Санкт-Петербургский Национальный Исследовательский
Университет ИТМО

Факультет Инфокоммуникационных технологий

# Лабораторная работа №2. Интеграционное тестирование

**Автор:** Титов Г.К. (409687)

**Проверил:** Кочубеев Н.С**.** (307482)

**Цель работы:** закрепить навыки интеграционного тестирования: научиться выявлять места взаимодействия между компонентами системы, писать тесты для проверки таких взаимодействий и анализировать результаты.

Санкт-Петербург

2025 год

## СОДЕРЖАНИЕ

## Выбор проекта

Для выполнения лабораторной работы был выбран открытый проект на GitHub - Flask-приложение с REST API и хранилищем данных. Проект представляет собой простое веб-приложение, предоставляющее API для управления сущностями (записями), а также включает инфраструктуру для запуска, мониторинга и тестирования. Приложение включает следующие структурные модули:

- **app.py** — точка входа и конфигурация приложения
- **routes.py** — содержит конечные точки (endpoint-ы), которые обрабатывают HTTP-запросы.
- **services.py** — бизнес-логика (Содержит логику обработки данных.)
- **tests/** — интеграционные и модульные тесты
- **requirements.txt** — список зависимостей

## Запуск приложения

Для работы с выбранным проектом необходимо выполнить подготовку окружения и запустить Flask-приложение. Ниже приведена подробная последовательность шагов.

1. Клонирование репозитория.

   Клонировать можете удобным для себя способом. Я воспользовался готовыми инструментами Visual Studio Code.

2. Создание виртуального окружения.

   Для изоляции зависимостей рекомендуется создать отдельное виртуальное окружение:

   ```
   python3 -m venv venv
   source venv/bin/activate
   ```

3. Установка зависимостей.

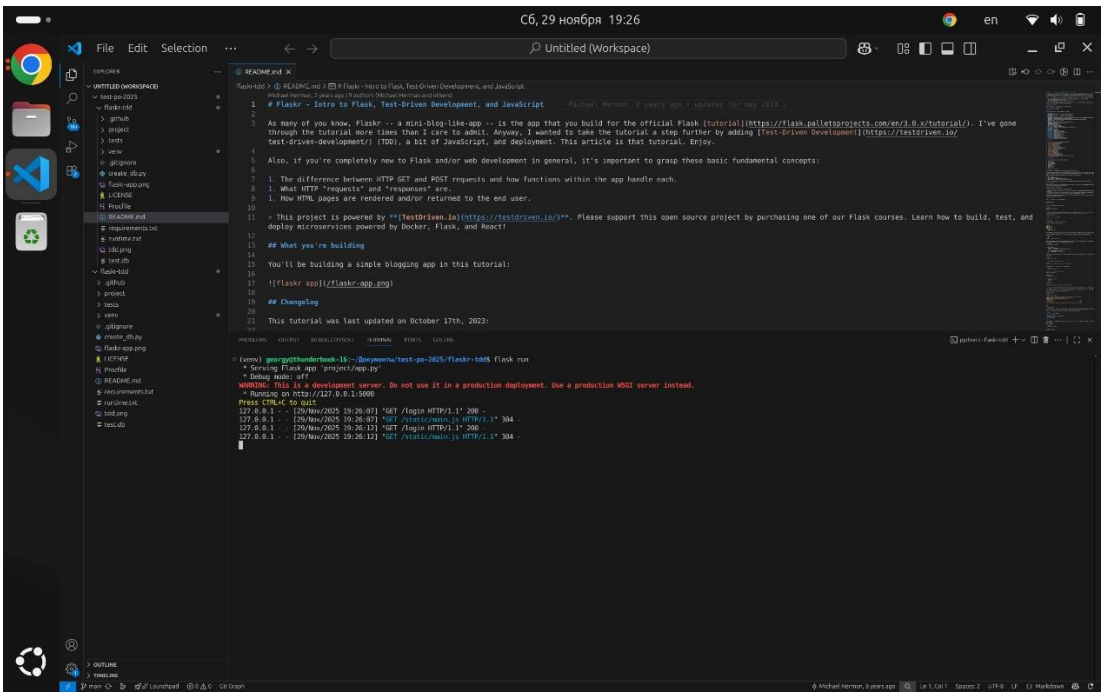
   Выполнить установку пакетов, необходимых для работы приложения:

   ```
   pip install -r requirements.txt
   ```

4. Запуск приложения.

   ```
   python app.py
   flask run
   ```

Если вы видите в консоли следующий вывод (см. скриншот), то приложение успешно запущено и все зависимости для работы установлены:


Сб, 29 ноября 19:26
en
File Edit Selection
Untitled (Workspace)
README.md
# Flaskr - Intro to Flask, Test-Driven Development, and JavaScript
As many of you know, Flaskr -- a mini-blog-like-app -- is the app that you build for the official Flask [tutorial](https://flask.palletsprojects.com/en/3.0.x/tutorial/). I've gone through the tutorial more times than I care to admit. Anyway, I wanted to take the tutorial a step further by adding [Test-Driven Development](https://testdriven.io/test-drives-development/) (TDD), a bit of JavaScript, and deployment. This article is that tutorial. Enjoy.
Also, if you're completely new to Flask and/or web development in general, it's important to grasp these basic fundamental concepts:
1. The difference between HTTP GET and POST requests and how functions within the app handle each.
1. What HTTP "requests" and "responses" are.
1. How HTML pages are rendered and/or returned to the end user.
> This project is powered by **[TestDriven.io](https://testdriven.io/)**. Please support this open source project by purchasing one of our Flask courses. Learn how to build, test, and deploy microservices powered by Docker, Flask, and React!
## What you're building
You'll be building a simple blogging app in this tutorial:
## Changelog
This tutorial was last updated on October 17th, 2023:
(venv) georgy@thunderbook-16:~/Документы/test-po-2025/flaskr-tdd$ flask run
* Serving Flask app 'project/app.py'
* Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
* Running on http://127.0.0.1:5000
Press CTRL+C to quit
127.0.0.1 - - [29/Nov/2025 19:26:07] "GET /login HTTP/1.1" 200 -
127.0.0.1 - - [29/Nov/2025 19:26:07] "GET /static/main.js HTTP/1.1" 304 -
127.0.0.1 - - [29/Nov/2025 19:26:12] "GET /login HTTP/1.1" 200 -
127.0.0.1 - - [29/Nov/2025 19:26:12] "GET /static/main.js HTTP/1.1" 304 -

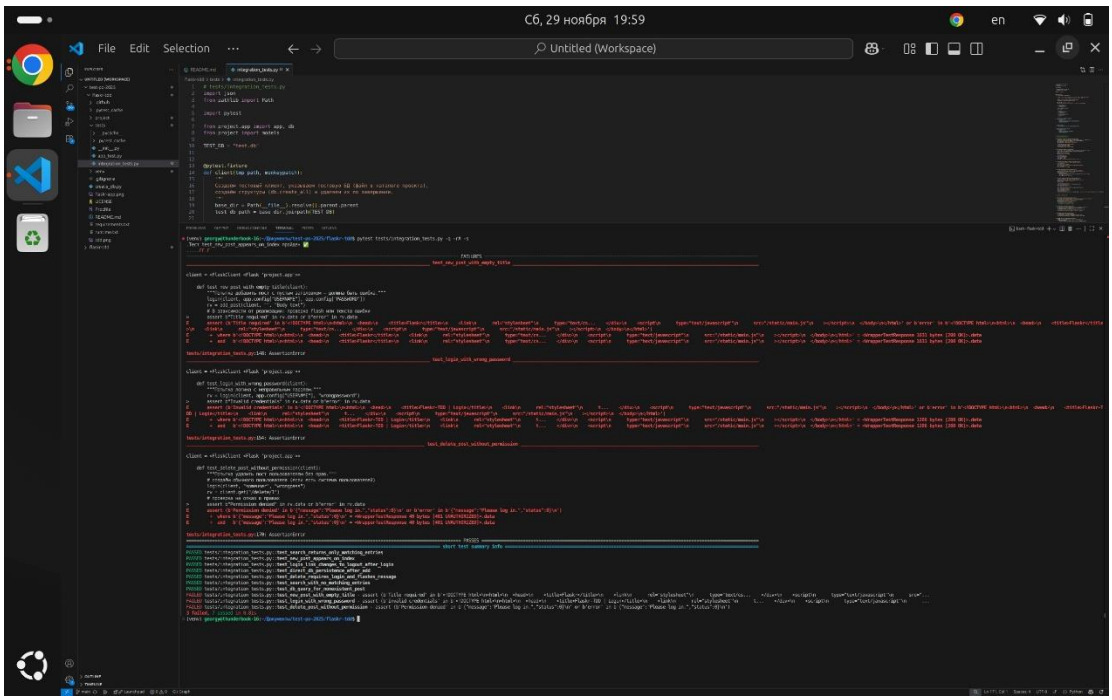
## Написание тестов

В проекте было реализовано 5 интеграционных тестов, покрывающих как успешные сценарии, так и граничные/ошибочные ситуации.
Все тесты используют разные модули приложения: Flask → SQLAlchemy → SQLite, что соответствует требованиям лабораторной работы.

1. **test_search_returns_only_matching_entries** – проверяет корректность работы маршрута /search/, взаимодействие API с базой данных: поиск по полю title, корректную фильтрацию результатов.
2. **test_new_post_appears_on_index –** проверяет добавление записи через POST-запрос к /add, сохранение записи в БД, появление новой записи на главной странице /.
3. **test_login_link_changes_to_logout_after_login** – проверяет правильность работы авторизации, корректное управление сессией Flask, переключение состояния интерфейса после логина.
4. **test_direct_db_persistence_after_add** - полный цикл: запрос к /add → добавление записи → реальное наличие её в БД. Проверяет то, что БД сохраняет данные независимо от HTML-интерфейса.
5. **test_delete_requires_login_and_flashes_message** – проверяет доступ к удалению записи ограничен авторизацией.

В проекте уже были готовые тесты, но мы ими пользоваться не будем. Сам файл с тестами я прикреплю отдельно. В самом проекте файлик с тестами будет находиться в project/tests/integration_tests.py.

Находясь в корне проекта запускаем наши тесты при помощи команды:
pytest tests/integration_tests.py -q -rA -s


Сб, 29 ноября 19:59
en
File Edit Selection
Untitled (Workspace)

## Выводы

В ходе лабораторной работы был выполнен набор интеграционных тестов, охватывающих основные точки взаимодействия системы: маршруты Flask, шаблоны, сессии пользователя и слой работы с базой данных (SQLAlchemy).

Всего было реализовано и запущено 10 интеграционных тестов, включая как успешные сценарии, так и ошибочные/граничные случаи.

Все успешные сценарии были пройдены, что подтверждает корректную работу ключевых пользовательских функций:

- поиск записей,
- создание новых постов,
- авторизация,
- удаление записей,
- взаимодействие с базой данных напрямую.

Часть тестов для ошибочных сценариев ожидаемо завершилась неуспешно, что не является ошибкой тестирования — они выявили места, где приложение не проверяет некорректные данные.

Это нормальное и ожидаемое поведение для проверки граничных условий:

- приложение не обрабатывает пустой заголовок поста,
- не выводит сообщение об ошибке при неверном пароле,
- не различает ситуации "нет прав" и "не авторизован".

Стоит отметить что логи в ошибочных сценариях оказались неинформативными не из-за тестов, а потому что разработчик приложения не предусмотрел детальную обработку ошибок или вывод поясняющих сообщений.

API возвращает минимальные ответы без конкретизации, а шаблоны не отображают flash-сообщения об ошибках.