**🔒 Этап 1: Рефакторинг и улучшение безопасности**

На этом этапе мы сделаем код более структурированным и безопасным. Основные задачи:

* Перенести токен в переменные окружения для защиты от утечек.
* Структурировать проект по модулям для удобства поддержки и добавления новых функций.

**1. Перенос токена в переменные окружения**

**📌 Что и зачем?**

* Токен бота сейчас хранится в коде, что небезопасно.
* При публикации кода в GitHub токен может быть случайно раскрыт.
* Перенос токена в переменные окружения защитит его.

**✅ Шаги:**

1. **Изменить код:**  
   Замените строку с токеном на:
2. import os
3. TOKEN = os.getenv("TOKEN")
4. **Добавить переменную в Railway:**
   * Зайдите в проект на Railway.
   * Перейдите в **Settings → Variables**.
   * Добавьте переменную:
     + **Key:** TOKEN
     + **Value:** ваш токен от BotFather
5. **Проверить работоспособность:**
   * Задеплойте изменения на Railway.
   * Убедитесь, что бот работает корректно и принимает команды.

**2. Структуризация кода по модулям**

**📌 Что и зачем?**

* Сейчас весь код находится в одном файле.
* Разделение на модули улучшит читаемость, тестируемость и возможность расширения.
* Это также упростит добавление новых функций в будущем.

**✅ Шаги:**

1. **Создать структуру проекта:**
2. my\_bot/
3. ├── main.py # Основной файл запуска бота
4. ├── config.py # Конфигурация (например, токен)
5. ├── handlers/
6. │ ├── \_\_init\_\_.py
7. │ ├── tasks.py # Хендлеры для задач
8. │ └── progress.py # Хендлеры для прогресса
9. └── utils/
10. ├── \_\_init\_\_.py
11. └── storage.py # Работа с данными (сохранение и загрузка)
12. **Разделить код на файлы:**
    * **main.py:** запускает бота, подключает хендлеры.
    * **config.py:** хранит настройку токена.
    * **handlers/tasks.py:** обработка задач (создание, завершение).
    * **handlers/progress.py:** обработка команд для показа прогресса.
    * **utils/storage.py:** функции для загрузки и сохранения данных.
13. **Пример кода:**

* **config.py:**
* import os
* TOKEN = os.getenv("TOKEN")
* **main.py:**
* from aiogram import Bot, Dispatcher
* from config import TOKEN
* from handlers import tasks, progress
* import asyncio
* bot = Bot(token=TOKEN)
* dp = Dispatcher()
* tasks.register\_handlers(dp)
* progress.register\_handlers(dp)
* async def main():
* await bot.delete\_webhook(drop\_pending\_updates=True)
* await dp.start\_polling(bot)
* if \_\_name\_\_ == "\_\_main\_\_":
* asyncio.run(main())
* **handlers/tasks.py:**
* from aiogram import types, Router
* task\_router = Router()
* @task\_router.message(commands=["add\_task"])
* async def add\_task(message: types.Message):
* await message.answer("Введите задание:")
* def register\_handlers(dp):
* dp.include\_router(task\_router)

1. **Обновить импорты:**
   * Убедитесь, что во всех файлах импорты корректны с учётом новой структуры.

**3. Проверка работоспособности и деплой**

1. **Тестирование локально:**
   * Запустите бота локально, чтобы убедиться, что всё работает корректно.
   * Проверьте, что бот принимает команды и сохраняет данные.
2. **Коммит изменений:**
3. git add .
4. git commit -m "Refactor: структура проекта и переменные окружения"
5. git push
6. **Деплой на Railway:**
   * Railway автоматически обнаружит изменения и задеплоит новую версию.
   * Проверьте логи в Railway, чтобы убедиться в отсутствии ошибок.

**4. Документация изменений**

1. **Обновить README.md:**
   * Объясните новую структуру проекта.
   * Опишите, как добавить новые хендлеры или изменить настройки.
2. **Добавить комментарии в коде:**
   * Добавьте понятные комментарии к ключевым частям кода, особенно в местах, связанных с новыми модулями и конфигурацией.

**🏁 Итог:**

После выполнения всех шагов бот станет более безопасным (благодаря использованию переменных окружения) и структурированным (разделение кода на модули). Это упростит добавление новых функций на следующих этапах. Если нужна помощь с реализацией — дайте знать! 🚀