Django REST API Implementation Guide - Step by Step

🎯 Project Overview

Building a comprehensive Django REST API for a freelance developer portfolio with 7 core domains: Users/Clients, Site Content, Projects, Blog, Services, Products, and Business Operations.

Phase 1: Foundation Setup 🛠️

Step 1: Environment Preparation

- Create Python 3.10+ virtual environment
- Initialize Git repository with README and ERD documentation
- Install core dependencies: Django, DRF, PostgreSQL adapter, authentication, filtering, documentation tools
- Set up `.env` file for environment variables (database credentials, secret keys)

Step 2: Project Structure Creation

- Generate main Django project
- Create 7 specialized apps mapping to ERD domains:
  - `accounts` - Users and client management
  - `core` - Homepage and about content
  - `projects` - Portfolio showcase
  - `blog` - Content publishing
  - `services` - Service offerings
  - `products` - Digital products (separate from services)
  - `business` - Orders, testimonials, communications

Step 3: Settings Configuration

- Configure database connection to PostgreSQL
- Add all apps and DRF to INSTALLED_APPS
- Set custom user model as AUTH_USER_MODEL
- Configure CORS settings for frontend integration
- Set up media and static file handling
- Configure timezone and internationalization

Phase 2: Data Layer Implementation 📊

Step 4: Model Development (App by App)

accounts/models.py

- Custom User model extending AbstractBaseUser with role-based access
- Client model with business information and account balance tracking
- User manager for custom authentication flow

core/models.py

- HeroSection model for dynamic homepage content
- AboutSection model with JSONB field for social media links
- Ensure single-row tables with proper constraints

projects/models.py

- Project model with status enum and client relationships
- Technology model as master reference list
- ProjectGalleryImage for multiple screenshots
- ProjectComment with approval workflow
- Many-to-many through table for project technologies

blog/models.py

- BlogPost model with publish/draft status and view tracking
- Tag model for content categorization
- BlogComment with moderation system
- Many-to-many relationship for post tags

services/models.py

- Service model with flexible pricing models
- ServicePricingTier for package offerings
- ServiceFeature for tier comparisons
- Supporting models: FAQ, ProcessStep, Deliverable, Tool, UseCase

products/models.py (Keep Separate from Services)

- Product model with type enum and licensing
- ProductPurchase for transaction tracking
- ProductReview with rating system
- ProductGalleryImage for product screenshots
- Reuse existing Technology and Tag models through junction tables

business/models.py

- Order model linking clients to services/products
- Testimonial model for client feedback
- Notification system for admin alerts
- ContactMessage for lead capture

Step 5: Database Migration Strategy

- Create initial migrations for each app
- Run migrations in dependency order (accounts first)
- Verify database schema matches ERD
- Create superuser account for admin access

Step 6: Admin Interface Setup

- Register all models in respective admin.py files
- Customize admin display with list_display, search_fields, filters
- Create inline editing for related models (galleries, tiers, etc.)
- Set up admin permissions for different user roles

Phase 3: API Layer Development 🔌

Step 7: Serializer Architecture

Core Serializer Patterns

- Create ModelSerializer for each model
- Implement nested serializers for read operations (include related data)
- Separate serializers for list vs detail views where needed

- Handle sensitive fields (passwords) with write_only configuration

Advanced Serializer Features

- Custom validation methods for business rules
- Method fields for computed properties (full_name, average_rating)
- SerializerMethodField for complex read-only data
- Nested creation/update handling for related objects

Step 8: ViewSet Implementation

Standard ViewSet Structure

- ModelViewSet for full CRUD operations on main entities
- ReadOnlyModelViewSet for reference data (technologies, tags)
- Custom ViewSet methods for specific business logic

Custom Actions and Endpoints

- @action decorators for non-CRUD operations (like/unlike, publish/unpublish)
- Custom permission classes for role-based access
- Bulk operations for admin efficiency
- Custom lookup fields (slug instead of ID for public endpoints)

Step 9: URL Configuration and Routing

- DefaultRouter setup for automatic RESTful URL generation
- Nested routing for related resources (project comments, service tiers)
- Custom URL patterns for non-standard endpoints
- API versioning strategy for future updates

Phase 4: Security and Authentication 🔒

Step 10: Authentication System

- JWT token authentication setup
- Custom user authentication flow
- Token refresh mechanism
- Password reset and email verification workflow

Step 11: Permission Framework

- Role-based permissions (developer, admin)
- Object-level permissions (own content only)
- Public vs authenticated endpoint separation
- Custom permission classes for complex business rules

Step 12: Data Validation and Security

- Input sanitization and validation
- Rate limiting for API endpoints
- CORS configuration for frontend integration
- Secure file upload handling for images

Phase 5: API Enhancement Features 🚀

Step 13: Filtering and Search

- django-filter integration for complex queries
- Search functionality across text fields
- Date range filtering for time-based queries
- Custom filter classes for business-specific filtering

Step 14: Pagination and Performance

- Pagination configuration for large datasets
- Database query optimization with select_related and prefetch_related
- Caching strategy for frequently accessed data
- Database indexing for performance-critical queries

Step 15: File Handling and Media

- Image upload and processing for galleries
- File validation and security checks
- CDN integration for media delivery
- Thumbnail generation for product/project images

Phase 6: Quality Assurance 🧪

Step 16: Testing Strategy

- Unit tests for models (validation, methods)
- Serializer tests (input/output validation)
- API endpoint tests (CRUD operations, permissions)
- Integration tests for complex workflows

Step 17: Test Implementation

- pytest setup with Django integration
- Factory classes for test data generation
- Mock external services and file uploads
- Automated test running in CI/CD pipeline

Step 18: API Documentation

- Swagger/OpenAPI documentation generation
- Endpoint descriptions and examples
- Schema documentation for complex requests
- Authentication documentation for frontend developers

Phase 7: Deployment Preparation 📦

Step 19: Production Configuration

- Environment-specific settings separation
- Database connection pooling
- Static file serving configuration
- Logging and monitoring setup

Step 20: Performance Optimization

- Database query optimization
- API response caching
- Background task setup for heavy operations
- Memory usage optimization

Step 21: Security Hardening

- Production security settings
- HTTPS configuration
- Database security best practices
- API rate limiting and throttling

Phase 8: Integration and Launch 🚀

Step 22: Frontend Integration Points

- CORS configuration for Next.js frontend
- API endpoint documentation for frontend team
- Error handling and response format standardization
- Real-time features setup (WebSocket for chat)

Step 23: Data Migration and Seeding

- Initial data import scripts
- Sample data for development and demo
- Production data migration strategy
- Backup and recovery procedures

Step 24: Monitoring and Maintenance

- API monitoring and alerting
- Performance tracking and optimization
- Regular security updates
- Database maintenance and backup automation

📋 Implementation Checklist

Pre-Development

- [ ] ERD finalized and reviewed
- [ ] Development environment set up
- [ ] Git repository initialized
- [ ] Dependencies installed and configured

Development Phases

- [ ] Phase 1: Foundation (Steps 1-3)
- [ ] Phase 2: Data Layer (Steps 4-6)
- [ ] Phase 3: API Layer (Steps 7-9)
- [ ] Phase 4: Security (Steps 10-12)
- [ ] Phase 5: Enhancement (Steps 13-15)
- [ ] Phase 6: Quality Assurance (Steps 16-18)
- [ ] Phase 7: Deployment Prep (Steps 19-21)
- [ ] Phase 8: Integration (Steps 22-24)

Success Metrics

- [ ] All models created and migrated successfully
- [ ] Complete CRUD API for all entities
- [ ] Authentication and authorization working
- [ ] Admin interface fully functional
- [ ] API documentation complete
- [ ] Test coverage above 80%
- [ ] Performance benchmarks met
- [ ] Security audit passed

🎯 Key Success Factors

Technical Excellence

- Follow Django/DRF best practices consistently
- Maintain clean, documented code
- Implement comprehensive error handling
- Optimize for performance from the start

Business Alignment

- Map every API endpoint to portfolio business needs
- Ensure admin interface supports content management workflow
- Design API structure to support your 30-day transition goals
- Plan for scalability as client base grows

Development Efficiency

- Use consistent patterns across all apps
- Leverage Django admin for rapid prototyping

- Implement automated testing early
- Document decisions and patterns for future reference

This step-by-step approach ensures systematic development while maintaining focus on your portfolio business objectives and technical excellence.