Enterprise Multi-Tenant CRM System

Complete Implementation Guide

📋 Table of Contents

🏗️ System Architecture Overview

Technology Stack

Backend:

- Django 4.2+ (Python 3.11+)
- Django REST Framework 3.14+
- PostgreSQL 14+ with Row-Level Security
- Redis 7+ for caching and sessions
- Celery for background tasks

Frontend:

- React 18+ with TypeScript
- Tailwind CSS for styling
- Axios for API calls

- React Router v6 for routing
- Recharts for data visualization

Key Features

✅ Multi-tenant architecture with company isolation
✅ Unlimited territory management with hierarchies
✅ Complete Quote-to-Cash workflow
✅ Vendor management and procurement
✅ Custom document templates
✅ Sales forecasting and DSO tracking
✅ Role-based access control
✅ Mobile-responsive design

📦 Prerequisites & Tools

Required Software

1. **Python 3.11+**

```bash
python --version  # Should be 3.11 or higher
```

2. **PostgreSQL 14+**

```bash
psql --version  # Should be 14 or higher
```

3. **Node.js 18+**

```bash
node --version  # Should be 18 or higher
```

4. **Redis 7+**

```bash
redis-server --version  # Should be 7 or higher
```

5. **Git**

```bash
git --version
```

6. **Code Editor**: VS Code (recommended)

Optional but Recommended

- Docker & Docker Compose
- Postman or Insomnia (API testing)
- pgAdmin or DBeaver (database management)

🚀 Phase 1: Backend Setup

Step 1.1: Create Project Structure

```bash
# Create main project directory
mkdir enterprise-crm
cd enterprise-crm

# Create backend directory
mkdir backend
cd backend

# Create Python virtual environment
python -m venv venv

# Activate virtual environment
# On Windows:
venv\Scripts\activate
# On macOS/Linux:
source venv/bin/activate
```

Step 1.2: Install Python Dependencies

```bash
```

```
# Install Django and related packages
pip install django==4.2.8
pip install djangorestframework==3.14.0
pip install psycopg2-binary==2.9.9
pip install django-cors-headers==4.3.1
pip install djangorestframework-simplejwt==5.3.1
pip install django-filter==23.5
pip install celery==5.3.4
pip install redis==5.0.1
pip install python-decouple==3.8
pip install pillow==10.1.0
pip install reportlab==4.0.7
pip install weasyprint==60.1
pip install openpyxl==3.1.2

# Save dependencies
pip freeze > requirements.txt
```

Step 1.3: Create Django Project

```bash
# Create Django project
django-admin startproject config .

# Create apps
python manage.py startapp core         # Core models (User, Company)
python manage.py startapp crm          # CRM models (Account, Contact, Lead, Deal)
python manage.py startapp sales        # Sales models (Quote, Order, Invoice)
python manage.py startapp procurement # Procurement models (Vendor, PO)
python manage.py startapp territories # Territory management
python manage.py startapp reporting   # Reports and analytics
```

Step 1.4: Configure Settings

Create `config/settings/base.py`:

```python
```

```
import os
from pathlib import Path
from decouple import config

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

SECRET_KEY = config('SECRET_KEY', default='your-secret-key-here-change-in-production')

DEBUG = config('DEBUG', default=False, cast=bool)

ALLOWED_HOSTS = config('ALLOWED_HOSTS', default='localhost,127.0.0.1').split(',')

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

    # Third-party apps
    'rest_framework',
    'rest_framework_simplejwt',
    'corsheaders',
    'django_filters',

    # Local apps
    'core',
    'crm',
    'sales',
    'procurement',
    'territories',
    'reporting',
]

MIDDLEWARE = [
    'django.middleware.security.SecurityMiddleware',
    'corsheaders.middleware.CorsMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'core.middleware.MultiTenantMiddleware',  # Custom multi-tenant middleware
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
]

ROOT_URLCONF = 'config.urls'

TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [BASE_DIR / 'templates'],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
            ],
        },
    },
]

WSGI_APPLICATION = 'config.wsgi.application'

# Database
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql',
        'NAME': config('DB_NAME', default='enterprise_crm'),
        'USER': config('DB_USER', default='postgres'),
        'PASSWORD': config('DB_PASSWORD', default='postgres'),
        'HOST': config('DB_HOST', default='localhost'),
        'PORT': config('DB_PORT', default='5432'),
        'OPTIONS': {
            'options': '-c search_path=public'
        }
    }
}

# Custom User Model
AUTH_USER_MODEL = 'core.User'

# REST Framework
REST_FRAMEWORK = {
    'DEFAULT_AUTHENTICATION_CLASSES': [
        'rest_framework_simplejwt.authentication.JWTAuthentication',
    ],
    'DEFAULT_PERMISSION_CLASSES': [
        'rest_framework.permissions.IsAuthenticated',
    ],
    'DEFAULT_PAGINATION_CLASS': 'rest_framework.pagination.PageNumberPagination',
    'PAGE_SIZE': 25,
    'DEFAULT_FILTER_BACKENDS': [
        'django_filters.rest_framework.DjangoFilterBackend',
        'rest_framework.filters.SearchFilter',
        'rest_framework.filters.OrderingFilter',
    ],
}

# JWT Settings
from datetime import timedelta

SIMPLE_JWT = {
    'ACCESS_TOKEN_LIFETIME': timedelta(hours=1),
    'REFRESH_TOKEN_LIFETIME': timedelta(days=7),
    'ROTATE_REFRESH_TOKENS': True,
}

# CORS Settings
CORS_ALLOWED_ORIGINS = config(
    'CORS_ALLOWED_ORIGINS',
    default='http://localhost:3000,http://127.0.0.1:3000'
).split(',')

CORS_ALLOW_CREDENTIALS = True

# Redis/Celery
REDIS_HOST = config('REDIS_HOST', default='localhost')
REDIS_PORT = config('REDIS_PORT', default='6379')
REDIS_URL = f'redis://{REDIS_HOST}:{REDIS_PORT}/0'

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

```
CELERY_BROKER_URL = REDIS_URL
CELERY_RESULT_BACKEND = REDIS_URL

# Static files
STATIC_URL = '/static/'
STATIC_ROOT = BASE_DIR / 'staticfiles'

# Media files
MEDIA_URL = '/media/'
MEDIA_ROOT = BASE_DIR / 'media'

# Email configuration (for production)
EMAIL_BACKEND = 'django.core.mail.backends.smtp.EmailBackend'
EMAIL_HOST = config('EMAIL_HOST', default='')
EMAIL_PORT = config('EMAIL_PORT', default=587, cast=int)
EMAIL_HOST_USER = config('EMAIL_HOST_USER', default='')
EMAIL_HOST_PASSWORD = config('EMAIL_HOST_PASSWORD', default='')
EMAIL_USE_TLS = True

DEFAULT_AUTO_FIELD = 'django.db.models.BigAutoField'
```

Create `.env` file:

```env
DEBUG=True
SECRET_KEY=your-secret-key-here-generate-random
DB_NAME=enterprise_crm
DB_USER=postgres
DB_PASSWORD=your-postgres-password
DB_HOST=localhost
DB_PORT=5432
REDIS_HOST=localhost
REDIS_PORT=6379
CORS_ALLOWED_ORIGINS=http://localhost:3000
```

💾 Phase 2: Database Implementation

Step 2.1: Create Database

```sql
```

```
-- Connect to PostgreSQL
psql -U postgres

-- Create database
CREATE DATABASE enterprise_crm;

-- Connect to the database
\c enterprise_crm

-- Enable required extensions
CREATE EXTENSION IF NOT EXISTS "uuid-ossp";
CREATE EXTENSION IF NOT EXISTS "pgcrypto";
```

Step 2.2: Run Database Schema

Use the complete SQL schema provided in the artifacts:

- Database Schema Part 1 (Core tables)
- Database Schema Part 2 (Activities & Quote-to-Cash)
- Database Schema Part 3 (Sales Orders & Vendors)

```bash
# Run the schema files
psql -U postgres -d enterprise_crm -f schema_part1.sql
psql -U postgres -d enterprise_crm -f schema_part2.sql
psql -U postgres -d enterprise_crm -f schema_part3.sql
```

Step 2.3: Create Django Models

Copy the Django models from the artifacts into your Django apps:

- `core/models.py` - User, Company, UserCompanyAccess
- `territories/models.py` - Territory, TerritoryRule
- `crm/models.py` - Account, Contact, Lead, Deal
- `sales/models.py` - Quote, SalesOrder, Invoice
- `procurement/models.py` - Vendor, PurchaseOrder

Step 2.4: Create and Run Migrations

```bash
```

```
# Create migrations
python manage.py makemigrations

# Apply migrations
python manage.py migrate

# Create superuser
python manage.py createsuperuser
```

🔌 Phase 3: API Development

Step 3.1: Create Serializers

For each model, create serializers in `<app>/serializers.py`:

```python
# Example: crm/serializers.py
from rest_framework import serializers
from .models import Account, Contact, Lead, Deal

class AccountSerializer(serializers.ModelSerializer):
    owner_name = serializers.CharField(source='owner.full_name', read_only=True)

    class Meta:
        model = Account
        fields = '__all__'
        read_only_fields = ['id', 'created_at', 'updated_at']
```

Step 3.2: Create ViewSets

Create ViewSets in `<app>/views.py`:

```python
```

```python
# Example: crm/views.py
from rest_framework import viewsets
from rest_framework.permissions import IsAuthenticated
from .models import Account
from .serializers import AccountSerializer

class AccountViewSet(viewsets.ModelViewSet):
    serializer_class = AccountSerializer
    permission_classes = [IsAuthenticated]

    def get_queryset(self):
        company_id = self.request.session.get('active_company_id')
        return Account.objects.filter(company_id=company_id)
```

Step 3.3: Configure URLs

python

```python
# config/urls.py
from django.contrib import admin
from django.urls import path, include
from rest_framework.routers import DefaultRouter

router = DefaultRouter()
# Register all viewsets here

urlpatterns = [
    path('admin/', admin.site.urls),
    path('api/v1/', include(router.urls)),
    path('api/v1/auth/', include('core.urls.auth')),
]
```

Step 3.4: Test API Endpoints

bash

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

# Test in another terminal or Postman
curl http://localhost:8000/api/v1/accounts/
```

🎨 Phase 4: Frontend Setup

Step 4.1: Create React App

```bash
# Go back to project root
cd ..

# Create React app with TypeScript
npx create-react-app frontend --template typescript

cd frontend

# Install dependencies
npm install axios react-router-dom@6
npm install tailwindcss@latest postcss@latest autoprefixer@latest
npm install recharts date-fns react-hook-form
npm install lucide-react react-hot-toast
npm install @tanstack/react-query
```

Step 4.2: Configure Tailwind CSS

```bash
npx tailwindcss init -p
```

Edit `tailwind.config.js`:

```javascript
module.exports = {
  content: [
    "./src/**/*.{js,jsx,ts,tsx}",
  ],
  theme: {
    extend: {},
  },
  plugins: [],
}
```

Step 4.3: Create Project Structure

Follow the structure provided in the "React Frontend Structure" artifact.

```bash
mkdir -p src/{api,components,pages,context,hooks,utils,routes,styles}
mkdir -p src/components/{common,layout,forms,cards,charts,widgets}
mkdir -p src/pages/{Auth,Dashboard,Accounts,Contacts,Leads,Deals}
```

Step 4.4: Set Up API Client

Create `src/api/client.ts` as shown in the artifacts.

Step 4.5: Create Context Providers

Create AuthContext and CompanyContext as shown in artifacts.

🔗 Phase 5: Integration

Step 5.1: Connect Frontend to Backend

Update `.env` in frontend:

```env
REACT_APP_API_BASE_URL=http://localhost:8000/api/v1
```

Step 5.2: Implement Authentication Flow

1. Create Login page
2. Implement JWT token handling
3. Add protected routes
4. Implement company switching

Step 5.3: Build Core Features

Implement in this order:

1. Dashboard with summary widgets
2. Accounts module (list, create, detail)
3. Contacts module
4. Leads module
5. Deals & Pipeline

6. Quotes & RFQ
7. Sales Orders
8. Vendors & Procurement
9. Territories
10. Reports

🧪 Phase 6: Testing

Backend Tests

```python
# tests/test_accounts.py
from django.test import TestCase
from crm.models import Account

class AccountTestCase(TestCase):
    def setUp(self):
        # Create test data
        pass

    def test_create_account(self):
        # Test account creation
        pass
```

Run tests:

```bash
python manage.py test
```

Frontend Tests

```bash
npm test
```

🚀 Phase 7: Deployment

Production Checklist

- [ ] Set DEBUG=False
- [ ] Use environment variables for secrets
- [ ] Configure ALLOWED_HOSTS
- [ ] Set up HTTPS/SSL
- [ ] Configure static file serving
- [ ] Set up database backups
- [ ] Configure logging
- [ ] Set up monitoring (Sentry)
- [ ] Optimize database queries
- [ ] Add caching
- [ ] Set up CDN for static files

Deployment Options

Option 1: Traditional Server (DigitalOcean, AWS EC2)

- Deploy Django with Gunicorn + Nginx
- Serve React build files
- PostgreSQL on same server or managed service
- Redis on same server

Option 2: Platform as a Service

- Railway.app or Render for backend
- Vercel or Netlify for frontend
- Managed PostgreSQL (Railway, Supabase)
- Redis from Railway or Upstash

Option 3: Docker Containers

- Create Dockerfiles for backend and frontend
- Use Docker Compose for local development
- Deploy to AWS ECS, Google Cloud Run, or DigitalOcean App Platform

---

📈 Next Steps After MVP

1. **Phase 2 Features:**
    - AI-based lead scoring
    - Predictive sales forecasting
    - Advanced workflow automation
    - Email integration (Gmail, Outlook)
    - Calendar integration
    - Mobile apps (React Native)
2. **Optimization:**
    - Performance monitoring
    - Database query optimization
    - Implement caching strategies
    - Add full-text search (Elasticsearch)
3. **Integrations:**
    - Payment gateways
    - Accounting software (QuickBooks)
    - Marketing automation
    - E-signature (DocuSign)
    - Telephony integration
4. **Scaling:**
    - Load balancing
    - Database replication
    - Microservices architecture
    - API rate limiting

📚 Resources

- Django Documentation: https://docs.djangoproject.com/
- Django REST Framework: https://www.django-rest-framework.org/
- React Documentation: https://react.dev/
- PostgreSQL RLS: https://www.postgresql.org/docs/current/ddl-rowsecurity.html

- Tailwind CSS: https://tailwindcss.com/docs

🆘 Getting Help

If you encounter issues:

1. Check error logs
2. Review database migrations
3. Verify environment variables
4. Test API endpoints with Postman
5. Check browser console for frontend errors

Ready to start building? Let's go phase by phase! 🚀