

Лабораторная работа №6

Использование архитектурных решений для обеспечения качества ПО

1 Цель работы

1.1 Изучить процесс разработки приложений с использованием микросервисной архитектуры.

1.2 Изучить процесс развертывания приложений с использованием микросервисной архитектуры.

2 Литература

2.1 Зверева В. П., Сопровождение и обслуживание программного обеспечения компьютерных систем : учебник для студ. учреждений сред. проф. Образования / В. П. Зверева, А. В. Назаров. – М. : Издательский центр «Академия», 2018. – 256 с.

3 Подготовка к работе

3.1 Повторить теоретический материал (см. п.2).

3.2 Изучить описание лабораторной работы.

4 Основное оборудование

4.1 Персональный компьютер.

5 Задание

5.1 Разработка API

5.1.1 Разработать 2 отдельных api-сервиса со следующими методами

ProductService:

GetProducts

GetProduct(id)

CreateProduct(Product)

OrderService:

GetOrders

GetOrder(id)

CreateOrder(Order)

DeleteOrder(id)

5.1.2 Разработать модели и контекст БД для данных api-сервисов

В качестве БД использовать MySQL, а строку подключения получать из переменной окружения.

```
services.AddDbContext<ProductServiceDbContext>(options  
=>  
options.UseMySQL(Environment.GetEnvironmentVariable("CONNECTION_STRING")));
```

5.1.3 Настроить автоматическую миграцию для контекста БД

5.2 Разработка API-шлюза

5.2.1 Добавить новый проект типа Web-API, удалить стандартный код.

5.2.2 Установить в проекте nuget-пакет Ocelot

5.2.3 При помощи Ocelot добавить в Program.cs api-шлюз, который будет выполнять роль посредника для данных api-сервисов.

```
var builder = WebApplication.CreateBuilder(args);

builder.Configuration.AddJsonFile("ocelot.json",
optional: false, reloadOnChange: true);
builder.Services.AddOcelot();

var app = builder.Build();

app.UseRouting();
app.MapControllers();
app.UseOcelot().Wait();

app.Run();
```

5.3 Указать настройки Api-шлюза в файле ocelot.json

```
{
  "Routes": [
    {
      "DownstreamPathTemplate":
"/api/products/{everything}",
      "DownstreamHostAndPorts": [
        {
          "Host": "productservice",
          "Port": 8080
        }
      ],
      "UpstreamPathTemplate": "/products/{everything}",
      "UpstreamHttpMethod": [ "Get", "Post", "Put",
"Delete" ]
    }
    //то же для сервиса заказов
  ],
  "GlobalConfiguration": {
    "BaseUrl": "http://localhost:5000"
  }
}
```

5.4 Настроить решение для работы с контейнерами.

5.4.1 В контекстном меню обозревателя решений во всех проектах

выбрать опцию Добавить > Поддержка оркестратора контейнеров

5.4.2 Добавить в проект docker-compose.yml

5.4.3 Получившаяся структура решения должна выглядеть следующим образом:

```
Решение/  
  ApiGateway/  
  ProductService/  
  OrderService/  
  docker-compose.yml
```

5.4.4 В файл docker-compose вписать конфигурацию для развертывания разработанной системы. В листинге представлена часть конфигурации для ProductService и ApiGateway. Самостоятельно добавьте сервис заказов в настройки файла.

```
services:  
  apigateway:  
    container_name: "apigateway"  
    image: ${DOCKER_REGISTRY-}apigateway  
    build:  
      context: .  
      dockerfile: ApiGateway/Dockerfile  
    environment:  
      - ASPNETCORE_ENVIRONMENT=Development  
    ports:  
      - "5000:8080"  
    depends_on:  
      - productservice  
    links:  
      - productservice  
    networks:  
      - net  
  
  productservice:  
    container_name: "productservice"  
    image: ${DOCKER_REGISTRY-}productservice  
    build:  
      context: .  
      dockerfile: ProductService/Dockerfile  
    environment:  
      - ASPNETCORE_ENVIRONMENT=Development  
      -  
      CONNECTION_STRING=Server=productdb;Database=Products;User=root;Password=root;Port=3306
```

```
depends_on:
  - productdb
links:
  - productdb
networks:
  - net

productdb:
  container_name: "productdb"
  image: mysql:latest
  restart: always
  environment:
    MYSQL_ROOT_PASSWORD: root
    MYSQL_DATABASE: Products
    MYSQL_USER: root
    MYSQL_PASSWORD: root
  volumes:
    - productdb:/var/lib/mysql
  networks:
    - net

volumes:
  productdb:

networks:
  net:
```

5.4.5 В Docker-файле ApiGateway укажите EXPOSE 5000 вместо 8080

5.4.6 Запустить docker-compose (sudo docker compose up -d) и проверить корректность работы API по адресу <http://localhost:5000/products/>.

6 Порядок выполнения работы

- 6.1 Повторить теоретический материал п. 3.1;
- 6.2 Выполнить задания 5.1-5.4
- 6.3 Ответить на контрольные вопросы п. 8;
- 6.4 Заполнить отчет п. 7.

7 Содержание отчета

- 7.1 Титульный лист;
- 7.2 Цель работы;
- 7.3 Ответы на контрольные вопросы п. 6.3;
- 7.4 Вывод по проделанной работе.

8 Контрольные вопросы

8.1 Какие преимущества и недостатки существуют у микросервисной архитектуры