=== Programlama ve Tasarım ===

--- DataSnap XE in Action ---

--- İş İçin DataSnap ---

Geçen sene RAD, Delphi 2010 ve üstü için DataSnap ile ilgili olarak bir döküman yayınladı. Bu döküman çok katmanlı uygulamalar ve DataSnap'in ana kuralları için iyi bir başlangıç temsil ediyordu. Bununla beraber pek çok önemli gelişim ve değişim olmuştur. Esas dökümanı tekrar yazmak yerine DataSnap'i pratik olarak kullanmayı öğretmeyi, bunu yaparken de var olan ve özellikle yeni ve fonksiyonel olan noktaları vurgulamayı doğru bulduk.

Yeni, DataSnap fonksiyonelliği, bu döküman içerisinde aşağıda belirtilen noktalarda anlatılmaya çalışıldı.

DataSnap Sunucu Sihirbazları
ISAPI DLL'leri için HTTPS Desteği
Kriptolama Filtreleri (HTTPS kullanmadığınız zaman)
Geliştirilmiş TAuthenticationManager Bileşeni
Sunucu metodları için rol-tabanlı Yetkilendirme özellikleri
TDataSetProvider için rol-tabanlı Yetkilendirme
TDSSessionManager
HTTPS/SSL üzerinde dağıtım (IIS içinde)

Ayrıca DataSnap'in daha önceki versiyonunda hali hazırda var olan özelliklerin de üzerinden geçeceğim, ama bunlar bir önceki dökümanımda anlattığım özellikler olmayacak. Burada bahsedeceğim özellikler de aşağıda belirtilmiştir.

Dağıtılacak Sunucu Modül Metodlarını nasıl gizleriz?
Anahtar alanları otomatik olarak artırma
İstemciden sunucuya parametre değerleri göndermek
Sunucu tarafında parametre değerleri atamak

Bu dökümanda, bir kaç saat içinde hazırlanan ve aynı gün içerisinde hizmete sunulan bir DataSnap uygulamasından bahsedeceğim.

Uygulama sorun monitörleme ve izleme ile ilgili olan, Gelişim Olay Raporlama Aracı (Development Issue Report Tool - DIRT (türkçe kirli demek)) olarak adlandırılan bir uygulamadır.Bu sistem hali hazırda zaten bazı müşterilerim tarafından kullanılmakta ve programı kullanırken kendi aralarında hadi bazı kirlileri (DIRT)temizleyelim diye birbirlerine şakalar yapmaktadır.

--- Veri Modeli ---

Uygulamayı geliştirmeye başlamadan önce sorunları tanımlamak ve geliştiricilerin bu sorunlar üzerinde çalışarak rapor çıkarmasını sağlayacak bir veri modeli geliştirmemiz gerekmektedir. Bu veri modelini geliştirmek, son kullanıcılarla yapılan bir beyin fırtınası ile mümkün olmakta, ve çıkan sonuç aşağıda gösterilmektedir.

--- Kullanıcı Tablosu ---

Kullanıcı tablosu, uygulamaya erişmek isteyecek diğer kullanıcılar ve testçi olarak atanacak geliştirici kullanıcıları içeren bir tablodur. Gerekli olan bilgiler; bir kullanıcı adı, bir hashlenmiş şifre ve bir adet e-posta adresidir. Adres ve telefon numarası gibi diğer alanlar ihtiyaç duyulduğunda eklenebilir. Bizim örneğimizde kullanıcının, "geliştirici", "testçi", "yönetici" gibi bir rolü olacaktır. Sistem kullanıcısı bu rolleri özel bir "admin" rolü ile birlikte kullanabilecektir. Bu teknik bir kullanıcı için birden çok rol gerektiği zaman virgüllerle yanına eklenerek genişletilebilir.

```
CREATE TABLE [dbo].[User](
	[UserID] [int] IDENTITY(1,1) NOT NULL,
	[Name] [nvarchar](50) NOT NULL,
	[PasswordHASH] [nvarchar](50) NOT NULL,
	[Role] [nvarchar](32) NOT NULL,
	[Email] [nvarchar](50) NOT NULL
)
```

Kullanıcının sadece HASHlenmiş şifresini saklamalı, gerçek şifresini herhangi bir yerde saklamamalıyız. Bu demektir ki uygulamanın istemci ve sunucu taraflarında güvenli bir bağlantı için sadece hashlenmiş şifreler gönderilecektir. Hash bilgisinin kısaca nasıl oluşturulduğunu ve güvenli bir bağlantının kurulduğundan nasıl emin olacağımızı ilerde göreceğiz.

--- Rapor Tablosu ---

Rapor tablosu en detaylı datasetimizdir. Burada Projeyi, Versiyonu (opsiyonel), Modülü (opsiyonel), Önceliğini, Sorunun Türünü anlatacak bir rapor oluşturabilmeliyiz. (Twit edilmeye hazır olacak şekilde 140 karakterlik bir kısa özeti de bulunmalı) Sorunun Durum bilgisi (raporlandı, atandı, açıldı, çözüldü, test edildi, yayınlandı, kapandı) ile ilgili olarak özel bir alan tutulmalıdır. Ayrıca; rapor tarihi, rapor eden şahıs, soruna atanan şahıs (sonra atanacak olan şahıs opsiyoneldir), başlangıçtan sorunu kimin çözeceğine bağlı olarak önemli bilgilerdir.

```
CREATE TABLE [dbo].[Report](
	[ReportID] [int] IDENTITY(1,1) NOT NULL,
	[Project] [nvarchar](50) NOT NULL,
	[Version] [nvarchar](20) NULL,
	[Module] [nvarchar](50) NULL,
	[IssueType] [int] NOT NULL,
	[Priority] [int] NOT NULL,
	[Status] [int] NOT NULL,
	[ReportDate] [datetime] NOT NULL,
	[ReporterID] [int] NOT NULL,
	[AssignedTo] [int] NULL,
	[Summary] [nvarchar](140) NOT NULL,
	[Report] [nvarchar](4000) NOT NULL
)
```

Unutmamak gerekir ki diyagramda da belirtildiği gibi ReporterID ve AssignedTo alanları Kullanıcı tablosuna atanmış birer yabancı anahtardır (foreign key). Ayrıca

unutmamak gerekir ki IssueType (Sorun Türü), Priority (Öncelik), Status (Durum) alanları da tamsayı (integer) alanlardır. Mesela 1den 10a kadar olan bu değerler daha önce bir yerlerde tanımlanmış olmalıdır. Bunlarla ilgili olarak tanımlamaların string türünde yapıldığı tabloları ekledik ama GUI (grafik arayüz) tarafında bunları göstererek rpogramın içerisinde tamsayı değerleri kullandık.

Burada üzerinde anlaştığımız bir kural var ki Öncelik ve Sorun Türü alanlarındaki değerler arttıkça sorunun da önemi artar. Yani Öncelik Değeri ve Sorun Türü Değeri 10 olan bir sorun, Öncelik Değeri 1 ve Sorun Türü Değeri 1 olan bir sorundan daha önemlidir.

Ayrıca Durum alanı ile ilgili olarak da ek bir tablo oluşturduk ama GUI'nin kendi kendi halletmesi için izin verdik aynı şekilde (Sorun Türü ve Öncelik ile ilgili özel tablolar dökümanda var)

--- Yorum Tablosu ---

Yorum tablosu, sorun ile ilgili olarak neler yapıldığını, ne raporlandığını vs. bilgileri içeren bir log dosyasına benzer bir biçimde iş akışı veya log kitabı gibi bilgileri içeren bir tablodur. Bu demektir ki her sorun raporu için, kullanıcının yerleştireceği bir yoruma sahip olabiliriz. (Esas işlemi görmek amacıyla belli bir tarihte)

```
CREATE TABLE [dbo].[Comment](
	[CommentID] [int] IDENTITY(1,1) NOT NULL,
	[ReportID] [int] NOT NULL,
	[UserID] [int] NOT NULL,
	[CommentDate] [datetime] NOT NULL,
	[Comment] [nvarchar](4000) NOT NULL,
)
```

Bu 3 tablo DIRT projesi ile ilgili olarak DataSnap Sunucu uygulamasını geliştirebilmemiz için birbirlerine farklı ReportID, ReporterID, AssignedTo ve UserID alanları ile bağlanmıştır. Kesinlikle daha çok tablo ekleyebiliriz. Ama burada amacımız, mümkün olduğunca basit bir şekilde optimum fonksiyonelliği gerçekleştirebilmektir.

--- DataSnap Sunucusu ---

Delphi XE Enterprise veya Architect Object Repository içerisinde DataSnap Sunucusu oluşturabilmemiz için en az 3 adet Sihirbaz sunar.

DIRT uygulaması için, bize dünyanın her yanından güvenli ve doğru bir şekilde erişilebilecek bir DataSnap sunucu türüne ihtiyacımız bulunmaktadır. (Bazı sorun raporları, herkese göstermek istemeyeceğimiz bazı dahili detaylara sahip olabilir.) Farklı mobil cihazlar ve kablosuz bağlantıların erişimine açık olması için HTTPS (HTTP Secure - Güvenli HTTP) protokolünün kullanımı seçilir. SSL/TLS sertifikası da sunucunun güvenli bir şekilde tanımlanması ve kriptolu iletişim için kullanılır. HTTPS, HTTP'den daha güvenlidir, zaten bu yüzden bu protokole HTTPS denmektedir. Ayrıca, TCP/IP'nin, HTTP ve HTTPS'den daha hızlı olduğu ispatlanmıştır ama buradaki hız durumu bizi ilgilendirmemektedir bu proje için. Eğer daha fazla hız istiyorsanız, TCP/IP protokolünü kullanmalısınız.

HTTPS seçeneği, sihirbazların oluşturacağı 2 adet DataSnap Sunucu hedefinin olduğunu gösteriyor. DataSnap WebBroker tarafından oluşturulacak bir ISAPI DLL'i veya DataSnap REST Uygulama sihirbazı. İkisi arasındaki fark, REST uygulaması için oluşturulan ek dosyalardır. (JavaScript, template, şekil sayfaları ve resimler) Bu ek dosyalara şu anda ihityacımız olmadığı için seçimimiz DataSnap WebBroker sihirbazı

vasıtasıyla oluşturulacak bir ISAPI DLL'idir. Bu ISAPI DLL'i, daha önce de belirttiğimiz gibi IIS üzerinden dağıtılacaktır.

DataSnap WebBroker uygulama sihirbazının 2. sayfasında eğer Doğrulama ve Yetkilendirme istiyorsak ve eğer bazı örnek metodları ile bir Sunucu Metodu sınıfı istiyorsak bunları belirtebiliriz. Doğrulama, bir kullanıcının gerçekten bahsettiği kişi olup olmadığını belirlemektir (bir kullanıcı adı ve şifre kombinasyonuyla) ve aynı zamanda hangi fonksiyonelliğin kullanıcı rolüne tanımlı veya yasaklı olduğunu da belirtmektir.

Bizim durumumuzda, biz hem Doğrulama (Authentication) hem de Yetkilendirme (Authırization) ayrıca örnek metodlara ihtiyaç olmayacak şekilde bir Sunucu Metodu Sınıfı istiyoruz. Biz kendimi basit bir şekilde kendi metodlarımızı ekleyebiliriz.

WebBroker Uygulaması sihirbazının son adımında Sunucu Metodu Sınıfının ata sınıfını (özelliklerini kalıtacağı sınıf) belirleyebiliyoruz. Eğer aynı dataset (TDataSetProviders) ve sunucu metodlarını paylaşmak istiyorsak bu bölümde TDSServerModule'i seçmeliyiz. TComponent ata sınıfı seçeneğini seçersek göstermek için sunucu metodlarını tanımlayabilir, ve ek olarak görsel olmayan bileşenler için TDataModule kullanabiliriz, ama TDSServerModule ata sınıfı arka tarafta TDSProviderDataModuleAdapter sınıfını kullanarak IAppsServerFastClass interface'inin metodlarını da göstermektedir.

Sihirbazda Bitir butonuna tıkladığımız zaman yeni bir proje 2 adet kaynak dosya ile birlikte oluşturulmuş olur. Bir ServerMethods unit'i ve bir web modülü ile birlikte. Projeyi DirtServer olarak (ya da kendi DIRT projenize ne isim vermek istiyorsanız o isimle) sunucu metodları dosyasını DirtServerMethods.pas ve web modülünü de DirtWebMod.pas olarak kaydetmeliyiz.

Unutmamak gerekir ki sunucu metodu unit'ini yeniden isimlendirdiğimiz zaman, proje derlenemez. Web modül ünitindeki uses bölümünde ServerMethods1 yazan yeri DirtServerMethods olarak değiştirerek kaydetmemiz gerekmektedir.

Aynı web modülü içinde OnGetClass olay tetikleyicisi içerisinde PersistentClass özelliğine gerekli olan atamayı yapmamız gerekiyor ki bunla ilgili kod aşağıdadır.

```
procedure TWebModule1.DSServerClass1GetClass(
DSServerClass: TDSServerClass;
var PersistentClass: TPersistentClass);
begin
        PersistentClass := DirtServerMethods.TServerMethods1;
end;
```

Bu iki düzeltme ile bir ISAPI DLL'i oluşturacak DirtServer DataSnap sunucu projesini derlememiz mümkün olabilir. Bu demektir ki doğrulama kontrollerinden başlayarak, artık gerçek fonksiyonelliği eklemeye başlayabiliriz.

--- Bağlantı ve Doğrulama ---

Daha önceden modifiye ettiğimiz web modülü aynı zamanda DataSnap sunucusu için doğrulama ve yetkilendirme işlemlerini de yapacağımız yerdir. Bu görev için elimizdeki anahtar yeni TDSAuthenticationManager bileşenidir.

Bu bileşende bizim Kullanıcıyı (User), Şifreyi (Password, Bu DIRT veritabanı içinde Kullanıcı tablosunda bulunmaktadır) ve aynı zamanda erişim için gerekli olan protokolü (burada HTTP olmayıp HTTPS olduğundan emin olmalıyız) belirleyebileceğimiz bir OnUserAuthenticate olay tetikleyicisi bulunmaktadır. En başta

ilgili olay tetikleyicisinin kodu iletişim protokolünün HTTPS olacağından emin olacağımız bir şekilde şu şekilde yazılabilir.

```
procedure TWebModule1.DSAuthenticationManager1UserAuthenticate(
Sender: TObject; const Protocol, Context, User, Password: string;
var valid: Boolean; UserRoles: TStrings);
begin
        valid := LowerCase(Protocol) = 'https';
        // now also validate User and Hashed password…
end;
```

Bir sonraki adım Kullanıcı ve Şifrenin (esas şifrenin hashlenmiş değerinin) doğrulanacağı, bunun için de bu iki bilginin Kullanıcı tablosunda karşılığı olan bir kombinasyonunu sorgulama işlemi yapılır, kısım olacaktır.

Bunun için temel amacı Kullanıcı tablosuna direkt bir uygulama yapmak olan lokal bir TSQLConnection bileşeni kullanabiliriz. Alternatif olarak kullanıcı adları ve hashlenmiş şifreleri sunucu makinada bir konfigürasyon dosyasında veya passwd.ini dosyasında tutarak kullanıcı bağlanmayı denediğinde bu dosyayı okuyabilir.

Eğer veritabanına direk bir bağlantı istiyorsak, web modülü üzerine bir TSQLConnection bileşeni ekler, Driver (Sürücü) özelliğini MSSQL olarak atar, ve veritabanına geçerli bir bağlantı kurmak için HostName, Database, UserName ve Password alanlarına gerekli değerleri atarız. Login Prompt özelliğine False özelliğini atadığımızdan emin olmalı, ve Connected özelliğine True değerini atayarak başarılı bir bağlantı yapıp yapmadığımızı kontrol etmeliyiz. Veritabanı ile konuşmak, iletişim kurmak için TSQLConnection bileşeninin dışında Kullanıcı tablosundaki Kullanıcı Adı ve Şifre alanına sorgu yapmak için bir de TSQLDataSet bileşeni kullanmalıyız.

--- Lokal Veritabanı Bağlantısı ---

Eğer web modülü üzerine TSQLConnection ve TSQLDataSet bileşenlerini eklemek istemiyorsak, bunu dinamik olarak da oluşturabiliriz. Özellikle sadece TSQLConnection bileşenine ihtiyacımız olduğu durumlarda (bağlantıyı doğrulamak) bu temiz bir çözümdür. Özellikle SQLConnection parametre ayarları bir .ini veya konfigürasyon dosyasında okunabilir. (Okuyucu için bir egzersiz olur.)

Her iki durumda da gerçekleştirmemiz gereken sorgu cümlesinde Kullanıcı tablosundan UserID ve Role alanlarına SELECT uygulamasını WHERE kalıbı içerisinde Kullanıcı ve hashlenmiş Şifre bilgilerini parametre olarak geçerek kullanacağız. Unutmamak gerekir ki şifre bilgisini hashlememize gerek yoktur, zaten hashlenmiş bir şekilde gelmektedir. (İstemcinin yapmasına ihtiyaç duyacağı bir şey)

Eğer SELECT sorgu cümleciği bir kayıt döndürürse UserID değerini, o anki thread için TDSSessionManager.GetThreadSession kullanarak ve PutData'yı çağırıp UserID adındaki bir alana UserId alanını atayarak, o anki oturumda saklayabiliriz. SELECT sorgu cümlesinden dönen Role alanını UserRoles (kullanıcı rolleri) kolleksiyonuna eklemek için kullanabiliriz.

```
procedure TWebModule1.DSAuthenticationManager1UserAuthenticate(
Sender: TObject; const Protocol, Context, User, Password: string;
var valid: Boolean; UserRoles: TStrings);
var
        SQLConnection: TSQLConnection;
        SQLQuery: TSQLQuery;
```

```
        Role: String;
    begin
        valid := LowerCase(Protocol) = 'https';
        if valid then
        begin // validate User and Hashed password
            SQLConnection := TSQLConnection.Create(nil);
            try
                SQLConnection.LoginPrompt := False;
                SQLConnection.DriverName := 'MSSQL';
                SQLConnection.Params.Clear;
                // The following parameters can be read from a config file
                SQLConnection.Params.Add('HostName=.');
                SQLConnection.Params.Add('Database=DIRT');
                SQLConnection.Params.Add('User_Name=sa');
                SQLConnection.Params.Add('Password=********');
                SQLQuery := TSQLQuery.Create(nil);
                SQLQuery.SQLConnection := SQLConnection;
                SQLQuery.CommandText :=
                    'SELECT UserID, Role FROM [User] WHERE ' +
                    ' (Name = :UserName) AND (PasswordHASH
= :Password)';
                SQLQuery.ParamByName('UserName').AsString := User;
                SQLQuery.ParamByName('Password').AsString :=
Password;
                try
                        SQLQuery.Open;
                        if not SQLQuery.Eof then
                        begin
                        valid := True;

    TDSSessionManager.GetThreadSession.PutData('UserID',
                        SQLQuery.Fields[0].AsString);
                        Role := SQLQuery.Fields[1].AsString;
                        if Role <> '' then
                            UserRoles.Add(Role)
                        end
                        else
                        valid := False
                finally
                        SQLQuery.Close;
                        SQLQuery.Free
                end
            finally
                SQLConnection.Connected := False;
                SQLConnection.Free
            end
        end
    end;
```

Unutmamak gerekir ki yukarıdaki kod parçası bir kullanıcı için sadece bir role izin vermektedir. Bunu data modele ek tablolar ekleyerek genişletmek mümkün. (Rolleri UserRoles kolleksiyonu olarak tanımlayarak) veya Role alanındaki değerleri virgülle ayırarak birden çok rol eklemek de mümkün. Bu iki durumu okuyucuya egzersiz

olarak bırakıyorum (Bunlar DataSnap XE Development Essentials adındaki dökümanımda ele alınmıştır)

--- Yetkilendirme ---

Yetkilendirme olayını ele almamız gerekir. DataSnap kullanıcı rollerine sunucu sınıfları ve sunucu metodları ile birlikte bağlanma imkanı tanımaktadır. Yetkilendirme ile başa çıkmak için DSAuthenticationManager bileşeninin OnUserAuthenticate olay tetikleyicisi kullanılabilir. Basitçe yetkilendirmeyi ele almanın iyimser ve kötümser yaklaşım olmak üzere 2 farklı yolu vardır. İyimse yaklaşım, açıkça yasaklanmadığı taktirde herhangi bir işleme izin verilmesi, kötümser yaklaşım da açıkça izin verilmediği taktirde herhangi bir işleme izin verilmemesi anlamına gelmektedir. Kullanılacak en iyi yöntem, sizin geliştireceğiniz uygulamanın türüne bağlıdır. Bu durumda (bu uygulama için) güvenlik önemli olduğundan biz kötümser yaklaşımı seçeceğiz. (Yetkisi olmayan şahısların sorun raporlarının veya yorumların detaylarını görmesini veya düzeltmesini istemiyoruz)

Kısaca, OnUserAuthorize olay tetikleyicisinde varsayılan değer olan geçerli bilgisine True değerini atamak yerine False değerini atarız. Sonra Yetkilendirilmiş Roller (Authorized Roles) koleksiyonuna bakarak bizim kullanıcı rolümüzün bu koleksiyon içerisinde bulunup bulunmadığını kontrol ederiz. İlgili kod aşağıdadır.

```
if Assigned(AuthorizeEventObject.AuthorizedRoles) then
        for UserRole in AuthorizeEventObject.UserRoles do
                if AuthorizeEventObject.AuthorizedRoles.IndexOf(UserRole) >=
0
                        then valid := True;
```

Bununla beraber istemci taraftaki DataSnap proxy sınıflarının üretimi gibi herhangi bir Yetkilendirimiş Rol'ün açık olarak atanmadığı işlemler de bulunmaktadır. Bu fonksiyonellik, eğer yukarıdaki kodu kullanırsak erişilemez olur. Bu sebeple alternatif bir yaklaşım uygulamalıyız. Eğer herhangi bir Yetkilendirilmiş Rol açıkça belirtilmemişse, işleme kullanıcı rolü açık bir şekilde Reddedilmiş Roller (Denied Roles) listesinde bulunmuyorsa izin verilebilir. (Uygulamada, genelde Yetkilendirilmiş Roller boşsa, Reddedilmiş Roller de boş olmaktadır.)

OnUserAuthorize olay tetikleyicisinin yukarıdaki duruma göre güncellenmiş hali aşağıda bulunmaktadır.

```
procedure TWebModule1.DSAuthenticationManager1UserAuthorize(Sender:
TObject;
AuthorizeEventObject: TDSAuthorizeEventObject; var valid: Boolean);
var
        UserRole: String;
begin
        if Assigned(AuthorizeEventObject.AuthorizedRoles) then
                valid := False // assume NOT OK, unless explicitly allowed   !!
        else valid := True; // no Authorized Roles?

        if not valid then
                if Assigned(AuthorizeEventObject.AuthorizedRoles) then
                        for UserRole in AuthorizeEventObject.UserRoles do
                                if
AuthorizeEventObject.AuthorizedRoles.IndexOf(UserRole) >= 0
```

```
                                                then valid := True;

        if valid then
                if Assigned(AuthorizeEventObject.DeniedRoles) then
                        for UserRole in AuthorizeEventObject.UserRoles do
                                                        if
AuthorizeEventObject.DeniedRoles.IndexOf(UserRole) >= 0
                                                then valid := False;
    end;
```

Unutmamak gerekir ki, yukarıdaki kod aynı zamanda bir kullanıcı rolünün hem Yetkilendirilmiş Rollerde hem de Reddedilmiş Rollerde bulunması durumunu açık bir şekilde kontrol etmektedir. (Kullanıcıların bazı sunucu metodları için reddedileceği durumlar olabilir.)

Yetkilendirmelerin role bağlı olarak atanması, sunucu metodları seviyesinde yapılması gereken bir işlemdir. Bir admin'in yeni kullanıcı ekleyebildiğinden, bir yöneticinin salt okunut bir yöntemle bütün sorunları görebileceğinden, bir testçinin yeni sorunlar rapor edebileceğinden ve hem testçinin hem de geliştiricinin sorunla ilgili olarak yorum ekleyebileceğinden emin olmamız gerekir. (Sorunları tekrar çözmek için çalıştıklarından)

--- İstemci İçin Sunucu Metodları ---

DirtServerMethods unit dosyasına giderek, TDSServerModule sınıfından türetilen TServerMethods1 sınıfını sunucu metodlarını tanımlamak ve yazmak için genişletebiliriz. Buraya ekleyeceğim bir kaç metod, istemci uygulamayı, doğru seçenek setlerine ve işlemlere izin verilip verilmeyeceği konusunda desteklemek içindir. Kullanıcı rolünü baz alarak, mesela bir kullanıcı bütün sorunları görebilir, veya belirli sorunları düzenleyebilecektir (Kullanıcıya bağlı olarak bir yolla veya öbürüyle)

Bu amaç için GetCurrentUserID ve GetCurrentUserRoles adında 2 adet sunucu metodu geliştirdim.

```
    public
        { Public declarations }
        function GetCurrentUserID: Integer;
        function GetCurrentUserRoles: String;
```

GetCurrentUserID metodu, yeni bir sorun raporu eklemek için (veya herhangi bir soruna yorum yazmak için) bizim UserID (kullanıcı ID bilgisi) lazım olacağı için yararlı olacaktır. GetCurrentUserRoles ise o anki kullanıcının sahip olduğu rolleri döndürdüğünden, ve istemci tarafta bu fonksiyonelliği açabileceğimizden bizim için daha yararlı bir metoddur.

GetCurrentUserRoles metodunun kodunda TDSSerssionManager.GetThreadSession tarafından çağrılan o anki oturumun, UserRoles.Text değerinin döndürüldüğü bir satır bulunmaktadır sadece. Kod aşağıdadır.

```
    function TServerMethods1.GetCurrentUserRoles: String;
    begin
        Result := TDSSessionManager.GetThreadSession.UserRoles.Text;
    end;
```

Unutmamak gerekir ki tek bir string içerisinde birden çok rol bulunursa UserRoles.Text alanı 0 değerini döndürür, eğer birden çok rol varsa CRLF (satır sonu) çiftleriyle ayrılmış olmalıdır.

GetCurrentUserID metodunun kodu ilkine göre biraz daha karmaşıktır. Aynı oturumu kullanarak GetData metodunu çağırarak 'UserID' alanının değerini geri dönürebiliriz. (Kullanıcının başarılı bir bağlantı yaptığını hesaba katıyoruz) Eğer UserID alanı bulunmuyor veya geçerli olamyan bir değeri varsa, değer olarak -1 değerini döndürürüz.

```
function TServerMethods1.GetCurrentUserID: Integer;
begin
        Result := StrToIntDef(
        TDSSessionManager.GetThreadSession.GetData('UserID'), -1);
end;
```

Şu anda UserID değerini kullanmayacağız ama bu bizi ilginç bir başlığa götürecek. Kullanıcıların listesini elde etmek ve/veya yeni kullanıcılar eklemek.

--- Kullanıcı Adlarını Almak İçin Sunucu Metodu ---

Yeni bir sorun ekleneceği ve özellikle bu sorunu bir kullanıcıya atama durumu sözkonusu olduğu zaman, istemci uygulamanın kullanıcı adlarının listesini (ve UserID değerlerini) gösterebiliyor olması çok faydalı olacaktır. Bu amaç için TServerMethods1 sınıfına kullanıcı adları ve UserID değerlerinden oluşan bir dataset döndüren GetUserNames adında 3. bir sunucu metodu ekleyeceğim.

Bu sunucu metodu sadece Tester (testçi) ve Developer (geliştirici) rolüne sahip olan kullanıcılar tarafından çağrılabilecek, çünkü sadece bu kullanıcılar yeni sorun rapor edip var olan sorunlarla ilgili olarak yorum yazma yetkisine sahiptirler. Sonuç olarak bu metodun tanımlaması yetkilendirilmiş rolleri açık bir şekilde listeleyen TRoleAuth adında bir özellikle dekore edilerek aşağıdaki gibi yazılır.

```
[TRoleAuth('Tester,Developer')]
function GetUserNames: TDataSet;
```

TRoleAuth özelliğine hangi sunucu metodunun açık bir şekilde yasaklandığı rolleri belirten ikinci bir argüman ekleyebiliriz, Bununla beraber kötümser yaklaşımı benimsediğimizden böyle bir şeye gerek yoktur. Ve TRoleAuth özelliğinde sadece bu sunucu metodunu uygulamaya açık bir şekilde izin verilen rolleri kullanıcıları ile birlikte tanımlayacağız.

Bu sunucu metodunun kodlaması için sadece TSQLConnection bileşenine değil, aynı zamanda sqlUser adında bir TSQLDataSet bileşenine de ihtiyacımız var. TSQLDataSet bileşeni, TSQLConnection bileşenine bağlanacak kendi SQLConnection özelliğine sahiptir ve bu bileşenin CommandType özelliğini de Dbx.SQL olarak atamalıyız. CommandText özelliği Kullanıcı tablosundan kullanıcı adı ve UserID alanlarını çekecek aşağıdaki SELECT cümlesine sahip olmalıdır.

```
SELECT UserID, Name FROM [User]
```

Bu belirtilen SELECT sorgu cümleciği ile birlikte GetUserNames sunucu metodunun kodlamasına başlayabiliriz. Bu aşamada sqlUser tablosunu açmak ve ilk kayda gitmek (bu durumda zaten açıktır ama ilk kayda yönlenmemiştir) işlemlerini yapmak gerekir.

```
function TServerMethods1.GetUserNames: TDataSet;
begin
	try
		sqlUser.Open;
		sqlUser.First;
		Result := sqlUser
	except
		on E: Exception do
		begin
			Result := nil;
			sqlUser.Close;
			SQLConnection1.Close
		end
	end;
end;
```

Unutmamak gerekir ki biz TSQLDataSet bileşenini açtığımız için TSQLConnection bileşenini açmamıza gerek bulunmamaktadır. Metod bitmeden önce sqlUser'in kapanıp kapanmadığından emin olmamıza gerek yoktur çünkü açılan TSQLDataSet bu sunucu metodun sonucuna metod olarak geçilmelidir.

Var olan kullanıcıların listesini çıkarmak güzeldir, ama başlangıçta kullanıcı olmayacaktır. Özelleşmiş bir şifre ve rolle veritabanına yeni bir kullanıcı ekleyecek sunucu metodunu yazmanın zamanı geldi artık.

--- Yeni Kullanıcı Eklemek İçin Sunucu Metodu ---

Sadece Admin rolündeki bir kullanıcı açık bir şekilde yeni bir kullanıcı ekleyebilir. TRoleAuth özelliğini sadece Admin rolündeki kullanıcının metodu çağırmasına izin verilecek şekilde tekrar kullanımı ile 4. sunucu metodumuz olan AddUser metodunu TServerMethods1 sınıfına ekleriz.

```
[TRoleAuth('Admin')]
procedure AddUser(const User,Password,Role,Email: String);
```

AddUser metodu için gerekli parametreler, yeni kullanıcı adı, şifre (kriptolanmamış hali, AddUser sunucu metodu tarafından otomatik olarak hashlenecektir, bununla beraber bu durumu değiştirip hashlenmiş halini direk parametre olarak gönderebilirsiniz), rol ve e-posta adresidir.

AddUser metodu DIRT veritabanına bağlanacak bir TSQLConnection bileşenine ihtiyaç duyacaktır. Bu bileşen Sunucu Metodları unit dosyasında bir kere bulunduğu için AddUser sunucu metodunun kodu aşağıdaki gibi olur.

```
procedure TServerMethods1.AddUser(const User, Password, Role, Email:
String);
var
	SQLQuery: TSQLQuery;
begin
	SQLQuery := TSQLQuery.Create(nil);
	SQLQuery.SQLConnection := SQLConnection1;
	SQLQuery.CommandText := 'INSERT INTO [User] ' +
	' (Name, PasswordHASH, Role, Email) ' +
```

```
		' VALUES (:Name, :PasswordHASH, :Role, :Email)';
		SQLQuery.ParamByName('Name').AsString := User;
		SQLQuery.ParamByName('PasswordHASH').AsString :=
HashMD5(Password);
		SQLQuery.ParamByName('Role').AsString := Role;
		SQLQuery.ParamByName('Email').AsString := Email;
		SQLConnection1.Open;
		try
			SQLQuery.ExecSQL;
		finally
			SQLConnection1.Close;
		end;
	end;
```

HashMD5 metodu TServerMethods1 sınıfı içerisinde tanımlanan private bir metoddur ve kodlaması Indy IdHashMessageDigest unit'ini TIdHashMessageDigest5 türü ile birlikte kullanarak şu şekilde yazılabilir.

```
	function TServerMethods1.HashMD5(const Str: String): String;
	var
		MD5: TIdHashMessageDigest5;
	begin
		MD5 := TIdHashMessageDigest5.Create;
		try
			Result := LowerCase(MD5.HashStringAsHex(Str,
TEncoding.UTF8))
		finally
			MD5.Free
		end
	end;
```

Unutmamak gerekir ki eğer HashMD5 metodunu istemci tarafta gösterilecek bir sunucu metod olarak tanımlamak istemiyorsak, mutlaka private veya protected olarak tanımlamamız gerekmektedir.

En sonunda unutmamak gerekir ki diğer kullanıcıları ekleyebilmek için öncelike Admin rolünde bir kullanıcı eklemek gerekir. Ve bu Admin rolündeki kullanıcıyı elle eklemek gerekir. Bu yapılmadığı zaman en başta bağlanıp kullanıcı ekleyememekle ilgili olarak bir yumurta ve tavuk problemi ile karşılaşırız. (Yumurta mı tavuktan, tavuk mu yumurtadan)

--- Bütün Sorunları Almak ıçin Sunucu Metodu ---

Bütün bu idareden sonra, artık biraz gerçek fonksiyonel işler yapmanın zamanı geldi. Bu bölümde, raporlanan sorunların bir listesinin gösterilmesini işleyeceğiz. Durumdaki bir filtre ile, sonuçları ile birlikte MinStatus ve MaxStatus değişkenleri arasında kalan durumları bir dataset olarak geri çekebilir, ve bunu istemci tarafında herhangi bir grid bileşeni veya diğer data ile ilgili yerlere bağlayabiliriz.

5. sunucu metodumuz GetIssues adında bir fonksiyondur ve sadece rolü Manager (Yönetici) olan kullanıcılar tarafından erişilebilmelidir. İlgili kod aşağıdaki gibidir.

```
	[TRoleAuth('Manager')]
```

```
// Return all issues (read-only) between MinStatus..MaxStatus
function GetIssues(MinStatus,MaxStatus: Integer): TDataSet;
```

Bu sunucu metodunun kodlanması için TSQLConnection dışında sqlReports adında bir de TSQLDataSet bileşenine de ihtiyacımız bulunmaktadır.

TSQLDataSet bileşeninin kendi SQLConnection özelliği, TSQLConnection değerini alır, CommandType özelliğini de Dbx.SQL olarak atarız. CommandText özelliği ise Rapor tablosundan Where deyimi ile Maksimum ve Minimum değerler arasındaki bütün kayıtları getirecek Select sorgu cümleciği ile atanır. Örnek cümlecik aşağıdaki gibidir.

```
SELECT "ReportID", "Project", "Version", "Module", "IssueType",
"Priority", "Status", "ReportDate", "ReporterID", "AssignedTo",
"Summary", "Report"
FROM "Report"
WHERE Status >= :MinStatus AND Status <= :MaxStatus
ORDER BY Status
```

Bu basit sorgu cümleciğinin kullanımındaki dezavantaj, istemci tarafta bazı alanların çevrilebileceğindendir. IssueType (Sorun Türü), Priority (Öncelik), Status (Durum) gibi alanlarda herhangi büyük bir problem olmasa da ReporterID ve AssignedTo alanlarında UserID değerlerinden elimizde bir kullanıcı listesi olabilir. Bu durumda GetUserNames metodunu çağırarak kim olduğunu öğrenebiliriz ama bunu da sadece Testçi ve Geliştirici rolündeki bir kullanıcı çağırabilir.

Sorgu cümleciği her ne kadar bize salt okunur bir liste verecek olsa da biz bunu Kullanıcı tablosu ile birleştirerek ve ReporterID ve AssignedTo alanlarına Kullanıcı tablosundan ilgili kullanıcının adını yerleştirerek şu şekilde düzenleyebiliriz.

```
SELECT "ReportID", "Project", "Version", "Module", "IssueType",
"Priority", "Status", "ReportDate",
UReporterID.Name AS "Reporter", UAssignedTO.Name AS Assigned,
"Summary", "Report"
FROM "Report"
LEFT OUTER JOIN [User] UReporterID ON UReporterID.UserID = ReporterID
LEFT OUTER JOIN [User] UAssignedTO ON UAssignedTO.UserID = AssignedTO
WHERE Status >= :MinStatus AND Status <= :MaxStatus
ORDER BY Status
```

Unutmamak gerekir ki LEFT OUTER JOIN deyimini kullandım çünkü, Kullanıcı tablosunda AssignedTo gibi bir alana gereksinim duyulmamaktadır. (Bu NULL değer olabilir, yani bir sorun herhangi bir kullanıcıya atanmamış olabilir. Burada normal bir LEFT JOIN deyimi herhangi bir sonuç döndürmez.)

Bu TSQLDataSet ile tamsayı olan MinStatus ve MaxStatus değişkenlerini, sorgu cümlesi içerisindeki ilgili alanlara parametre olarak göndererek GetIssues sunucu metodunu aşağıdaki gibi yazabiliriz.

```
function TServerMethods1.GetIssues(MinStatus,MaxStatus: Integer):
TDataSet;
begin
        try
                SQLConnection1.Open;
                sqlReports.Close;
                sqlReports.ParamByName('MinStatus').Value := MinStatus;
```

```
                sqlReports.ParamByName('MaxStatus').Value := MaxStatus;
                sqlReports.Open;
                Result := sqlReports
        except
                on E: Exception do
                begin
                        Result := nil;
                        sqlReports.Close;
                        SQLConnection1.Close
                end
        end;
end;
```

Unutmamak gerekir ki, sqlReports nesnesini kapatmamalıyız ama TSQLDataSet'in fonksiyon işlendiği sürece açık kalmasına izin vermemiz gerekir.

--- Yeni Sorun Raporlama İçin Sunucu Metodu ---

Manager (Yönetici) rolündeki bir kullanıcı, bütün raporlanmış sorunların salt okunur bir kuşbakışını görebilmeli ve Tester (Testçi) rolündeki bir kullanıcı da yeni sorunlar ekleyebilmelidir. Bunun için alttaki tanıtımda belirtilen 6. sunucu metodumuz olan ReportNewIssue metoduna ihtiyacımız vardır.

```
[TRoleAuth('Tester'}]
function ReportNewIssue(Project, Version, Module: String;
IssueType, Priority: Integer; // Status = 1
Summary, Report: String;
AssignedTo: Integer = -1): Boolean;
```

Unutmamak gerekir ki, Rapor tablosunun ReportID (kimlik gibi, otomatik artan değer), ReporterID ve Status (Durum) alanları haricinde bütün alanlarını metoda parametre olarak gönderdik. ReportedID alanı, sunucuda loglanan UserID alanından otomatik olarak alınabilir ve benzer şekilde Status alanı da otomatik olarak "reported" "raporlandı" anlamına gelen 1 değerini alabilir. Bu değer raporlanan sorunların daha henüz bir kullanıcıya atanmadığı anlamına gelmektedir. AssignedTo alanındaki değere göre yazılacak sorgu cümlelerinde değişmeler olabilir. (AssignedTo alanı ve parametresi ile birlikte yazılabilir veya yazılmayabilir)

ReportNewIssue metodunun kaynak kodu aşağıdaki gibi olur.

```
function TServerMethods1.ReportNewIssue(Project, Version, Module: String;
IssueType, Priority: Integer; // Status = 1
Summary, Report: String;
AssignedTo: Integer = -1): Boolean;
var
        InsertSQL: TSQLQuery;
begin
        Result := False;
        InsertSQL := TSQLQuery.Create(nil);
        try
                InsertSQL.SQLConnection := SQLConnection1;
```

```
                    if AssignedTo >= 0 then
                        InsertSQL.CommandText :=   'INSERT INTO [Report]
([Project], ' +
                                                '
[Version],[Module],[IssueType],[Priority],[Status], ' +
                                                '
[ReportDate],[ReporterID],[AssignedTo],[Summary],[Report]) ' +
                                                '   VALUES (:Project,:Version,:Module,:
IssueType,:Priority, '   +
                                                '   1,
@Date, :ReporterID,:AssignedTo,:Summary,:Report) '
                    else   // No AssignedTo
                        InsertSQL.CommandText :=   'INSERT INTO [Report]
([Project], ' +
                                                '
[Version],[Module],[IssueType],[Priority],[Status], ' +
                                                '
[ReportDate],[ReporterID],[Summary],[Report]) ' +
                                                ' VALUES
(:Project,:Version,:Module,:IssueType,:Priority, '   +
                                                '   1, @Date, :ReporterID,:Summary,:Report) ';

                    try
                        InsertSQL.ParamByName('Project').AsString := Project;
                        InsertSQL.ParamByName('Version').AsString := Version;
                        InsertSQL.ParamByName('Module').AsString := Module;
                        InsertSQL.ParamByName('IssueType').AsInteger :=
issueType;
                        InsertSQL.ParamByName('Priority').AsInteger :=
Priority;
                        InsertSQL.ParamByName('ReporterID').AsInteger :=

        StrToIntDef(TDSSessionManager.GetThreadSession.GetData('UserID'),0);
                        if AssignedTo >= 0 then

        InsertSQL.ParamByName('AssignedTo').AsInteger := AssignedTo;
                        InsertSQL.ParamByName('Summary').AsString :=
Summary;
                        InsertSQL.ParamByName('Report').AsString := Report;
                        Result := InsertSQL.ExecSQL = 1
                    except
                        on E: Exception do
                                CodeSite.SendException(E)
                    end
                finally
                    InsertSQL.Free;
                    SQLConnection1.Close
                end;
        end;
```

Bu fonksiyon yeni bir kayıt doğru bir şekilde eklendiği zaman True değeri döndürecektir. Unutmamak gerekir ki burada hata yakalama şimdilik sadece istisnai

durumu CodeSite'ye gönderme olarak yapılmıştır. İstisnai durumu tekrar yükseltmek veya kendi hata tutma kontrolünüzü yazmak konusunda serbestsiniz.

Bu noktada en az 6 adet sunucu metodu ile bilgi alma (GetCurrentUserID, GetCurrentUserRoles, GetUserNames, GetIssues), yeni kullanıcı ekleme (AddUsers) veya yeni rapor ekleme (ReportNewIssue) gibi işlemleri yapabilmekteyiz. GetUserNames ve GetIssues metodları dataset döndürür ama bu dataset'ler de salt okunur olup üzerinde herhangi bir değişiklik yapılamaz.

Eğer Developer (Geliştirici) rolündeki kullanıcılara raporları almak ve yorum yazmak iznini vereceksek, o zaman sunucu modülüne şu anda sunucu metodlarının sunduklarının ötesinde fonksiyonellik eklememiz gerekecektir.

--- Veriyi Dışarı Aktarmak - Açık Sorunlar ---

Eğer bir geliştiriciye raporlanmış sorunları sadece göstermek değil aynı zamanda düzelttirmek istiyorsak, veri üzerinde değişiklik yapabilecek (ekleme, güncelleme veya silme) ve bu değişiklikleri sunucuya gönderecek, istemci tarafından kullanılabilecek bir TDataSetProvider bileşenini dışarıya aktarmamız gerekmektedir.

Bir sorun raporu üzerinde çalışan geliştirici, sadece ilk sorun raporunu görmemeli, aynı zamanda belirlenen sorun hakkında yazılmış bütün yorumları da görebilmelidir. Sonuç olarak TDataSetProvider sadece Raporlar tablosundaki kayıtları dışarıya aktarmayacak, aynı zamanda master-detay ilişkisi içerisinde Yorumlar (Comments) tablosundaki kayıtları da dışarıya aktaracaktır.

Bunu gerçekleştirebilmek için sqlReportUser ve sqlComments adında 2 adet TSQLDataSet ve dsReportUser adında bir TDataSource bileşeni eklememiz gerekir. dsReportUser'in DataSet özelliği sqlReportUser'ı işaret edecek ve sqlComment bileşeninden dönecek DataSource olarak kullanılacaktır.

En sonunda dspReportUserWithComments adında bir TDataSetProvider bileşeni eklenecek ve DataSet özelliğine de master olan sqlReportUser dataset'i atanacaktır.

İki TSQLDataSet bileşeninin SQLConnection özellikleri de TSQLConnection bileşenini göstermelidir doğal olarak. sqlReportUser bileşeninin SQL özelliği de aşağıda belirtildiği gibi Raporlar tablosundan girilen parametreler olan minimum ve maksimum değerler arasındaki kayıtlardan ReporterID ve AssignedTo alanları o anki Kullanıcıya eşit olan kayıtları getirecektir.

```
SELECT "ReportID", "Project", "Version", "Module", "IssueType",
"Priority", "Status", "ReportDate", "ReporterID", "AssignedTo",
"Summary", "Report"
FROM "Report"
WHERE (Status >= :MinStatus) AND (Status <= :MaxStatus)
AND ((AssignedTo = :AssignedTo) OR (ReporterID = :ReporterID))
```

Unutmamak gerekir ki o anki kullanıcının raporladığı sorunlar ve o kullanıcıya atanan raporlar beraber dönecektir bu sorgu sonucu. Aynı kullanıcıya ait olan dikkate alınan sorunlar da olabilir bunlar.

Bu sorgu cümleciği bize 12 adet alan döndürür. sqlReportUser bileşenine çift tıklayarak Fields Editör (Alan Editörü) ü başlatırız ve bu editöre sağ tıklayarak "Add all fields" "Bütün alanları ekle" seçeneğini seçeriz. Bu bize sorgunun oluşturacağı 12 adet alanı üretir.

Buradaki alanlardan bazıları özel davranış istemektedir. ReportID alanı, otomatik olarak artan bir alan olduğundan buna örnek olarak verilebilir. Bu demektir ki bu alanın ProviderFlags özelliğini düzeltmemiz gerekir. pfInUpdate alt özelliğine False değeri atamalıyız (bu birincil anahtara herhangi bir ilk değer ataması yapamayız, ayrıca

herhangi bir şekilde güncelleme de yapamayız.), ama pfInKey alt özelliğine True değerini atamalıyız. pfInWhere zaten True değerini almış olmalıdır.

Unutmamak gerekir ki sorgu cümlesinde 4 adet parametre (MinStatus, MaxStatus, AssignedTo ve ReporterID) bulunur. Bunlardan ikisi istemci tarafından sağlanır ve diğer ikisi sunucu tarafından otomatik olarak doldurulur (ve sonuç olarak listeden kaldırılması gerekir.)

MinStatus ve MaxStatus girdi parametreleri ftInteger türünde olmalı ve bu şekilde saklanmalıdır. AssignedTo ve ReporterID parametreleri ise listeden kaldırılmalı ve istemci tarafta gösterilmemelidir. (sunucu tarafında o anki kullanıcının oturumundaki UserID alanına bağlı olarak doldurulmalıdır.)

Detaya bakacak olursak, sqlComments dataset bileşeni, DataSource özelliğinde dsReportUser'ı böylece master (ana,ata) sqlReportUser'ı işaret eder. Yorumlarla ilgili olarak yazılacak sorgu cümlesinde ReportID adında bir parametre kullanılarak Yorumları (Comments) Raporlara (Reports) bağlayarak aşağıdaki gibi yazılır.

```
SELECT CommentID, ReportID, UserID, CommentDate, Comment
FROM Comment
WHERE (ReportID = :ReportID)
```

Şimdi de birincil anahtar (Primary Key) olan CommentID alanının PropertyFlags özelliğini düzenlemek gerekecek. Bu yüzden sqlComments bileşenine çift tıklarayak çıkan Alan Editörüne sağ tıklayarak daha önce de yaptığımız gibi "Add all fields" seçeneğini seçeriz.

CommentID alanını seçeriz, Object Inspector içerisinde çıkan PropertyFlags ağacını genişletiriz. pfInUpdate alt özelliğine False, pfInKey alt özelliğine True (pfInWhere zaten True olmalı) değerlerini atarız. Bu şekilde CommentID alanının herhangi bir şekilde sunucuya atama amaçlı gönderilemeyeceğinden ve bir kayda konumlanabilmek için herhangi bir sorgu cümlesinden Where deyimi içerisinde kullanılabileceğinden emin oluruz.

sqlComments dataset'i de ana sqlReportUser dataset'ine bağlandığından TDataSetProvider ikisini de iç-içe çağrılmış datasetler olarak dışarı aktaracaktır. (Raporu master olarak, yorumu da içine yuvalanmış detay dataset olarak.) Bunu istemci uygulamada daha detaylı olarak göreceğiz.

dspReportUserWithComments bileşeninde değiştirilmek isteyebileceğiniz bir değer daha bulunur, UpdateMode değeri. Varsayılan olarak bu özellik upWhereAll değerini alır. Bu demektir ki UPDATE ve DELETE sorgu cümleleri için, doğru kayda ulaşmak için bütün alanlar WHERE deyimi içerisinde yer almalıdır. (Blob alanları istisna olarak tutuyoruz) Bu durum, eğer tablonun birincil bir anahtarı yoksa ihtiyaç duyulan bir şeydir. Bununla beraber, eğer tablomuzda bir birincil anahtarımız varsa, bu özelliği upWhereAll'dan upWhereChanged olarak değiştirmeliyiz. Bu ayarlama ile WHERE deyimi sadece birincil anahtar ve ve değişen alanları içerir. (Değişmeyen alanları içermez) Bu bize kayıtları birleştirme, ki bu durumda düzeltilen sorun raporları oluyor bu, imkanı sağlar (Bu durum, 2 farklı şahıs aynı anda aynı alanı düzenlemediği sürece gerçekleşir)

En sonunda unutmamak gerekir ki upWhereKeyOnly adında da bir seçenek vardır ama bu seçenek tehlikelidir. Çünkü UPDATE ve DELETE sorgularında WHERE deyimine diğer kullanıcıların yapacakları değişiklikleri yoksayarak sadece birincil anahtarı parametre olarak gönderir.

İki dataset de tek yönlü salt okunur dataset'ler olduğundan master-detay ilişkisinin çalışması için son bir iş yapmamız gerekmektedir. Master veriseti üzerinde ilerlerken, ne zaman ana kayıt setinde bir sonraki kayda bakarsak o zaman detay dataset'ini de sıfırlamamız gerekir. Bu detay dataset'i de tek yönlü olduğundan otomatik olarak yapılamayacak bir işlemdir. sqlReportUser'in AfterScroll olay

tetikleyicisinde sqlComments datasetini kapatıp tekrar açabiliriz ve yeni ReportID değerini parametre olarak gönderebiliriz. İlgili kod aşağıdadır.

```
procedure TServerMethods1.AS_sqlReportUserAfterScroll(DataSet: TDataSet);
begin
	sqlComments.Close;
	sqlComments.ParamByName('ReportID').AsInteger :=
	sqlReportUser.FieldByName('ReportID').AsInteger;
	sqlComments.Open;
end;
```

Bu aslında, master dataset'i üzerinde her scroll hareketinde detay sorgunun çalıştığından ve sunucu tarafta içiçe dataset'i doldurup TDataSetProvider üzerinden istemciye aktardığımızdan emin olmamız anlamına geliyor.

Unutmamak gerekir ki olay tetikleyicisinin adı varsayılan adı olan sqlReportUserAfterScroll yerine AS_sqlReportUserAfterScroll olmalıdır. Bu bir amaç için yapılmıştır. Sunucu metodu sınıfındaki herhangi bir metod gösterilecektir ve buna sunucu taraftaki olay tetikleyicileri de dahildir. Bunun tek bir istisnası vardır, o da AS_ ile başlayan metodlardır. Bu metodlar gizlidir ve sunucu metodu sınıfından dışarıya gösterilmez. Bu bilgiyi olay tetikleyicilerini gizlemek için kullanırız, böylece istemci taraftaki sunucu metodu proxy unit dosyasında da tekrardan görünmemiş olur. (Zaten istemci tarafından dinlemede kullanışlı olmayan bir şeydir.)

İstemci taraf MinStatus ve MaxStatus değerlerini bize sağlayacaktır , biz de sunucu tarafında AssignedTo ve ReporterID alanlarını girmemiz gerekir. TDataSetProvider bileşeninin BeforeGetRecords olay tetikleyicisinde yapılabilir bu. Burada sqlReportUser açılmadan önce parametrelerini doldurabiliriz mesela. ReporterID ve AssignedTo alanlarının ikisine de o anki kullanıcının login olduktan sonra tutulan UserID bilgisini atarız. Bu aşağıdaki gibi yapılabilir.

```
procedure
TServerMethods1.AS_dspReportUserWithCommentsBeforeGetRecords(
	Sender: TObject; var OwnerData: OleVariant);
	var
		UserID: Integer;
	begin
		UserID :=
StrToIntDef(TDSSessionManager.GetThreadSession.GetData('UserID'),0);
		if sqlReportUser.Params.FindParam('ReporterID') = nil then
		begin
			with sqlReportUser.Params.AddParameter do
			begin
				DataType := ftInteger;
				ParamType := ptInput;
				Name := 'ReporterID';
				AsInteger := UserID
			end
		end
		else
			sqlReportUser.Params.FindParam('ReporterID').AsInteger :=
UserID;

		if sqlReportUser.Params.FindParam('AssignedTo') = nil then
		begin
```

```
                with sqlReportUser.Params.AddParameter do
                begin
                        DataType := ftInteger;
                        ParamType := ptInput;
                        Name := 'AssignedTo';
                        AsInteger := UserID
                end
        end
        else
                sqlReportUser.Params.FindParam('AssignedTo').AsInteger :=
UserID;
    end;
```

Bununla, sqlReportUser'ın bütün parametrelerinin sorgu çalıştırılmadan önce doldurulduğunu ve sonuç olarak, dspReportUserWithComments'deki Raporlar ve Yorumlardan talepte bulunan herhangi bir kullanıcı, sadece o anki kullanıcının rapor yazdığı, veya yazılan raporun o kullanıcıya atandığı veya her ikisini de görebilir. Sadece kendi raporlarınızı görebilirsiniz, başkasına ait olan raporları göremezsiniz. Bunun için Yönetici (Manager) rolüne sahip olmalısınız ve GetIssues metodunu çağırarak bütün raporlarla ilgili olarak kuşbakışı bir incelemeyi yapabilirsiniz. (Ama Yorumlar'ı göremezsiniz metodun özelliğinden dolayı)

--- TDataSetProvider İçin Rol Tabanlı Yetkilendirme ---

DataSnap sunucusunda unuttuğumuz son bir şey kaldı. TDataSetProvider için rol tabanlı yetkilendirme. Sunucu metodlarından farklı olarak, sunucu modülündeki bileşene herhangi bir özel özellik ekleyemeyiz. Bunun yerine sunucu modülünden DirtWebMod'a geçer ve TDSAuthenticationManager bileşenini kullanırız.

Bu bileşenin Roles (Roller) adında ApplyTo, AuthorizedRoles, DeniedRoles adında alt özellikleri olan ve koleksiyon olarak tanımlayabileceğimiz bir özelliği bulunur. ApplyTo özelliği metod adı, sınıf adı veya sınıf adı.metod adı gibi özelleştirmeleri yapabildiğimiz bir yerdir. Bizim projemizde ben sadece Geliştirici roldeki kullanıcıların çağırmasına izin verilecek önceden tanımlanmış 7 adet IAppServerFastCall metodu belirlemek istiyorum. Bunun için aşağıdaki resimdeki gibi Apply özelliğine bu 7 metodu eklemem gerekir.

Unutmamak gerekir ki eğer bu metodların güvenliği için özelleşmiş bir sunucu metodu sınıfına ait olduğunu belirtmek istiyorsak bunlara önek olarak TServerMethods1 eklemeliyiz.

Apply özelliğini bir kere belirledikten sonra , AuthorizedRoles (Yetkilendirilmiş Roller) özelliğine Developer (Geliştirici) değerini ekleyebiliriz. Böylece dışarı aktarılan TDataSetProvider'dan sadece Geliştirici rolüne sahip kullanıcıların yararlanmasına izin verileceğini belirlemiş oluruz.

Sadece Geliştirici rolündeki kullanıcıya raporu düzenleyerek yorum ekleme izninin verilmesini katı bir karar olarak görebilirsiniz. Bu listeye Testçi ve Yönetici rollerini eklemekte özgürsünüz. (Buna rağmen kullanıcıya raporu düzenleme ve/veya yorum ekleme özelliğini vermek istiyorsanız bu Yönetici'ye bağlıdır.)

--- Sunucu Dağıtımı ---

DataSnap DIRT Sunucusunun güvenli bir şekilde dağıtımının sağlanması için, HTTPS protokolünü kullanarak ISAPI DLL'ini IIS üzerinden dağıtmak (yayınlamak) ya da

iletişim kanallarını kriptolamak için RSA ve PC1 filtreleri ile tek başına DataSnap Sunucusu olarak dağıtmak gerekir. Son durumda HTTP kullanmayıp, iletişim (nakil) protokolü için TCP/IP kullanmanız tavsiye edilir. Çünkü TCP/IP daha hızlıdır. Gerçek hayattaki uygulamalardaki dağıtım seçeneklerimiz şunlardır

- HTTPS protokolünü kullanarak ISAPI DLL'ini IIS üzerinden dağıtmak
- PC1 ve RSA filtreleri ile tek başına TCP/IP kullanarak

IIS'in ek yararı uygulama havuzlarını yapılandırmamıza izin vererek otomatik yük dengeleme, geri dönüşüm, sunucunun bellek ve işlemci kullanımına sınırlama getirme gibi destekleri sunabilmesidir. Bu sebeple DIRT DataSnap Sunucusu domain için kurulmuş bir sertifikaya sahip olan bir web sunucusu üzerindeki IIS'te ISAPI DLL olarak dağıtılmış ve yayınlanmıştır. Bu dökümandaki amaçlar için www.bobswart.nl adresi kullanılmıştır. Sanal bir DataSnapXE dizini oluşturulmuş, ve bu sanal dizin içerisinde uygulamalar için özel bir uygulama havuzu yapılandırılmıştır. En sonunda, bu sanal dizin içerisine DirtServer.dll dosyası yerleştirilmiştir. Dosya https://www.bobswart.nl/DataSnapXE/DirtServer.dll URL adresindedir.

Bu URL adresini çağırmak bize sadece "DataSnap XE Server" yazısını gösterir ama bu en azından sunucunun yayınlandığının doğrulaması açısından yeterlidir. Elbette DirtServer.dll dosyasının veritabanına bağlanabildiğinden de emin olmamız gerekmektedir. Bu amaç için ihtiyaç duyulan veritabanı sürücü dosyasının da sunucu makinasında dağıtılmış olması gerekmektedir. Mesela SQL Server 2008 için dbxmss.dll dosyasını arama yoluna yerleştirmemiz gerekir (Bu yol windows\system32 gibi DirtServer.dll dosyasının yükleyebileceğinden emin olduğumuz bir yer olmalı).

Unutmamak gerekir ki DirtServer.dll dosyası ve dbxmss.dll dosyası dışında herhangi bir dosyayı web sunucusunda dağıtmamıza, yayınlamamıza gerek yoktur. MIDAS.dll dosyasına da ihtiyacımız yoktur, ilgili unit dosyalarında MidasLib'i uses kısmına eklediğimiz zaman sunucu proje içerisine MIDAS.dll dosyasını otomatik olarak bağlamaktadır zaten.

Eğer DataSnap sunucusunu tek başına TCP/IP protokolünü ve RSA ve PC1 filtrelerini kullanarak dağıtmak, yayınlamak isterseniz ayrıca 2 adet özel Indy SSL DLL'ini (libeay32.dll ve ssleay32.dll) de dağıtmak, yayınlamak ya da sunucu makinada bulunduğundan emin olmanız gerekir. Bu DLL'lere RSA filtresi tarafından ihtiyaç duyulur. (RSA filtresi, PC1 filtresi kullanan şifreyi kriptolamak için kullanılır) Bu DLL'ler olmadan (sunucu makinada) sunucuya bağlanmak isteyen istemciler "Connection Closed Gracefully" yani "Bağlantı İncelikle Kapandı" mesajı ile karşılaşırlar. Çünkü sunucu PC1 anahtarlarını kriptolamak için gerekli olan RSA filtresini başlatacak 2 DLL dosyasını bulamamıştır.

Sırası gelmişken, bu 2 DLL dosyası ISAPI DLL'ini HTTPS yöntemi olsa da , TCP/IP ve RSA ile PC1 ile kullanılsa da istemci taraf da ihtiyaç duyar. Ama önce istemciyi oluşturalım, dağıtımı için sonra endişeleniriz.

--- DataSnap İstemcisi ---

DIRT DataSnap Sunucusunu dağıttıktan ve test ettikten sonra DataSnap İstemci'sini yazmaya başlayabiliriz. Bu akıllı bir istemci olacak, USB veya herhangi benzer bir şeye konulabilecek tek başına çalışan bir uygulama (.exe) dosyası olacak ve herhangi bir Windows makinasında çalışabilecek (Bunun için DIRT projesine erişim için, https://www.bobswart.nl adresine erişebilmek için internet bağlantısının olması gerekmektedir.)

Yeni bir VCL Form uygulaması oluştururuz, ve DataSnap sunucusuna erişecek bileşenleri yerleştirmek için bir de Data Module (Data Modülü) ekleriz. İlk adım olarak Data Modül üzerine bir TSQLConnection bileşeni yerleştiririz. Driver (Sürücü) özelliğine

DataSnap değerini atarız, sonra bu özellik ağacını genişleterek Object Inspector üzerinde Driver özelliğinin alt özelliklerini görürüz. Burada CommunicationProtocol alt özelliğine https, Port'a 443, HostName alt özelliğine de www.bobswart.nl (kendi sunucunuza bağlanmakta özgürsünüz tabi ki) değerlerini atarız. URLPath özelliğine web sunucusu üzerinde DirtServer'ın bulunduğu ../DataSnapXE/DirtServer.dll değeri verilmelidir. LoginPrompt özelliğine False değerini atamayı unutmamalısınız, yoksa sunucuya ne zaman bağlanmak isteseniz karşınıza bir bağlantı diyaloğu çıkar. Ve bu bizim istemciden sunucuya gerçek şifre yerine onun hashlenmiş halini göndermek istediğimizden her defasında diyalog çıkması bizim için son derece kullanışsız bir durumdur.

Burada bir dikkat çekici bir husus var.. Doğru bir DSAuthUser ve DSAuthPassword değeri olmadan sunucuya bağlanmamıza izin verilmez, sonuç olarak da DataSnap İstemci Sınıflarının oluşturulması başarısız olur. Benim makinamdaki DirtServer için bu bir problem olur. (Ama eğer isterseniz proje arşivinde daha önceden oluşturulmuş istemci sınıf unit dosyası olan DBXClientClasses.pas dosyasını kullanabilirsiniz.) Kendi DataSnap Sunucunuz için, OnUserAuthenticate olay tetikleyicisini devre dışı bırakmanız gerekir, Valid alanına True değerini atayarak öncelikle İstemci sınıfı oluşturur, sonra yetkilendirme işlemi için gerekli kontrolleri tekrar etkinleştirirsiniz.

İstemci sınıflar bir kere oluşturulduktan sonra public türdeki sunucu metodlarından oluşan TServerMethods1Client sınıfını görebilirsiniz.

```
type
TServerMethods1Client = class(TDSAdminClient)
        private
                ....
        public
                constructor Create(ADBXConnection: TDBXConnection);
overload;
                constructor Create(ADBXConnection: TDBXConnection;
                AInstanceOwner: Boolean); overload;
                destructor Destroy; override;

                function GetCurrentUserID: Integer;
                function GetCurrentUserRoles: string;
                function GetUserNames: TDataSet;
                procedure AddUser(User: string; Password: string;
                Role: string; Email: string);
                function ReportNewIssue(Project: string; Version: string;
                Module: string; IssueType: Integer; Priority: Integer;
                Summary: string; Report: string; AssignedTo: Integer): Boolean;
                function GetIssues(MinStatus: Integer; MaxStatus: Integer):
TDataSet;
end;
```

Ama öncesinde TServerMethods1Client'ten bir varlık oluşturmalı, ve sunucu metodlarını çağırarak, öncelikle sunucuya bir bağlantı yapabildiğimizden emin olmamız gerekmektedir.

--- Bağlantı ---

Projeye yeni bir form ekler ve onu bizim bağlantı formumuz gibi aşağıdaki gibi tasarlarız.

Şifre girebilmemiz için edPassword olarak adlandırdığımız TEdit bileşenimizin PasswordChar özelliğine gerekli değeri atadığımızdan emin olmalıyız.

Bu diyalog bir model diyaloğu olarak gösterilebilir, ve sonuç kullanıcı adı ve şifre TSQLConnection'un Params özelliğine gönderilecektir, ekleme veya var olan değeri değiştirme yöntemiyle. Bir Login (Bağlantı) olay tetikleyicisinde sunucu uygulamada yaptığımızla aynı yolla şifreyi hashlememiz mümkündür. İlgili kod aşağıdadır.

```
procedure TFormClient.Login1Click(Sender: TObject);
var
	MD5: TIdHashMessageDigest5;
	Server: TServerMethods1Client;
	UserRoles: String;

begin
	DataModule1.SQLConnection1.Connected := False;
	UserID := -1;

	with TFormLogin.Create(Self) do
	try
		if ShowModal = mrOK then
		begin

SQLConnection1.Params.Values['DSAuthenticationUser'] := Username;
			MD5 := TIdHashMessageDigest5.Create;
			try

SQLConnection1.Params.Values['DSAuthenticationPassword'] :=
				LowerCase(MD5.HashStringAsHex(Password,
TEncoding.UTF8));

					SQLConnection1.Connected := True; // try
to login...
					Server :=
TServerMethods1Client.Create(SQLConnection1.DBXConnection);
					try
					UserID := Server.GetCurrentUserID;
					UserRoles := Server.GetCurrentUserRoles;

					// Adjust GUI capabilities based on
UserRoles...

					finally
					Server.Free
					end
			finally
					MD5.Free
			end
		end
	finally
		Free // TFormLogin
	end
```

end;

Unutmamak gerekir ki kullanıcı rollerine bağlı olarak GUI kısmında (arayüzde) etkinleştirilmeyecek veya gizlenecek parçalarla ilgili olarak bütün kodu göstermedim. Bu okuyucu için egzersiz olarak bırakılmıştır. Elbette DirtServer ve DirtClient projesinin kaynak kodlarını indirebilir ve detaylı olarak inceleyebilirsiniz.

Login formu DataSnap sunucusuna var olan bağlantıyı kapatacak ve TSQLConnection bileşeninin DSAuthenticationServer ve DSAuthenticationPassword özelliklerine yeni değerler atayacaktır. Bağlantının başaarılı olup olmadığı, GetCurrentUserID ve GetCurrentUserRoles sunucu metodlarının çağrılmasından sonra bağlantının yeniden kurulmasıyla görülebilir.

--- Data Modül ve Sunucu Metodları ---

Data Modül üzerindeki TSQLConnection bileşenini kullanarak DataSnap istemci sınıflarını oluşturduktan sonra egzersiz olarak sunucu metodlarını direk çağırmaya nadiren ihtiyaç duyarız. Login kodu zaten GetCurrentUserID ve GetCurrentUserRoles olmak üzere 2 metodu çağırıyor, ama diğer sunucu metodları genelde bir salt okunur olarak veya düzeltilebilir bir şekilde bir dataset'i almak içindir. Bunun tek istisnası ReportNewIssue metodudur ve bunu gelecek konularda irdeleyeceğiz.

Şimdilik data modüle dönerek, 4 tane yeni bileşeni modül üzerine yerleştiririz. Bunlar; SqlServerMethodGetIssues olarak isimlendireceğimiz TSqlServerMethod, dspIssues olarak isimlendireceğimiz TDataSetProvider, cdsIssues olarak isimlendireceğimiz TClientDataSet ve dsIssues olarak isimlendireceğimiz bir TDataSource bileşenidir.

SqlServerMethodGetIssues bileşeni, SQLConnection özelliği ile SQLConnection1'e bağlanma ihtiyacı duyar, ki bu şekilde DataSnap sunucusu ile iletişim kurabilelim. ServerMethodName sürükle bırak listesini kullanarak çalıştırmak istediğimiz sunucu metodunu seçeriz. Unutmamak gerekir ki TSQLConnection bileşeni çalışan bir DataSnap Sunucusuna bağlanmalı ki, sonra çalıştırabileceğimiz sunucu metodlarını görebilelim.

Bizim örneğimiz için burada, TServerMethods1.GetIssue sunucu metodunu çalıştırmaya ihtiyacımız var. Bu metod sadece Yönetici rolündeki bir kullanıcı tarafından çağrılabilir, ve bütün sorunlar ile salt okunur bir dataset geri döndürür. (Belirlenen Status değerleri arasındaki)

SqlServerMethodGetIssues'de Sürükle bırak listesinden, sunucu metodlar seçilebileceği gibi, Params özelliğine de bakmamız gerekir. Burası bize 3 adet parametre gösterecektir. (MinStatus, MaxStatus ve ReturnParameter (Dönüş parametresi, bir dataset)).

MinStatus ve MaxStatus parametreleri için bir varsayılan değer tanımlayabilir veya kod içerisinde değer atamalarını yapabilirsiniz. (Bir sonraki kod parçasının göstereceği şey budur)

dspIssues DataSetProvider'ı DataSet özelliği ile SqlServerMethodGetIssues'e bağlanmalıdır.

Sonra cdsIssues ClientDataSet bileşeninin ProviderName özelliğine dspIssues değerini atamalıyız. Sonuç olarak dönecek dataset üzerinde oynama yapamaycağımızdan cdsIssues'in ReadOnly özelliğine True değerini atamamız iyi bir fikirdir, böylece bu dataset'e bağlanacak herhangi bir veri erişimi ile ilgili bileşen kullanıcıya değişiklik yapması için izin vermeyecektir. (İlerde değişikliklerin kaydedilmediğini veya sunucuya geri gönderildiğini öğrendiğiniz zaman sizi çıldırtabilecek bir özelliktir.)

En sonunda dsIssues DataSource bileşeninin DataSet özelliğine cdsIssues değerini atarız ve böylece dsIssues'in veri ile ilgili kontrollerine bağlanabiliriz.

Şimdi istemcinin ana formunda bu veya diğer sunucu metodları vasıtası ile raporlarla dolduracağımız TDBGridReports olarak adlandıracağımız bir TDBGrid yerleştirelim. GetIssues sunucu metodunun sonuçlarını yerleştirmek ve MinStatus ve MaxStatus filtrelerine dışarıdan değerleri geçmek için aşağıdaki kodu yazarız.

```
procedure TFormClient.ViewAllIssues1Click(Sender: TObject);
// Manager - all reports (read-only)
begin
	DBGridReports.DataSource := nil;
	DataModule1.SQLConnection1.Connected := True;
	try
		DataModule1.cdsIssues.Close;
		DataModule1.SqlServerMethodGetIssues.Params.
		ParamByName('MinStatus').AsInteger := MinStatus;
		DataModule1.SqlServerMethodGetIssues.Params.
		ParamByName('MaxStatus').AsInteger := MaxStatus;

		DataModule1.cdsIssues.Open;
		DBGridReports.DataSource := DataModule1.dsIssues;
	finally
		DataModule1.SQLConnection1.Connected := False
	end
end;
```

Unutmamak gerekir ki data modül üzerindeki TSQLConnection bileşenini kullanarak DataSnap Sunucusuna bağlandık, SqlServerMethodGetIssues'in parametrelerini doldurduk, sunucu metodunu çağırdık, DBGridReport bileşenin DataSource özelliğine datasource'u atayarak sonucu görebildik ve en sonunda DataSnap sunucu bağlantısını tekrardan kapadık. Bununla beraber bağlantıyı açık tutabilirsiniz, talepten sonra bağlantının kapatılması istemciyi daha sağlam yapar (her zaman taze bir bağlantı ile başlamış olursunuz.) ama biraz yavaşlatır (her zaman bağlantıyı açmaya ihtiyacınız olacaktır.), ama sunucu biraz daha ölçülebilir olur (sadece aktif bağlantılar korunmalıdır.)

Sonuç olarak dönen dataset, IssueType (Sorun Türü), Priority (Öncelik), Status (Durum) gibi sadece tamsayı değer içeren alanlar da içerecektir. Bu değerleri daha insan dostu string değerlere çevirebiliriz. Data modül üzerindeki cdsIssues ClientDataSet bileşenine çift tıklama suretiyle Fields Editör'ünü açarız ve sağ tıklayarak "Add all fields" "Bütün alanları ekle" seçeneğini seçeriz.

Sorun Türü, Öncelik ve Durum alanları için OnGetText olay tetikleyicisi içerisinde basit tamsayı değerleri insan dostu string değerlere çeviren aşağıdaki kodu yazabiliriz.

```
procedure TDataModule1.IssueTypeGetText(Sender: TField;
var Text: string; DisplayText: Boolean);
begin
	case Sender.AsInteger of
		1..4: Text := 'Minor ' + Sender.AsString;
		5..8: Text := 'Major ' + Sender.AsString;
		9..10: Text := 'Critical ' + Sender.AsString;
	end;
end;
```

```
procedure TDataModule1.PriorityGetText(Sender: TField;
var Text: string; DisplayText: Boolean);
begin
	case Sender.AsInteger of
		1..3: Text := 'Low ' + Sender.AsString;
		4..7: Text := 'Medium ' + Sender.AsString;
		8..10: Text := 'High ' + Sender.AsString;
	end;
end;

procedure TDataModule1.StatusGetText(Sender: TField; var Text: string;
DisplayText: Boolean);
begin
	case Sender.AsInteger of
		1: Text := 'Reported';
		2: Text := 'Assigned';
		3: Text := 'Opened';
		6: Text := 'Solved';
		7: Text := 'Tested';
		8: Text := 'Deployed';
		10: Text := 'Closed';
	end;
end;
```

Bu alanlara kendi çevirmeniz için istediğiniz kodu yazmakta özgürsünüz, ama amacımız için bu kod parçası yeterlidir. Unutmamak gerekir ki orjinal tamsayı değeri hala Sorun Türü ve Öncelikte gösterilmektedir ki gerçek değerin ne olduğu ufak da olsa hatırlatılsın diye.

--- UserNames (Kullanıcı Adları) Sunucu Metodu ---

Benzer bir yolla GetUserNames sunucu metodunu çağırmak için bir TSqlServerMethod bileşeni yerleştiririz data modül üzerine. Ayrıca dspUserNames olarak adlandıracağımız bir TDataSetProvider ve kullanıcı adları ile UserID değerlerinin listesini tutmak için de cdsUserNames adında bir TClientDataSet bileşeni yerleştiririz.

SqlServerMethodsGetUserNames adındaki TSqlServerMethod bileşeni ile TSQLConnection'a bağlanıp buradan TServerMethods1.GetUserNames metodunu seçeriz. dspUserNames DataSetProvider'i ise cdsUserName ClientDataSet'ine bütün kayıtları alabilmesi için bağlamak amacıyla kullanılır.

Fields Editör kullanarak cdsUserNames'in Name ve UserID olmak üzere 2 alan tuttuğunu doğrularız. Bu dataset bir dizi ismi sağlamak için bir destek dataseti olarak veya bir UserID değerini (mesela AssignedTo veya ReporterID alanlarından gelen) ID'den okunabilir bir isme çevirmek için kullanılabilir.

--- DSProviderConnection ---

Şimdi Geliştirici veya Testçi rolündeki bir kullanıcı için raporlanan olaylara bakalım. Bu durumda GetIssues sunucu metodunu çağıramayız, ama sunucudan dışarı aktarılmış dspReportUserWithComments adındaki TDataSetProvider bileşenini kullanabiliriz.

Bunun için data modül üzerine bir TDSProviderConnection bileşeni, cdsReports ve cdsComments olarak adlandıracağımız 2 tane TClientDataSet ve cdsReports'a bağlanması için dsReports olarak adlandıracağımız bir TDataSource bileşeni ekleriz.

DSProviderConnection bileşeni DataSnap sunucusundaki sunucu metodu sınıfına bağlanacak bir bileşendir. Bizim projemizde bu sadece TServerMethods1 sınıfıdır. DSProviderConnection bileşenini SQLConnection özelliğine TSQLConnection değerini atamalıyız, ve sonrasında ServerClassName özelliğine sunucu metod sınıfının tam adını yazmalıyız. Bu isim TServerMethods1'dir (Kesinlikle sunucu metodu sınıfının adının aynı olup olmadığını kontrol edebilirsiniz.)

Bir sonraki adım olarak cdsReport ClientDataSet bileşenini DSProviderConnection bileşenine RemoteServer özelliğine gerekli değeri atayarak bağlarız. Her şey doğru bir şekilde kurulduğunda ve DataSnap sunucusu çalışıp gelen bağlantıları kabul ettiğinde cdsReports'un ProviderName özelliğindeki sürükle bırak listesini açabilir ve buradan dspReportUserWithComments'i seçebiliriz.

Sonuç olarak dspReportUserWithComments'ten dışarı aktarılan master-detay özelliğindeki dataset artık cdsReports içerisinde saklanabilir. Artık parametreler gibi alanları da yapılandırabiliriz. En başta cdsReports'a sağ tıklar ve Fetch Params seçeneğini seçeriz. Bu cdsReports'un Params koleksiyonunu dolduracaktır (MinStatus ve MaxStatus değerlerini otomatik atacak ama AssignedTo ve ReporterID alanlarını atmayacak çünkü biz onu sunucu tarafından engelledik istemcide görünmesini)

Bu parametrelere varsayılan değerler atayabiliriz ama kod içerisinde kısaca dışarıdan değer atamayı zaten yapacağız.

Ayrıca cdsReports üzerine çift tıklayıp Fields Editör'ünü başlatır ve burada sağ tıklayıp "Add all fields" seçeneğini seçeriz. Bu, bize en sonunda özel bir alan olan sqlComments ile birlikte 13 adet alan döndürür.

Bu alan içiçe detay kayıtları içerir, bu projede bunlar master Rapor kaydına ait olan "yorum" kayıtlarıdır. Buradaki TDataSetField alanını cdsComments adındaki TClientDataSet bileşenini doldurmak için kullanabiliriz.

cdsComments bileşenini seçerek ve Object Inspector'u kullanarak cdsReportssqlComments değerini bileşenin DataSetField özelliğine atarız. cdsComments dataset'i artık otomatik olarak cdsReports dataset'indeki master kaydın detay kayıtları ile doldurulacaktır. (Raporun yorumları)

Ayrıca cdsComments'e çift tıklayıp gelen Fields Editör'De sağ tıklayarak "Add all fields" diyerek alanlarını getiririz.

İki ClientDataSet'in de ilk alanları olan birincil anahtarında biraz yapılandırmaya ihtiyaç duyulmaktadır. ReportID ve CommentID alanlarını otomatik artan alanlar olarak belirlediğimizden bunları veritabanına yazmamamız gerekmektedir. Sonuç olarak cdsReportsReportID ve cdsCommentsCommentID alanlarının ProviderFlags özelliklerini düzenlemeli, pfInUpdate alt özelliklerini False, pfInKey alt özelliklerini True yapmalıyız. Böylece ClientDataSet bileşenleri birincil anahtarın gerçekten anahtar olduğunu bilir, ama bunları sunucuya yazamaz veya güncelleyemez.

--- Arama Alanları (Lookup Fields) Eklemek ---

Şimdi cdsReports'a geri dönelim ve alan listesine tekrar bakalım. Burada daha önceden Yönetici kullanıcısı için dönen cdsIssues'de olduğu gibi çevrilmesi gereken bir kaç alan vardır. Bu alanlar daha önce de kod içerisinde ele aldığımız OnGetText olay tetikleyicilerinde ele alınan Issue Type (Sorun Türü), Priority (Öncelik) ve Status (Sorun) alanlarıdır (Aynı data modülde ele alınan).

Ayrıca ReporterID ve AssignedTo alanları da çevrilmeye ihtiyaç duyulan alanlardır. Bu alanlar sadece UserID değeri tutan ama kullanıcı adı bilgisini tutmayan alanlardır. cdsUserName bileşenini arama kaynağı olarak kullanarak cdsReports üzerine

2 adet arama alanı ekleriz. Bunun için cdsReports'un Fields Editör'ünde sağ tıklayıp "New Field" "Yeni alan" seçeneğini seçer ve ReporterName adında yeni bir arama alanı ekleriz.

Aynısını cdsReports'taki AssignedTo alanı ile ilgili olarak AssignedName adında bir arama alanı daha ekleriz. Ve cdsComments içindeki UserID alanını unutmadan oraya da UserName adında bir arama alanı ekleriz. (Belgedeki örnek resme bakmak lazım)

--- Otomatik Artan Alanlar ---

Özellikle master - detay ilişkilerde otomatik artan alanların daha iyi çalışması için bir kaç fazla adım uygulamak gerekiyor. Sunucuya ReportID ve CommentID değerlerini yazamamamıza rağmen, istemci tarafta yeni bir sorun veya yeni bir yorum oluştururken bir değer atamasına izin vermemiz gerekir. Bu yeni eklenecek kayıtlar için -1'den başlayıp geriye doğru giden negatif anahtar değerler kullanmalıyız. İki ClientDataSet bileşeninde AfterInsert olay tetikleyicisinde aşağıdaki kodla bu gerçekleştirilebilir.

```
procedure TDataModule1.cdsReportsAfterInsert(DataSet: TDataSet);
begin
	cdsReportsReportID.AsInteger := -1;
end;

procedure TDataModule1.cdsReportsOrCommentsAfterPostOrDelete(DataSet:
TDataSet);
begin
	cdsReports.ApplyUpdates(-1)
end;
```

Unutmamak gerekir ki burada yazılan kod, sadece -1 değerini atayıp geriye doğru sayım yapmıyor. İstemci uygulama herhangi bir yazma veya silme olayında bütün güncellemeleri sunucuya gönderiyorsa geriye doğru sayıma gerek yoktur. Bu demektir ki olası negatif değere sahip 1'den fazla kayıt (o anda güncellenen) bulunmayacaktır. Ekleme veya düzenleme işleminden sonra yeni kayıt sunucuya gönderilecek, ve yapmamız gereken tek şey cdsReports ve cdsComments'in içeriğini yenileyerek otomatik artan değerleri almasını sağlamak olacaktır.

Bu amaçla iki ClientDataSet bileşeninin (cdsReports ve cdsComments) AfterPost ve AfterDelete olay tetikleyicilerindeki kodu yazmamız gerekmektedir. İyi haber yapmamız gereken şeyle ilgili olarak 4 olay tetikleyicisinin de tek bir yerden paylaşılabiliyor olmasıdır. İlgili kod aşağıdadır.

```
procedure TDataModule1.cdsReportsOrCommentsAfterPostOrDelete(DataSet:
TDataSet);
begin
	cdsReports.ApplyUpdates(0);
	cdsReports.Refresh
end;
```

Bu kod ile sunucuya herhangi bir değişikliği göndereceğimizi ve Refresh (yenile) metodu ile sunucudaki datayı yenileyeceğimizden emin oluyoruz.

Refresh (Yenileme) sadece birincil alan değerlerini yenilemez, seçtiğimiz kayıtlar içerisindeki bütün verileri yeniler. Bu büyük sayıdaki kayıtlar için akıllıca bir iş olmayabilir, kötü bir fikir olabilir ama bu durumda dışarıya aktarılan DataSetProvider

sadece o kullanıcıya ait olan ve MinStatus ve MaxStatus parametre değerleri arasında kalan Raporları (Yorumları ile birlikte) döndürdüğünden eğer testçi veya geliştirici rolündeki kullanıcı fanatikse belki kullanıcı başına binlerce kayıtlık iş olur.

Geliştirici veya Testçi rolündeki bir kullanıcının sorunları görebilmesi için gerekli kod aşağıdaki gibidir.

```
procedure TFormClient.ViewMyIssues1Click(Sender: TObject);
// Developer/Tester personal reports
begin
	DBGridReports.DataSource := nil;
	DataModule1.SQLConnection1.Connected := True;
	try
		DataModule1.cdsReports.Close;

DataModule1.cdsReports.Params.ParamByName('MinStatus').AsInteger :=
		MinStatus;

DataModule1.cdsReports.Params.ParamByName('MaxStatus').AsInteger :=
		MaxStatus;
		DataModule1.cdsReports.Open;
		DBGridReports.DataSource := DataModule1.dsReports;

		// nested dataset

DBGridReports.Columns[DBGridReports.Columns.Count-1].Visible := False;
	finally
		DataModule1.SQLConnection1.Connected := False
	end
end;
```

TDBGrid içerisindeki son kolon içiçe (yuvalanmış) bir datasettir. cdsReports içerisindeki Fields Editör'den bu alanın Visible özelliğine False değeri atayarak ya da TDBGrid içerisindeki son kolonun Visible özelliğine False değeri atayarak görünüp görünmeyeceğini ayarlayabiliriz. Yukarıdaki kod üzerinde ayarlamak da bir seçenektir.

Unutmamak gerekir ki en sondaki içiçe kolon dışında raporlanan sorunla ilgili kod SqlServerMethodGetIssues tarafından dışarı çıkarılan Yönetici rolündeki kullanıcının çağırdığı GetIssues metodunun koduna çok benzemektedir. Bununla beraber Yöneticinin çağırdığı metoddan dönen değer salt okunur bir datasettir (ReadOnly özelliğine gerekli değer atandığından), Geliştirici ve Testçi'nin çağırdığı metoddan dönen değer üzerinde değişiklik yapabileceğimiz bir datasettir.

Bununla beraber bu değişiklikler için kullanıcıların veriyi grid üzerinden düzeltmesini istemediğimizden kişisel sorun raporlarını düzenlemeleri için yeni bir form tasarlamak gerekiyor.

--- Raporlanan Sorun Formu ---

Oluşturulacak Raporlanan Sorun Formu, sadece 2 adet TDataSource (dsComments ve dsReports) ve master Rapor ve detay Yorum kayıtlarını gösterecek bir miktar veri kontrol bileşeni içerecektir. Amacımız için formun tasarımı aşağıdaki resimdeki gibi olacaktır ama siz kendi tasarımınızı yapmak konusunda özgürsünüz.

Yorumlar bir TDBGrid içerisinde, yorumlar ile ilgili olarak o anki yorumun detay bilgileri ise bir TDBMemo bileşeni içerisinde gösterilecektir. Yorum eklemek grid üzerinden mümkün olmayacak (öyle bir imkan olmasına rağmen), yeni bir formda yapılması daha iyi olur. Yeni yorum eklemek için özel bir form tasarlayacağız ve bununla ilgili kod aşağıdaki gibi olacaktır.

```
procedure TFormIssue.btnNewCommentClick(Sender: TObject);
begin
	with TFormComments.Create(Self) do
	try
		ShowModal;
	finally
		Free
	end
end;
```

Yeni form, 2 buton, bir adet TMemo ve bir adet scrollbar'dan oluşan basit bir görünüme sahip olacaktır. Bununla ilgili olarak aşağıdaki resme bakabilirsiniz.

Unutmamak gerekir ki sadece o andaki yorumun yeni bir yorum eklenmeye ihtiyacı vardı, istemci uygulama hangi rapora yorum ekleyeceğimizi bildiğinden (o anki rapor), o da o anki kullanıcıyı ve tarihi bildiğinden, elle gireceğimiz tek alan o an ekleyeceğimiz yorumun kendisidir.

Sonuç olarak yorum ekleme ile ilgili butonun Click olay tetikleyicisindeki kod aşağıdaki gibidir.

```
procedure TFormComments.BitBtn1Click(Sender: TObject);
begin
	DataModule1.cdsComments.Insert;
	DataModule1.cdsCommentsCommentID.AsInteger := -1;
	DataModule1.cdsCommentsUserID.AsInteger := UserID;
	DataModule1.cdsCommentsCommentDate.AsDateTime := Now;
	DataModule1.cdsCommentsComment.AsString := Memo1.Text;
	DataModule1.cdsComments.Post;
	Close
end;
```

Bu ayrıca diyaloğu kapatarak, kullanıcıya rapor ve yorumların yenilenmiş halini geri döndürecektir.

--- Yeni Sorun Raporlama ---

En son bir olayımız daha kaldı, yeni bir sorun raporlamak. Bunun için elle ReportNewIssue metodunu Project, Version, Module, IssueType, Priority, Summary, Report ve seçenekli olarak AssignedTo argümanlarını parametre olarak geçerek çağırmalıyız. Yapılması zor bir şey değildir, basit girdi kontrolleri ve sunucudaki metodu direkt çağırarak yapılabilir.

Bununla beraber cdsReports tablosuna yeni bir kayıt eklemek için DSProviderConnection bileşenine bağlanmış hazırda var olan cdsReports bileşenini kullanabiliriz. Otomatik olarak birincil anahtar değerine -1 atayarak, ve GUI (arayüz) için gerekli olan veri kontrol bileşenlerini ekleyerek bu işlemi yapabiliriz. ReportNewIssue sunucu metodunu yoksayarak (sadece Testçi rolündeki kullanıcıların

çağırmasına izin veriliyor), Testçi ve Geliştirici tarafından kullanılmasına izin verilen dataset sağlayıcı fonksiyonelliğini kullanabiliriz.

Veri kontrolüne (data aware) sahip Report New Issue formu aşağıdaki resimdeki gibi tasarlanabilir. Formda data modül üzerindeki cdsReports'a bağlanan bir TDataSource ve veri kontrol bileşenleri bulunur.

FormCreate olay tetikleyici içerisinde cdsReports dataset'ine bir Ekleme (Insert) işlemi yapılır. Bununla ilgili olarak UserID değerini (o anki kullanıcının) ReporterID değerini atayarak, Status değerini "reported" anlamına gelen 1 değerini atayarak, ReportDate değerine Now (şu an) değerini atayarak yaparız. Form üzerindeki ReporterID, Status ve ReportDate alanları salt okunur alanlardır.

Rapor eklemek için gerekli olan butonun (Formda OK yazan buton) Click olay tetikleyicisinde Post (yazma) işlemi yapar ve formu kapatır, Cancel (İptal) butonu da rapor ekleme işlemini iptal eder ve formu kapatır. İlgili kod aşağıdaki gibidir.

```
procedure TFormNewIssue.FormCreate(Sender: TObject);
begin
	dsReports.DataSet.Open;
	dsReports.DataSet.Insert;
	dsReports.DataSet.FieldByName('ReporterID').AsInteger := UserID;
	dsReports.DataSet.FieldByName('Status').AsInteger := 1;
	dsReports.DataSet.FieldByName('ReportDate').AsDateTime := Date;
end;

procedure TFormNewIssue.btnOKClick(Sender: TObject);
begin
	dsReports.DataSet.Post;
	Close
end;

procedure TFormNewIssue.btnCancelClick(Sender: TObject);
begin
	dsReports.DataSet.Cancel;
	Close
end;
```

Unutmamak gerekir ki, IssueType, Priority ve AssignedTo alanlarına gerekli olan değerler basit TDBEdit bileşenleri ile atanır. IssueType ve Priority alanları için sayısal spinedit bileşeni kullanmak iyi bir fikirdir, benzer olarak da AssignedTo alanına bilinen kullanıcı değerlerini atamak için bir sürükle bırak TComboBox bileşeni kullanmak iyi bir fikirdir. AssignedTo alanına ihtiyaç duyulmuyorsa TDBLookUpComboBox bileşeni kullanmamalıyız. (Eğer raporu herhangi bir kullanıcıya atamak istemesek bile bu bileşen herhangi bir kullanıcı seçmemize zorlar bizi.) Bu yüzden FormCreate olay tetikleyicisindeki kullanıcı adları ile doldurulan bir düzenli TComboBox bileşeni kullanırız. Bununla ilgili kod aşağıdadır.

```
DataModule1.cdsUserNames.Open; // in case it was closed
DataModule1.cdsUserNames.First;
cbAssignedTo.Items.Clear;
cbAssignedTo.Items.Add('Nobody');

while not DataModule1.cdsUserNames.Eof do
begin
	// current user name
```

```
            if DataModule1.cdsUserNamesUserID.AsInteger = UserID then
                edReporter.Text := DataModule1.cdsUserNamesName.AsString;

            cbAssignedTo.Items.AddObject(
            DataModule1.cdsUserNamesName.AsString,
            TUserID.Create(DataModule1.cdsUserNamesUserID.AsInteger));

            DataModule1.cdsUserNames.Next;
        end;
```

TUserID sınıfı UserID değerlerini tutmak ve ComboBox'ın Items özelliğine AddObject metoduna 2. argüman olarak gönderebilmek için oluşturduğum bir destek sınıfıdır.

```
        type
        TUserID = class
            UserID: Integer;
            constructor Create(const NewUserID: Integer);
        end;

        constructor TUserID.Create(const NewUserID: Integer);
        begin
            inherited Create;
            UserID := NewUserID
        end;
```

Bellek sızıntısını (yetmezliğini) engellemek için FormClose olay tetikleyicisi içerisinde form kapandığı zaman TComboBox bileşeni içerisindeki değerleri temizlememiz gerekir. Bununla ilgili kod aşağıdadır.

```
        procedure TFormNewIssue.FormClose(Sender: TObject;
        var Action: TCloseAction);
        var
            i: Integer;
        begin
            for i:=0 to cbAssignedTo.Items.Count-1 do
                if Assigned(cbAssignedTo.Items.Objects[i]) then
                    cbAssignedTo.Items.Objects[i].Free
        end;
```

Unutmamak gerekir ki o anki kullanıcının ad bilgisini edReporter olarak adlandırdığımız bir TEdit bileşenini Text özelliğine atamaktayız. Bu yüzden raporu onaylayanın ad bilgisini görme imkanımız bulunmaktadır. (Bu bizim kendi adımız olmalıdır)

Report New Issue formunun tasarımı aşağıdaki resimdeki gibi olacaktır.

Unutmamak gerekir ki IssueType ve Priority TSpinEdit bileşenlerinin varsayılan değerlerine 1, ve MinValue özelliği 1, MaxValue özelliğine de 10 değerleri atanmalıdır. Mesela daha acil sorunları raporlamak için bu bileşenlerin varsayılan değerlerini 5 olarak atayabilirsiniz, bu konuda özgürsünüz.

Son olarak opsiyonel olarak, o anki dataset üzerinde Project, Verison ve Module alanlarının değerlerini alabilmek için Project, Version, Module girdi alanları için TComboBox bileşenlerini form üzerine ekleyebiliriz. Bununla beraber bu okuyucuya egzersiz olarak bırakılmıştır.

--- İstemci Dağıtımı ---

Eğer DataSnap İstemci projesinin uses kısmına MidasLib değerini eklersek bu tek başına çalışan bir uygulama dosyası halini alır. Herhangi bir veritabanı sürücüsüne (database driver) ihtiyaç bulunmamaktadır. Bununla beraber ISAPI DLL'i HTTPS ve TCP/IP üzerinden tek başına çalışan uygulama dosyası (.exe) RSA ve PC1 filtrelerini kullandığından DataSnap istemcisi ile birlikte libeay32.dll ve ssleay32.dll dosyalarını da dağıtmalı (yayınlamalı) veya istemci makinada bulunduğundan emin olmalıyız.

DirtCleaner uygulama dosyası ve 2 adet DLL dosyasının DirtServer'a (veritabanı ve DataSnap sunucusu) bağlanması için internet bağlantısına ihtiyaç duyulmaktadır. Bunun dışında herhangi bir şeye ihtiyaç duyulmamaktadır. Sunucuya bağlanıp Yönetici, Testçi ve Geliştirici rollerinden birini alabilmek için geçerli bir kullanıcı adı ve şifreye ihtiyaç duyulmaktadır. DirtServer ve DirtCleaner projelerinin kaynak kodları indirilmek üzere bulunmaktadır. Bu projenin daha rafine versiyonu benim DataSnap XE kurs materyalimde bulunacak ve gösterilecektir. (Kayıtlı okuyucularım için rapor atlamaları ve hata raporları da kurs materyalim içerisinde bulunacaktır)