Contents

1 REF Manager v2.0 - Technical Documentation

Developer and System Administrator Reference

Version: 2.0.0
Last Updated: November 3, 2025
For: Developers, System Administrators, Technical Staff

1.1 Table of Contents

1.2 Architecture Overview

REF Manager follows Django’s MTV (Model-Template-View) architecture pattern with a clear separation of concerns.

1.2.1 System Architecture

```
┌─────────────────────────────────────────────────┐
│              Web Browser (Client)               │
│       HTML5 | CSS3 | JavaScript (jQuery)        │
└─────────────────────────────────────────────────┘
                  │ HTTP/HTTPS
┌─────────────────▼───────────────────────────────┐
│              Web Server (Nginx)                 │
│   Static Files | SSL/TLS | Reverse Proxy        │
└─────────────────────────────────────────────────┘
                  │
┌─────────────────▼───────────────────────────────┐
│         Application Server (Gunicorn)           │
│              WSGI Interface                     │
└─────────────────────────────────────────────────┘
                  │
┌─────────────────▼───────────────────────────────┐
│         Django Application (v4.2+)              │
│  ┌───────────────────────────────────────────┐  │
│  │ URL Router → View Functions → Templates   │  │
│  │                 ↕                         │  │
│  │          Django ORM (Models)              │  │
│  └───────────────────────────────────────────┘  │
└─────────────────────────────────────────────────┘
                  │
```

```
                         │
┌────────────────────────▼─────────────────────────┐
│      Database (PostgreSQL / SQLite)              │
│         Relational Data Storage                  │
└──────────────────────────────────────────────────┘
```

1.2.2 Application Flow

```
User Request
    ↓
Nginx (port 80/443)
    ↓
Gunicorn (port 8000)
    ↓
Django URL Dispatcher (urls.py)
    ↓
View Function (views.py)
    ↓
Model Query (models.py via ORM)
    ↓
Database (PostgreSQL/SQLite)
    ↓
Template Rendering (templates/)
    ↓
HTTP Response
    ↓
User Browser
```

1.3 □ Technology Stack

1.3.1 Backend

Core Framework: - **Django 4.2+**: Web framework - **Python 3.10+**: Programming language (tested up to 3.13)

Key Libraries: - `django-crispy-forms`: Form rendering - `crispy-bootstrap4`: Bootstrap 4 integration - `openpyxl`: Excel file handling - `python-dotenv`: Environment management - `gunicorn`: WSGI HTTP server - `psycopg2-binary`: PostgreSQL adapter

Python Standard Library Used: - `datetime`, `timezone`: Date/time handling - `decimal.Decimal`: Precise calculations - `json`: JSON processing - `csv`: CSV file handling - `io.BytesIO`: In-memory file operations - `collections.defaultdict`: Data structures

1.3.2 Frontend

UI Framework: - **Bootstrap 4.6**: Responsive design - **Font Awesome 5**: Icons - **jQuery 3.6**: DOM manipulation

Styling: - Custom CSS for REF Manager branding - Bootstrap utilities - Responsive layouts

1.3.3 Database

Development: - **SQLite 3**: File-based database - Zero configuration - Perfect for development

Production: - **PostgreSQL 12+**: Robust relational database - ACID compliance - Advanced features

1.3.4 Server Infrastructure

Production Stack: - **Nginx**: Web server and reverse proxy - **Gunicorn**: Python WSGI HTTP server - **Ubuntu 20.04+**: Operating system - **systemd**: Service management

1.3.5 Development Tools

- **Git**: Version control
- **pip**: Package management
- **venv**: Virtual environments
- **Django Debug Toolbar**: Development debugging (optional)

1.4 □ Project Structure

```
ref-manager/
│
├── ref_manager/                   # Project configuration
│   ├── __init__.py
│   ├── settings.py               # Django settings
│   ├── urls.py                   # Root URL configuration
│   ├── wsgi.py                   # WSGI application
│   └── asgi.py                   # ASGI application (future)
│
├── core/                         # Main application
│   ├── migrations/             # Database migrations
│   │   ├── __init__.py
│   │   ├── 0001_initial.py
│   │   ├── 0002_employment_status.py     # v2.0
│   │   ├── 0003_colleague_categories.py   # v2.0
│   │   ├── 0004_internal_panel.py         # v2.0
│   │   └── 0005_tasks.py                  # v2.0
│   │
│   ├── templatetags/             # Custom template filters
│   │   ├── __init__.py
│   │   └── custom_filters.py    # Custom filters for templates
│   │
│   ├── __init__.py
│   ├── models.py                 # Data models (1000+ lines)
│   ├── views.py                  # View functions (2000+ lines)
│   ├── views_export.py           # Export views (v2.0, 500+ lines)
│   ├── forms.py                  # Form definitions (800+ lines)
│   ├── urls.py                   # URL patterns (150+ lines)
│   ├── admin.py                  # Admin configuration (300+ lines)
│   ├── tests.py                  # Test cases
│   └── apps.py                   # App configuration
│
├── templates/                     # HTML templates
│   ├── base.html                 # Base template with navigation
│   ├── registration/             # Authentication templates
│   │   ├── login.html
│   │   └── password_reset.html
│   │
│   └── core/                     # App-specific templates
│       ├── dashboard.html
│       │
│       ├── colleague_list.html
│       ├── colleague_detail.html
│       ├── colleague_form.html
│       ├── colleague_confirm_delete.html
│       │
│       ├── output_list.html
│       ├── output_detail.html
│       ├── output_form.html
│       ├── output_import.html           # v2.0
│       │
│       ├── criticalfriend_list.html
│       ├── criticalfriend_detail.html
│       ├── criticalfriend_form.html
│       │
│       ├── internalpanel_list.html      # v2.0
│       ├── internalpanel_detail.html    # v2.0
│       ├── internalpanel_form.html      # v2.0
│       │
│       ├── task_list.html               # v2.0
│       ├── task_detail.html             # v2.0
│       ├── task_form.html               # v2.0
│       │
│       ├── request_list.html
│       ├── request_detail.html
│       ├── request_form.html
│       ├── request_confirm_complete.html # v2.0
│       ├── request_confirm_delete.html   # v2.0
│       │
│       ├── export_assignments.html       # v2.0
│       │
│       └── report_generate.html
│
├── static/                        # Static files
│   ├── css/
│   │   └── style.css             # Custom styles
│   ├── js/
│   │   └── script.js             # Custom JavaScript
│   └── img/
│       └── logo.png              # REF Manager logo
│
├── staticfiles/                 # Collected static files (production)
│
├── media/                         # User-uploaded files
│   └── pdfs/                     # Research paper PDFs
│
├── logs/                          # Application logs
│   ├── django.log
│   ├── gunicorn-access.log
│   └── gunicorn-error.log
│
├── backups/                       # Database backups
│
├── documentation/                 # Generated documentation
│   ├── pdf/                      # PDF files
│   └── latex/                    # LaTeX sources
│
├── venv/                          # Virtual environment
│
├── manage.py                      # Django management script
├── requirements.txt               # Python dependencies
├── .env                          # Environment variables (not in git)
├── .env.example                  # Environment template
├── .gitignore                    # Git ignore rules
├── gunicorn_config.py            # Gunicorn configuration
├── README.md                     # Main documentation
└── build_docs.sh                 # Documentation build script
```

1.4.1 Key Files Explained

settings.py: Django configuration - Database settings - Installed apps - Middleware - Static/media file paths - Security settings

urls.py: URL routing - Root URL patterns - Include app URLs - Admin URLs

models.py: Data models - Colleague - Output - CriticalFriend, CriticalFriendAssignment - InternalPanelMember, InternalPanelAssignment (v2.0) - Request - Task (v2.0)

views.py: Business logic - Dashboard - CRUD operations - List/detail views - Form processing

views_export.py (v2.0): Export functionality - Excel export with links - CSV export - Assignment filtering

---

1.5 Database Models

1.5.1 Model Relationships Diagram

```
┌─────────────┐
│    User     │  (Django auth)
└──────┬──────┘
       │ one-to-one
       │
┌──────▼──────────┐       one-to-many        ┌─────────────┐
│   Colleague     │─────────────────────────►│   Output    │
│                 │                          │             │
│ - employment    │                          │ - title     │
│   status (v2.0) │                          │ - quality   │
│ - category      │                          │ - pdf_file  │
│   (v2.0)        │                          └──────┬──────┘
└────┬────────────┘                                 │
     │                                              │
     │ one-to-many                                  │ many-to-many
     │                                              │
┌────▼────────────────┐  (v2.0)                ┌────▼─────────────────┐
│ InternalPanelMember │                        │  CriticalFriend      │
│                     │                        │                      │
│ - role              │                        │ - institution        │
│ - expertise         │                        │ - expertise          │
└────┬────────────────┘                        └─────┬────────────────┘
     │ one-to-many                                   │ one-to-many
     │                                               │
┌────▼──────────────────────┐  (v2.0)  ┌─────────────▼───────────┐
│ InternalPanelAssignment   │          │CriticalFriendAssignment │
│                           │          │                         │
│ - status                  │          │ - status                │
│ - rating_recommendation   │          │ - rating                │
│ - due_date                │          │ - due_date              │
└───────────────────────────┘          └─────────────────────────┘

┌─────────────┐
│   Request   │
│             │
│ - type      │
│ - status    │
│ - priority  │
└─────────────┘
```

```
┌─────────────┐
│    Task     │ (v2.0)
│             │
│ - category  │
│ - priority  │
│ - status    │
│ - due_date  │
└─────────────┘
```

1.5.2 Core Models

1.5.2.1 Colleague Model

```
class Colleague(models.Model):
    """Staff member information"""

    user = models.OneToOneField(User, on_delete=models.CASCADE)
    title = models.CharField(max_length=50, blank=True)

    # Employment tracking (v2.0)
    employment_status = models.CharField(
        max_length=20,
        choices=[
            ('current', 'Current'),
            ('former', 'Former'),
        ],
        default='current'
    )
    employment_end_date = models.DateField(null=True, blank=True)

    # Category classification (v2.0)
    colleague_category = models.CharField(
        max_length=30,
        choices=[
            ('independent', 'Independent Researcher'),
            ('non_independent', 'Non-Independent Researcher'),
            ('postdoc', 'Post-Doctoral Researcher'),
            ('research_assistant', 'Research Assistant'),
            ('academic', 'Academic Staff'),
            ('support', 'Support Staff'),
            ('employee', 'Current Employee'),
            ('former', 'Former Employee'),
            ('coauthor', 'Co-author (External)'),
        ],
        default='employee'
    )

    fte = models.DecimalField(max_digits=3, decimal_places=2, default=1.0)
    unit_of_assessment = models.CharField(max_length=100, blank=True)
    office = models.CharField(max_length=100, blank=True)
    phone = models.CharField(max_length=20, blank=True)
    research_interests = models.TextField(blank=True)

    @property
    def required_outputs(self):
        """Calculate required outputs (FTE × 2.5)"""
        if self.employment_status == 'current':
            return float(self.fte) * 2.5  # Python 3.13 compatibility
        return 0

    def __str__(self):
        return self.user.get_full_name()
```

1.5.2.2 Output Model

```
class Output(models.Model):
    """Research output/publication"""

    title = models.CharField(max_length=500)
    all_authors = models.TextField()
    publication_venue = models.CharField(max_length=300)

    PUBLICATION_TYPES = [
        ('article', 'Journal Article'),
        ('conference', 'Conference Paper'),
        ('book', 'Book'),
        ('chapter', 'Book Chapter'),
        ('monograph', 'Monograph'),
        ('report', 'Report'),
        ('other', 'Other'),
    ]
    publication_type = models.CharField(max_length=20, choices=PUBLICATION_TYPES)

    publication_date = models.DateField()
    volume = models.CharField(max_length=50, blank=True)
    issue = models.CharField(max_length=50, blank=True)
    pages = models.CharField(max_length=50, blank=True)
    doi = models.CharField(max_length=200, blank=True)
    url = models.URLField(max_length=500, blank=True)

    pdf_file = models.FileField(upload_to='pdfs/', null=True, blank=True)

    # REF specifics
    colleague = models.ForeignKey(
        Colleague,
        on_delete=models.CASCADE,
        related_name='outputs'
    )
    unit_of_assessment = models.CharField(max_length=100)

    QUALITY_CHOICES = [
        (4, '4* World-Leading'),
        (3, '3* Internationally Excellent'),
        (2, '2* Recognized Internationally'),
        (1, '1* Recognized Nationally'),
        (0, 'Unclassified'),
    ]
    quality_rating = models.IntegerField(choices=QUALITY_CHOICES, default=0)

    ref_eligible = models.BooleanField(default=True)

    STATUS_CHOICES = [
        ('draft', 'Draft'),
        ('under_review', 'Under Review'),
        ('accepted', 'Accepted'),
        ('published', 'Published'),
    ]
    status = models.CharField(max_length=20, choices=STATUS_CHOICES, default='draf

    keywords = models.CharField(max_length=500, blank=True)
    abstract = models.TextField(blank=True)
    notes = models.TextField(blank=True)

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

    def __str__(self):
        return self.title
```

1.5.2.3 InternalPanelMember Model (v2.0)

```
class InternalPanelMember(models.Model):
    """Internal panel member for evaluation"""

    colleague = models.ForeignKey(
        Colleague,
        on_delete=models.CASCADE,
        related_name='panel_memberships'
    )

    ROLE_CHOICES = [
        ('chair', 'Chair'),
        ('member', 'Member'),
        ('specialist', 'Specialist'),
        ('external_liaison', 'External Liaison'),
    ]
    role = models.CharField(max_length=20, choices=ROLE_CHOICES, default='member')

    expertise_area = models.CharField(max_length=200, blank=True)
    available = models.BooleanField(default=True)
    max_assignments = models.IntegerField(default=10)
    notes = models.TextField(blank=True)
    created_at = models.DateTimeField(auto_now_add=True)

    def __str__(self):
     return f"{self.colleague.user.get_full_name()} ({self.get_role_display()})"
```

1.5.2.4 Task Model (v2.0)

```
class Task(models.Model):
    """General task tracking"""

    title = models.CharField(max_length=200)
    description = models.TextField()

    CATEGORY_CHOICES = [
        ('administrative', 'Administrative'),
        ('submission', 'Submission'),
        ('review', 'Review'),
        ('meeting', 'Meeting'),
        ('documentation', 'Documentation'),
        ('deadline', 'Deadline'),
        ('other', 'Other'),
    ]
   category = models.CharField(max_length=20, choices=CATEGORY_CHOICES)

    PRIORITY_CHOICES = [
        ('low', 'Low'),
        ('medium', 'Medium'),
        ('high', 'High'),
        ('urgent', 'Urgent'),
    ]
   priority = models.CharField(max_length=10, choices=PRIORITY_CHOICES, default='m

    STATUS_CHOICES = [
        ('pending', 'Pending'),
        ('in_progress', 'In Progress'),
        ('completed', 'Completed'),
        ('cancelled', 'Cancelled'),
    ]
   status = models.CharField(max_length=15, choices=STATUS_CHOICES, default='pendi

    assigned_to = models.ForeignKey(
        User,
        on_delete=models.SET_NULL,
        null=True,
        blank=True,
        related_name='assigned_tasks'
    )
    created_by = models.ForeignKey(
        User,
        on_delete=models.CASCADE,
        related_name='created_tasks'
    )

    start_date = models.DateField(null=True, blank=True)
    due_date = models.DateField()
    completed_at = models.DateTimeField(null=True, blank=True)

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

    @property
    def is_overdue(self):
        """Check if task is overdue"""
        if self.status in ['completed', 'cancelled']:
            return False
        return timezone.now().date() > self.due_date

    def __str__(self):
        return self.title
```

1.5.3 Database Queries Optimization

Best Practices:

```
# Use select_related() for foreign keys (one-to-one, many-to-one)
colleagues = Colleague.objects.select_related('user').all()

# Use prefetch_related() for reverse foreign keys and many-to-many
colleagues = Colleague.objects.prefetch_related('outputs').all()

# Combine for complex queries
outputs = Output.objects.select_related(
    'colleague',
    'colleague__user'
).prefetch_related(
    'internal_assignments',
    'critical_friend_assignments'
).all()

# Annotate for counts
from django.db.models import Count
colleagues = Colleague.objects.annotate(
    output_count=Count('outputs')
).all()

# Filter efficiently
current_colleagues = Colleague.objects.filter(
    employment_status='current'
).select_related('user')
```

1.6 □ URL Configuration

1.6.1 Root URLs (ref_manager/urls.py)

```
from django.contrib import admin
from django.urls import path, include
from django.conf import settings
from django.conf.urls.static import static

urlpatterns = [
    path('admin/', admin.site.urls),
    path('', include('core.urls')),
    path('accounts/', include('django.contrib.auth.urls')),
]

# Serve media files in development
if settings.DEBUG:
    urlpatterns += static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)
```

1.6.2 App URLs (core/urls.py)

```
from django.urls import path
from . import views
from . import views_export  # v2.0

app_name = 'core'

urlpatterns = [
    # Dashboard
    path('', views.dashboard, name='dashboard'),

    # Colleagues
    path('colleagues/', views.colleague_list, name='colleague_list'),
    path('colleagues/<int:pk>/', views.colleague_detail, name='colleague_detail'),
    path('colleagues/add/', views.colleague_create, name='colleague_create'),
    path('colleagues/<int:pk>/edit/', views.colleague_update, name='colleague_upda
    path('colleagues/<int:pk>/delete/', views.colleague_delete, name='colleague_de

    # Outputs
    path('outputs/', views.output_list, name='output_list'),
    path('outputs/<int:pk>/', views.output_detail, name='output_detail'),
    path('outputs/add/', views.output_create, name='output_create'),
    path('outputs/<int:pk>/edit/', views.output_update, name='output_update'),
    path('outputs/<int:pk>/delete/', views.output_delete, name='output_delete'),
    path('outputs/import/', views.output_import, name='output_import'), # v2.0

    # Internal Panel (v2.0)
    path('internal-panel/', views.internal_panel_list, name='internal_panel_list')
    path('internal-panel/<int:pk>/', views.internal_panel_detail, name='internal_p
    path('internal-panel/add/', views.internal_panel_create, name='internal_panel_
   path('internal-panel/<int:pk>/edit/', views.internal_panel_update, name='inter
   path('internal-panel/<int:pk>/delete/', views.internal_panel_delete, name='int
   path('internal-panel/<int:pk>/assign/<int:output_id>/', views.internal_panel_a

    # Tasks (v2.0)
    path('tasks/', views.task_list, name='task_list'),
    path('tasks/<int:pk>/', views.task_detail, name='task_detail'),
    path('tasks/create/', views.task_create, name='task_create'),
   path('tasks/<int:pk>/edit/', views.task_update, name='task_update'),
   path('tasks/<int:pk>/delete/', views.task_delete, name='task_delete'),
   path('tasks/<int:pk>/complete/', views.task_complete, name='task_complete'),

    # Requests
    path('requests/', views.request_list, name='request_list'),
   path('requests/<int:pk>/', views.request_detail, name='request_detail'),
   path('requests/create/', views.request_create, name='request_create'),
   path('requests/<int:pk>/edit/', views.request_update, name='request_update'),
   path('requests/<int:pk>/complete/', views.request_complete, name='request_comp
   path('requests/<int:pk>/delete/', views.request_delete, name='request_delete')

    # Export (v2.0)
   path('export/assignments/', views_export.export_assignments_view, name='export
   path('export/assignments/excel/', views_export.export_assignments_excel, name=
   path('export/assignments/csv/', views_export.export_assignments_csv, name='exp

    # Reports
    path('reports/', views.report_generate, name='report_generate'),
]
```

1.6.3 URL Pattern Naming Convention

- List views: `{model}_list`
- Detail views: `{model}_detail`
- Create views: `{model}_create`
- Update views: `{model}_update`
- Delete views: `{model}_delete`
- Custom actions: `{model}_{action}`

1.7 Views and Business Logic

1.7.1 View Structure

REF Manager uses function-based views (FBVs) for clarity and simplicity.

1.7.2 Common View Patterns

1.7.2.1 List View Pattern

```
@login_required
def model_list(request):
    """List all instances with filtering"""

    # Get query parameters
    filter_param = request.GET.get('filter', 'all')
    search_query = request.GET.get('q', '')

    # Base queryset with optimization
    queryset = Model.objects.select_related('foreign_key').all()

    # Apply filters
    if filter_param != 'all':
        queryset = queryset.filter(field=filter_param)

    # Apply search
    if search_query:
        queryset = queryset.filter(
            Q(field1__icontains=search_query) |
            Q(field2__icontains=search_query)
        )

    # Pagination
    paginator = Paginator(queryset, 25)
    page_number = request.GET.get('page')
    page_obj = paginator.get_page(page_number)

    context = {
        'page_obj': page_obj,
        'filter_param': filter_param,
        'search_query': search_query,
    }

    return render(request, 'app/model_list.html', context)
```

1.7.2.2 Create/Update View Pattern

```
@login_required
def model_create(request):
    """Create new instance"""

    if request.method == 'POST':
        form = ModelForm(request.POST, request.FILES)
        if form.is_valid():
            instance = form.save(commit=False)
            instance.created_by = request.user
            instance.save()
            messages.success(request, 'Created successfully!')
            return redirect('app:model_detail', pk=instance.pk)
    else:
        form = ModelForm()
    return render(request, 'app/model_form.html', {'form': form})


@login_required
def model_update(request, pk):
    """Update existing instance"""

    instance = get_object_or_404(Model, pk=pk)

    if request.method == 'POST':
      form = ModelForm(request.POST, request.FILES, instance=instance)
        if form.is_valid():
            form.save()
            messages.success(request, 'Updated successfully!')
            return redirect('app:model_detail', pk=instance.pk)
    else:
        form = ModelForm(instance=instance)

    return render(request, 'app/model_form.html', {
        'form': form,
        'instance': instance
    })
```

1.7.3 Export Views (v2.0)

1.7.3.1 Excel Export with Links

```
from openpyxl import Workbook
from openpyxl.styles import Font, PatternFill, Alignment, Border, Side
from django.http import HttpResponse
from io.BytesIO import BytesIO

@login_required
def export_assignments_excel(request):
    """Export assignments to Excel with clickable PDF links"""

    # Get filter parameters
    reviewer_type = request.GET.get('reviewer_type', 'all')

    # Create workbook
    wb = Workbook()
    ws = wb.active
    ws.title = "Review Assignments"

    # Headers
    headers = [
        'Reviewer Name', 'Email', 'Output Title',
        'Author', 'Quality', 'Status', 'Due Date',
        'Priority', 'PDF Link', 'Notes'
    ]
    # Style headers
   header_fill = PatternFill(start_color="4472C4", end_color="4472C4", fill_type='
    header_font = Font(bold=True, color="FFFFFF")

    for col_num, header in enumerate(headers, 1):
        cell = ws.cell(row=1, column=col_num)
        cell.value = header
        cell.fill = header_fill
        cell.font = header_font

    # Query data
    assignments = get_filtered_assignments(request)

    # Add data rows
    for row_num, assignment in enumerate(assignments, 2):
        # Color coding based on type
        if assignment.is_internal:
        fill = PatternFill(start_color="D9E1F2", fill_type="solid") # Light blue
        else:
        fill = PatternFill(start_color="FFF2CC", fill_type="solid") # Light yello

        # Add data
        data = [
            assignment.reviewer_name,
            assignment.reviewer_email,
            assignment.output.title,
            assignment.output.colleague.user.get_full_name(),
            assignment.output.get_quality_rating_display(),
            assignment.get_status_display(),
        assignment.due_date.strftime('%Y-%m-%d') if assignment.due_date else '',
            assignment.get_priority_display(),
            'View PDF',
            assignment.notes
        ]

        for col_num, value in enumerate(data, 1):
            cell = ws.cell(row=row_num, column=col_num)
            cell.value = value
            cell.fill = fill

            # Make PDF link clickable
            if col_num == 9 and assignment.output.pdf_file:
           full_url = request.build_absolute_uri(assignment.output.pdf_file.url)
                cell.hyperlink = full_url
                cell.font = Font(color="0563C1", underline="single")

    # Adjust column widths
    column_widths = {'A': 20, 'B': 25, 'C': 40, 'D': 20, 'E': 15,
                     'F': 15, 'G': 12, 'H': 12, 'I': 15, 'J': 30}
    for col, width in column_widths.items():
        ws.column_dimensions[col].width = width

    # Freeze header
    ws.freeze_panes = 'A2'

    # Create response
    output_buffer = BytesIO()
    wb.save(output_buffer)
    output_buffer.seek(0)

    # Generate filename
    timestamp = datetime.now().strftime('%Y%m%d_%H%M%S')
    filename = f'review_assignments_{timestamp}.xlsx'

    response = HttpResponse(
        output_buffer.read(),
     content_type='application/vnd.openxmlformats-officedocument.spreadsheetml.s
    )
   response['Content-Disposition'] = f'attachment; filename="{filename}"'

    return response
```

1.8 □ Forms and Validation

1.8.1 Form Classes

Forms use `django-crispy-forms` for Bootstrap styling.

1.8.1.1 Model Form Example

```
from django import forms
from crispy_forms.helper import FormHelper
from crispy_forms.layout import Layout, Submit, Div, Field
from .models import Colleague

class ColleagueForm(forms.ModelForm):
    """Form for creating/updating colleagues"""

    class Meta:
        model = Colleague
        fields = [
            'title', 'fte', 'unit_of_assessment',
            'employment_status', 'employment_end_date',  # v2.0
            'colleague_category',  # v2.0
            'office', 'phone', 'research_interests'
        ]
        widgets = {
        'employment_end_date': forms.DateInput(attrs={'type': 'date'}),
            'research_interests': forms.Textarea(attrs={'rows': 4}),
        }

    def __init__(self, *args, **kwargs):
        super().__init__(*args, **kwargs)

        # Crispy forms helper
        self.helper = FormHelper()
        self.helper.form_method = 'post'
        self.helper.add_input(Submit('submit', 'Save'))

        # Custom field attributes
        self.fields['fte'].widget.attrs['step'] = '0.1'
        self.fields['fte'].widget.attrs['min'] = '0.1'
        self.fields['fte'].widget.attrs['max'] = '1.0'

    def clean_employment_end_date(self):
        """Validate employment end date"""
        status = self.cleaned_data.get('employment_status')
        end_date = self.cleaned_data.get('employment_end_date')

        if status == 'former' and not end_date:
            raise forms.ValidationError(
                "Employment end date required for former staff"
            )

        if status == 'current' and end_date:
            raise forms.ValidationError(
                "Current staff should not have an end date"
            )

        return end_date
```

1.8.2 Custom Validation

```
def clean(self):
    """Cross-field validation"""
    cleaned_data = super().clean()

    start_date = cleaned_data.get('start_date')
    end_date = cleaned_data.get('end_date')

    if start_date and end_date:
        if end_date < start_date:
            raise forms.ValidationError(
                "End date must be after start date"
            )

    return cleaned_data
```

1.9 Templates and Frontend

1.9.1 Template Inheritance

```
base.html                      # Base template
├── registration/
│   ├── login.html            # Auth templates
│   └── password_reset.html
└── core/
    ├── dashboard.html         # Extends base
    ├── colleague_list.html    # Extends base
    └── ...
```

1.9.2 Base Template (base.html)

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
       <meta  name="viewport"  content="width=device-width,  initial-
scale=1.0">
    <title>{% block title %}REF Manager{% endblock %}</title>

    <!-- Bootstrap CSS -->
   <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap@4.6.0/dist

    <!-- Font Awesome -->
   <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-
awesome/5.15.4/css/all.min.css">

    <!-- Custom CSS -->
    {% load static %}
    <link rel="stylesheet" href="{% static 'css/style.css' %}">

    {% block extra_css %}{% endblock %}
</head>
<body>
    <!-- Navigation -->
    <nav class="navbar navbar-expand-lg navbar-dark bg-primary">
        <a class="navbar-brand" href="{% url 'core:dashboard' %}">
            <i class="fas fa-graduation-cap"></i> REF Manager
        </a>

                <button  class="navbar-toggler"  type="button"  data-
toggle="collapse" data-target="#navbarNav">
            <span class="navbar-toggler-icon"></span>
        </button>

        <div class="collapse navbar-collapse" id="navbarNav">
            <ul class="navbar-nav mr-auto">
```

```
                <li class="nav-item">
              <a class="nav-link" href="{% url 'core:colleague_list' %}">
                        <i class="fas fa-users"></i> Colleagues
                    </a>
                </li>
                <li class="nav-item">
               <a class="nav-link" href="{% url 'core:output_list' %}">
                        <i class="fas fa-book"></i> Outputs
                    </a>
                </li>
                <!-- v2.0 additions -->
                <li class="nav-item">
              <a class="nav-link" href="{% url 'core:internal_panel_list' %}">
                     <i class="fas fa-user-shield"></i> Internal Panel
                    </a>
                </li>
                <li class="nav-item">
                <a class="nav-link" href="{% url 'core:task_list' %}">
                        <i class="fas fa-tasks"></i> Tasks
                    </a>
                </li>
                <li class="nav-item">
              <a class="nav-link" href="{% url 'core:export_assignments' %}">
                        <i class="fas fa-download"></i> Export
                    </a>
                </li>
            </ul>

            <ul class="navbar-nav">
                {% if user.is_authenticated %}
                <li class="nav-item dropdown">
              <a class="nav-link dropdown-toggle" href="#" id="userDropdown" data-
toggle="dropdown">
                       <i class="fas fa-user"></i> {{ user.username }}
                    </a>
                    <div class="dropdown-menu dropdown-menu-right">
                <a class="dropdown-item" href="{% url 'admin:index' %}">
                            <i class="fas fa-cog"></i> Admin
                        </a>
                        <div class="dropdown-divider"></div>
                   <a class="dropdown-item" href="{% url 'logout' %}">
                           <i class="fas fa-sign-out-alt"></i> Logout
                        </a>
                    </div>
                </li>
                {% else %}
                <li class="nav-item">
                    <a class="nav-link" href="{% url 'login' %}">
                        <i class="fas fa-sign-in-alt"></i> Login
                    </a>
                </li>
                {% endif %}
            </ul>
        </div>
    </nav>

    <!-- Messages -->
    {% if messages %}
    <div class="container mt-3">
        {% for message in messages %}
               <div  class="alert  alert-{{  message.tags  }}  alert-
dismissible fade show" role="alert">
            {{ message }}
            <button type="button" class="close" data-dismiss="alert">
                <span>&times;</span>
            </button>
        </div>
        {% endfor %}
    </div>
    {% endif %}

    <!-- Content -->
    <div class="container mt-4">
        {% block content %}{% endblock %}
    </div>

    <!-- Footer -->
    <footer class="mt-5 py-4 bg-light">
        <div class="container text-center">
            <p class="text-muted mb-0">
                REF Manager v2.0 &copy; 2025 University of York
            </p>
        </div>
    </footer>

    <!-- jQuery -->
   <script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>

    <!-- Bootstrap JS -->
   <script src="https://cdn.jsdelivr.net/npm/bootstrap@4.6.0/dist/js/bootstrap.bu

    <!-- Custom JS -->
    <script src="{% static 'js/script.js' %}"></script>

    {% block extra_js %}{% endblock %}
</body>
</html>
```

1.9.3 Template Tags and Filters

1.9.3.1 Custom Template Filters (core/templatetags/custom_filters.py)

```
from django import template

register = template.Library()

@register.filter
def get_item(dictionary, key):
    """Get item from dictionary"""
    return dictionary.get(key)

@register.filter
def multiply(value, arg):
    """Multiply value by arg"""
    try:
        return float(value) * float(arg)
    except (ValueError, TypeError):
        return ''

@register.filter
def badge_class(status):
    """Return Bootstrap badge class for status"""
    badges = {
        'pending': 'badge-warning',
        'in_progress': 'badge-info',
        'completed': 'badge-success',
        'cancelled': 'badge-secondary',
    }
    return badges.get(status, 'badge-secondary')
```

Usage in templates:

```
{% load custom_filters %}

<span class="badge {{ task.status|badge_class }}">
    {{ task.get_status_display }}
</span>
```

1.10 □ Static Files and Media

1.10.1 Static Files Structure

```
static/
├── css/
│   └── style.css              # Custom styles
├── js/
│   └── script.js              # Custom JavaScript
└── img/
    └── logo.png               # Application logo
```

1.10.2 Media Files

```
media/
└── pdfs/                       # Uploaded research papers
    ├── output_1_paper.pdf
    ├── output_2_paper.pdf
    └── ...
```

1.10.3 Static Files Configuration

```
# settings.py

STATIC_URL = '/static/'
STATIC_ROOT = os.path.join(BASE_DIR, 'staticfiles')
STATICFILES_DIRS = [
    os.path.join(BASE_DIR, 'static'),
]

MEDIA_URL = '/media/'
MEDIA_ROOT = os.path.join(BASE_DIR, 'media')
```

1.10.4 Collecting Static Files

```
# Development
python manage.py collectstatic --noinput

# Production (in deployment script)
python manage.py collectstatic --noinput --clear
```

1.11 □ Security Considerations

1.11.1 Authentication and Authorization

```
# Use login_required decorator
from django.contrib.auth.decorators import login_required

@login_required
def protected_view(request):
    # Only authenticated users can access
    pass

# Permission checks
from django.contrib.auth.decorators import permission_required

@permission_required('core.add_output')
def create_output(request):
    pass
```

1.11.2 CSRF Protection

Django’s CSRF protection is enabled by default:

```
<!-- In forms -->
<form method="post">
    {% csrf_token %}
    <!-- form fields -->
</form>
```

1.11.3 SQL Injection Prevention

Django ORM automatically escapes queries:

```
# Safe - parameterized
Colleague.objects.filter(user__username=username)

# Avoid raw SQL unless necessary
# If using raw SQL, always use parameters
Colleague.objects.raw('SELECT * FROM core_colleague WHERE id = %s', [id])
```

1.11.4 XSS Prevention

Django templates auto-escape by default:

```
<!-- Automatically escaped -->
{{ user_input }}

<!-- If you need raw HTML (be careful!) -->
{{ trusted_html|safe }}
```

1.11.5 File Upload Security

```
# Validate file types
def validate_pdf(file):
    if not file.name.endswith('.pdf'):
        raise ValidationError('Only PDF files allowed')

    # Check file size (20MB limit)
    if file.size > 20 * 1024 * 1024:
        raise ValidationError('File too large (max 20MB)')

    return file

# In models
pdf_file = models.FileField(
    upload_to='pdfs/',
    validators=[validate_pdf],
    null=True, blank=True
)
```

1.11.6 Production Security Settings

```
# settings.py (production)

DEBUG = False
ALLOWED_HOSTS = ['your-domain.com', 'www.your-domain.com']

# HTTPS/Security
SECURE_SSL_REDIRECT = True
SESSION_COOKIE_SECURE = True
CSRF_COOKIE_SECURE = True
SECURE_BROWSER_XSS_FILTER = True
SECURE_CONTENT_TYPE_NOSNIFF = True
X_FRAME_OPTIONS = 'DENY'

# HSTS
SECURE_HSTS_SECONDS = 31536000
SECURE_HSTS_INCLUDE_SUBDOMAINS = True
SECURE_HSTS_PRELOAD = True
```

1.12 □ Testing

1.12.1 Test Structure

```
# core/tests.py

from django.test import TestCase, Client
from django.contrib.auth.models import User
from .models import Colleague, Output

class ColleagueModelTest(TestCase):
    """Test Colleague model"""

    def setUp(self):
        """Set up test data"""
        self.user = User.objects.create_user(
            username='testuser',
            email='test@example.com',
            password='testpass123'
        )
        self.colleague = Colleague.objects.create(
            user=self.user,
            fte=1.0,
            employment_status='current',
            colleague_category='independent'
        )

    def test_colleague_creation(self):
        """Test colleague is created correctly"""
        self.assertEqual(self.colleague.user.username, 'testuser')
        self.assertEqual(self.colleague.fte, 1.0)

    def test_required_outputs(self):
        """Test required outputs calculation"""
        self.assertEqual(self.colleague.required_outputs, 2.5)

    def test_string_representation(self):
        """Test __str__ method"""
        self.assertEqual(str(self.colleague), self.user.get_full_name())


class OutputModelTest(TestCase):
    """Test Output model"""

    def setUp(self):
        user = User.objects.create_user('testuser', 'test@example.com', 'pass')
        self.colleague = Colleague.objects.create(user=user, fte=1.0)
        self.output = Output.objects.create(
            title='Test Output',
            all_authors='Author 1, Author 2',
            publication_venue='Test Journal',
            publication_type='article',
            publication_date='2025-01-01',
            colleague=self.colleague,
            unit_of_assessment='UoA 27',
            quality_rating=4
        )

    def test_output_creation(self):
        """Test output is created"""
        self.assertEqual(self.output.title, 'Test Output')
        self.assertEqual(self.output.quality_rating, 4)


class DashboardViewTest(TestCase):
    """Test dashboard view"""

    def setUp(self):
        self.client = Client()
        self.user = User.objects.create_user('testuser', 'test@example.com', 'pass')
        self.client.login(username='testuser', password='pass')

    def test_dashboard_loads(self):
        """Test dashboard page loads"""
        response = self.client.get('/')
        self.assertEqual(response.status_code, 200)
        self.assertTemplateUsed(response, 'core/dashboard.html')

    def test_dashboard_requires_login(self):
        """Test dashboard requires authentication"""
        self.client.logout()
        response = self.client.get('/')
        self.assertEqual(response.status_code, 302)  # Redirect to login
```

1.12.2 Running Tests

```
# Run all tests
python manage.py test

# Run specific app tests
python manage.py test core

# Run specific test class
python manage.py test core.tests.ColleagueModelTest

# Run with verbosity
python manage.py test --verbosity=2

# Keep test database
python manage.py test --keepdb

# Run with coverage
coverage run --source='.' manage.py test
coverage report
coverage html
```

1.13 ⚡ Performance Optimization

1.13.1 Database Optimization

1. Use select_related() and prefetch_related()

```
# Without optimization - N+1 queries
colleagues = Colleague.objects.all()
for colleague in colleagues:
    print(colleague.user.username)  # Extra query each time

# With optimization - 2 queries total
colleagues = Colleague.objects.select_related('user').all()
for colleague in colleagues:
    print(colleague.user.username)  # No extra queries
```

2. Use annotate() for aggregations

```
from django.db.models import Count

# Count in Python - many queries
colleagues = Colleague.objects.all()
for colleague in colleagues:
    output_count = colleague.outputs.count()  # Query per colleague

# Count in database - single query
colleagues = Colleague.objects.annotate(
    output_count=Count('outputs')
).all()
```

3. Use only() and defer()

```
# Load only needed fields
colleagues = Colleague.objects.only('id', 'user__username', 'fte')

# Defer large fields
colleagues = Colleague.objects.defer('research_interests')
```

4. Database indexing

```
# In models.py
class Output(models.Model):
    title = models.CharField(max_length=500, db_index=True)
    quality_rating = models.IntegerField(db_index=True)

    class Meta:
        indexes = [
         models.Index(fields=['publication_date', 'quality_rating']),
            models.Index(fields=['colleague', 'quality_rating']),
        ]
```

1.13.2 Caching

```
from django.core.cache import cache
from django.views.decorators.cache import cache_page

# Cache view for 15 minutes
@cache_page(60 * 15)
def report_view(request):
    # Expensive computation
    return render(request, 'report.html', context)

# Manual caching
def get_quality_statistics():
    stats = cache.get('quality_stats')
    if stats is None:
        stats = compute_expensive_statistics()
        cache.set('quality_stats', stats, 60 * 60)  # 1 hour
    return stats
```

1.13.3 Query Optimization Tips

```
# Bad - loads all into memory
all_outputs = list(Output.objects.all())

# Good - use iterator for large datasets
for output in Output.objects.iterator(chunk_size=100):
    process_output(output)

# Bad - multiple queries
for colleague in Colleague.objects.all():
    if colleague.outputs.filter(quality_rating=4).exists():
        # process

# Good - single query with annotation
colleagues_with_4star = Colleague.objects.annotate(
    has_4star=Count('outputs', filter=Q(outputs__quality_rating=4))
).filter(has_4star__gt=0)
```

1.14 □ Deployment Guide

See main README.md for complete deployment instructions.

Quick reference:

```
# 1. Set up environment
python3 -m venv venv
source venv/bin/activate
pip install -r requirements.txt

# 2. Configure settings
cp .env.example .env
# Edit .env with production values

# 3. Database
python manage.py migrate
python manage.py createsuperuser

# 4. Static files
python manage.py collectstatic --noinput

# 5. Run with Gunicorn
gunicorn ref_manager.wsgi:application --bind 0.0.0.0:8000

# 6. Set up Nginx reverse proxy
# (see production deployment section in README)
```

1.15 □□ Development Workflow

1.15.1 Setting Up Development Environment

```
# Clone repository
git clone https://github.com/yourusername/ref-manager.git
cd ref-manager

# Create virtual environment
python3 -m venv venv
source venv/bin/activate

# Install dependencies
pip install -r requirements.txt

# Install development dependencies
pip install django-debug-toolbar coverage black flake8

# Set up database
python manage.py migrate

# Create superuser
python manage.py createsuperuser

# Run development server
python manage.py runserver
```

1.15.2 Code Style

Python: - PEP 8 compliance - Use Black for formatting: `black .` - Use flake8 for linting: `flake8`

JavaScript: - ES6+ syntax - Semicolons required - 2-space indentation

HTML: - Django template syntax - 4-space indentation - Semantic HTML5

1.15.3 Git Workflow

```
# Create feature branch
git checkout -b feature/new-feature

# Make changes and commit
git add .
git commit -m "Add new feature"

# Push to remote
git push origin feature/new-feature

# Create pull request on GitHub
# After review and approval, merge to main
```

1.15.4 Adding New Features

1. Create model (if needed)

```
# models.py
class NewModel(models.Model):
    # fields
    pass
```

2. Create migration

```
python manage.py makemigrations
python manage.py migrate
```

3. Create forms

```
# forms.py
class NewModelForm(forms.ModelForm):
    class Meta:
        model = NewModel
        fields = '__all__'
```

4. Create views

```
# views.py
@login_required
def new_model_list(request):
    # implementation
    pass
```

5. Add URLs

```
# urls.py
path('newmodel/', views.new_model_list, name='new_model_list'),
```

6. Create templates

```
<!-- templates/core/newmodel_list.html -->
{% extends 'base.html' %}
{% block content %}
<!-- content -->
{% endblock %}
```

7. Update navigation

```
<!-- base.html -->
<li class="nav-item">
    <a class="nav-link" href="{% url 'core:new_model_list' %}">
        New Feature
    </a>
</li>
```

8. Write tests

```
# tests.py
class NewModelTest(TestCase):
    def test_creation(self):
        # test code
        pass
```

9. Run tests

```
python manage.py test
```

10. Update documentation

1.16 □ Additional Resources

Django Documentation: - Official Docs: https://docs.djangoproject.com/ - Tutorial: https://docs.djangoproject.com/en/4.2/intro/tutorial01/

REF Information: - REF 2029: https://www.ref.ac.uk/

Python Resources: - PEP 8: https://pep8.org/ - Python Docs: https://docs.python.org/3/

Frontend: - Bootstrap 4: https://getbootstrap.com/docs/4.6/ - Font Awesome: https://fontawesome.com/

Version: 2.0.0
Last Updated: November 3, 2025
Maintained by: George Tsoulas
Institution: Department of Language and Linguistic Science, University of York

For more information, see the complete documentation suite.