**REST API**

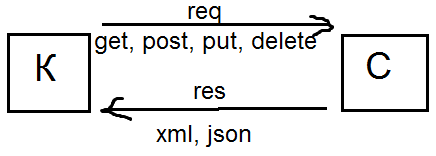
**REST:** Representational State Transfer – передача сост. предст-я  
предст – uri, упр. ресурсом с пом глаголов (GET, POST, PUT, DELETE)  
Альтернитива RESTу – RPC (набор удал процедур), оба стиля развив и начинают сливаться

**REST** – это интерфейс веб-сервисов; это архитектура SOA-приложения; это набор URI, который представляет ресурс, к которому можно обращаться с запросом.

**REST-FULL** – описание веб-службы, поддерж rest-ин-с в полном объеме. Сегодня нет офиц. стандарта REST (это минус). У RPC – SOAP (на практике он исп. и для REST)

**HTTP, URI, XML, JSON** – на эти стандарты REST опир-ся  
\* модель КС  
\* нет состояния на С (жц «запрос-ответ»)  
\* сохр. сост допуск на К или в рамках др. С (кот. обнов БД)  
\* кеширование т. на стор К и упр-е кешем с пом Са (он явно управляет сохр-ем сост на К: cookie, control-cash)   
**WEB-API** – м. закешировать, но это не сочет с треб-ями REST API

\* единообразие ин-сов, какой-то принцип д/разраб URI, чтобы юзеры не путались  
\* для К, С должен казаться конечным, мы не д. знать, что он к кому-то обращ (специф ATIL как организ IT-предпр) к др. сервисам  
\* допуск, что на сторону К мб выгружен код

   
Сервис дб предст Клиенту как храниище, где м. доб, уд…

**Общепринятые правила (HATEOAS):**  
\* общ. префикс /api/… лучше цел. домен api.belstu.by  
\* 2 типа рес: коллекц (users), эл-т коллекции (users/238)  
\* иерарх связь ../api/users/238/cards/AB1998 🡨 запрос исп. 2 коллекции  
\* ограничить кол-во http-статусов, сопроводить сообщ доп кодом (напр. 2003)  
сделать отд. ресурс (HATEOAS link) для пояснения ошибок http://ccc/api/errors/20003  
\* подавление статуса ответа ../api/students/ef3d26?status\_code=200  
\* версионность /api/students/ef3d26?v=7

\* 2 параметра (постраничное получ. д-х): limit – огранич ск. вывести, offset – смещ отн начала  
\* сортировка sort: фильтры после ‘?’ gender=m  
\* json или xml дб по умолч, 2й – по параметру, обознач в запросе формат сообщ-й  
\* обозначать в запросе формат сообщ: Accept  
\* глоб поиск: ../api/search?q=196+Иван. Берется коллекция, все эл-ты конкатен в 1 строку и выбир т. те элем коллекции, в кот. есть строка ..find. Если нах -> соотв. элем выбир

**Недостатки:**  
\* нет общепризн. стандарта RESTful API  
\* не все браузеры поддерж словарь REST-методов (PUT, DELETE). На практике исп. т. GET, POST (insert, delete, update)  
\* не однозначны коды сост

**ЛЕКЦИЯ 2. ASMX**

**WSDL** – Web Services Definition Language – язык описания сервисов.

Есть раздел type который описывает типы передаваемых данных и результат. Все типы делятся на 2 вида: простые(int, float..) и композитные ()

Message – сообщения между К и С

PortType – методы

Binding – связывает с транспортным протоколом

Появился новый стандарт – WSDL 2.0.

* Новая секция interface

asmx – частный случай wcf  
мы разраб прокси клиент, кот генерировала исх файл прокси исходя из wsdl (кот мы разработали)

Мы генерировали клиента (прокси) для этого использовали VS. Она работала, считывала WSDL и на исходный файл генерировала прокси.

WSDL.exe – утилита, которая вызывается вижлой для генерации прокси. Именно эта утилита вызывается и прокси генерируется. с ее пом можно генер сервис .Имея wsdl, м сгенеировать сервис (до этого мы генер только клиент)  
Откуда wsdl знает о логике работы?

Он не знает, что выполняет каждый метод, но знает какие параметры передаются и что возвращается. Эта утилита генерирует абстрактный класс, этот класс содержит абстрактные методы. Имея абстрактный класс, мы можем взять его за базовый для нашего сервера и реализовать эти методы. Т.е формально знает параметры, что возвр, и генер абстрактный класс, кот содержит абстр методы, кот формально имеют те параметры, кот описаны в wsdl, но нет реализации  
=> сгенерив абстр класс, мы можем взять его в качестве базового для нашего сервиса и просто реализовать эти методы

(на лабе будем исп ч/cmd. Есть 2 варика работать с абстр классом: 1) созд класс кот наследует => придется перетаскивать атрибуты в свой класс; 2) тупо переделать сгенерир класс в свой сервис: поубирать слово abstract методов и подсунуть реализацию = это все и для WCF, ASMX)

Как использовать службы wcf для работы с ajax?

**WCF**

это отд платформа для разработки приложений КС архитектуры на основе .NET-фреймворка

**WCF** – Windows Communication Fondation; технология основанная на .NET Framework для разработки SOA-архитектуры приложений;

Все взаимоотношения в WCF сводятся к отношениям К-С. Между клиентом и сервисом разраб хорошо продуманная архитектура взаимоотношений. Она состоит из **3 компонентов**:  
a) address – где нах компонент  
b) binding – как можно добраться  
c) contract – что эта конечная точка м делать (какие методы)

WCF может поддерживать множество точек доступа. Они отличаются транспортом, который обеспечиваются для доступа. Не может быть 2 точек, к которым транспортом является TCP. Нельзя организовать доступ через HTTP в полной мере, но можно по TCP. Напр, подписчик-издатель полностью по http не реализовать

Когда разрабатываем точки и они имеют разный транспорт, мы можем разработать собственный контракт (интерфейс, который описывает что точка может делать). Таким образом, у точек могут быть разные контракты доступа.

Коммуникация между К и С в WCF может описываться контрактами:

* Сервис-контракт – контракт службы, описывает что умеет делать эта точка – интерфейс c#, который содержит специальные атрибуты и наш сервис фактически – реализация этого контракта.

Сервис – класс, который реализует интерфейс контракта.

[OperationContract] – для того, чтобы данные были доступны из вне.

* Контракты данных – указывает каким образом данные будут сериализованы;

Описываем свойства, геттеры и сеттеры – помечаем их DataMember для того, чтобы данные были доступны извне. Если не пометить – данные будут доступны только внутри приложения.

На основании этого описания в WSDL будут внесены описания этих данных;

все типы данных – простые и композитные (класс). Если у нас сложная структурный тип данных с разными св-вами, мы должны создать *data contract*, пометить свойства как data member. На основании этого описания wsdl внесено описание этих данных

Если я все методы пометил как operation contract, на самом деле я могу создать ин-с которые не помечены так, но они будут не доступны извне, не видны в точке доступа, будет доступен толко изнутри приложения, то же самое с контрактом данных и data member

* Контракт сообщений

Высокий уровень – К и С общаются, получают данные; между К и С ходят сообщения. Мы, как разработчики на высоком уровне не понимаем что ходит между к и с и не можем управлять содержимым сообщений, которые ходят между К и С

сервис обесп методы, которые прописаны сервис-контрактом, понятно что между КС ходят сообщения. Мы как разработчики выс. уровня не знаем что в них находится. Не можем управлять на этом уровне содержимым этих сообщений. А message contract позволяет влезть в этот уровень, на уровне протокола soap добавить свою инфу, кот будет перемещаться вместе с обычной инфой. Контракт сообще – это тоже класс, декорир аннотациями messageHeader, messageBody. SOAP сообщ сост из envelope (header,body,…). Мы также должны обесп соотв проргаммный код, кот будет эти сообщения заполнять и их обраб

[MessageContract] - позволяют влезть в этот уровень и добавить в сообщения, которые ходят между К и С добавить свою инфу, которая будет перемешаться вместе с той инфой, которая ходит там.

если на более низком уровне

[MessageHeader] – записать свои собственные данные в хедер

[MessageBodyMember] – записать свои собственные данные в боди.

На стороне К и С нужен доп.программный код, который будет добавлять свои данные и обрабатывать их.

Контракт - соглашение между К и С о принципах взаимоотношений между 2 абонентами.

**Хост** – контейнер сервиса.

**Привязка** – инкапсулирует все технологии, позволяющие переправить сообщений от одной конечной точки к другой (транспорт, кодировк, протоколы (безопасность, сессия), правила обмена сообщениями (размер, таймаут)). Указыются для конечной точки.

Рассм взаимоотношения точки доступа на стороне К и сервиса подробнее. На wcf м разраб собс протоколы. Есть неск характеристик, опис взаимоотношения КС:  
\* обесп надежный обмен  
\* безопасность  
\* кодирование  
\* транспорт

Есть набор **привязок**, готовых для использ – указ для конечных точек (либо собственную): какой протокол исп, как будет кодироваться, на каком уровне обесп security, поддерж ли сессия, транзакции, дуплекс канал связи?

Когда будем создавать сервис с неск точакми, етсь еще точка доступа для получения метаданных (MEXT) по точкам доступа выношу в web.config, вместо этоо м с пом классов описать внутри кода, не исп-я web.config

На базе WCF есть много полезных продуктов – OData (Open Data) – веб-протокол, позв вып операции с ресурсами и получить данные формата xml, json: у нас есть источник д-х, к к-му м организовать доступ и получать рез-т в формате xml/json. Все запросы (и insert… даже ddl ! ) к этому ресурсу явл GET http-запросами и всё что мы хотим получить, описывается с пом параметров (с пом спец языка – по мощности равен SQL). Разработан OASIS-ом (они уже разработали кучу стандартов передачи д-х, не надо придумывать ничего нового). Чаще у нас удаленная sql-бд. Фактически он берет наш запрос на языке OData, конвертирует его в sql и делает запрос.

Есть 2 версии технологии data services: я делал на более старой, либо можно на базе MVC. Если вижла новая, надо доустапн элемент (wcf data services), все просто

Есть БД, для нее посмотрим EDMX-модель, для нее построил контекст :DbContext, построил OdataServices, в этом классе указал этот DbContext => у нас готовая служба  
м работать как с обычным REST-сервисом, исп-я язык запросов

Есть парам, указ формат в к-м хотим вернуть данные

**02.10 Микросервисы**

**Микросервисы** – компания Амазон. Надо было представить клиентам Амазона сервис для работы.  
**Осн. задача:** быстрое реагирование на потребности клиента  
1) быстро разраб. новые сервисы  
2) быстро переделывать сервисы, которые есть

**Микросервисы** – арх. подход, осн задача – увел. скорость разработки новых сервисов и переделки старых. Важна именно скорость

Неотрывная часть – **BIOS**. Дб написаны сразу все тесты, построена инфраструктура. Микросервисы, BIOS и команда – неразрывные части

**Микросервис** – средство продажи сервиса, а не программы. Он не явл прогр продуктом. Микросервисы – один из подходов к разработке. Должен быть легким! Маленькие и легковесные.   
**Микросервис** – прогр продукт, кот заново мб написан за 2 недели

Осн цель – продать, сделать устойчивым к сбоям  
Чтобы запустить на неск серверах одновременно, есть балансировщик, кот раскидывает нагрузку, микросервисы дб **автономные**. Отд сервис кот тоже может размножатьс и их хранилище реплицируют др в друга.   
Сервис дб автономный и независимый!

Игнорируется, на каком коде написано. Вычисления – лучше С++, обработка данных – лучше Jango. Позволяют сделать работу программистов эффективнее. «Спринты» программистов и за 2 недели выпускают продукт

Микросервис – это сервис, кот вып 1 элементарную ф-ю. В сервисе скрывается и бэк и фронт, обычно фронт это single page прил. Например, микросервис кот принимает заявку на кредит. 1 функция. Его изменение не д затрагивать другие сервисы – независимость

Есть готовые фреймворки – **нэти** – позв создавать модульную стуктуру, кот м наращиваться (.NET)

На самом деле работает не само наше приложение, а егок копия (iis). Если делаем новую публикацию, то если есть еще подключенный клиент, он еще со старой версией работает. Остальные уже с новой. => с пом iis м менять наше приложение не останавливая работоспос прил-я. В core так делать нельзя

Для микросервисов разработан спец док **CLA (agreement)** – соглашение, от берет на себя та организация, кот предоставляет сервис. Там гарантии, как он работает и т.д.

**ЛК 09.10.20 - NANCY**

Этот фреймворк раб в ASP.NET CORE – Nancy. OWIN – арх core приложения – связ компоненты: хост, веб-сервер и наше прил. Эти компоненты явл независимыми и связ с пом ин-са OWIN.

Katana: берем готовую спецификаицю и дорабатываем только application – для этих апл. есть спец. фреймворки (Nancy, напр.)

Хост, веб-сервер, appl – части OWIN  
Nancy – фреймворк позв разраб прил

Nancy фреймворк скачивается с пом Nuget  
разраб обычое консольное прил – и буду инсталировать package (SelfHosting?...). Внутри main запускаем хост. Есть готовый хост и сервер. И теперь я могу с пом этого фрейморвка обавлять свою фун-сть – та часть, кот наз application

Все что будем добавлять – наз. модули – это класс, явл наследником баз. класса (отд файл .cs) NancyModule. Внутри класса только конструктор – запис. обработчик запроса – выглядит как подобие обработчика маршрутизации.

Nancy в авто- режиме сериализует наш объект в xml/json в завис от того какой Accept приходит от К.

QUERY-парам: мб  
**BIND-модель**: спец механизм, позв связывать параметры, кот есть в запросе, с параметрами вашего модуля.

Генерит класс – указ что те параметры кот мб в запросе, будут преобразованы станд. образом в одноименные переменным структуры/класса, кот мы указ в кач-ве типа

Если есть тип parm – у него есть 2 св-ва: x,y. То такой бинд будет в запросе отыскивать параметры x,y и пытаться их преобразовать

Мб еще св-ва, не предназначенные для binding-а – м их внести в black list, кот мы потом при связывании не используем вот такого типа имена для свзывания. Это чтобы если параметры каким-то образом появится параметр с именем кот. есть в моем классе, а я хочу чтобы он не сработал. Могу указать 2 строки с именами параметров, кот не подлежат связыванию. Теперь если в запросе будет параметр с именем R, Не будет свзяывать эти паарметры со значениями кот есть в parm. Это защита чтобы не было испорчено значение, кот есть в классе.

Это get-модель – часть кот сильно развита в NANCU и позв много чего делать: связывать параметры со св-вами объектов, также позвол валидировать: есть с-ма с пом кот мы можем сразу валидировать параметры кот пришли в запросе. Отд пакет – IsValidate

Если делается с браузера запрос, было бы удобно прилагать инструкцию по использованию этого сервиса. АНалогичный подход здесь – позв создавать какую-то разметку если запрос идет на AccpetHtml. Nancy поддреж 3 разметки: собственную sshtml, html, cshtml – позв в простом режиме вывести к-то текст: написать инструкцию по использованию этого сервиса.

Надо написать view – html или cshtml, у кот. имя как у класса (parms): В html доступан разметка sshtml – super simple html. Accept на html если с браузера – нанси вызываем view с именем parms – в этом view есть модель, в кот доступны эти св-ва : x, y.

Есть класс c именем parms, html c именем parms.

**Обработка POST-параметров:**еще один модуль  
через форму могу передавать параметры – они будут нормально восприниматься если аккуратно расставлен accpet и content-type.

Если accept = content-type = json, То json передаю в кач-ве параметры и ответ получаю тоже json. В зависим от conetnt-type по-разному интерпретируется body. Если даные с формы нормально их обрабаытывает в завис от content-type. Мы пишем один и тот же код и он по-разному работает в завис от тогок акой content-type и accpet

Сонтент-тайп: форма, Json…  
Один код, а формат д-х которые приходят мб различным – и в любом случае nancy их нормально обрабатывате

BodyOnly – учит только параметры из тела  
Если заданы и query И body параметры, мы м. управлять какие из них учитывать: bodyonly – query Игнрируются. М сделать get, post put delete в одном модуле – обычно так и делают. Обрабывают все 4 типа запроса кот предумсотрены в rest

М исп шаблоны auery-параметров: констрейны (как в ПИС). Тоже есть атрибутные способы маршрутизации.   
указ **dynamic** – динамич метод (Пацей): как js-объект – м динамически создавать свойства. Например в ViewBag.type = ; (авто- появляются свойства). Но свойства не имеют опред. типа, надо ставить впееди приведение к типу. Потому что dynamic Не знает о типе объекта, кот ему присвоен.

В NANCY сть 2 типа разметки: html, sshtlm (super simple), но м также применять razer view engine cshtml. М тоже использ ViewBag при передаче параметров во view. Можно тоже использовать partial – частичные представления. Есть такие же section . М применять шаблон и razor разметку.