RIS-Web Backend

Release 1.1

Maximiliano Santander

Dec 31, 2024

CONTENTS:

CHAPTER

ONE

OVERVIEW

Custom CMS developed for RIS Web & Software Development GmbH & Co. KG. Implements multi-language content management with Redis caching.

CHAPTER

TWO

FEATURES

- Multi-language support (DE/EN)
- Redis cache integration
- Dynamic page blocks
- Custom template system

2.1 Getting Started

2.1.1 Introduction

Welcome to the RIS-Web Backend documentation. This section will guide you through the initial setup and configuration of the project.

2.1.2 Requirements

- Python 3.8+
- PostgreSQL
- Redis
- Java (for PlantUML)

2.1.3 Installation Steps

Follow these steps to set up your development environment:

1. Clone the Repository

```
git clone <repository-url>
```

2. Set Up Python Environment

```
python -m venv .venv

# On Windows
.venv\Scripts\activate

# On macOS/Linux
source .venv/bin/activate

pip install -r requirements.txt
```

2.1.4 Database Setup

1. Install and Configure PostgreSQL:

```
CREATE DATABASE ris_db;
CREATE USER ris_admin WITH PASSWORD 'yourpassword';
GRANT ALL PRIVILEGES ON DATABASE ris_db TO ris_admin;
```

2. Configure Environment Create a *.env* file with the following variables:

```
DB_NAME=ris_db
DB_USER=ris_admin
DB_PASSWORD=yourpassword
DB_HOST=localhost
DB_PORT=5432
REDIS_URL=redis://127.0.0.1:6379/1
```

3. Run Migrations

```
python manage.py migrate
```

2.1.5 Start Development Server

Run the following command:

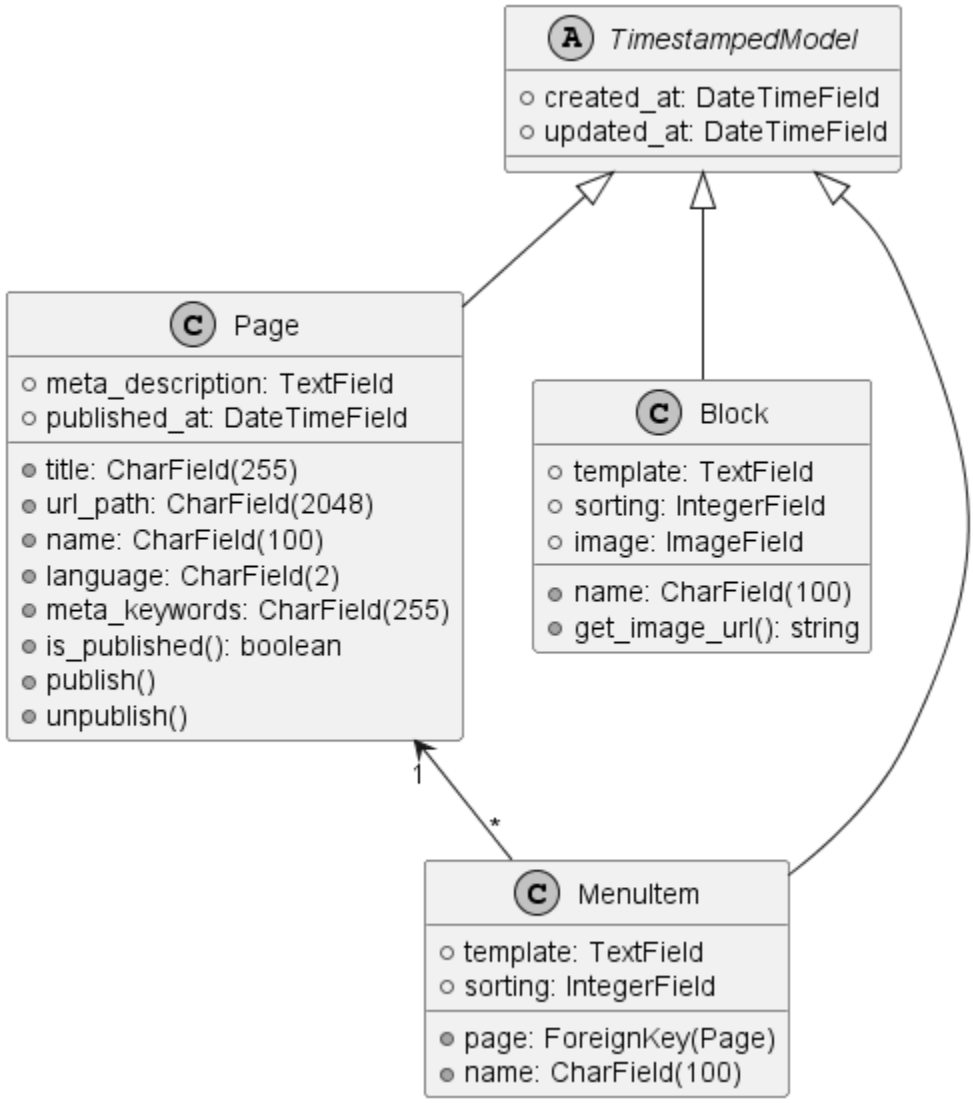
```
python manage.py runserver
```

The application will be available at `http://localhost:8000`.

2.2 Database Schema

2.2.1 Model Overview

The RIS-Web Backend uses several interconnected models to manage content and navigation. Below is a diagram showing the relationships between these models.



2.2.2 Model Details

Page Model

The Page model is the core content type, managing multilingual pages with SEO capabilities and publication workflow.

Block Model

Blocks represent reusable content components that can be positioned within pages.

MenuItem Model

MenuItems handle navigation structure by linking to pages with customizable templates.

2.2.3 Key Features

- Automatic timestamp tracking via TimestampedModel
- Version history using django-simple-history
- SEO optimization fields
- Multilingual support (EN/DE)
- Flexible content positioning

2.3 Views Documentation

2.3.1 Overview

Documentation for the RIS CMS views implementation.

Home View Documentation

Overview

The home view module implements the main page rendering logic for the RIS CMS, including caching and language support.

Key Functions

render_page

pages_app.views.home.**render_page**(*request*, *url_path=''*)

Renders a page with caching support and language-specific content.

This view handles the main page rendering logic with the following features: - Path normalization - Language-specific content - Redis caching implementation - Dynamic block loading - Menu item integration

Parameters

- **request** (*HttpRequest*) – The HTTP request object
- **url_path** (*str, optional*) – The URL path to render. Defaults to empty string

Returns
Rendered page response

Return type
HttpResponse

Raises
Http404 – If the requested page is not found or not published

Cache:

- Key Format: "page_{normalized_path}_{lang}"
- Storage: Redis (configured in settings.CACHES)
- TTL: 15 minutes (settings.CACHE_TTL)

Related Models:

- [*Page*](pages_app/models/page.py): Content and metadata
- [*Block*](pages_app/models/block.py): Reusable content blocks
- [*MenuItem*](pages_app/models/menu_item.py): Navigation elements
- [*PageBlock*](pages_app/models/page_block.py): Page-Block relationships

Cache Implementation

The view implements Redis caching with the following features:

- Language-specific caching
- Page content caching
- Cache invalidation on content updates

```
# Cache key format
cache_key = f"page_{normalized_path}_{language}"
```

Dependencies

- `django.shortcuts`
- `django.utils.translation`
- `django.core.cache`
- *pages_app.models*

Models Used

- *pages_app.models.Page*
- *pages_app.models.Block*
- *pages_app.models.MenuItem*
- *pages_app.models.PageBlock*

Configuration

Cache settings are defined in `ris_prj/settings/base.py`:

```
CACHES = {
    'default': {
        'BACKEND': 'django_redis.cache.RedisCache',
        'LOCATION': 'redis://127.0.0.1:6379/1',
        'OPTIONS': {
            'CLIENT_CLASS': 'django_redis.client.DefaultClient',
        }
    }
}
```

Page Views Documentation

Overview

Documentation for the page-specific views in the RIS CMS.

Key Components

Page Rendering

pages_app.views.home.**render_page**(*request*, *url_path=''*)

Renders a page with caching support and language-specific content.

This view handles the main page rendering logic with the following features: - Path normalization - Language-specific content - Redis caching implementation - Dynamic block loading - Menu item integration

Parameters

- **request** (*HttpRequest*) – The HTTP request object
- **url_path** (*str, optional*) – The URL path to render. Defaults to empty string

Returns
Rendered page response

Return type
HttpResponse

Raises
Http404 – If the requested page is not found or not published

Cache:

- Key Format: "page_{normalized_path}_{lang}"
- Storage: Redis (configured in settings.CACHES)
- TTL: 15 minutes (settings.CACHE_TTL)

Related Models:

- [*Page*](pages_app/models/page.py): Content and metadata
- [*Block*](pages_app/models/block.py): Reusable content blocks
- [*MenuItem*](pages_app/models/menu_item.py): Navigation elements
- [*PageBlock*](pages_app/models/page_block.py): Page-Block relationships

Main view function that handles page rendering with: - Multi-language support (DE/EN) - Redis caching implementation - Dynamic block loading

Page Model Integration

Integrates with:

- *pages_app.models.Page* - Core content model
- *pages_app.models.Block* - Reusable content blocks
- *pages_app.models.MenuItem* - Navigation elements

Cache Configuration

Uses Redis for caching with:

```
CACHES = {
    'default': {
        'BACKEND': 'django_redis.cache.RedisCache',
        'LOCATION': 'redis://127.0.0.1:6379/1',
        'OPTIONS': {
            'CLIENT_CLASS': 'django_redis.client.DefaultClient',
        }
    }
}
```

Dependencies

- Django Views Framework
- Redis Cache Backend
- Language Middleware

2.3.2 Home View

Main rendering logic for pages with caching.

2.3.3 Page View

Page-specific view functionality.

2.4 Deployment

2.4.1 Overview

This section provides guidelines for deploying the RIS-Web Backend to a production environment.

2.4.2 Deployment Steps

1. Set up the production environment: - Install necessary software (e.g., PostgreSQL, Redis, Nginx) - Configure environment variables
2. Collect static files: .. code-block:: bash

   python manage.py collectstatic
3. Apply database migrations: .. code-block:: bash

   python manage.py migrate
4. Configure Gunicorn: - Create a Gunicorn service file - Start the Gunicorn service
5. Configure Nginx: - Set up Nginx to serve static files and proxy requests to Gunicorn
6. Enable HTTPS: - Obtain an SSL certificate (e.g., Let's Encrypt) - Configure Nginx for HTTPS

2.4.3 Example Configuration

Example Gunicorn service file: .. code-block:: ini

[Unit] Description=gunicorn daemon After=network.target

[Service] User=www-data Group=www-data WorkingDirectory=/path/to/your/project ExecStart=/path/to/your/venv/bin/gunicorn –access-logfile - –workers 3 –bind unix:/path/to/your/project.sock ris_prj.wsgi:application

[Install] WantedBy=multi-user.target

Example Nginx configuration: .. code-block:: nginx

server {
listen 80; server_name your-domain.com;

location /static/ {
alias /path/to/your/project/static/;

}

location / {
proxy_pass http://unix:/path/to/your/project.sock; proxy_set_header Host $host; proxy_set_header X-Real-IP $remote_addr; proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for; proxy_set_header X-Forwarded-Proto $scheme;

}

}

2.5 API Documentation

2.5.1 Overview

API endpoints and functionality documentation for RIS CMS.

API Endpoints

Overview

Documentation of available API endpoints in the RIS CMS.

Core Endpoints

Page Management

```
GET /api/pages/
POST /api/pages/
GET /api/pages/<id>/
PUT /api/pages/<id>/
DELETE /api/pages/<id>/
```

Block Management

```
GET /api/blocks/
POST /api/blocks/
GET /api/blocks/<id>/
```

(continues on next page)

(continued from previous page)

```
PUT /api/blocks/<id>/
DELETE /api/blocks/<id>/
```

Authentication

Overview

Authentication in the RIS-Web Backend is handled through Django's built-in authentication system.

Authentication Methods

1. Session Authentication - Used for admin interface - Cookie-based authentication - CSRF protection enabled
2. Development Settings - Debug mode authentication - Local development users

Security Configuration

```
# Security settings in ris_prj/settings/base.py
AUTHENTICATION_BACKENDS = [
    'django.contrib.auth.backends.ModelBackend',
]

SESSION_COOKIE_SECURE = True
CSRF_COOKIE_SECURE = True
```

Examples

API Usage Examples

Page Creation

```
# Create a new page
from pages_app.models import Page

page = Page.objects.create(
    title="Example Page",
    url_path="/example",
    language="EN",
    is_published=True
)
```

Block Management

```
# Add a block to a page
from pages_app.models import Block, PageBlock

block = Block.objects.create(
    name="Header Block",
    sorting=1
)
```

(continues on next page)

(continued from previous page)

```
PageBlock.objects.create(
    page=page,
    block=block
)
```

Cache Operations

```
# Cache management example
from django.core.cache import cache

# Cache a page
cache_key = f"page_{page.url_path}_{page.language}"
cache.set(cache_key, page_data, timeout=3600)
```

2.5.2 Authentication

Details about authentication mechanisms.

2.5.3 Endpoints

Available API endpoints and usage.

2.5.4 Examples

Code examples and usage patterns.

2.6 Modules

2.6.1 pages_app package

Subpackages

pages_app.migrations package

Submodules

pages_app.migrations.0001_initial module

class pages_app.migrations.0001_initial.**Migration**(*name*, *app_label*)

Bases: `Migration`

dependencies = []

initial = True

operations = [<CreateModel name='Block', fields=[('id', <django.db.models.fields.BigAutoField>), ('created_at', <django.db.models.fields.DateTimeField>), ('updated_at', <django.db.models.fields.DateTimeField>), ('template', <django.db.models.fields.TextField>), ('sorting', <django.db.models.fields.IntegerField>)], options={'indexes': [<Index: fields=['sorting'] name='pages_app_b_sorting_0e143c_idx'>]}>, <CreateModel name='Page', fields=[('id', <django.db.models.fields.BigAutoField>), ('created_at', <django.db.models.fields.DateTimeField>), ('updated_at', <django.db.models.fields.DateTimeField>), ('title', <django.db.models.fields.CharField>), ('url_path', <django.db.models.fields.CharField>), ('language', <django.db.models.fields.CharField>), ('meta_description', <django.db.models.fields.TextField>), ('meta_keywords', <django.db.models.fields.CharField>), ('is_published', <django.db.models.fields.BooleanField>), ('published_at', <django.db.models.fields.DateTimeField>)], options={'indexes': [<Index: fields=['url_path'] name='pages_app_p_url_pat_a7516e_idx'>, <Index: fields=['is_published'] name='pages_app_p_is_publ_d11ecf_idx'>]}>, <CreateModel name='MenuItem', fields=[('id', <django.db.models.fields.BigAutoField>), ('created_at', <django.db.models.fields.DateTimeField>), ('updated_at', <django.db.models.fields.DateTimeField>), ('template', <django.db.models.fields.TextField>), ('sorting', <django.db.models.fields.IntegerField>), ('page', <django.db.models.fields.related.ForeignKey>)], options={'indexes': [<Index: fields=['sorting'] name='pages_app_m_sorting_aa33a4_idx'>]}>, <CreateModel name='PageBlock', fields=[('id', <django.db.models.fields.BigAutoField>), ('block', <django.db.models.fields.related.ForeignKey>), ('page', <django.db.models.fields.related.ForeignKey>)], options={'constraints': [<UniqueConstraint: fields=('page', 'block') name='unique_page_block'>]}>]

pages_app.migrations.0002_menuitem_name module

class pages_app.migrations.0002_menuitem_name.**Migration**(*name*, *app_label*)

Bases: Migration

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

operations = [<AddField model_name='menuitem', name='name', field=<django.db.models.fields.CharField>>]

pages_app.migrations.0003_block_name_page_name module

class pages_app.migrations.0003_block_name_page_name.**Migration**(*name*, *app_label*)

Bases: Migration

dependencies = [('pages_app', '0002_menuitem_name')]

operations = [<AddField model_name='block', name='name', field=<django.db.models.fields.CharField>>, <AddField model_name='page', name='name', field=<django.db.models.fields.CharField>>]

pages_app.migrations.0004_alter_block_name module

class pages_app.migrations.0004_alter_block_name.**Migration**(*name*, *app_label*)

Bases: Migration

dependencies = [('pages_app', '0003_block_name_page_name')]

operations = [<AlterField model_name='block', name='name', field=<django.db.models.fields.CharField>>]

pages_app.migrations.0005_historicalblock_historicalmenuitem_historicalpage_and_more module

class pages_app.migrations.0005_historicalblock_historicalmenuitem_historicalpage_and_more.**Migration**(*na*
ap

Bases: Migration

dependencies = [('pages_app', '0004_alter_block_name'), ('auth', '__first__')]

```
operations = [<CreateModel name='HistoricalBlock', fields=[('id',
<django.db.models.fields.BigIntegerField>), ('created_at',
<django.db.models.fields.DateTimeField>), ('updated_at',
<django.db.models.fields.DateTimeField>), ('template',
<django.db.models.fields.TextField>), ('sorting',
<django.db.models.fields.IntegerField>), ('name',
<django.db.models.fields.CharField>), ('history_id',
<django.db.models.fields.AutoField>), ('history_date',
<django.db.models.fields.DateTimeField>), ('history_change_reason',
<django.db.models.fields.CharField>), ('history_type',
<django.db.models.fields.CharField>), ('history_user',
<django.db.models.fields.related.ForeignKey>)], options={'verbose_name':
'historical block', 'verbose_name_plural':  'historical blocks', 'ordering':
('-history_date', '-history_id'), 'get_latest_by':  ('history_date', 'history_id')},
bases=(<class 'simple_history.models.HistoricalChanges'>, <class
'django.db.models.base.Model'>)>, <CreateModel name='HistoricalMenuItem',
fields=[('id', <django.db.models.fields.BigIntegerField>), ('created_at',
<django.db.models.fields.DateTimeField>), ('updated_at',
<django.db.models.fields.DateTimeField>), ('name',
<django.db.models.fields.CharField>), ('template',
<django.db.models.fields.TextField>), ('sorting',
<django.db.models.fields.IntegerField>), ('history_id',
<django.db.models.fields.AutoField>), ('history_date',
<django.db.models.fields.DateTimeField>), ('history_change_reason',
<django.db.models.fields.CharField>), ('history_type',
<django.db.models.fields.CharField>), ('history_user',
<django.db.models.fields.related.ForeignKey>), ('page',
<django.db.models.fields.related.ForeignKey>)], options={'verbose_name':
'historical menu item', 'verbose_name_plural':  'historical menu items', 'ordering':
('-history_date', '-history_id'), 'get_latest_by':  ('history_date', 'history_id')},
bases=(<class 'simple_history.models.HistoricalChanges'>, <class
'django.db.models.base.Model'>)>, <CreateModel name='HistoricalPage', fields=[('id',
<django.db.models.fields.BigIntegerField>), ('created_at',
<django.db.models.fields.DateTimeField>), ('updated_at',
<django.db.models.fields.DateTimeField>), ('title',
<django.db.models.fields.CharField>), ('url_path',
<django.db.models.fields.CharField>), ('name', <django.db.models.fields.CharField>),
('language', <django.db.models.fields.CharField>), ('meta_description',
<django.db.models.fields.TextField>), ('meta_keywords',
<django.db.models.fields.CharField>), ('is_published',
<django.db.models.fields.BooleanField>), ('published_at',
<django.db.models.fields.DateTimeField>), ('history_id',
<django.db.models.fields.AutoField>), ('history_date',
<django.db.models.fields.DateTimeField>), ('history_change_reason',
<django.db.models.fields.CharField>), ('history_type',
<django.db.models.fields.CharField>), ('history_user',
<django.db.models.fields.related.ForeignKey>)], options={'verbose_name':
'historical page', 'verbose_name_plural':  'historical pages', 'ordering':
('-history_date', '-history_id'), 'get_latest_by':  ('history_date', 'history_id')},
bases=(<class 'simple_history.models.HistoricalChanges'>, <class
'django.db.models.base.Model'>)>, <CreateModel name='HistoricalPageBlock',
fields=[('id', <django.db.models.fields.BigIntegerField>), ('history_id',
<django.db.models.fields.AutoField>), ('history_date',
<django.db.models.fields.DateTimeField>), ('history_change_reason',
<django.db.models.fields.CharField>), ('history_type',
<django.db.models.fields.CharField>), ('block',
<django.db.models.fields.related.ForeignKey>), ('history_user',
<django.db.models.fields.related.ForeignKey>), ('page',
<django.db.models.fields.related.ForeignKey>)], options={'verbose_name':
'historical page block', 'verbose_name_plural':  'historical page blocks',
```

pages_app.migrations.0006_block_image_historicalblock_image module

class pages_app.migrations.0006_block_image_historicalblock_image.**Migration**(*name*, *app_label*)

Bases: Migration

dependencies = [('pages_app', '0005_historicalblock_historicalmenuitem_historicalpage_and_more')]

operations = [<AddField model_name='block', name='image', field=<django.db.models.fields.files.ImageField>>, <AddField model_name='historicalblock', name='image', field=<django.db.models.fields.TextField>>]

pages_app.migrations.0007_remove_page_pages_app_p_is_publ_d11ecf_idx_and_more module

class pages_app.migrations.0007_remove_page_pages_app_p_is_publ_d11ecf_idx_and_more.**Migration**(*name*, *app_label*)

Bases: Migration

dependencies = [('pages_app', '0006_block_image_historicalblock_image')]

operations = [<RemoveIndex model_name='page', name='pages_app_p_is_publ_d11ecf_idx'>, <RemoveField model_name='historicalpage', name='is_published'>, <RemoveField model_name='page', name='is_published'>, <AddIndex model_name='page', index=<Index: fields=['published_at'] name='pages_app_p_publish_9c7cf5_idx'>>]

pages_app.migrations.0008_remove_page_pages_app_p_publish_9c7cf5_idx_and_more module

class pages_app.migrations.0008_remove_page_pages_app_p_publish_9c7cf5_idx_and_more.**Migration**(*name*, *app_label*)

Bases: Migration

dependencies = [('pages_app', '0007_remove_page_pages_app_p_is_publ_d11ecf_idx_and_more')]

operations = [<RemoveIndex model_name='page', name='pages_app_p_publish_9c7cf5_idx'>, <AddField model_name='historicalpage', name='is_published', field=<django.db.models.fields.BooleanField>>, <AddField model_name='page', name='is_published', field=<django.db.models.fields.BooleanField>>, <AddIndex model_name='page', index=<Index: fields=['is_published'] name='pages_app_p_is_publ_d11ecf_idx'>>]

pages_app.migrations.0009_remove_page_pages_app_p_is_publ_d11ecf_idx_and_more module

class pages_app.migrations.0009_remove_page_pages_app_p_is_publ_d11ecf_idx_and_more.**Migration**(*name*, *app_label*)

Bases: Migration

dependencies = [('pages_app', '0008_remove_page_pages_app_p_publish_9c7cf5_idx_and_more')]

operations = [<RemoveIndex model_name='page',
name='pages_app_p_is_publ_d11ecf_idx'>, <RemoveField model_name='historicalpage',
name='is_published'>, <RemoveField model_name='page', name='is_published'>,
<AddIndex model_name='page', index=<Index: fields=['published_at']
name='pages_app_p_publish_9c7cf5_idx'>>]

pages_app.migrations.0010_remove_page_pages_app_p_publish_9c7cf5_idx_and_more module

class pages_app.migrations.0010_remove_page_pages_app_p_publish_9c7cf5_idx_and_more.**Migration**(*name*, *app_label*)

Bases: Migration

dependencies = [('pages_app',
'0009_remove_page_pages_app_p_is_publ_d11ecf_idx_and_more')]

operations = [<RemoveIndex model_name='page',
name='pages_app_p_publish_9c7cf5_idx'>, <AddField model_name='historicalpage',
name='is_published', field=<django.db.models.fields.BooleanField>>, <AddField
model_name='page', name='is_published',
field=<django.db.models.fields.BooleanField>>, <AddIndex model_name='page',
index=<Index: fields=['is_published'] name='pages_app_p_is_publ_d11ecf_idx'>>]

pages_app.migrations.0011_remove_page_pages_app_p_is_publ_d11ecf_idx_and_more module

class pages_app.migrations.0011_remove_page_pages_app_p_is_publ_d11ecf_idx_and_more.**Migration**(*name*, *app_label*)

Bases: Migration

dependencies = [('pages_app',
'0010_remove_page_pages_app_p_publish_9c7cf5_idx_and_more')]

operations = [<RemoveIndex model_name='page',
name='pages_app_p_is_publ_d11ecf_idx'>, <RemoveField model_name='historicalpage',
name='is_published'>, <RemoveField model_name='page', name='is_published'>,
<AddIndex model_name='page', index=<Index: fields=['published_at']
name='pages_app_p_publish_9c7cf5_idx'>>]

pages_app.migrations.0012_block_script_historicalblock_script module

class pages_app.migrations.0012_block_script_historicalblock_script.**Migration**(*name*, *app_label*)

Bases: Migration

dependencies = [('pages_app',
'0011_remove_page_pages_app_p_is_publ_d11ecf_idx_and_more')]

operations = [<AddField model_name='block', name='script',
field=<django.db.models.fields.TextField>>, <AddField model_name='historicalblock',
name='script', field=<django.db.models.fields.TextField>>]

pages_app.migrations.0013_remove_block_script_remove_historicalblock_script module

class pages_app.migrations.0013_remove_block_script_remove_historicalblock_script.**Migration**(*name*, *app_label*)

Bases: Migration

dependencies = [('pages_app', '0012_block_script_historicalblock_script')]

operations = [<RemoveField model_name='block', name='script'>, <RemoveField model_name='historicalblock', name='script'>]

Module contents

pages_app.models package

Submodules

pages_app.models.base module

Base model that adds timestamp tracking.

This module provides the TimestampedModel abstract base class that automatically tracks creation and modification dates for all models that inherit from it.

pages_app.models.base.**created_at**

Timestamp when record was created

Type
DateTimeField

pages_app.models.base.**updated_at**

Timestamp when record was last modified

Type
DateTimeField

class pages_app.models.base.**TimestampedModel**(**args*, ***kwargs*)

Bases: Model

Abstract base model that automatically tracks creation and modification dates.

created_at

Timestamp of when the record was created

Type
DateTime

updated_at

Timestamp of when the record was last modified

Type
DateTime

Note

This is an abstract base class and should not be used directly. Inherit from this class to add timestamp functionality to your models.

class Meta

Bases: object

abstract = False

created_at

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

get_next_by_created_at(*, *field=<django.db.models.fields.DateTimeField: created_at>*, *is_next=True*, ***kwargs*)

get_next_by_updated_at(*, *field=<django.db.models.fields.DateTimeField: updated_at>*, *is_next=True*, ***kwargs*)

get_previous_by_created_at(*, *field=<django.db.models.fields.DateTimeField: created_at>*, *is_next=False*, ***kwargs*)

get_previous_by_updated_at(*, *field=<django.db.models.fields.DateTimeField: updated_at>*, *is_next=False*, ***kwargs*)

updated_at

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

pages_app.models.block module

Block model for content fragments.

This module provides the Block model for managing reusable content blocks.

pages_app.models.block.**name**

Name of the block

Type
CharField

pages_app.models.block.**template**

Template content

Type
TextField

pages_app.models.block.**sorting**

Display order

Type
IntegerField

pages_app.models.block.**image**

Optional image

Type
ImageField

class pages_app.models.block.**Block**(**args*, ***kwargs*)

Bases: *TimestampedModel*

Represents a reusable content block in the CMS.

Each block can be assigned to multiple pages through PageBlock relationships. Content is rendered using a specified template, with optional image attachment.

name

Unique identifier for the block

Type
str

template

Path to the template used for rendering

Type
str

content

Main content of the block

Type
text

sorting

Position in page layout

Type
int

image

Optional associated image

Type
ImageField

history

Version tracking

Type
HistoricalRecords

Examples

```
>>> block = Block.objects.create(
...     name='header',
...     template='blocks/header.html',
...     content='Welcome to our site'
... )
```

exception DoesNotExist

Bases: `ObjectDoesNotExist`

exception MultipleObjectsReturned

Bases: `MultipleObjectsReturned`

created_at

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

get_image_url()

Get the URL of the attached image.

Returns
Image URL if exists, empty string otherwise

Return type
str

get_next_by_created_at(*, *field=<django.db.models.fields.DateTimeField: created_at>*, *is_next=True*, ***kwargs*)

get_next_by_updated_at(*, *field=<django.db.models.fields.DateTimeField: updated_at>*, *is_next=True*, ***kwargs*)

get_previous_by_created_at(*, *field=<django.db.models.fields.DateTimeField: created_at>*, *is_next=False*, ***kwargs*)

get_previous_by_updated_at(*, *field=<django.db.models.fields.DateTimeField: updated_at>*, *is_next=False*, ***kwargs*)

history = <django.db.models.manager.HistoryManagerFromHistoricalQuerySet object>

id

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

image

Just like the FileDescriptor, but for ImageFields. The only difference is assigning the width/height to the width_field/height_field, if appropriate.

name

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

objects = <django.db.models.manager.Manager object>

pageblock_set

Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

save_without_historical_record(**args*, ***kwargs*)

Save model without saving a historical record

Make sure you know what you're doing before you use this method.

sorting

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

`template`

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

`updated_at`

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

pages_app.models.menu_item module

MenuItem Model Module

This module defines the MenuItem model which represents navigation items in the CMS. MenuItems are linked to pages and can be sorted.

Key Features:

- Linked to pages
- Template-based content
- Sorting order
- Historical tracking

Dependencies:

- django.db.models
- simple_history.models.HistoricalRecords
- .base.TimestampedModel
- .page.Page

`class pages_app.models.menu_item.MenuItem(*args, **kwargs)`

Bases: *TimestampedModel*

Represents a navigation item in the CMS.

`page`

Reference to the linked page

Type
ForeignKey

`name`

Name of the menu item

Type
str

`template`

Template content for the menu item

Type
str

`sorting`

Sorting order

Type
int

history

Tracks changes to the menu item

Type
HistoricalRecords

Examples

```
>>> page = Page.objects.get(slug="home")
>>> menu_item = MenuItem.objects.create(
...     page=page,
...     name="Home",
...     template="<a href='/home'>Home</a>",
...     sorting=1
... )
```

exception DoesNotExist

Bases: ObjectDoesNotExist

exception MultipleObjectsReturned

Bases: MultipleObjectsReturned

created_at

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

get_next_by_created_at(*, *field=<django.db.models.fields.DateTimeField: created_at>*, *is_next=True*, ***kwargs*)

get_next_by_updated_at(*, *field=<django.db.models.fields.DateTimeField: updated_at>*, *is_next=True*, ***kwargs*)

get_previous_by_created_at(*, *field=<django.db.models.fields.DateTimeField: created_at>*, *is_next=False*, ***kwargs*)

get_previous_by_updated_at(*, *field=<django.db.models.fields.DateTimeField: updated_at>*, *is_next=False*, ***kwargs*)

history = <django.db.models.manager.HistoryManagerFromHistoricalQuerySet object>

id

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

name

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

objects = <django.db.models.manager.Manager object>

page

Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Child.parent` is a `ForwardManyToOneDescriptor` instance.

page_id

save_without_historical_record(**args*, ***kwargs*)

Save model without saving a historical record

Make sure you know what you're doing before you use this method.

sorting

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

template

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

updated_at

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

pages_app.models.page module

Page model for content management.

This module provides the Page model for managing web pages in the CMS.

pages_app.models.page.**title**

The page title

Type
CharField

pages_app.models.page.**url_path**

URL path to the page

Type
CharField

pages_app.models.page.**name**

Internal reference name

Type
CharField

pages_app.models.page.**language**

Language code (e.g. "DE", "EN")

Type
CharField

pages_app.models.page.**meta_description**

SEO meta description

Type
TextField

pages_app.models.page.**meta_keywords**

SEO meta keywords

Type
CharField

pages_app.models.page.**published_at**

Publication date and time

Type
DateTimeField

class pages_app.models.page.**Page**(**args*, ***kwargs*)

Bases: *TimestampedModel*

Represents a dynamic page in the CMS. Inherits from TimestampedModel for automatic timestamp tracking.

title

Page title

Type
str

url_path

URL path for the page

Type
str

name

Page identifier, max 100 chars

Type
str

language

Language code (EN/DE)

Type
str

meta_description

SEO meta description

Type
str

meta_keywords

SEO meta keywords

Type
str

published_at

When the page was published

Type
datetime

history

Version tracking

Type
HistoricalRecords

Example

```
>>> page = Page(
...     title="Homepage",
...     url_path="/home",
...     name="Homepage",
...     language="EN"
... )
>>> page.save()
```

exception DoesNotExist

Bases: ObjectDoesNotExist

LANGUAGE_CHOICES = [('EN', 'English'), ('DE', 'Deutsch')]

exception MultipleObjectsReturned

Bases: MultipleObjectsReturned

clean()

Custom model validation. Ensures URL path starts with a forward slash. :raises ValidationError: If URL path is invalid

created_at

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

get_language_display(*, *field=<django.db.models.fields.CharField: language>*)

get_next_by_created_at(*, *field=<django.db.models.fields.DateTimeField: created_at>*, *is_next=True*, ***kwargs*)

get_next_by_updated_at(*, *field=<django.db.models.fields.DateTimeField: updated_at>*, *is_next=True*, ***kwargs*)

get_previous_by_created_at(*, *field=<django.db.models.fields.DateTimeField: created_at>*, *is_next=False*, ***kwargs*)

get_previous_by_updated_at(*, *field=<django.db.models.fields.DateTimeField: updated_at>*, *is_next=False*, ***kwargs*)

history = <django.db.models.manager.HistoryManagerFromHistoricalQuerySet object>

id

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

property is_published

language

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

menuitem_set

Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

meta_description

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

meta_keywords

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

name

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

objects = <django.db.models.manager.Manager object>

pageblock_set

Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

publish()

Publish the page and set publication timestamp.

published_at

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

save(**args*, ***kwargs*)

Custom save method. Ensures URL path is valid before saving. Calls full_clean to validate the model.

save_without_historical_record(**args*, ***kwargs*)

Save model without saving a historical record

Make sure you know what you're doing before you use this method.

title

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

unpublish()

Unpublish the page and clear publication timestamp.

updated_at

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

url_path

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

pages_app.models.page_block module

PageBlock Model Module

Manages the relationship between Pages and Blocks in the CMS. Provides ordering and positioning of blocks within pages.

Key Features:

- Many-to-many relationship handler
- Block ordering within pages
- Historical tracking
- Automatic timestamp management

Dependencies:

- django.db.models
- simple_history.models.HistoricalRecords
- .page.Page
- .block.Block

class pages_app.models.page_block.**PageBlock**(**args*, ***kwargs*)

Bases: Model

Associates blocks with pages and manages their positioning.

page

Reference to the parent page

Type
ForeignKey

block

Reference to the content block

Type
ForeignKey

position

Order position within the page

Type
IntegerField

history

Tracks changes to assignments

Type
HistoricalRecords

Examples

```
>>> header = Block.objects.get(name="Header")
>>> home_page = Page.objects.get(slug="home")
>>> page_block = PageBlock.objects.create(
...     page=home_page,
...     block=header,
...     position=1
... )
```

exception DoesNotExist

Bases: ObjectDoesNotExist

exception MultipleObjectsReturned

Bases: MultipleObjectsReturned

block

Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

Child.parent is a ForwardManyToOneDescriptor instance.

block_id

history = <django.db.models.manager.HistoryManagerFromHistoricalQuerySet object>

id

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

objects = <django.db.models.manager.Manager object>

page

Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

Child.parent is a ForwardManyToOneDescriptor instance.

page_id

save_without_historical_record(**args*, ***kwargs*)

Save model without saving a historical record

Make sure you know what you're doing before you use this method.

pages_app.models.signals module

Signals Module

Handles automatic cache invalidation when model instances are modified. Implements Django's signal system to maintain cache consistency.

Key Features:

- Automatic cache clearing on model changes
- Monitors Block, MenuItem, and PageBlock models
- Uses post_save and post_delete signals
- Full cache invalidation strategy

Dependencies:

- django.db.models.signals # a module that defines signals and receivers
- django.dispatch.receiver # a decorator that connects a signal receiver function to a signal
- django.core.cache # a module that provides a cache system for Django applications

pages_app.models.signals.**invalidate_cache**(*sender*, ***kwargs*)

Signal handler that invalidates cache when monitored models change.

Arguments

- **sender:** The model class that triggered the signal
- **kwargs: Signal parameters including:**
  - **instance:** The actual model instance
  - **created:** Boolean, True if new instance (post_save only)
  - **using:** Database alias

Example

When a Block is updated: >>> block = Block.objects.first() >>> block.save() # Triggers cache invalidation

Module contents

Pages App Models Module

This module initializes and exports the core models for the RIS CMS. The models work together to create a flexible, multi-language content management system.

Models

- Page: Core content model supporting multi-language pages with meta information
- Block: Reusable content blocks that can be assigned to pages
- MenuItem: Navigation elements linked to pages

- PageBlock: Junction model managing page-block relationships

> **Note**
>
> All models use Django's ORM and follow the MTV (Model-Template-View) pattern. Performance is optimized through proper database indexing and Redis caching.

class pages_app.models.**Block**(**args*, ***kwargs*)

Bases: *TimestampedModel*

Represents a reusable content block in the CMS.

Each block can be assigned to multiple pages through PageBlock relationships. Content is rendered using a specified template, with optional image attachment.

name

Unique identifier for the block

Type
str

template

Path to the template used for rendering

Type
str

content

Main content of the block

Type
text

sorting

Position in page layout

Type
int

image

Optional associated image

Type
ImageField

history

Version tracking

Type
HistoricalRecords

Examples

```
>>> block = Block.objects.create(
...     name='header',
...     template='blocks/header.html',
...     content='Welcome to our site'
... )
```

exception DoesNotExist

Bases: ObjectDoesNotExist

exception MultipleObjectsReturned

Bases: MultipleObjectsReturned

created_at

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

get_image_url()

Get the URL of the attached image.

Returns
Image URL if exists, empty string otherwise

Return type
str

get_next_by_created_at(*, *field=<django.db.models.fields.DateTimeField: created_at>*, *is_next=True*, ***kwargs*)

get_next_by_updated_at(*, *field=<django.db.models.fields.DateTimeField: updated_at>*, *is_next=True*, ***kwargs*)

get_previous_by_created_at(*, *field=<django.db.models.fields.DateTimeField: created_at>*, *is_next=False*, ***kwargs*)

get_previous_by_updated_at(*, *field=<django.db.models.fields.DateTimeField: updated_at>*, *is_next=False*, ***kwargs*)

history = <django.db.models.manager.HistoryManagerFromHistoricalQuerySet object>

id

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

image

Just like the FileDescriptor, but for ImageFields. The only difference is assigning the width/height to the width_field/height_field, if appropriate.

name

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

objects = <django.db.models.manager.Manager object>

pageblock_set

Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

save_without_historical_record(**args*, ***kwargs*)

Save model without saving a historical record

Make sure you know what you're doing before you use this method.

sorting

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

template

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

updated_at

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

class pages_app.models.**MenuItem**(**args*, ***kwargs*)

Bases: *TimestampedModel*

Represents a navigation item in the CMS.

page

Reference to the linked page

Type
ForeignKey

name

Name of the menu item

Type
str

template

Template content for the menu item

Type
str

sorting

Sorting order

Type
int

history

Tracks changes to the menu item

Type
HistoricalRecords

Examples

```
>>> page = Page.objects.get(slug="home")
>>> menu_item = MenuItem.objects.create(
...     page=page,
...     name="Home",
...     template="<a href='/home'>Home</a>",
```

(continues on next page)

(continued from previous page)

```
...     sorting=1
... )
```

exception DoesNotExist

Bases: ObjectDoesNotExist

exception MultipleObjectsReturned

Bases: MultipleObjectsReturned

created_at

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

get_next_by_created_at(*, *field=<django.db.models.fields.DateTimeField: created_at>*, *is_next=True*, ***kwargs*)

get_next_by_updated_at(*, *field=<django.db.models.fields.DateTimeField: updated_at>*, *is_next=True*, ***kwargs*)

get_previous_by_created_at(*, *field=<django.db.models.fields.DateTimeField: created_at>*, *is_next=False*, ***kwargs*)

get_previous_by_updated_at(*, *field=<django.db.models.fields.DateTimeField: updated_at>*, *is_next=False*, ***kwargs*)

history = <django.db.models.manager.HistoryManagerFromHistoricalQuerySet object>

id

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

name

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

objects = <django.db.models.manager.Manager object>

page

Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Child.parent` is a `ForwardManyToOneDescriptor` instance.

page_id

save_without_historical_record(**args*, ***kwargs*)

Save model without saving a historical record

Make sure you know what you're doing before you use this method.

sorting

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

template

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

updated_at

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

class pages_app.models.**Page**(**args*, ***kwargs*)

Bases: *TimestampedModel*

Represents a dynamic page in the CMS. Inherits from TimestampedModel for automatic timestamp tracking.

title

Page title

Type
str

url_path

URL path for the page

Type
str

name

Page identifier, max 100 chars

Type
str

language

Language code (EN/DE)

Type
str

meta_description

SEO meta description

Type
str

meta_keywords

SEO meta keywords

Type
str

published_at

When the page was published

Type
datetime

history

Version tracking

Type
HistoricalRecords

Example

```
>>> page = Page(
...     title="Homepage",
...     url_path="/home",
...     name="Homepage",
...     language="EN"
... )
>>> page.save()
```

exception DoesNotExist

Bases: ObjectDoesNotExist

LANGUAGE_CHOICES = [('EN', 'English'), ('DE', 'Deutsch')]

exception MultipleObjectsReturned

Bases: MultipleObjectsReturned

clean()

Custom model validation. Ensures URL path starts with a forward slash. :raises ValidationError: If URL path is invalid

created_at

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

get_language_display(*, *field=<django.db.models.fields.CharField: language>*)

get_next_by_created_at(*, *field=<django.db.models.fields.DateTimeField: created_at>*, *is_next=True*, ***kwargs*)

get_next_by_updated_at(*, *field=<django.db.models.fields.DateTimeField: updated_at>*, *is_next=True*, ***kwargs*)

get_previous_by_created_at(*, *field=<django.db.models.fields.DateTimeField: created_at>*, *is_next=False*, ***kwargs*)

get_previous_by_updated_at(*, *field=<django.db.models.fields.DateTimeField: updated_at>*, *is_next=False*, ***kwargs*)

history = <django.db.models.manager.HistoryManagerFromHistoricalQuerySet object>

id

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

property is_published

language

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

menuitem_set

Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

meta_description

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

meta_keywords

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

name

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

objects = <django.db.models.manager.Manager object>

pageblock_set

Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

publish()

Publish the page and set publication timestamp.

published_at

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

save(**args*, ***kwargs*)

Custom save method. Ensures URL path is valid before saving. Calls full_clean to validate the model.

save_without_historical_record(**args*, ***kwargs*)

Save model without saving a historical record

Make sure you know what you're doing before you use this method.

title

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

unpublish()

Unpublish the page and clear publication timestamp.

updated_at

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

url_path

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

class pages_app.models.**PageBlock**(**args*, ***kwargs*)

Bases: Model

Associates blocks with pages and manages their positioning.

page

Reference to the parent page

Type
ForeignKey

block

Reference to the content block

Type
ForeignKey

position

Order position within the page

Type
IntegerField

history

Tracks changes to assignments

Type
HistoricalRecords

Examples

```
>>> header = Block.objects.get(name="Header")
>>> home_page = Page.objects.get(slug="home")
>>> page_block = PageBlock.objects.create(
...     page=home_page,
...     block=header,
...     position=1
... )
```

exception DoesNotExist

Bases: ObjectDoesNotExist

exception MultipleObjectsReturned

Bases: MultipleObjectsReturned

block

Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Child.parent` is a `ForwardManyToOneDescriptor` instance.

block_id

history = <django.db.models.manager.HistoryManagerFromHistoricalQuerySet object>

id

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

objects = <django.db.models.manager.Manager object>

page

Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Child.parent` is a `ForwardManyToOneDescriptor` instance.

page_id

save_without_historical_record(**args*, ***kwargs*)

Save model without saving a historical record

Make sure you know what you're doing before you use this method.

pages_app.tests package

Submodules

pages_app.tests.test_cache module

Cache Tests Module

Tests Redis cache implementation for RIS CMS. Verifies cache operations, timeouts, and cleanup.

Test Cases

- Basic cache operations (set/get)
- Cache timeout functionality
- Cache deletion operations

Dependencies

- django.core.cache
- django.test.TestCase
- time

class pages_app.tests.test_cache.**CacheTests**(*methodName='runTest'*)

Bases: TestCase

Test suite for Redis cache implementation.

Tests cache operations and behavior ensuring proper: - Data storage and retrieval - Timeout functionality - Cache clearing

setUp()

Initialize test environment. Clears cache before each test to ensure isolation.

tearDown()

Clean up test environment after each test.

Ensures:

- Cache is completely cleared
- No test data remains
- Next test starts with empty cache

test_cache_clear()

Test clearing entire cache.

Verifies:

- Multiple values can be stored
- Cache clear removes all values
- All keys return None after clear

Example

```
>>> cache.set("key1", "value1")
>>> cache.set("key2", "value2")
>>> cache.clear()
>>> cache.get("key1"), cache.get("key2")
None, None
```

test_cache_delete()

Test manual cache deletion.

Verifies:

- Data can be manually removed from cache
- Deleted data returns None

Example

```
>>> cache.set("key", "value")
>>> cache.delete("key")
>>> cache.get("key")
None
```

test_cache_set_get()

Test basic cache set and get operations.

Verifies:

- Data can be stored in cache
- Retrieved data matches stored data

Example

```
>>> cache.set("key", "value", 60)
>>> cache.get("key")
'value'
```

test_cache_timeout()

Test cache timeout functionality.

Verifies:

- Data expires after timeout period
- Expired data returns None

Example

```
>>> cache.set("key", "value", 1)  # 1 second timeout
>>> time.sleep(2)
>>> cache.get("key")
None
```

pages_app.tests.test_models module

class pages_app.tests.test_models.**BlockModelTest**(*methodName='runTest'*)

Bases: TestCase

Test suite for Block model functionality.

Tests cover:

- Block creation and validation
- Template handling
- Block ordering

Related Model:
[*Block*](pages_app/models/block.py)

test_block_creation()

Test block creation with all fields.

Verifies:

- Block name is correctly set
- Block template is correctly set

`test_block_ordering()`

Test block ordering by sorting field.

Verifies:

- Blocks are ordered by sorting field

`class pages_app.tests.test_models.MenuItemModelTest(methodName='runTest')`

Bases: `TestCase`

Test suite for MenuItem model functionality.

Tests cover:

- MenuItem creation and validation
- MenuItem ordering
- MenuItem deletion behavior

Related Model:
[*MenuItem*](pages_app/models/menu_item.py)

`setUp()`

Initialize test data

`test_menu_item_creation()`

Test menu item creation with all fields.

Verifies:

- MenuItem page relationship is correctly set
- MenuItem template is correctly set

`test_menu_item_ordering()`

Test menu item ordering by sorting field.

Verifies:

- MenuItems are ordered by sorting field

`test_menu_item_page_deletion()`

Test cascade deletion behavior for MenuItem relationships.

Verifies:

- MenuItem is deleted when parent Page is deleted

`class pages_app.tests.test_models.PageBlockModelTest(methodName='runTest')`

Bases: `TestCase`

Test suite for PageBlock model functionality.

Tests cover:

- PageBlock creation and relationships
- Cascade deletion behavior

Related Models:

- [*Page*](pages_app/models/page.py)

- [*Block*](pages_app/models/block.py)
- [*PageBlock*](pages_app/models/page_block.py)

setUp()

Initialize test data

test_pageblock_cascade_deletion()

Test cascade deletion behavior for PageBlock relationships.

Verifies:

- PageBlock is deleted when parent Page is deleted
- Associated Block remains after Page deletion
- Proper cleanup of relationship records

Related Models:

- [*Page*](pages_app/models/page.py)
- [*Block*](pages_app/models/block.py)
- [*PageBlock*](pages_app/models/page_block.py)

Database Constraints:

- ON DELETE CASCADE for page foreign key
- Block relationship preserved

test_pageblock_creation()

Test PageBlock model creation and relationships.

Verifies:

- PageBlock instance creation
- Correct page relationship
- Correct block relationship

Related Models:

- [*Page*](pages_app/models/page.py)
- [*Block*](pages_app/models/block.py)
- [*PageBlock*](pages_app/models/page_block.py)

class pages_app.tests.test_models.**PageModelTest**(*methodName='runTest'*)

Bases: TestCase

Test suite for Page model functionality. Tests creation, publication state, and timestamps.

setUp()

Initialize test data

test_page_creation()

Test page creation with all fields.

Verifies:

- Page title is correctly set
- URL path is correctly set

- Language is correctly set
- Page is not published by default

`test_page_language_validation()`

Test page language validation constraints.

Verifies that:

- Only valid language codes (EN/DE) are accepted
- Invalid language codes raise ValidationError

Related Model:
[*Page*](pages_app/models/page.py)

Raises
`ValidationError` – When invalid language code is used

`test_page_publish_unpublish()`

Test page publish and unpublish functionality.

Verifies:

- Page is unpublished by default
- Page can be published
- Page can be unpublished

`test_page_url_validation()`

Test page URL validation constraints.

Verifies that:

- URLs must start with forward slash
- Invalid URL formats raise ValidationError
- URL format matches routing requirements

Related Model:
[*Page*](pages_app/models/page.py)

Raises
`ValidationError` – When URL doesn't start with forward slash

`test_publish_unpublish()`

Test page publication state management.

Verifies:

- Default unpublished state
- Successful publication
- Successful unpublication
- State transitions

Related Model:
[*Page*](pages_app/models/page.py)

Test Steps:

1. Create unpublished page
2. Verify initial state
3. Publish page
4. Verify published state
5. Unpublish page
6. Verify final state

`test_url_path_validation()`

Test URL path format validation.

Verifies that:

- URL paths must start with forward slash
- Invalid URL paths raise ValidationError
- URL path format matches routing requirements

Related Model:
[*Page*](pages_app/models/page.py)

Raises
ValidationError – When URL path doesn't start with slash

`class pages_app.tests.test_models.TimestampedModelTest(methodName='runTest')`

Bases: `TestCase`

Test suite for timestamp functionality in Page model.

Tests automatic timestamp handling for:

- Creation timestamps
- Update timestamps
- Timestamp accuracy

Related Model:
[*Page*](pages_app/models/page.py)

`setUp()`

Initialize test data

`test_timestamps_creation()`

Test that timestamps are set on creation.

Verifies:

- created_at is set on creation
- updated_at is set on creation
- created_at and updated_at are equal on creation

`test_updated_at_auto_updates()`

Test that updated_at changes on update.

Verifies:

- updated_at is updated on save

- created_at remains unchanged on save

pages_app.tests.test_views module

class pages_app.tests.test_views.**ViewTests**(*methodName='runTest'*)

Bases: TestCase

Test suite for the views in the pages_app.

setUp()

Set up test data for the test cases.

tearDown()

Clean up after tests.

test_home_view()

Test the home page view.

Ensure that the home page is rendered correctly with the expected content.

test_render_page_existing_page()

Test rendering an existing page.

Ensure that an existing page is rendered correctly with the expected content.

test_render_page_nonexistent_page()

Test requesting a non-existent page returns 404.

Ensure that a request to a non-existent page returns a 404 status code.

test_render_page_unpublished_page()

Test that unpublished pages are not accessible.

Ensure that a request to an unpublished page returns a 404 status code.

Module contents

Test Suite for RIS CMS

This package contains test modules for the RIS CMS implementation:

Test Modules

- [*test_models.py*](pages_app/tests/test_models.py): Model validation and behavior
- [*test_views.py*](pages_app/tests/test_views.py): View rendering and routing
- [*test_cache.py*](pages_app/tests/test_cache.py): Redis caching implementation

Test Configuration

- Uses Django's TestCase
- Redis for cache testing
- PostgreSQL for database operations

Note

All tests can be run with:
python manage.py test pages_app.tests

Coverage reports are generated in coverage_report_*.txt

pages_app.views package

Submodules

pages_app.views.home module

pages_app.views.home.**render_page**(*request*, *url_path=''*)

Renders a page with caching support and language-specific content.

This view handles the main page rendering logic with the following features: - Path normalization - Language-specific content - Redis caching implementation - Dynamic block loading - Menu item integration

Parameters

- **request** (*HttpRequest*) – The HTTP request object
- **url_path** (*str, optional*) – The URL path to render. Defaults to empty string

Returns
Rendered page response

Return type
HttpResponse

Raises
Http404 – If the requested page is not found or not published

Cache:

- Key Format: "page_{normalized_path}_{lang}"
- Storage: Redis (configured in settings.CACHES)
- TTL: 15 minutes (settings.CACHE_TTL)

Related Models:

- [*Page*](pages_app/models/page.py): Content and metadata
- [*Block*](pages_app/models/block.py): Reusable content blocks
- [*MenuItem*](pages_app/models/menu_item.py): Navigation elements
- [*PageBlock*](pages_app/models/page_block.py): Page-Block relationships

pages_app.views.page module

Page rendering view module.

This module handles the rendering of pages based on URL paths and manages the retrieval of related page blocks and menu items.

pages_app.views.page.**render_page**(*request*, *url_path=''*)

Render a page based on its URL path.

This view handles the rendering of pages, including their associated blocks and menu items. It ensures that only published pages are displayed.

Parameters

- **request** (*HttpRequest*) – The Django request object.
- **url_path** (*str, optional*) – The URL path to the page. Defaults to empty string.

Returns
Rendered page with its context.

Return type
HttpResponse

Raises
Http404 – If the requested page doesn't exist or isn't published.

> **Note**
>
> The page must be published (published_at not null and <= current time) and match the current language to be displayed.

Module contents

Pages App Views Module

Handles all view logic for the RIS CMS pages application. Implements caching and multilingual page rendering.

Key Features:

- Page rendering with Redis cache
- Multi-language support (DE/EN)
- Dynamic block rendering
- Menu item handling

Components:

- render_page: Main page rendering function with caching
- error handlers: 404, 500 error handling
- cache invalidation

Dependencies:

- Django shortcuts
- Django cache framework
- Django utils
- Pages app models

pages_app.views.**render_page**(*request*, *url_path=''*)

Renders a page with caching support and language-specific content.

This view handles the main page rendering logic with the following features: - Path normalization - Language-specific content - Redis caching implementation - Dynamic block loading - Menu item integration

Parameters

- **request** (*HttpRequest*) – The HTTP request object
- **url_path** (*str, optional*) – The URL path to render. Defaults to empty string

Returns
Rendered page response

Return type
HttpResponse

Raises
Http404 – If the requested page is not found or not published

Cache:

- Key Format: "page_{normalized_path}_{lang}"
- Storage: Redis (configured in settings.CACHES)
- TTL: 15 minutes (settings.CACHE_TTL)

Related Models:

- [*Page*](pages_app/models/page.py): Content and metadata
- [*Block*](pages_app/models/block.py): Reusable content blocks
- [*MenuItem*](pages_app/models/menu_item.py): Navigation elements
- [*PageBlock*](pages_app/models/page_block.py): Page-Block relationships

Submodules

pages_app.admin module

class pages_app.admin.**BlockAdmin**(*model*, *admin_site*)

Bases: SimpleHistoryAdmin

Admin view for Block model with history tracking.

history_list_display = ['name', 'sorting']

list_display = ('name', 'sorting')

property media

class pages_app.admin.**MenuItemAdmin**(*model*, *admin_site*)

Bases: SimpleHistoryAdmin

Admin view for MenuItem model with history tracking.

history_list_display = ['template']

list_display = ('name', 'sorting')

property media

class pages_app.admin.**PageAdmin**(*model*, *admin_site*)

Bases: SimpleHistoryAdmin

Admin view for Page model with history tracking.

history_list_display = ['title', 'language', 'is_published']

list_display = ('title', 'language', 'is_published')

property media

search_fields = ['title', 'language']

class pages_app.admin.PageBlockAdmin(*model*, *admin_site*)

Bases: SimpleHistoryAdmin

Admin view for PageBlock model with history tracking.

get_block_name(*obj*)

Retrieve the name of the associated Block.

Parameters
obj – The PageBlock instance.

Returns
The name of the associated Block.

get_page_title(*obj*)

Retrieve the title of the associated Page.

Parameters
obj – The PageBlock instance.

Returns
The title of the associated Page.

history_list_display = ['get_page_title', 'get_block_name']

list_display = ('get_page_title', 'get_block_name')

list_filter = ('page__title', 'block__name')

property media

search_fields = ('page__title', 'block__name')

pages_app.apps module

class pages_app.apps.PagesAppConfig(*app_name*, *app_module*)

Bases: AppConfig

Configuration class for the pages_app Django application.

default_auto_field = 'django.db.models.BigAutoField'

name = 'pages_app'

ready()

Override this method to include the application's signal handlers.

pages_app.urls module

URL Configuration for Pages App

This module defines URL patterns for the dynamic content management system. All paths are handled by a single view function that loads content from the database.

URL Patterns:

1. Root URL ('/') - Serves as the home page - Maps to home view with empty url_path parameter - Example: http://ris.dev/
2. Dynamic Pages - Matches any URL path using regex pattern - Passes path as 'url_path' parameter to view - Loads corresponding page from database - Example: http://ris.dev/about/, http://ris.dev/services/

Pattern Matching:

- URLs are normalized to include leading slash
- Trailing slashes are optional
- Language prefix is handled via query parameter
- Invalid paths redirect to home page

Example Usage:

URL: /services/web-development/ - Normalized path: /services/web-development/ - View receives: url_path='services/web-development' - Loads page with matching url_path from database - Falls back to home page if not found

See Also:

- pages_app.views.home: Main view function handling all requests
- pages_app.models.Page: Database model for page content

pages_app.views module

Pages App Views Module

Handles all view logic for the RIS CMS pages application. Implements caching and multilingual page rendering.

Key Features:

- Page rendering with Redis cache
- Multi-language support (DE/EN)
- Dynamic block rendering
- Menu item handling

Components:

- render_page: Main page rendering function with caching
- error handlers: 404, 500 error handling
- cache invalidation

Dependencies:

- Django shortcuts
- Django cache framework
- Django utils
- Pages app models

pages_app.views.**render_page**(*request*, *url_path=''*)

Renders a page with caching support and language-specific content.

This view handles the main page rendering logic with the following features: - Path normalization - Language-specific content - Redis caching implementation - Dynamic block loading - Menu item integration

Parameters

- **request** (*HttpRequest*) – The HTTP request object
- **url_path** (*str, optional*) – The URL path to render. Defaults to empty string

Returns
Rendered page response

Return type
HttpResponse

Raises
Http404 – If the requested page is not found or not published

Cache:

- Key Format: "page_{normalized_path}_{lang}"
- Storage: Redis (configured in settings.CACHES)
- TTL: 15 minutes (settings.CACHE_TTL)

Related Models:

- [*Page*](pages_app/models/page.py): Content and metadata
- [*Block*](pages_app/models/block.py): Reusable content blocks
- [*MenuItem*](pages_app/models/menu_item.py): Navigation elements
- [*PageBlock*](pages_app/models/page_block.py): Page-Block relationships

Module contents

Pages Application Package.

This package provides functionality for managing and rendering dynamic pages in a Django application. It includes models for pages, blocks, and menu items, along with the necessary views and utilities.

pages_app.**__version__**

The current version of the pages application.

Type
str

pages_app.**__author__**

The author of the package.

Type
str

CHAPTER

THREE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

p

INDEX

Symbols

A

B

C

D

G

H

I

L

M

N

O

P

T

U

V