التعديلات المقترحة لنظام إدارة أصول تكنولوجيا المعلومات لدعم الشركات المتعددة

تحليل الدعم الحالي للشركات المتعددة

يحتوي نظام إدارة أصول تكنولوجيا المعلومات الحالي على بعض الدعم الأساسي للشركات المتعددة من خلال نموذج `OwnerCompany`، الذي يسمح بربط الأصول بشركات مختلفة. كما يتم ربط الموظفين بالشركات من خلال علاقة المفتاح الأجنبي. ومع ذلك، فإن هذا التنفيذ يعاني من عدة قيود تمنعه من أن يكون قابلاً للتكيف بشكل كامل عبر الهياكل التنظيمية المتنوعة:

1. يفترض النظام بنية مستأجر واحد حيث تتشارك جميع الشركات نفس قاعدة البيانات والواجهة.
2. لا يوجد فصل لرؤية البيانات بين الشركات - يمكن للمستخدمين رؤية الأصول والموظفين من جميع الشركات.
3. خيارات الأقسام مبرمجة بشكل ثابت ومشتركة بين جميع الشركات.
4. نظام المصادقة لا يأخذ في الاعتبار الأدوار والصلاحيات الخاصة بكل شركة.
5. التقارير ولوحات المعلومات تجمع البيانات من جميع الشركات دون تصفية خاصة بالشركة بشكل افتراضي.

التعديلات الموصى بها

1. تعزيز بنية تعدد المستأجرين

التنفيذ الحالي: يستخدم النظام قاعدة بيانات واحدة مع نموذج `OwnerCompany` للتمييز بين الشركات، لكنه يفتقر إلى العزل المناسب للبيانات.

التعديلات الموصى بها: - تنفيذ بنية قوية متعددة المستأجرين باستخدام أحد هذه الأساليب: - تعدد المستأجرين القائم على المخطط: إنشاء مخططات قاعدة بيانات منفصلة لكل شركة مع الحفاظ على مثيل قاعدة بيانات واحد - أمان على مستوى الصف: تعزيز جميع النماذج بمعرف الشركة وتنفيذ التصفية على مستوى استعلام قاعدة البيانات - إضافة طبقة وسيطة تقوم تلقائيًا بتصفية جميع الاستعلامات بناءً على شركة المستخدم

تفاصيل التنفيذ:

```
# إضافة نموذج أساسي ترث منه جميع النماذج الأخرى
class CompanyRelatedModel(models.Model):
    company = models.ForeignKey(OwnerCompany, on_delete=models.PROTECT, related_name='+')

    class Meta:
```

```
        abstract = True

# تعديل النماذج الحالية لترث من هذا النموذج الأساسي
class Department(CompanyRelatedModel):
    # الحقول الحالية...

class Employee(CompanyRelatedModel):
    # إزالة حقل الشركة المباشر لأنه موروث الآن
    # الحقول الأخرى الحالية...

class ITAsset(CompanyRelatedModel):
    # إعادة تسمية حقل 'owner' لتجنب الالتباس مع حقل 'company' الموروث
    asset_owner = models.ForeignKey(OwnerCompany, on_delete=models.PROTECT,
related_name='owned_assets')
    # الحقول الأخرى الحالية...

# إضافة middleware للتصفية التلقائية حسب الشركة
class CompanyMiddleware:
    def __init__(self, get_response):
        self.get_response = get_response

    def __call__(self, request):
        if request.user.is_authenticated and hasattr(request.user, 'employee_profile'):
            request.company = request.user.employee_profile.company
        else:
            request.company = None

        response = self.get_response(request)
        return response
```

2. إعدادات خاصة بالشركة

التنفيذ الحالي: خيارات الأقسام مبرمجة بشكل ثابت في نموذج `Department`، مما يحد من المرونة للشركات ذات الهياكل التنظيمية المختلفة.

التعديلات الموصى بها: - إنشاء نموذج `CompanyConfiguration` لتخزين إعدادات خاصة بالشركة - السماح بتخصيص أسماء الأقسام وأنواع الأصول وخيارات الحالة لكل شركة

تفاصيل التنفيذ:

```
class CompanyConfiguration(models.Model):
    company = models.OneToOneField(OwnerCompany,
on_delete=models.CASCADE, related_name='configuration')
    custom_department_structure = models.BooleanField(default=False)
    custom_asset_types = models.BooleanField(default=False)
    custom_statuses = models.BooleanField(default=False)
```

```
    logo = models.ImageField(upload_to='company_logos/', null=True, blank=True)
    primary_color = models.CharField(max_length=7, default='#007bff')  # لون
Bootstrap الأساسي الأزرق
    secondary_color = models.CharField(max_length=7, default='#6c757d')  # لون
Bootstrap الثانوي الرمادي

    # إضافة إعدادات خاصة بالشركة مثل تنسيقات التاريخ والعملة وما إلى ذلك
    date_format = models.CharField(max_length=20, default='YYYY-MM-DD')
    currency_symbol = models.CharField(max_length=5, default='$')

    class Meta:
        verbose_name = 'إعدادات الشركة'
        verbose_name_plural = 'إعدادات الشركات'
```

3. تعزيز نظام المستخدمين والصلاحيات

التنفيذ الحالي: يستخدم النظام نظام المصادقة المدمج في Django ولكنه لا يأخذ في الاعتبار الأدوار والصلاحيات الخاصة بالشركة.

التعديلات الموصى بها: - تنفيذ نظام صلاحيات قائم على الأدوار مع أدوار خاصة بالشركة - إضافة مسؤولي شركات يمكنهم إدارة إعدادات شركتهم - تنفيذ ضوابط الوصول عبر الشركات لمسؤولي الشركات المتعددة

تفاصيل التنفيذ:

```
class CompanyRole(models.Model):
    ROLE_CHOICES = [
        ('admin', 'مسؤول الشركة'),
        ('manager', 'مدير القسم'),
        ('staff', 'عضو فريق'),
        ('readonly', 'مستخدم للقراءة فقط'),
    ]

    user = models.ForeignKey(User, on_delete=models.CASCADE,
related_name='company_roles')
    company = models.ForeignKey(OwnerCompany, on_delete=models.CASCADE,
related_name='user_roles')
    role = models.CharField(max_length=20, choices=ROLE_CHOICES)
    departments = models.ManyToManyField(Department, blank=True,
related_name='role_assignments')

    class Meta:
        unique_together = ('user', 'company')
        verbose_name = 'دور الشركة'
        verbose_name_plural = 'أدوار الشركة'
```

4. لوحة معلومات وتقارير خاصة بالشركة

التنفيذ الحالي: تعرض لوحة المعلومات بيانات مجمعة من جميع الشركات دون تصفية سهلة حسب الشركة.

التعديلات الموصى بها: - تعديل لوحة المعلومات لعرض البيانات الخاصة بشركة المستخدم فقط بشكل افتراضي - إضافة خيارات تصفية الشركة للمستخدمين الذين لديهم وصول إلى شركات متعددة - إنشاء تقارير وتحليلات خاصة بالشركة - تنفيذ تقارير مقارنة لمسؤولي الشركات المتعددة

تفاصيل التنفيذ:

```
@login_required
def home(request):
    # الحصول على شركة المستخدم
    company = request.company

    # تصفية جميع الاستعلامات حسب الشركة
    total_assets = ITAsset.objects.filter(company=company).count()
    total_employees = Employee.objects.filter(company=company).count()
    available_assets = ITAsset.objects.filter(company=company,
status='available').count()
    # ... استعلامات مصفاة أخرى

    # بالنسبة لمسؤولي الشركات المتعددة، أضف اختيار الشركة
    available_companies = []
    if request.user.is_superuser or CompanyRole.objects.filter(
        user=request.user, role='admin'
    ).exists():
        available_companies = OwnerCompany.objects.all()

    context = {
        # ... بيانات السياق الحالية
        'available_companies': available_companies,
        'current_company': company,
    }
    return render(request, 'inventory/home.html', context)
```

5. واجهة برمجة التطبيقات (API) للتكامل مع الأنظمة الأخرى

التنفيذ الحالي: النظام مستقل بذاته بدون واجهة برمجة تطبيقات للتكامل مع أنظمة الشركة الأخرى.

التعديلات الموصى بها: - إضافة Django REST Framework - تنفيذ واجهة برمجة تطبيقات RESTful باستخدام خاصة بالشركة - إنشاء نقاط نهاية للأصول والموظفين والأقسام - تمكين التكامل مع أنظمة API مصادقة ومفاتيح الشركة الأخرى مثل الموارد البشرية والمشتريات والمالية

تفاصيل التنفيذ:

```
# إضافة إلى requirements.txt
# djangorestframework==3.14.0

# إضافة إلى settings.py
INSTALLED_APPS = [
    # ... التطبيقات الحالية
    'rest_framework',
    'rest_framework.authtoken',
]

REST_FRAMEWORK = {
    'DEFAULT_AUTHENTICATION_CLASSES': [
        'rest_framework.authentication.TokenAuthentication',
        'rest_framework.authentication.SessionAuthentication',
    ],
    'DEFAULT_PERMISSION_CLASSES': [
        'rest_framework.permissions.IsAuthenticated',
    ],
}

# طرق عرض API جديدة
from rest_framework import viewsets
from rest_framework.permissions import IsAuthenticated
from .models import ITAsset, Employee, Department
from .serializers import ITAssetSerializer, EmployeeSerializer,
DepartmentSerializer

class CompanyFilteredViewSet(viewsets.ModelViewSet):
    """ViewSet أساسي يقوم بتصفية مجموعة الاستعلام حسب شركة المستخدم"""

    def get_queryset(self):
        queryset = super().get_queryset()
        if not self.request.user.is_superuser:
            company = self.request.company
            queryset = queryset.filter(company=company)
        return queryset

class ITAssetViewSet(CompanyFilteredViewSet):
    serializer_class = ITAssetSerializer
    queryset = ITAsset.objects.all()

class EmployeeViewSet(CompanyFilteredViewSet):
    serializer_class = EmployeeSerializer
    queryset = Employee.objects.all()

class DepartmentViewSet(CompanyFilteredViewSet):
    serializer_class = DepartmentSerializer
    queryset = Department.objects.all()
```

6. سير عمل وإشعارات قابلة للتخصيص

التنفيذ الحالي: النظام لديه سير عمل ثابت لإدارة الأصول بدون خيارات تخصيص.

التعديلات الموصى بها: - تنفيذ محرك سير عمل يسمح للشركات بتحديد عمليات الموافقة الخاصة بها - إضافة إشعارات قابلة للتخصيص لأحداث مختلفة (تعيينات الأصول، انتهاء الضمان، إلخ) - إنشاء قوالب خاصة بالشركة للبريد الإلكتروني والتقارير - السماح بتكوين تنبيهات تلقائية بناءً على سياسات الشركة

تفاصيل التنفيذ:

```
class WorkflowDefinition(CompanyRelatedModel):
    name = models.CharField(max_length=100)
    description = models.TextField(blank=True)
    workflow_type = models.CharField(max_length=50, choices=[
        ('asset_request', 'طلب أصل'),
        ('asset_return', 'إرجاع أصل'),
        ('maintenance', 'طلب صيانة'),
    ])
    requires_approval = models.BooleanField(default=True)
    approvers = models.ManyToManyField(User,
related_name='approval_workflows')

    class Meta:
        unique_together = ('company', 'workflow_type')

class WorkflowInstance(models.Model):
    definition = models.ForeignKey(WorkflowDefinition, on_delete=models.CASCADE)
    asset = models.ForeignKey(ITAsset, on_delete=models.CASCADE, null=True,
blank=True)
    requester = models.ForeignKey(User, on_delete=models.CASCADE,
related_name='requested_workflows')
    status = models.CharField(max_length=20, choices=[
        ('pending', 'قيد الانتظار'),
        ('approved', 'تمت الموافقة'),
        ('rejected', 'مرفوض'),
        ('completed', 'مكتمل'),
    ], default='pending')
    created_at = models.DateTimeField(auto_now_add=True)
    updated_at = models.DateTimeField(auto_now=True)
```

7. تعزيز الاستيراد/التصدير مع القوالب

التنفيذ الحالي: النظام لديه وظائف استيراد/تصدير Excel أساسية ولكنه يفتقر إلى قوالب خاصة بالشركة.

التعديلات الموصى بها: - إنشاء قوالب استيراد/تصدير خاصة بالشركة - إضافة دعم لتنسيقات ملفات مختلفة (CSV، JSON) - تنفيذ التحقق من صحة البيانات الخاصة بمتطلبات الشركة - إضافة عمليات جماعية لإدارة الأصول

تفاصيل التنفيذ:

```
class ImportTemplate(CompanyRelatedModel):
    name = models.CharField(max_length=100)
    template_type = models.CharField(max_length=50, choices=[
        ('employee', 'استيراد الموظفين'),
        ('asset', 'استيراد الأصول'),
    ])
    template_file = models.FileField(upload_to='import_templates/')
    field_mapping = models.JSONField(default=dict)  # تخزين تعيينات الأعمدة
    created_at = models.DateTimeField(auto_now_add=True)
    updated_at = models.DateTimeField(auto_now=True)

    class Meta:
        unique_together = ('company', 'template_type', 'name')
```

8. التوطين والتدويل

التنفيذ الحالي: يبدو أن النظام بشكل أساسي باللغة الإنجليزية مع دعم محدود للتوطين.

التعديلات الموصى بها: - تنفيذ التدويل الكامل باستخدام إطار عمل i18n في Django - السماح بإعدادات لغة خاصة بالشركة - دعم تنسيقات متعددة للتاريخ والوقت والأرقام - إضافة قدرات الترجمة للمحتوى الذي ينشئه المستخدم

تفاصيل التنفيذ:

```
# تعديلات settings.py
MIDDLEWARE = [
    # ... الوسيطة الحالية
    'django.middleware.locale.LocaleMiddleware',
]

USE_I18N = True
USE_L10N = True

LANGUAGES = [
    ('en', 'الإنجليزية'),
    ('es', 'الإسبانية'),
    ('fr', 'الفرنسية'),
    ('ar', 'العربية'),
    # أضف المزيد من اللغات حسب الحاجة
]

LOCALE_PATHS = [
    BASE_DIR / 'locale',
]
```

```
# إضافة إلى نموذج الشركة
class OwnerCompany(models.Model):
    # ... الحقول الحالية
    default_language = models.CharField(max_length=10,
choices=settings.LANGUAGES, default='en')
    date_format = models.CharField(max_length=20, default='Y-m-d')
    time_format = models.CharField(max_length=20, default='H:i')
```

9. تعزيز ميزات الأمان

التنفيذ الحالي: مصادقة Django الأساسية مع ميزات أمان محدودة.

التعديلات الموصى بها: - تنفيذ المصادقة الثنائية - إضافة قيود الوصول المستندة إلى IP لكل شركة - إنشاء سجل تدقيق للعمليات الحساسة - إضافة تشفير البيانات للحقول الحساسة - تنفيذ إعدادات مهلة الجلسة لكل شركة

تفاصيل التنفيذ:

```
 # إضافات إلى requirements.txt
# django-two-factor-auth==1.14.0
# django-axes==5.40.1

# إضافات إلى settings.py
INSTALLED_APPS = [
    # ... التطبيقات الحالية
    'django_otp',
    'django_otp.plugins.otp_totp',
    'django_otp.plugins.otp_static',
    'two_factor',
    'axes',
]

MIDDLEWARE = [
    # ... الوسيطة الحالية
    'axes.middleware.AxesMiddleware',
]

# نموذج إعدادات أمان الشركة
class CompanySecurity(models.Model):
    company = models.OneToOneField(OwnerCompany,
on_delete=models.CASCADE, related_name='security_settings')
    require_2fa = models.BooleanField(default=False)
    session_timeout_minutes = models.IntegerField(default=30)
    password_expiry_days = models.IntegerField(default=90)
    allowed_ip_ranges = models.TextField(blank=True,
help_text="قائمة نطاقات IP مفصولة بفواصل")
    failed_login_attempts = models.IntegerField(default=5)
    account_lockout_minutes = models.IntegerField(default=30)
```

10. بنية قابلة للتوسع للمؤسسات الكبيرة

التنفيذ الحالي: يبدو أن النظام مصمم للمؤسسات الصغيرة إلى المتوسطة بدون تحسينات محددة للمؤسسات الكبيرة.

التعديلات الموصى بها: - تنفيذ تحسين استعلامات قاعدة البيانات لمجموعات البيانات الكبيرة - إضافة طبقات التخزين المؤقت للبيانات التي يتم الوصول إليها بشكل متكرر - إنشاء نظام قائمة مهام للمعالجة في الخلفية - تنفيذ قدرات التوسع الأفقي - إضافة مراقبة الأداء والتحليلات

تفاصيل التنفيذ:

```
 # إضافات إلى requirements.txt
# django-redis==5.2.0
# celery==5.2.7
# django-celery-beat==2.4.0

# إضافات إلى settings.py
CACHES = {
    'default': {
        'BACKEND': 'django_redis.cache.RedisCache',
        'LOCATION': 'redis://127.0.0.1:6379/1',
        'OPTIONS': {
            'CLIENT_CLASS': 'django_redis.client.DefaultClient',
        }
    }
}

# تخزين مؤقت للبيانات التي يتم الوصول إليها بشكل متكرر
@method_decorator(cache_page(60 * 15), name='dispatch')
class DashboardView(LoginRequiredMixin, TemplateView):
    template_name = 'inventory/home.html'

    def get_context_data(self, **kwargs):
        context = super().get_context_data(**kwargs)
        company = self.request.company

        # استخدام select_related و prefetch_related لتحسين الاستعلامات
        context['recent_assets'] =
ITAsset.objects.filter(company=company).select_related(
            'asset_type', 'owner', 'assigned_to'
        ).order_by('-id')[:5]

        # ... استعلامات محسنة أخرى

        return context
```

استراتيجية التنفيذ

لتنفيذ هذه التعديلات بشكل فعال، أوصي بالنهج المرحلي التالي:

المرحلة 1: أساس دعم الشركات المتعددة

1. تنفيذ نموذج `CompanyRelatedModel` الأساسي وتعديل النماذج الحالية
2. إنشاء middleware للشركة للتصفية التلقائية
3. تعزيز نظام المستخدمين والصلاحيات
4. تحديث طرق العرض لاحترام حدود الشركة

المرحلة 2: التخصيص الخاص بالشركة

1. تنفيذ نموذج `CompanyConfiguration`
2. إنشاء لوحات معلومات وتقارير خاصة بالشركة
3. إضافة سير عمل قابل للتخصيص
4. تنفيذ قوالب خاصة بالشركة

المرحلة 3: الميزات المتقدمة والتوسع

1. تطوير واجهة برمجة التطبيقات REST
2. تنفيذ التدويل والتوطين
3. إضافة ميزات أمان معززة
4. تحسين الأداء وقابلية التوسع

الخلاصة

ستعمل هذه التعديلات على تحويل نظام إدارة أصول تكنولوجيا المعلومات من أداة لمؤسسة واحدة إلى منصة مرنة متعددة الشركات مناسبة للهياكل التنظيمية المتنوعة. سيوفر النظام المحسن عزلاً مناسباً للبيانات، وتخصيصاً خاصاً بالشركة، وميزات على مستوى المؤسسات مع الحفاظ على الوظائف الأساسية لإدارة الأصول والموظفين.

من خلال تنفيذ هذه التغييرات، سيكون النظام قابلاً للتكيف مع: - الشركات متعددة الجنسيات ذات الشركات التابعة المختلفة - مزودي الخدمات المدارة الذين يدعمون شركات عملاء متعددة - الشركات القابضة ذات وحدات أعمال متنوعة المؤسسات ذات الهياكل القسمية المعقدة - الشركات ذات متطلبات الامتثال والتنظيم المختلفة -

يتيح النهج النمطي لهذه التعديلات التنفيذ التدريجي، حيث تبني كل مرحلة على المرحلة السابقة لتعزيز قدرات النظام تدريجياً مع تقليل الاضطراب للمستخدمين الحاليين.