100-Point Django & Python Best Practices Checklist

A comprehensive checklist to verify your Django codebase aligns with industry best practices, security standards, performance optimization, and leverages the full power of Django and Python.

PROJECT STRUCTURE & ARCHITECTURE (15 points)

1. **Modular App Architecture** - Each app has single responsibility, organized by feature/domain
2. **Environment-Specific Settings** - Separate settings files for development, production, testing, staging
3. **Settings from Environment Variables** - Sensitive data in `.env` files, never hardcoded
4. **src/ or config/ Directory Structure** - Clear separation of concerns
5. **Base Models in Core App** - Abstract base models (timestamps, UUIDs) in dedicated `core/models.py`
6. **Custom Managers Organized** - QuerySet and Manager classes in dedicated `managers.py`
7. **Signals in Dedicated File** - Django signals in `signals.py` with `ready()` in `apps.py`
8. **Management Commands Folder** - Custom commands in `management/commands/` directory
9. **Service Layer Pattern** - Business logic separated into `services.py`
10. **Celery Tasks Separated** - Async tasks in dedicated `tasks.py`
11. **Static & Media Folders** - Proper organization of static files separate from code
12. **Tests in Dedicated Folder** - Comprehensive test directory with `conftest.py`
13. **Migrations in Git** - All migrations committed and tracked
14. **Constants Centralized** - Magic numbers/strings in `constants.py` per app
15. **URL Organization** - App-level `urls.py` with namespace routing

SECURITY (18 points)

16. **ALLOWED_HOSTS Configured** - Proper ALLOWED_HOSTS in production (no wildcards)
17. **DEBUG=False in Production** - Never DEBUG=True in production environment
18. **SECURE_SSL_REDIRECT=True** - Force HTTPS in production
19. **SESSION_COOKIE_SECURE=True** - Session cookies only sent over HTTPS
20. **CSRF_COOKIE_SECURE=True** - CSRF tokens only sent over HTTPS
21. **HSTS Headers Set** - SECURE_HSTS_SECONDS, SECURE_HSTS_INCLUDE_SUBDOMAINS, SECURE_HSTS_PRELOAD
22. **Content Security Policy (CSP)** - CSP headers configured with django-csp
23. **X-Frame-Options Header** - Set to DENY or SAMEORIGIN
24. **X-Content-Type-Options Header** - Set to nosniff to prevent MIME sniffing
25. **No SQL Injection** - All queries use Django ORM parameterized queries
26. **No XSS Vulnerabilities** - User input escaped in templates

27. **CSRF Protection Active** - CsrfViewMiddleware enabled on all POST/PUT/DELETE
28. **Input Validation** - All user inputs validated with forms or serializers
29. **Password Validators** - Custom password validators configured
30. **Permission Classes Enforced** - DRF endpoints have explicit permission classes
31. **Rate Limiting/Throttling** - API endpoints have rate limiting
32. **Secret Key Rotation** - SECRET_KEY different across environments
33. **Dependencies Updated** - Regular security updates for Django and packages

DATABASE & ORM OPTIMIZATION (22 points)

34. **N+1 Query Problem Avoided** - Use select_related() for ForeignKey/OneToOneField
35. **Prefetch Related Optimized** - Use prefetch_related() for reverse relationships
36. **Prefetch Objects Used** - Use Prefetch() for complex related data fetching
37. **Query-Specific Field Selection** - Use .values() or .only() for needed fields only
38. **Deferred Field Loading** - Use .defer() to exclude large fields when not needed
39. **Database Indexes Strategic** - Indexes on frequently filtered/sorted fields
40. **Composite Indexes** - Composite indexes for multi-field queries
41. **Database-Level Calculations** - Use F() expressions instead of Python
42. **Aggregation Functions** - Use Count(), Sum(), Avg() in database
43. **Annotations for Complex Queries** - Use .annotate() with Case/When
44. **Bulk Operations** - Use bulk_create() and bulk_update() for batches
45. **Transactions for Data Consistency** - Use @transaction.atomic decorator
46. **Savepoints for Nested Transactions** - Use savepoints for complex scenarios
47. **Query Lazy Evaluation** - Leverage lazy evaluation; chain queries
48. **Connection Pooling** - Use django-db-connection-pool for efficiency
49. **Database Connection Limits** - Set appropriate CONN_MAX_AGE
50. **Raw SQL Minimized** - Use ORM for 95%+ of queries
51. **Database Profiling** - Use Django Debug Toolbar or django-silk
52. **Queryset Caching** - Cache expensive querysets in Redis/Memcached
53. **Foreign Key Choices Optimized** - Use .select_related() with FK choices
54. **Zero-Downtime Migrations** - Migrations designed for no-downtime deployments
55. **Model Normalization** - Tables follow normalization principles (3NF)

MODELS & DATA DESIGN (12 points)

56. **Singular Model Names** - User, not Users; BlogPost, not Blogs
57. **Descriptive Field Names** - created_at, not cr_time
58. **Appropriate Field Types** - CharField for short text, TextField for long content
59. **Choices for Fixed Values** - Use choices= parameter for enums
60. **Blank vs Null Handled** - Explicit blank= and null= (rarely both True)
61. **Default Values Specified** - Sensible defaults where applicable
62. **Meta Options Set** - Meta.ordering, Meta.verbose_name, Meta.constraints
63. **Custom Managers with Queryset** - Reusable query logic in managers
64. **str Method Implemented** - All models have readable __str__() representation
65. **Computed Properties in Code** - Use @property for derived values
66. **Avoid Redundant Fields** - No duplicate data; compute via @property
67. **Model Constraints Defined** - Use Meta.constraints for complex constraints

VIEWS & VIEWSETS (14 points)

68. **Class-Based Views (CBV) Used** - Leverage CBVs (ListView, DetailView, CreateView)
69. **DRF ViewSets for APIs** - Use DRF ViewSets (ModelViewSet, ReadOnlyModelViewSet)
70. **Serializers for Data Validation** - All input data validated via serializers
71. **Nested Serializers for Complex Data** - Proper nested serializers for relations
72. **Pagination Implemented** - API endpoints paginated (never return 10k+ items)
73. **Filtering Supported** - API endpoints support filtering
74. **Ordering/Sorting** - API endpoints support ordering by fields
75. **Search Implemented** - Full-text search on text fields
76. **Permissions & Authentication** - Each endpoint has explicit permission classes
77. **Throttling/Rate Limiting** - Public endpoints have rate limiting
78. **HTTP Status Codes Correct** - Proper 201, 204, 400 status codes
79. **Error Responses Standardized** - Consistent error response format
80. **CORS Configured** - CORS headers properly configured
81. **Response Optimization** - Payloads optimized (only needed fields)

TESTING (10 points)

82. **Unit Tests Written** - Comprehensive unit tests for models, managers, services
83. **Integration Tests** - Tests for API endpoints and database interactions
84. **Factories for Test Data** - Use model factories (factory_boy) not fixtures
85. **Pytest Configuration** - `conftest.py` with reusable pytest fixtures
86. **Test Isolation** - Each test independent; no order dependencies
87. **Coverage Above 80%** - Code coverage for critical paths >80%
88. **CI/CD Integration** - Tests run on every commit via GitHub Actions/GitLab
89. **Async Tests Support** - Async views tested with `@pytest.mark.asyncio`
90. **Mock External APIs** - External calls mocked in tests
91. **Database Transaction Tests** - Transaction behavior tested

⚡ PERFORMANCE & CACHING (10 points)

92. **Caching Strategy Implemented** - Multi-level caching (database, view, template)
93. **Cache Backend Configured** - Redis or Memcached (not default LocMemCache)
94. **Cache Invalidation** - Proper invalidation on data updates with TTL
95. **CDN for Static Files** - Static files via CDN (CloudFront, Cloudflare)
96. **Gzip Compression Enabled** - GZipMiddleware enabled
97. **Database Connection Pooling** - Pool configured for high concurrency
98. **Query Batching** - Related queries batched to minimize round trips
99. **Async Task Queue** - Long-running tasks offloaded to Celery
100. **Performance Monitoring** - New Relic, DataDog, or similar monitoring

CODE QUALITY & STANDARDS (8 bonus points)

101. **PEP 8 Compliant** - Code formatted per PEP 8 (use Black, isort)
102. **Type Hints Used** - Type hints added to functions (Python 3.9+)
103. **Linting Enabled** - flake8, pylint, or prospector in CI/CD
104. **Code Documentation** - Docstrings on modules, classes, public functions
105. **README Updated** - Comprehensive README with setup, deployment, API docs
106. **Comments Meaningful** - Comments explain *why*, not what

107. **DRY Principle** - No code duplication; logic extracted to utilities
108. **Git Hygiene** - Clean commits, meaningful messages, no secrets

DEPLOYMENT & OPERATIONS (8 points)

109. **Production Web Server** - Nginx or Apache (not Django dev server)
110. **WSGI App Server** - Gunicorn, uWSGI (not Django's runserver)
111. **Containerization** - Dockerfile for production deployments
112. **Environment Configuration** - All config via environment variables
113. **Logging Strategy** - Structured logging to centralized service
114. **Health Checks** - Liveness and readiness endpoints
115. **Database Backups** - Regular automated backups with tested restore
116. **Secrets Management** - AWS Secrets Manager or HashiCorp Vault

ADVANCED PYTHON & DJANGO FEATURES (16 points)

117. **Decorators Utilized** - Custom decorators for cross-cutting concerns
118. **Context Managers** - Context managers for resource management
119. **Middleware Custom** - Custom middleware for auth, logging, errors
120. **Signals Properly Used** - Django signals for cross-app communication
121. **Async Views & Middleware** - Django 4.0+ async views for concurrency
122. **AsyncClient for Testing** - Django 5.0+ AsyncClient for async tests
123. **Generators for Large Data** - Generators for streaming (no full loads)
124. **Context Processors** - Custom context processors for template variables
125. **Template Tags & Filters** - Custom tags/filters for reusable logic
126. **Form Widgets Customized** - Custom widgets for complex fields
127. **Model Inheritance Strategy** - Abstract, multi-table, and proxy models
128. **Middleware Hooks Optimized** - Minimal middleware; remove non-essential
129. **Settings as Module or Object** - Use django-configurations or django-environ
130. **Lazy Loading Querysets** - Querysets evaluated lazily
131. **Python Metaclasses** - Metaclasses for model customization
132. **Functional Programming** - Map, filter, reduce for transformations

SECURITY & COMPLIANCE (10 bonus points)

133. **GDPR Compliance** - Data export, deletion, privacy mechanisms
134. **Data Encryption** - Sensitive data encrypted at rest
135. **API Rate Limiting** - Per-user/IP rate limits to prevent DDoS
136. **Input Sanitization** - HTML/SQL/Command injection prevention
137. **File Upload Validation** - File type, size, and content validation
138. **Dependency Scanning** - Regular scans via Dependabot or safety.io
139. **Security Headers** - All recommended security headers configured
140. **Audit Logging** - Critical actions logged (login, data changes)
141. **Incident Response Plan** - Documented incident response procedures
142. **Regular Security Review** - Quarterly code reviews and pen testing

OPTIMIZATION & SCALABILITY (10 points)

143. **Horizontal Scaling** - Application designed for horizontal scaling
144. **Load Balancing** - Multiple instances behind load balancer
145. **Database Sharding** - Sharding strategy for massive datasets
146. **Read Replicas** - Database read replicas for read-heavy workloads
147. **Search Engine Integration** - Elasticsearch for full-text search
148. **Message Queue Scaling** - Message queues for job distribution
149. **Microservices Ready** - API design allows microservices decomposition
150. **API Versioning** - API versioned for backward compatibility

---

Scoring Guide

Production Readiness Scale

Score	Status	Action
90-150 points	✅ Production-ready	Deploy with confidence
70-89 points	⚠ Good quality	Address critical gaps
50-69 points	⚠ Work needed	Major refactoring required
<50 points	❌ Severe issues	Complete overhaul needed

Implementation Priority

Phase 1: Critical (Security & Performance)

Points 16-33 (Security)

- Prevents 90% of common exploits
- HTTPS, CSRF, CORS, authentication, input validation

Points 34-55 (Database Optimization)

- 10-100x query performance improvement
- Select_related, prefetch_related, N+1 prevention

Points 92-100 (Performance & Caching)

- 50-80% reduction in response time
- Redis/Memcached, CDN, monitoring

Phase 2: Important (Quality & Testing)

Points 82-91 (Testing)

- 60% fewer bugs reaching production
- Unit tests, integration tests, CI/CD

Points 101-108 (Code Quality)

- 70% faster onboarding for new developers
- PEP 8, type hints, linting, documentation

Phase 3: Enhancement (Advanced)

Points 117-132 (Advanced Python/Django)

- Better code organization, maintainability
- Async views, custom decorators, advanced patterns

Points 133-150 (Advanced Optimization)

- Handle 10x traffic growth
- Horizontal scaling, microservices readiness

Key Metrics by Category

Security Readiness

Category	Target	Current
HTTPS & Headers	10/10	_
Authentication	5/5	_
Input Validation	3/3	_

Performance Readiness

Category	Target	Current
Query Optimization	22/22	_
Caching	10/10	_
Async Operations	3/3	_

Code Quality

Category	Target	Current
Testing	10/10	_
Standards	8/8	_
Documentation	5/5	_

Quick Reference by Focus Area

For Performance-Critical Applications

Priority Points: 34-55, 92-100, 143-150

- N+1 query optimization
- Caching strategy
- Horizontal scaling

For Security-Critical Applications

Priority Points: 16-33, 133-142

- HTTPS and headers
- Input validation and sanitization
- Audit logging

For Large Teams

Priority Points: 101-108, 1-15, 82-91

- Code standards and documentation
- Project structure
- Testing coverage

For MVP/Startup

Priority Points: 16-20, 34-44, 68-77, 82-85

- Basic security
- Core performance
- API structure
- Basic testing

Last Updated

January 2026 | Django 5.x | Python 3.10+