Министерство науки и высшего образования Российской Федерации

ФГАОУ ВО «Уральский федеральный университет

имени первого Президента России Б.Н.Ельцина»

Институт радиоэлектроники и информационных технологий

Отчёт

по лабораторной работе №4

«Тестирование бэкенд приложения»

Выполнил: студент гр. РИМ-150950

Коренева Екатерина Александровна

Екатеринбург

2025 г.

Цель работы: познакомится со способами тестирования приложения, написать рабочие тесты.

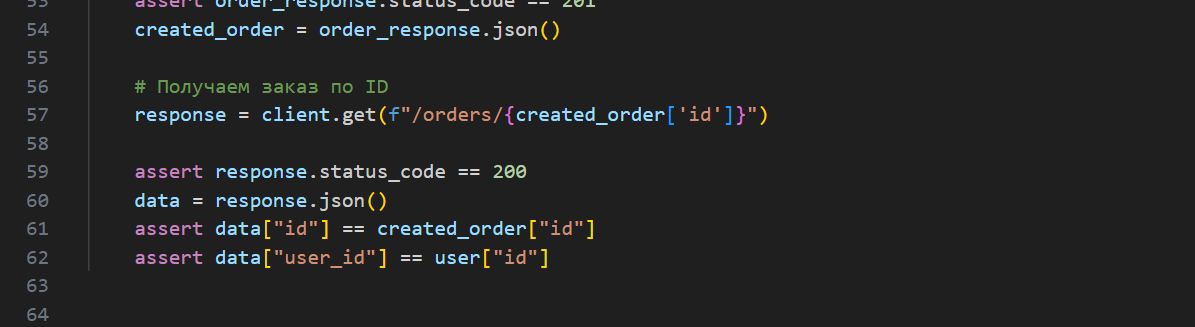
Задачи: написать тесты для проверки каждой функции приложения

Ход работы:

1. Тесты Controllers. Тест test\_get\_orders\_list проверяет, что эндпоинт GET /orders возвращает успешный HTTP-ответ (200) и корректную структуру данных — список заказов под ключом orders

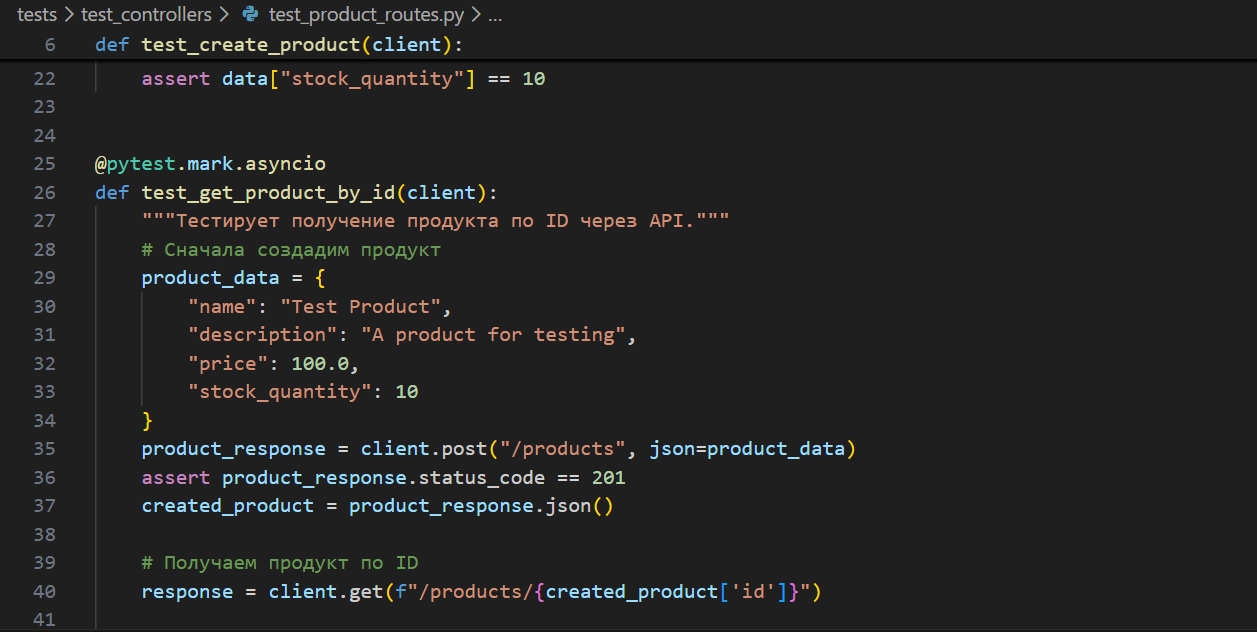


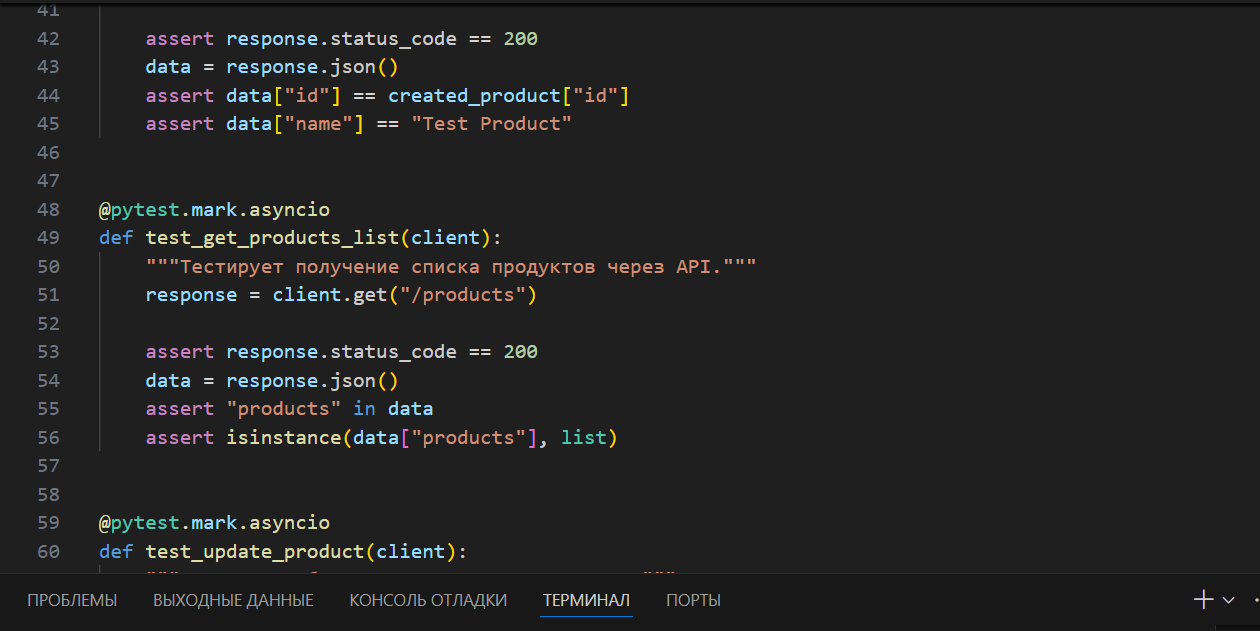




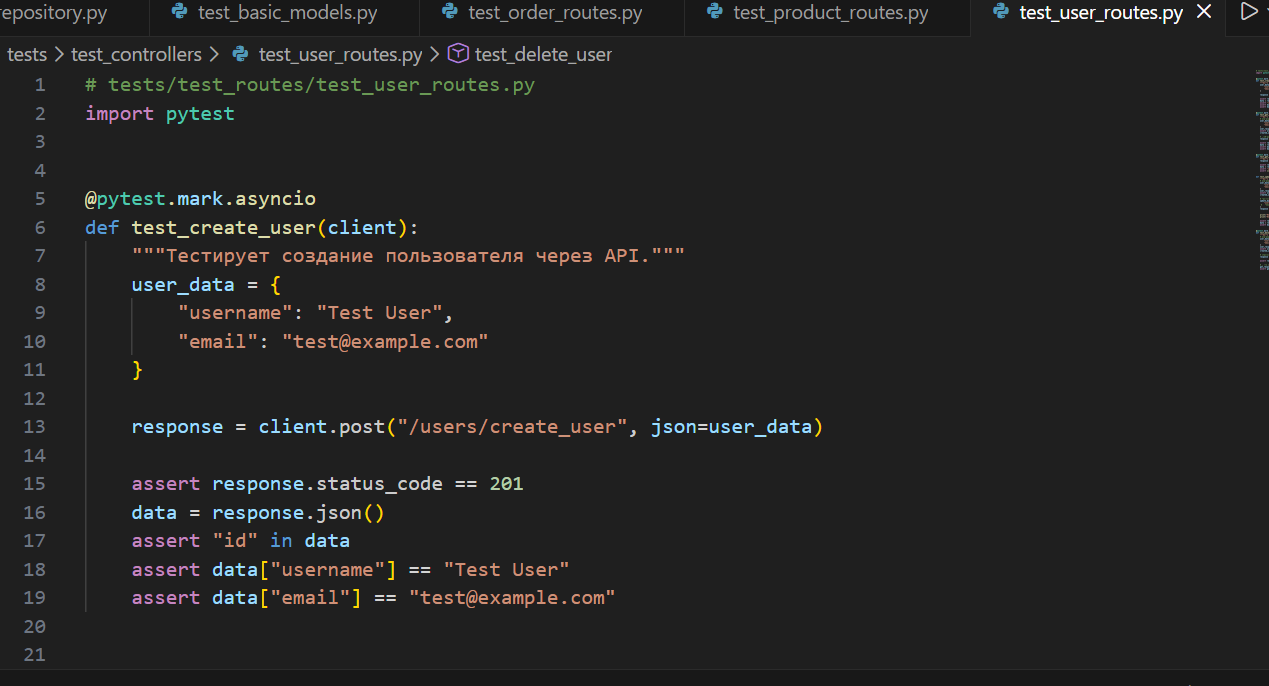
1. test\_product\_routes - проверяют работу эндпоинтов для управления продуктами (создание, получение, обновление, удаление) через HTTP-запросы к API с использованием TestClient.

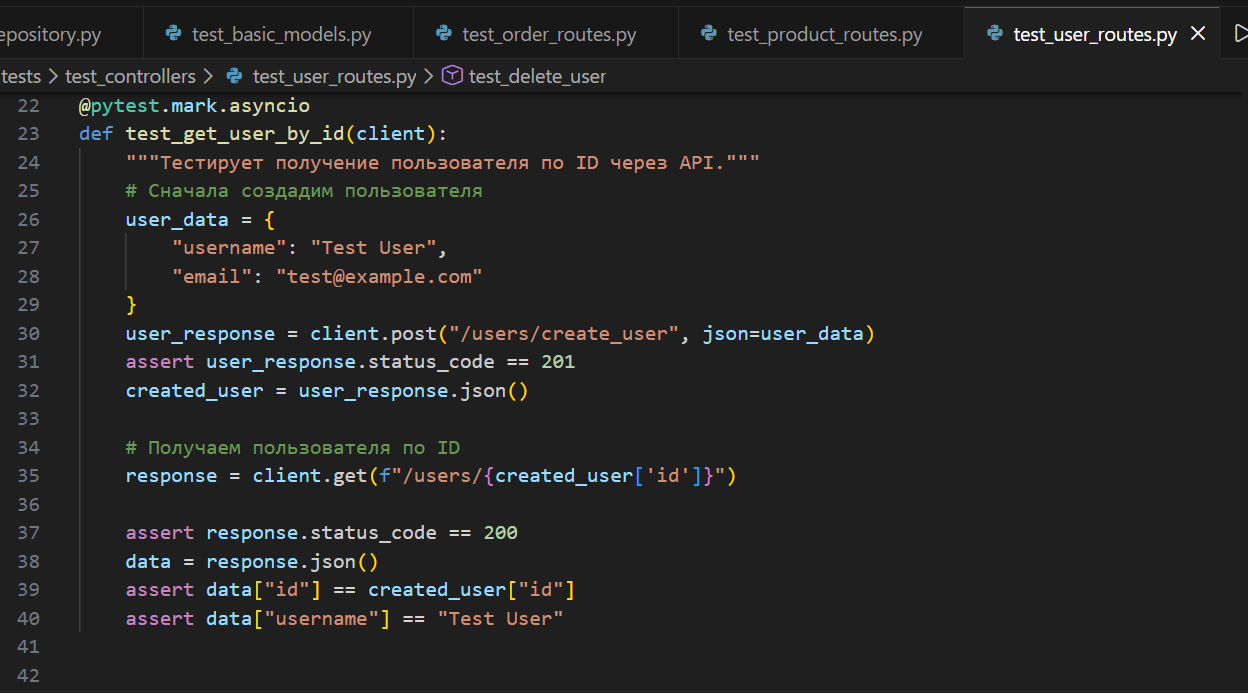




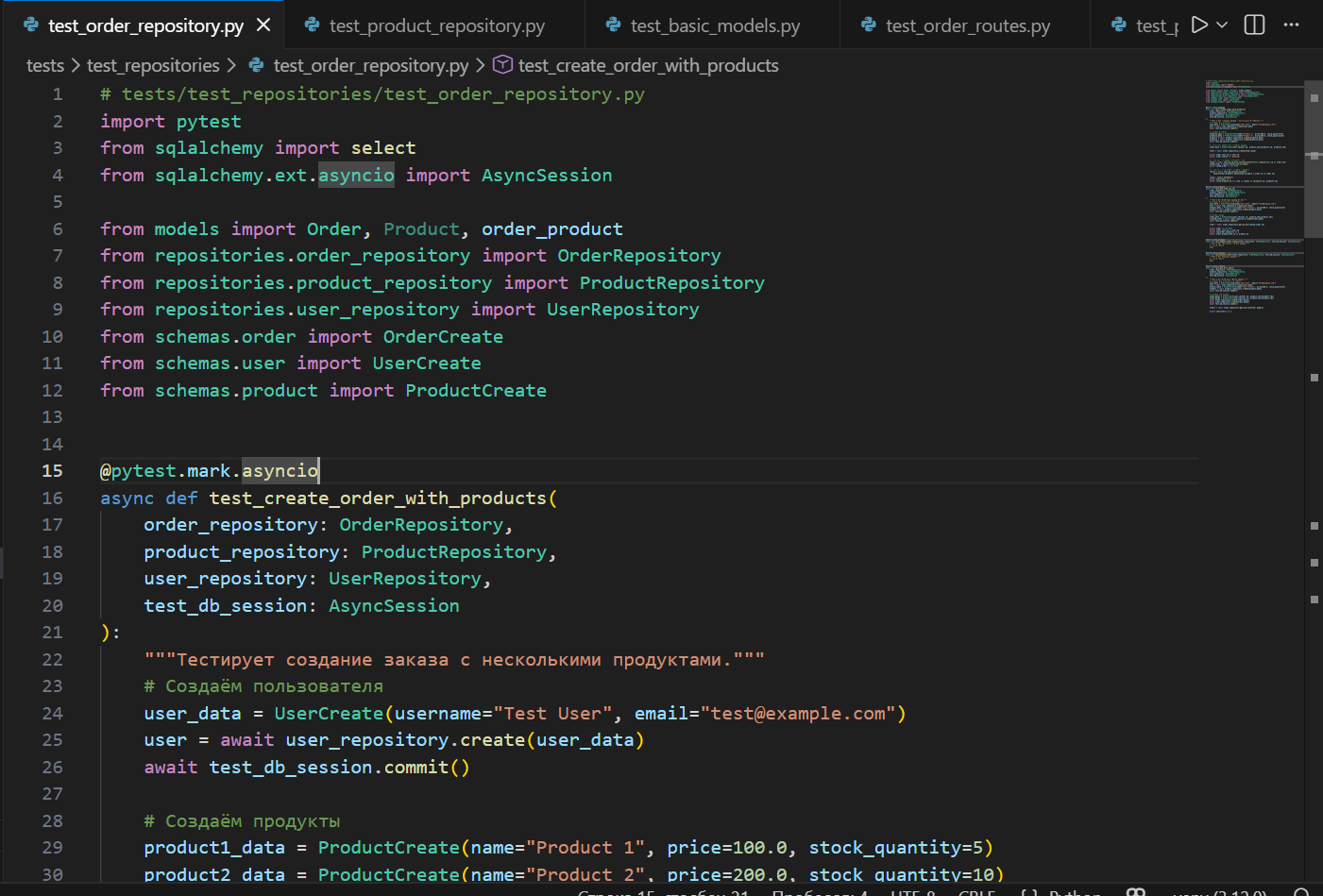


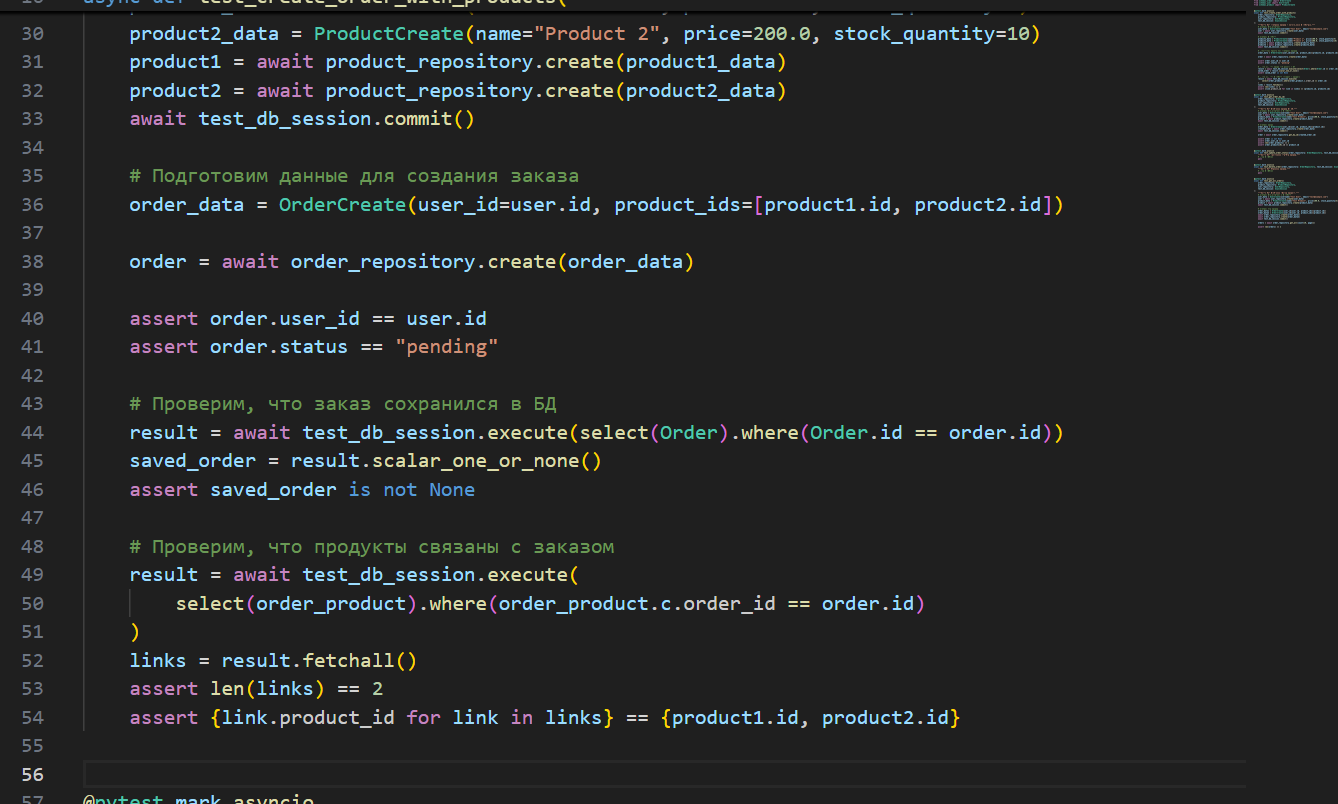
1. test\_user\_routes – проверяют работу эндпоинтов для управления пользователями (создание, получение, обновление, удаление) через HTTP-запросы к API с использованием TestClient.





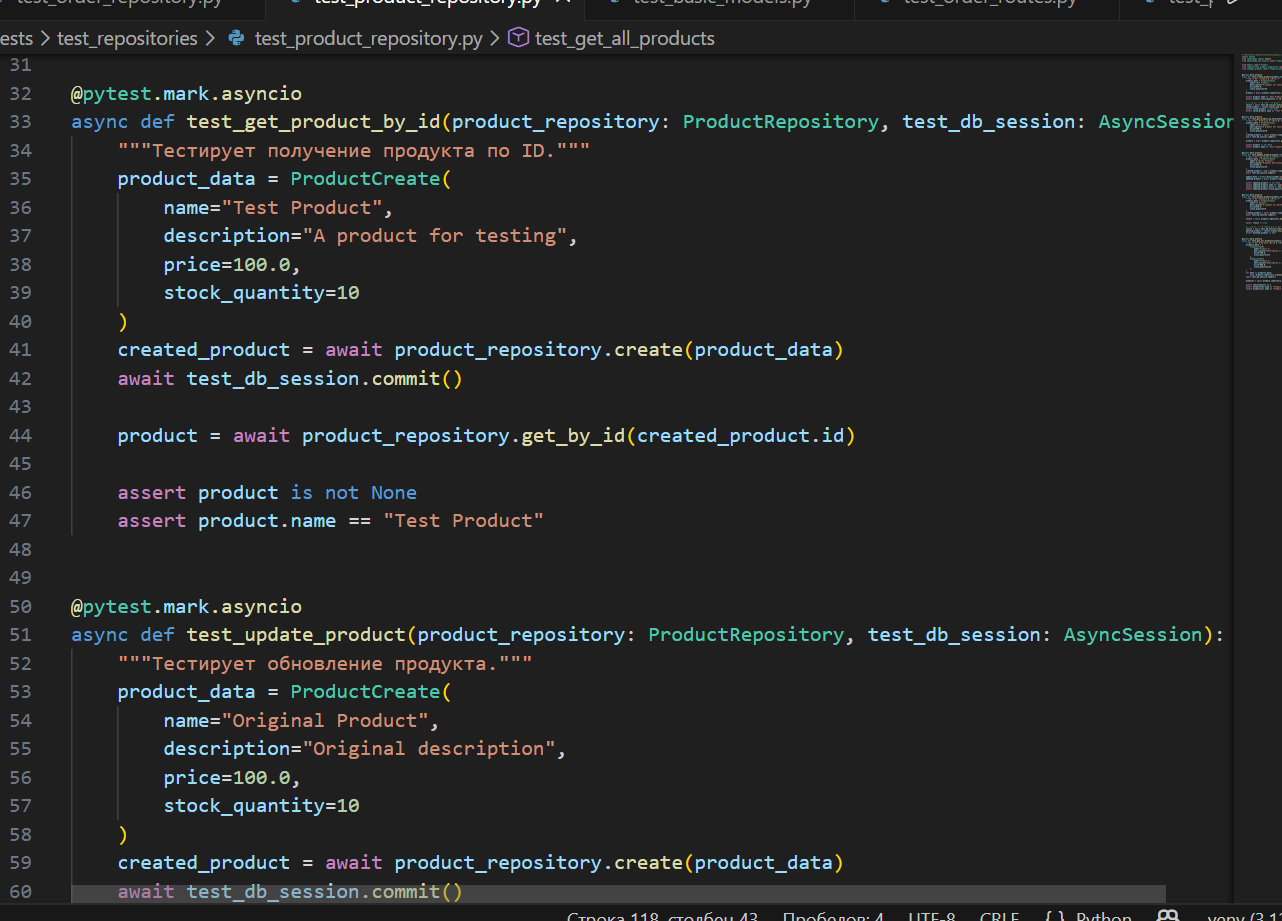
1. test\_order\_repository.py проверяют работу слоя репозитория для заказов (CRUD-операции, связь с продуктами и пользователями) с использованием фикстуры тестовой базы данных.



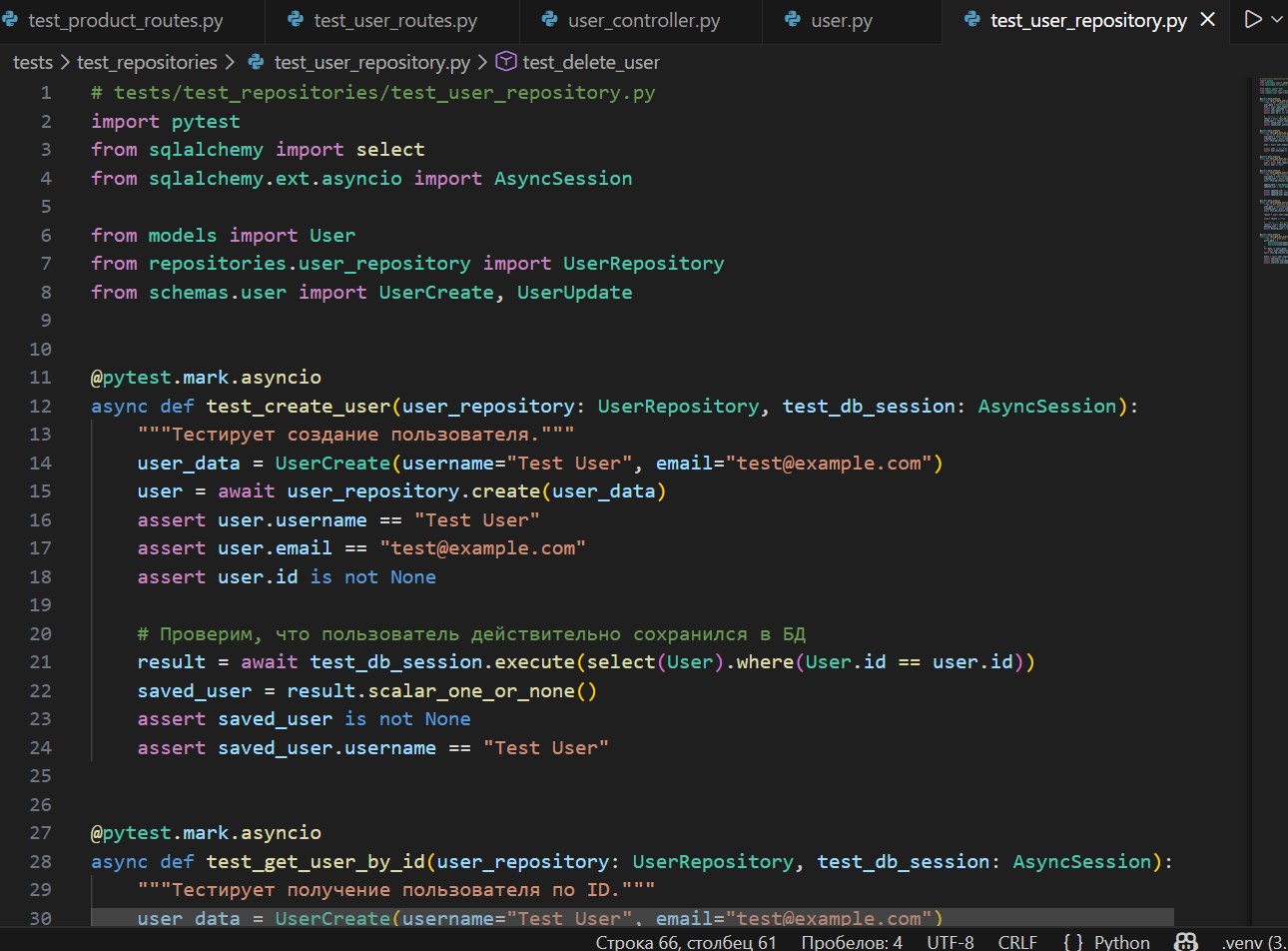


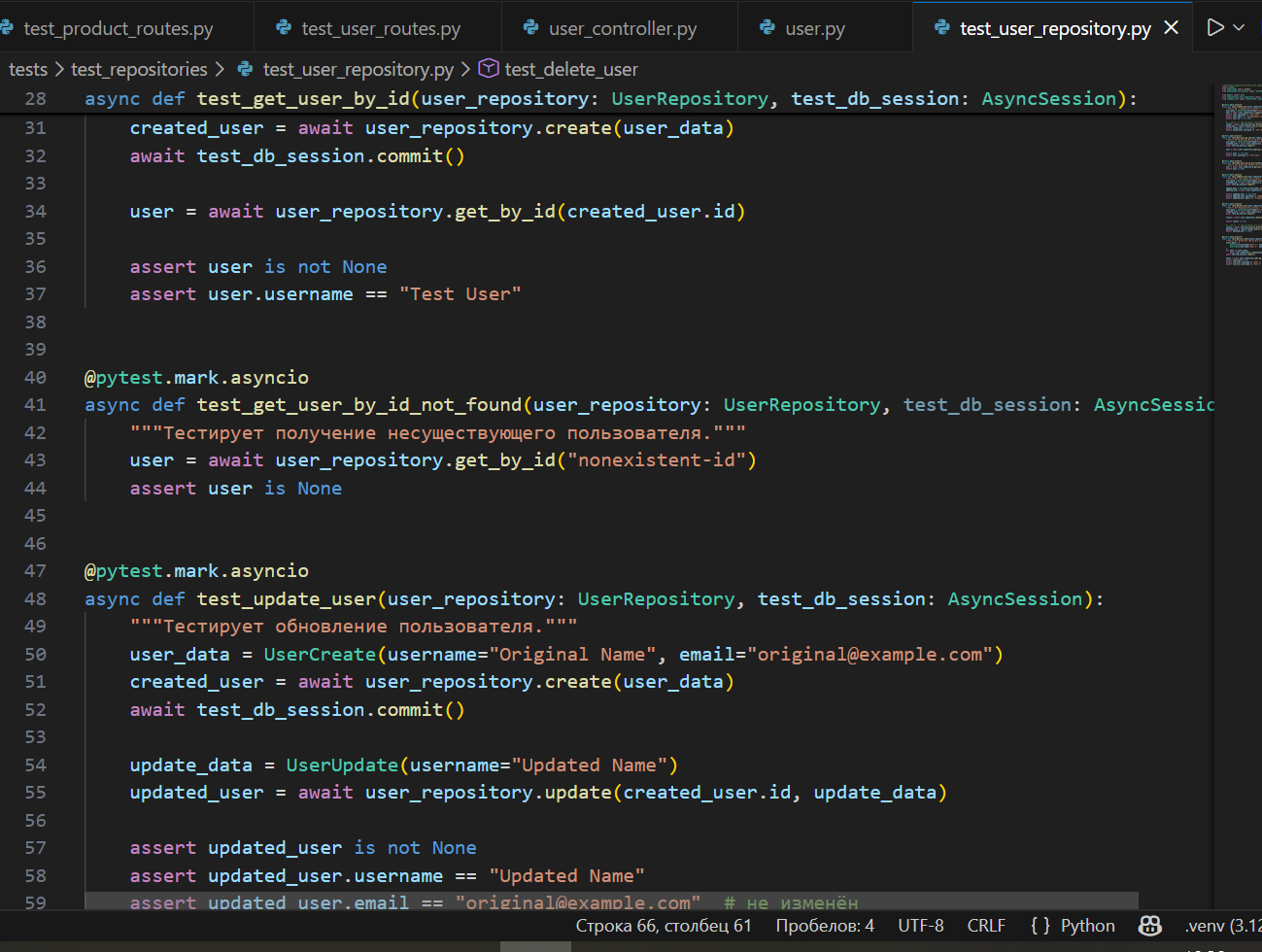
1. test\_product\_repository проверяют работу слоя репозитория для продуктов (CRUD-операции, изменение остатков) с использованием фикстуры тестовой базы данных.



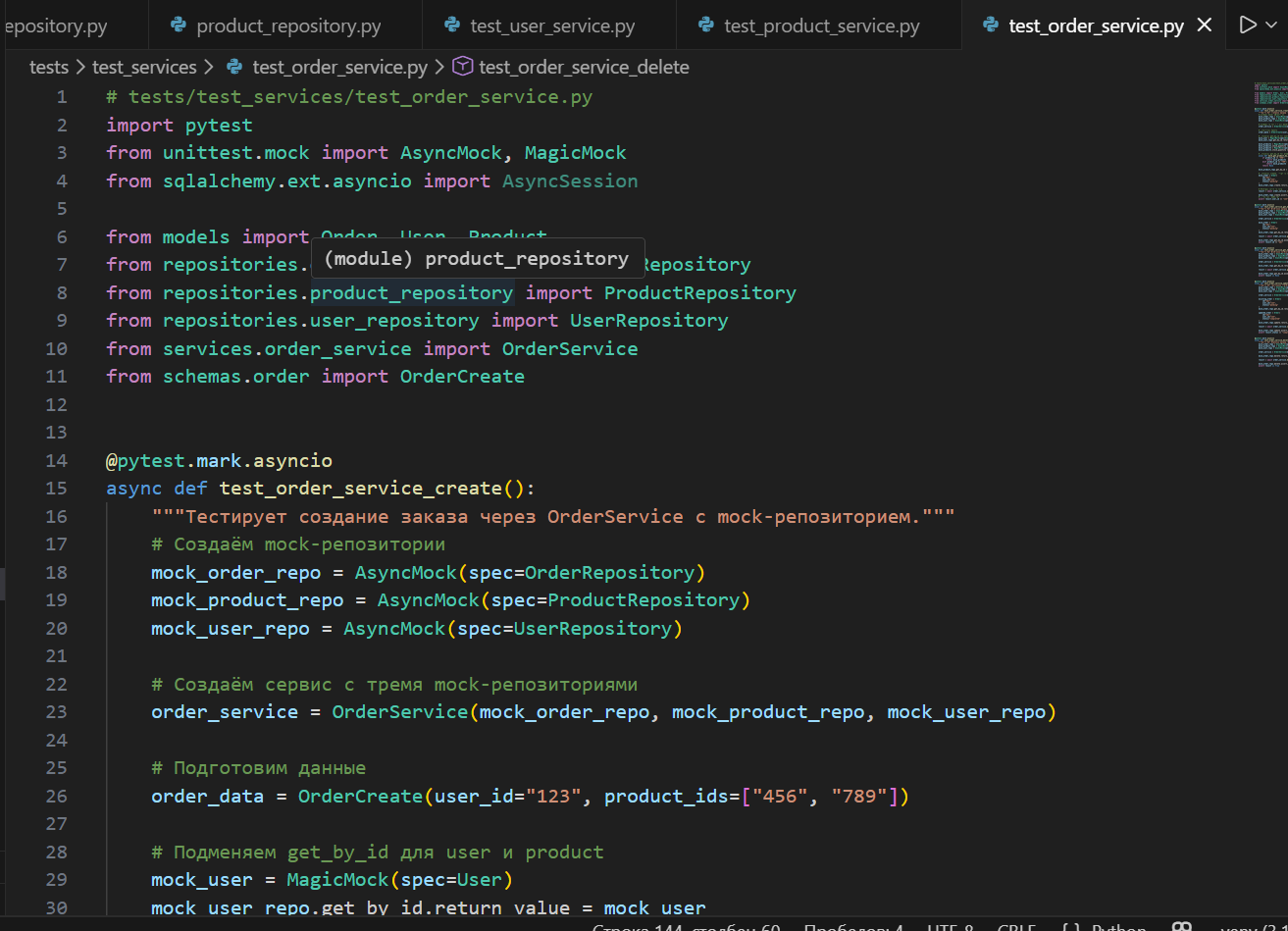


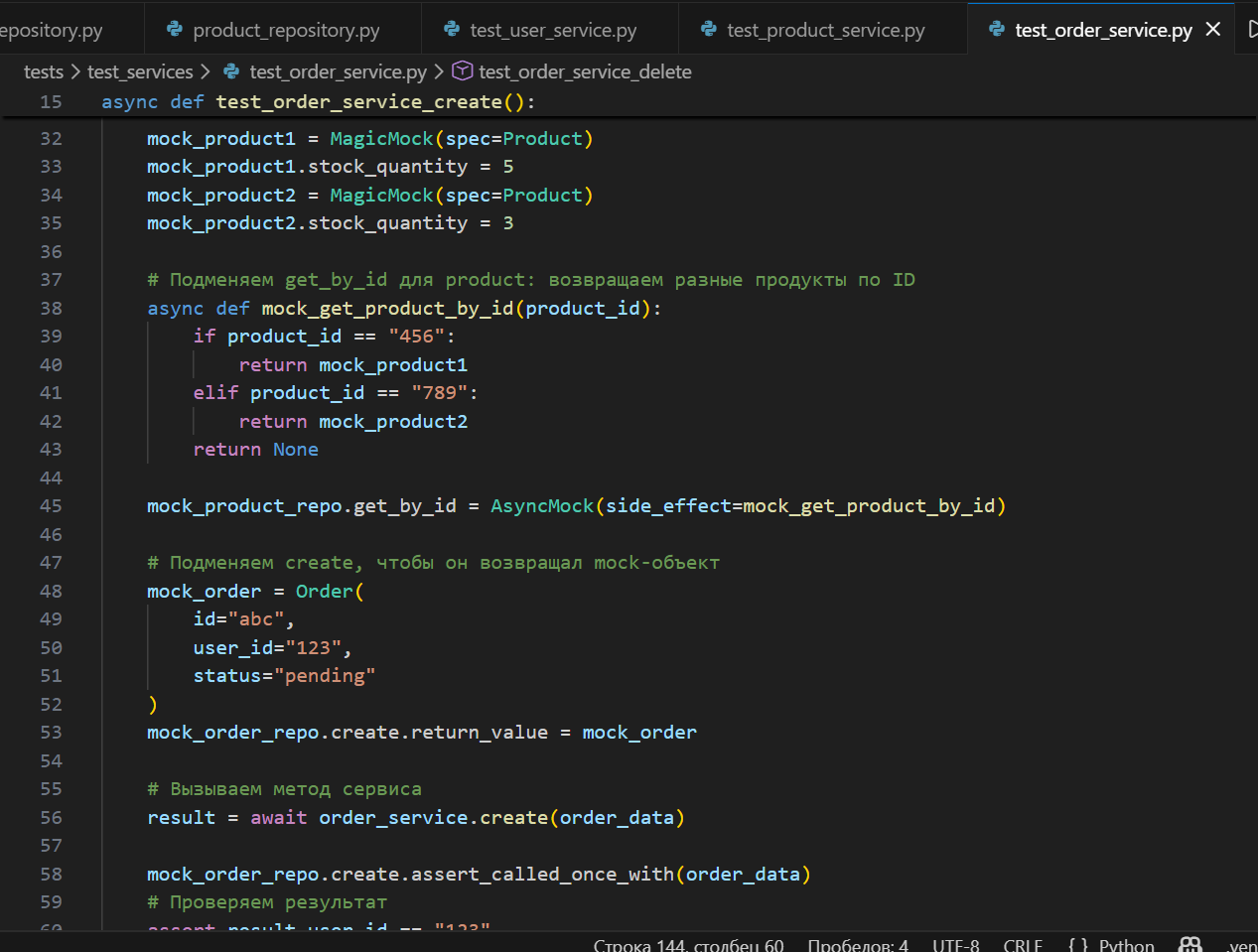
1. test\_user\_repository



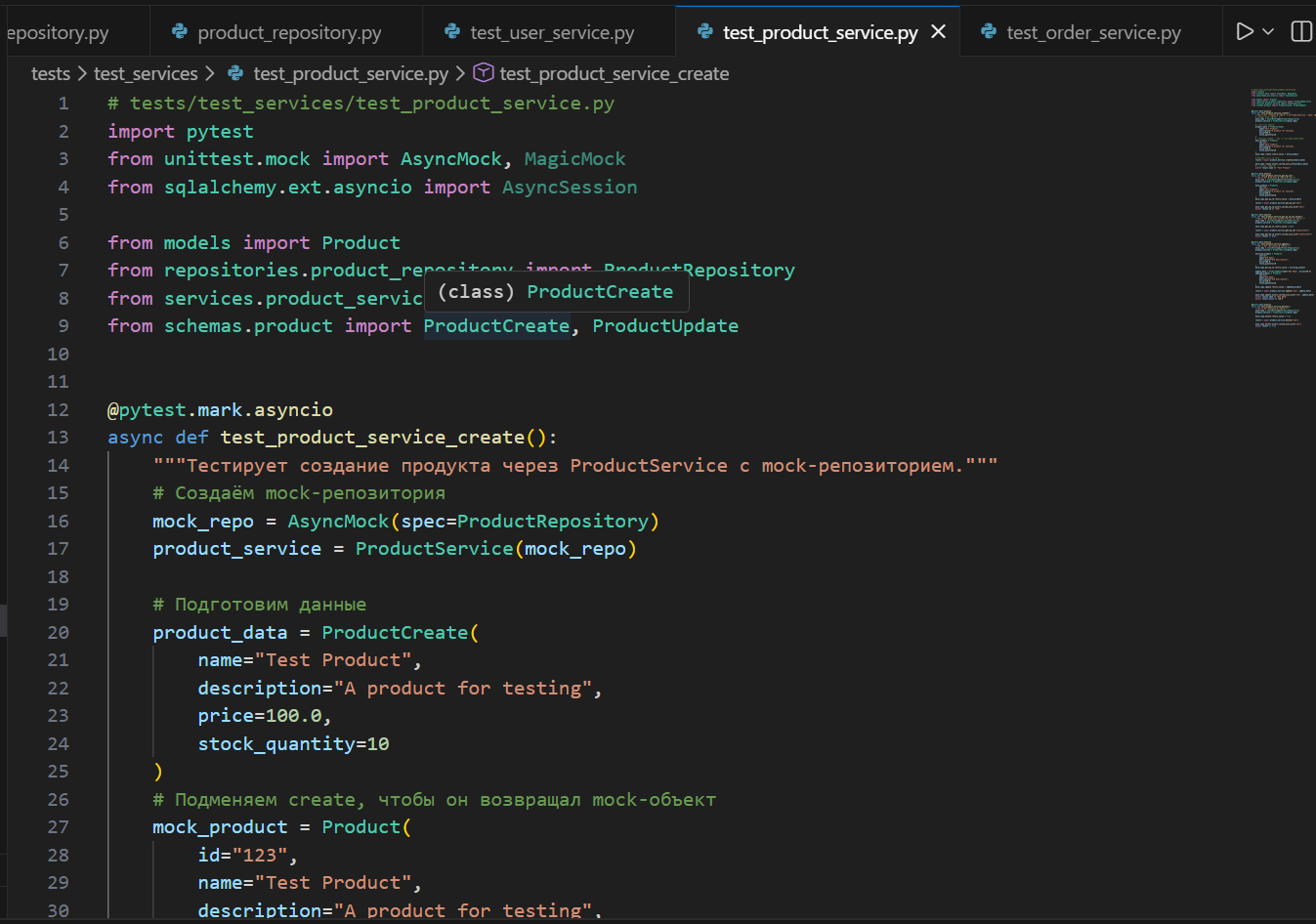


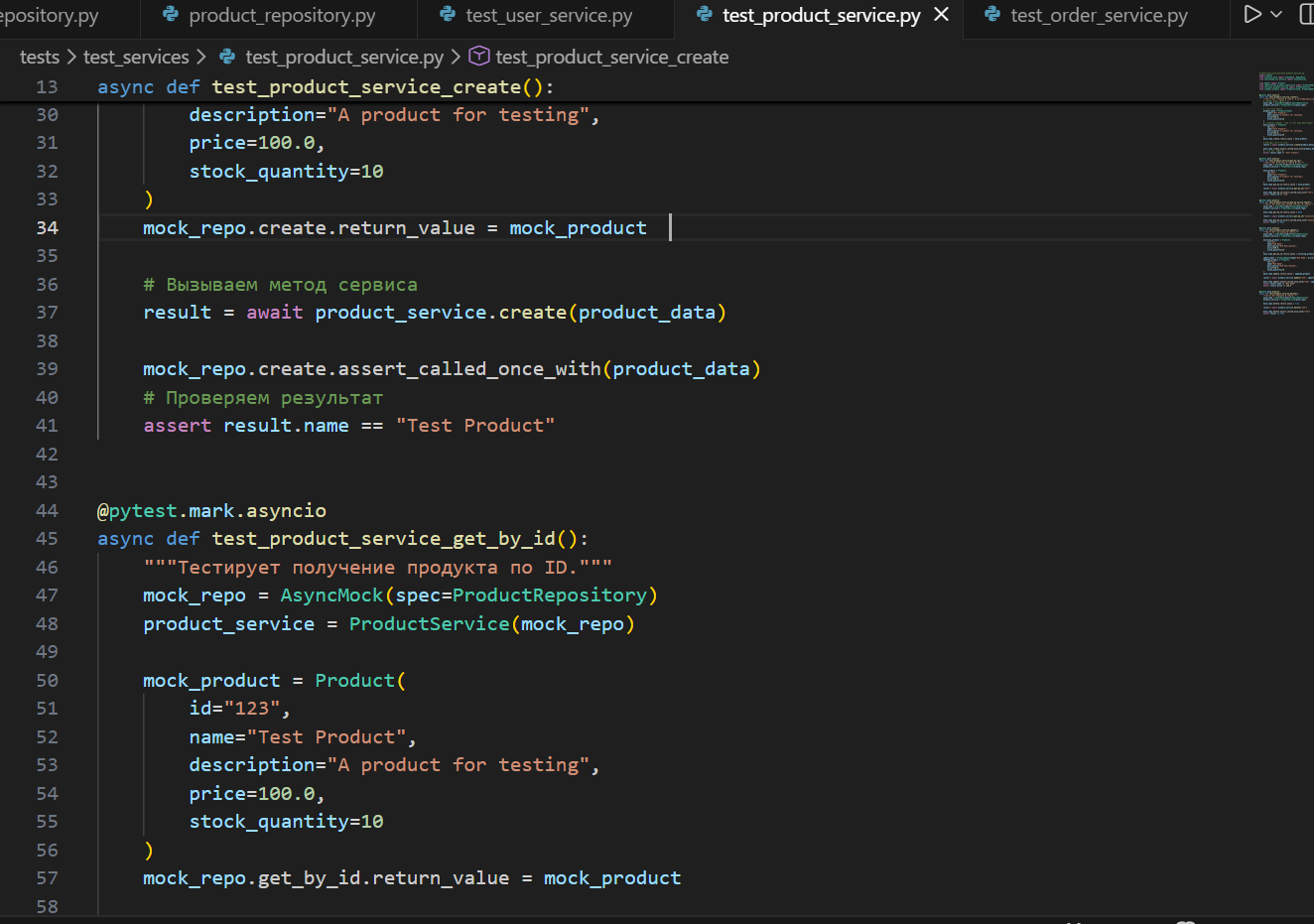
1. test\_order\_service проверяют работу сервисного слоя для заказов (создание, обновление статуса, удаление) с использованием mock-объектов для изоляции от репозиториев и БД.



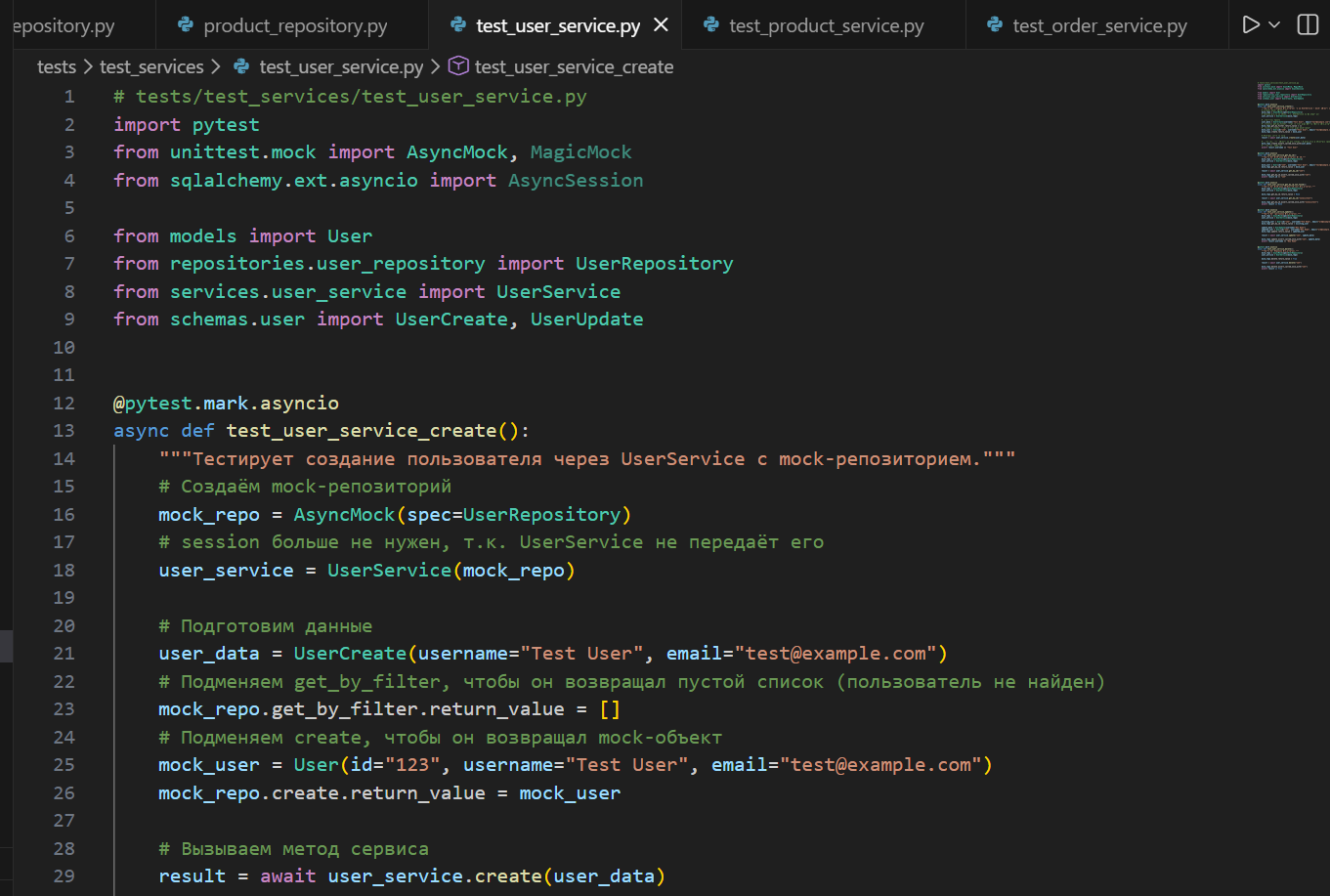


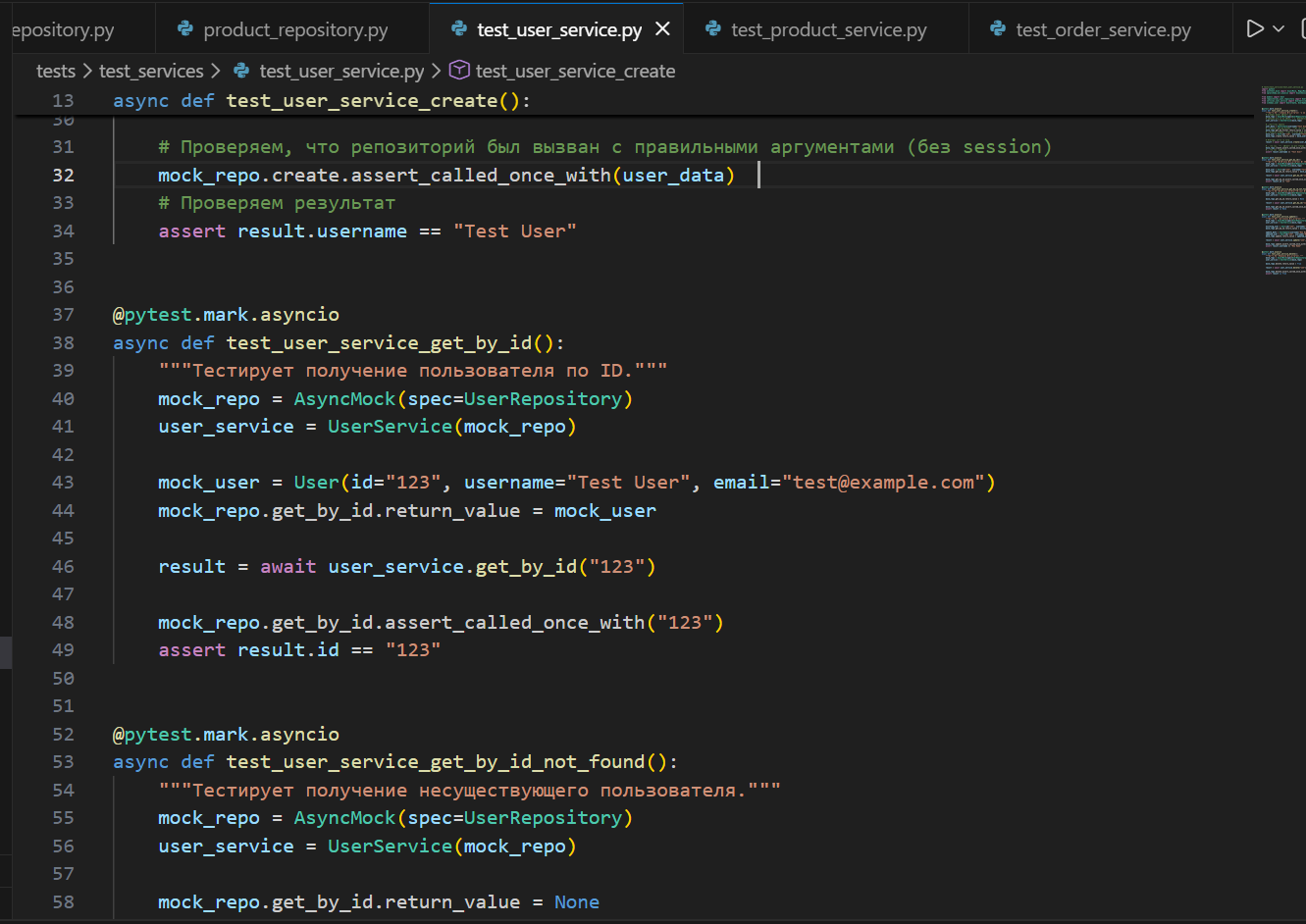
1. test\_product\_service проверяют работу сервисного слоя для продуктов (создание, обновление, удаление, проверка остатков) с использованием mock-объектов для изоляции от репозиториев и БД.

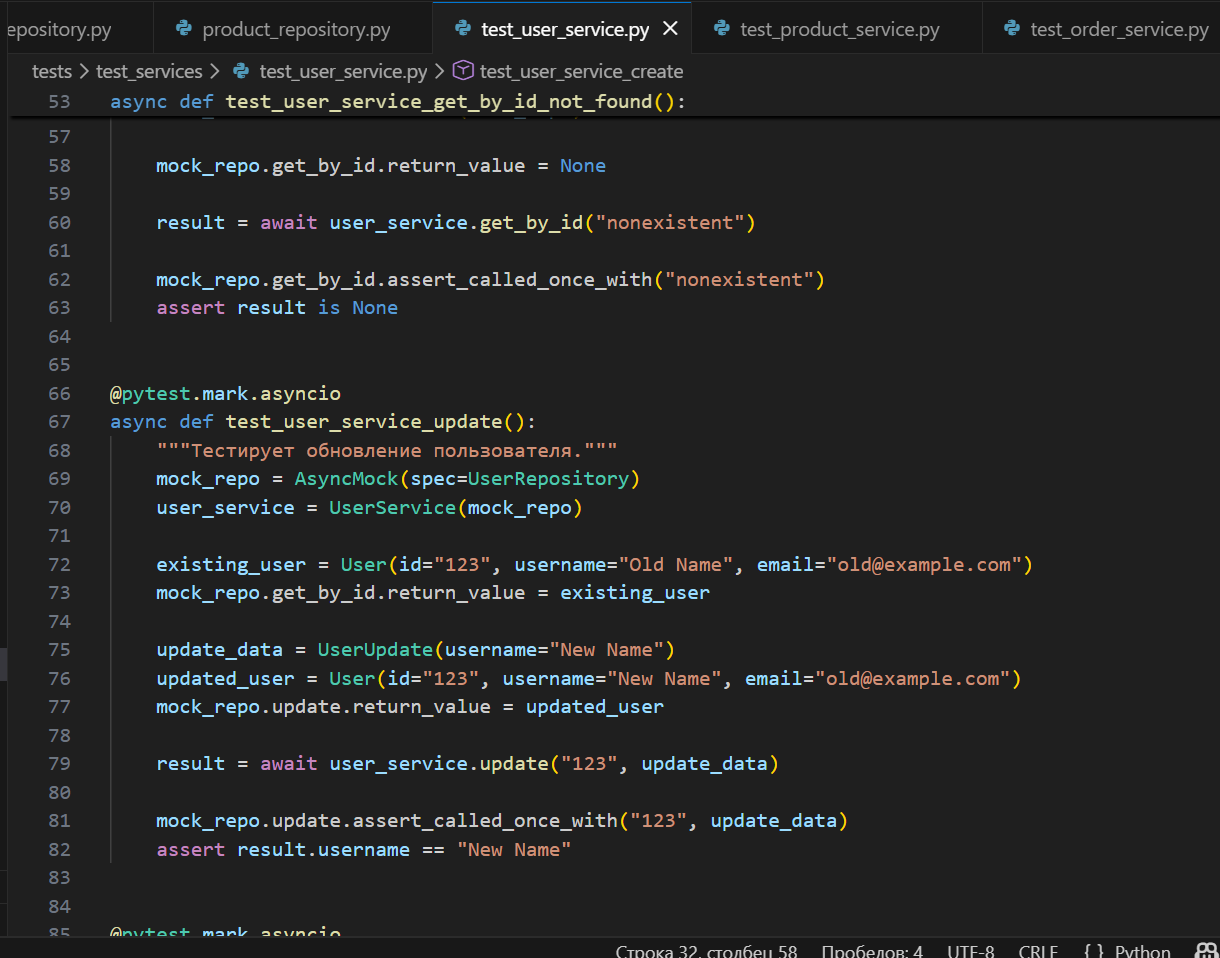




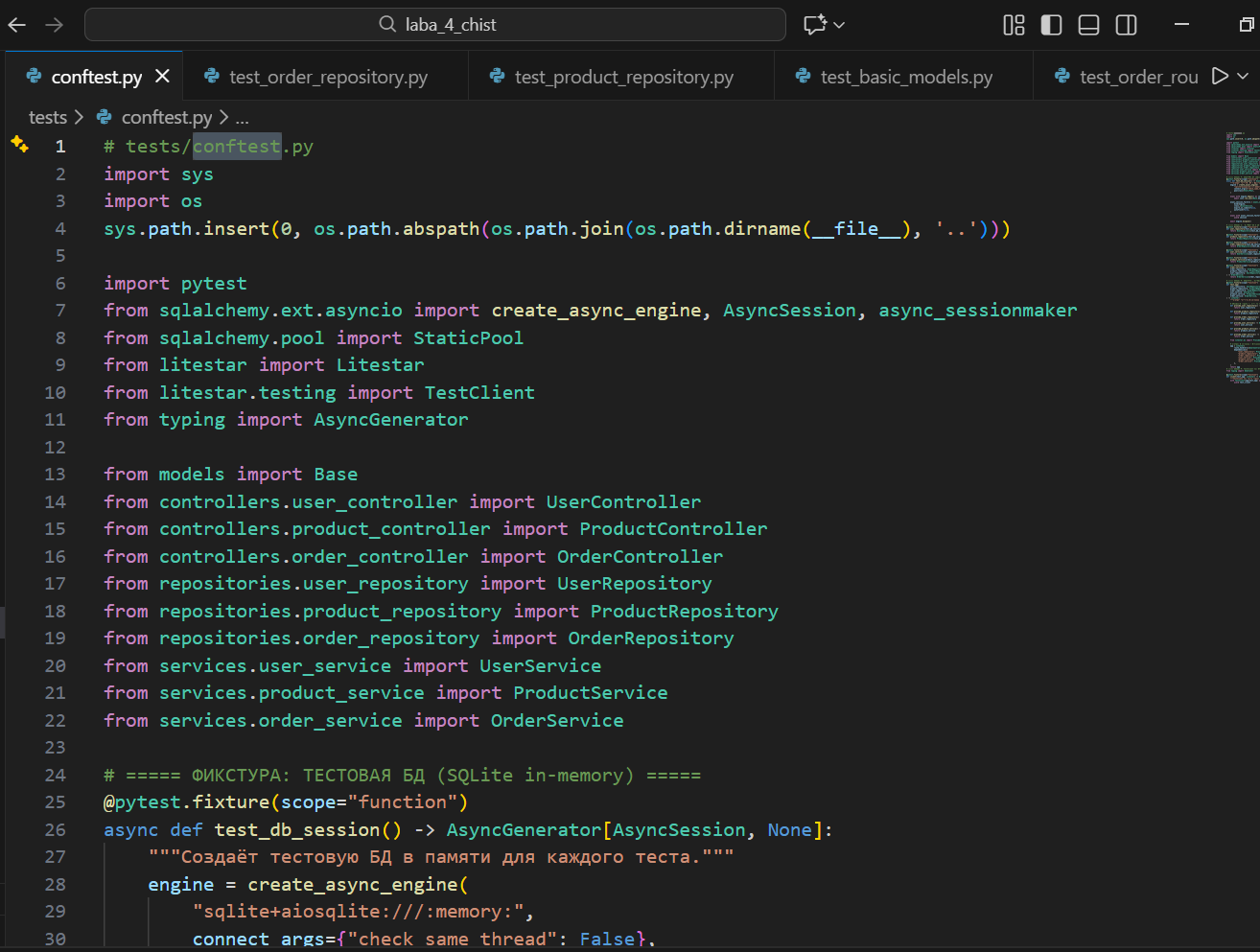
1. test\_user\_service

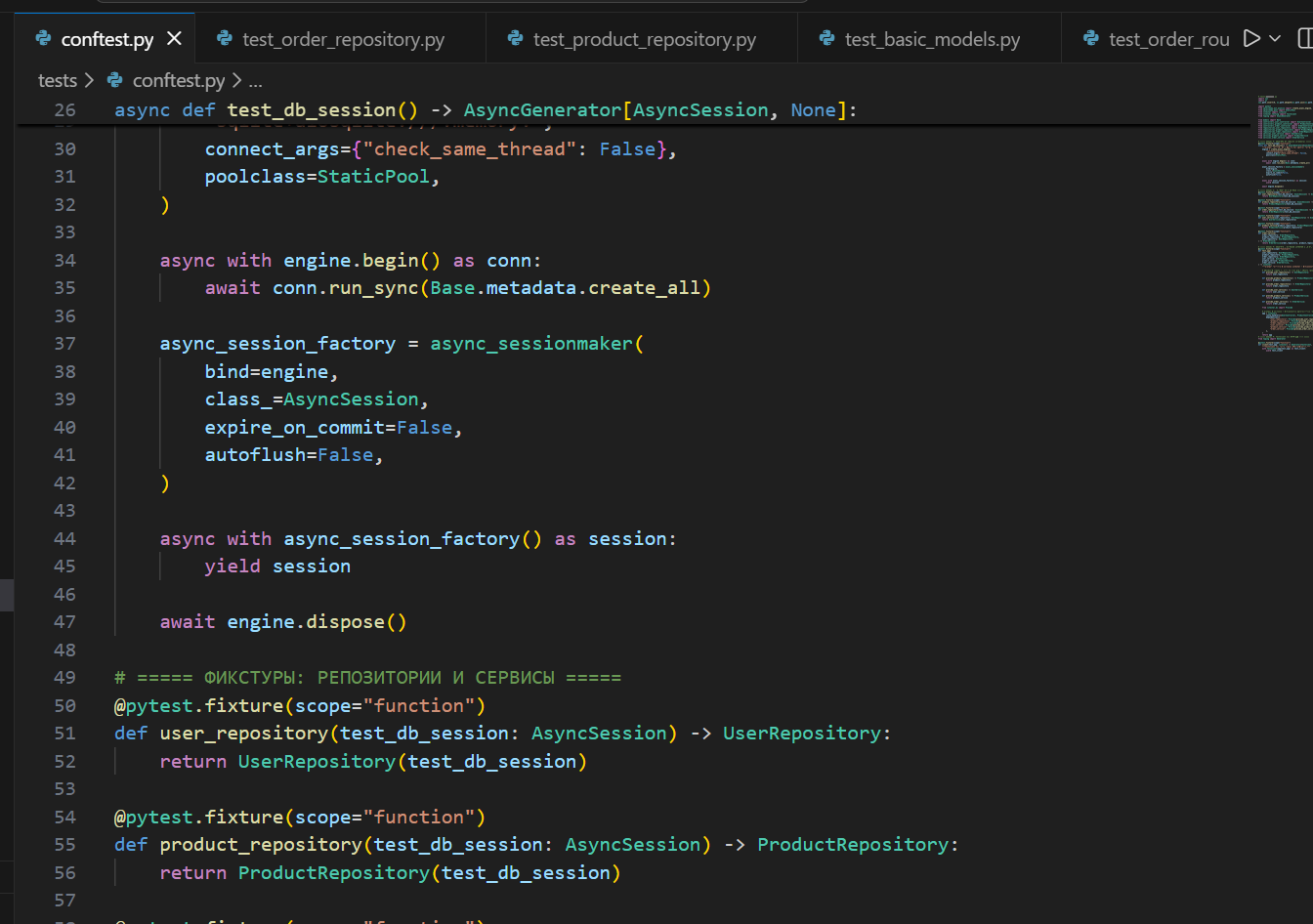


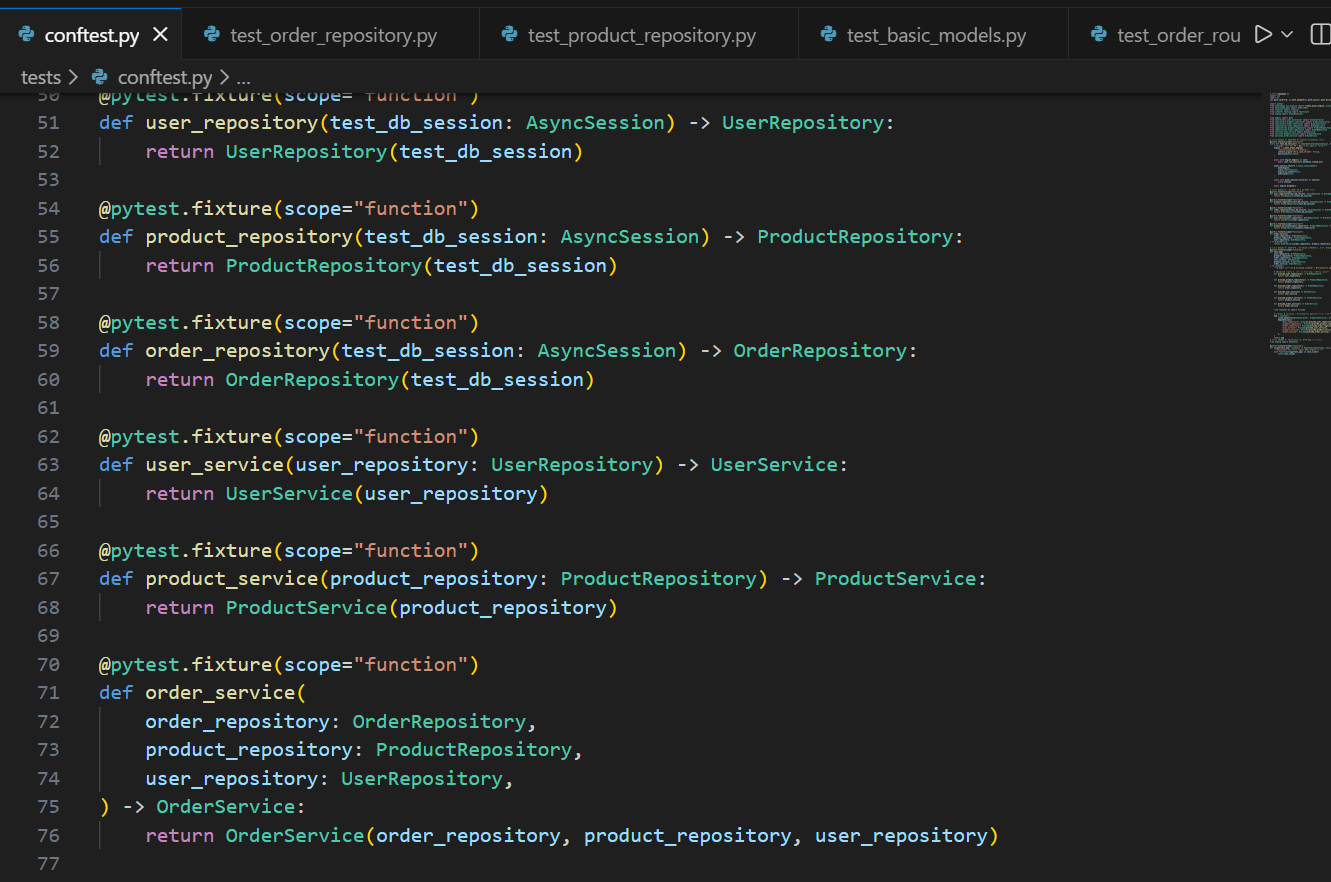


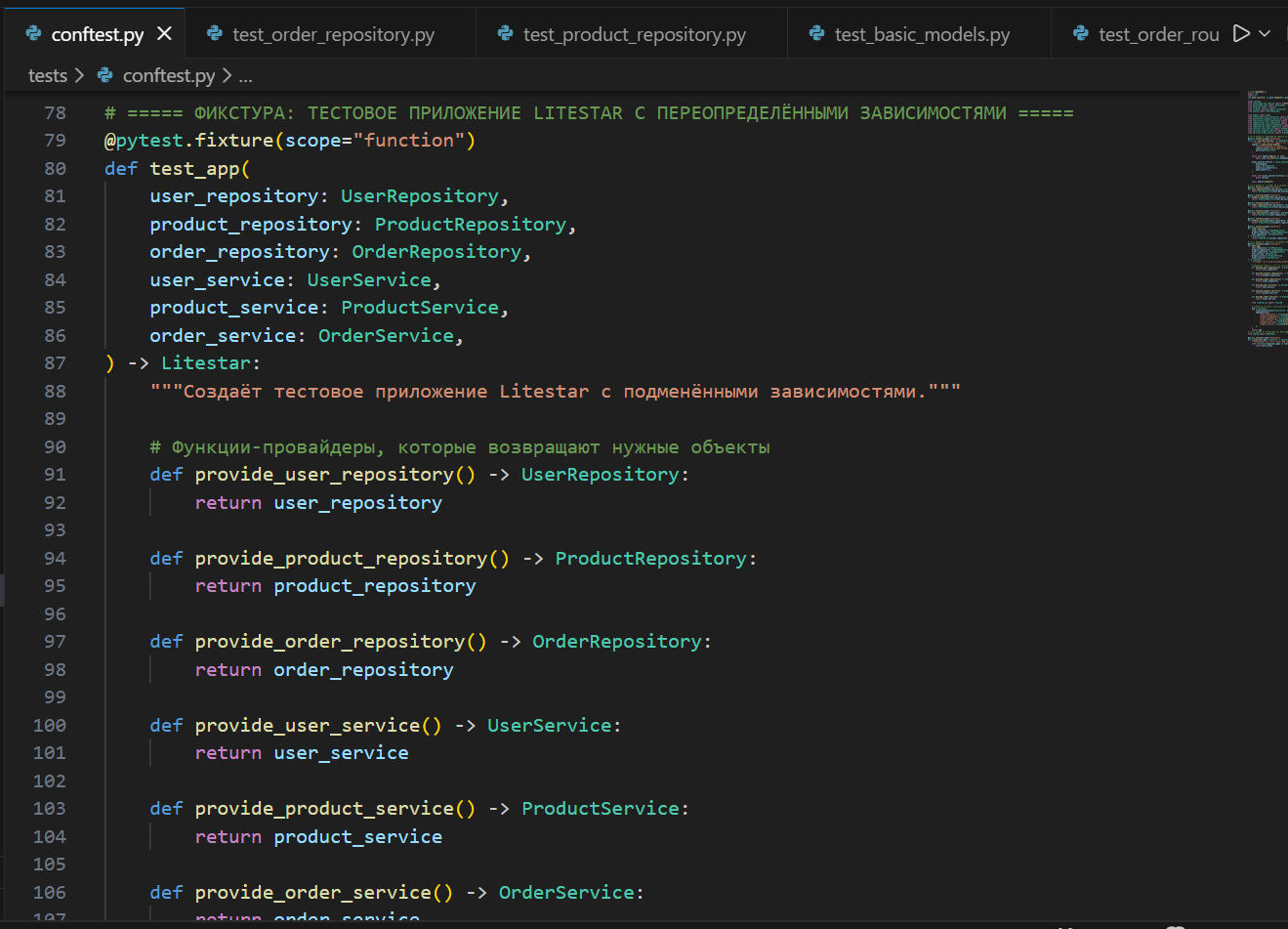


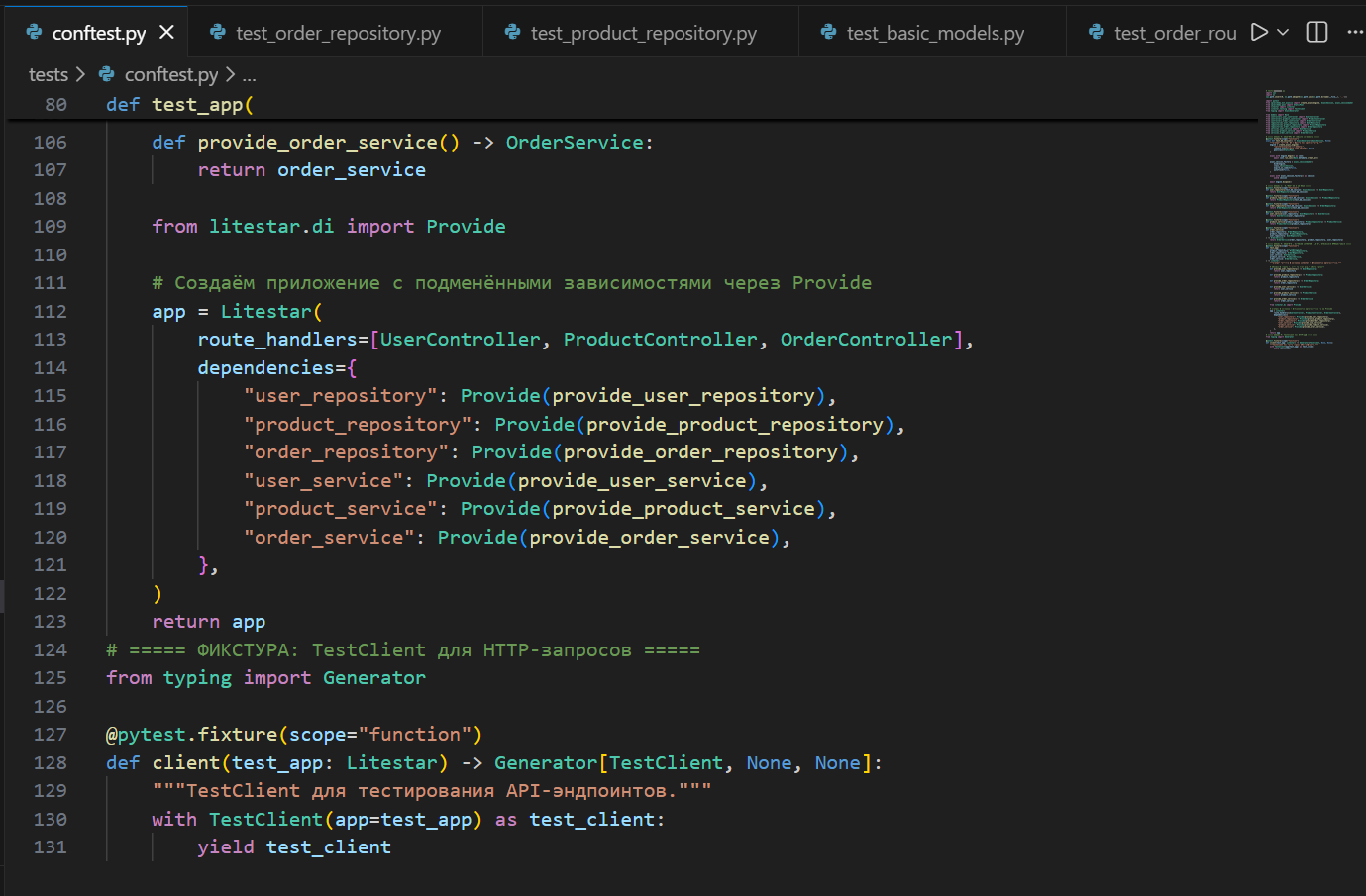
1. conftest.py содержит фикстуры (fixtures) для pytest, которые обеспечивают изолированное тестовое окружение: создают тестовую базу данных в памяти (SQLite), настраивают сессию, репозитории и сервисы, а также предоставляют TestClient для тестирования API-эндпоинтов.





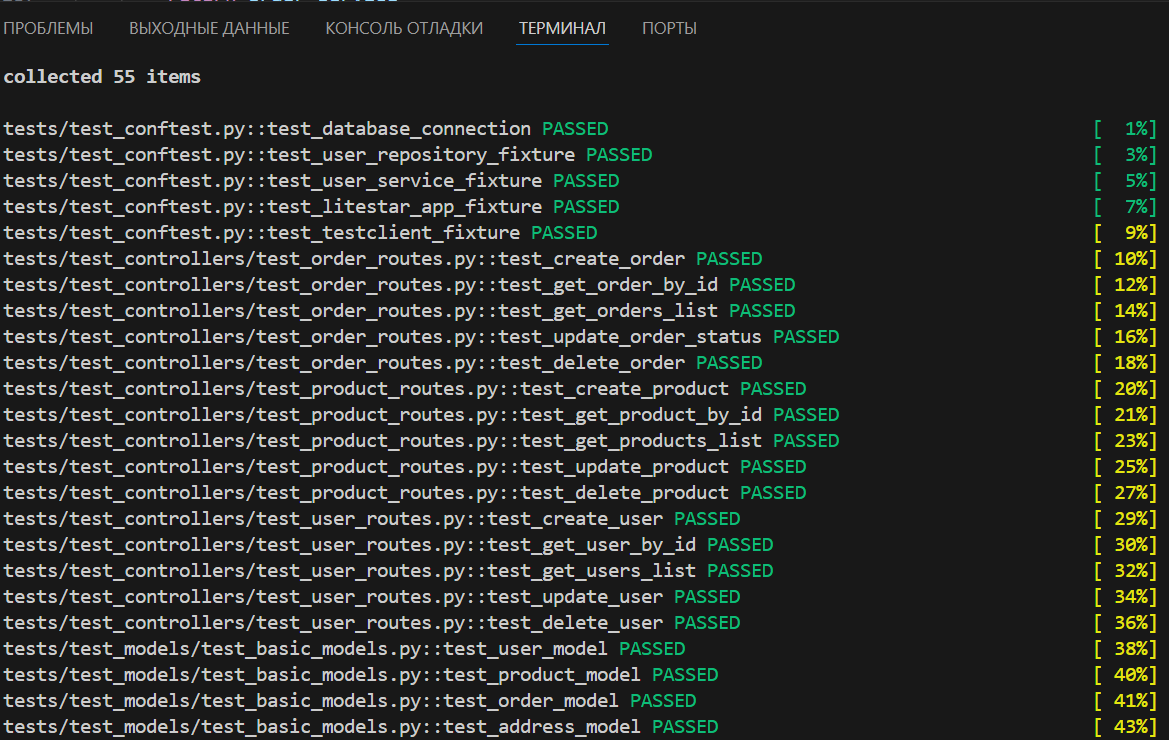


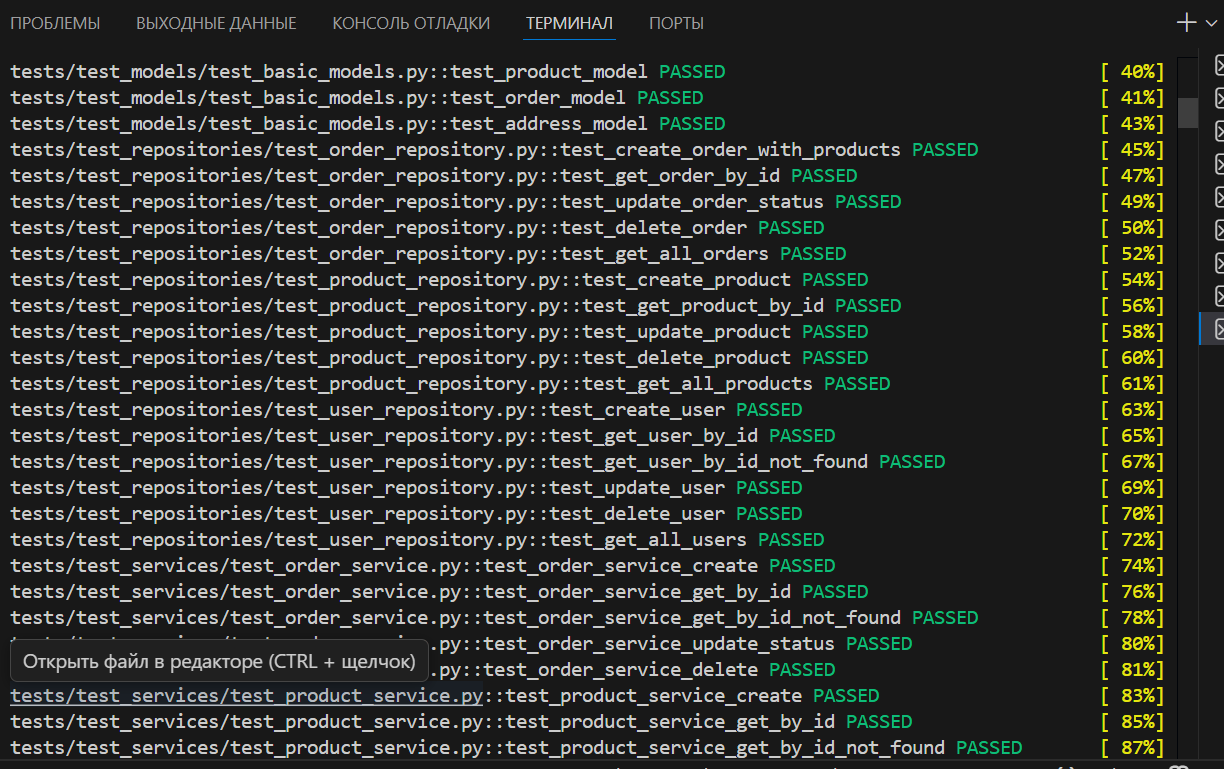




Запуск тестов

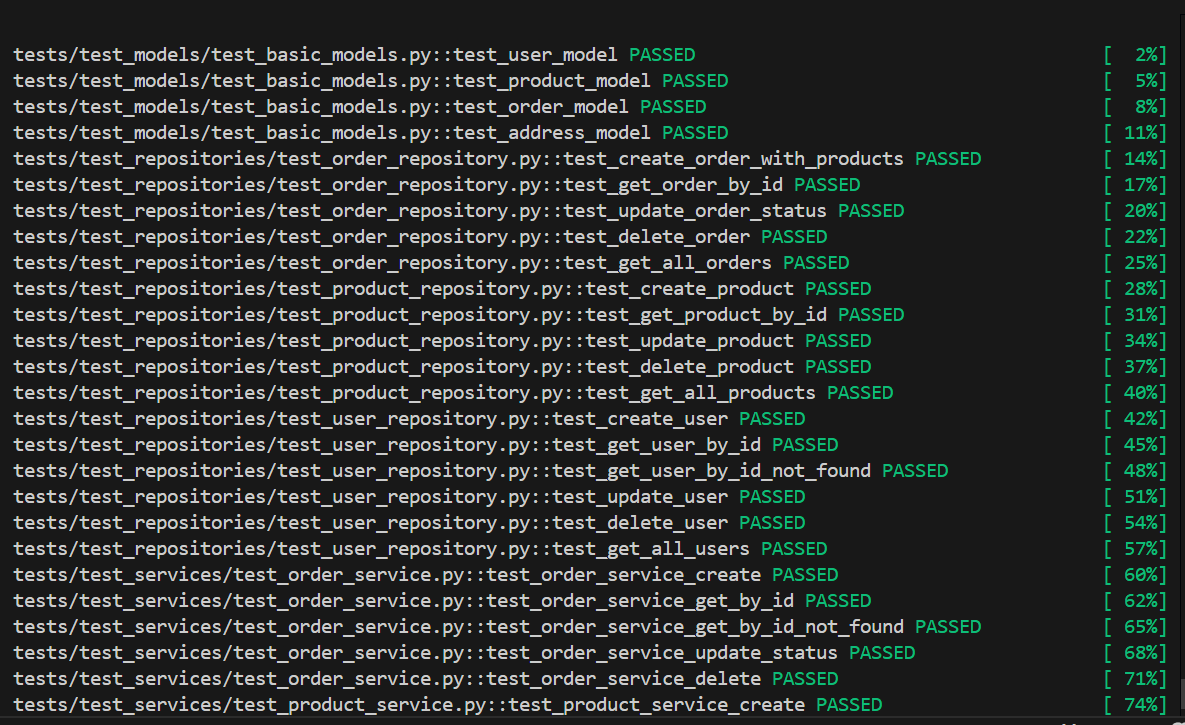
Все тесты

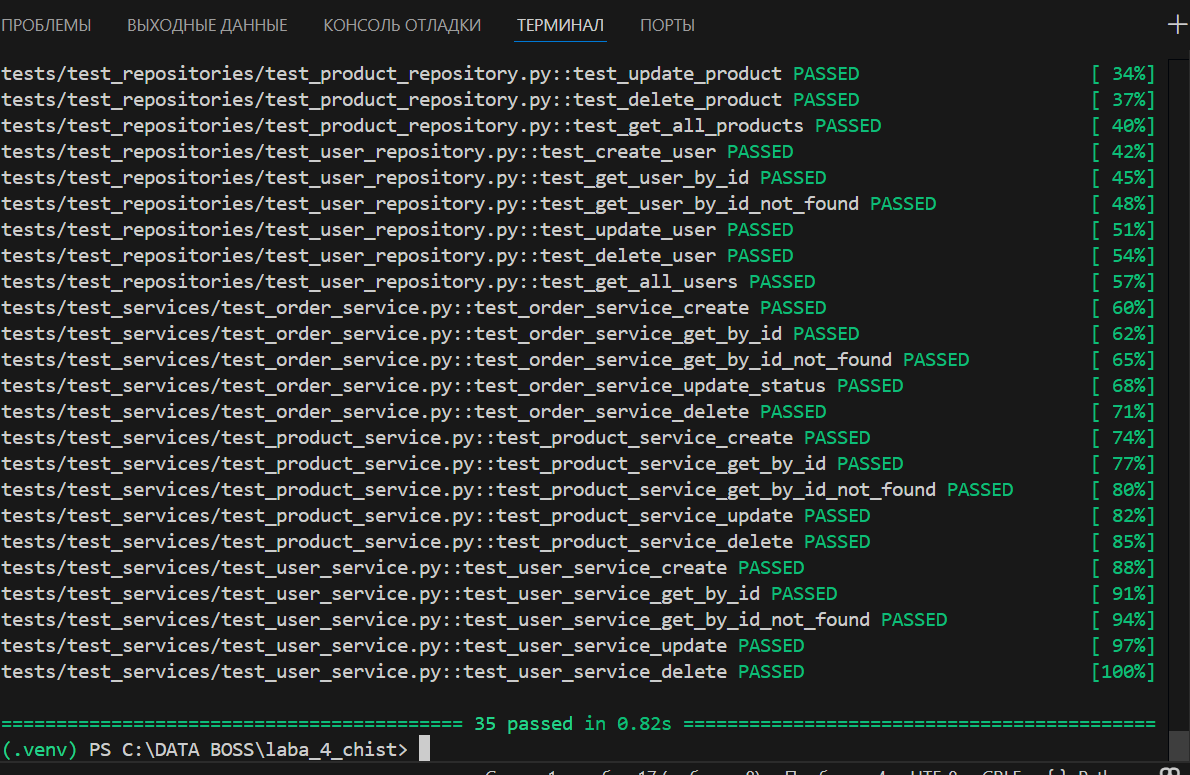




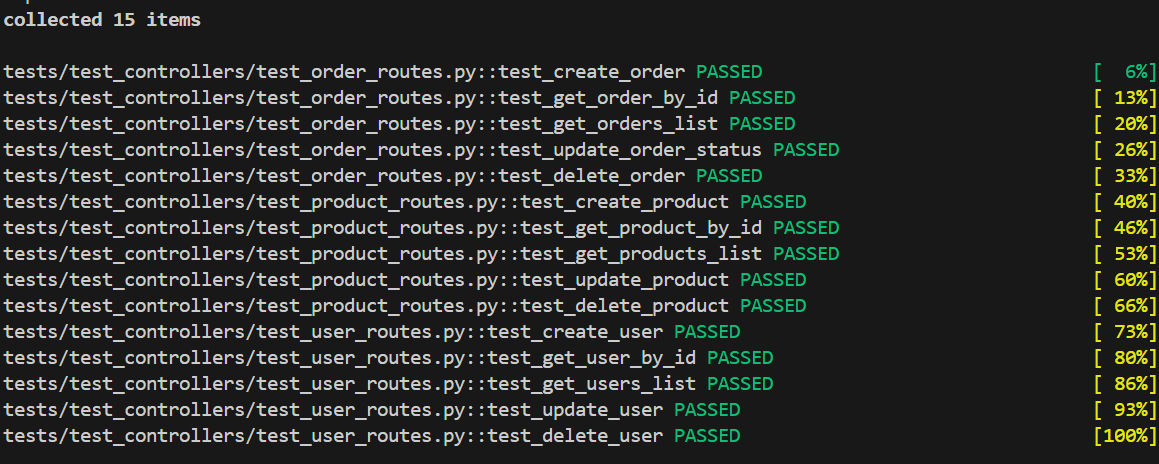


Только unit-тесты





Только API тесты



Вывод: В ходе лабораторной работы были написаны тесты для проверки каждой функции приложения. С помощью pytest протестирован бэкенд: проверена работа с базой данных, логика приложения и API, включая заказы с несколькими товарами. Использованы моки и временная база данных, чтобы тесты не мешали друг другу.

1. Тестовая база не портит реальные данные и делает тесты предсказуемыми. Production-база может быть изменена или повреждена.
2. TestClient имитирует запросы без запуска сервера. Это быстрее и надёжнее обычных HTTP-запросов.
3. Проверяй заказы с пустыми, несуществующими или некорректными товарами. Также — дубли, отрицательные количества, превышение лимитов.
4. Замени отправку email на mock и проверь, вызывается ли она при смене статуса на "shipped". Убедись, что email не отправляется при других статусах.
5. Проверяй, что ответ содержит правильное число товаров, ссылки на следующую/предыдущую страницу и корректные offset/limit. Убедись, что пустые страницы обрабатываются.
6. Используй изолированную базу и откатывай изменения после каждого теста. Это нужно, чтобы тесты не влияли друг на друга и работали стабильно.