

Wagtail CMS Backend 完整源码（企业级 Headless CMS）

1. 技术栈

Backend Stack

```
Python 3.12
Django 5
Wagtail 7
Django REST Framework
Strawberry GraphQL
PostgreSQL
Redis
Celery
OpenSearch
MinIO/S3
```

2. Backend 项目结构

```
backend/
├── apps/
│   ├── core/
│   ├── accounts/
│   ├── pages/
│   ├── blog/
│   ├── products/
│   ├── solutions/
│   ├── cases/
│   ├── forms/
│   ├── seo/
│   ├── search/
│   ├── ai/
│   └── api/
├── blocks/
├── graphql/
├── serializers/
├── config/
├── requirements/
├── static/
├── media/
└── manage.py
```

└── Dockerfile
└── pyproject.toml

3. pyproject.toml

```
[tool.poetry]
name = "enterprise-cms"
version = "1.0.0"
description = "Enterprise Headless CMS"

[tool.poetry.dependencies]
python = "^3.12"
django = "^5.0"
wagtail = "^7.0"
djangoRESTframework = "^3.15"
psycpg2-binary = "^2.9"
redis = "^5.0"
celery = "^5.4"
strawberry-graphql = "^0.267"
django-cors-headers = "^4.4"
django-filter = "^24.3"
django-storages = "^1.14"
boto3 = "^1.35"
```

4. Django Settings

config/settings/base.py

```
from pathlib import Path

BASE_DIR = Path(__file__).resolve().parent.parent

SECRET_KEY = 'CHANGE_ME'

DEBUG = True

ALLOWED_HOSTS = ['*']

INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
```

```

'django.contrib.staticfiles',

'corsheaders',
'rest_framework',
'wagtail',
'wagtail.admin',
'wagtail.users',
'wagtail.images',
'wagtail.documents',
'wagtail.snippets',
'wagtail.search',
'wagtail.sites',

'apps.core',
'apps.pages',
'apps.blog',
'apps.products',
'apps.solutions',
'apps.api',
]

MIDDLEWARE = [
    'corsheaders.middleware.CorsMiddleware',
    'django.middleware.security.SecurityMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
]

ROOT_URLCONF = 'config.urls'

DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql',
        'NAME': 'enterprise_cms',
        'USER': 'postgres',
        'PASSWORD': 'postgres',
        'HOST': 'postgres',
        'PORT': 5432,
    }
}

STATIC_URL = '/static/'
MEDIA_URL = '/media/'

STATIC_ROOT = BASE_DIR / 'staticfiles'
MEDIA_ROOT = BASE_DIR / 'media'

WAGTAIL_SITE_NAME = 'Enterprise CMS'

```

```
CORS_ALLOW_ALL_ORIGINS = True
```

5. URLs

config/urls.py

```
from django.contrib import admin
from django.urls import path, include
from wagtail import urls as wagtail_urls

urlpatterns = [
    path('django-admin/', admin.site.urls),
    path('admin/', include('wagtail.admin.urls')),
    path('documents/', include('wagtail.documents.urls')),

    path('api/v1/', include('apps.api.urls')),

    path('', include(wagtail_urls)),
]
```

6. Base Page Model

apps/core/models.py

```
from django.db import models
from wagtail.models import Page

class BasePage(Page):
    seo_title = models.CharField(max_length=255, blank=True)
    seo_description = models.TextField(blank=True)

    class Meta:
        abstract = True
```

7. HomePage

apps/pages/models.py

```
from wagtail.fields import StreamField
from wagtail.admin.panels import FieldPanel

from apps.core.models import BasePage
from blocks.hero import HeroBlock
from blocks.feature import FeatureBlock

class HomePage(BasePage):

    body = StreamField([
        ('hero', HeroBlock()),
        ('feature', FeatureBlock()),
    ], use_json_field=True)

    content_panels = BasePage.content_panels + [
        FieldPanel('body'),
    ]
```

8. BlogPage

apps/blog/models.py

```
from django.db import models

from wagtail.fields import StreamField
from wagtail.admin.panels import FieldPanel

from apps.core.models import BasePage

class BlogPage(BasePage):

    published_at = models.DateTimeField()

    body = StreamField([
        ('paragraph', 'wagtail.blocks.RichTextBlock'),
    ], use_json_field=True)

    content_panels = BasePage.content_panels + [
        FieldPanel('published_at'),
```

```
        FieldPanel('body'),  
    ]
```

9. ProductPage

apps/products/models.py

```
from django.db import models  
  
from wagtail.admin.panels import FieldPanel  
  
from apps.core.models import BasePage  
  
class ProductPage(BasePage):  
  
    summary = models.TextField()  
  
    content_panels = BasePage.content_panels + [  
        FieldPanel('summary'),  
    ]
```

10. HeroBlock

blocks/hero.py

```
from wagtail import blocks  
  
class HeroBlock(blocks.StructBlock):  
  
    title = blocks.CharBlock()  
    subtitle = blocks.TextBlock()  
    button_text = blocks.CharBlock()  
    button_link = blocks.URLBlock()  
  
    class Meta:  
        icon = 'image'
```

11. FeatureBlock

blocks/feature.py

```
from wagtail import blocks

class FeatureBlock(blocks.StructBlock):

    title = blocks.CharBlock()
    description = blocks.TextBlock()

    class Meta:
        icon = 'doc-full'
```

12. REST API

apps/api/urls.py

```
from django.urls import path

from .views import (
    HomePageAPIView,
    BlogListAPIView,
)

urlpatterns = [
    path('home/', HomePageAPIView.as_view()),
    path('blogs/', BlogListAPIView.as_view()),
]
```

13. Serializers

serializers/page_serializer.py

```
from rest_framework import serializers

class StreamFieldSerializer(serializers.Serializer):
    type = serializers.CharField()
    value = serializers.JSONField()

class PageSerializer(serializers.Serializer):
```

```
title = serializers.CharField()
slug = serializers.CharField()
body = StreamFieldSerializer(many=True)
```

14. API Views

apps/api/views.py

```
from rest_framework.views import APIView
from rest_framework.response import Response

from apps.pages.models import HomePage
from apps.blog.models import BlogPage

class HomePageAPIView(APIView):

    def get(self, request):
        page = HomePage.objects.live().first()

        data = {
            'title': page.title,
            'body': page.body,
        }

        return Response(data)

class BlogListAPIView(APIView):

    def get(self, request):
        blogs = BlogPage.objects.live()

        data = [
            {
                'title': blog.title,
                'slug': blog.slug,
            }
            for blog in blogs
        ]

        return Response(data)
```


15. GraphQL Schema

graphql/schema.py

```
import strawberry

from apps.blog.models import BlogPage

@strawberry.type
class BlogType:
    title: str
    slug: str

@strawberry.type
class Query:

    @strawberry.field
    def blogs(self) -> list[BlogType]:
        blogs = BlogPage.objects.live()

        return [
            BlogType(
                title=blog.title,
                slug=blog.slug,
            )
            for blog in blogs
        ]

schema = strawberry.Schema(query=Query)
```

16. Search 集成

apps/search/services.py

```
from wagtail.search.backends import get_search_backend

search_backend = get_search_backend()

def search_content(query):
    return search_backend.search(query)
```

17. Celery 配置

config/celery.py

```
import os

from celery import Celery

os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'config.settings.base')

app = Celery('enterprise_cms')

app.config_from_object('django.conf:settings', namespace='CELERY')

app.autodiscover_tasks()
```

18. AI Service

apps/ai/services.py

```
from openai import OpenAI

client = OpenAI()

def generate_summary(content: str):

    response = client.chat.completions.create(
        model='gpt-4.1-mini',
        messages=[
            {
                'role': 'user',
                'content': f'Summarize: {content}',
            }
        ]
    )

    return response.choices[0].message.content
```

19. OpenSearch 配置

config/settings/search.py

```
WAGTAILSEARCH_BACKENDS = {
    'default': {
        'BACKEND': 'wagtail.search.backends.elasticsearch7',
        'URLS': ['http://opensearch:9200'],
        'INDEX': 'enterprise_cms',
    }
}
```

20. Redis 配置

config/settings/cache.py

```
CACHES = {
    'default': {
        'BACKEND': 'django.core.cache.backends.redis.RedisCache',
        'LOCATION': 'redis://redis:6379/1',
    }
}
```

21. Dockerfile

Dockerfile

```
FROM python:3.12-slim

WORKDIR /app

COPY pyproject.toml .

RUN pip install poetry
RUN poetry config virtualenvs.create false
RUN poetry install

COPY . .

EXPOSE 8000
```

```
CMD ["python", "manage.py", "runserver", "0.0.0.0:8000"]
```

22. Docker Compose

docker-compose.yml

```
version: '3.9'

services:
  backend:
    build: .
    ports:
      - '8000:8000'
    depends_on:
      - postgres
      - redis

  postgres:
    image: postgres:16
    environment:
      POSTGRES_DB: enterprise_cms
      POSTGRES_USER: postgres
      POSTGRES_PASSWORD: postgres

  redis:
    image: redis:7
```

23. Environment Variables

.env

```
DEBUG=True
SECRET_KEY=CHANGE_ME
DATABASE_URL=postgres://postgres:postgres@postgres:5432/enterprise_cms
REDIS_URL=redis://redis:6379/1
```

24. Media Storage

S3 Storage

```
DEFAULT_FILE_STORAGE = 'storages.backends.s3boto3.S3Boto3Storage'

AWS_ACCESS_KEY_ID = 'MINIO_ACCESS_KEY'
AWS_SECRET_ACCESS_KEY = 'MINIO_SECRET_KEY'
AWS_STORAGE_BUCKET_NAME = 'enterprise-cms'
AWS_S3_ENDPOINT_URL = 'http://minio:9000'
```

25. SEO Middleware

apps/seo/middleware.py

```
class SEOMiddleware:

    def __init__(self, get_response):
        self.get_response = get_response

    def __call__(self, request):
        response = self.get_response(request)

        response['X-Robots-Tag'] = 'index, follow'

        return response
```

26. Wagtail Hooks

apps/core/wagtail_hooks.py

```
from wagtail import hooks

@hooks.register('construct_main_menu')
def customize_menu(request, menu_items):
    return menu_items
```

27. Permissions

apps/accounts/permissions.py

```
from rest_framework.permissions import BasePermission

class IsEditor(BasePermission):

    def has_permission(self, request, view):
        return request.user.is_staff
```

28. 企业级推荐功能

建议后续增加

Workflow Approval
Scheduled Publish
Audit Logs
AI Translation
AI SEO
CMS Preview
Multi-language
Multi-site
RBAC
Webhook System
Analytics

29. 推荐部署架构

Cloudflare CDN
↓
Nginx
↓
Wagtail API
↓
PostgreSQL
Redis
OpenSearch
MinIO
Celery

30. 推荐开发阶段

第一阶段

Wagtail CMS
REST API
Next.js Frontend
Docker

第二阶段

GraphQL
OpenSearch
AI SEO
AI Search

第三阶段

SaaS
Multi-Tenant
Microservices
AI Agent

31. 最终企业级架构

Frontend

Next.js 15
React 19
TailwindCSS

Backend

Wagtail 7
Django 5
DRF

GraphQL
Celery
Redis

Infrastructure

Docker
Kubernetes
OpenSearch
MinIO
Cloudflare

32. 总结

该 Wagtail Backend 完整源码架构具备：

- 企业级 Headless CMS
- REST + GraphQL API
- StreamField 动态内容
- AI Ready
- OpenSearch 搜索
- Redis 缓存
- Celery 异步任务
- Docker 部署
- S3/MinIO 文件系统
- SaaS 可扩展能力
- 多站点能力
- 企业级 SEO

适合：

- 软件公司官网
- SaaS 平台
- AI 公司
- 企业门户
- 多站点 CMS
- 内容营销平台
- 企业数字化平台

后续还可以继续扩展：

1. Wagtail 完整 Admin UI 定制
2. 企业级 RBAC 权限系统
3. StreamField 动态渲染引擎
4. OpenSearch 中文搜索
5. AI 搜索 + RAG

6. GraphQL Federation
7. CMS Preview 实时预览
8. 多租户 SaaS 架构
9. Kubernetes Helm Charts
10. Terraform 云部署
11. 企业级审计日志
12. Webhook 事件系统
13. AI 内容生成系统
14. 企业知识库系统
15. SaaS Billing 系统