🌓 第六学习指南：一致性修复与工程化加固（从“能跑”到“可演进”）

> **目标读者**：已完成前五份学习指南，准备把项目提升到“可维护、可协作、可演进”水平的开发者
> **项目**：Todo API（Python + FastAPI + SQLAlchemy + JWT + Alembic）
> **最后更新**：2026-02-15

目录

一、为什么要做这轮修复

前五份指南已经帮你完成了：

- 基础架构（分层、CRUD、认证）
- 开发流程（Git、测试、Docker）
- 扩展能力（新字段、用户系统）

但一个项目从“学习可用”走向“工程可用”，关键在三件事：

1. **契约一致**：文档必须和真实行为一致。
2. **可演进**：数据库变更必须可追踪、可回滚。
3. **可防退化**：高风险行为必须有自动化测试和 CI 兜底。

本指南就是围绕这三件事展开的加固。

二、七个问题与逐项修复

1. 文档与真实接口不一致

问题

- README 只写了 `/todos/*`，没写认证接口。
- 真实代码里 `/todos/*` 全部需要 JWT。

风险

- 新人按 README 调接口会直接失败（401）。
- 文档失信，团队沟通成本上升。

修复

- 重写 `README.md`：
  - 明确 `/auth/register`、`/auth/login`。
  - 明确 Todo 接口需要 `Authorization: Bearer <token>`。
  - 补充迁移、Docker、依赖拆分、CI 说明。

核心文件

- `README.md`
- `src/app/routers/todo.py`
- `src/app/routers/auth.py`

---

2. 数据库演进能力不足（create_all）

问题

- 启动时直接 `create_all`，没有迁移体系。

风险

- 一旦表结构升级（加列、改列、索引调整），线上变更不可控。
- 无法审计“谁在什么时候改了什么结构”。

修复

- 引入 Alembic：
  - `alembic.ini`
  - `migrations/env.py`
  - `migrations/versions/20260215_01_initial_schema.py`
- 应用启动不再自动建表，改由迁移驱动。
- Docker 启动时先执行 `alembic upgrade head`。

核心文件

- `src/app/main.py`
- `alembic.ini`
- `migrations/env.py`
- `migrations/versions/20260215_01_initial_schema.py`
- `Dockerfile`

---

3. 并发注册下可能出现 500（竞态条件）

问题

- 先查重再插入有 TOCTOU（检查与写入之间有窗口）。

风险

- 高并发时两个请求都通过查重，提交时触发唯一约束异常，可能出现 500。

修复

- 保留业务层“预检查”（更友好的常规提示）。
- 在 `commit` 处捕获 `IntegrityError`，回滚后稳定返回 409。

核心文件

- `src/app/routers/auth.py`

工程原则

- **数据库唯一约束才是最终防线**，应用层检查只是“优化提示”。

---

4. 分页缺少稳定排序与边界控制

问题

- `skip/limit` 无范围约束。
- 查询未排序，分页结果可能不稳定。

风险

- 客户端传入异常参数导致资源浪费。
- 分页结果在并发写入时抖动，前端体验不稳定。

修复

- 使用 `Query` 增加边界：
  - `skip >= 0`
  - `1 <= limit <= 100`
- 增加稳定排序：
  - `order_by(created_at desc, id desc)`

核心文件

- `src/app/routers/todo.py`

---

5. 配置写法技术债（Pydantic v2 弃用）

问题

- 使用 `class Config`，已被 Pydantic v2 标记弃用。

风险

- 当前可用但未来升级风险大，测试持续出现 deprecation warning。

修复

- 改为 `SettingsConfigDict` + `model_config`。

核心文件

- `src/app/config.py`

6. 测试缺少“权限隔离”核心场景

问题

- 有未登录 401 测试，但缺 A/B 用户越权测试。

风险

- 权限边界未来可能被改坏却不自知（高危回归）。

修复

- 新增隔离测试：
  - 用户 B 不能读取用户 A 的 Todo（404）
  - 用户 B 不能更新用户 A 的 Todo（404）
  - 用户 B 不能删除用户 A 的 Todo（404）
  - 列表仅返回当前用户数据
- 补充分页参数非法值测试（422）。

核心文件

- `tests/test_todo.py`

---

7. 工程化不完整（依赖、CI、许可证）

问题

- `requirements.txt` 混合运行时与测试/传递依赖。
- 缺 CI 流水线。
- README 声明 MIT 但无 LICENSE 文件。

修复

- 依赖分层：
  - `requirements.txt`：最小运行时依赖
  - `requirements-dev.txt`：开发/测试依赖（包含 pytest/httpx）
- 新增 CI：`.github/workflows/ci.yml`
- 新增 `LICENSE`（MIT）

核心文件

- `requirements.txt`
- `requirements-dev.txt`
- `.github/workflows/ci.yml`
- `LICENSE`

三、修复后的工程评价

1. 完整度变化（主观工程评估）

维度	修复前	修复后	变化点
文档一致性	中	高	README 与真实接口、鉴权规则对齐
数据库演进	低	中高	从 `create_all` 迁移到 Alembic
并发安全	中	中高	注册竞态有异常兜底
API 稳定性	中	中高	分页边界 + 稳定排序
测试可信度	中	中高	增加权限隔离关键路径
工程协作	中	高	依赖分层 + CI + LICENSE

2. 当前仍可继续改进的方向

1. 增加 Alembic 的“变更模板规范”（命名、回滚策略、发布流程）。
2. 补充服务层（service layer），进一步降低 router 复杂度。
3. 增加 API 限流、审计日志、错误码规范化（统一 error schema）。
4. 将 CI 扩展为“lint + test + migration check”。

四、你应该形成的工程思维

1. “能跑”不是完成，“可演进”才是完成

- 没有迁移体系的数据库项目，不可持续。
- 没有回归测试的权限系统，不可靠。

2. 应用层检查不是安全边界，数据库约束才是

- 先查重再插入永远可能被并发打穿。
- 唯一约束 + 异常捕获是标准组合。

3. 文档是接口的一部分

- README 不是宣传页，是可执行协作协议。
- 文档不准，团队效率会持续损耗。

4. 把“规范”固化成自动化

- 依赖分层 + CI + 测试 = 项目自我保护机制。
- 手工约定会被遗忘，自动化规则不会。

五、自测清单

- [ ] 我能解释为什么 `create_all` 不能替代迁移体系吗？
- [ ] 我能说清楚“预检查 + 唯一约束兜底”这套并发防护吗？
- [ ] 我知道为什么分页必须有稳定排序吗？
- [ ] 我能写出“用户 A 不能访问用户 B 资源”的回归测试吗？
- [ ] 我能区分运行时依赖和开发依赖吗？

- [ ] 我知道 CI 最小闭环应该至少包含哪些步骤吗？

> **结论**： 这轮修复不是“补丁式改代码”，而是把项目从“学习级可运行”推进到“工程级可维护”。 当你开始主动处理一致性、演进性、并发性和自动化时，你已经在用专业工程师的方式写代码了。