**Лабораторная работа №3**

В лабораторной работе №3 необходимо было разбить монолитный сервис на несколько микросервисов. Во-первых, нужно было разделить данные по разным базам. Созданы 3 базы: 1)База с таблицей Customers и Deliveries, 2)База с таблицей Goods и 3)база с таблицей Orders.

1)База с таблицей Customers и Deliveries:

Таблица Customers:

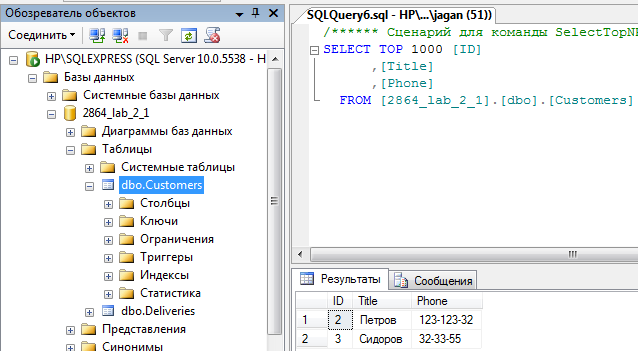
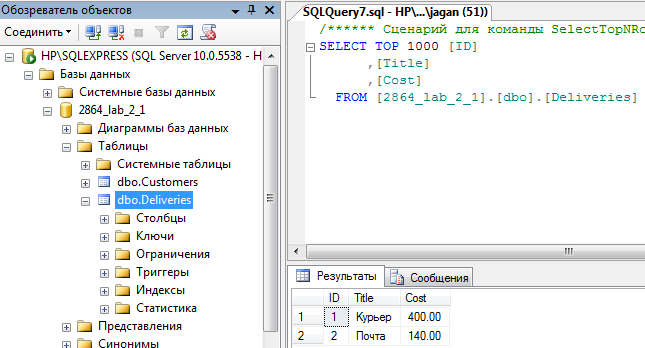
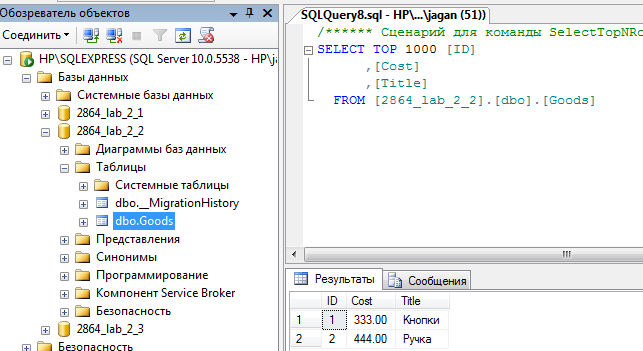


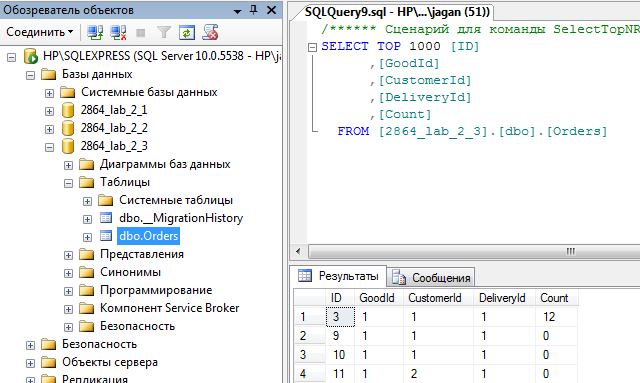
Таблица Deliveries:



2)База с таблицей Goods:



3)База с таблицей Orders:



Во-вторых, созданы 3 микросервиса для работы с этими базами. Для реализации микросервисов применена технология WCF сервиса с возможностью самостоятельно хостинга без IIS. За это отвечает приложение RunService.

На примере GoodService опишем принцип работы.

Сделаем сервис

[ServiceContract]

[ServiceBehavior(IncludeExceptionDetailInFaults = true)]

public class GoodsService

{

[OperationContract]

public string GetById(int id)

{

using (GoodsContext context = new GoodsContext())

{

return JsonConvert.SerializeObject(context.Goods.Where(x => x.ID == id).FirstOrDefault());

}

}

[OperationContract]

public string GetPage(int page, int size)

{

using (GoodsContext context = new GoodsContext())

{

return JsonConvert.SerializeObject(context.Goods.OrderBy(x => x.ID).Skip((page - 1) \* size)

.Take(size).ToList());

}

}

}

Данный сервис можно запустить командами

ServiceHost host = new ServiceHost(type, baseAddress);

ServiceMetadataBehavior smb = new ServiceMetadataBehavior();

smb.HttpGetEnabled = true;

smb.MetadataExporter.PolicyVersion = PolicyVersion.Policy15;

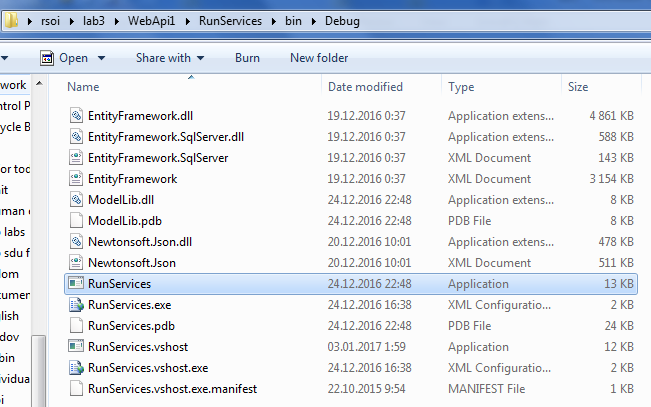
host.Description.Behaviors.Add(smb);

host.Open();

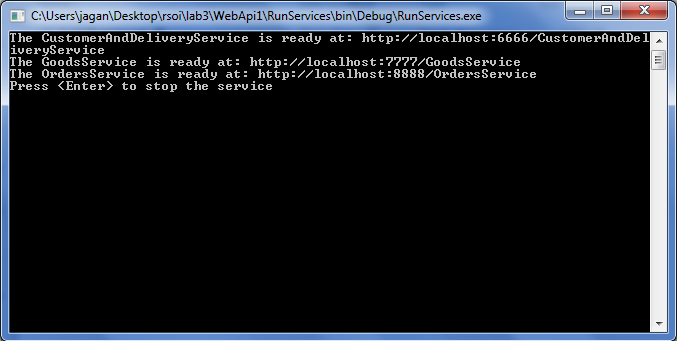
Это позволит обращаться к этому сервису по tcp/ip протоколу.

Консольное приложение содержит 3 микросервиса и после запуска три сервиса становяться активным и доступными для работы (запускать нужно под именем администратора). Внизу показано поэтапно как запускать эти микросервисы:

1-шаг: находим RunServices.exe



2-шаг: запускаем его под именем администратора



Наверху на рисунке показано что после запуска микросервисов в командном строке окна запустились три микросервисов:

1)Customer and Delivery сервис на localhost:6666

2)Goods сервис на localhost:7777

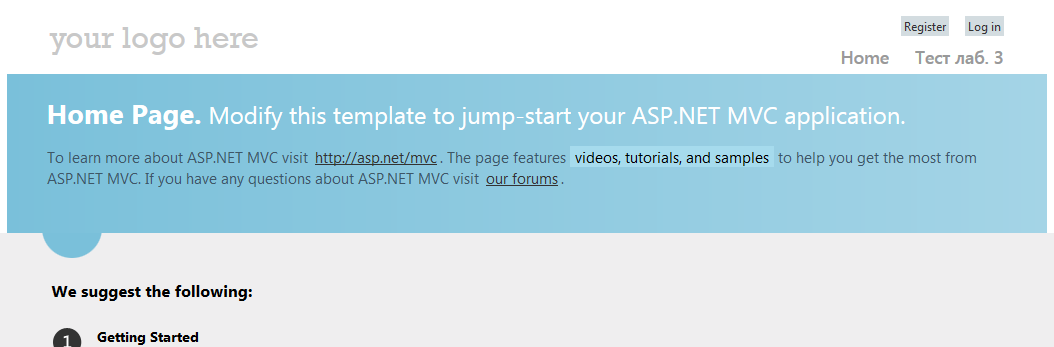
3)Orders сервис на localhost:8888

По аналогии с классом MainControllers.cs сделаем класс MainControllers2.cs который будет содержать API для работы с микросервисами. Внешне набор API методов не изменился (только добавилась двойка например api/orders2).

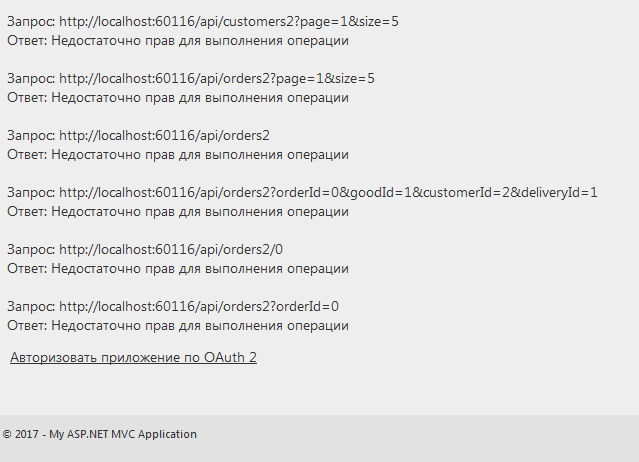
Сервисы подключаются через диалог AddServiceReference. По wsdl генерируется клиентские классы, через которые удобно обращаться к методам сервисов.

Процедуры авторизации те же как в лабороторной работе-2, только страница для инициализации другая – contract и которая содержит всю логику тестирования.   
Давайте покажем авторизацию программу поэтапно:

**1-шаг:** запускаем программу и видим ссылку «Home» и «Тест лаб. 3»

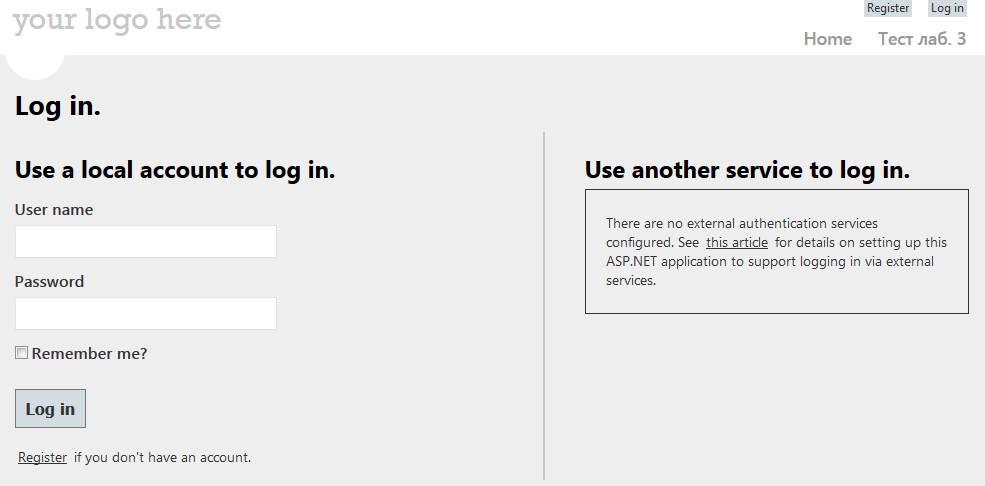


**2-шаг:** На нем нажимаем на ссылку «Тест лаб. 3» без авторизаций

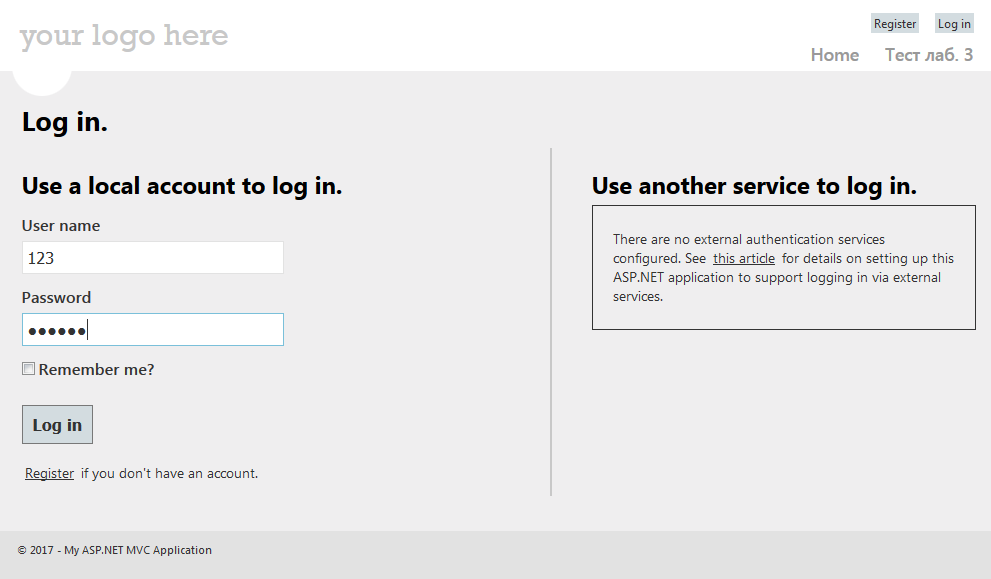


И мы видим что без авторизаций у нас не достаточно прав для выполнения операции, но на самом конце мы видим еще и ссылку «Авторизовать приложение по Oauth 2».

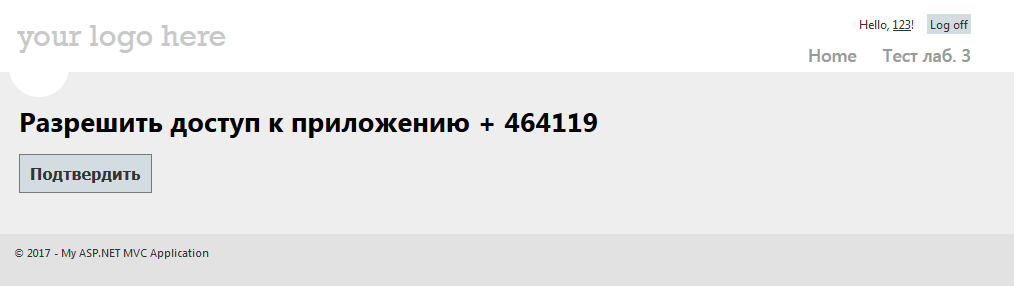
**3-шаг:** если нажать на ссылку «Авторизовать приложение по Oauth 2» то получаем:



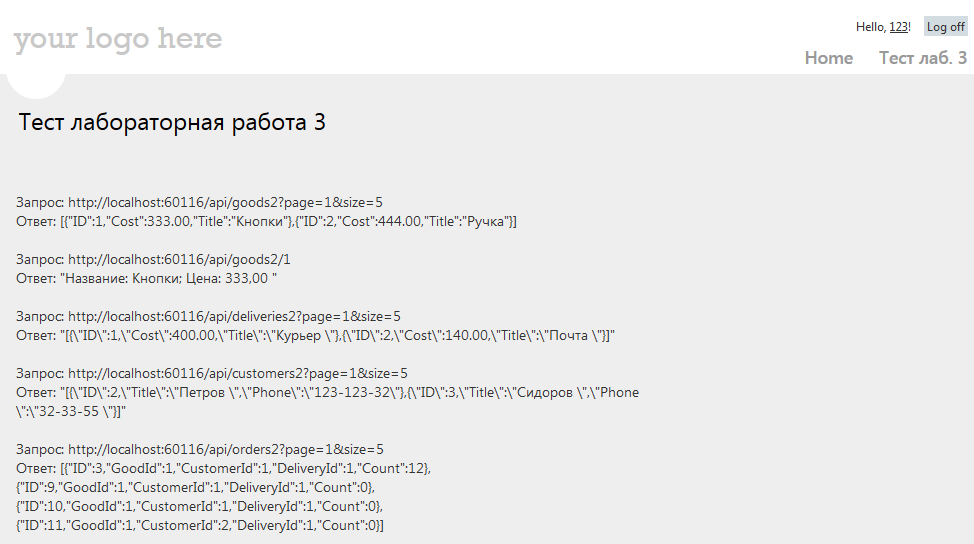
**4-шаг:** можно регистрироваться или если ранее регистрировались то можно и вести логин и пароль.



**5-шаг:** после того как ввели логин и пароль, нажимаем на log in:



**6-шаг:** нажимаем на подтверждение досупа:



Вот так мы запускаем тест для лабораторной работы-3 с авторизацией

Если посмотреть внимательно на содержимое, то видно отличие по данным, т.к. данные для лабораторной работы №2 и №3 берутся из разных источников.

**Соответствие требованиям**

1. Данные каждого сервиса можно хранить как в SQL, так и в

NoSQLбазе. Для упрощения допускается хранить данные на одной базе, но в разных схемах. При этом каждый сервис должен взаимодействовать только со своей схемой, получение данных, не относящихся к текущему сервису строго запрещено.

3 базы MS SQL и каждый микросервис общается со своей базой.

2. Нельзя использовать готовые библиотеки для авторизации по

OAuth2.0.

Сделано в ручную

3. Для токена нужно реализовать время жизни (expires) и обновление токена (через refresh token).

Сделано HomeController/AuthMethod

4. Должен быть хотя бы один запрос, требующий агрегированной информации от двух и более сервисов.

Orders2Controller метод Get(id)

5. Все взаимодействие между сервисами выполнить в парадигме

RESTful.

Да, есть.

6. Предусмотреть работу системы в случае отказа одного из компонентов системы.

В случает отказа микросервиса данные из других сервисов будут читаться.

7. При получении списков данных предусмотреть пагинацию.

Есть

8. Сделать подробное логгирование выполняемых действий н

а каждом сервисе.

Есть. В конфиге Nlog.config в каждом проекте указан файл в который пишутся конфиги. Сейчас он везде один – C:\log.txt.

В него пишут все микрофреймворки выполняя методы.

9. Подготовить шаблоны запросов или маленький скрипт для демонстрации работы.

Есть тестовая веб страница