Production-Ready Python Backend Checklist (FastAPI + Azure SDK)

FOR EACH ITEM IN CHECKLIST BELOW, PLEASE PROVIDE EVIDENCES TO CONFIRM THAT IT IS PRODUCTION READY!

---

1. Code Style & Formatting

- [ ] Code follows **PEP 8**, **PEP 257**, and **PEP 20**.
- [ ] Use **4 spaces** for indentation (no tabs).
- [ ] All lines ≤ 79 characters (≤72 for comments/doc).
- [ ] No code file exceeds **170 lines**.
- [ ] Only one top-level concept (class/function) per file.
- [ ] Module, class, function, variable, and constant naming follows convention.
- [ ] Imports grouped: stdlib → third-party → local, each separated by blank lines.
- [ ] Use **Black** for formatting and **isort** for import sorting.
- [ ] Use **Flake8**, **Pylint**, or **Ruff** for linting.
- [ ] **Pre-commit hooks** are set up and enforced locally.

---

2. Project Structure & Organization

Routing Breakdown

- [ ] Each **domain or feature** (e.g. `users`, `auth`, `documents`) has its own subfolder with a dedicated `router.py` or `routes.py` file.
- [ ] Routers are split into **manageable files** (≤170 lines per file).
- [ ] Route handlers are grouped by responsibility (e.g., `user_crud.py`, `user_auth.py`).
- [ ] Each router file only defines related endpoints and logic—no unrelated side utilities or business logic.
- [ ] Main app mounts all routers cleanly in `main.py` (or `app.py`) using `include_router()`.
- [ ] Configuration uses **Pydantic BaseSettings**.
- [ ] No secrets/configs hardcoded — use Azure Key Vault or any secret manager.
- [ ] Every router is defined using `APIRouter()` with a consistent `prefix` and `tags` for OpenAPI grouping.
- [ ] Route parameters and request models are fully typed with Pydantic.
- [ ] No business logic in route files — delegated to service layer (e.g., `services/user_service.py`).

---

3. Testing, CI/CD & Coverage

- [ ] All business logic and APIs are **unit or integration tested**.

- [ ] Tests located under `tests/`, mirroring app structure.
- [ ] Uses **pytest** and FastAPI's `TestClient`.
- [ ] Test coverage ≥ 80% (critical paths).
- [ ] `pytest-cov` or `coverage.py` integrated.
- [ ] CI/CD pipeline runs on push/PR:
    - [ ] `black` check
    - [ ] `isort` check
    - [ ] Linting (e.g., `flake8`, `ruff`)
    - [ ] Tests (via `pytest`)
    - [ ] Type checking (e.g., `mypy`, `pyright`)
    - [ ] Security scan (`bandit`, `safety`)
- [ ] Builds fail on any test, lint, or type errors.

---

4. Security Best Practices

General Security

- [ ] API served over **HTTPS only** - could use Istio in K8s to upgrade connection or APIM.
- [ ] **OAuth2 + JWT** or **Azure AD** or any equivalent method for authentication.
- [ ] **Pydantic** models used for input validation (use model_validator/field_validator).
- [ ] No stack traces or internal errors exposed to users.
- [ ] No hardcoded credentials or tokens in code.

Secrets & Azure Security

- [ ] Azure **Managed Identity** used instead of static credentials. Another choice is Key Vault but need to set carefully with secret injection.
- [ ] Azure Key Vault or secret manager used for all secrets/configs.
- [ ] Minimal Azure RBAC permissions granted (least privilege principle).
- [ ] All access tokens/keys rotate periodically.

Logging & Monitoring

- [ ] Use **structured logging** (e.g., `structlog`).
- [ ] Logs contain at least `timestamp`, `trace_id`, `log_level`.
- [ ] Logs **do not** include:
    - [ ] Auth tokens
    - [ ] PII (e.g., SSN, Tax ID)
    - [ ] Any secret/sentitive data
- [ ] Audit logs are implemented for sensitive actions.
- [ ] Integrated with Azure Monitor, Log Analytics or App Insights.

Web/API Protections

- [ ] CORS enabled **only** for trusted domains.
- [ ] Security headers set (CSP, HSTS, X-Content-Type).
- [ ] All external input is sanitized.
- [ ] OWASP Top 10 and CWE risks reviewed periodically.
- [ ] Static security scan tools with SBOM analysis used (e.g., Bandit, Snyk, etc.).

---

5. FastAPI & Azure SDK Practices

FastAPI

- [ ] API routes are split using `APIRouter`.
- [ ] Use `Depends()` (Dependency injection) for:
    - [ ] DB sessions
    - [ ] Auth
    - [ ] Config
- [ ] All request/response bodies use **Pydantic** models (DTO object).
- [ ] Prefer `async` I/O libraries where available (e.g. `httpx`, `asyncpg`, `aioboto3`).
- [ ] If a blocking call must be used, wrap it in `run_in_threadpool()` to avoid blocking the event loop.

Docs & Models

- [ ] Auto-generated docs via OpenAPI (`/docs`) are clear and usable.
- [ ] All models use `Field(..., description="...")` so it is descriptive.
- [ ] Each public route/class/function has a docstring.

Azure SDK

- [ ] Use `DefaultAzureCredential` for authentication when you can. Else can use Azure Key Vault alternative.
- [ ] Use official Azure SDKs (e.g., BlobServiceClient, SecretClient).
- [ ] Library versions pinned in `requirements.txt`.
- [ ] Environment-specific settings configured via Azure App Settings or Key Vault.

Certainly! Here's the updated **Section 6: Clean Code Principles** with all the expanded checklist items merged under a structured heading.

---

6. Clean Code Principles

General Guidelines

- [ ] Code follows core principles: **DRY**, **KISS**, **YAGNI**, **SOLID**.
- [ ] Functions ≤ 50 lines.
- [ ] Classes ≤ 100 lines.
- [ ] Files ≤ 170 lines.
- [ ] Avoid duplicated logic — extract common code to utilities/services.
- [ ] Avoid deeply nested logic (max 2–3 levels).
- [ ] Use descriptive and self-explanatory names for functions and variables.

- [ ] Use named constants or enums — no magic numbers or strings.
- [ ] Keep one level of abstraction per function (don’t mix DB, business logic, formatting).
- [ ] Use early returns to reduce nesting and improve readability.
- [ ] Remove dead code, TODOs, and unused imports.

DRY — Don’t Repeat Yourself

- [ ] Shared logic is factored into reusable services or utility modules.
- [ ] Validation patterns are reused via decorators, base models, or validators.
- [ ] Avoid copy-paste coding — reuse config loaders, loggers, database connectors, etc.

KISS — Keep It Simple, Stupid

- [ ] Prefer readability over clever hacks or overly abstract solutions.
- [ ] Use built-in language features and libraries over custom code where possible.
- [ ] Keep conditionals and loops concise and readable.
- [ ] Avoid unnecessary indirection or complex abstractions.

YAGNI — You Aren’t Gonna Need It

- [ ] Only build what is needed now — no speculative features.
- [ ] No premature abstraction or configurability.
- [ ] Clean up outdated flags, toggles, and commented code from deprecated features.

SOLID Principles

- [ ] **S — Single Responsibility:** Every class/module has one clear job.
- [ ] **O — Open/Closed:** Modules can be extended without changing existing logic.
- [ ] **L — Liskov Substitution:** Subtypes behave correctly when used as their parent types.
- [ ] **I — Interface Segregation:** Interfaces are small and focused (no "god interfaces").
- [ ] **D — Dependency Inversion:** Code depends on abstractions (e.g. inject services, use interfaces).

Error Handling

- [ ] Only known exceptions are caught — should not exist `except:` without exception type.
- [ ] Custom exception classes are defined where needed (e.g. `InvalidDocumentException`).
- [ ] HTTP status codes are used appropriately:
  - [ ] `400` – Bad Request
  - [ ] `401` – Unauthorized
  - [ ] `403` – Forbidden
  - [ ] `404` – Not Found
  - [ ] `500` – Internal Server Error
- [ ] Error messages do not expose internal logic, stack traces, or sensitive info.

---

7. Documentation

- [ ] `README.md` includes setup, structure, and run instructions.
- [ ] `CONTRIBUTING.md` defines code style, tests, PR rules.
- [ ] `docs/` directory exists with:
  - [ ] Architecture Overview
  - [ ] API References
  - [ ] Deployment Flow
- [ ] All Pydantic models have field descriptions and examples.
- [ ] Optional: use `mkdocs`, `Sphinx`, or Swagger export for versioned docs.

---

8. Code Review Checklist

- [ ] File length ≤ 170 LOC.
- [ ] All functions and methods typed.
- [ ] Code is free of TODOs/print/debug/logging leaks.
- [ ] See [Section 4: Logging & Monitoring] for log safety and secret handling policies.
- [ ] PR includes tests and updates coverage.
- [ ] Code is clean, clear, and Pythonic.
- [ ] CI checks pass (lint, type, test).
- [ ] Dependencies updated in `requirements.txt` if needed.
- [ ] Docstrings and docs updated where applicable.