Министерство Образования, Культуры и Исследований Республики Молдова

Технический Университет Молдовы

Факультет Вычислительной Техники, Информатики и Микроэлектроники

Департамент Программной Инженерии и Автоматики

**Лабораторная Работа №2**

по Сетевому Программированию

Тема: «**Создание и взаимодействие двух REST API приложений.»**

Выполнил Cara Natalia

ст. гр. SI-222

Проверил

Макеев Никита

Кишинев – 2025

## Цель работы.

Понять принципы REST и научиться создавать приложения, умеющие взаимодействовать друг с другом через HTTP.

## Теоретическая часть.

1. **REST: основные принципы, методы, форматы запросов и ответов (JSON/XML)**

REST (Representational State Transfer) — это архитектурный стиль для создания веб-сервисов, основанный на стандартах HTTP. Основные принципы REST включают: клиент-серверную архитектуру, отсутствие состояния (каждый запрос содержит всю необходимую информацию для его обработки), кэширование, единообразие интерфейса и многоуровневую систему. REST использует стандартные HTTP-методы, такие как GET (получение данных), POST (создание данных), PUT (обновление данных) и DELETE (удаление данных). Данные передаются в форматах JSON (JavaScript Object Notation) или XML (eXtensible Markup Language), где JSON является более популярным благодаря своей простоте и легкости в использовании.

2. **Коды ответов HTTP, заголовки**

HTTP-коды ответов используются для информирования клиента о результате выполнения запроса. Основные категории кодов: 1xx (информационные), 2xx (успешные, например, 200 OK), 3xx (перенаправления), 4xx (ошибки клиента, например, 404 Not Found) и 5xx (ошибки сервера, например, 500 Internal Server Error). Заголовки HTTP (headers) содержат метаданные запроса или ответа, такие как Content-Type (тип данных, например, application/json), Authorization (данные для аутентификации) и Cache-Control (управление кэшированием). Заголовки помогают управлять поведением клиента и сервера.

3. **Преимущества микросервисной архитектуры**

Микросервисная архитектура предполагает разделение приложения на небольшие, независимые сервисы, каждый из которых выполняет одну бизнес-функцию. Основные преимущества включают: масштабируемость (каждый сервис можно масштабировать независимо), гибкость (возможность использовать разные технологии для разных сервисов), упрощение разработки и тестирования (каждый сервис разрабатывается и тестируется отдельно), устойчивость к сбоям (ошибка в одном сервисе не влияет на работу других) и независимость команд (разные команды могут работать над разными сервисами одновременно). Это делает микросервисы идеальными для сложных и крупномасштабных приложений.

## Ход работы.

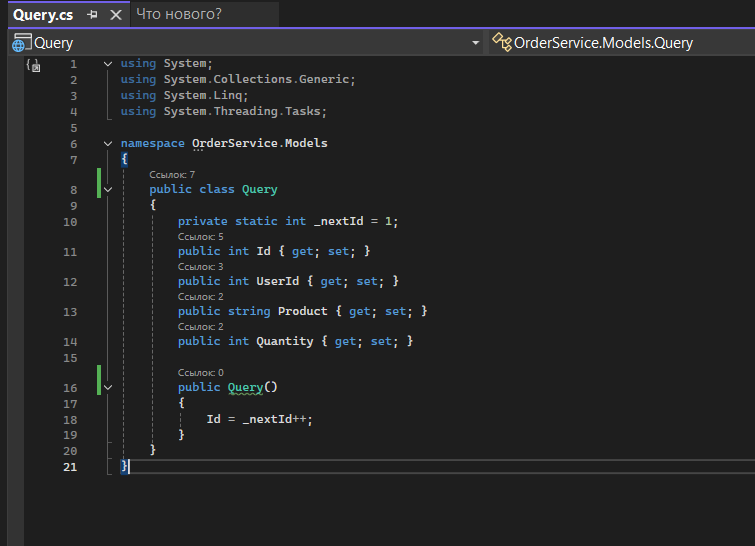
1. Создание первого REST API приложения.

Для создания приложения был выбран ASP.Net Core.

Простое веб-приложение создаётся с помощью команды:

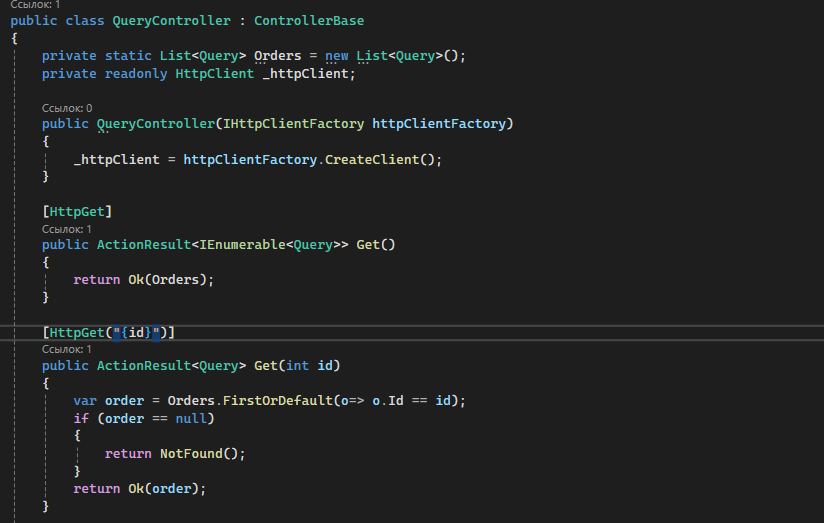
dotnet new webapi -n UserService.

Следующий шаг – создание модели.

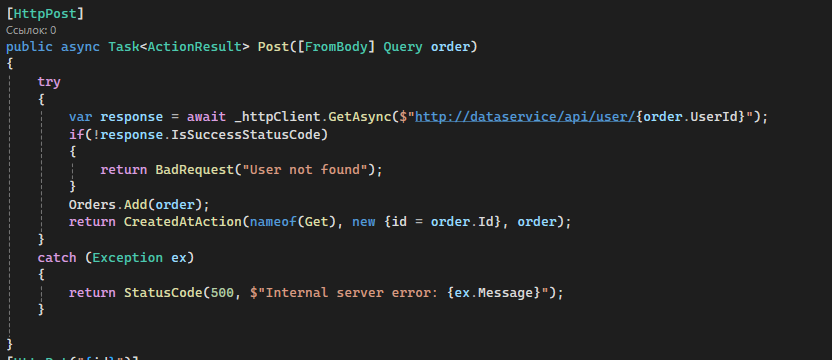


Дальше я создаю UserController в папке Controllers.

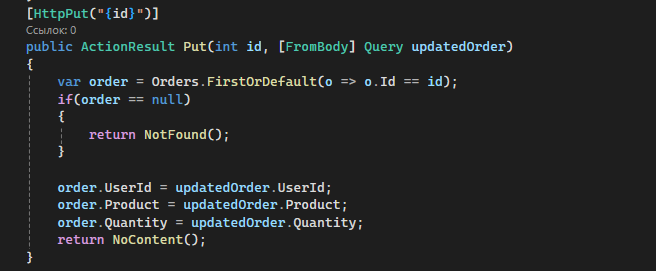
Код для команды GET



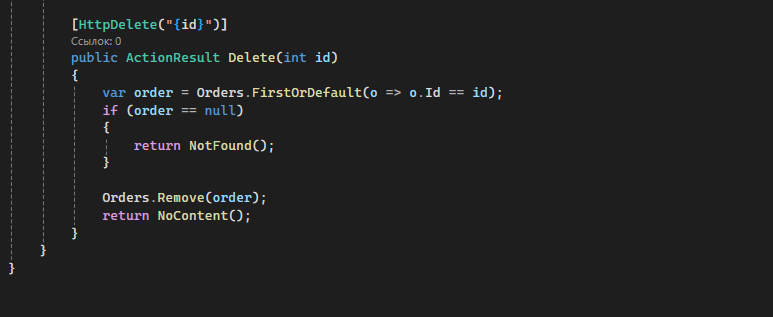
Код для команды POST



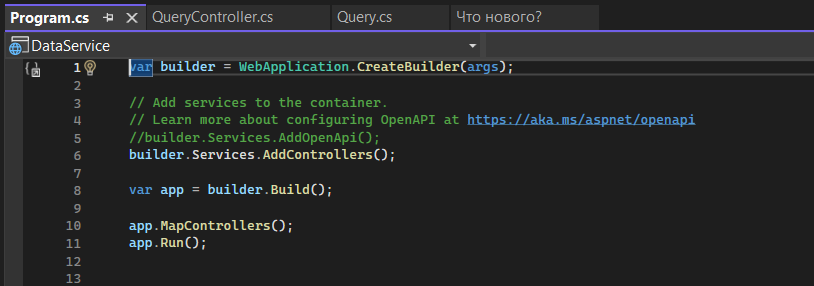
Код для команды PUT



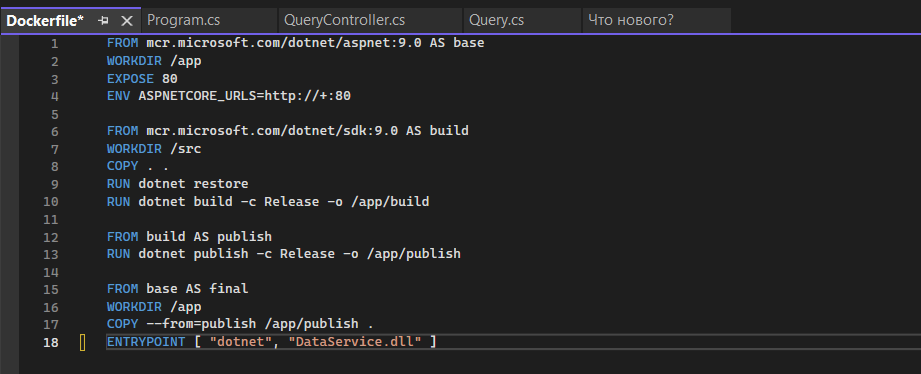
Код для команды DELETE



Создание билдера и подключение контроллеров



Написание Dockerfile.



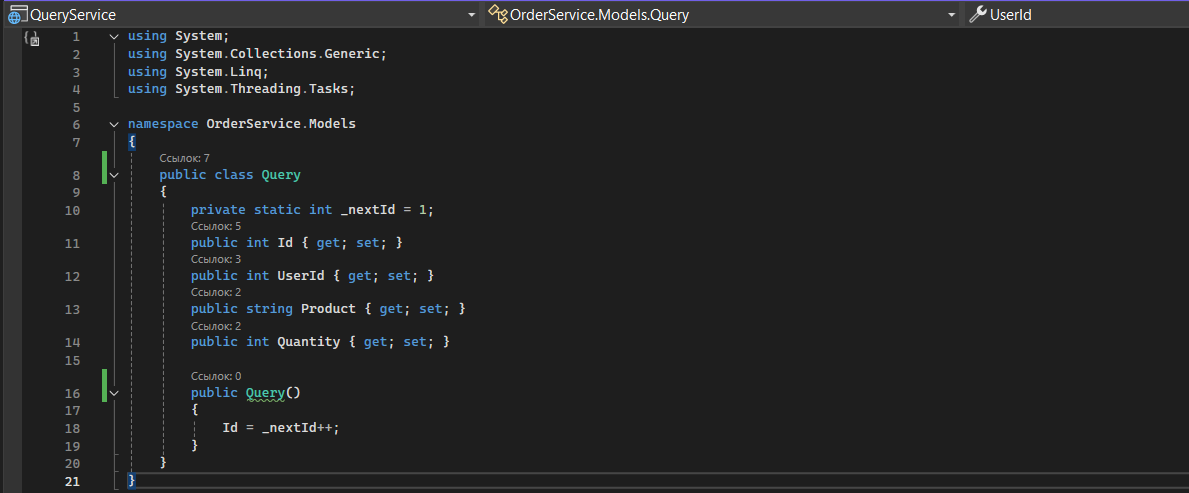
1. Создание второго REST API приложения.

Для создания приложения был выбран ASP.Net Core.

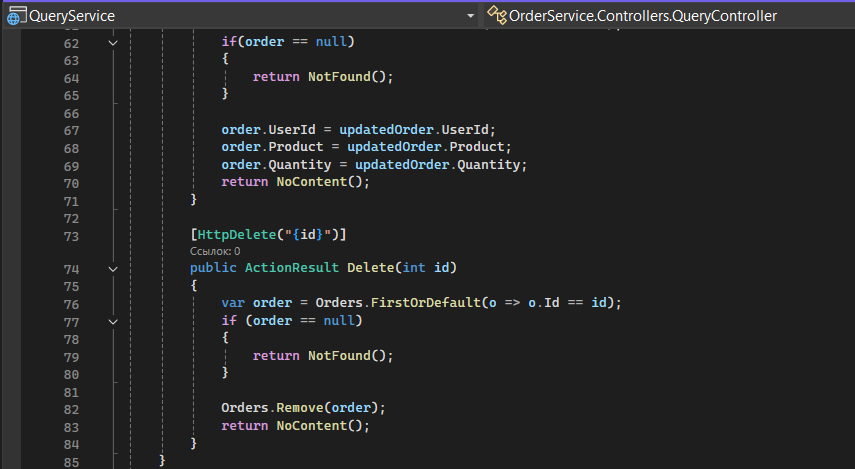
Простое веб-приложение создаётся с помощью команды:

dotnet new webapi -n OrderService.

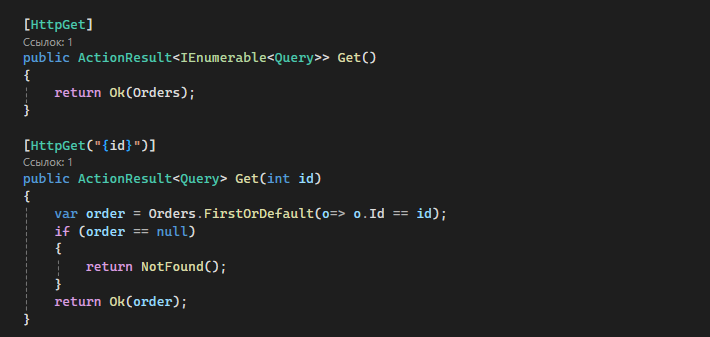
Следующий шаг – создание модели.



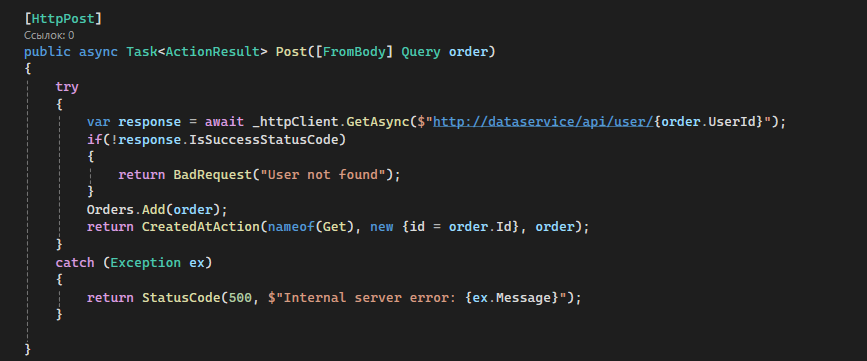
Создание QueryController в папке Controllers



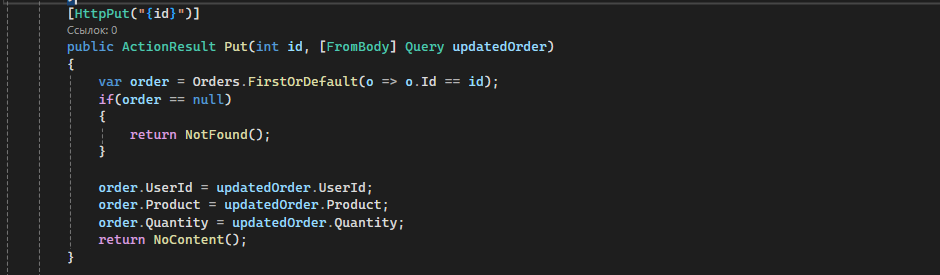
Код для команды GET



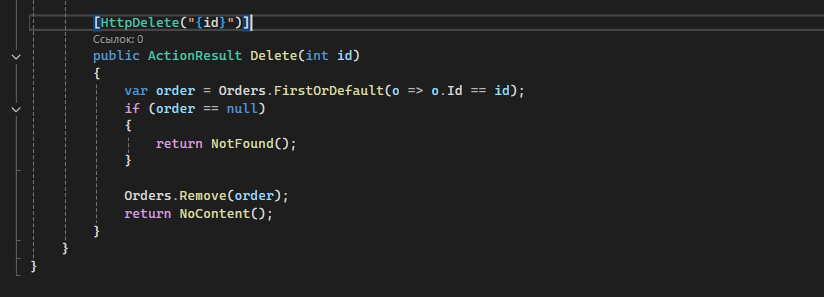
Код для команды POST



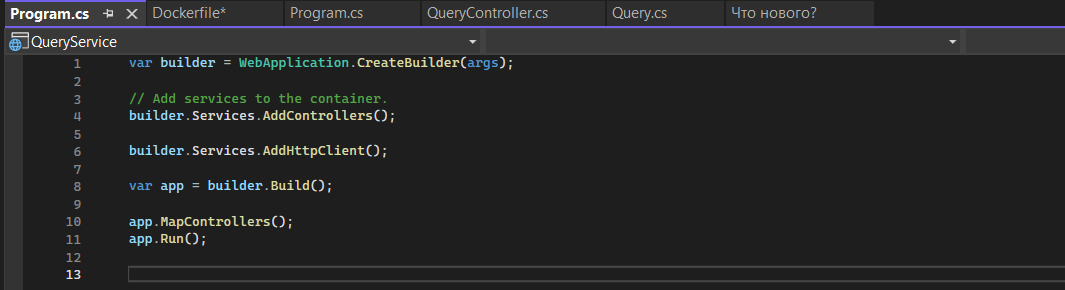
Код для команды PUT



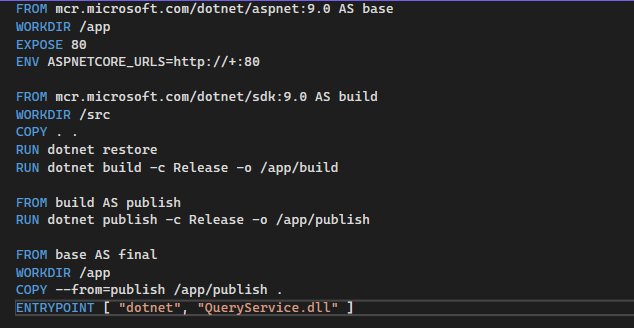
Код для команды DELETE



Создание билдера и подключение контроллеров

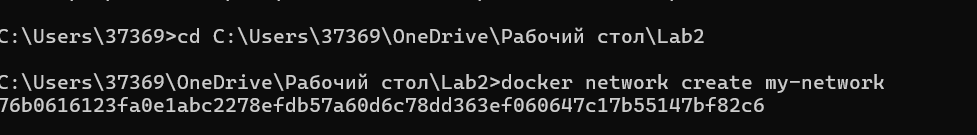


Написание Dockerfile

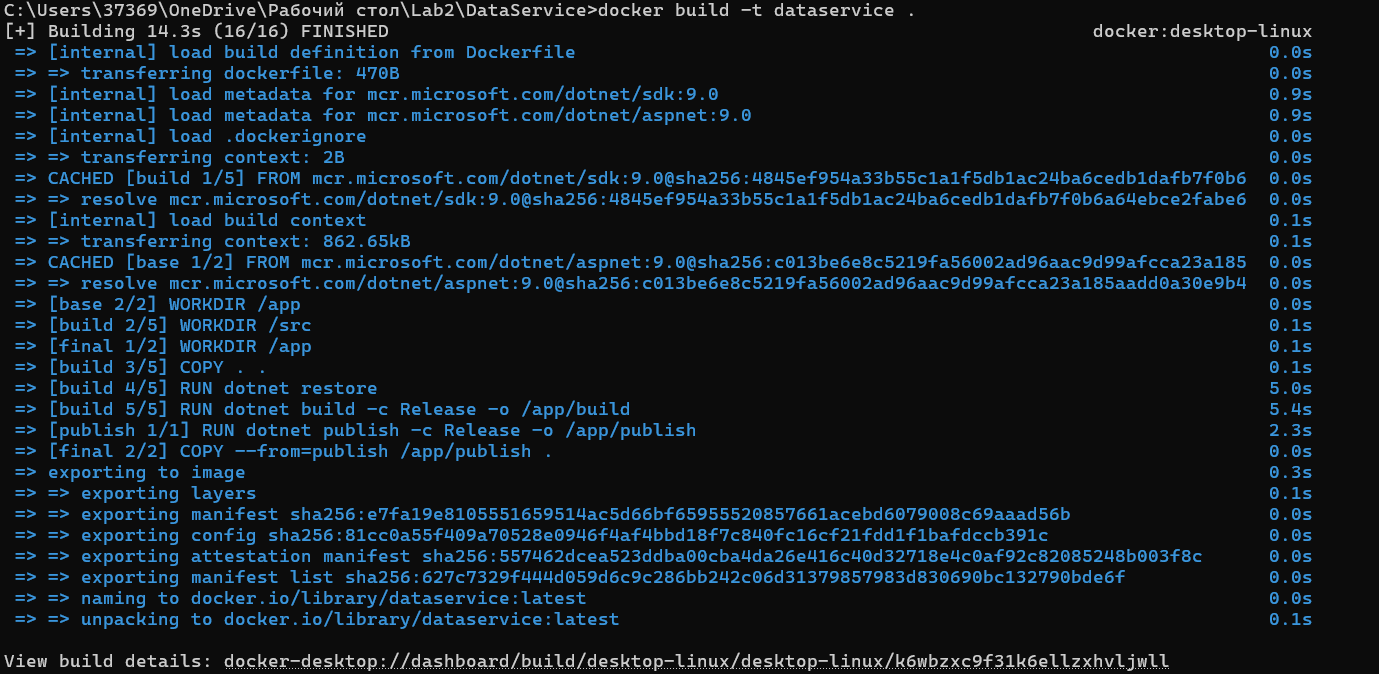


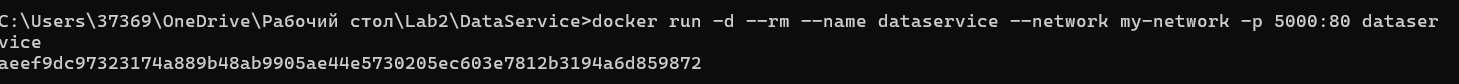
1. Запуск приложений

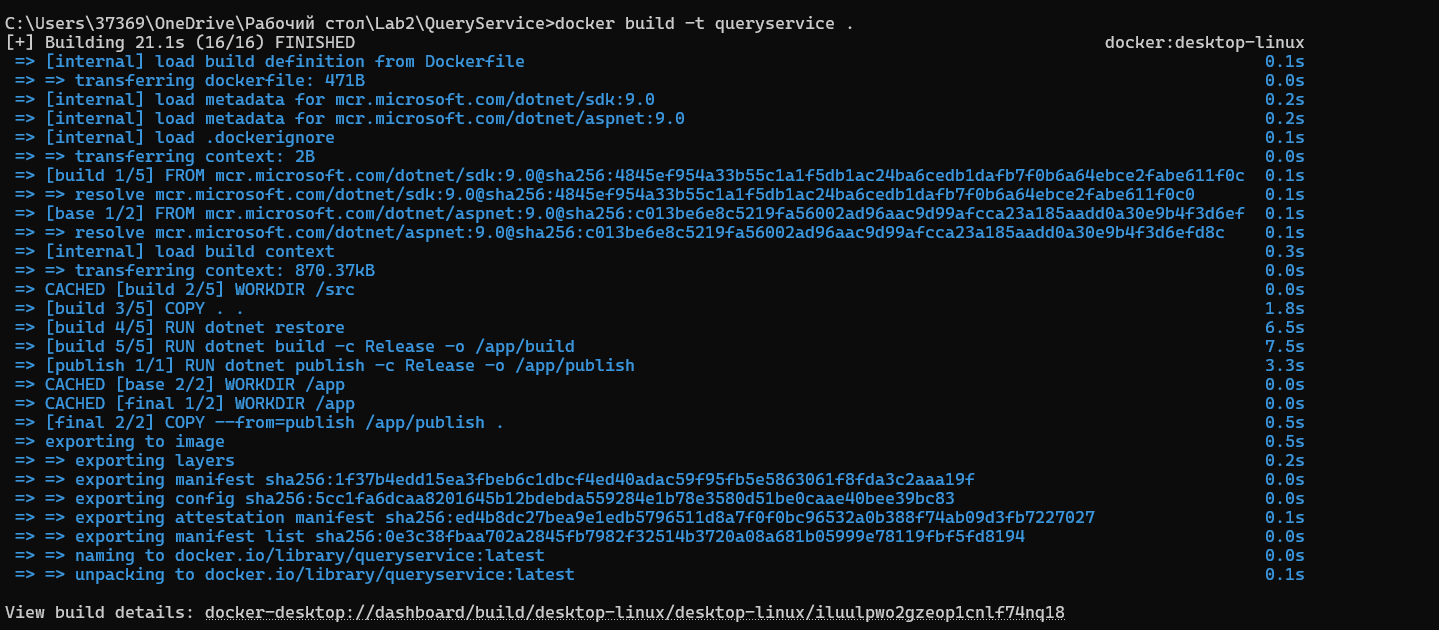
Создание сети



Создание образов и контейнеров приложений

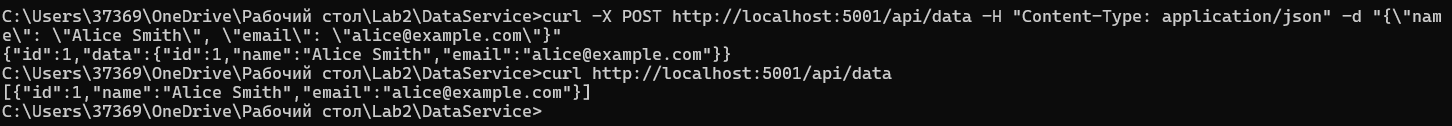


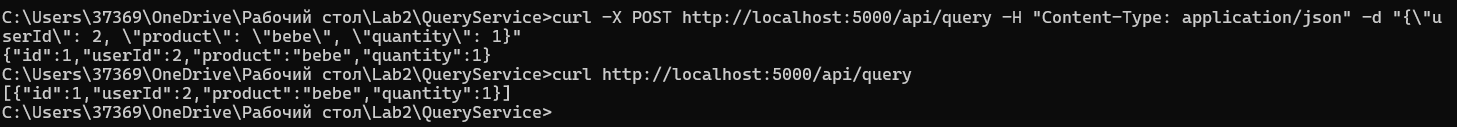




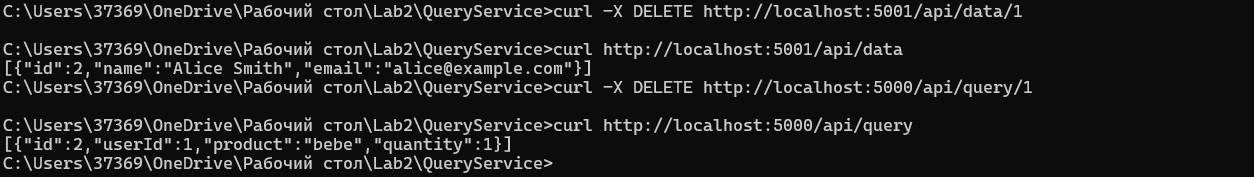


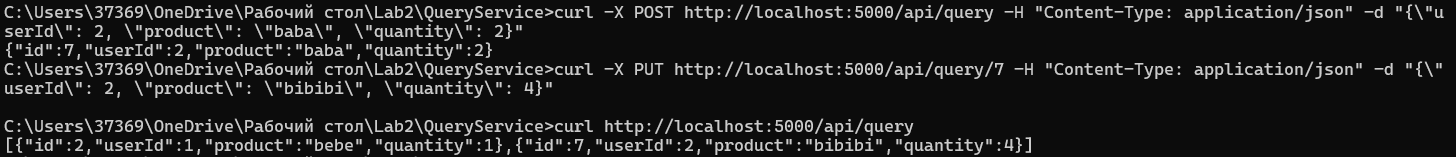
Запросы к приложениям через команду curl











## Выводы.

В ходе выполнения лабораторной работы я успешно создал и протестировал два REST API приложения (UserService и OrderService), которые взаимодействуют между собой по протоколу HTTP. Я реализовал основные HTTP-методы (GET, POST, PUT, DELETE), что позволило мне глубже понять принципы REST, включая единообразие интерфейса, отсутствие состояния и использование форматов JSON для обмена данными. Оба приложения я упаковал в Docker-контейнеры, что помогло мне оценить преимущества микросервисной архитектуры, такие как независимость сервисов, масштабируемость и упрощение разработки. В результате я получил практические навыки создания и взаимодействия RESTful сервисов, а также их тестирования с использованием инструментов curl и Postman.

## Приложение.

<https://github.com/MihailBancov/Lab2_PR>