Практическая работа.
Первый CI-конвейер для Микросервисного приложения.

Для начала создадим Монорепозиторий (Monorepo), где живут все микросервисы, и настроим GitHub Actions так, чтобы он проверял ***качество кода (Linting)*** и ***работоспособность (Testing)*** для всей системы. Организуем код микросервисов в репозиторий и настроим автоматическую проверку качества (CI) в облаке GitHub, используя Python и Git.

Необходимые инструменты:

- **PyCharm** (для написания кода и работы с Git).
- **Python** (установлен локально).
- **Аккаунт на GitHub**.
- **Исходные файлы** (текстовые файлы, которые я раздам: 1-products_service.txt и т.д.).

Этап 1: Подготовка монорепы (Local)

В реальной жизни микросервисы часто хранят в разных репозиториях. Но для обучения мы сложим их в один проект, чтобы видеть всю картину целиком.

1. **Создайте проект в PyCharm.** Назовите его microshop-monorepo.
2. **Создайте структуру папок.** Внутри проекта создайте 4 папки для бэкенда:
    a. products_service
    b. orders_service
    c. customers_service
    d. api_gateway
3. **Заполните кодом:**
    a. В каждой папке создайте файл app.py.
    b. Скопируйте в них код из соответствующих текстовых файлов из ПЗ про микросервисы и API Gateway (1-products..., 4-api-gateway... и т.д.).
4. **Создание зависимостей.** В корне проекта (не в папках сервисов!) создайте общий файл requirements.txt, пропишите в нем: **Flask Requests Flake8 pytest**
5. **Установка:** Откройте терминал PyCharm и выполните:
    pip install -r requirements.txt

```
microshop-monorepo/              <-- Корневая папка вашего проекта
│
├── .github/                     <-- Служебная папка (точка в начале важна!)
│   └── workflows/               <-- Папка для сценариев
│       └── ci_pipeline.yml      <-- Файл конфигурации нашего CI-конвейера
│
├── api_gateway/                 <-- Папка микросервиса "Шлюз"
│   └── app.py                   <-- Код приложения (из 4-api-gateway.txt)
│
├── customers_service/           <-- Папка микросервиса "Клиенты"
│   └── app.py                   <-- Код приложения (из 3-customers...)
│
├── orders_service/              <-- Папка микросервиса "Заказы"
│   └── app.py                   <-- Код приложения (из 2-orders...)
│
├── products_service/            <-- Папка микросервиса "Товары"
│   └── app.py                   <-- Код приложения (из 1-products...)
│
├── venv/                        <-- Виртуальное окружение
│                                    (эту папку мы НЕ трогаем и НЕ создаем вручную)
│
├── .gitignore                   <-- Правила исключений для Git
├── requirements.txt             <-- Список библиотек (Flask, Pytest, Flake8)
└── test_products.py             <-- Файл с нашим юнит-тестом
```

Этап 2: Написание Теста (Unit Test)

У нас есть код сервисов, но нет тестов. CI/CD бесполезен без тестов. Напишем простой тест для сервиса продуктов.

1. Создайте файл test_products.py в корне проекта.
2. Вставьте код из скрипта 1_test.txt этот тест проверяет, что функция получения продуктов возвращает список.
3. **Локальная проверка.** Запустите в терминале pytest. Если видите зеленый цвет, то можно переходить к облаку в гитхаб.

Этап 3: Настройка Git и публикация

1. Создайте файл .gitignore в корне, чтобы не отправить лишнее и добавьте в него:

Venv/

.idea/

__pycache__/

***.pyc**

2. Инициализируйте репозиторий и отправьте на GitHub:
    a. На верхней панели инструментов найдите VCS >> Enable Version Control Integration >> Git.
    b. Сделайте **Commit** всех файлов.
    c. Сделайте **Push** в ваш пустой репозиторий на GitHub (создайте его на сайте заранее).

Этап 4: Магия CI (GitHub Actions)

Теперь самое главное. Нужно задать инструкцию GitHub: "Каждый раз, когда я присылаю код, выдели мне компьютер (Runner), установи туда Python и проверь мой код".

1. В PyCharm в корне проекта создайте структуру папок: .github/workflows (точка в начале важна).
2. Внутри создайте файл ci_pipeline.yml.
3. Вставьте YAML конфигурацию из файла 2_yaml.txt.
4. Сделайте **Commit** и **Push** этого файла.

Этап 5: Проверка и "Слом" системы

1. Перейдите на GitHub во вкладку **Actions**.
2. Убедитесь, что ваш пайплайн запустился и прошел успешно (зеленая галочка).
3. **Задание на внимательность:**
    a. Откройте products_service/app.py в PyCharm.
    b. Сделайте синтаксическую ошибку (например, удалите двоеточие: после def get_products()).
    c. Сделайте **Commit** и **Push**.
    d. Посмотрите на GitHub. Пайплайн должен **упасть** (красный крестик) на этапе Lint with Flake8.
    e. Вам придет уведомление на почту, что вы "сломали билд". Это и есть суть CI в мгновенной обратной связи.
4. Исправьте ошибку и запушьте снова.