Федеральное государственное бюджетное образовательное учреждение

высшего профессионального образования

«Национальный исследовательский Томский политехнический университет»

Сервис-ориентированная архитектура, REST веб-сервисы

Методические указания по выполнению лабораторной работы

по курсу «МЕТОДЫ И СРЕДСТВА ПРОЕКТИРОВАНИЯ ИНФОРМАЦИОННЫХ СИСТЕМ И ТЕХНОЛОГИЙ»

для студентов, обучающихся по направлению

09.03.02 Информационные системы и технологии

Томск – 2013

Содержание

[Цель работы 3](#_Toc429736013)

[Описание сервис-ориентированной архитектуры 3](#_Toc429736014)

[Пример REST веб-сервиса 5](#_Toc429736015)

[Задание 8](#_Toc429736016)

[Дополнительная литература 9](#_Toc429736017)

# Цель работы

Целью работы является изучение сервис-ориентированной архитектуры программного обеспечения.

# Описание сервис-ориентированной архитектуры

Сервис-ориентированная архитектура – способ организации программного приложения, при котором его функциональность представляется в виде набора сервисов. Сервис-ориентированная архитектура является одним из шагов для создания промежуточного программного обеспечения (middleware). Сервисно-ориентированная архитектура позволяет создавать распределнное программное обеспечение, состоящее из набора независмых сервисов. Благодаря сервисно-ориентированной архитектуре могут создаваться приложения, являющиеся композицей нескольких сервисов.

Описание веб-сервиса включает его название, расположение и требования к осуществлению процесса обмена данными.

Веб-сервис – программный компонент, идентифицируемый веб-адресом и выполняющие операции, продекларированные в поддерживаемом им интерфейсе. Другие программные компоненты взаимодействуют с веб-сервисами посредством обмена сообщениями.

Веб-сервисы в настоящее время становятся стандартом де-факто для интеграции распределенных систем программных приложений. Создание программных приложений, которые способны использовать веб-сервисы, недостаточно для полной поддержки бизнес-процессов. Организация взаимодействующих друг с другом веб-сервисов в сети позволяет реализовывать сложные рабочие процессы.

REST — метод взаимодействия компонентов приложения с использованием протокола HTTP для вызова процедуры. Пот этом необходимые данные передаются в качестве параметров запроса. Этот способ является альтернативой более сложным методам, таким как SOAP, CORBA и RPC.

Термин REST (representation state transfer, передача состояния представления) введен и определен в 2000 году Роем Филдингом в его кандидатской диссертации «Архитектурные стили и проектирование архитектур сетевого программного обеспечение».

Веб-сервисы REST являются веб-сервисами, реализуемые с использованием HTTP и принципов REST. Как правило, Web-сервис RESTful определяет URI основного ресурса, поддерживаемые MIME-типы представления/ответа и операции.



*Рис. 4.5. REST веб-сервис*

Листинг . Пример запроса к REST веб-сервису

GET /StockPrice/IBM HTTP/1.1

Host: example.org

Accept: text/xml

Accept-Charset: utf-8

Листинг . Пример ответа на запрос к REST веб-сервису

HTTP/1.1 200 OK

Content-Type: text/xml;

charset=utf-8;

Content-Length:nnn;

<?xmlversion="1.0"?>  
 <s:Quote xmlns:s="http://example.org/stock-service">

<s:TickerSymbol>IBM</s:TickerSymbol>

<s:StockPrice>45.25</s:StockPrice>

</s:Quote>

Ресурс может являться практически любым понятным изначимым адресуемым объектом.

Представление ресурса — это обычно документ, отражающий текущее или требуемое состояние ресурса. Ресурсы обычно представлются документами в форматах XML или JSON.

JSON (Javascript j,ject notation) — эффективный текстовый формат кодирования данных, обеспечивающий быстрый обмен небольшими объемами данных между клиентскими браузерами и веб-службами с поддержкой AJAX.

JSON-текст представляет собой (в закодированном виде) одну из двух структур:

* Набор пар ключ: значение. В различных языках это реализовано как объект, запись, структура, словарь, хэш-таблица, список с ключом или ассоциативный массив. Ключом может быть только строка, значением — любая форма.
* Упорядоченный набор значений. Во многих языках это реализовано как массив, вектор, список или последовательность.

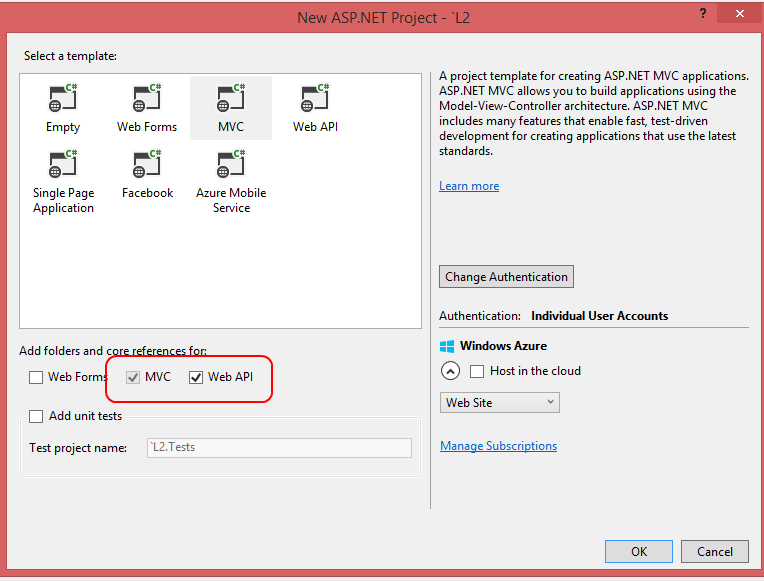
REST и горизонтальный подход. Стратегия, опирающаяся на горизонтальный подход к протоколам, наиболее радикальна. Слово «горизонтальный» означает в данном случае сохранение существующего уровня, без выстраивания уровней поверх него. Предполагается отказаться от разработки новых протоколов, а использовать несколько хорошо проверенных, считая, что для работы с объектами вполне достаточно уметь выполнять четыре типа действий: создание (Creation), восстановление (Retrieval), изменение (Update) и уничтожение (Destruction). Из этих действий получается так называемый «шаблон проектирования» CRUD. Протокол Hypertext Transfer Protocol определяет методы GET/PUT/POST/DELETE, которые и реализуют шаблон CRUD. Абревиатура (шаблон) CRUD обозначает перечень основных операций с объектом: create (создать), read (считать, загрузить), update (обновить, изменить, отредактировать) и delete (удалить).

Таблица 4.1

*HTTP-методы в REST веб-сервисах*

|  |  |
| --- | --- |
| HTTP-метод | CRUD операция |
| GET | Read |
| POST | Create (иногда используется для операций update, delete) |
| PUT | Create (иногда используется для операции update) |
| DELETE | Delete |

# Пример REST веб-сервиса

Демонстрационный веб-сервис создан с использованием среды разработки Microsoft Visual Web Developer. Все упомянутые далее классы размещены в проекте ASP.NET MVC Application.  
  
Рисунок 3

Для демонстрационного веб-сервиса создан класс-сущность Individual (физическое лицо):

public class Individual

{

public int Id;

public string Name;

public int Age;

}

Также создан репозиторий-заглушка, реализующих методы CRUD:

public class IndividialsRepository

{

List<Individual> data = new List<Individual>();

public IndividialsRepository()

{

data.Add(new Individual() { Id = 1, Name = "Алексей", Age = 10 });

data.Add(new Individual() { Id = 2, Name = "Светлана", Age = 6 });

data.Add(new Individual() { Id = 3, Name = "Петр", Age = 5 });

}

public IEnumerable<Individual> Get()

{

return data;

}

public void Add(Individual item)

{

data.Add(item);

}

public void Update(Individual item)

{

var tmp = data.Where(x => x.Id == item.Id).FirstOrDefault();

tmp = item;

}

public bool Delete(int Id)

{

var item = data.Where(x => x.Id == Id).FirstOrDefault();

if (item != null)

{

data.Remove(item);

return true;

}

return false;

}

}

Непосредственно методы REST-сервиса реализованы в классе IndividualAPIController:

public class IndividualAPIController : ApiController

{

IndividialsRepository repository = new IndividialsRepository();

// GET api/individualapi

public IEnumerable<Individual> Get()

{

return repository.Get();

}

// GET api/individualapi/5

public Individual Get(int id)

{

return repository.Get().Where(x=>x.Id==id).First();

}

// POST api/individualapi

public void Post([FromBody]Individual ind)

{

repository.Add(ind);

}

// PUT api/individualapi/5

public void Put(int id, [FromBody]Individual ind)

{

repository.Update(ind);

}

// DELETE api/individualapi/5

public void Delete(int id)

{

}

}

В классе-контроллере используется соглашение об именовании методо: названия методов класса должны совпадать с методами протокола HTTP.

Возвращаемое XML- значение по HTTP-запросу GET /api/individualapi формируется в результате сериализации массива и выглядит следующим образом:

<ArrayOfIndividual>

<Individual>  
 <Age>10</Age>

<Id>1</Id>

<Name>Алексей</Name>

</Individual>

<Individual>

<Age>6</Age>

<Id>2</Id>

<Name>Светлана</Name>

</Individual>

<Individual>

<Age>5</Age>

<Id>3</Id>

<Name>Петр</Name>

</Individual>

</ArrayOfIndividual>

Также возможно получить JSON-результат:

[{"Id":1,"Name":"Алексей","Age":10},{"Id":2,"Name":"Светлана","Age":6},{"Id":3,"Name":"Петр","Age":5}]

Для получения JSON-результата требуется указать в параметрах запроса тип принимаемых данных:

GET /api/individualapi HTTP/1.1

User-Agent: Fiddler

Host: localhost:9883

Content-Type: application/json; charset=utf-8

Пример JavaScript-клиента:

<div>

<button id="btnLoad">

Load

</button>

</div>

<div id="results"></div>

<script type="text/javascript">

$(function () {

$("#btnLoad").click(function () {

$.ajax({

type: 'GET',

url: 'http://localhost:10916/api/IndividualAPI',

// data: { get\_param: 'value' },

dataType: 'json',

success: function (data) {

$.each(data, function (index, element) {

$('#results').append(element.Name+"<br/>");

});

}

});

});

});

</script>

# Задание

Измените программное приложение, созданное при выполнении лабораторной работы №1. Реализуйте функциональность в виде набора REST веб-сервисов. В качестве типа программного приложения клиента используйте веб-клиент, созданный с использованием библиотеки *JavaScript JQuery*. Формат передаваемых и получаемых данных − XML или JSON (на выбор студента). В клиентском приложении должна быть только одна HTML-страница, а функциональность приложения реализуется исключительно средствами JavaScript и JQuery.

Отчет по лабораторной работе должен включать следующие разделы:

1. Задание.
2. Описание решения.
   1. Описание API-контроллера (класса, производного от типа *APIController*).
   2. Отрывки исходного кода с пояснениями.
   3. Примеры отправляемых и получаемых данных для каждого метода веб-сервиса.
3. Заключение.
4. Список использованных источников (не менее 4 источников).

# Дополнительное задание

Измените программное приложение, созданное при выполнении лабораторной работы №3. Используя Nuget-пакет Unity.WebAPI, настройте REST-контроллеры для работы с контейнером зависимостей (https://docs.microsoft.com/en-us/aspnet/web-api/overview/advanced/dependency-injection#dependency-resolution-with-the-unity-container).

Дополнительная литература

1. Руководство Microsoft по проектированию архитектуры приложений. 2е издание. URL: http://apparchguide.ms/Book
2. <http://www.asp.net/web-api>
3. <http://api.jquery.com/jquery.ajax/>
4. <https://habrahabr.ru/post/42426/>
5. <https://www.syncfusion.com/resources/techportal/ebooks/jquery>
6. <https://www.syncfusion.com/resources/techportal/ebooks/webapi>