REF-Manager

Technical Documentation

Developer and Administrator Reference

University of York

November 2024

Contents

1 Architecture Overview

1.1 Technology Stack

Layer	Technology
Backend	Django 4.2, Python 3.10+
Database	SQLite (dev), PostgreSQL (prod)
Frontend	Bootstrap 4, Chart.js
Forms	django-crispy-forms
Export	openpyxl, custom LaTeX
Server	Gunicorn, Nginx
Container	Docker

Table 1: Technology stack

1.2 Application Structure

The application follows Django's standard project structure with two main apps:

- **core**: Main application with models, views, and templates
- **reports**: Reporting, export, and portfolio optimisation

2 Project Structure

```
ref-manager/
|-- config/ # Django project settings
|-- core/ # Main application
| |-- models.py # Database models
| |-- views.py # View controllers
| |-- forms.py # Form definitions
| |-- templates/ # HTML templates
| |-- management/ # Management commands
| +-- migrations/ # Database migrations
|-- reports/ # Reporting module
|-- templates/ # Base templates
|-- static/ # Static assets
|-- documentation/ # Documentation
|-- docker-compose.yml # Docker configuration
+-- requirements.txt # Python dependencies
```

3 Database Schema

3.1 Core Models

3.1.1 Colleague Model

```
class Colleague(models.Model):
    user = models.OneToOneField(User)
    staff_id = models.CharField(max_length=50, unique=True)
    fte = models.DecimalField(max_digits=3, decimal_places=2)
    contract_type = models.CharField(max_length=50)
    employment_status = models.CharField(max_length=10)
    colleague_category = models.CharField(max_length=50)
```

```
    unit_of_assessment = models.CharField(max_length=100)
    is_returnable = models.BooleanField(default=True)
```

3.1.2 Output Model

```
class Output(models.Model):
    colleague = models.ForeignKey(Colleague)
    title = models.CharField(max_length=500)
    publication_type = models.CharField(max_length=1)
    publication_year = models.IntegerField()
    status = models.CharField(max_length=20)

    # Quality ratings
    quality_rating_internal = models.CharField(max_length=2)
    quality_rating_external = models.CharField(max_length=2)
    quality_rating_self = models.CharField(max_length=2)

    # Risk fields
    content_risk_score = models.DecimalField()
    timeline_risk_score = models.DecimalField()
    overall_risk_score = models.DecimalField()
```

3.1.3 Access Control Models

```
class Role(models.Model):
    ROLE_CHOICES = [
        ('ADMIN', 'Administrator'),
        ('OBSERVER', 'Observer'),
        ('INTERNAL_PANEL', 'Internal␣Panel'),
        ('COLLEAGUE', 'Colleague'),
    ]
    code = models.CharField(max_length=20, unique=True)
    permissions = models.JSONField()

class UserProfile(models.Model):
    user = models.OneToOneField(User)
    roles = models.ManyToManyField(Role)
```

4 Configuration

4.1 Environment Variables

Variable	Description	Default
DJANGO_SECRET_KEY	Secret key	Required
DEBUG	Debug mode	True
ALLOWED_HOSTS	Allowed hosts	localhost
DB_ENGINE	Database engine	sqlite3
DB_NAME	Database name	db.sqlite3
DB_USER	Database user	–
DB_PASSWORD	Database password	–

Table 2: Environment variables

4.2 Production Settings

```
# Security settings
SECURE_SSL_REDIRECT = True
SESSION_COOKIE_SECURE = True
CSRF_COOKIE_SECURE = True
SECURE_HSTS_SECONDS = 31536000
```

5 Management Commands

5.1 setup_roles

Create default roles and user profiles.

```
# Create roles only
python manage.py setup_roles

# Create profiles for existing users
python manage.py setup_roles --create-profiles

# Make superusers administrators
python manage.py setup_roles --superusers-admin
```

5.2 assign_roles

Manage user role assignments.

```
# List all users and roles
python manage.py assign_roles --list

# Add role
python manage.py assign_roles username --add ADMIN

# Set exact roles
python manage.py assign_roles username --set ADMIN

# Remove role
python manage.py assign_roles username --remove OBSERVER
```

6 API Reference

6.1 View Mixins

```
from core.mixins import (
    AdminRequiredMixin,
    OutputAccessMixin,
    CanEditMixin,
)

class MyView(AdminRequiredMixin, ListView):
    model = Output
```

6.2 Decorators

```
from core.decorators import (
    admin_required,
    permission_required,
)

@admin_required
def admin_view(request):
    pass

@permission_required('can_export_data')
def export_view(request):
    pass
```

6.3 Template Tags

```
{% load ref_permissions %}

{% if output|can_edit:user.ref_profile %}
    <a href="...">Edit</a>
{% endif %}

{% if user.ref_profile|has_permission:'can_export_data' %}
    <a href="...">Export</a>
{% endif %}
```

7 Deployment

7.1 Docker Deployment

```
# Build and start
docker-compose up -d --build

# Run migrations
docker-compose exec web python manage.py migrate

# Create superuser
docker-compose exec web python manage.py createsuperuser
```

7.2 Traditional Deployment

```
# Install dependencies
pip install -r requirements.txt
pip install gunicorn psycopg2-binary

# Run with Gunicorn
gunicorn config.wsgi:application \
    --bind 0.0.0.0:8000 \
    --workers 3
```

8 Security

8.1 Production Checklist

- DEBUG = False

- Secure SECRET_KEY
- HTTPS enforced
- Database password set
- ALLOWED_HOSTS configured
- CSRF protection enabled
- Session cookies secure

9 Maintenance

9.1 Database Backup

```
# PostgreSQL backup
pg_dump -U refuser refmanager > backup.sql

# Restore
psql -U refuser refmanager < backup.sql
```

9.2 Log Locations

- Application: `/var/log/ref-manager/gunicorn.log`
- Nginx: `/var/log/nginx/`
- PostgreSQL: `/var/log/postgresql/`