=== Programlama ve Tasarım ===

--- REST Servers in Delphi XE Using DataSnap ---

Delphi XE'de DataSnap Kullanarak REST Server Oluşturma

--- Yürütülebilir Özet ---

Representational State Transfer yani REST (Temsili Durum Transferi) endüstride önemli bir etkisi olan Web Servisleri için yeni bir teknolojidir. Google, Yahoo, Amazon ve şimdi Microsoft gibi büyük satıcıların pek çok halka açık web servisi çoklu kaynaklardan bilgileri paylaşmak ve birleştirmek için REST teknolojisine güvenmektedir.

REST mimarisinin uygulanması HTTP ve XML gibi basit teknolojileri kullanmaya dayanır ve Delphi bunun için tarihi olarak bir desteğe sahiptir. Delphi 2010, DataSnap altyapısını REST desteğini eklemiş, ve Delphi XE bu modeli WebBroker entegrasyonu ve web servis tarafından gösterilen metodlar için JavaScript oluşturma özelliği ile bir adım ileri taşımıştır.

Bu belge Delphi XE üzerinde REST sunucuları geliştirmeyi, kutuda var olan özellikleri nasıl kullancağımızı, var olan özellikleri Delphi destek kodu ile nasıl genişleteceğimizi ve JQuery kütüphanesini almanın yararlarını anlatacaktır. DataSnap tarafından teklif edilen modeller içerisinde bizim özellikle ilgilileneceğimiz alan, Delphi REST sunucusunu geliştirmek, ve JavaScript ile JQuery esas alınarak yazılan tarayıcı tabanlı uygulamalarla birbinine bağlamak, çiftleştirmektir. Unutmamak gerekir ki, aynı REST sunucuları mobil cihaz tabanlı uygulamalar gibi pek çok çeşitli palatformda da kullanılabilir.

--- Giriş - REST'in Arkasındaki Fikirler ---

Son 10 yılda Web'deki patlamaya tanık olduk ki artık şu anda Web 2.0 olarak adlandırılıyor. İzlemeye başladığımız şey, farklı web siteleri arasındaki, web siteleri ve istemci uygulamalar arasındaki ve web siteleri ile işle ilgili veritabanları arasındaki otomatik etkileşim, ki bu global bir etkileşim olup bunu tamamen anlamak çok zordur.

Web'de veri bizim göz atmamızdan çok daha hızlı hareket eder, bu yüzden satış verileri, finansal bilgi, çevrimiçi topluluklar, satış kampanyaları ve diğer pek çok uygulama gibi bilgiyi izleme, takip etme, monitörleme ile ilgili olarak pek çok programın isteği de bu yöndedir. Aynı zamanda sunucu taraftaki işletim, istemci uygulamalar ve web tarayıcı üzerinden doğal olarak çalışan uygulamalar tarafından kaldırılabilmektedir (kaldıraç kuvveti anlamında).

Web servis fikri biraz daha soyuttur. Teknolojilere baktığımız zaman geliştiricileri etkileyen 2 adet teknoloji olduğunuz görürüz. İlki Simple Object Access Protocol yani SOAP (Basit Nesne Erişim Protokolü) olup http://www.w3.org/TR/soap/ adresinde tanıtılmaktadır. Delphi'nin birkaç yıldır SOAP desteği bulunmakta, Delphi XE'de de güncel iç hatları ile birlikte bu destek bulunmaktadır. Diğer web servis çözümü ise REST yaklaşımıdır.

Bu belgede REST üzerine yoğunlaşacağız, çünkü daha esnektir ve istemci tarayıcı uygulamaları için en iyi şekilde uyugn bir arka uçtur (ayrıca pek çok diğer senaryo için de). Buradaki resmi isim ve teoriye giriş tamamen yenidir. İlerde bahsedilenler, herhangi bir resmi REST standardı olmadığı ile alakalıdır.

Fikir, herhangi bir web kaynağına erişildiği zaman (bir tarayıcı veya istemci bir uygulama vasıtasıyla), sunucu kaynakla ilgili olarak bir suret gönderecektir. (Bir HTML sayfası, bir resim, biraz ham data) istemci verilmiş bir durumda alır bu sureti. İstemci daha fazla bilgiye veya sayfaya (belki bir link (bağlantı) kullanarak) eriştiğinde durumu değişir, bir önceki durumdan transfer edilerek. Roy Fielding'in sözleri ile;

"REST iyi tasarlanmış bir web uygulamasının davranışlarının nasıl bir resme çağrıştırılacağını planlar; bir web sayfaları ağı (sanal durum makinası), kullanıcı uygulamada linkleri seçerek işletim yaptığında (durum geçişleri), kullanıcıya kullanmaları için transfer edilir ve bir sonraki sayfa ile sonuçlanır (uygulamanın bir sonraki durumunu gösterme) "

--- REST Mimarisinin Anahtar Noktaları ---

Bu yüzden REST bir mimari ise (hatta daha iyisi, mimari bir stil ise) bunun bir standardı yoktur, bununla beraber HTTP, URL ve güncel veri için pek çok artı formatlı standardı kullanmaktadır.

SOAP'ın karşıtı olarak, REST mimarisi HTTP ve bunun data formatını (genelde XML ve JSON) kullanır.

- REST, sunucudaki bir kaynağı kimliklendirmek için URL kullanır. (SOAP detaylı olarak mektubunda anlatıldığı üzere pek çok sitek için tek bir URL kullanır.) Unutmamak gerekir ki burada fikir URL'yi kaynak üzerinde işlem yapmak için değil, kaynağı kimliklendirmek için kullanılıyor.
- REST, hangi işlemin gerçekleştirileceği ile ilgili olarak HTTP metodlarını kullanır (retrieve (almak) veya HTTP get, create (oluşturmak) veya HTTP PUT, update (güncelleme) veya HTTP POST, delete (silme) veya HTTP DELETE)
- REST, sunucudan daha detaylı bilgi sağlayabilmek için HTTP parametrelerini kullanır (hem sorgu parametreleri hem de POST parametreleri)
- REST, yetkilendirme, kriptolama ve güvenlik (HTTPS kullanarak) için HTTP'ye güvenir.
- REST, veriyi çeşitli formatlar kullanarak (XML, JSON, resim ve daha pek çok format) sade döküman olarak geri döndürür

Bu tarz bir senaryoda bir kaç mimari element daha düşünülmelidir. REST sistemden şunları talep eder.

- Doğal bir istemci/sunucu ilişkisi (veritabanı ile (RDBMS) direkt olarak herhangi bir işlem yapılmayacak)
- Doğal olarak durumsuz
- Cache-dostu (Sıralı olarak aynı şeyi iki kez çağırdığımda sunucu tarafında data değişmediği sürece aynı URL aynı datayı geri döndürmelidir, sunucu ve istemci arasına proxy ve cache sunucularının eklenmesine müsade etmelidir, buradan çıkacak sonuç bütün GET işlemlerinin herhangi bir etkide bulunmayacağıdır.)

Elbette burada kısaca belirttiğimizin yanında REST teorisi ile ilgili pek çok nokta daha olacaktır, ama bence bu kadarı teoriye başlamanız için yeterlidir. Delphi kodu ile birlikte gelecek pratik örnekler ana fikirleri açıklığa kavuşturacaktır.s

--- REST Teknolojileri ve Delphi ---

Daha önce herhangi bir REST standardı olmadığını ve REST geliştirmek için herhangi bir özel araca ihtiyaç olmadığını anlatmıştık. Ama var olan bazı standartlar REST üzerinde de tekrar edilebilir ve bundan kısaca bahsedeceğiz (Uzunca bahsetmek

istersek her bir başlık bir kitap olacaktır.) Özellikle odaklanacağımız nokta, Delphi'nin bu teknolojilere sağladığı destektir.

HTTP, Web'in kalbindeki standarttır ve herhangi bir girişe ihtiyaç bulunmamaktadır. Ayrıca HTTP, sadece web tarayıcıları için değil, diğer uygulamalar için de kullanılabilir.

Delphi uygulamalarında HTTP kullanan bir istemci uygulama yazmanın en basit yolu, INDY HTTP istemci bileşeni olan IdHttp kullanmaktır. Eğer bu bileşenin bir URL adresini parametre göstererek Get metodunu çağırırsak, herhangi bir web sayfasının ve REST sunucularının içeriğini alabiliriz. Bazen diğer parametrelere değer atamak gerekebilir, yetkilendirme bilgisini girmek ve SSL desteği için başka bir bileşen eklemek gerekebilir. (İlerde bazı örneklerde göreceğiz) Bileşen Get dışında çeşitli HTTP metodlarını da desteklemektedir.

Unutmamak gerekir ki güvenli tarafta, IdHttp isteklerini bir thread içerisinde oluşturmak gerekir. Çünkü Indy seti, thread bloklama kullanır, yani programın kullanıcı arayüzü istek geri dönene kadar sabitlenir (yavaş bir web sunucusunda veya büyük veri transferinde bu işlem uzun zaman alabilir.) Bu belgedeki örneklerde basitlik amaçlı olarak thread kullanmayacağımi ama size tavsiye edeceğim yaklaşım thread kullanarak yapılacak yaklaşımdır.

Sunucu tarafında Delphi'de bir web sunucusu veya web sunucusu uzantıları oluşturmak için pek çok mimari kullanılabilir. IdHttpServer bileşeni kullanarak tek başına çalışan bir web sunucusu oluşturmak veya CGI uygulamaları, ISAPI veya Apache modülleri gibi web sunucusu uzantıları oluşturmak mümkündür. Delphi XE ile başladığımızda WebBroker mimarisi iki modeli de destekler (Daha önce uzantılarla ilgili destek sınırlıydı). Gerçekten, Delphi XE sihirbazı ile bir DataSnap REST sunucusu oluşturmak mümkündür, böylece dolaylı olarak IdHttpServer bileşenini temel alan tek başına çalışan bir web sunucusu elde etmek mümkündür.

--- Delphi'de JSON ---

Web servisler XML veya JSON'dan herhangi birini kullanırken Delphi REST sunucularının bu konuda varsayılan değeri ikincisidir. Bu yüzden buradaki Delphi ve REST ile ilgili girişi, JSON (JavaScript Object Notation = JavaScript Nesne Gösterimi) ile ilgili kısa bir giriş yapmak için sonlandırıyorum

--- JSON Gösterimi ---

JSON, JavaScript nesnelerini tanıtmak için text tabanlı bir gösterimdir, böylece sabit olup bir uygulamadan öbürüne aktarılabilirler (Veya bir bilgisayardan diğerine). Bir kaç yıl önce karmaşık veri yapılarını kullanmak için genelde XML gösterimi kullanılıyordu, ama son yıllarda JSON'un popülerliği arttı çünkü daha sıkı, programlama dilleri kavramlarına daha uygun, JavaScript tarayıcı tabanlı uygulamalarda ve artan kütüphanelerle araçlarla çok daha fazla programlama dili ile yazılmış uygulamalarla çözümlenmesi daha kolaydır.

JSON'la ilgili olarak daha fazla bilgi için IEFT'in (Internet Engineering Task Force - İnterner Mühendisliği Görev Gücü) RFC 4627 belgesi ve JSON'un resmi web sitesindeki dökümanları takip edebilirsiniz. Adresler aşağıdadır.

http://www.ietf.org/rfc/rfc4627.txt
http://json.org

Ayrıca sadece kısa bir girişi okuyabilirsiniz, JSON göreceli olarak anlaşılması kolay, 4 ilkel tip ve 2 yapıdan oluşmaktadır. İlkel türler sayılar, stringler, Boolean değerler (true veya false) ve null değeridir.

İki JSON veri yapısı da;

JSON Objects (JSON Nesneleri) : İsim ve değer çiftlerinden oluşan koleksiyonlardır, küme parantezi ile açılıp kapatılır ve virgüllerle ayrılır. (Bir çiftin iki elemanı iki nokta ile ayrılır) Koleksiyon, kayıt veya nesneyi temsil eder.

JSON Arrays (JSON Dizileri) : Kare parantez içerisinde virgül ile ayrılan değerler listesidir. Liste, burada koleksiyon veya diziyi temsil eder.

Burada iki çiftten oluşan bir nesne (çift isimlerini içeren bütün stringler çift tırnak içine alınmıştır) ve bir sayı ile bir string'den oluşan 2 değerli bir listenin gösterimi verilmiştir.

```
{ "Name":"Marco",
"Value":100
}

[22, "foo"]
```

Elbette bu yapıları birleştirebilirsiniz, nesne ve dizileri bir çiftin değerleri veya bir dizinin elemanları olarak gösterebilirsiniz.

```
{ "Me": {
        "FirstName":"Marco",
        "LastName":"Cantù",
        "Wife": "Lella",
        "Kids": [
                { "Name":"Benedetta",
                        "Age":11
                },
                { "Name":"Jacopo",
                        "Age":7
                }
        ]
}}
```

--- Delphi 2010'da JSON ---

Geçmişte Delphi için bir kaç JSON kütüphanesi olsa da, doğal destek olan ilk versiyonu Delphi 2010'dur. Doğal JSON desteği dbExpress framework'u ile ilişkili olmayan uygulamalarda kulllanılan, DBXJSON unit dosyası içerisinde tanımlanmış bir dizi sınıf ile sağlanır.

DBXJSON unit dosyası, TJSONValue nesnesinden türeyen çeşitli JSON veri türleri ile çalışmayı sağlayacak sınıfları tanımlar. (çeşitli türde özel değerler, diziler, çiftler ve nesneler)

TJSONNull, TJSONNumber, TJSONTrue, TJSONFalse, TJSONString'i içeren ilkel türler

TJSONObject (dahili olarak TJSONPair'i içerir) ve TJSONArray gibi veri yapılarını

Burada farklı ilkel türlerin çıktısını gösteren, JsonTests projesindeki basit bir kod parçası bulunmakta. Unutmamak gerekir ki LogAndFree özel destek metodunu ekleyerek geçici olarak oluşturduğumuz her nesne elle serbest bırakılmalıdır. (free)

```
procedure TFormJson.LogAndFree (jValue: TJSONValue);
begin
	try
		Log (jValue.ClassName + ' > ' + jValue.ToString);
	finally
		jvalue.Free;
	end;
end;

procedure TFormJson.btnValuesClick(Sender: TObject);
begin
	LogAndFree (TJSONNumber.Create(22));
	LogAndFree (TJSONString.Create('sample text'));
	LogAndFree (TJSONTrue.Create);
	LogAndFree (TJSONFalse.Create);
	LogAndFree (TJSONNull.Create);
end;
```

Beklenen çıktı da şu şekilde olacaktır.

```
TJSONNumber > 22
TJSONString > "sample text"
TJSONTrue > true
TJSONFalse > false
TJSONNull > null
```

Dizi ve diğer nesneleri oluşturmak için DBXJSON unit dosyası içerisindeki sınıfları kullanmak da son derece kolaydır. Dizi, içerisinde değer ekleyebildiğimiz bir yapıdır (Dizileri ve nesneleri içerir)

```
procedure TFormJson.btnSimpleArrayClick(Sender: TObject);
var
	jList: TJSONArray;
begin
	jList := TJSONArray.Create;
	jList.Add(22);
	jList.Add('foo');
	jList.Add(TJSonArray.Create (TJSONTrue.Create));
	(jList.Get (2) as TJSonArray).Add (100);
	Log (jList.ToString);
	jList.Free;
end;
```

JSON çıktısı içiçe iki tane içiçe diziyi şu şekilde gösterir.

```
[22,"foo",[true,100]]
```

Unutmamak gerekir ki JSON taşıyıcıları (diziler ve nesneler) kendi dahili elemanlarına sahiptir, bu yüzden belleği temizlemek için taşıyıcıyı serbest bırakarak, içerisinde bulunan bütün JSON değerlerini de serbest bırakmış olursunuz.

Eğer bir nesneniz varsa, buna ekleyebileceğiniz tek eleman bir çifttir, ama bu çiftin değerleri içiçe bir nesne gibi herhangi bir JSON değeri olabilir.

```
procedure TFormJson.btnSimpleObjectClick(Sender: TObject);
var
	jsonObj, subObject: TJSONObject;
begin
	jsonObj := TJSONObject.Create;
	jsonObj.AddPair(TJSONPair.Create ('Name', 'Marco'));
	jsonObj.AddPair(TJSONPair.Create ('Value',
	TJSONNumber.Create(100)));

	subObject := TJSONObject.Create(
	TJSONPair.Create ('Subvalue', 'one'));
	jsonObj.AddPair(
	TJSONPair.Create ('Object', subObject));

	Log (jsonObj.ToString);
	jsonObj.Free;
end;
```

Bu nesnenin JSON gösterimi şu şekildedir (Okunabilirliği artsın diye elle biraz formatını düzelttim)

```
{ "Name":"Marco",
	"Value":100,
	"Object": {
	"Subvalue":"one"
	}
}
```

--- Delphi'de JSON'u Ayrıştırmak (Parse) ---

DBXJSON sınıflarını kullanarak JSON veri yapıları oluşturmak ve ilişkili JSON gösterimini oluşturmak ilginçtir, ama daha ilginç olan bunun tersini yapabilmek, yani JSON gösterimi olan bir string'i ayrıştırarak ilişkili Delphi nesnelerini oluşturmaktır.

Öncelikle bir JSON string'imiz olsun, biz bunu TJSONObject sınıfının ParseJSONValue sınıf metoduna gönderelim, ve bu metoddan bize TJSONValue türünde bir nesne dönsün. Olaylarda nerede JSON nesnesi döndürdüğümüzü bilmemiz, bunu uygun bir türe aktarmamız (cast) gerekmektedir. ParseJSONValue sınıf metodu String türünden bir nesneyi parametre olarak kabul etmemektedir, ama ASCII veya UTF8 türünden (UTF8 Delphi 2010'da yok, Delphi XE ile gelen bir özellik) bir byte dizisini kabul etmektedir. İki durumda da string'i alarak uygun Encoding (dil kodlaması) sınıfını kullanarak byte dizisine çevirmemiz gerekmektedir. İlgili kod aşağıdadır.

```
TEncoding.ASCII.GetBytes(strParam) // a string converted to ASCII
TEncoding.UTF8.GetBytes(sUtf8) // any UTF8 (or Unicode) string
```

UTF8 string durumunda ParseJSONValue metoduna ek biri parametre göndermek veya ParseJSONValueUTF8 metodunu çağırmamız gerekmektedir. Sonuçta JsonTests örneğindeki ana form üzerinde bulunan ilgili butonun Click olay tetikleyicisi olan btnParseObjClick metodunun başlangıç parçası aşağıdaki gibidir.

```
var
        strParam: string;
        jsonObj: TJSONObject;
begin
        strParam := '{"value":3}';
        jsonObj := TJSONObject.ParseJSONValue(
        TEncoding.ASCII.GetBytes(strParam), 0) as TJSONObject;
```

Geri kalan kod bütün nesneyi çıktı olarak verir (eğer herhangi bir şey yanlış gitmezse, orjinal JSON gösterimine geri dönülür), son (ve tek) isim/değer çifti ve TJSONObject nesnesi serbest bırakılır (tekrar belirtmek gerekir ki bu kodu unutmak bellek sızıntılarına çok kolay bir şekilde neden olur)

```
Log (jsonObj.ToString);
Log (jsonObj.Get (jsonObj.Size - 1).ToString);
jsonObj.Free;
```

--- Delphi XE'de DataSnap REST Sihirbazı ---

Delphi'de REST ve JSON ile ilgili bazı anahtar kavramlarla giriş yaptığıma göre, şimdi Delphi XE'de DataSnap'in sunduğu REST desteği ile ilgili olarak biraz pratik yapabiliriz. (Unutmamak gerekir ki DataSnap desteği sadece ürünün Enterprise ve Architect versiyonlarında bulunmaktadır.)

Balşama noktamız, yeni DataSnap REST Sihirbazı ile bir REST sunucusu oluşturmaktır. Alternatif olarak, HTTP desteği olan daha genel bir DataSnap sunucu uygulaması oluşturabiliriz ki, bu da bir REST sunucusu temin etmenin diğer yoludur. Bununla beraber il yaklaşımla, JavaScript desteği olan ve HTMP template'leri kullanabileceğimiz bir uygulama temin etmiş oluruz. Göreceğiniz gibi bu gelişmiş destek, DataSnap REST sunucumuzu tam gelişmiş bir web uygulamasına dönüştürmek için oldukça iyi bir başlangıç noktasıdır.

Sihirbazda sade bir DataSnap Sunucusu, DataSnap WebBroker uygulaması ve DataSnap REST Sunucusu olmak üzere 3 ayrı sunucu seçeneği olduğunu görüyorsunuz. Biz sonuncusunu seçeceğiz.

--- Sihirbazı Çalıştırmak ---

Delphi XE'de ilk basit REST sunucumuzu oluşturmak için, DataSnap REST Application seçeneğini kullanabiliriz. Sihirbazın ilk sayfasında az çok DataSnap WebBroker Uygulamasına benzer bir şekilde tamamı WebBroker mimarisine dayanan çoklu alternatifler bulunmaktadır.

IdHTTPServer bileşeni ile WebBroker entegrasyonunu kullanarak 2 adet tek-başına (stand-alone) seçeneğimiz vardır, böylece uygulama etkin bir şekilde bir web sunucusu olabilir. 3. seçenek ISAPI'dir. Potansiyel olarak Web App Debugger ve Apache Module gibi diğer Web Broker seçeneklerini de kullanabilirsiniz.

İlk proje için test ve hata ayıklama için en uygun seçenek olan "Stand-Alone VCL Application" "Tek Başına VCL Uygulaması" seçeneğini seçiyorum. Tek başına çalışan bir web servis oluşturuyorum, sonra bir port numarası seçiyorum ve uygun olup olmadığını

test ediyorum. Sihirbazın direkt olarak tavsiye ettiği varsayılan port numarası 8080'dir ama makinanızda uygun olan herhangi bir port numarası da olabilir bu. Bu sadece sunucunun bir özelliğidir ve sonradan değiştirilebilir.

Üçüncü adım sunucunun kimlik doğrulama ve yetkilendirme gibi bazı özelliklerini belirlemektir.

Eğer sihirbazda şimdi kimlik doğrulamayı sihirbazdan seçilebilir hale getiremiyorsanız üzülmeyin, bunu sonradan projeye kolay bir şekilde ekleyebilirsiniz. Diğer taraftan, sihirbaz ilk kez kullanırken sunucu sınıfı bir kaç örnek metodla eklemek iyi bir fikirdir. Bu sunucu sınıfı metodlarını REST servis talepleri olarak dışarıya gösteren bir sınıftır ve sonra projeye bir sınıf daha (birden çok da olabilir) ekleyebilirsiniz.

Sonra, eğer örnek bir sunucu sınıfı seçtiyseniz, Sihirbaz size hangi sınıfı ata sınıf olarak kabul edeceğinizi, ona göre sınıfı oluşturacağını soracak. Mevcut seçenekler, TComponent, TDataModule taşıyıcısı (veri erişim bileşenleri ekleyebildiğiniz gibi görsel olmayan bileşenleri de kolayca ekleyebilirsiniz.), ve DataSnap Sunucu Modülüdür. (TDSServerModule) Son seçenek DataSnap REST uygulaması için mantıklıdır, diğer türden DataSnap sunucuları için ilginç bir seçenektir.

İlk proje için ana sınıf olarak TComponent ata sınıfını seçiyorum. Sihirbazın son sayfasında proje için ana dizini seçebilirsiniz. Bir dizinin yolunu verdiğinizde dizin adı (yol bilgisinin son alanı) proje adı olarak çiftlenir. Proje adının geçerli bir ad olması gerektiğinden, dizin adı içerisinde boşluk olmamalı ve sayı ile başlamamalıdır.

--- Sihirbaz Tarafından Oluşturulan Kod ---

Yukarıda belirtilen ayarlarla Delphi 3 unit dosyasına sahip bir proje oluşturur. Bunların kaynak kodunu görmek isterseniz, sihirbazda aynı adımları takip edin ve bu makale içerisinde yer alan RestXeOne'ın kaynak kodlarına bakın.

- Kısıtlı yönetim seçeneklerine sahip bir ana form. Bu form, siz proje yapısını ISAPI DLL'ine çevirdiğiniz zaman yok olur.
- DataSnap sunucu bileşenlerini, bir kaç diğer destek bileşenlerini, ve WebBroker uygulamasının çekirdeğine ev sahipliği yapan, TWebModule'den türetilmiş bir web data modülü
- REST sunucusuna metodları ekleyebileceğimiz, sunucu sınıfının bulunduğu bir unit dosyası

Ana form; çalışma zamanında port numarasını değiştirecek bir editbox bileşeni, bir Başlama (Start) ve Durdurma (Stop) buton bileşenleri, sunucuyu başlatan ve varsayılan web sayfasını tarayıcınızda açacak bir butondan oluşur. Her hangi bir değişiklik yapmadan önce formun tasarım zamanlı görünümü şu şekildedir. (Bunun için ilgili resme bakın)

Application Event bileşeni, OnIdle olay tetikleyicisini kullanarak, sunucunun durumuna (status) göre kullanıcı arayüzünü güncellemek için kullanılır. Özel bir alanda (FServer) kaydedilen TIdHTTPWebBrokerBridge nesnesi de form için önemli olan diğer bir elemandır. Bu, TIdCustomHTTPServer'dan türetilen bir sınıftır ve bu dahili web sunucusunun WebBroker mimarisi ile entegrasyonunu destekler. Diğer 2 unit dosyası, mimariye tamamen nötr olup, bir ISAPI modülü veya WebBroker uygulaması için de tamamen aynıdır.

--- Sihirbaz Tarafından Oluşturulan Web Data Modülü ---

Şimdi DataSnap REST Sihirbazı tarafından oluşturulan web data modülüne programa biraz kod ekleyip test etmeden önce biraz daha detaylı olarak bakalım. Web

modülü WebBroker mimarisinin çekirdek elemanıdır. Yol isimlerine (pathname) bağlı olarak birden çok hareket (action) tanımlayabilir, dispatcher (işlemci zamanlayıcısı) bileşenleri ile diğer yol isimlerini yakalayabilir, ve herhangi bir HTTP isteğine işlem öncesi ve işlem sonrası uygulamaları yapabilir.

Burada sunucunun ele aldığı yol isimleri ve uzantılar, bunları yöneten bileşen veya hareketler ve bilginin kaynağı bulunmaktadır.

Yol veya uzantı	Bileşen veya Hareket
Hedef	
/	DefaultAction (Varsayılan hareket)
WebModuleDefaultAction metodu	
/ServerFunctionInvoker	ServerFunctionInvokerAction
ServerFunctionInvoker sayfa üreticisi	
/ReverseString	ReverseStringAction
ReverseString sayfa üreticisi	
/datasnap*	DSHTTPWebDispatcher1
DataSnap işlemleri	
/datasnap/rest	DSHTTPWebDispatcher1
DataSnap REST işlemleri	
*.css, *.js, *.png	WebFileDispatcher1
Dosya sistemindeki (file system) dizinler	

Varsayılan hareket çok sınırlı bir role sahiptir. (İsteği yeniden yönlendirmek veya ReverseString (stringi tersine çevirme) sayfa üreticisinin çıktısını döndürmek) Bu üretici, aynı şekilde kendi yol ismine ve hareketine (action) bağlanmıştır, ilerde inceleyeceğimiz örnek bir HTML sayfası döndürecektir.

Bu web modülde çekirdek rol, DataSnap ve REST isteklerinin giriş noktası olduğundan DSHTTPWebDispatcher1'e aittir. Bu bileşen "datasnap" (vasrsayılan değeri) ile başlayan bir URL'den gelen bir isteği yakalayıp, web modül içerisinde bulunan DataSnap sunucu bileşenine (DSServer) gönderir. "datasnap" ile başlayan ve "rest" (bu da bir varsayılan değerdir) ile devam eden isteklerde işletim dahili REST makinasına yönlendirilir. Diğer bir deyimle "datasnap/rest" yoluyla bağlayan istekler, REST istekleri olarak tanımlanır.

Tasarım zamanında Sihirbaz tarafından otomatik eklenen bileşenlerle birlikte web modülün görünümü aşağıdaki gibidir (İlgili resme bakın)

Burada çeşitli bileşenlerin rolleri yer almaktadır.

DSServer1, bu mimarideki bütün çeşitli bileşenleri bir araya getiren çekirdek DataSnap sunucu bileşenidir. Rolünün merkei olmasına rağmen sadece bir kaç özelliği bulunmaktadır.

DSServerClass1, metodlarını REST servisleri olarak sunan, nesnelerinin yaşam döngüsünde onları oluşturan ve yöneten sınıfa referans olan bir bileşendir. Bunun rolü COM içerisindeki bir sınıf fabrikasına (class factory) paraleldir. Esas sınıf olan bu Sunucu Sınıfı, bir referans parametresi ile bileşenin OnGetClass olay tetikleyicisinde döndürür.

DSHTTPWebDispatcher1, daha önce de bahsedildiği gibi WebBroker mimarisi ile DataSnap sunucusu arasında köprü vazifesi görmektedir.

WebFileDispather1, sihirbaz tarafından oluşturulan bazı destek dosyalarını (JavaScript dosyaları, resimler, CSS'ler) ve bizim de ekleyebileceğimiz dosyaları döndürmek için kullanılan bir WebBroker bileşenidir (Delphi XE ile gelen bir yenilik). Bu bileşenin bir işlem-öncesi olay tetikleyicisi uygulamayı her tekrar-derleme ve tekrar-çalıştırmada serverfunctions.js dosyasını günceller. Bu JavaScript dosyası REST sunucusu tarafından gösterilen

metodların "proxy" arayüzünü (interface) içerir. Bu dosya altta bahsedilecek 2 bileşenin desteği ile güncellenir.

DSProxyGenerator1, çeşitli programlama dilleri için (Delphi, C++, JavaScript) proxy arayüzleri oluşturmak için kullanılan yeni bir Delphi XE bileşenidir. Bu. çağrılan URL adresini elle oluşturmaya gerek kalmadan, REST sunucusundaki bu metodları bu dillerde başlatmak için kullanılan yardımcı sınıflardır (helper class). Bu olayda bileşenin Writer (yazıcı) özelliğine "JavaScript REST" değeri atanmıştır. Bu bileşen bir alttaki bileşen ile birlikte çalışır.

DSServermetaDataProvider1, DSProxyGenerator1 bileşenine sunucunun çekirdek metadata bilgisini sunar. Bir destek bileşeni olup özelleşmiş herhangi bir ayarlaması yoktur.

ReverseString, template'ler içerisinde bulunan ReverseString.html dosyasına referansta bulunan bir sayfa oluşturucudur (page producer). Bu dosya, hernagi bir sunucu metodunuz için tamamen bir web sunucusunu, HTML ve JavaScript'te yazabileceğiniz örnek bir kullanıcı arayüzü ve uygulamayı içerir ve sağlar. Bu çoğu uygulamada başlangıç noktamız olacaktır. Bu dosyanın detayları ilerleyen bölümlerdedir.

ServerFunctionInvoker, metadatadan oluşturulan generic bir kullanıcı arayüzüne ev sahipliği yapan ikinci bir sayfa üreticisidir. Üretilen sayfa, sunucudaki bütün metodları listeler ve girilen bir parametre ile çıkacak sonuçları JSON formatında gösterir. Bu sayfa bir geliştirme ve hata ayıklama aracı olarak kullanılabilir ve son kullanıcı için değildir. (Gerçekte, sadece lokalde başlatılırsa gösterilir)

Bu bileşenlerle ilgili olarak pek çok teknik detay ve yapılandırma ayarları bulunmaktadır, ama bu makale için yukarıda anlattıklarım yeterli olacaktır. Sunucuyu çalıştırmanın gerçek etkisini ölçmeden önce 2 elemanı daha tartışmamız gerekmektedir. Bunlar; metodlarının gösterileceği sunucu sınıfın kodu ve HTML ve JavaScript kullanıcı arayüzünün rolüdür (bu kısmı sonra inceleyeceğiz).

--- Sihirbaz Tarafından Oluşturulan Örnek Sunucu Sınıfı ---

DataSnap REST Uygulama sihirbazı tarafından oluşturulan 3. unit dosyası, örnek sunucu sınıfı olup REST tarafından çağrılan sınıf metodlarını içerir. Bütün kodun büyük ölçüde sonlandığı yer olan, daha öncede de bahsettiğimiz OnGetClass olay tetikleyicisinin bulunduğu DSServerClass bileşenine bağlanan Delphi sınıfıdır.

Oluşturulan sınıfın iskeleti gayet basittir, sihirbazdaki örnek metodları istediğimizi yansıtır. İlgili kod aşağıdadır.

```
type
{$METHODINFO ON}
	TServerMethods1 = class(TComponent)
	private
		{ Private declarations }
	public
		{ Public declarations }
		function EchoString(Value: string): string;
		function ReverseString(Value: string): string;
	end;
{$METHODINFO OFF}
```

Unutmamak gerekir ki bu sınıf TComponent ana sınıfından türetilmiştir ve public türdeki metodlar için extra RTTI bilgisi üretecek özel bir derleyici seçeneğine sahiptir ({$METHODINFO ON}) (Delphi 2010 RTTI'yi genişletti)

EchoString ve ReverseString metodları girilen parametreyi veya tersini döndüren metodlardır. İlerleyen bölümlerde hangi veri türlerinin parametre olarak geçilebileceğini ve döneceğini göreceğiz.

--- REST Sunucusunu Derlemek ve Test Etmek ---

Artık sunucuyu derleyip çalışıp çalışmadığını görebiliriz. Programı derleyip çalıştırdıktan sonra Start (Başla) butonunu kullanarak sunucuyu açmak için basit kullanıcı arayüzünü kullanabiliriz. Sonra, varsayılan web kullanıcı arayüzünü açarız (Open Browser butonu ile), ama buna bakmadan önce REST yürütmesinin nasıl çalıştığına dair temelleri anlatacağım.

Test yapmak için bir web tarayıcısı açıp, sunucu metodlarından birinin URL bilgisini elle tarayıcıya gireriz. Bu URL bilgisi, uygulama konum adresi ve port numarası (localhost:8080), REST yolu (/datasnap/rest), sunucu sınıfının adı (TServerMethods1), çalıştırmak istediğimiz metodun adı (mesela EchoString) ve metoda gönderilecek parametre bilgisinden (burası opsiyonel, parametresiz bir metod da olabilir) oluşur.

localhost:8080/datasnap/rest/TServerMethods1/EchoString/hello%20world

Buradaki %20 ifadesi, boşluk yerine geçer, ama siz tarayıcınıza boşluk karakteri bırakabilirsiniz. Metodu çalıştıdrığınızda REST sunucusundan gelen JSON cevabını görebilirsiniz. (Makalede ilgili resme bakın)

Sunucudan dönen JSON değeri, içerisinde bir dizi değer olan "result" "sonuç" alanı olan bir nesnedir. Bu nesne, metodun sonucunu ve referans olarak gönderilen parametre'yi içerir (Bu biz dizidir çünkü dönecek birden fazla değer olabilir.)

--- Örnek HTML ve JavaScript İstemcileri ---

Elbette bu tür bir URL adresini direk tarayıcıya yazmak mantıklı değildir, ama tarayıcı tabanlı bir uygulama geliştirmenin de başlangıç noktasıdır. Gerçekte bu tür bir örnek uygulama DataSnap REST Uygulama Sihirbazı tarafından HTML ve JavaScript kodları yönünden oluşturulur. Eğer web sunucusunun varsayılan hareketini tarayıcı içerisinde çalıştırırsak, aşağıdaki gibi bir şey görürüz (Makalede ilgili resme bakın)

Bu iskeleti ReverseString.html dosyasında saklanan ve uygulamanın web modülündeki ReverseString sayfasından dönen HTML sayfasıdır. Sayfanın HTML kodu (eğer bağlantı ve ilerde bahsedilecek hususlarla ilgili kodları yok sayarsak) oldukça basittir.

```
<#serverfunctioninvoker>
<h1>DataSnap REST Project</h1>

<div>Page loaded at <span id="timeElement"></span></div>
<div id="contentdiv" class="contentdiv">
        <table>
                <tr>
                        <td>
                                        <input id="valueField" type="text"
value="A B C" />
```

```
                                </td>
                                <td>
                                                <button
onclick='javascript:onReverseStringClick();'>
                                                ReverseString</button>
                                </td>
                        </tr>
                </table>
        </div>
```

karakteri ile verilen özel etiketler PageProducer'ın (sayfa üretici) OnTag olay tetikleyicisi tarafından işlenir ve sayfaya her sorduğunuzda dinamik olarak Delphi uygulaması ile yer değiştirir. Bu etiketler sınıfın yürüteceği etiketleri içerir, "server function invoker" "sunucu fonksiyon yürütücüsü" test sayfasının mevcutluğu ve bir kaç şey gibi.

```
procedure TWebModule2.ServerFunctionInvokerHTMLTag(
Sender: TObject; Tag: TTag; const TagString: string;
TagParams: TStrings; var ReplaceText: string);
begin
        if SameText(TagString, 'urlpath') then
                ReplaceText := string(Request.InternalScriptName)
        else if SameText(TagString, 'port') then
                ReplaceText := IntToStr(Request.ServerPort)
        else if SameText(TagString, 'host') then
                ReplaceText := string(Request.Host)
        // and so on...
```

Yukarıdaki HTML kodunu okurken tahmin edebileceğiniz gibi anahtar rol, aşağıdaki kodu içeren onReverseStringClick JavaScript metoduna aittir. (aynı HTML dosyasının bir parçası)

```
function onReverseStringClick()
{
        var valueField = document.getElementById('valueField');
        var s = serverMethods().ReverseString(valueField.value);
        valueField.value = s.result;
}
```

Bu metod girdi alanındaki referansı saklar, buradaki değeri ReverseString metoduna parametre olarak gönderir, ve girdi alanındaki değeri sonuç ile günceller. serverMethods fonksiyonu serverFunction.js kaynak kod dosyasında (bu dosya sunucunun her tekrar derlenmesi ve çalıştırılmasında DSProxyGenerator bileşeni tarafından güncellenir) tanımlanan bir JavaScript proxy nesnesinden bir varlığı döndürür.

Bu Proxy nesnesi, Delphi sunucu metodlarını, HTTP isteklerini eş zamanlı olmayan (asenkron) bir şekilde sarmalayarak (AJAX teknolojisini kullanarak) JavaScript tarayıcı tabanlı uygulamalar tarafından kolaylıkla erişilebilir hale getirir.

```
function TServerMethods1(connectionInfo)
        ...
        this.ReverseString = function(Value) {
        var returnObject = this.executor.executeMethod(
```

```
        'ReverseString', "GET", [Value], arguments[1],
        true, arguments[2], arguments[3]);
        if (arguments[1] == null) {
                if (returnObject != null && returnObject.result != null
                && isArray(returnObject.result)) {
                        var resultArray = returnObject.result;
                        var resultObject = new Object();
                        resultObject.Value = Value;
                        resultObject.result = resultArray[0];
                        return resultObject;
                }
                return returnObject;
        }
};
```

Bu, REST sunucusu tarafından gösterilen sınıfların RTTI bilgilerinin Delphi tarafından oluşturulmuş JavaScript kodudur. Bu JavaScript metodu parametre ve dönüş değerini yönetir, ilkini URL içinde geçerek ve ikincisini REST metodunun döndürdüğü JSON nesnesinden döndürerek yapar. Bu JavaScript tarayıcı uygulamaları tarafından REST sunucu metodlarının göreceli olarak basitçe çağrıldığını gösterir.

AJAX desteği sağlamak, JSON encoding (şifreleme) ve buna benzer pek çok çağrılabilecek JavaScript destek dosyaları vardır, ama bunlar derinine inmek için oldukça karmaşıktır ve bu makalenin konusu değildir.

--- Delphi İstemcisinden REST Sunucusunu Çağırma ---

Artık sunucuyu inşa ettiğimize, sunucu uygulamanın web uygulamasını bir istemci tarayıcıya nasıl gösterdiğini gördüğümüze göre aynı sunucu için bir Delphi istemci uygulaması yazmaya başlayabiliriz. Burada 2 farklı yaklaşım uygulayacağız.

İlk yaklaşım, Delphi XE'de tanımlanan özelleşmiş REST istemci desteğini kullanarak bir Delphi DataSnap istemcisi oluşturmaktır. İkinci yaklaşım tek başına bir REST istemcisi oluşturmak, IdHTTP bileşeni ile sunucuyu çağırmak ve DBXJSON unit dosyasını kullanarak sonucu ayrıştırmaktır. RestXeOneClient demosunda, bu iki yaklaşımı da kullandım, elle (manuel) olanla başladım ve sonra otomatik desteği aldım, ki bu genelde tercih edilendir.

Öncelikle standart bir VCL uygulaması oluşturdum ve ana formuna bir buton , bir edit box ve IdHTTP istemci bileşenini ekledim. Butonun OnClick olay tetikleyicisine aşağıdaki kodu yazdım.

```
const
        strServerUrl = 'http://localhost:8080';
        strMethodUrl = '/datasnap/rest/TServerMethods1/ReverseString/';

procedure TRestClientForm.btnManualClick(Sender: TObject);
var
        strParam: string;
begin
        strParam := edInput.Text;
        ShowMessage (IdHTTP1.Get(strServerUrl + strMethodUrl + strParam));
end;
```

Bu çağrı uygun bir URL adresini, sunucu adresi, REST sunucusu içerisinde verilen metodun adresi ve parametre bilgisi ile birleştirerek oluşturur. Eğer edit box bileşeninin içerisine "Hello World" değerini yazarsanız, aşağıdaki resimdeki sonuca ulaşırsınız.

Şimdi daha da ilginç olanı sunucudan dönen JSON veri yapısını daha önceki "Delphi'de JSON Ayrıştırma" konusunda işlediğimiz şekilde Delphi'nin desteğini kullanarak ayrıştırmaktır.

Gördüğümüz gibi sunucudan dönen JSON verisi string'dir, ama REST sunucusu isimlendirilmiş bir değeri olan bir nesne oluşturur ve bunu dizideki güncel değerine koyar. Bu yüzden metodun son versiyonunda HTTP'den gelen sonucu JSON veri yapısına ayrıştırdıktan sonra nesneden içerdiği çifte, ve çiftten tek eleman dizisi haline çevirmek gerekiyor. İlgili kod aşağıdadır.

```
procedure TRestClientForm.btnManualClick(Sender: TObject);
var
	strParam, strHttpResult, strResult: string;
	jValue: TJSONValue;
	jObj: TJSONObject;
	jPair: TJSONPair;
	jArray: TJSOnArray;
begin
	strParam := edInput.Text;
	strHttpResult := IdHTTP1.Get(strServerUrl +
	strMethodUrl + strParam);
	jValue := TJSONObject.ParseJSONValue(
	TEncoding.ASCII.GetBytes(strHttpResult), 0);
	try
		jObj := jValue as TJSONObject;
		jPair := jObj.Get(0); // get the first and only JSON pair
		jArray := jPair.JsonValue as TJsonArray; // pair value is an array
		strResult := jArray.Get(0).Value; // first-only element of array
		edInput.Text := strResult;
	finally
		jValue.Free;
	end;
end;
```

Tekrar etmek gerekirse karmaşıklık, sunucudan dönen veri yapısından kaynaklanmaktadır, diğer durumlarda olduğu gibi JSON sonucunu ayrıştırarak erişmek çok daha kolaydır.

--- REST İstemci Desteğini Kullanmak ---

İstemci taraftaki kodu daha kolay yazmak için ve sade REST arayüzüne bağlı kalmak için (DataSnap HTTP desteğine bağlı kalmamak için) Delphi XE'de bulunan DataSnap REST İstemci Modülü Sihirbazını kullanmalıyız. Bu sihirbazı çalıştırmadan önce sunucu uygulamasının çalıştığından ve Delphi hata ayıklayıcısı (debugger) tarafından çalıştırılmadığından emin olmamız gerekir. Bunun için Delphi IDE'sinde bulunan Run Without Debugging (Hata Ayıklamadan Çalıştır) seçeneğini kullanabilirsiniz.

Sihirbazı çalıştırdığınızda, ilk sayfada size sunucunun lokalde mi yoksa uzak bir makinada mı (genelde lokal seçilir) ve ikinci sayfada sunucunun hangi DataSnap mimarisini temel aldığını sorar.

Bu örnek için yukarıdaki resimde yaptığım gibi "Stand-alone WebBroker Application" "Tek Başına Çalışan WebBroker Uygulaması" seçeneğini seçmelisiniz. 3. sayfada sihirbaz size sunucunun port numarasını, DataSnap içeriğini (varsayılan olarak /datasnap), ve opsiyonel olan hesap bilgisini sorar. Bitir dediğiniz zaman sihirbaz projenize otomatik olarak 2 unit dosyası daha ekler.

- REST bağlantı yapılandırmaları ile, DSRestConnection bileşenine ev sahipliği yapan bir data modülü içeren istemci modül unit dosyası
- Sunucuyu çağırmak için proxy sınıfları içeren bir unit dosyası. Geleneksek DataSnap bağlantısı için üretilen istemci sınıflardan farklı olarak bu sınıf, sunucuyu yürütmek için DBXConnection bileşeni yerine DSRestConnection bileşenini kullanır.

Unutmamak gerekir ki burada proxy nesnesini elle oluşturmanıza gerek yoktur, çünkü sihirbaz tarafından eklenen özel bir özellikle istemci modül tarafından bu nesne otomatik olarak oluşturulur.

```
// property ServerMethods1Client: TServerMethods1Client
function TClientModule1.GetServerMethods1Client: TServerMethods1Client;
begin
        if FServerMethods1Client = nil then
                FServerMethods1Client:= TServerMethods1Client.Create(
        DSRestConnection1, FInstanceOwner);
        Result := FServerMethods1Client;
end;
```

Bu demektir ki biz sunucuyu istemci modülün özelliğini referans olarak göstererek yürütüebilir ve aynı şekilde metodu da yürütebiliriz. Söylemem gereksiz olacak ama daha önce yazdığım metodu direkt çağırarak çıkan JSON sonucunu ayrıştırmak yerine aşağıda yazacağım kod (bu teknoloji ile yapılan) çok daha basit olacaktır.

```
procedure TRestClientForm.btnProxyClick(Sender: TObject);
begin
        edInput.Text := ClientModule1.ServerMethods1Client.
        ReverseString(edInput.Text);
end;
```

İki teknik de tamamen aynı sonuca ulaşır. Aklınıda tutmanız için burada ana kavramlar, Delphi DataSnap REST Sunucusu oluşturmak, web uygulaması inşa etmek, Delphi istemcisinin bunu çağırmasına erişilebilir kılmak, ve her hangi bir dilde yazılmış her hangi bir istemci ile sunucuyu çağırabilmenin mümkün olmasıdır (Burada yapılan şey programlama dili farketmeksizin bir HTTP çağrısı oluşturarak sonuç olarak bir JSON veri yapısı elde etmektir).

--- Sunucu Fonksiyon Yürütücüsü ---

DataSnap REST Uygulama sihirbazı tarafından oluşturulan sunucunun daha önce tartışmadığımız tek bir özelliği kaldı. Bu da Sunucu Metod Yürütücüsü olup, sunucuyu kullanılabilir bir hale getirmek için çeşitli fonksiyonları test eden üreysel

(generic) bir mekanizmadır. Bu özellik web modülü üzerindeki 2. sayfa üreticisi, ilişkili bir HTML template'i ve bir kaç JavaScript dosyası tarafından desteklenmektedir.

Basitçe "/ServerFunctionInvoker" yolunu girerek, veya ReverseString sayfasındaki yolu takip ederek, sunucu tarafından gösterilen sınıflarla ilişkili 2 alan içeren bir HTML sayfası elde ederiz. Bu sınıflar yönetimsel sınıf olan DSAdmin'e ek olarak uygulama ile ilişkili özel sınıftır (Bu örnek için TServerMethods1). Sınıf tarafındaki artı işaretine tıklarız ve sayfa bize metodların listesini gösterecek şekilde genişler. (Makalede ilgili resme bakın)

Eğer burada bir metod seçerseniz, sayfanın arkasındaki JavaScript kodu girdi alanlarını otomatik olarak oluşturur ki bu alanlar metoda girilecek olan parametre alanlarıdır. Buraya bir değer girip kenardaki bir butona tıkladıktan sonra JSON cevabını görebilirsiniz.

Hedef sunucu sınıfta herhangi bir güncelleme, değişiklik veya genişletme işlemi yaptığınız zaman veya sunucuya yeni sınıflar eklediğiniz zaman bu sayfa kendini otomatik olarak günceller. Bu sayfa, hata ayıklama ve test için oldukça faydalıdır.

--- Kimlik Doğrulama ve Yetkilendirme ---

DataSnap REST uygulamasının basit özelliklerini gördükten sonra sihirbazın oluşturduğu kod içerisindeki (sihirbazda bu özelliği atlayabilirsiniz) özel bir özelliğe göz atabiliriz. Bu da kimlik doğrulama (authentication) desteğidir

Kimlik doğrulama yönetimi 2 parça halinde anlatılacaktır. İlki oturum (session) yönetimi olup aynı istemci makinadan gelen farklı (durumsuz - stateless) HTTP isteklerini nasıl çözüldüğünü anlatır. İkincisi ise yetkilendirme yönetimi olup, hangi kullanıcıların hangi sunucu metodlarını çağırabileceği ile ilgili olarak bir yönetimdir. Bu 3 başlık da bu ana konu içerisinde ele alınacaktır.

Bunun için kimlik doğrulama ve yetkilendirme seçeneklerini sçerek yeni bir DataSnap REST uygulaması oluşturmakla işe başlıyorum. Sonuç uygulamada web modülünde DSAuthenticationManager adında ek bir bileşen bulunacaktır. Bu da DSHTTPWebDispatcher bileşeninin AuthenticationManager özelliğine referans olarak kullanılacaktır.

--- Oturumlar ---

Yetkilendirme desteğine bakmadan önce DataSnap'de oturum yönetiminden bahsetmek istiyorum. En başta bir istemci sunucuyu çağırır, bu yeni bir oturum ID bilgisi oluşturur ve bunu geleneksel bir başlık (header) ("dssession") içinde döndürür. İstemci tarafından gelen ard arda istekler oturum ID bilgisi ile birlikte "Pragma" başlığı içermeli, böylece sunucu gelen taleplerin aynı istemciden geldiğini anlayabilmeli.

Bir oturum zaman aşımına uğradığı zaman (varsayılan olarak 1200 saniye yani 20 dakika) biter ve bu özel bir CloseSession URL adresi kullanılarak kapatılmaya zorlanır. Eğer olmayan bir oturuma bu URL geçilmeye çalışılırsa (mesela bu arada sunucuyu kapatıp tekrar açtığınızı düşünelim) bir istisnai durum (exception) alırsınız ve oturum ID bilgisi sıfırlanır.

Sunucu tarafında o anki oturuma aşağıdaki gibi ulaşabilirsiniz.

```
var
        session: TDSSession;
begin
        session := TDSSessionManager.GetThreadSession;
```

Ayrıca string değerlerini ve PutData, HasData, GetData, RemoveData metodlarını kullanarak oturum verisine isim-değer çiftleri ekleyebilirsiniz.

JavaScript istemcisi kullandığınız zaman, sihirbazın oluşturduğu destek kodu, global bir JavaScript değişkeni ile ($$SessionID$$) oturum ID bilgisini yönetir ve aynı sayfa içerisindeki her AJAX çağrısında bu değer parametre olarak geçer (pass). Eğer sayfayı değiştirirseniz veya yenilerseniz oturum ID bilgisi kaybolur.

Bununla beraber JavaScript destek kodunu, oturum ID bilgisini tarayıcı çerezlerine (cookie) eklemek, böylece üzerindeki uygun sayfalarda bu bilginin mevcut olmasını sağlamak için zorlayabilirsiniz. Bunu yapmak için her HTML dosyasındaki onLoad() JavaScript fonksiyonuna şu çağrıyı eklemeniz gerekir.

```
initSessionData(true);
```

Yapmak isteyebileceğiniz ikinci bir değişiklik de loginRequired değişkeninin yönetimini değiştirmek olabilir ama bu ekstra satır kodlar için makaledeki kimlik doğrulama bölümüne bakmanız gerekiyor.

--- Kimlik Doğrulama (Authentication) ---

DSAuthenticationManager bileşeni OnUserAuthenticate ve OnUserAuthorize adında 2 adet anahtar olay içerir. İlki, sunucuya kimlik doğrulaması yapılmamış bir kullanıcıdan ne zaman bir istek gelse çalışır. Kullanıcı ve oturumunun kimlik doğrulaması gerçekleştikten sonra bu olay tetiklemesi bir daha çalışmaz. Eğer gerçek güvenliği aramıyorsanız, bu metodu nasıl yazacağınızı belirtir. İlgili kod aşağıdadır.

```
procedure TWebModule2.DSAuthenticationManager1UserAuthenticate(
Sender: TObject; const Protocol, Context, User, Password: string;
var valid: Boolean; UserRoles: TStrings);
var
        session: TDSSession;
begin
        valid := (user = password) and (user <> '');

        if valid then
        begin
                Session := TDSSessionManager.GetThreadSession;
                Session.PutData ('username', user);
```

Genelde kullanıcıların ve şifrelerinin geçerli olup olmadığını öğrenmek için veritabanına, yapılandırma dosyasına veya işletim sistemi mekanizmasına bakılır. Bu olayda, ben boş olmayan herhangi bir kullanıcı adı ile çakışan şifreyi kabul ediyorum. Yukarıdaki kodda kullanıcı adını ayrı bir oturum parametresi olarak ekliyorum ki, bu bilgi oturum nesnesinin kullanıcı adı özelliğinde zaten var olan bir bilgi olduğu için hiç de kullanışlı bir şey değildir.

Diğer taraftan GetUserName adında yeni bir sunucu fonksiyonu ekledim. Bu fonksiyon, oturum kullanıcı adını ve daha önce oluşturduğum ekstra oturum parametre değerini geri döndürüyor. İlgili kod aşağıdadır.

```
function TServerMethods1.GetUserName: string;
var
        Session: TDSSession;
begin
```

```
        Session := TDSSessionManager.GetThreadSession;
        Result := Session.GetData ('username') + ':' + Session.Username;
end;
```

Oturum bilgisi bir yana, eğer bir REST URL'sini tarayıcı üzerinden direk yürütmek isterseniz, standart bir HTTP başlantı talebi alırsınız. JavaScript kullanıcı arayüzünde bunun yerine JavaScript'in başlangıç kodunda referansı olan <#loginrequired> alanını kullanan ve sunucu tarafından bilgilendirilen bir DSAuthenticationManager bileşeni yerleştirilir.

```
var loginRequired;
function onLoad()
{
        loginRequired = <#loginRequired>;
```

Şimdi sunucu tarafında bu değer aşağıdaki kaynak kod tarafından işlenir.

```
  else if SameText(TagString, 'loginrequired') then
        if DSHTTPWebDispatcher1.AuthenticationManager <> nil then
                ReplaceText := 'true'
        else
                ReplaceText := 'false'
```

LoginRequired değişkeni etkinleştirildiğinde, tarayıcı tabanlı uygulama standart kullanıcı arayüzünü gizler ve standart HTML template'i olan bir bağlantı formu gösterilir. Bu, HTML sayfasının iki parçası olan logindiv ve contentdiv parçalarının duruma göre birinin gösterilip birinin gizlenmesi ile ilgili olan ShowLogin fonksiyonunda yer alır.

Web sunucusunun ana sayfasını ilk kez açtığınızda beklenen kullanıcı arayüzü değil de bir bağlantı iletisi görürsünüz (Unutmamak gerekir ki bu örnekte daha güzel görünmesi açısından basit bir CSS uyguladım) (Makalede ilgili resme bak)

Kullanıcı adı ve şifre ile bağlandığı zaman, standart arayüzü görürsünüz. Şimdi sayfayı yenilerseniz oturum bilginiz kaybolacak ve tekrar bağlanmak zorunda kalacaksınız. Daha önce de belirttiğim gibi oturum yönetimini çerez olarak düeltmek gibi bir yöntem vardır, ama gene de bir bağlantı iletisi ile karşılaşırsınız, çünkü bağlantı isteği, oturumun o anki durumundan çok Kimlik Doğrulama Yöneticisinin (Authentication Manager) varlığına bağlıdır. Bu da lokal oturum değerine test yapan ilk JavaScript kodunu aşağıdaki kod ile değiştirerek çözülebilir.

```
var loginRequired = false;

function onLoad()
{
        initSessionData(true);
        showTime();
        loginRequired = (getSessionID() == null);
```

Diğer bir deyimle, bağlantı işlemi, sadece istemci uygulamanın geçerli bir oturum ID bilgisi olmadığı zaman talep edilir. (getSessionID() fonksiyonu aynı zamanda oturum ID bilgisinin zaman aşımına uğrayıp uğramadığını da kontrol eder)

--- Roller ve Yetkilendirme ---

Sunucu bir kere istemci uygulamanın kimlik doğrulamasını yaptıktan sonra, istemci uygulama, sunucu tarafından sergilenen bütün metodları çağırabilir. AuthenticationManager bileşeninin OnUserAuthorize olay tetikleyicisi içerisinde metodları kullanıcıya bağlı olarak filtreleyebilirsiniz. Bu oldukça güçlüdür ama can sıkıcıdır. Unutmamak gerekir ki eğer OnUserAuthorize olay tetikleyicisini kullanmayı engellemeye karar verirseniz, sunucu tarafından valid (geçerli) referans değeri True olarak atanan kod satırını kaldırmanız gerekmektedir.

Daha basit bir alternatif olarak, DataSnap mimarisinde kullanıcı rolleri için hazır destek vardır. Kullanıcının kimlik doğrulamasını yaparken, kullanıcıya bir veya daha fazla rol atayabiliriz. Bu amaçla örnekteki OnUserAuthenticate olay tetikleyicisinin kodunun içine şu ekstra kodu ekleriz.

```
if valid then
begin
        UserRoles.Add ('standard');
        if user = 'admin' then
                UserRoles.Add ('admin');
```

Her kullanıcı standart role eklenir, ama idarecinin ek olarak admin rolü bulunmaktadır. Bu bir string list olabilir ve her kullanıcıya birden çok rol eklenebilir. Ama hangi metodun kullanıcı rolüne bağlı olarak çağrıldığını nasıl bileceğiz? Bu da sunucu sınıfa TRoleAuth geleneksel özelliğinin (custom attribute) tamamının veya metodlarından bir kaçının eklenmesiyle çözülebilir. (Geleneksel özellik, ilk kez Delphi 2010'da gösterilen bir dil özelliğidir ve bununla ilgili olarak detaylı bilgiyi Delphi 2010 Handbook kitabımda bulabilirsiniz.)

```
uses
        DSAuth;

type
{$METHODINFO ON}
TServerMethods1 = class(TComponent)
        public
                [TRoleAuth ('admin')]
                function EchoString(Value: string): string;
                [TRoleAuth ('standard')]
                function ReverseString(Value: string): string;
                [TRoleAuth ('standard')]
                function GetUserName: string;
```

Unutmamak gerekir ki burada önemli olan nokta, eğer unit dosyasının uses kısmında DSAuth değerini eklemezsek, derleyici geleneksel özelliği anlamaz, belirsiz bir uyarı üretir (Unsupported language feature: 'custom attribute') (Desteklenmeyen dil özelliği: 'Geleneksel özellik'), ve roller uygulamanız için etkinleştirilemez. Ayrıca TRoleAuth özellik parametresi birbirinden virgülle ayrılan rol listesi olabilir ve alternatif olarak reddedilen rollerin listesinin tanımlandığı ikinci bir parametre de eklenebilir.

Belirtilen özelliklerle, bütün kullanıcılar ReverseString ve GetUserName metodunu çağırabilir ama echoString metodu sadece admin yetkisinde olan personel tarafından çağrılabilir (Yukarıdaki koda bakın). Eğer herhangi bir kullanıcı (admin yetkisi olmayan) EchoString metodunu çağırmayı denerse, sunucu istemciye JSON cevabı olarak bir istisnai durum (exception) gönderir. Tarayıcıda Sunucu Fonksiyon Yürütücü sayfasında örnek bir hata görülebilmektedir. (Makalede ilgili resme bakın)

--- DataSnap REST Sunucusunun Veri Türleri ---

Şimdi size oturumlar, kimlik doğrulama ve yetkilendirme ile ilgili birtakım bilgiler verdiğime göre yeni bir başlığa geçebiliriz. Bu ilk demoda sunucu metodları hep string döndürüyor. Kesinlikle daha doğal türler olan Tamsayılar, Karakterler ve Boolean türde değerler de kullanabilirsiniz.

Peki ya daha karmaşık veri türleri? DataSnap nesne, dataset ve stream desteği sağlamasının yanı sıra, JSON cevaplarını direk olarak yönetme imkanı verir. Ayrıca bu doğal yapıların bazılarının otomatik olarak filtreleme ve önbelleğe yerleştirme (cache) desteği de vardır. Bütün bunlar makalenin bu bölümünde ele alacağım konulardır. Bu özellikleri gösterebilmek için sunucu sınıfı pek çok farklı veri türü döndüren tek başına geniş bir uygulama oluşturdum.

Unutmamak gerekir ki bu bölümde sunucu taraftaki kod ve oluşturulan JSON veri yapıları üzerine yoğunlaşacağım. Ama sonraki konu olan “jQuery and the Data Types Demo” "jQuery ve Veri Türleri Demosu" konusunda bu sunucudaki çoğu metodun nasıl çağrılacağını gösterecek bir HTML istemci uygulaması oluşturacağım. Bu gecikmenin sebebi ise, öncelikle jQuery'yi tanıtıp sonrasında bunu istemci taraftaki uygulama geliştirmek için kullanmak istememdir.

--- Sıralı Türler ---

DataSnap REST sunucusu, mümkün olduğu zaman ilişkili JSON yapılarına eşlenmiş doğal türleri döndürür. Aşağıda basit türler döndüren bir takım fonksiyonlar yer almaktadır.

```
public
        function AddInt (a, b: Integer): Integer;
        function AddDouble (a, b: Double): Double;
        function IsOK: Boolean;
        function GetLargeString (nCount: Integer): string;
```

Aşağıdakiler de bu dört metoddan dönen örnek sonuçlardır (Sonuncusu, girilen karakter sayısı parametre alınarak ve her 10 karakterden sonra bir x karakteri eklenerek bir string değer döndürmektedir, neden böyle olduğu ilerde anlatılacaktır.)

```
Executed:   TServerMethods1.AddInt
Result:     {"a":"1","b":"2","result":3}

Executed:   TServerMethods1.AddDouble
Result:     {"a":"1.5","b":"4.4","result":5.9}

Executed:   TServerMethods1.IsOK
Result:     {"result":true}

Executed:   TServerMethods1.GetLargeString
Result:     {"nCount":"20","result":"ooooooooxooooooooox"}
```

İlk 3 durumda sunucu, geri dönen değeri doğal JSON formatında (sayı veya boolean gibi) döndürüp string (tırnak içine alınmış) formatında döndürmüyor. Set ve enum değerler gibi diğer basit veri türleri direkt olarak döndürülemez. Aynı şey tarih için de geçerlidir. Kayıt (record) gibi kullanıcı-tanımlı bazı veri yapıları da yönetilemez.

--- Nesneler ve Marshaling ---

Çok önemli bir özellik, REST sunucusu içeren bir DataSnap sunucusundan bir nesne döndürme yeteneğidir. Delphi 2010'da bunu TJSONMarshal (ve TJSONUnmarshal) destek sınıflarını kullanarak becerebilirken, Delphi XE'de bu işlem otomatik olarak yer almaktadır.

Marshaling desteği, özel alanlar için RTTI kullanımına dayanır. Delphi geliştiricileri genelde bir nesnenin verisini onun yayınlanmış özelliklerinin (published properties) bulunduğu set olarak düşündüklerinden, bu durumda nesnenin dahili verisini hesaba katmamız gerekir. Elbette alanların bütün veri türlerine değinilmiyor, ya da kısmen değiniliyor. DBXJSONReflect unit dosyasının durumundaki yorum gibi;

"Tamsayı, string, Karakter, Enum, Kesirli Sayı, Nesne türleri için otomatik olarak bir çevrim ve eski haline geri döndürme özelliği bulunmaktadır. Metod, varyant, arayüz, işaretçi, dynArray, classRef, dizi türleri ise alan değerleri yoksayılan ve kullanıcı tarafında bir çevrim gerektiren türlerdir."

Diğer alan türleri için bu unit dosyasında hazır olarak bulunan veya kendi yazacağınız kullanıma hazır bir çevrimi etkinleştirebilirsiniz. Çevrimi JSONReflect özelliğini kullanarak etkinleştirebilirsiniz. Bununla beraber bütün seçenekleri gözden geçirmek için vaktim olmadığından (neden olduğunu sonra açıklayacağım), burada bir kaç örnek vereceğim.

Öncelikle aşağıdaki gibi basit bir nesne yapısını ele alalım

```
type
TPerson = class
private
        FAge: Integer;
        FLastName: string;
        FFirstName: string;
        procedure SetAge(const Value: Integer);
        procedure SetFirstName(const Value: string);
        procedure SetLastName(const Value: string);
public
        property LastName: string read FLastName write SetLastName;
        property FirstName: string read FFirstName write SetFirstName;
        property Age: Integer read FAge write SetAge;
end;
```

Bu da nesne döndüren bir metoddur.

```
function TServerMethods1.GetPerson: TPerson;
begin
        Result := TPerson.Create;
        Result.LastName := 'Smith';
        Result.FirstName := 'Joe';
        Result.Age := 44;
end;
```

Bu da sonuç olarak gösterilen JSON veri yapısıdır.

```
Executed:   TServerMethods1.GetPerson
Result: {
```

```
        "result": {
                "type":"ServerMethodsUnit1.TPerson",
                "id":1,
                "fields": {
                        "FAge":44,
                        "FLastName":"Smith",
                        "FFirstName":"Joe"
}}}
```

Şimdi TStringList türünden bir nesne döndürdüğümüzü düşünelim. Alanların dahili setini almak işe yaramaz çünkü gerçek datayı içermez, bir işaretçi tarafından refere edilir.

```
function TServerMethods1.GetNames: TStringList;
begin
        Result := TStringList.Create;
        Result.Add('one');
        Result.Add('two');
        Result.Add('three');
end;
```

Alacağınız kullanışsız JSON verisi şu şekilde olacaktır. (Kullanışsızdır, çünkü esas içerik olan stringlist içerisindeki string değerleri burada değildir.)

```
Executed:   TServerMethods1.GetNames
Result:     {
        "result":{
        "type":"Classes.TStringList",
        "id":1,
        "fields":{
                "FCount":3,
                "FCapacity":4,
                "FSorted":false,
                "FDuplicates":"dupIgnore",
                "FCaseSensitive":false,
                "FOwnsObject":false,
                "FEncoding":null,
                "FDefaultEncoding":{omitted},
                "FDelimiter":"",
                "FLineBreak":"",
                "FQuoteChar":"",
                "FNameValueSeparator":"",
                "FStrictDelimiter":false,
                "FUpdateCount":0,
                "FWriteBOM":true
}}}
```

Farz edelim ki sınıfımızda bir TStringList alanı var. Daha önce kullandığımız TPerson sınıfında olduğu gibi

```
type
```

```
TPersonData = class (TPerson)
private
        FDateOfBirth: TDate;
        FMoreData: TStringList;
        procedure SetMoreData(const Value: TStringList);
        procedure SetDateofBirth(const Value: TDate);
public
        constructor Create;
        destructor Destroy; override;
        property MoreData: TStringList read FMoreData write SetMoreData;
        property DateofBirth: TDate
        read FDateofBirth write SetDateofBirth;
end;
```

Burada 2 adet problemimiz vardır. Dönecek değer sayısal gösterim ve içindeki gerçek veri olmayan string list şeklinde olur ki bunu yukarda gösterdik zaten. Bunu çözmek için çevrim metodlarını etkinleştirmek (TStringList için) veya yazmak gerekir (TDate için). Daha önce de bahsettiğim gibi bu Delphi XE'de JSONReflect özelliği kullanarak yapılabilmekte. Bu özellik belli sayıda alana ve sınırlanmış dökümantasyona sahip olduğundan kullanması kolay değildir ama burada tekrardan derinlerine inmek istemiyorum

İki özellikle sınıf tanımını görmektesiniz (Aşağıda). İlk JSONReflect tarih nesnesini string türüne ServerProjectWithUserTypes demosunda (Delphi XE resmi DataSnap demolarından bir parça) bulduğum TISODateTimeInterceptor destek sınıfı ile eşleştirmekte. Diğer JSONReflect ise string list'i verileri ile birlikte bir nesneye eşleştirmektedir.

```
type
TPersonData = class (TPerson)
private
        [JSONReflect(ctString, rtString,
        TISODateTimeInterceptor, nil,true)]
        FDateOfBirth: TDate;

        [JSONReflect(ctTypeObject, rtTypeObject,
        TStringListInterceptor,nil,true)]
        FMoreData: TStringList;

        procedure SetMoreData(const Value: TStringList);
        procedure SetDateofBirth(const Value: TDate);
public
        ... as above
```

Eğer bu sınıfta nesneyi verilerini doldurarak döndürecek bir metod yazarsak, aşağıdaki gibi string list'teki veri ve esas string değerlerine sahip olan şu JSON'u elde ederiz.

```
Executed:   TServerMethods1.GetPersonWithData
Result:     {
        "result":{
        "type":"ServerMethodsUnit1.TPersonData",
        "id":1,
```

```
        "fields":{
                "FDateOfBirth":"1899-12-30 00:00:00",
                "FMoreData":{
                "type":"DBXJSONReflect.TSerStringList",
        "id":2,
        "fields":{
                        "FSerStringItemList":[
                        {"type":"DBXJSONReflect.TSerStringItem",
        "id":3,
        "fields":{"FString":"Hello", "FObject":null}},
                        {"type":"DBXJSONReflect.TSerStringItem",
        "id":4,
                "fields": {"FString":"Joe","FObject":null}}
        ],
        "FSorted":false,
        "FDuplicates":"dupIgnore",
        "FCaseSensitive":false
        }},
        "FAge":44,
        "FLastName":"Smith",
        "FFirstName":"Joe"
}}}
```

Dahili veri yapısı basit değildir, ama string list hakkında baya dahili bilgileri sergiliyoruz. (potansiyel olarak ona bağlanan nesnelerle de ilgili) Şimdi bu güçlü yana bakıp, neden yaklaşımın bu tarafında olduğumu merak ediyorsunuzdur. Çok fazla ilginç bulduğum nokta, nesneden JSON'a yapılan Marshaling Delphi'den Delphiye olan iletişim katmanlarında mantıklıdır. Benim odaklandığım nokta ise, tarayıcıyı bir istemci olarak kullanmak yerine, Delphi'nin dahili sınıf isimleri ve veri yapılarına dayanan bir nesne gösteriminin sınırlandırılmış olarak kullanılmasıdır. JavaScript'in buna kesinlikle ihtiyacı yoktur.

Alternatif olarak ilerdde göreceğiniz gibi, JavaScript tarafında daha iyi ve direk yönetim için, direk TJSONObject döndürmektir. Benim tercih ettiğim ve Delphi'nin belirlenmiş doğal desteğini bitirdikten sonra üzerine yoğunlaşacağım konu bu olacak.

--- Veri Seti ve Akışlar (Dataset and Streams) ---

DataSnap REST sunucusunun otomatik ele aldığı ilk ilişkili karmaşık veri yapısı datasetdir (veri seti). TDataSet bileşeninden türeyen herhangi bir sınıfı nesne olarak döndürebilir ve tamamen JSON'a çevirebilirsiniz.

Örnek olarak tamamen bir ClientDataSet döndüren aşağıdaki sunucu metoduna bakabiliriz.

```
function TServerMethods1.GetCustomers: TDataSet;
begin
        ClientDataSet1.Open;
        ClientDataSet1.MergeChangeLog; // clean up, just in case
        Result := ClientDataSet1;
end;
```

Bileşenin basit bir yapılandırması vardır, veriyi programın olduğu dizinle aynı bir dosya içerisine yüklemek gibi.

```
object ClientDataSet1: TClientDataSet
        FileName = 'customer.cds'
End
```

Sonuç JSON'dan aşağıdaki alıntıda, Delphi data ve metadata'yı ayırarak döndürecek ve veriyi kolonlara göre sıralayacaktır. (Biraz kullanışsız bulduğum birşey)

```
Executed:   TServerMethods1.GetCustomers
Result:     {"result":
        {"table":[
                ["CustNo",7,0,0,0,8,0,0,false,false,0,false,false],
                ["Company",1,1,0,30,31,0,0,false,false,0,false,false],
                ["Addr1",1,2,0,30,31,0,0,false,false,0,false,false]...],
        "CustNo":[1221,1231,1351,1354,...],
        "Company":["Kauai Dive Shoppe","Unisco",
        "Sight Diver Webinarist","Cayman Divers World Unlimited",...],
        "Addr1":["4-976 Sugarloaf Hwy","PO Box Z-547","1 Neptune Lane",
        "PO Box 541",...],
...}}
```

Tablo (table) dizisi, alan tanımlamalarına, son derece okunabilir olmasa da tür ve diğer ayarlamalara sahiptir, takip eden alanlar da kolonları, her biri tamamen bir değer seti ile temsil eder. Görebileceğiniz gibi belli kolon değerleri arasındaki, belirli bir kayıt sayısı ile başlayan, kısaca sınırlı bir dataset döndürmek de mümkündür. Bu uzantıya filtreleme ile ilgili konuda detaylı olarak bakacağız.

DataSnap REST Sunucusu tarafından doğal olarak yönetilen ikinci bir Delphi sınıfı TStreamdir (ve bundan türeyen sınıflar). Aşağıdaki sunucu metodu inceleyelim.

```
function TServerMethods1.GetImage: TStream;
var
        fStream: TFileStream;
begin
        fStream := TFileStream.Create('images\expand.png', fmOpenRead);
        Result := fStream;
end;
```

Eğer Sunucu Fonksiyon Yürütücüsü istemcisini kullanıcrsak, aşağıdaki gibi bir sonuç elde ederiz. (Bu sonucun ilk parçasıdır)

```
Executed:   TServerMethods1.GetImage
Result:     "‰PNG\r\n\u001a\n\u0000\u0000\u0000\...
```

Gördüğünüz gibi bu JSON değildir. İkili bir akış döndürdüğünüzde (binary stream) gerçekte, verinin gerçek halini almak istersiniz, text gösterimini değil. http://localhost:8080/datasnap/rest/TServerMethods1/getimage URL adresini İnternet Explorer üzerinde açarsanız, aşağıdaki resimde gösterildiği gibi resimi görürsünüz (Makalede ilgili resme bakın). Ama Google Chrome gibi diğer tarayıcılar bunu sunamaz çünkü HTTP cevabında uygun Content-Type Header (İçerik Türü Başlığı) eksikliği vardır.

Bununla beraber akışlar (stream) için URL adresine "json_true" özel değerini yazarak JSON türüne çevirmesini sağlayabiliriz. Eğer aşağıdaki URL adresini kullanırsanız, onun altındaki cevaba ulaşırsınız.

http://localhost:8080/datasnap/rest/TServerMethods1/getimage?json=true

{"result":[[137,80,78,71,13,10,26,10,0,0,0,...]]}

Unutmamak gerekir ki, eğer fonksiyon çağrısında herhangi başka bir var parametresi (değişken yani) yoksa, dönen sonucu sadece ikili (binary) format olarak destekler

--- JSON'u Direk Döndürmek ---

DataSnap REST tarafından direk destek verilen bir diğer veri türü de TJSONValue türü ve TJSONObject ve TJSONArray gibi alt sınıflarıdır. Daha önce de belirttiğim gibi eğer amacınız istemci taraftaki uygulamayı JavaScript ile yazmaksa, bu durum oldukça ilginç bir durumdur. Mesela bir nesnenin datasını, onunla ilişkili olduğu JSON nesnesini oluşturarak döndürebiliriz.

```
function TServerMethods1.GetPersonJson: TJSONObject;
begin
        Result := TJSONObject.Create;
        Result.AddPair ('LastName', 'Smith');
        Result.AddPair ('FirstName', 'Joe');
        Result.AddPair ('Age', TJsonNumber.Create (44));
end;
```

Bu isteğe gelecek cevap, JavaScript istemcisinden çok daha kolay bir şekilde ele alınacak düz JSON veri yapısıdır.

```
Executed:   TServerMethods1.GetPersonJson
Result:     {"result": {
        "LastName":"Smith",
        "FirstName":"Joe",
        "Age":44}}
```

Benzer bir şekilde elimizde olan bir isim listesini (daha önce ele aldığımız StringList), özellikle istemci tarafındaki işlemde, JSON dizisini direk olarak kullanarak yapabiliriz. Şu anda sunucu metoduna bakalım.

```
function TServerMethods1.GetNamesJson: TJSONArray;
begin
        Result := TJSONArray.Create;
        Result.Add('one');
        Result.Add('two');
        Result.Add('three');
end;
```

Tekrar baktığımızda, JSON sonucu çok daha basittir. TStringList özelliği ile karşılaştırarak (daha önceki konudaki çıktıya bakın) aradaki farkı açıkça görebilirsiniz.

Executed: TServerMethods1.GetNamesJson
Result: {"result":["one","two","three"]}

Bu örnekte sunucunun döndürebildiği veri yapılarına odaklandık. İlerleyen demolarda bütün resmi görebilmeniz için aynı JSON sonuçlarını kullanacak komple bir istemci yazacağım.

--- Filtreleme ve Ön Belleğe Alma (Filtering and Caching) ---

JavaScript istemcisindeki örneklere yoğunlaşmadan önce, Delphi XE'deki DataSnap'te tanıtılan REST desteği ile alakalı olarak 2 özellikten bahsetmem gerekiyor, ama bunların derinine fazla inmeyeceğim.

İlki, sonucu (dataset veya sitring gibi) özel bir filtreleme sözdizimi kullanarak direk çağırarak (URL kullanarak) veya JavaScript proxy çağrılarında argüman dizisine gerekli değerleri sağlayarak filtrelemektir.

İki filtremiz, substring (ss) ve table'dır (t) ve ikisi de count (c), range (r) ve offset(o) olmak üzere 3 adet işletmeye sahiptir. Bu bütün elemanlar, URL adresine sunucu fonksiyonunun yolu ve fonksiyon parametrelerinden sonra (yolda belirtilen (URL'de)) parametre olarak eklenir.

Bu örnekte, 100 karaktere sahip olan bir string'den 20 karakteri çekeceğiz (sunucu daha önceki örnekle aynı) veya ilk 85 karakteri atlayacağız. Burada 2 URL adresi ve çıkan JSON sonuçları yer almaktadır.

/datasnap/rest/TServerMethods1/getlargestring/100?ss.c=20
{"result":["ooooooooxooooooooox"]}

/datasnap/rest/TServerMethods1/getlargestring/100?ss.o=85
{"result":["ooooxooooooooox"]}

Range (sıralama) filtresi, offset (başlangıç noktasına olan uzaklık) ve count (sayı) olmak üzere 2 adet parametre alır. Mesela bir tablonun sadece 3. kaydını almak için "t,r=2,1" filtresini kullanırız, 2 olmasının sebebi tablonun ilk kaydı 0 olarak görmesidir.

/datasnap/rest/TServerMethods1/GetCustomers?t.r=2,1
{"result":[{"table":[..],"CustNo":[1351],"Company":["Sight Diver Webinarist"],"Addr1":["1 Neptune Lane"],"Addr2":[null],"City":["Kato Paphos"],"State":[null],"Zip":[null],"Country":["Cyprus"],"Phone":["357-6-876708"],"FAX":["357-6-870943"],"TaxRate":[0],"Contact":["Phyllis Spooner"],"LastInvoiceDate":["1994-10-18 19:20:30.0"]}]}

Her istek için bir parça veri, HTML istemcisinden gelen büyük bir dataset içerisinde numaralandırmayı kolaylaştırır, mesela dönecek değer için her defasında 20 kayıt getir ve gelen kayıtlar için çoklu sayfalar oluştur gibi.

--- Veriyi Ön Belleğe Alma (Caching Data) ---

Bir akış (stream) veya karmaşık bir veri yapısı döndürdüğümüzde, JSON gösterimindense tüm ikili veriyi (binary data) döndürmek daha iyidir. Ama ayrıca diğer veri yapılarını döndürmek veya tek bir çağrıda farklı iki akış döndürmek için neye ihtiyacımız var?

Bu nokta, DataSnap REST ön belleğe alma (caching) desteğinin ne kadar faydalı olduğunu gördüğümüz noktadır. Her zaman çoklu veri türü döndüren (var veya out

parametrelerini kullanarak) bir metodunuz olabilir, ve veriyi JSON sonucu yerine belirlenmiş bir formatta isteyebilirsiniz.

Sunucu tarafta şöyle bir metodumuz olduğunu düşünelim. (RestXEDataTypes demosunda daha önce işlendi)

```
function TServerMethods1.ImageAvailable(const strImageName: string;
out aStream: TStream): Boolean;
begin
	Result := False;
	if strImageName = 'expand.png' then
	begin
		aStream := TFileStream.Create('images\expand.png',
fmOpenRead);
		Result := True;
	end;
end;
```

Eğer standart bir çağrıda bulunursanız, alacağınız şey akışa ek olarak verinin JSON gösterimi olacaktır. (Unutmamak gerekir ki, bir akış döndürdüğünüz zaman geleceği format ikili formattır ama opsiyonel olarak JSON formatında talep edebilirsiniz.) Bu olayda ayrıca HTTP talebinin Accept (kabul etme) başlığında (header) application/rest formatında karar kılabilirsiniz. Eğer çağrı için JavaScript kullanıyorsanız proxy kullanmak yerine çekirdek ServerFunctionExecutor nesnesine direk çağrı yapmak zorundasınız.

Burada çağrı ile ilgili JavaScript kod parçası bulunmaktadır.

```
function onReverseStringClick()
{
	var valueField = document.getElementById('valueField');
	var methExec = new ServerFunctionExecutor("TServerMethods1");

	var resultObj = methExec.executeMethod("ImageAvailable", "GET",
	[valueField.value], null, true, null, "application/rest");
	document.getElementById('image1').src =
	methExec.getCacheURL(resultObj.result[0],true);
}
```

ServerFunctionExecutor nesnesinin ExecuteMethod fonksiyonuna geçtiğimiz parametreler; metod adı, istek türü, parametreler, geri çağırma (callback), eğer sonuç varsa talep edilen filtreler ve kabul etme formatıdır. Bu çağrı yapıldığı zaman aşağıdaki JSON cevabı dönecektir.

```
{"result":["0\/0\/0",true],"cacheId":0,"cmdIndex":0}
```

Burada yukarıdaki son JavaScript satırında da gördüğümüz gibi , URL'yi kullanarak ve bir GET talebinde bulunarak cache işlemini yapabiliriz, bu URL de şunun gibi bir şey olur.

http://localhost:8080/datasnap/cache/0/0/0?sid=<...>

Bu, akışın ikili verisini, sunucu metodunu tekrar çalıştırmadan ama daha önce oluşturulmuş nesneyi (ön bellekte saklanan nesneyi) döndürecektir. Bu yüzden tekrardan belirtiyorum ki, yukarıdaki son JavaScript satırında bir sayfadaki resim

elemanının ön belleğe saklanmış akışını, URL adresine atayabiliriz. Ve böylece aşağıdaki gibi resmi görebiliriz. (Makalede ilgili resme bakın)

--- jQuery Kullanma ---

Delphi'nin DataSnap REST sunucuları, çoğunluğu sunucuyu kendi başlarına çağırmak için tahsis edilen, bir dizi JavaScript destek dosyasına sahiptir. DOM hilesi'ne ilişkin olarak (Tarayıcı içerisindeki HTML sayfalarına dinamik değişiklikler yapabilirsiniz.) sade eski JavaScripti kullanacaksınız. Bu dile dayalı olarak tarayıcı tabanlı uygulamaları geliştirmeyi kolaylaştırmak için çeşitli JavaScript klütüphaneleri bulunmaktadır. En çok bilinenlerinden ve günümüzde en sık kullanılanlarından biri (Google ve Microsoft gibi firmalar da kullanıyor) jQuery'dir. Bu gerçekten güçlü ve açık kaynak kodlu bir JavaScript kütüphanesi olup http://jquery.com adresinde bulunur.

jQuery içerisine derinleme girmek ve alternatifleri dururken neden bunu tercih ettiğimi anlatmak için yeterli zamanım yok. Ama jQuery'yi Delphi kodları ile birleştirerek, Delphi DataSnap REST Sihirbazı tarafından oluşturulan standart HTML sayfasını nasıl geliştireceğimizi kısa bir şekilde anlatacağım. (Eğer Delphi 2010 Handbook kitabımı temel alarak yazdığım geçen seneki makaleme bakarsanız, orada jQuery'yi tek başına kullandığımı, çünkü Delphi 2010'un herhangi bir JavaScript istemci desteği olmadığını göreceksiniz.)

jQuery'nin nasıl çalıştığı ile ilgili olarak bir kaç anahtar kavram bulunmaktadır. İlki, jQuery bir nesnedir ve bunu var olan bir JavaScript nesnesine veya DOM elemanına sarabiliriz (sürükle bırak gibi düşünelim) (metodları ile birlikte). Bunu yapmak için, jQuery fonksiyonunu bir nesneye direkt olarak veya bir ayırıcı (bir veya birden çok nesneyi ID, sınıf, tag bilgisine göre çekmek için kural) ile geçerek kullanmak gerekiyor.

Mesela verilen bir web sayfasında sırasıyla sayfadaki bütün hyperlink'leri gizleyebilirsiniz, verilen bir sınıftaki bütün elemanları gösterebilirsiniz ve verilen ID bilgisi'nin ait olduğu elemanın görünürlüğünü düzenleyebilirsiniz. Bunu aşağıdaki 3 satırla yapabilirsiniz.

```
$("a").hide()
$(".myclass").show()
$("#theid").toggle()
```

jQuery'deki bir diğer anahtar kavram ise JavaScripti bağlayarak HTML elemanlarına herhangi bir davranış ekleyemezsiniz. (Bu, kodun okunmasını ve bir web tarasımcısı tarafından değiştirilmesini oldukça zorlaştırır.)

--- jQuery'yi Sihirbazın HTML şablonuna (Template) Ekleme ---

Biliyorum bu çok kısa bir tanımlama oldu, ama iyi bir girişin gerektirdiği bir kaç düzine sayfayı yazamam şu anda. Şimdilik bunu yerine, Sihirbaz tarafından oluşturulan kodu nasıl alacağımıza ve jQuery'yi proje içerisine gömerek bu kodu nasıl geliştireceğimize odaklanalım (Bu RestXeOneJQuery demo projesinde gösteriliyor.)

Bütün değişikliklerimiz, ilk örnekte, ReverseString.html şablonunda (template) olacak. Öncelikle kodumuza, jQuery'yi eklemek için bir satırlık bir kod ekliyoruz. (bunu jQuery.js dosyasını projemiz içindeki js dizinine ekledikten sonra yapıyoruz.)

```
<script type="text/javascript" src="js/jquery.js"></script>
```

Alternatif olarak, jQuery'nin bulunduğu public bir adres bilgisini de referans olarak gösterebiliriz, ama bu biraz tehlikelidir çünkü kütüphanede bazı şeyler değişebilir ve bu da kodun çalışmasının durmasına sebep olabilir. Sonra onReverseStringClick fonksiyonundaki koda odaklanıyoruz. Orjinal hali şu şekildeydi.

```
function onReverseStringClick()
{
	var valueField = document.getElementById('valueField');
	var s = serverMethods().ReverseString(valueField.value);
	valueField.value = s.result;
}
```

jQuery'yi getElementById sınıfı ile yer değiştirmek ve elemanların değerine erişmek için val() fonksiyonunu kullanmak amacıyla kullanabiliriz.

```
function onReverseStringClick()
{
	var valueField = $('#valueField');
	var s = serverMethods().ReverseString(valueField.val());
	  valueField.val (s.result);
}
```

İkinci bir değişiklik de metodu HTML'e direkt olarak bağlamaktır.

```
<table>
	<tr>
		<td><input id="valueField" type="text" value="A B C" /></td>
		<td><button onclick='javascript:onReverseStringClick();'>
			ReverseString</button></td>
	</tr>
</table>
```

jQuery kullanarak onclick özelliğini kaldırıp bir belirleyici ekleyebiliriz.

```
<td><button id="reverseButton">ReverseString</button></td>
```

Artık HTML'in koda herhangi bir referansı yoktur. Bu çok önemlidir çünkü, kısıtlı uygulama kodu ve ihtiyaç bilgisi dahilinde HTML, bir web tasarımcısı tarafından gelenekselleştirmeye daha uygun hale gelir.

jQuery kullanarak yapılan yaklaşım, alttaki başlangıç fonksiyonunu kullanarak kod (HTML sayfasından ayrı bir sayfa olabilir) içerisindeki DOM elemanları ile 2 olay tetikleyicisini birleştirir.

```
$(document).ready(function() {
	$('#reverseButton').click (function (){
		onReverseStringClick();
	});
});
```

Doküman olarak çalıştırılan bu kod (HTML sayfası) hazır hale gelir ve anonim bir metodu kullanarak verilen elemanın clikc olayı için bir tetikleyici fonksiyon kurar. (onReverseStringClick olarak döner)

--- jQuery ve Veri Türleri Demosu ---

Artık Delphi DataSnap REST Sihibazı tarafından otomatik oluşturulan HTML şablonuna jQuery'yi nasıl gömdüğümüzü gördüğümüze göre bu kütüphaneyi gerçek hayatta karşılaşabileceğimiz durumlarda kullanmaya bakabiliriz. "DataSnap REST Sunucusundaki Veri Türleri" konusunda oluşturduğumuz, çeşitli metodların mevcut olduğu ServerProjectWithUserTypes demo projesi için bir istemci oluşturdum (JavaScript kodu ile bir HTML sayfası).

Bu yeni kullanıcı arayüzünü desteklemesi için, projenin web modülüne yeni bir hareket ekledim (/jclient) ve JClient.html adındaki yeni HTML şablonuna bağlanacak ilişkili bir Page Producer (sayfa üretici) bileşeni ekledim. Sayfanın ilk parçası aşağıda görünmektedir. (Hatta bazı taleplerin cevapları ile birlikte) (Makalede ilgili resme bakın)

HTML sayfası çeşitli metodlar için bir kaç sayfadan oluşsa bile, ben belirlenmiş bir yapıyı benimsedim. Her formu metoddan sonra bir div ile isimlendirerek ve form elemanlarına birleşmiş id bilgilerini referans olarak göstererek çevreledim. Örnek olarak AddInt işlemi ile ilgili formun kodu aşağıdadır.

```
<h3>function AddInt (a, b: Integer): Integer</h3>
<div id="AddInt">
	<table>
		<tr>
			<td><label>a:</label>
				<input id="a" type="text" value="10"
/></td>
			<td><label>b:</label>
				<input id="b" type="text" value="32"
/></td>
			<td><label>result: </label>
				<input id="result" type="text" value=""
/></td>
			<td><button id="button">AddInt</button></td>
		</tr>
	</table>
</div>
```

Sayfa başlangıç kodu ($(document).ready), çeşitli girdi kontrollerini referans olarak göstermek için, buton üzerinde bir olay tetikleyici kurar.

```
$('#AddInt #button').click (function (){
	res = serverMethods().AddInt(
		$('#AddInt #a').val(),
		$('#AddInt #b').val());
	$('#AddInt #result').val (res.result);
});
```

Bütün formlar ve metodlar için aynı yaklaşım kullanılacaktır, AddDouble ve GetLargeString formlarında olduğu gibi ilişkili kodlarda bunlardan bazılarına göz atabilirsiniz. IsOK fonksiyonunun formu parametre almadığından ve fonksiyondan dönen değer text'in rengini değiştirmek için kullanıldığından diğerlerine göre biraz daha farklıdır.

```
$('#IsOK #button').click (function (){
	res = serverMethods().IsOK ();
```

```
        if (res.result)
        {
                $('#IsOK #result').val ('ok');
                $('#IsOK #result').attr('style', 'color: green;');
        };
});
```

GetLargeString fonksiyonunu test etmek için, sunucuya sonuç olarak gelecek JSON'u filtrelemek için (mesela sadece ilk 20 karakteri alacak) bir kaç buton ekledim. İhtiyacımız olan, ekstra bir HTTP alanını geçmek, ServerFunctionExecuter nesnesini kullanarak düşük seviyede direkt bir çağrıda bulunmak ve HTTP alanlarını bir JSON nesnesi içerisinde 2. parametre olarak göndermektir.

```
$('#GetLargeString #button2').click (function (){
        var methExec = serverMethods().executor;
        res = methExec.executeMethod("GetLargeString", "GET", [100],
        null, true, {"ss.c": "20"}, null);
        $('#GetLargeString #result').val (res.result);
});
```

Server tarafından mashallanan bir nesneye erişmek oldukça basittir, sadece fields olarak tanımlanan alana, ekstradan bir elemanı referans olarak göndermek yeterlidir.

```
res = serverMethods().GetPerson();
var obj = res.result.fields;
```

İşler, mesela stringlist parametresi, içerisindeki stringlere erişmeye kalktığımız zaman biraz kontrolden çıkabilir.

```
var stringlist =
res.result.fields.FMoreData.fields.FSerStringItemList;
        $('#GetPersonWithData #list').val (
stringlist[0].fields.FString + ' ' +
stringlist[1].fields.FString);
```

Bu kod gerçekten çirkindir, ve karmaşık durumlarda Delphi'nin otomatik marshal yöntemini kullanmayı engellememiz için iyi bir sebeptir.

İstemci tarafından yönetilebilen ama biraz karmaşık olan bir veri yapısı, TDataSet için Delphi'den dönen bir JSON formatıdır. Gördüğümüz gibi ilk alanın adı "table" dır ve bir table header (tablo başlığı) ile dinamik olarak ayrıştırılabileceği şekilde metadatayı içerir. (alanlar ve onların ayarlarından oluşan bir dizi)

```
res = serverMethods().GetCustomers().result;
var strHtml = '<tr>';
for (var n=0; n < res.table.length; n++) {
        strHtml = strHtml + '<th>' + res.table[n][0] + '</th> ';
}
strHtml = strHtml + '</tr>'
$('#GetCustomers #result').html(strHtml);
```

JSON sonucunda her alan için değerlerin listesi ile birlikte ek bir alan daha vardır. Bu alanlar "kolona göre" olmasına rağmen, bu dizi yapısı ve "kayda göre" olan ekstra

alanları ayrıştırabiliriz. Bunu gerçekleştirmenin bir yolu, alanlardan biri için kayıt sayısını hersaplamak (mesela res.CustNo.length), kayıt sayıları üzerinde döngü kurmak (m değişkeni ile), metadata içerisinde tekrar döngü kurmak (n değişkeni ile), ana ana nesne düzeyinde eşleşen bir alan bulmaktır.

```
fieldname = res.table[n][0];
fieldvalue = res[fieldname][m];
```

Kulağa kafa karıştırıcı geldiğini biliyorum, ama bütün koda bakmanın işleri anlamayı kolaylaştıracağını pek zannetmiyorum.

```
for (var m=0; m < res.CustNo.length; m++) {
        var strHtml = '<tr>';
        for (var n=0; n < res.table.length; n++) {
                fieldname = res.table[n][0];
                strHtml = strHtml + '<td>' + res[fieldname][m] + '</td> ';
        }
        strHtml = strHtml + '</tr>'
};
```

İşleme, kolay olmaktan uzak olmasına rağmen, ki burada kolon tabanlı yapıyı kayıt tabanlı bir yapıya çevirmekten bahsediyoruz, işlenen kodla ilgili olarak güzel olan şey; bunu sayfalamayı etkinleştirecek hale adapte etmek, REST sunucusunun kurulmuş olan filtrelerine teşekkür ederek, son derece kolaydır. Yapmamız gereken şey, "t.r" HTTP parametresini geçmek, ve sunucu metod çalıştırıcıyı sayfa numarasını arttırmak için her defasında kullanmaktır. Bu örnekte, tamamen bir yerleştirme istiyorsak ki genelde bunu isteriz, bir sayfada işleyip bir sonrakine geçeriz. Aşağıdaki önceki koda göre yapılacak anahtar değişiklikler bulunmaktadır (Bütün kod için değil).

```
var page = 0;

$('#GetCustomers_paging #button').click (function (){
        var methExec = serverMethods().executor;
        res = methExec.executeMethod("GetCustomers", "GET", [],
        null, true, {"t.r": (page*3+1 + ",3")}, null);

        res = res.result[0];
        // processing as above

        page++;
        $('#GetCustomers_paging #button').html("page"+(page+1));
});
```

jQuery'de basitçe üstesinden gelebileceğimiz bir durum, resimler için dönen ikili akıştır. Burada yapmamız gereken şey, sunucuya ikili gösterimi döndürmesi için izin vermek (JSON döndürmemesi için), ve resim tag'inin src özelliğini (attribute) uygun bir URL adresine atamaktır.

```
$('#GetImage #result').attr ('src',
          '/datasnap/rest/TServerMethods1/GetImage');
```

Bahsetmek istediğim son fonksiyon, sunucu tarafında elle oluşturulan (bir TJSONArray yapısı kullanarak), stringlerden oluşan bir JSON dizisi döndüren GetNamesJson fonksiyonudur.

```
res = serverMethods().GetNamesJson();
var array = res.result;
var strarray = '';
for (var n=0; n<array.length; n++) {
        strarray = strarray + array[n] + ' ';
};
$('#GetNamesJson #result').val (strarray);
```

Şimdi Delphi REST Sunucusunun döndüreceği, karmaşık veri yapılarını işlemek için jQuery'yi nasıl kullanacağımızı biliyoruz, burada mevcut seçeneklerden herhangi birini seçebiliriz, biri üreysel veri girişine kesinlikle uygundur. Gerçektei demo projeler genelde sunucu taraflı bilgiyi yansıtıyor, ben ise bir adım ileri giderek bir nesnenin değerlerini düzenleyecek bir istemci uygulaması geliştirmek istiyorum. Düzenleme ile birlikte, aynı zamanda Delphi DataSnap REST sunucusunun, 4 anahtar HTTP metodunu destekleyerek nasıl güncelleme, oluşturma, silme taleplerini ele aldığını da öğrenmiş olacaksınız.

--- Nesneleri ve HTTP Metodlarını Yönetmek ---

Şimdi Delphi REST sunucusunun döndürdüğü veri türlerini ve TJSONValue için direkt desteğini, jQuery'nin basit bir şekilde anlatımını gördüğümüze göre, artık ben çoklu özellikleri içeren verilen bir URL tarafından desteklenen çeşitli HTTP metodlarını yönetmek için gömülü desteği olan bir proje geliştirmek istiyorum. Burada fikir, sunucudan veriyi get metodunu kullanarak çekmek değil, ayrıca JavaScript ile yazılmış basit bir tarayıcı tabanlı istemci üzerinden, sunucu taraftaki nesneleri kullanıcının düzenlemesine izin vermektir.

Bu uygulamanın arkasındaki veri yapısı, geleneksel bir türdeki nesnelerin listesidir. (Bu daha nesne yönelimli bir şekilde yazılabilir, ama uygulamanın hatrı için işleri basit tutmaya çalıştım)

```
type
        TMyData = class (TPersistent)
        public
                Name: String;
                Value: Integer;
        public
                constructor Create (const aName: string);
        end;
```

Nesneler, Delphi 2009'dan beri Generics.Collections unit dosyasında tanımlanan, TObjectDictionary<TKey, TValue> üreysel taşıyıcı sınıfı tarafından üretilen bir sözlükte saklanır. Bu global nesne, program başladığı zaman bir kaç önceden tanımlanmış nesne ile birlikte başlatılır. Unutmamak gerekir ki, nesneleri eklemek, nesne adının sözlük anahtarı ile senkronize olup olmadığından emin olmak ve eğer herhangi biri sağlanmamışsa onlara rastgele bir değer atamak için belirlediğim bir AddToDictionary prosedürü kullanıyorum.

```
var
```

```
        DataDict: TObjectDictionary <string,TMyData>;

    procedure AddToDictionary (const aName: string; nVal: Integer = -1);
    var
        md: TMyData;
    begin
        md := TMyData.Create (aName);
        if nVal <> -1 then
        md.Value := nVal;
        DataDict.Add(aName, md);
    end;

    initialization
        DataDict := TObjectDictionary <string,TMyData>.Create;
        AddToDictionary('Sample');
```

Bu veri yapısına sahip olduktan sonra, erişmek ve değiştirmek için metodları yazabiliriz. Veriyi okumak için yazdığım MyData fonksiyonuna bakarsak.

```
    function TObjectsRest.MyData(name: string): TJSONObject;
    var
        md: TMyData;
    begin
        md := DataDict[name];
        Result := TJSONObject.Create;
        Result.AddPair(
        TJSONPair.Create ('Name', md.Name));
        Result.AddPair(
        TJSONPair.Create ('Value',
        TJSONNumber.Create(md.Value)));
    end;
```

Gördüğünüz gibi, bir TJSONObject oluşturdum ve isim ve değer alanına iki çift veya özellik ekledim.

```
    {"result":[{
        "Name":"Sample",
        "Value":"7016"
    }]}
```

Eğer aşağıdaki URL adresini tarayıcınıza yazıp çalıştırırsanız yukarıdaki sonucu elde edersiniz.

http://localhost:8080/datasnap/rest/TObjectsRest/MyData/Sample

Yukarıdaki URL, Delphi'nin REST kodlamasının neden parametreleri yolda geçmeniz gerektiğine açıklık getirmektedir. Bir REST URL'si kaynağa referans gösterdiğinde, ve her kaynak için bir metod yazmak istemiyorsanız, nesne ID bilgisini (burada Sample) yolda bulundurmanı, ayrıca parametre olarak geçmeniz gerekmektedir.

--- TJSONArray ile Nesneleri Listeleme ---

Bir nesne listesine sahipseniz, bunun elemanlarına erişme ihtiyacınız olur. Kullanıcı arayüzünü oluşturmak için ihtiyacımız olan şey, listedeki nesnelerin isimleridir. Liste döndürmek için TJSONArray kullanmalıyız, bu olayda sözlükteki Keys alanları için bir enum kullanmak için oluşturduğum bir string dizisidir.

```
function TObjectsRest.List: TJSONArray;
var
	str: string;
begin
	Result := TJSONArray.Create;
	for str in DataDict.Keys do
	begin
		Result.Add(str);
	end;
end;
```

Bu çağrının sonucu, result adındaki bir diziye parametre olarak gönderdiğimiz JSON formatında bir dizidir. (Bundan dolayı ikili olarak içiçe geçen bir gösterim vardır)

```
{"result":[
	["Test","Sample"]
]}
```

Şimdi, değer listelerini döndürerek ve her başlıbaşına her elemanı getirebildiğimizden, bir kullanıcı arayüzü oluşturmaya başlayabiliriz.

--- İstemciyle Başlamak - Liste ve Değerler ---

Kullanıcının bir tanesini seçmesi için nesnelerin listeleri ile ilk HTML'i oluşturmak yerine, tamamen AJAX modelini kullanabiliriz. Başlangıçtaki HTML sayfası, HTML elemanları ve JavaScript kodları dışında gerçek veriye sahip olmayacaktır. Bu, boş HTML'dir.

```
<div>List of objects:
	<br>
	<div id="list" style="padding-left: 20px"></div>
	<br>
	<a href="#" id="refresh">Refresh List</a>
</div>
<hr>
```

Kullanıcının araya girme durumu olmadan sayfa yüklendiğinde, sayfa sunucuya gerçek veriyi (refreshList() fonksiyonunda) sormak için bir AJAX çağrısında bulunacak ve kullanıcı arayüzünü yerleşim bölgesi haline getirecektir (populate). Aynı fonksiyon refresh anchor'u içerisindeki click olayında bir olay tetikleyicisi olarak kurulur.

```
$(document).ready(function() {
	// populate the UI
	refreshList();

	// redo the operation when refresh is clicked
```

```
	$("#refresh").click(function(e) {
		refreshList();
});
```

refreshList() fonksiyonu sonuç olarak dönen diziyi taramak için bir döngü kullanır. Burada jQuery'nin $.each numaralandırma (enumeration) mekanizmasını kullanabilirdim ama bu, kodu okumak için fazla karmaşık hale getirirdi. Bu for döngüsü, verilen ID ile zaman yer tutucusunda (span placeholder) ilerde gösterilecek olan, HTML'i oluşturur.

```
	function refreshList()
	{
		thearray = serverMethods().List().result;
		var theMarkup = ["<br>"];
		for (var i=0; i < thearray.length; i++) {
			theMarkup = theMarkup +   "<a href='#'>" +
			thearray[i] + "</a><br>";
		};
	};
```

JSON dizisinde dönen değerlerin listesinin bulunduğu örnek çıktı aşağıdadır (Makalede ilgili resme bakın)

Burada biraz daha kod geliştirmemiz gerekiyor. Liste için HTML'imiz olduğunda, bu linklerin koduna, seçilen her giriş için sayfanın alt tarafında yer alan ilişkili sunucu taraflı nesneyi istemci uygulamanın yükleyebilmesi için, bağlanmamız gerekir. Bu, div altındaki bütün anchorlara (<a> tagları) tek bir olayla bağlanan standart bir jQuery stili ile becerilebilir. Bütün kodu ile birlikte refreshList() fonksiyonunun son hali aşağıdaki gibidir.

```
	function refreshList()
	{
		thearray = serverMethods().List().result;
		var theMarkup = ["<br>"];
		for (var i=0; i < thearray.length; i++) {
			theMarkup = theMarkup +   "<a href='#'>" +
			thearray[i] + "</a><br>";
		}
		// add markup to container
		$("#list").html(theMarkup);
		$("#list").find("a").click(function(e) {
			theObj = serverMethods().MyData ($(this).html());
			TMyDataToForm (theObj.result);
		});
	};
```

Unutmamak gerekir ki yukarıdaki kodda $(this) deyiminin kullanımı, o anki nesne veya olay tetikleyen nesneye referans olur. (Az çok VCL olay tetikleyicisindeki, Sender parametresine benzer şekilde davranır, her ne kadar dil terminolojisinde this is deyimi self deyimine daha çok benzese de)

Kullanıcı, nesne listesindeki bir linke tıkladığı zaman, JavaScript uygulaması HTML metni içinden nesnenin adını alır ve MyData sunucu fonksiyonunu çağırmak için bir parametre olarak kullanır. Nesne verisi için geliştirilen kullanıcı arayüzü, ilerde nesne

verisini işletmek için de kullanılacak olan 2 adet input box'tan oluşur. Veri, bu 2 inputa yardımcı fonksiyon olan TMyDataToForm kullanılarak taşınır.

```
function TMyDataToForm (anObj)
{
	$("#TMyData #Name").val(anObj.Name);
	$("#TMyData #Value").val(anObj.Value);
};
```

Bu kod, kullanıcı arayüzünün sınıf adı ile aynı bir div'e ve JSON nesnelerinin özelliklerine benzer bir şekilde çağrılan 2 input elemanı olduğu zannıyla çalışır. Aşağıdaki kod, bir altındaki resmi tarayıcı içerisinde sayfanın tamamında gösteren bir ilişkili HTML kod kesimidir. (Makalede ilgili resme bakın)

```
<div id="TMyData">
	<p>Current Element:
	<br/>
	<input type="text" id="Name" size="50" value="">
	<br/>
	<input type="text" id="Value" size="50" value="">
</div>
```

Yukarıda da gördüğümüz gibi, program bir veriyi almamıza izin verir, ayrıca en sık kullanılan işlemler için 3 adet buton barındırır (CRUD arayüzü - Create,Read,Update,Delete : Oluştur,Oku,Güncelle,Sil). Bu HTML'de PUT,GET,POST ve DELETE adı verilen 4 çekirdek HTML metodu ile desteklenir. Bir sonraki konuda Delphi REST Sunucusunun bu metodları nasıl desteklediğini işleyeceğiz.

--- Oku,Güncelle,Yaz,Sil (Get,Post,Put,Delete) ---

Şimdi REST sunucusundan sadece veriyi nasıl alacağımızı gördük, ama güncelleme nasıl olacak? REST'teki genel fikir birliği, işlemleri tanımlamak için özel URL kullanmayı engellemek, ama sunucu taraflı nesneleri tanımlamak için özel bir URL kullanmak (daha önce listelediğim MyData/Sample gibi) ve ne yapılacağını belirtmek için HTTP metodlarını kullanmaktır.

Eğer Delphi'nin REST desteği basitçe URL'leri sunucu sınıf metoduna eşleştirmeyi destekliyorsa, şanssızız demektir. (Aynı metod 4 işlemi de kullanabilir) Delphi, bunun yerine, URL ve HTTP metodunu sunucu metodu ile eşleştirir. Kullanılan şema oldukça basittir, yapılacak işlemin adıi metod adının önüne eklenir.Şu haritalamayı kullanır

GET	get (varsayılandır, ihmal edilebilir)
POST	update (güncelleme)
PUT	accept (kabul etme)
DELETE	cancel (iptal etme)

Bu eşleştirmeleri, DSHTTPWebDispatcher bileşeninin ilişkili 4 olay tetikleyicisini ele alarak değiştirebilirsiniz. Eğer çeşitli işlemler için standart isimlendirme kurallarını kullanmak isterseniz, sunucu sınıfı aşağıdaki gibi tanımlamanız gerekir.

```
type
```

```
TObjectsRest = class(TPersistent)
        public
                function List: TJSONArray;
                function MyData (name: string): TJSONObject;
                procedure updateMyData (const aName: string;
                jObject: TJSONObject);
                procedure cancelMyData (name: string);
                procedure acceptMyData (const aName: string;
                jObject: TJSONObject);
end;
```

Bir elemanı almak veya silmek istersek, sadece nesnenin ID bilgisi veya ismine ihtiyacımız olacaktır. Oluşturma veya güncelleme için, ayrıca veriyi de geçmemiz gerekir. Bu çok basit durumda, sadece ikinci bir veri olur, ama genelde bundan çok daha fazla veri girilecektir. İlginç bir çözüm ise (tamamen DataSnap tarafından desteklenen) ikinci parametreyi, kullanıcı arayüzündeki bütün alanları içerecek bir nesne olarak göndermektir.

Bu 3 yeni metodun yazılışı daha kolay ve direkttir çünkü herhangi bir değer döndürmemize gerek yoktur. (Bütün parametrelerin dolu olup olmadığını ve sunucu tarafındaki nesnenin mevcut olup olmadığını kontrol ettiğimi söylemem gerek yok herhalde)

```
procedure TObjectsRest.updateMyData (const aName: string;
jObject: TJSONObject);
var
        md: TMyData;
begin
        md := DataDict[aName];
        md.Value := StrToIntDef (
        jObject.Get('Value').JsonValue.Value, 0);
end;

procedure TObjectsRest.cancelMyData(const name: string);
begin
        DataDict.Remove(name);
end;

procedure TObjectsRest.acceptMyData(const aName: string;
jObject: TJSONObject);
var
        md: TMyData;
begin
        md := TMyData.Create (aName);
        md.Value := StrToIntDef (
        jObject.Get('Value').JsonValue.Value, 0);
        DataDict.Add(aName, md);
end;
```

--- İstemci Tarafta Düzeltme ---

Artık sunucu tarafında CRUD işlemlerine sahip olduğumuza göre, şimdi JavaScript istemci uygulamamızı 3 adet düzenleme butonunu kaynak kodları ile ekleyerek bitirebiliriz. (Tarayıcı tabanlı kullanıcı arayüzü resmiyle birlikte daha önce gösterildi)

Delphi tarafından üretilen JavaScript proxy'si sunucu taraftaki eşdeğerlerine benzer isimlerde metodlar içerecektir. (Ne olursa olsun aynı URL ve farklı HTTP metodudur bunlar). Diğer önemli bir parça bilgi ise Delphi'ye bir JSON nesnesi geçmek için, sadece yapmamız gereken bir JavaScript nesnesi geçmektir. 3 buton ve kaynak kodu şu şekildedir.

```
$("#buttonUpdate").click(function(e) {
        var mydata = new Object;
        FormToTMyData (mydata);
        serverMethods().updateMyData (mydata.Name, mydata);
});

$("#buttonDelete").click(function(e) {
        serverMethods().cancelMyData ($("#TMyData #Name").val());
        refreshList();
});

$("#buttonNew").click(function(e) {
        var mydata = new Object;
        FormToTMyData (mydata);
        serverMethods().acceptMyData (mydata.Name, mydata);
        refreshList();
});
```

Veriyi kullanıcı arayüzüne taşımaya benzer olarak, istemci program bunu geri almak için bir destek fonksiyonu kullanır.

```
function FormToTMyData (anObj){   anObj.Name = $("#TMyData
#Name").val();
anObj.Value = $("#TMyData #Value").val();};
```

Sınıfımıza eklenen 3 metod ve uygun JavaScript çağrıları ile birlikte REST sunucusunda nesne oluşturmak ve düzenlemek için tamamen tarayıcı tabanlı bir kullanıcı arayüzü örneğine sahip olmuş oluyoruz. Bu projede Delphi ve JavaScript'te kolayca işlenebilecek bir format olan JSON nesnelerini hem ileri hem de geri olarak parametre olarak gönderdik.

--- İşlemi RTTI ile Otomatikleştirmek ---

Şimdi çok alana sahip daha karmaşık bir veri yapısı olduğunu farz edelim. Delphi sınıflarından JSON oluşturup bunu sunucu uygulama içerisinde kaydetmek, ve tarayıcıda JavaScript nesnelerini HTML kullanıcı arayüzü elemanları ile eşleştirmek vs. gerekir.

Pek çok durumda bu tür bir eşleştirme basitleştirilebilir, ama tamamen otomatikleştirilemez. Mesela bu eşleştirmeyi yapmak için Delphi'nin RTTI'sini kullanabiliriz. (Yeni genişletilmiş RTTI, Delphi 2010'dan beri mevcuttur) Verilen bir Delphi nesnesinin public özelliklerinin JSON gösterimini aşağıdaki kodla oluşturabiliriz (Örneğe eklediğim RestPlusUtils unit dosyasının bir parçasıdır)

```
function ObjectToJson (obj: TObject): TJSOnObject;
var
	context: TRttiContext;
	aProperty: TRttiProperty;
begin
	Result := TJSONObject.Create;
	for aProperty in context.GetType(obj.ClassInfo).GetProperties do
		Result.AddPair(TJSONPair.Create (
			aProperty.Name,
			aProperty.GetValue(obj).ToString));
end;
```

Gerçek hayatta, daha özel veri türlerini (Boolean ve Date gibi) ele almak zorunda kalırız ama koddaki ana fikir gayet açıktır.

Tersi olarak, sunucu bir JSON nesnesi aldığında, bu JSON verisini Delphi datası olarak kaydedebilir. (Bu zaman, en az bası veri türlerini hesaba katmak gerekir) Unutmamak gerekir ki, her özellik için uygun bir JSON değeri aramalıyız, ama kaybolmuşsa çok da sorun değil.

```
procedure JsonToObject (obj: TObject; jObject: TJSOnObject);
var
	context: TRttiContext;
	aProperty: TRttiProperty;
	jPair: TJsonPair;
begin
	for aProperty in context.GetType(obj.ClassInfo).GetProperties do
	begin
		jPair := jObject.Get (aProperty.Name);
		if Assigned (jPair) then
		begin
			if aProperty.PropertyType.IsOrdinal then
					aProperty.SetValue(obj, StrToIntDef (
				jPair.JsonValue.Value, 0)) // not always string
			else
					aProperty.SetValue(obj,
jPair.JsonValue.Value); // string
		end;
	end;
end;
```

Artık JSON ile eşleştirilen Delphi nesnelerini döndürebilir ve alabiliriz.

```
function TObjectsRest.MyData(const aName: string): TJSONObject;
var
	md: TMyData;
begin
	md := DataDict[aName];
	Result := ObjectToJson (md);
end;

procedure TObjectsRest.updateMyData (const aName: string;
jObject: TJSONObject);
```

```
var
	md: TMyData;
begin
	md := DataDict[aName];
	JsonToObject (md, jObject);
end;
```

Be şekilde yazıldığında, sunucu verilen bir nesnenin JSON versiyonunu döndürebilir ve JSON verisini herhangi bir Delphi nesnesini güncellemek için kabul edebilir. Ayrıca program, JavaScript nesnesini (JSON tarafından temin edilen) HTML kullanıcı arayüzüne eşleştirmenize yardımcı olabilir. Div'i sınıf adı ve özellik isimleri ile birlikte birbirinden ayrı girdilerde kullanmak şartıyla, Delphi uygulamasının içinden eşleşmiş JavaScript kodunu oluşturmanız mümkündür.

Daha önce gösterilen (TMyDataToForm ve FormToTMyData) 2 JavaScript eşleştirme metodu tamamen oluşturuldu. GenerateMapper adındaki sunucudaki fonksiyon, RTTI'yi temel alır ve RestPlusUtils unit dosyasının bir parçasıdır. Onun kodundan bir parça aşağıdadır.

```
sList.Add ('function ' + aClass.classname + 'ToForm (anObj)');
sList.Add ('{');
for aProperty in context.GetType(aClass.ClassInfo).GetProperties do
begin
	// outputs JS like: $("#TMyData #Name").val(anObj.Name);
	sList.Add ('    $("#' + aClass.classname + ' #' + aProperty.Name +
	'").val(anObj.' + aProperty.Name + ');');
end;
sList.Add ('};');
```

Bu JavaScript kodu sunucuda nasıl mevcut olabilir? WebFileDispatcher1BeforeDispatch olay tetikleyicisini düzenledim (varsayılan olarak serverfunctions.js dosyasını yeniden oluşturmakla ilgilenen metod) ve aşağıdaki durumu ekledim.

```
if SameFileName(ExtractFileName(AFileName), 'uimapper.js') then
begin
	Handled := True;
	Response.ContentType := 'text/javascript';
	Response.Content := GenerateMapper (TMyData);
end;
```

JavaScript dosyasını döndürmeden önce güncellemek yerine, sihirbaz tarafından oluşturulan kodun serverfunctions.js için yaptığı gibi, ben de JavaScript içeriğini dinamik olarak oluşturulan bir string olarak geri döndürüyorum. HTML içerisindeki bu JavaScript kaynağının URL bilgisine basitçe bir referans ekliyorum ve eğer sunucu tarafındaki nesneye yeni özellikler eklersek kodun çalıştığını ve otomatik olarak kendini güncellediğini görebiliyoruz. (Tabi HTML kullanıcı arayüzünü de buna göre güncellemek şartıyla)

Daha önce sadece iki alanı olan kullandığım sınıf, ve elle eşleştirme her şekilde basittir. jObjects örnek projesinde, bununla beraber çeşitli özellikleri olan bir ikinci sınıf daha vardır.

```
type
TCompanyData = class
```

```
        private
                ...
        public
                property CompanyName: string
                        read FCompanyName write SetCompanyName;
                 property Address: string
                        read FAddress write SetAddress;
                property City: string
                        read FCity write SetCity;
                property ZipCode: string
                        read FZipCode write SetZipCode;
                property State: string
                        read FState write SetState;
                property Country: string
                        read FCountry write SetCountry;
                property ContactFullName: string
                        read FContactFullName write SetContactFullName;
                property ContactEmail: string
                        read FContactEmail write SetContactEmail;
                property ContactPhone: string
                        read FContactPhone write SetContactPhone;
end;
```

Tekrardan belirtmek gerekirse, bu 4 HTTP metodunu bu tarz bir nesneyi yönetmek için gösteren bir sınıftır. (Bu olayda liste yoktur ama burda kodlamanın detayı için gerçek kaynak koda referans gösteren bir tek bellekte nesne vardır)

```
{$METHODINFO ON}
type
TCompanyServer = class (TPersistent)
        public
                function Company: TJSONObject;
                procedure updateCompany (jObject: TJSONObject);
                procedure cancelCompany;
                procedure acceptCompany (jObject: TJSONObject);
end;
```

Web modülüne TCompanyServer ile bağlanan yeni bir DataSnap sunucu sınıfı ekledim. Yeni bir hareket, yeni bir sayfa üreticisine bağlandı ve sınıf desteği de oluşturulan uimapper.js dosyasında bulunmaktadır.

Artık kullanıcı arayüzünü oluşturmam için yapmam gereken şey, bir önceki HTML dosyasını klonlamak ve veri girişi formunu uygun veri girişi alanlarını ekleyerek değiştirmektir. Yeni HTML şablonunda (company.html), butonların 4 işlemi gerçekleştireceği kemiğine kadar sıyrılmış basit bir veri girişi formu oluşturdum. Bu durumda gerçekten tembelsiniz (ya da uygulamanın başlangıç noktasındasınız), sunucu zaten kemiğine kadar sıyrılmış basit bir form oluşturabilir, ve kullanıcı arayüzü oluşturabilir. Bu, diğer web geliştirme çevrelerince de kullanılan ve iskele adı verilen bir tekniktir.

Çok başlangıç demosu olarak, RestPlusUtils unit dosyasına aşağıdaki özel tag kullanarak etkinleştirilecek bir HtmlFormForClass fonksiyonu ekledim.

```
<!-- automatic form -->
<#scaffolding class="CompanyData.TCompanyData">
```

Bu tag sunucu tarafından üretilir, sınıfın nitelendirilmiş adını (unit ve sınıf adı) tutan özellik değerini çıkarır. Ve bu tag yeni RTTI tarafından bir sınıfı kayıt olma gereksinimlerine gerek kalmadan bulmak için kullanılır.

```
// in TWebModule2.ServerFunctionInvokerHTMLTag
else if   SameText(TagString, 'scaffolding') then
begin
        ReplaceText := HtmlFormForClass (TagParams.Values['class']);
end;
```

HTML dosyasında o anki versiyonun biraz daha güzel formatta yazılmış hali olmasına rağmen, iskele tekniği kullanarak oluşturulan formun görünümü aşağıdaki gibidir. (Makalede ilgili resme bakın)

Sormadan önce, evet, bu çekirdek mimarinin genişletilmiş hali ile çalıştım. Bu genişletilmiş halinde, daha çok veri türü için belirlenmiş destek, genişletilmiş tarayıcı tabanlı doğrulama ve diğer pek çok güzellik var. Bununla ilgili daha çok bilgi edinmekte özgürsünüz.

--- Veri Yönelimli REST Sunucuları ---

Eğer DataSnap'in arkasındaki orjinal fikir, veri tablolarını çok katmanlı bir sunucudan istemci uygulamaya taşımaksa, DataSnap'deki REST desteğinin ilk olarak odaklandığı noktanın uzaktan metod çağırma olması biraz gariptir. REST sunucusundan bir dataset döndürebilirsiniz, ama kolon tabanlı format belirlenmiş bir kullanıcı arayüzü desteği olmadığı gerçeği ile birleşir, ve bu da dataset yönetim desteğini sınırlı bir hale getirir. Burada size, bu formatın tepesine basit bir HTML tablosu oluşturmayı ve sayfalama desteğine bağlanmayı göstereceğim.

Gördüğümüz gibi, bununla beraber, Delphi'nin REST mimarisinden geleneksel bir JSON veri türü döndürmek kolaydır, bu yüzden kullanıcı arayüzünü geliştirmenin yanında, sunucu tarafındaki işlemleri gelenekselleştirmek işleri daha basit hale getirir. Ayrıca geleneksel bir HTML kullanıcı arayüzü geliştirmektense, var olan data grid bileşenlerini avantajlarını kullanmak gerekir. Bu da son konuda ve örnekte odaklanacağımız konudur.

Sunucu program basittir, yaptığı tek şey bir DataSet'in bütün datasını, metadatası olsun olmasın, farklı JSON formatlarında döndürmektir. Ben her kayıttaki nesne için bütün dataset'i ve alanların listesini JSON dizisi olarak döndüren ayrı 2 adet fonksiyon yazdım.

```
function TServerData.Meta: TJSONArray;
var
        jRecord: TJSONObject;
        I: Integer;
begin
        ClientDataSet1.Open;
        Result := TJSonArray.Create;

        for I := 0 to ClientDataSet1.FieldDefs.Count - 1 do
                Result.Add(ClientDataSet1.FieldDefs[I].Name);
end;

function TServerData.Data: TJSONArray;
```

```
var
        jRecord: TJSONObject;
        I: Integer;
begin
        ClientDataSet1.Open;
        Result := TJSonArray.Create;
        while not ClientDataSet1.EOF do
        begin
                jRecord := TJSONObject.Create;
                for I := 0 to ClientDataSet1.FieldCount - 1 do
                        jRecord.AddPair(
                                ClientDataSet1.Fields[I].FieldName,
                                TJSONString.Create
(ClientDataSet1.Fields[I].AsString));
                Result.AddElement(jRecord);
                ClientDataSet1.Next;
        end;
end;
```

Verinin miktarını azaltmak için, tekrar eden alan isimlerini engelleyebilirim, her kayıt için bir dizi döndürebilirim, veya null değeri olan ya da boş olan alanları atlayabilirim. Bu iki metod istemci uygulama tarafından dinamik olarak HTML tablosu oluşturan jRestClient.html sayfası yüklendikten sonra yürütülür. HTML dosyasında, sadece tablo yapısı bulunur.

```
<table id="result" border="1" cellspacing="0">
        <thead id="head"></thead>
        <tbody id="body"></tbody>
</table>
```

İstemci uygulama önce metadatayı yükler ve tablonun başlangıç satırını oluşturacak diziyi işletir.

```
var theMetaArray = serverMethods().Meta().result;
var headMarkup = "<tr>"; // table header
for (var j=0; j < theMetaArray.length; j++) {
        headMarkup = headMarkup +   "<th>" + theMetaArray[j] + "</th>";
};
headMarkup = headMarkup +   "</tr>";
$("#head").html(headMarkup);
```

İkinci adım gerçek veriyi almak, ve metadata dizisini yerleştirmek için kullanmaktır. Bu olayda JavaScript kodu nesne özelliklerine notasyon nesnesi olan [“propertyname”] ile dinamik olarak erişir. Bu demektir ki, özellik sembolünü kullanmak (thearray[i].Company gibi) yerine kodun metadata dizisindeki (theMetaArray[j]) özellik adını okuyarak kullanması, şu deyimle sonuçlanır.

```
thearray[i][theMetaArray[j]])
```

Tablonun vücüdunu oluşturacak JavaScript kodunun tamamı aşağıdadır.

```
thearray = serverMethods().Data().result;
var bodyMarkup = ''; // table content
```

```
for (var i=0; i < thearray.length; i++) {
        bodyMarkup = bodyMarkup + "<tr>";
        for (var j=0; j < theMetaArray.length; j++) {
                bodyMarkup = bodyMarkup + "<td>" +
                thearray[i][theMetaArray[j]] + "</td>";
        };
        bodyMarkup = bodyMarkup + "</tr>";
};
$("#body").html(bodyMarkup);
```

Bu sayfanın çıktısı aşağıdaki ekran görüntüsü gibi olur. (Makalede ilgili resme bakın)

--- DataTables Bileşenini Kullanmak ---

Tekrardan söylüyorum ki, bu hizmetler başlangıç noktasıdır. Bir HTML tablosunu elle oluşturup ona basit özellikler eklemektense bir jQuery plug-in'inin avanlatjlarını kullanarak bir HTML tablosuna kayda değer özellikleri ekleyerek, bunu filtrelemei sıralama ve diğer yetenekleri ile birlikte güçlü bir grid haline getirmeyi tercih ederim.

Buna örnek olarak www.datatables.net adresinden temin edilebilecek, mevcut bir jQuery plug-in'i olan DataTables verilebilir. (Bu plug-ini özellikle seçtim, çünkü bununla biraz tecrübem var, ama bu kütüphanedeki plug-in gibi, jQuery için güçlü diğer grid bileşenleri de bulunmaktadır)

Veriyi DataTables bileşenine aktarmak için pek çok yol vardır, ama en basiti, veriyi içerisinde data (aaData, bir dizi dizisi) ve metadata (aoColumns, kolon başlıklarını ve format stilini içeren bir nesne dizisi) olan bir nesne içerecek şekilde bir JSON yapısı olarak en başta oluşturmaktır. Bu nedenle, sunucu taraftati sınıfa belirlenmiş JSON formatında döndürme için belirlenmiş bir metod ekledim.

```
function TServerData.DataTables: TJSONObject;
var
        dataArray, subArray: TJSONArray;
        metaArray: TJSONArray;
        I: Integer;
begin
        ClientDataSet1.Open;
        dataArray:= TJSONArray.Create;

        while not ClientDataSet1.EOF do
        begin
                subArray := TJSONArray.Create;
                for I := 0 to ClientDataSet1.FieldCount - 1 do
                        subArray.Add(ClientDataSet1.Fields[I].AsString);
                dataArray.AddElement(subArray);
                ClientDataSet1.Next;
        end;

        metaArray := TJSonArray.Create;
        for I := 0 to ClientDataSet1.FieldDefs.Count - 1 do
        begin
                metaArray.Add (TJSONObject.Create(
```

```
                TJSONPair.Create('sTitle', ClientDataSet1.FieldDefs[I].Name)));
        end;

        Result := TJSONObject.Create;
        Result.AddPair(TJSONPair.Create('aaData', dataArray));
          Result.AddPair(TJSONPair.Create('aoColumns', metaArray));
end;
```

Bu bileşenle (JavaScript kaynak kodunu da dahil ederek) ve mevcut olan bu JSON veri yapısı ile istemci tarafın kodu çok büyük derecede basit hale gelmektedir. Tablosu olan bir forma (basitlik için başlıkları düzenlenmiş bir bir tane oluşturdum) ve JSON datasını HTML tablosuna geçmek için dataTable metodunu uygulamak için başlangıçta aşağıdaki 2 satırlık kodu eklemek yeterli olacaktır.

```
var theDataTablesData = serverMethods().DataTables().result;
$('#example').dataTable(theDataTablesData);
```

Sonuç aşağıdaki resimdeki gibi görülür. (Makalede ilgili resme bakın) Bu data grid içinde, görünen satırların sayısını seçebilirsiniz (bu olayda sayfalama istemci tarafından yapılmakta, ama bileşeni sunucu tarafında sayfalama yapmak için ayarlayabilirsiniz.), kolonun başlığına tıklayarak içeriğini sırayalabilirsiniz ve görünen satırları bir search box ile filtreleyebilirsiniz. (Filtreleme olayı, bütün kolonlara varsayılan olarak uygulanır)