Phase 1: Core Platform & Multi-Company Foundation

Implementation Guide for Visual Drag-and-Drop ERP

Duration: 8–10 weeks
Version: 1.0
Date: October 2025

1. Overview

Phase 1 builds the technical foundation for the entire ERP system, focusing on modular architecture, multi-company data layer, security framework, and core platform services.

Key Objectives

- Implement modular plugin architecture
- Build multi-company data isolation layer
- Create authentication and authorization system
- Deploy embedded PostgreSQL database
- Establish API gateway and event bus
- Set up development and deployment infrastructure

2. Architecture Implementation

2.1 Project Structure

```
erp-platform/
├── backend/
│   ├── core/                      # Core platform services
│   │   ├── __init__.py
│   │   ├── settings.py           # Django settings
│   │   ├── urls.py               # URL routing
│   │   ├── wsgi.py
│   │   ├── asgi.py
│   │   └── celery.py             # Task queue config
│   ├── apps/
│   │   ├── authentication/       # Auth module
│   │   ├── companies/            # Multi-company management
│   │   ├── users/                # User management
│   │   ├── permissions/          # RBAC/ABAC
│   │   ├── workflows/            # Workflow engine
│   │   ├── api_gateway/          # API management
```

```
│   │   └── modules/              # Module registry
│   ├── shared/                   # Shared utilities
│   │   ├── middleware/           # Company context, auth
│   │   ├── models.py             # Base models
│   │   ├── serializers.py        # Base serializers
│   │   ├── permissions.py        # Permission helpers
│   │   └── utils.py
│   ├── embedded_db/              # PostgreSQL management
│   │   ├── init_db.py
│   │   ├── backup.py
│   │   └── migrate.py
│   ├── requirements.txt
│   ├── Dockerfile
│   └── manage.py
├── frontend/
│   ├── src/
│   │   ├── components/           # Reusable components
│   │   │   ├── Layout/
│   │   │   ├── CompanySwitcher/
│   │   │   ├── Navigation/
│   │   │   └── AIAssistant/      # Placeholder
│   │   ├── modules/              # Feature modules
│   │   │   ├── auth/
│   │   │   ├── companies/
│   │   │   └── dashboard/
│   │   ├── services/             # API clients
│   │   ├── store/                # State management
│   │   ├── utils/
│   │   ├── App.jsx
│   │   └── main.jsx
│   ├── public/
│   ├── package.json
│   ├── vite.config.js
│   └── Dockerfile
├── docker-compose.yml
├── nginx/
│   └── nginx.conf
├── scripts/
│   ├── setup_dev.sh
│   ├── init_embedded_db.sh
│   └── deploy.sh
├── docs/
│   ├── architecture.md
│   ├── api_docs.md
│   └── setup_guide.md
└── README.md
```

3. Multi-Company Data Layer Implementation

3.1 Base Model with Company Isolation

```
# backend/shared/models.py
from django.db import models
from django.contrib.auth import get_user_model

class CompanyAwareModel(models.Model):
    """
    Abstract base model that adds company isolation
    to all transactional data
    """
    company = models.ForeignKey(
        'companies.Company',
        on_delete=models.PROTECT,
        db_index=True,
        help_text="Company this record belongs to"
    )
    created_at = models.DateTimeField(auto_now_add=True)
    updated_at = models.DateTimeField(auto_now=True)
    created_by = models.ForeignKey(
        get_user_model(),
        on_delete=models.SET_NULL,
        null=True,
        related_name='+'
    )

    class Meta:
        abstract = True

    def save(self, *args, **kwargs):
        # Validate company access
        if not self.company_id:
            raise ValueError("Company must be specified")
        super().save(*args, **kwargs)
```

3.2 Company Model

```
# backend/apps/companies/models.py
from django.db import models
from django.core.validators import RegexValidator

class Company(models.Model):
    """
    Company/Legal Entity Master
    Supports parent-subsidiary relationships
    """
    code = models.CharField(
        max_length=10,
        unique=True,
        validators=[RegexValidator(r'^[A-Z0-9]+$')],
        help_text="Unique company code"
    )
    name = models.CharField(max_length=255)
    legal_name = models.CharField(max_length=255)
```

```
    parent_company = models.ForeignKey(
        'self',
        on_delete=models.PROTECT,
        null=True,
        blank=True,
        related_name='subsidiaries'
    )

    # Financial Settings
    currency_code = models.CharField(max_length=3, default='BDT')
    fiscal_year_start = models.DateField()

    # Tax &amp;amp; Legal
    tax_id = models.CharField(max_length=50, unique=True)
    registration_number = models.CharField(max_length=100)

    # Configuration
    settings = models.JSONField(default=dict, blank=True)
    is_active = models.BooleanField(default=True)

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

    class Meta:
        verbose_name_plural = "Companies"
        ordering = ['code']

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

    def get_hierarchy_level(self):
        """Returns depth in company hierarchy"""
        level = 0
        parent = self.parent_company
        while parent:
            level += 1
            parent = parent.parent_company
        return level
```

3.3 Company Context Middleware

```
# backend/shared/middleware/company_context.py
from django.utils.deprecation import MiddlewareMixin
from apps.companies.models import Company

class CompanyContextMiddleware(MiddlewareMixin):
    """
    Automatically injects company context into requests
    based on user session or header
    """
    def process_request(self, request):
        if request.user.is_authenticated:
            # Get company from session or user default
            company_id = request.session.get('active_company_id')
```

```
            if not company_id:
                # Get user's default or first company
                user_companies = request.user.companies.filter(is_active=True)
                if user_companies.exists():
                    company_id = user_companies.first().id
                    request.session['active_company_id'] = company_id

            if company_id:
                try:
                    request.company = Company.objects.get(
                        id=company_id,
                        is_active=True
                    )
                except Company.DoesNotExist:
                    request.company = None
            else:
                request.company = None
        else:
            request.company = None

        return None
```

3.4 QuerySet Manager with Auto-Filtering

```
# backend/shared/managers.py
from django.db import models

class CompanyQuerySet(models.QuerySet):
    def for_company(self, company):
        """Filter by company"""
        return self.filter(company=company)

    def for_user(self, user):
        """Filter by user's accessible companies"""
        return self.filter(company__in=user.companies.all())

class CompanyManager(models.Manager):
    def get_queryset(self):
        return CompanyQuerySet(self.model, using=self._db)

    def for_company(self, company):
        return self.get_queryset().for_company(company)

    def for_user(self, user):
        return self.get_queryset().for_user(user)
```

4. Authentication & Authorization System

4.1 User Model Extension

```
# backend/apps/users/models.py
from django.contrib.auth.models import AbstractUser
from django.db import models

class User(AbstractUser):
    """
    Extended user model with multi-company support
    """
    companies = models.ManyToManyField(
        'companies.Company',
        through='UserCompanyRole',
        related_name='users'
    )
    default_company = models.ForeignKey(
        'companies.Company',
        on_delete=models.SET_NULL,
        null=True,
        blank=True,
        related_name='default_users'
    )
    phone = models.CharField(max_length=20, blank=True)
    avatar = models.ImageField(upload_to='avatars/', null=True, blank=True)
    is_system_admin = models.BooleanField(default=False)

    class Meta:
        db_table = 'users'

    def has_company_access(self, company):
        """Check if user has access to company"""
        return self.companies.filter(id=company.id).exists()

class UserCompanyRole(models.Model):
    """
    Many-to-many relationship between users and companies
    with role assignment
    """
    user = models.ForeignKey(User, on_delete=models.CASCADE)
    company = models.ForeignKey('companies.Company', on_delete=models.CASCADE)
    role = models.ForeignKey('permissions.Role', on_delete=models.PROTECT)
    is_active = models.BooleanField(default=True)
    assigned_at = models.DateTimeField(auto_now_add=True)

    class Meta:
        unique_together = [['user', 'company', 'role']]
        db_table = 'user_company_roles'
```

4.2 Permission System

```
# backend/apps/permissions/models.py
from django.db import models

class Permission(models.Model):
    """
    Granular permissions for modules and actions
    """
    code = models.CharField(max_length=100, unique=True)
    name = models.CharField(max_length=255)
    module = models.CharField(max_length=50)
    description = models.TextField(blank=True)

    class Meta:
        db_table = 'permissions'

class Role(models.Model):
    """
    Role definitions with permission sets
    Can be company-specific or global
    """
    name = models.CharField(max_length=100)
    company = models.ForeignKey(
        'companies.Company',
        on_delete=models.CASCADE,
        null=True,
        blank=True,
        help_text="Null for global roles"
    )
    permissions = models.ManyToManyField(Permission)
    description = models.TextField(blank=True)
    is_system_role = models.BooleanField(default=False)

    class Meta:
        unique_together = [['name', 'company']]
        db_table = 'roles'

    def __str__(self):
        if self.company:
            return f"{self.name} ({self.company.code})"
        return self.name
```

5. API Gateway & Event Bus

5.1 API Versioning and Gateway

```
# backend/apps/api_gateway/urls.py
from django.urls import path, include

urlpatterns = [
    path('v1/auth/', include('apps.authentication.urls')),
```

```
    path('v1/companies/', include('apps.companies.urls')),
    path('v1/users/', include('apps.users.urls')),
    # Future modules will register here
]
```

5.2 Event Bus Implementation

```
# backend/shared/events.py
import redis
import json
from django.conf import settings

class EventBus:
    """
    Simple pub/sub event bus using Redis
    For inter-module communication
    """
    def __init__(self):
        self.redis_client = redis.Redis(
            host=settings.REDIS_HOST,
            port=settings.REDIS_PORT,
            db=settings.REDIS_DB
        )
        self.pubsub = self.redis_client.pubsub()

    def publish(self, channel, event_type, data):
        """Publish event to channel"""
        message = {
            'event_type': event_type,
            'data': data,
            'timestamp': timezone.now().isoformat()
        }
        self.redis_client.publish(channel, json.dumps(message))

    def subscribe(self, channel, callback):
        """Subscribe to channel with callback"""
        self.pubsub.subscribe(channel)
        for message in self.pubsub.listen():
            if message['type'] == 'message':
                data = json.loads(message['data'])
                callback(data)

# Usage example
event_bus = EventBus()

# Publish event
event_bus.publish(
    'company.events',
    'company.created',
    {'company_id': 1, 'name': 'ACME Corp'}
)
```

6. Embedded PostgreSQL Setup

6.1 Database Initialization Script

```
# backend/embedded_db/init_db.py
import os
import subprocess
import sys
from pathlib import Path

class EmbeddedPostgres:
    """
    Manages embedded PostgreSQL instance
    """
    def __init__(self, data_dir='./pgdata', port=54322):
        self.data_dir = Path(data_dir).absolute()
        self.port = port
        self.pg_ctl = 'pg_ctl'  # Assumes PostgreSQL in PATH

    def initialize(self):
        """Initialize PostgreSQL data directory"""
        if self.data_dir.exists():
            print(f"Database already initialized at {self.data_dir}")
            return

        print(f"Initializing PostgreSQL at {self.data_dir}...")
        cmd = ['initdb', '-D', str(self.data_dir), '-U', 'postgres']
        subprocess.run(cmd, check=True)

        # Configure port
        self._configure_port()
        print("PostgreSQL initialized successfully")

    def _configure_port(self):
        """Set custom port in postgresql.conf"""
        conf_file = self.data_dir / 'postgresql.conf'
        with open(conf_file, 'a') as f:
            f.write(f"\nport = {self.port}\n")

    def start(self):
        """Start PostgreSQL server"""
        cmd = [self.pg_ctl, '-D', str(self.data_dir), 'start']
        subprocess.run(cmd, check=True)
        print(f"PostgreSQL started on port {self.port}")

    def stop(self):
        """Stop PostgreSQL server"""
        cmd = [self.pg_ctl, '-D', str(self.data_dir), 'stop']
        subprocess.run(cmd, check=True)
        print("PostgreSQL stopped")

    def status(self):
        """Check PostgreSQL status"""
        cmd = [self.pg_ctl, '-D', str(self.data_dir), 'status']
        result = subprocess.run(cmd, capture_output=True)
```

```
        return result.returncode == 0

if __name__ == '__main__':
    db = EmbeddedPostgres()

    if len(sys.argv) &gt; 1:
        action = sys.argv[1]
        if action == 'init':
            db.initialize()
        elif action == 'start':
            db.start()
        elif action == 'stop':
            db.stop()
        elif action == 'status':
            print("Running" if db.status() else "Stopped")
    else:
        print("Usage: python init_db.py [init|start|stop|status]")
```

6.2 Django Database Configuration

```
# backend/core/settings.py
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql',
        'NAME': 'erp_db',
        'USER': 'postgres',
        'PASSWORD': 'your_secure_password',
        'HOST': 'localhost',
        'PORT': '54322',  # Embedded PostgreSQL port
    }
}
```

7. Frontend Foundation

7.1 Company Switcher Component

```
// frontend/src/components/CompanySwitcher/CompanySwitcher.jsx
import React, { useState, useEffect } from 'react';
import { Select } from 'antd';
import { useDispatch, useSelector } from 'react-redux';
import { setActiveCompany } from '../../store/companySlice';
import api from '../../services/api';

const CompanySwitcher = () =&gt; {
  const dispatch = useDispatch();
  const { companies, activeCompany } = useSelector(state =&gt; state.company);
  const [loading, setLoading] = useState(false);

  const handleCompanyChange = async (companyId) =&gt; {
    setLoading(true);
    try {
```

```
      await api.post('/companies/switch/', { company_id: companyId });
      dispatch(setActiveCompany(companyId));
      window.location.reload(); // Refresh to load company-specific data
    } catch (error) {
      console.error('Failed to switch company:', error);
    } finally {
      setLoading(false);
    }
  };

  return (
    &lt;Select
      value={activeCompany?.id}
      onChange={handleCompanyChange}
      loading={loading}
      style={{ width: 200 }}
      placeholder="Select Company"
    &gt;
      {companies.map(company =&gt; (
        &lt;Select.Option key={company.id} value={company.id}&gt;
          {company.code} - {company.name}
        &lt;/Select.Option&gt;
      ))}
    &lt;/Select&gt;
  );
};

export default CompanySwitcher;
```

7.2 API Service with Company Context

```
// frontend/src/services/api.js
import axios from 'axios';
import store from '../store';

const api = axios.create({
  baseURL: '/api/v1',
  headers: {
    'Content-Type': 'application/json',
  },
});

// Request interceptor - add auth and company context
api.interceptors.request.use(
  config =&gt; {
    const token = localStorage.getItem('token');
    if (token) {
      config.headers.Authorization = `Bearer ${token}`;
    }

    const activeCompany = store.getState().company.activeCompany;
    if (activeCompany) {
      config.headers['X-Company-ID'] = activeCompany.id;
    }
```

```
    return config;
  },
  error =&gt; Promise.reject(error)
);

// Response interceptor - handle errors
api.interceptors.response.use(
  response =&gt; response,
  error =&gt; {
    if (error.response?.status === 401) {
      // Redirect to login
      window.location.href = '/login';
    }
    return Promise.reject(error);
  }
);

export default api;
```

8. Development Environment Setup

8.1 Docker Compose Configuration

```
# docker-compose.yml
version: '3.8'

services:
  db:
    image: postgres:15-alpine
    environment:
      POSTGRES_DB: erp_db
      POSTGRES_USER: postgres
      POSTGRES_PASSWORD: dev_password
    ports:
      - "54322:5432"
    volumes:
      - postgres_data:/var/lib/postgresql/data

  redis:
    image: redis:7-alpine
    ports:
      - "6379:6379"

  backend:
    build: ./backend
    command: python manage.py runserver 0.0.0.0:8000
    volumes:
      - ./backend:/app
    ports:
      - "8000:8000"
    environment:
      - DATABASE_URL=postgresql://postgres:dev_password@db:5432/erp_db
      - REDIS_URL=redis://redis:6379/0
```

```
    depends_on:
      - db
      - redis

  frontend:
    build: ./frontend
    command: npm run dev
    volumes:
      - ./frontend:/app
      - /app/node_modules
    ports:
      - "5173:5173"
    environment:
      - VITE_API_URL=http://localhost:8000/api/v1

  celery:
    build: ./backend
    command: celery -A core worker -l info
    volumes:
      - ./backend:/app
    depends_on:
      - redis
      - db

volumes:
  postgres_data:
```

8.2 Setup Script

```
#!/bin/bash
# scripts/setup_dev.sh

echo "Setting up ERP development environment..."

# Check prerequisites
command -v docker &gt;/dev/null 2&gt;&amp;1 || { echo "Docker required"; exit 1; }
command -v docker-compose &gt;/dev/null 2&gt;&amp;1 || { echo "Docker Compose required";

# Start services
echo "Starting Docker services..."
docker-compose up -d db redis

# Wait for database
echo "Waiting for PostgreSQL..."
sleep 5

# Run migrations
echo "Running database migrations..."
docker-compose run --rm backend python manage.py migrate

# Create superuser
echo "Creating superuser..."
docker-compose run --rm backend python manage.py createsuperuser --noinput \
  --username admin --email admin@example.com || true
```

```
# Load initial data
echo "Loading initial data..."
docker-compose run --rm backend python manage.py loaddata initial_companies.json

# Start all services
echo "Starting all services..."
docker-compose up -d

echo "Setup complete!"
echo "Backend: http://localhost:8000"
echo "Frontend: http://localhost:5173"
echo "Admin: http://localhost:8000/admin"
```

9. Testing Strategy

9.1 Unit Tests Example

```
# backend/apps/companies/tests.py
from django.test import TestCase
from apps.companies.models import Company
from apps.users.models import User

class CompanyModelTest(TestCase):
    def setUp(self):
        self.parent = Company.objects.create(
            code='PARENT',
            name='Parent Company',
            legal_name='Parent Company Ltd.',
            tax_id='TAX001',
            fiscal_year_start='2024-01-01'
        )

    def test_company_creation(self):
        """Test basic company creation"""
        self.assertEqual(self.parent.code, 'PARENT')
        self.assertTrue(self.parent.is_active)

    def test_subsidiary_relationship(self):
        """Test parent-subsidiary relationship"""
        subsidiary = Company.objects.create(
            code='SUB01',
            name='Subsidiary',
            legal_name='Subsidiary Ltd.',
            tax_id='TAX002',
            fiscal_year_start='2024-01-01',
            parent_company=self.parent
        )

        self.assertEqual(subsidiary.parent_company, self.parent)
        self.assertEqual(subsidiary.get_hierarchy_level(), 1)
        self.assertEqual(self.parent.get_hierarchy_level(), 0)
```

9.2 Integration Tests

```
# backend/apps/api_gateway/tests.py
from rest_framework.test import APITestCase
from rest_framework import status
from apps.companies.models import Company
from apps.users.models import User

class CompanyAPITest(APITestCase):
    def setUp(self):
        self.user = User.objects.create_user(
            username='testuser',
            password='testpass123'
        )
        self.client.login(username='testuser', password='testpass123')

    def test_company_list(self):
        """Test retrieving company list"""
        response = self.client.get('/api/v1/companies/')
        self.assertEqual(response.status_code, status.HTTP_200_OK)

    def test_company_isolation(self):
        """Test company data isolation"""
        # Create two companies
        company1 = Company.objects.create(...)
        company2 = Company.objects.create(...)

        # User only has access to company1
        # Verify they can't access company2 data
        # ... test implementation
```

10. Implementation Checklist

Week 1-2: Foundation Setup

- ☐ Initialize Git repository
- ☐ Set up project structure (backend + frontend)
- ☐ Configure Docker development environment
- ☐ Set up embedded PostgreSQL
- ☐ Implement base models (CompanyAwareModel)
- ☐ Create Company model and migrations

Week 3-4: Authentication & Security

- ☐ Extend User model for multi-company
- ☐ Implement Permission and Role models
- ☐ Build authentication API (login, logout, token refresh)

- ☐ Create company context middleware
- ☐ Implement RBAC/ABAC permissions

Week 5-6: API & Communication

- ☐ Set up API gateway
- ☐ Implement event bus (Redis pub/sub)
- ☐ Create API documentation (Swagger/OpenAPI)
- ☐ Build company management API endpoints

Week 7-8: Frontend Foundation

- ☐ Set up React/Vue project with Vite
- ☐ Create layout components
- ☐ Build company switcher component
- ☐ Implement authentication UI
- ☐ Create company management UI

Week 9-10: Testing & Documentation

- ☐ Write unit tests (80% coverage)
- ☐ Write integration tests for APIs
- ☐ Create API documentation
- ☐ Write deployment guide
- ☐ Conduct Phase 1 review

11. Success Criteria

Metric	Target	Validation Method
Code Coverage	≥80%	pytest-cov report
API Response Time	<200ms (95th percentile)	Load testing
Multi-Company Isolation	100% data isolation	Security audit
Authentication	OAuth2 compliant	Security review
Database Performance	<50ms query time	Query profiling

12. Risks & Mitigation

Risk	Mitigation
PostgreSQL embedding complexity	Use Docker for development; documented installation
Multi-company data leakage	Automated RLS tests, manual security audits
Performance with 100 users	Load testing, query optimization, indexing
Team skill gaps	Training sessions, pair programming

13. Next Steps

Upon Phase 1 completion:

1. Conduct technical review with team
2. Perform security audit
3. Document all APIs
4. Begin Phase 2: MVP Business Modules

Document Control:

- **Version:** 1.0
- **Author:** ERP Development Team
- **Date:** October 2025
- **Dependencies:** Phase 0 completion