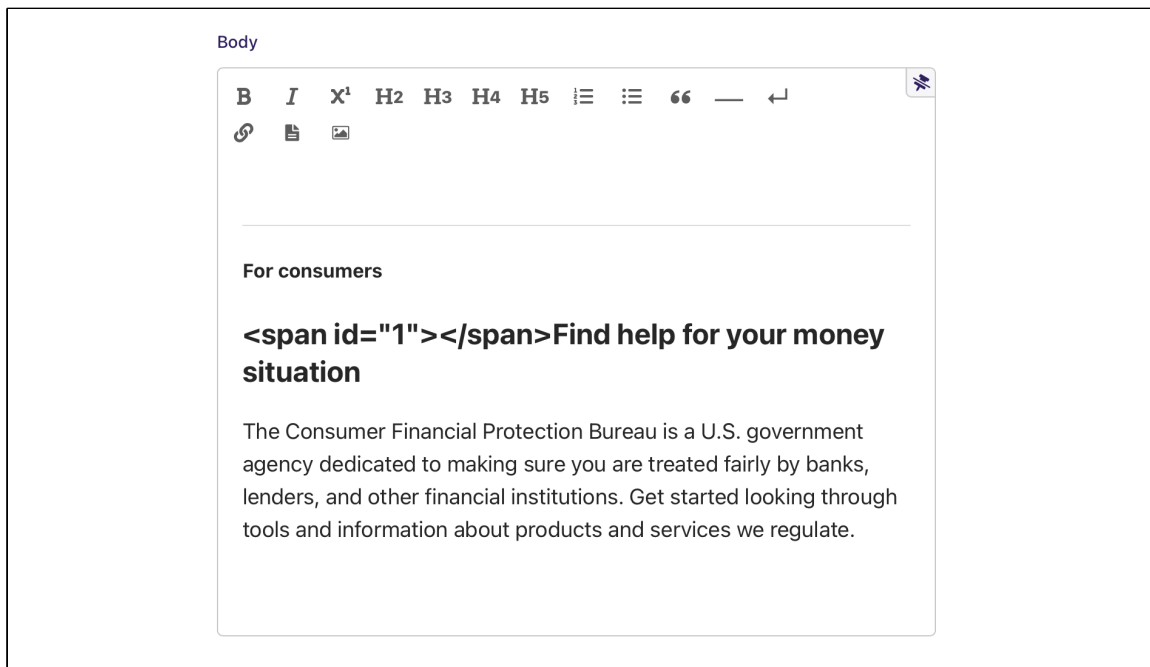
```
<strong>Learn more: Navigating financial rules and regulations</stro
```

Label

```
<h3 style="padding-top: 15px;" ><svg xmlns="http://www.w3.org
```

In our plaintext fields, we see examples like:

- Overriding the heading level and adding padding
- Adding strong and emphasis tags
- And... uh.
- Ok, so we have some user needs we're not meeting



```
<p data-block-key="papkm"></p><hr/><h5 data-block-  
key="c97tn">For consumers</h5><h2 data-block-  
key="85m57">&lt;span id="1">&gt;&lt;/span>&Find  
help for your money situation</h2><p data-block-  
key="cug70">The Consumer Financial Protection Bureau is a  
U.S. government agency dedicated to making sure you are  
treated fairly by banks, lenders, and other financial  
institutions. Get started looking through tools and  
information about products and services we  
regulate.<br/><br/></p>
```

But wait, those of you who might be passingly familiar with Draftail and how rich text works. How does that even work? It shouldn't work. That'll just get stored as HTML entities!

In the database that field's content looks like this

(Pause)

Right, so... this doesn't answer the question: how does that even work?

```
# Render the template with the context
html = template.render(new_context)
unescape = HTMLParser.HTMLParser().unescape(html)
# Return the rendered template as safe html
return Markup(unescape)
```

Well, it works because we made a fateful decision to do this when rendering stream child blocks.

This was when we were on Wagtail 1.2, in 2015, well before the Wagtail site launched publicly.

This is well before a lot of Wagtail's now-built-in rendering options for stream child blocks, before `include_block`, etc.

This code existed in our render cycle until after we did the audit this presentation is about earlier this year.

```
html = template.render(new_context)
unesaped = HTMLParser.HTMLParser().unescape(html)
```

This is how HTML entities in our rich text field become actual HTML

```
unescape(html)
```

This is how our CMS users could put any HTML they wanted into our rich text blocks and it would render.



So... maybe the problem isn't just span tags that are anchors? We can provide anchors in the rich text editor, sure, but what else is in there? How do we even find out what's there, and in which rich text blocks, which can be in so many different blocks on a page?

Breathe.

We've already solved the problem of walking through blocks, identifying their specific path, etc. Maybe all we need to do is use that pattern, make it a queryish-queryset, like we did before, and add a search inside the fields along the way.

```

def search_blocks(pattern, value, path=None):
    if path is None:
        path = []

    if isinstance(value, BoundBlock):
        local_path = path + [value.block]
        yield from search_blocks(pattern, value.value, path=local_path)

    elif isinstance(value, StructValue):
        local_path = path
        for child in value.bound_blocks.values():
            yield @dataclass
            class PageMatch:
                page_model: type
                page: Page
                field_name: str
                stream_field_path: list
                block_type: type
                result_path: list
                matches: list
            elif isinstance
            for index,
            local_
            yield
            r"<[a-zA-Z]+((?!>).)*>,"
            elif isinstance
            for index,
            local_
            yield
            result_path: list
            matches: list
            elif isinstance
            for row_index, row in enumerate(value.rows):
                local_path = path + [row_index]
                for column_index, child in enumerate(row):
                    local_path = local_path + [column_index]
                    yield from search_blocks(pattern, child, path=local_path)
            else:
                matches = pattern.findall(str(value))
                if len(matches) > 0:
                    yield path, matches

```

We probably want to start by filtering the page query using Django's `iregex` filter. That way we're only operating on a set of pages that has a match to our regular expressions somewhere in the field we're searching.

Then if the field is a streamfield, we walk through it like we did with the block usage tool,

And again we're using a dataclass to hold the results.

Of course, we'll want a regular express we use to match HTML hiding in entities. It's a bit gnarly, but we'll gloss over that.

```
PageSearchQuerySet().filter(  
    page_model="myapp.PageWithContent",  
    field="content",  
    search=r"[tT]est",  
)
```

```
./manage.py page_search --pagetype myapp.PageWithContent.content -s '[tT]est'
```

<https://github.com/cfpb/wagtail-content-audit>

What if we make a `PageSearchQuerySet` to do that. We'll want to filter that.

Maybe by page model

And the field

And then we'd want to be able to give it a search string, maybe even a regular expression, if we could have everything we ever wanted.

Maybe with a management command, for fun.

And, through a lot of iteration, that's what we ended up with. This is also part of our content audit library.

Page ID	Page Type	Field	Field Type	Stream Field Path	Block Type	Result Path
16918	LearnPage	content	wagtail.fields.StreamField	expandable_group.body	wagtail.blocks.field_block.RichTextBlock	1.expandable_group.body
16918	LearnPage	content	wagtail.fields.StreamField	expandable_group.body	wagtail.blocks.field_block.RichTextBlock	2.expandable_group.body
16918	LearnPage	content	wagtail.fields.StreamField	expandable_group.body	wagtail.blocks.field_block.RichTextBlock	3.expandable_group.body
16918	LearnPage	content	wagtail.fields.StreamField	expandable_group.body	wagtail.blocks.field_block.RichTextBlock	4.expandable_group.body
16918	LearnPage	content	wagtail.fields.StreamField	expandable_group.body	wagtail.blocks.field_block.RichTextBlock	5.expandable_group.body
16918	LearnPage	content	wagtail.fields.StreamField	expandable_group.body	wagtail.blocks.field_block.RichTextBlock	6.expandable_group.body
16918	LearnPage	content	wagtail.fields.StreamField	expandable_group.body	wagtail.blocks.field_block.RichTextBlock	7.expandable_group.body
12471	LearnPage	content	wagtail.fields.StreamField	expandable_group.expandables.item.content.paragraph	wagtail.blocks.field_block.RichTextBlock	6.expandable_group.expandables.item.1.content.0.paragraph
13081	LearnPage	content	wagtail.fields.StreamField	expandable_group.expandables.item.content.paragraph	wagtail.blocks.field_block.RichTextBlock	0.expandable_group.expandables.item.1.content.0.paragraph
13081	LearnPage	content	wagtail.fields.StreamField	expandable_group.expandables.item.content.paragraph	wagtail.blocks.field_block.RichTextBlock	1.expandable_group.expandables.item.1.content.0.paragraph
17363	LearnPage	content	wagtail.fields.StreamField	expandable_group.body	wagtail.blocks.field_block.RichTextBlock	1.expandable_group.body
17374	LearnPage	content	wagtail.fields.StreamField	expandable_group.body	wagtail.blocks.field_block.RichTextBlock	1.expandable_group.body
17376	LearnPage	content	wagtail.fields.StreamField	expandable_group.body	wagtail.blocks.field_block.RichTextBlock	1.expandable_group.body
17357	LearnPage	content	wagtail.fields.StreamField	expandable_group.body	wagtail.blocks.field_block.RichTextBlock	1.expandable_group.body
17369	LearnPage	content	wagtail.fields.StreamField	expandable_group.body	wagtail.blocks.field_block.RichTextBlock	1.expandable_group.body
16014	LearnPage	content	wagtail.fields.StreamField	expandable_group.body	wagtail.blocks.field_block.RichTextBlock	2.expandable_group.body
4789	LearnPage	content	wagtail.fields.StreamField	expandable_group.body	wagtail.blocks.field_block.RichTextBlock	0.expandable_group.body
4789	LearnPage	content	wagtail.fields.StreamField	expandable_group.body	wagtail.blocks.field_block.RichTextBlock	2.expandable_group.body

And our results have one row per match, with similar fields to our block usage report earlier, except we also get the exact location in a streamfield

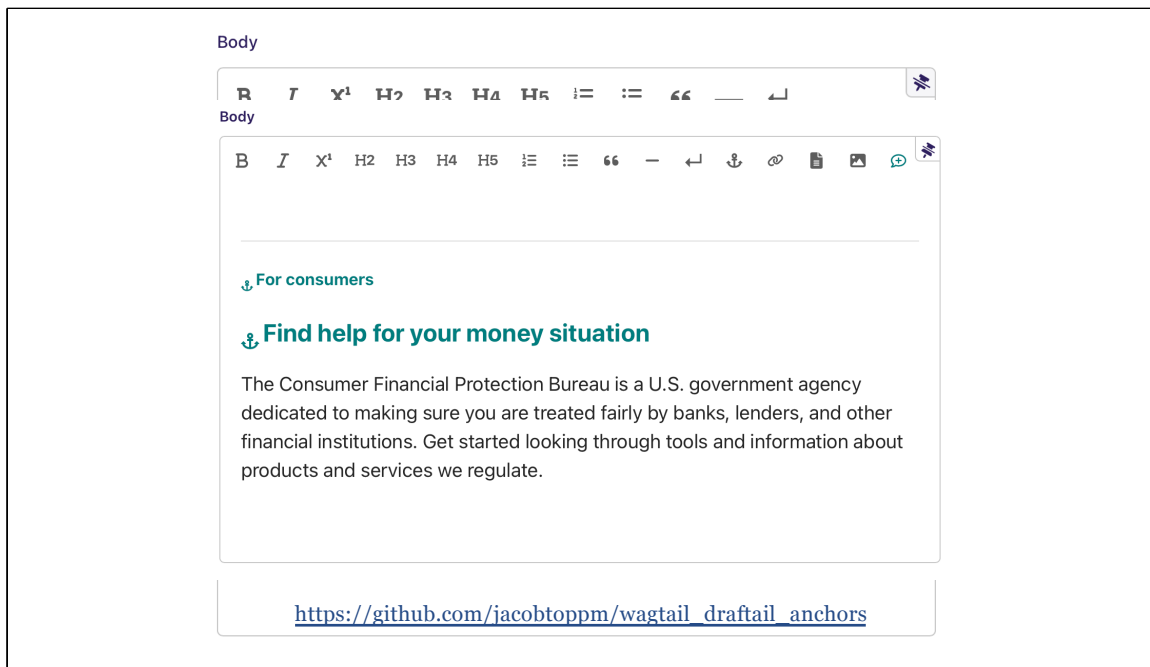
So, in the circled example, this match is in the 7th stream child, which is an expandable group, and the match is in the second item, and the first content paragraph.

This way we can open the page in the admin and pinpoint more easily which field is matching.

Remediating



Aside from some one-offs and things we could manually remediate that didn't need to be there we found two really broad needs our rich text editor wasn't meeting: anchors and inline SVGs.



So, there's a fantastic library, wagtail-drafttail-anchors, that we used to add anchor support in the rich text editor, and migrated our manual anchors to it.

We manually remediated a lot of other things we found, and we've isolated our SVG problem while we work on it.

Label

<h3 style="padding-top: 15px;" >Mortgages</h3>

Heading

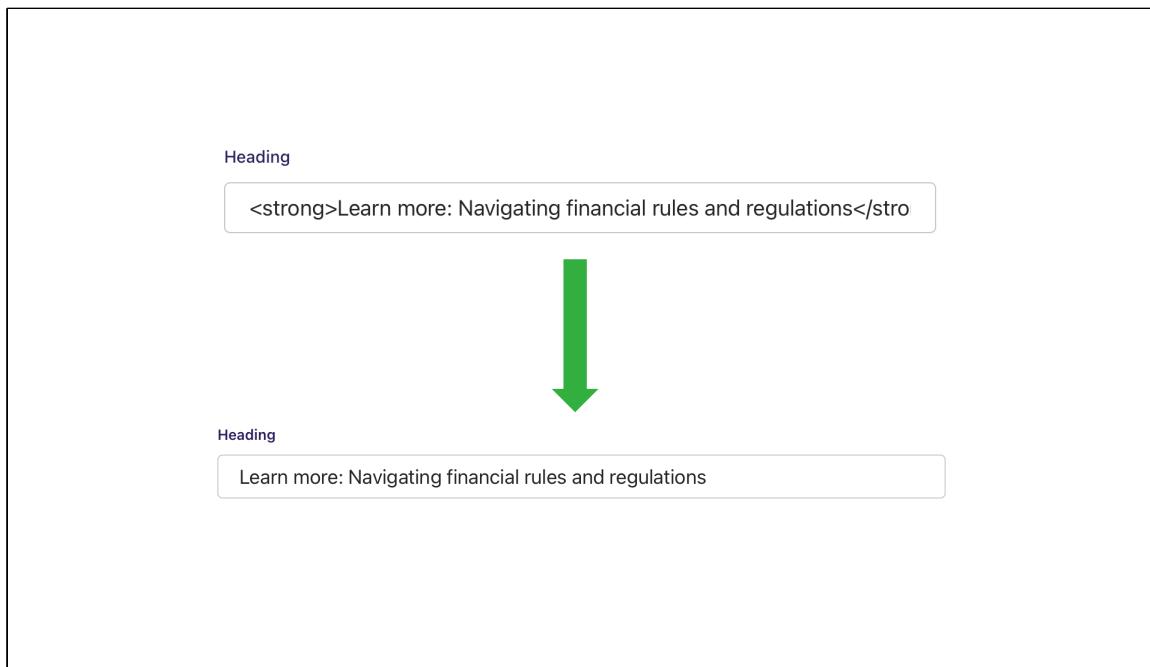
Learn more: Navigating financial rules and regulations</stro

Label

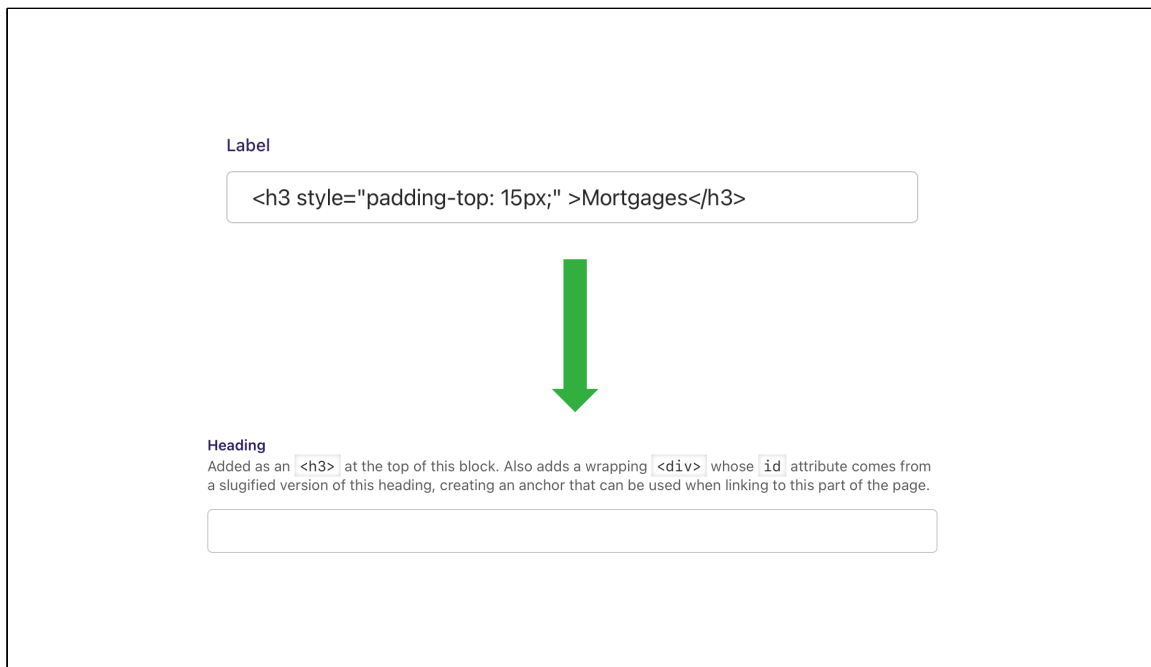
<h3 style="padding-top: 15px;" ><svg xmlns="http://www.w3.org

Coming back to these examples where we **just** have real HTML in a text field, we can use the same page search tool to identify blocks that have any HTML in them generally, filter out the raw HTML block and rich text blocks, which are the only ones that should be storing HTML, and then identify patterns in the rest of our blocks.

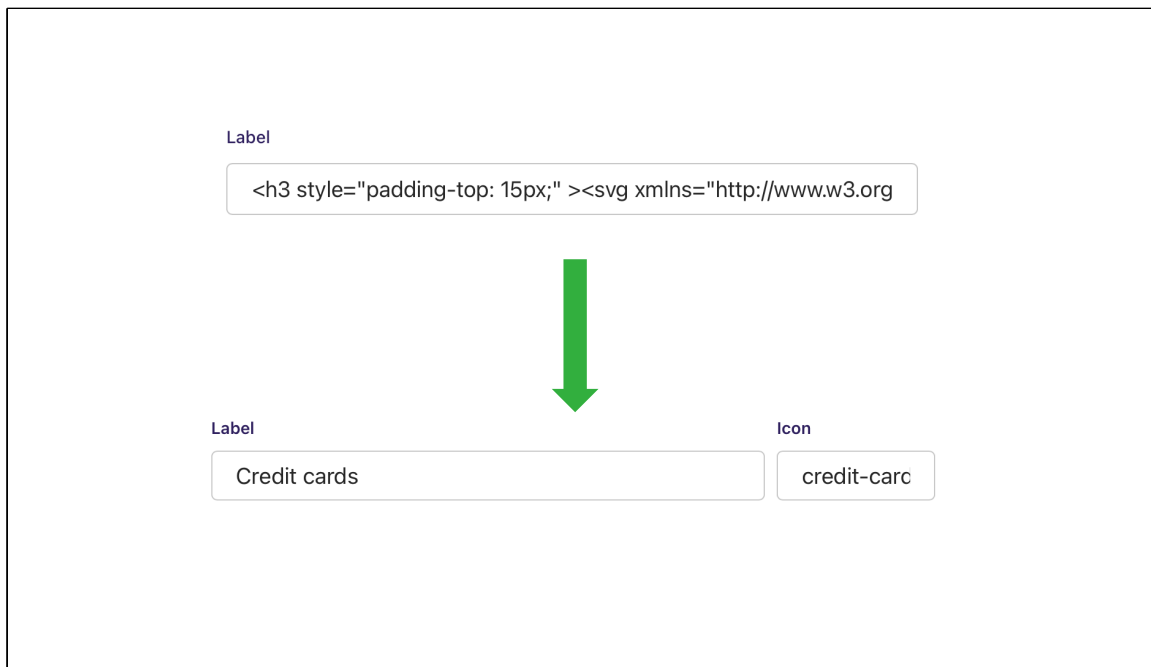
Then we can find a way to provide users with what they needed.



In some cases, we've just got a violation of our design system typography rules, and we can get rid of it entirely.



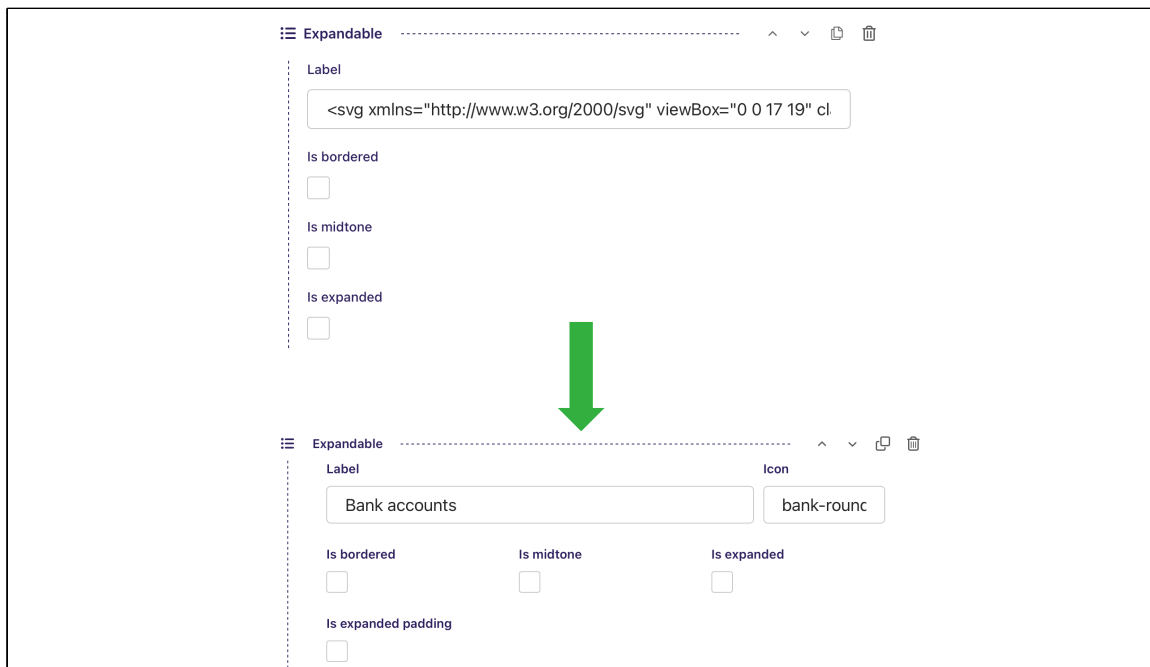
Some labels are being used as headings, so let's give our content managers heading fields.



And if icons are needed, let's give them icons.

The diagram illustrates a workflow for converting raw HTML into a structured form. At the top, a 'Text' field contains the raw HTML code: `<b>Explore key terms</b>`. A large green arrow points down to a more complex form below. This second form also has a 'Text' field, but it contains the plain text 'Explore key terms'. Below the text field are three additional fields: an 'Aria label' field (with a descriptive note: 'Add an ARIA label if the link text does not describe the destination of the link (e.g. has ambiguous text like "Learn more" that is not descriptive on its own)'), a 'Url' field containing the path `/consumer-tools/credit-reports-and-scores/answers/key-terms/`, and a checkbox labeled 'Is link boldface' which is currently checked.

The key to improving such a complex system is for developers to collaborate with the content managers – to add in features when the use of raw HTML suggests a clear need, for example!



This lets you standardize and formalize patterns of usage that will happen with or without back-end support (but will be much hackier without!)

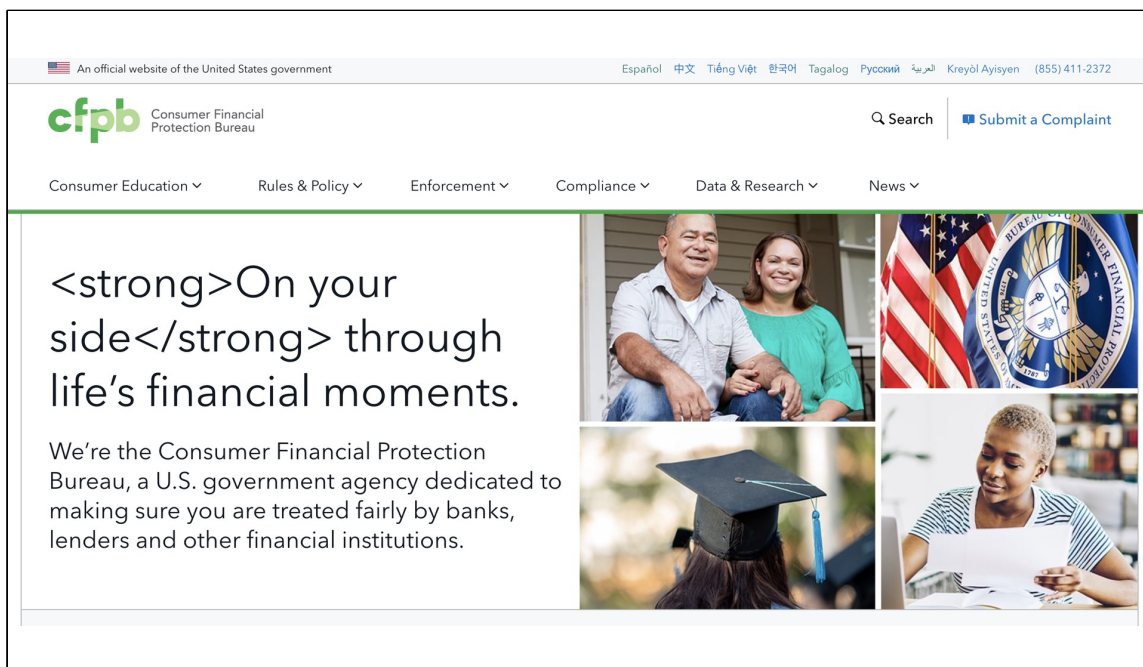
Several data migrations and manual remediations later...



And now that we've added these and a few other features our content managers needed,

```
- rendered = html.unescape(rendered)
```

We've removed that unescaped.



And quickly fixed a few minor things we missed.

How do you eat an Elephant?



So... this was a daunting problem to consider initially: how do we even start auditing content to see what patterns we have.

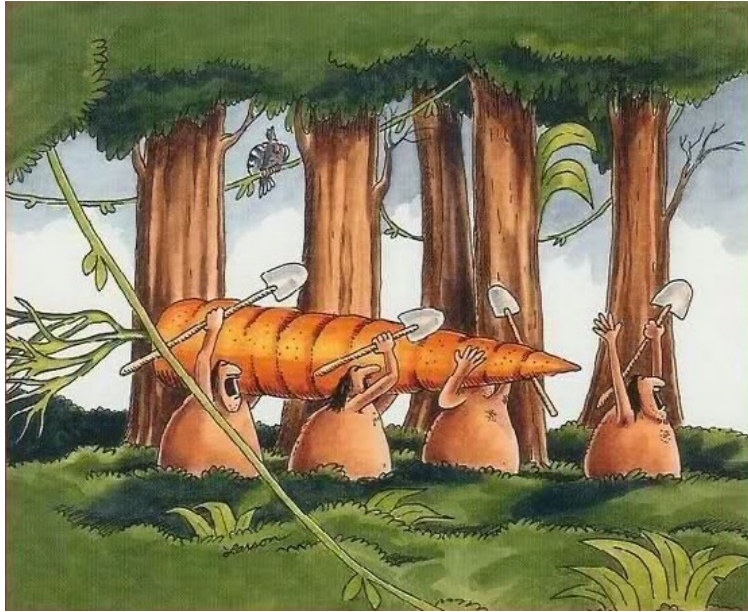
There's this quote that is usually attributed to Desmond Tutu

~~How do you eat an Elephant?~~

I'm a vegetarian, I don't.



Which... I mean this still works for this talk, how do you audit a massive pile of content: don't.



So, you do it with a little bit at a time, in small pieces, and thinking about our users the whole way.

Hopefully creating some useful tools for ourselves and others while we're at it.

Thank you!
Any questions?



REPEAT THE QUESTION!!!