Урок 3. Защита, CI и бэкапы: доводим planner до релиза v1.0.0

В первых двух уроках мы подняли FastAPI-приложение, подключили базу данных и выкатили его на VPS за nginx. Сегодня — финальный аккорд: делаем сервис защищённым, настраиваем автотесты и учимся не терять данные. К концу урока у проекта появится стабильная версия с тегом `v1.0.0`.

Что было до этого

Кратко напомню, с чего начинали:

- приложение `planner-app` работает через systemd и доступно на `https://work-list.ru`;
- база SQLite (`planner.db`) хранит пользователей и задачи;
- вход в систему — по JWT-токенам.

Шаг 1. Ограничиваем частоту запросов (rate limiting)

Первое, что делает злоумышленник с открытым API, — перебирает пароли. Чтобы сделать это бессмысленно, добавили точечные лимиты через библиотеку `slowapi`:

- `/token` — не больше 10 запросов в минуту с одного IP;
- `/register` — не больше 5 запросов в минуту.

Глобальный лимит на всё приложение мы сознательно не ставим: он мешал бы легитимным пользователям, которым нужно просто быстро поработать со списком задач.

Подводный камень, о котором стоит знать. В slowapi декоратор лимита требует, чтобы у функции-обработчика был явный параметр `request: Request`. Без него код запускается, но

при обращении к эндпоинту падает — и найти причину без логов непросто. Этот баг мы поймали именно через тесты.

Шаг 2. Сужаем CORS до своего домена

По умолчанию в тестовой конфигурации разрешены запросы отовсюду (`*`). Для продакшена это плохая практика: любой сайт сможет слать запросы к вашему API от имени пользователя. Мы ограничили список разрешённых источников одним доменом — `work-list.ru`.

Шаг 3. Секреты — в окружение

`SECRET_KEY`, которым подписываются JWT-токены, теперь живёт в файле `.env` на сервере, а не в коде. Репозиторий публичный, и ключ в нём — это открытая дверь в чужие аккаунты. Заодно проверили, что `.env` попал в `.gitignore`.

Шаг 4. Настраиваем CI на GitHub Actions

Теперь каждый пуш в `main` автоматически прогоняет `pytest`. Воркфлоу элементарный: поднимаем Python, ставим зависимости из `requirements.txt`, запускаем тесты.

Здесь нас ждал самый поучительный момент вечера — **конфликт версий bcrypt**. Первые прогоны падали, потому что свежая версия bcrypt несовместима с той, что использует `passlib`. Решение — «расслабить» версию в `requirements.txt`, чтобы CI брал совместимую. Итог: тесты зелёные, зависимости зафиксированы.

Как читать ленту прогонов. Красные крестики напротив старых коммитов — это не открытые проблемы, а история. GitHub никогда не перезапускает прошлые прогоны: раз упал — так и останется красным. Смотреть нужно только на самый верхний запуск: если он зелёный, текущее состояние кода здорово.

Шаг 5. Чистим базу и фиксируем релиз

Перед релизом удалили тестового пользователя из базы — напрямую через Python, поскольку `sqlite3` в системе не установлен:

bash

```
python -c "import sqlite3; c = sqlite3.connect('planner.db'); c.execute(\"DELETE FROM users WHERE username='test'\"); c.commit(); c.close()"
```

А затем поставили тег:

bash

```
git tag v1.0.0
git push origin v1.0.0
```

Теперь у проекта есть вечная точка возврата: в любой момент можно откатиться ровно на эту версию.

Шаг 6. Автоматический бэкап базы

Самая драматичная часть урока. Задача простая: копировать `planner.db` каждую ночь в 03:00. Но путь к рабочему решению оказался извилистым.

Урок первый: расписание — не команда. Строка вида `0 3 * * * cp ...` вызывает в обычном терминале ошибку `syntax error near unexpected token '('`. Bash не понимает синтаксис cron — эти строки живут только внутри `crontab -e`. И наоборот: чтобы посмотреть расписание, используют `crontab -l`, а не `cat crontab` (файл лежит в системной папке, а не в текущей директории).

Урок второй: экранирование процента. В crontab символ `%` имеет особое значение, поэтому его экранируют: `date +\%F`. В обычном терминале наоборот — слэш не нужен.

Урок третий: `$()` не всегда лучше кавычек. Конструкция `$(date +%F)` упорно отказывалась работать, а старый синтаксис сработал с первого раза:

bash

```
cp /var/www/planner-app/planner.db /backups/planner_`date +%F`.db
```

Итоговая строка в crontab выглядит так:

text

```
0 3 * * * cp /var/www/planner-app/planner.db /backups/planner_`date +\%F`.db
```

Проверить руками, не дожидаясь ночи, проще всего простой командой без расписания — если файл появился в `/backups/`, механика работает, а cron будет повторять её автоматически.

Итоги урока

- Точечный rate limiting на критичных эндпоинтах вместо глобального.
- CORS ограничен своим доменом, секреты вынесены из кода.
- CI гоняет тесты на каждый пуш, зависимости зафиксированы.
- Релиз зафиксирован тегом `v1.0.0`.
- База данных бэкапится каждую ночь автоматически.

Что дальше

В следующем уроке напрашиваются две темы: автоматический деплой из GitHub Actions по SSH (чтобы код выкатывался сам после пуша) и ротация старых бэкапов, чтобы диск не забивался за полгода. А пока — выдохнуть и порадоваться зелёной галочке в CI.

Revision #1
Created 2026-10-08 21:13:19 UTC by im
Updated 2026-10-08 21:13:41 UTC by im