Backend Developer Guide

Django + Wagtail Multi-Tenant SaaS Platform

📋 Table of Contents

🎯 Overview

This backend powers a multi-tenant SaaS platform where:

- Users can select React templates
- Each user gets their own subdomain/domain
- Content is managed through Wagtail CMS
- Deployments happen automatically via GitHub Actions

Key Features

✅ **Multi-tenant architecture**
✅ **Wagtail CMS for content management**
✅ **REST API for frontend templates**

✅ **Automated deployment pipeline**
✅ **Domain-based tenant detection**

🏗️ Architecture

```
┌─────────────────────────────────────────────────┐
│           Django + Wagtail                      │
│                                                 │
│  ┌──────────────┐    ┌──────────────┐           │
│  │  Tenants     │    │  Templates   │           │
│  │  Model       │───▶│   Model      │           │
│  └──────────────┘    └──────────────┘           │
│          │                   │                  │
│          ▼           ▼                          │
│  ┌──────────────────────────────────┐           │
│  │       REST API                   │           │
│  │ /api/tenant/?domain=...          │           │
│  │ /api/deploy/                     │           │
│  │ /api/pages/                      │           │
│  └──────────────────────────────────┘           │
└─────────────────────────────────────────────────┘
              │
              ▼
┌─────────────────────────────────────────────────┐
│          GitHub Actions API                     │
│  Triggers template build & deployment           │
└─────────────────────────────────────────────────┘
              │
              ▼
┌─────────────────────────────────────────────────┐
│           VPS (Apache2)                         │
│  Serves React templates at tenant domains       │
└─────────────────────────────────────────────────┘
```

🛠️ Setup & Installation

Prerequisites

```bash
```

```
Python 3.10+
PostgreSQL 13+
Redis (for caching)
Apache2 (on VPS)
```

Step 1: Create Project

```bash
# Create project directory
mkdir saas-backend
cd saas-backend

# Create virtual environment
python -m venv .venv
source .venv/bin/activate  # On Windows: .venv\Scripts\activate

# Install Django and Wagtail
pip install Django==4.2
pip install wagtail==5.2
pip install djangorestframework
pip install django-cors-headers
pip install psycopg2-binary
pip install requests
pip install python-decouple

# Create Django project
django-admin startproject config .

# Create apps
python manage.py startapp tenants
python manage.py startapp templates
python manage.py startapp api
```

Step 2: Project Structure

```
saas-backend/
├── config/
│   ├── __init__.py
│   ├── settings.py
│   ├── urls.py
│   └── wsgi.py
│
```

```
├── tenants/
│   ├── models.py
│   ├── views.py
│   ├── admin.py
│   └── serializers.py
│
├── templates_app/
│   ├── models.py
│   ├── views.py
│   └── admin.py
│
├── api/
│   ├── views.py
│   ├── urls.py
│   └── serializers.py
│
├── manage.py
└── requirements.txt
```

🗄️ Database Models

Tenant Model

```python
```

```python
# tenants/models.py
from django.db import models
from django.contrib.auth.models import User

class Tenant(models.Model):
    """
    Represents a SaaS customer/user who has selected a template
    """
    user = models.ForeignKey(User, on_delete=models.CASCADE)
    name = models.CharField(max_length=200)
    domain = models.CharField(max_length=255, unique=True)
    subdomain = models.CharField(max_length=100, unique=True)

    # Template selection
    template = models.ForeignKey('templates_app.Template', on_delete=models.PROTECT)

    # Branding
    logo = models.ImageField(upload_to='tenant_logos/', blank=True)
    primary_color = models.CharField(max_length=7, default='#3b82f6')
    secondary_color = models.CharField(max_length=7, default='#1e40af')
    accent_color = models.CharField(max_length=7, default='#f59e0b')
    font_family = models.CharField(max_length=100, default='Inter, sans-serif')

    # Status
    is_active = models.BooleanField(default=True)
    deployed = models.BooleanField(default=False)
    deployment_status = models.CharField(
        max_length=20,
        choices=[
            ('pending', 'Pending'),
            ('deploying', 'Deploying'),
            ('live', 'Live'),
            ('failed', 'Failed')
        ],
        default='pending'
    )

    # Metadata
    created_at = models.DateTimeField(auto_now_add=True)
    updated_at = models.DateTimeField(auto_now=True)

    class Meta:
        db_table = 'tenants'
        ordering = ['-created_at']

    def __str__(self):
        return f"{self.name} ({self.domain})"

    def get_full_domain(self):
        """Returns full domain URL"""
        return f"https://{self.domain}"
```

Template Model

```
python
```

```python
# templates_app/models.py
from django.db import models

class Template(models.Model):
    """
    Represents available React templates users can choose
    """
    name = models.CharField(max_length=100)
    slug = models.SlugField(unique=True)
    description = models.TextField()

    # GitHub info
    folder_name = models.CharField(max_length=100)  # e.g., 'school-template'
    github_path = models.CharField(max_length=255)

    # Preview
    thumbnail = models.ImageField(upload_to='template_thumbnails/')
    demo_url = models.URLField(blank=True)

    # Categorization
    category = models.CharField(
        max_length=50,
        choices=[
            ('education', 'Education'),
            ('healthcare', 'Healthcare'),
            ('restaurant', 'Restaurant'),
            ('ecommerce', 'E-Commerce'),
            ('portfolio', 'Portfolio'),
        ]
    )

    # Status
    is_active = models.BooleanField(default=True)
    featured = models.BooleanField(default=False)

    # Metadata
    created_at = models.DateTimeField(auto_now_add=True)

    class Meta:
        db_table = 'templates'
        ordering = ['name']
    def __str__(self):
        return self.name
```

Wagtail Page Models

```
python
```

```python
# tenants/wagtail_models.py
from wagtail.models import Page
from wagtail.fields import RichTextField, StreamField
from wagtail.admin.panels import FieldPanel
from wagtail import blocks
from wagtail.images.blocks import ImageChooserBlock

class TenantHomePage(Page):
    """
    Home page for each tenant site
    """
    tenant = models.OneToOneField(
        'tenants.Tenant',
        on_delete=models.CASCADE,
        related_name='homepage'
    )

    hero_title = models.CharField(max_length=200)
    hero_subtitle = models.TextField()
    hero_image = models.ForeignKey(
        'wagtailimages.Image',
        null=True,
        blank=True,
        on_delete=models.SET_NULL,
        related_name='+'
    )

    body = StreamField([
        ('heading', blocks.CharBlock()),
        ('paragraph', blocks.RichTextBlock()),
        ('image', ImageChooserBlock()),
        ('html', blocks.RawHTMLBlock()),
    ], use_json_field=True, blank=True)

    content_panels = Page.content_panels + [
        FieldPanel('hero_title'),
        FieldPanel('hero_subtitle'),
        FieldPanel('hero_image'),
        FieldPanel('body'),
    ]

    class Meta:
        verbose_name = "Home Page"
```

```
class ContentPage(Page):
    """
    Generic content page for tenants
    """
    body = StreamField([
        ('heading', blocks.CharBlock()),
        ('paragraph', blocks.RichTextBlock()),
        ('image', ImageChooserBlock()),
    ], use_json_field=True)

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

🌐 Wagtail Multi-Site Configuration

Step 1: Configure Settings

```
python
```

```
# config/settings.py
INSTALLED_APPS = [
    # Django apps
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',

    # Wagtail apps
    'wagtail.contrib.forms',
    'wagtail.contrib.redirects',
    'wagtail.embeds',
    'wagtail.sites',
    'wagtail.users',
    'wagtail.snippets',
    'wagtail.documents',
    'wagtail.images',
    'wagtail.search',
    'wagtail.admin',
    'wagtail',

    'modelcluster',
    'taggit',

    # Third party
    'rest_framework',
    'corsheaders',

    # Custom apps
    'tenants',
    'templates_app',
    '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',
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
    'wagtail.contrib.redirects.middleware.RedirectMiddleware',
]

# CORS Settings
CORS_ALLOWED_ORIGINS = [
    "http://localhost:5173",
    "https://yourdomain.com",
]
CORS_ALLOW_CREDENTIALS = True

# REST Framework
REST_FRAMEWORK = {
    'DEFAULT_PERMISSION_CLASSES': [
        'rest_framework.permissions.AllowAny',
    ],
    'DEFAULT_RENDERER_CLASSES': [
        'rest_framework.renderers.JSONRenderer',
    ],
}

# Wagtail Settings
WAGTAIL_SITE_NAME = 'SaaS Platform'
WAGTAILADMIN_BASE_URL = 'https://admin.yourdomain.com'
```

Step 2: Create Sites in Django Admin

python

```python
# After migrations, create sites programmatically
from django.contrib.sites.models import Site
from wagtail.models import Site as WagtailSite
from tenants.models import Tenant

def create_tenant_site(tenant):
    """
    Create Wagtail site for a tenant
    """
    # Create Django site
    django_site = Site.objects.create(
        domain=tenant.domain,
        name=tenant.name
    )

    # Create Wagtail site
    wagtail_site = WagtailSite.objects.create(
        hostname=tenant.domain,
        site_name=tenant.name,
        root_page=tenant.homepage,
        is_default_site=False
    )

    return wagtail_site
```

🔌 API Endpoints

Serializers

```
python
```

```
# api/serializers.py
from rest_framework import serializers
from tenants.models import Tenant
from templates_app.models import Template

class TemplateSerializer(serializers.ModelSerializer):
    class Meta:
        model = Template
        fields = [
            'id', 'name', 'slug', 'description',
            'thumbnail', 'demo_url', 'category'
        ]


class TenantSerializer(serializers.ModelSerializer):
    template_name = serializers.CharField(source='template.name', read_only=True)
    logo_url = serializers.SerializerMethodField()

    class Meta:
        model = Tenant
        fields = [
            'id', 'name', 'domain', 'subdomain',
            'template_name', 'logo_url',
            'primary_color', 'secondary_color', 'accent_color',
            'font_family', 'is_active', 'deployed',
            'deployment_status'
        ]

    def get_logo_url(self, obj):
        if obj.logo:
            request = self.context.get('request')
            return request.build_absolute_uri(obj.logo.url)
        return None
```

Views

```
python
```

```python
# api/views.py
from rest_framework.views import APIView
from rest_framework.response import Response
from rest_framework import status
from django.shortcuts import get_object_or_404
import requests
from decouple import config

from tenants.models import Tenant
from templates_app.models import Template
from .serializers import TenantSerializer, TemplateSerializer


class TenantByDomainView(APIView):
    """
    GET /api/tenant/?domain=school123.yourdomain.com
    Returns tenant configuration for React frontend
    """
    def get(self, request):
        domain = request.GET.get('domain')

        if not domain:
            return Response(
                {'error': 'domain parameter required'},
                status=status.HTTP_400_BAD_REQUEST
            )

        try:
            tenant = Tenant.objects.get(domain=domain, is_active=True)
            serializer = TenantSerializer(tenant, context={'request': request})
            return Response(serializer.data)
        except Tenant.DoesNotExist:
            return Response(
                {'error': 'Tenant not found'},
                status=status.HTTP_404_NOT_FOUND
            )


class TemplateListView(APIView):
    """
    GET /api/templates/
    Returns list of available templates
    """
    def get(self, request):
        templates = Template.objects.filter(is_active=True)
        serializer = TemplateSerializer(templates, many=True, context={'request': request})
        return Response(serializer.data)


class DeploymentView(APIView):
    """
    POST /api/deploy/
    Triggers automatic deployment via GitHub Actions
    """
    def post(self, request):
        user_id = request.data.get('user_id')
        template_id = request.data.get('template_id')
        domain = request.data.get('domain')

        if not all([user_id, template_id, domain]):
            return Response(
                {'error': 'user_id, template_id, and domain are required'},
                status=status.HTTP_400_BAD_REQUEST
            )

        try:
            template = Template.objects.get(id=template_id)

            # Create tenant record
            tenant = Tenant.objects.create(
                user_id=user_id,
                name=domain.split('.')[0].title(),
                domain=domain,
                subdomain=domain.split('.')[0],
                template=template,
                deployment_status='deploying'
            )

            # Trigger GitHub Actions
            success = self.trigger_github_deployment(
                user_id=str(tenant.id),
                template=template.folder_name,
                domain=domain
            )

            if success:
                return Response({
```

```
                'status': 'deploying',
                'tenant_id': tenant.id,
                'message': 'Deployment started successfully'
            })
        else:
            tenant.deployment_status = 'failed'
            tenant.save()
            return Response(
                {'error': 'Failed to trigger deployment'},
                status=status.HTTP_500_INTERNAL_SERVER_ERROR
            )

    except Template.DoesNotExist:
        return Response(
            {'error': 'Template not found'},
            status=status.HTTP_404_NOT_FOUND
        )

def trigger_github_deployment(self, user_id, template, domain):
    """
    Triggers GitHub Actions workflow via repository_dispatch
    """
    github_token = config('GITHUB_TOKEN')
    github_repo = config('GITHUB_REPO')  # format: 'username/repo'

    webhook_url = f"{config('BACKEND_URL')}/api/webhook/deployment/{user_id}/"

    response = requests.post(
        f'https://api.github.com/repos/{github_repo}/dispatches',
        headers={
            'Authorization': f'token {github_token}',
            'Accept': 'application/vnd.github.v3+json'
        },
        json={
            'event_type': 'deploy_template',
            'client_payload': {
                'user_id': user_id,
                'template': template,
                'domain': domain,
                'webhook_url': webhook_url
            }
        }
    )
```

```python
        return response.status_code == 204


class DeploymentWebhookView(APIView):
    """
    POST /api/webhook/deployment/<user_id>/
    Receives deployment status updates from GitHub Actions
    """
    def post(self, request, user_id):
        try:
            tenant = Tenant.objects.get(id=user_id)

            status_update = request.data.get('status')

            if status_update == 'live':
                tenant.deployment_status = 'live'
                tenant.deployed = True
                tenant.save()

                # Send notification to user
                self.notify_user(tenant, 'Your site is now live!')

            elif status_update == 'failed':
                tenant.deployment_status = 'failed'
                tenant.save()

                # Send failure notification
                self.notify_user(tenant, 'Deployment failed')

            return Response({'received': True})

        except Tenant.DoesNotExist:
            return Response(
                {'error': 'Tenant not found'},
                status=status.HTTP_404_NOT_FOUND
            )

    def notify_user(self, tenant, message):
        """
        Send notification to user (email, websocket, etc.)
        """
        # Implement your notification logic here
        pass
```

URLs

python

```python
# api/urls.py
from django.urls import path
from .views import (
    TenantByDomainView,
    TemplateListView,
    DeploymentView,
    DeploymentWebhookView
)

app_name = 'api'

urlpatterns = [
    path('tenant/', TenantByDomainView.as_view(), name='tenant-by-domain'),
    path('templates/', TemplateListView.as_view(), name='template-list'),
    path('deploy/', DeploymentView.as_view(), name='deploy'),
    path('webhook/deployment/<int:user_id>/', DeploymentWebhookView.as_view(), name='deployment-webhook'),
]
```

python

```python
# config/urls.py
from django.contrib import admin
from django.urls import path, include
from wagtail.admin import urls as wagtailadmin_urls
from wagtail import urls as wagtail_urls

urlpatterns = [
    path('django-admin/', admin.site.urls),
    path('admin/', include(wagtailadmin_urls)),
    path('api/', include('api.urls')),
    path('', include(wagtail_urls)),
]
```

🚀 Deployment Automation

Environment Variables

python

```
# .env
SECRET_KEY=your-secret-key
DEBUG=False
DATABASE_URL=postgresql://user:password@localhost/dbname

GITHUB_TOKEN=ghp_your_github_personal_access_token
GITHUB_REPO=yourusername/saas-frontend-templates
BACKEND_URL=https://api.yourdomain.com

VPS_HOST=your-vps-ip
VPS_USER=root
```

Management Command for Initial Setup

python

```python
# tenants/management/commands/setup_templates.py
from django.core.management.base import BaseCommand
from templates_app.models import Template

class Command(BaseCommand):
    help = 'Setup initial templates'

    def handle(self, *args, **options):
        templates = [
            {
                'name': 'School Template',
                'slug': 'school-template',
                'folder_name': 'school-template',
                'description': 'Perfect for educational institutions',
                'category': 'education',
            },
            {
                'name': 'Hospital Template',
                'slug': 'hospital-template',
                'folder_name': 'hospital-template',
                'description': 'Healthcare and medical services',
                'category': 'healthcare',
            },
            {
                'name': 'Restaurant Template',
                'slug': 'restaurant-template',
                'folder_name': 'restaurant-template',
                'description': 'Food and dining businesses',
                'category': 'restaurant',
            },
        ]

        for template_data in templates:
            Template.objects.get_or_create(
                slug=template_data['slug'],
                defaults=template_data
            )
            self.stdout.write(f"✓ Created {template_data['name']}")
```

🧪 Testing

Test Tenant API

python

```python
# api/tests.py
from django.test import TestCase
from rest_framework.test import APIClient
from tenants.models import Tenant
from templates_app.models import Template

class TenantAPITest(TestCase):
    def setUp(self):
        self.client = APIClient()
        self.template = Template.objects.create(
            name='Test Template',
            slug='test-template',
            folder_name='test-template',
            category='education'
        )
        self.tenant = Tenant.objects.create(
            name='Test School',
            domain='test.example.com',
            subdomain='test',
            template=self.template
        )

    def test_get_tenant_by_domain(self):
        response = self.client.get('/api/tenant/?domain=test.example.com')
        self.assertEqual(response.status_code, 200)
        self.assertEqual(response.data['name'], 'Test School')
```

🌐 Production Deployment

Step 1: Database Setup

bash

```
# Create PostgreSQL database
sudo -u postgres psql
CREATE DATABASE saas_platform;
CREATE USER saas_user WITH PASSWORD 'secure_password';
GRANT ALL PRIVILEGES ON DATABASE saas_platform TO saas_user;
\q
```

Step 2: Deploy Backend

```bash
# On VPS
cd /var/www
git clone https://github.com/yourusername/saas-backend.git
cd saas-backend

python3 -m venv .venv
source .venv/bin/activate

pip install -r requirements.txt

python manage.py migrate
python manage.py createsuperuser
python manage.py setup_templates
python manage.py collectstatic

# Run with Gunicorn
gunicorn config.wsgi:application --bind 0.0.0.0:8000
```

Step 3: Apache Configuration

```apache
<VirtualHost *:80>
    ServerName api.yourdomain.com

    ProxyPreserveHost On
    ProxyPass / http://127.0.0.1:8000/
    ProxyPassReverse / http://127.0.0.1:8000/

    ErrorLog ${APACHE_LOG_DIR}/api-error.log
    CustomLog ${APACHE_LOG_DIR}/api-access.log combined
</VirtualHost>
```

✅ Summary

This backend provides:
✅ Multi-tenant management
✅ Wagtail CMS integration
✅ REST API for React frontends
✅ Automated deployment via GitHub Actions
✅ Domain-based tenant routing

Version: 1.0
Last Updated: December 2024