УНИВЕРЗИТЕТ У НОВОМ САДУ
ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА У НОВОМ САДУ

CLOUD COMPUTING U ELEKTROENERGETSKIM SISTEMIMA

Osnove *Microsoft WindowsAzure* servisa

-SKRIPTA-

Novi Sad, 2018

Vežba 3 – Windows Azure Storage service

U vežbi 3 je prikazan osnovni *API storage* servisa i prikazan je način čuvanja podataka u okviru *Blob* i *Table storage* servisa.

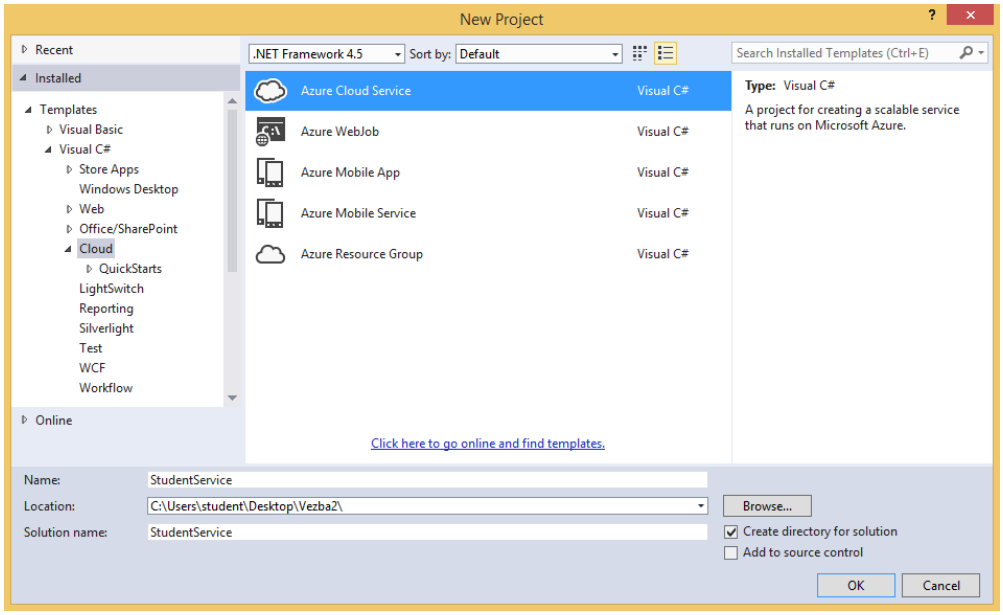
Vežba je izdeljena u tri celine. Prvi deo prikazuje kreiranje *cloud* projekta, u drugom delu je implementirana komunikacija sa *table storage* servisom i kreiran je *web role WindowsAzure* Servis, dok se treći deo odnosi na modifikaciju *web role* i kreiranje korisničkog interfejsa za aplikaciju.

1 Kreiranje *cloud* projekta koristeći *Visual Studio 2015*

U ovom delu vežbe, napraviće se *cloud* projekat koristeći šablon koji dolazi uz instalirani *SDK*.

Postupak za kreiranje *cloud* projekta:

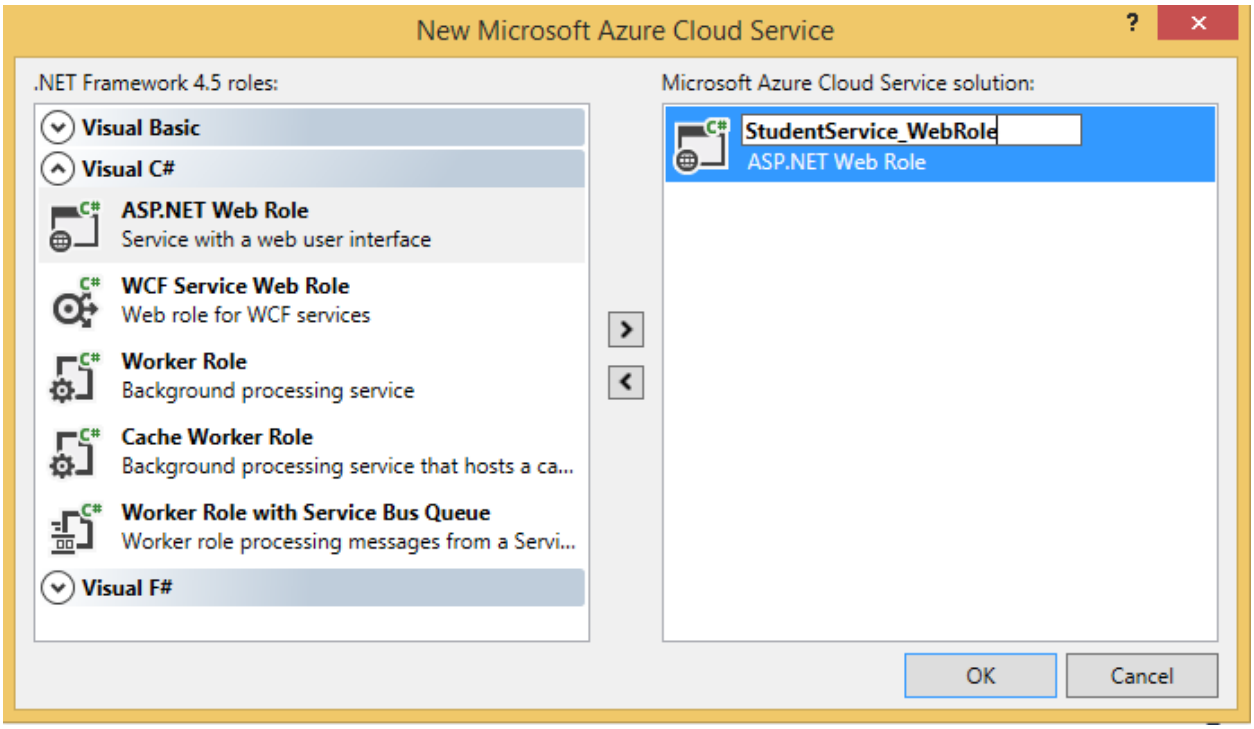
- Klikom na *File* meni, odabrati *New/Project*
- U *New Project* dijalogu, proširiti Visual C# u listi *Installed Templates* i selektovati *Cloud*, kao što je prikazano na slici 1. Odabrati šablon *Windows Azure Cloud Service*. Uneti ime projekta *StudentsService*. Kliknuti na OK.



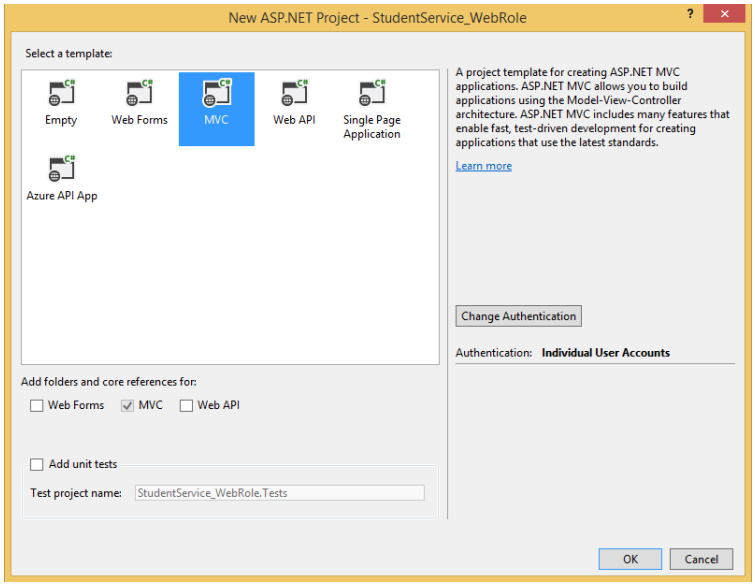
Slika 1 Kreiranje cloud projekta

U *New Windows Azure Project* dijalogu, u okviru *Roles* panela, otvoriti tab Visual C#. Selektovati *ASP.NET Web Role* kao što je prikazano na slici 2. Nazvati *web role* servis StudentService_WebRole.

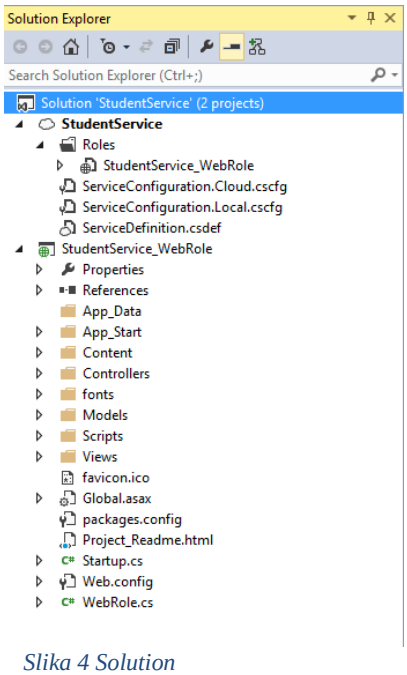
U narednom koraku izabrati *MVC* šablon i ostaviti podrazumevana podešavanja, kao na slici 3. Kliknuti OK. Nakon ovog koraka, generiše se solution koji sadrži dva projekta. Solution bi trebao da izgleda kao na slici 4.



Slika 2 Dodavanje web role



Slika 3 Kreiranje web projekta



Slika 4 Solution

Prvi projekat u rešenju sa slike 4, nazvan *StudentsService_WebRole*, predstavlja standardnu *ASP.NET MVC Web* aplikaciju ali modifikovanu za *Windows Azure compute* servis. Sadrži

dodatnu klasu koja predstavlja ulaznu tačku za *web role* procese i sadrži metode za inicijalizaciju, pokretanje i zaustavljanje role. *WebRole* klasa predstavlja šablon za implementiranje *Windows Azure* servisa.

Drugi projekat nazvan StudentsService predstavlja *cloud* projekat i sadrži konfiguraciju za *web* i *worker role*. Konfiguracije mogu biti lokalne i *cloud* konfiguracije. Na ovaj način *WindowsAzure* servis automatski učitava odgovarajuću konfiguraciju, i time omogućava lakše razdvajanje razvojnog okruženja od produkcionog.

2 Kreiranje modela podataka za entitete u *Table storage*

StudentsService aplikacija je zamišljena tako da prikazuje listu studenata tabelarno i da ima mogućnost dodavanja novog studenta. Podaci o studentu bi trebali da budu: broj indeksa, ime, prezime i slika studenta.

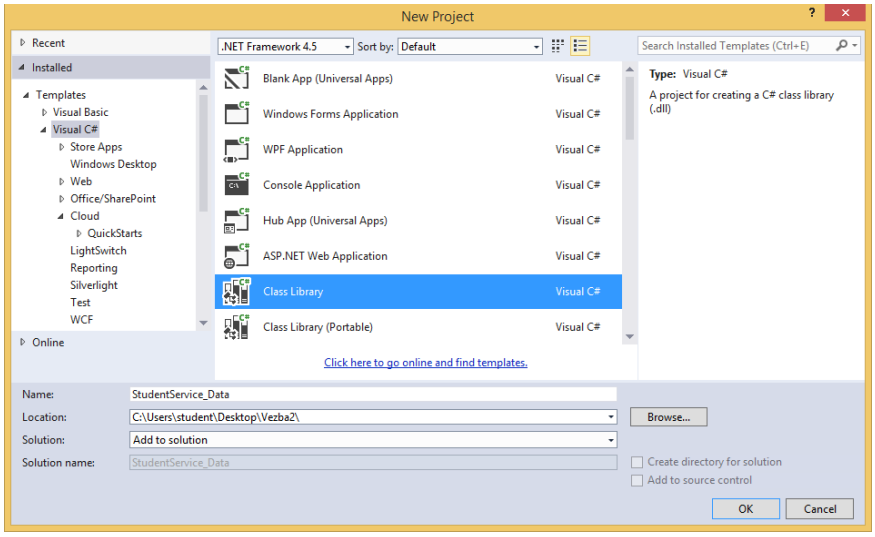
Da bi se mogla koristiti *WCF Data Services Client* biblioteka za pristup podacima u *table storage*, neophodno je kreirati *context* klasu koja nasledjuje klasu *TableServiceContext*. *TableServiceContext* klasa se nasledjuje iz *DataServiceContext* klase koja je deo *WCF Data Services* komponente. *Table Storage API* omogućava aplikacijama da kreiraju tabele koje koriste u ovim klasama. Da bi se kreirao odgovarajući kontekst, kontekstna klasa mora postaviti tabelu nad kojom će vršiti akcije, kao property tipa *IQueryable<SchemaClass>*, gde je *SchemaClass* klasa koja modeluje entitet. Jedna kontekstna klasa može sadržati više tabela.

U ovom delu vežbe 3, modeluje se entitet koji će *StudentsService* aplikacija koristiti i kreira se kontekstna klasa koja koristi *WCF Data Services* za pristup table storage servisu. Za model entieta je potrebno da bude ponovno upotrebljiv kroz više *WindowsAzure* servisa. Iz razloga ponovne upotrebljivosti, potrebno je kreirati biblioteku koja će sadržati model podataka i klase za rukovanje datim modelom.

2.1 Dodavanje novog projekta – biblioteke

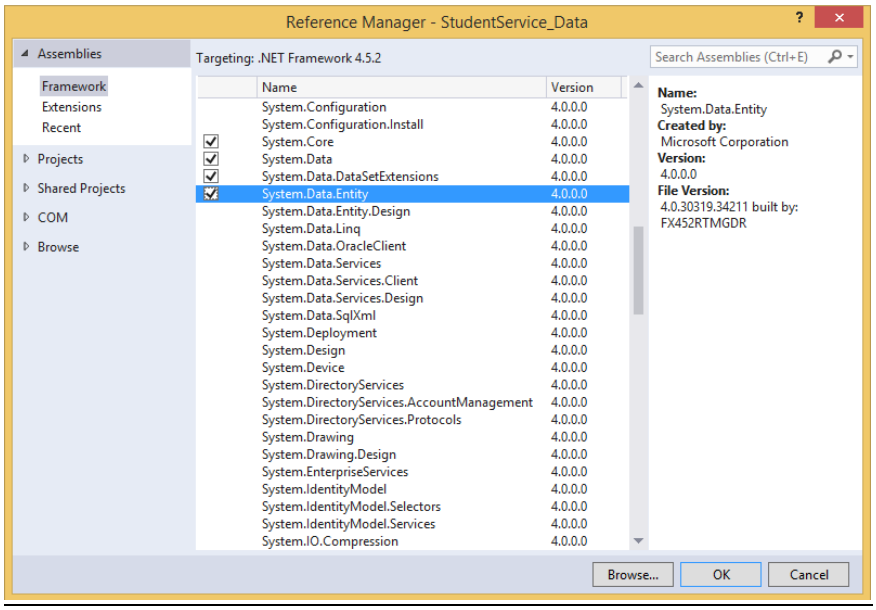
- Kreirati novu biblioteku za rad sa *table storage* servisom. Odabrati *file* meni, potom *Add/New Project*.

- U *Add New Project* dijalogu, proširiti Visual C# pa potom odabrati *Windows* kategoriju. Odabrati *ClassLibrary* kao šablon za projekat. Postaviti naziv projekta na *StudentsService_Data*, i birati opciju Solution: Add to solution i kliknuti OK. Videti sliku 5.



Slika 5 Kreiranje biblioteke

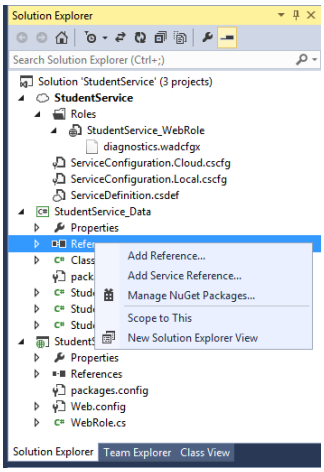
• Dodati referencu *.NET Client Library* za *WCF Data Services* u *StudentsService_Data* projektu. U *Solution Explorer*, desni klik na *StudentsService_Data* projekat, izabrati *Add Reference*, kliknuti na *.NET* tab, selektovati *System.Data.Services.Client* componentu i kliknuti OK. Videti sliku 6.



Slika 6 Dodavanje reference

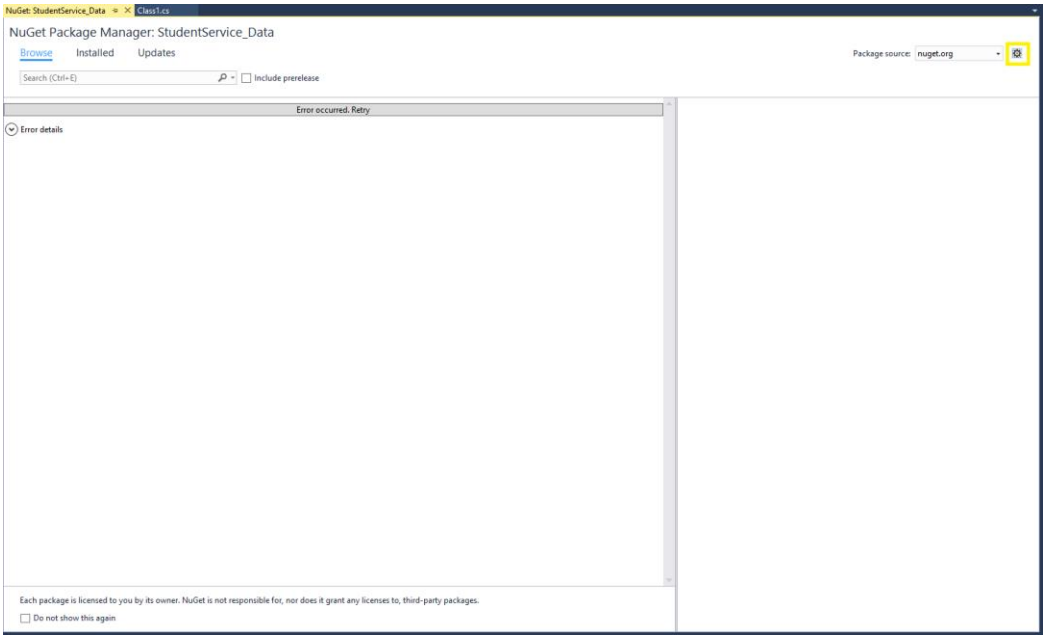
• Treba dodati *Microsoft.WindowsAzure.Storage* referencu na sledeći način (ako ne postoji):

  - Otvoriti "Manage NuGet Packages" prozor, desni klik na References i kliknuti "Manage NuGet Packages" iz kontekst menija.



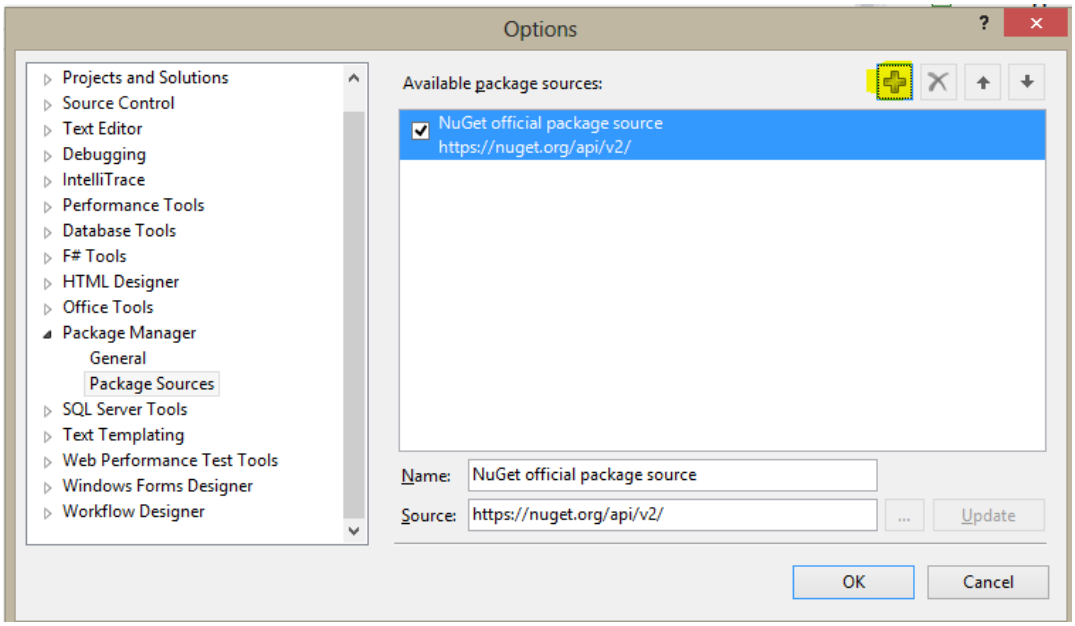
Slika 7 Dodavanje NuGet-a

  - Klik na "Settings"



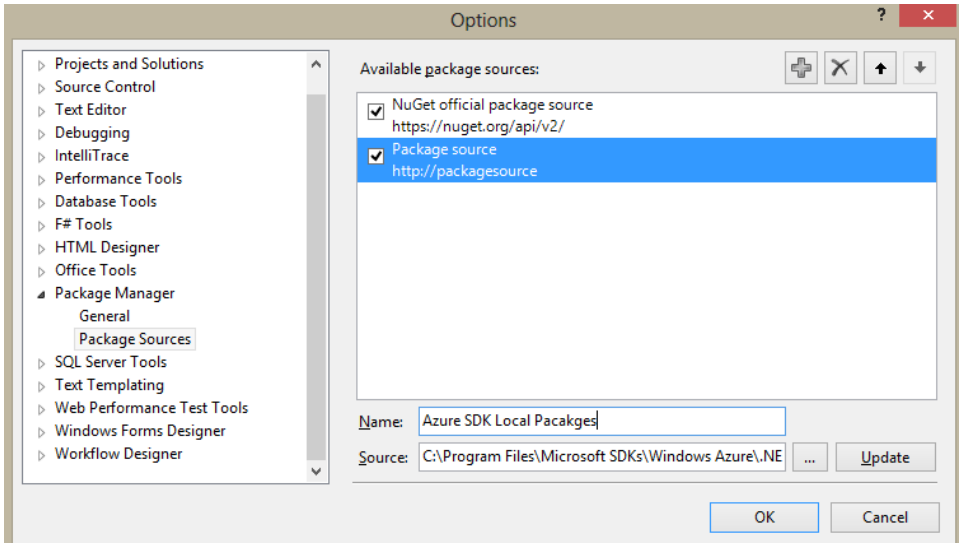
Slika 8 NuGet prozor

- Klik na "+" dugme na vrhu prozora:



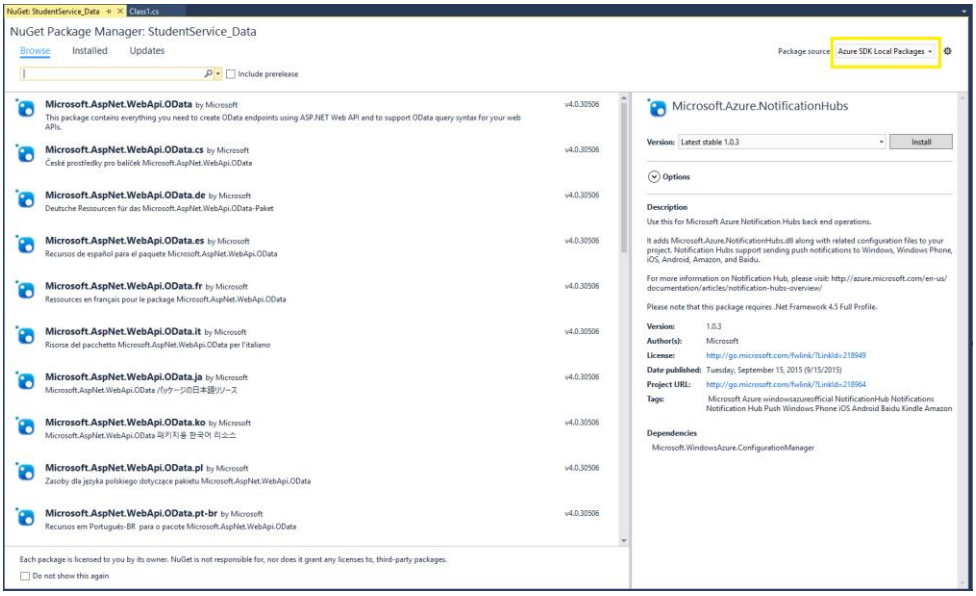
Slika 9 NuGet opcije

- Dodati ime i lokaciju ('**C:\Program Files\Microsoft SDKs\Azure\.NET SDK\v2.8\packages')**



Slika 10 Dodavanje opcija

- Primetićete da je lokalni folder dodat i možete ga izabrati desno u gornjem uglu instalirati package Windows Azure storage



Slika 11 Instaliranje paketa

2.2 Kreiranje klase entiteta

Da bi se entiteti čuvali u tabelama *Table Storage* servisa, prvo se mora izmodelovati entitet studenta.

• Desni klik na *StudentsService_Data* u *SolutionExplorer*, potom *Add/Class*. U *Add New Item* dijalogu, nazvati klasu Student i kliknuti *Add*.

• Prvo klasu Student treba modifikovati tako da bude *public* i da nasleđuje *TableEntity*. *Microsoft.WindowsAzure.Storage.Table namespace* se mora uključiti da bi *TableEntity* klasa bila vidljiva.

• Napraviti konstruktor klase koji će primati broj indeksa kao jedini parametar. U konstruktoru inicijalizovati *PartitionKey* literalom Student, a *RowKey* brojem indeksa.

• Dodati polja za *url* do slike, do *thumbnail* slike kao i za ime i prezime. Sačuvati klasu Student. U listingu 1 je dat konačan izgled klase Student.

Listing 1 – Student.cs

```
using System;
using Microsoft.WindowsAzure.Storage.Table;

namespace StudentService_Data
{
    public class Student : TableEntity
    {
        public String Name { get; set; }
        public String LastName { get; set; }
        public String PhotoUrl { get; set; }
        public String ThumbnailUrl { get; set; }

        public Student(String indexNo)
        {
            PartitionKey = "Student";
            RowKey = indexNo;
        }

       public Student() { }
    }
}
```

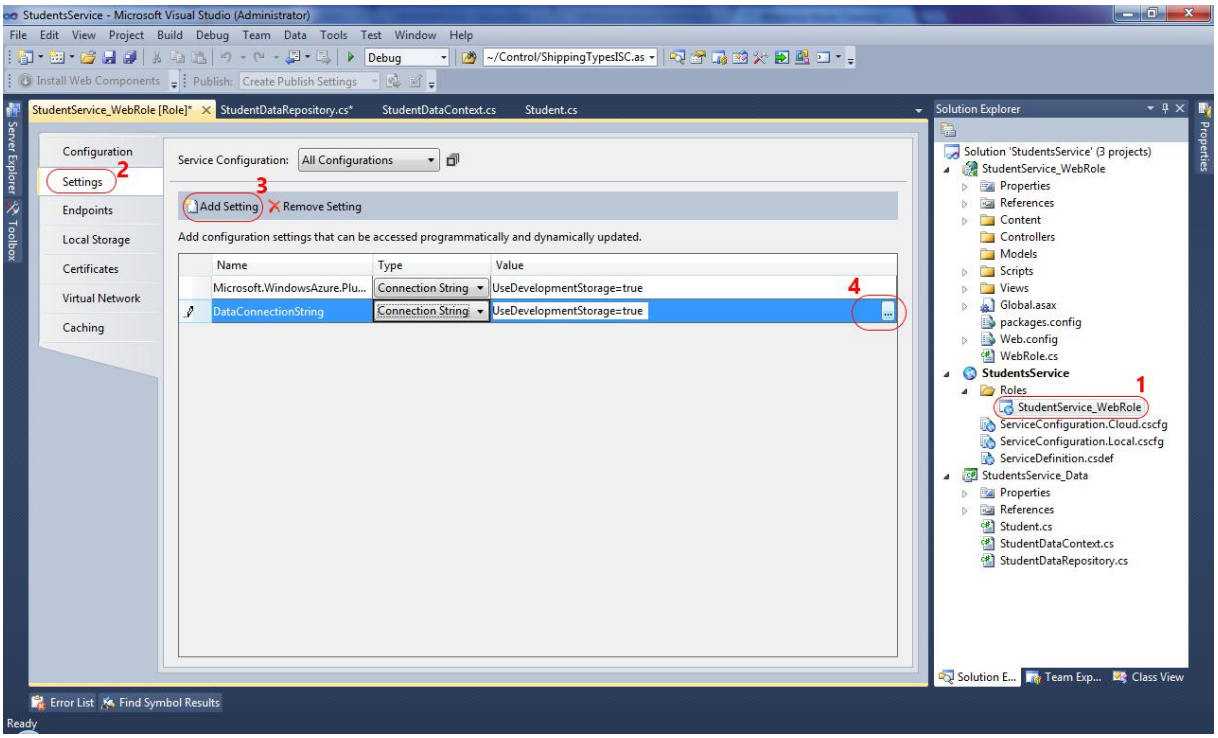
2.3 Kreiranje konteksta podataka

Potrebno je dodati kontekstnu klasu koja služi za pristup „*StudentTable*“ tabeli, manipulaciju entietima tako što izvršava čitanje, upis i modifikaciju entiteta. Komunikacija sa *Table Storage* servisom se posredno odvija koristeći *WCF Data Services*.

• Potrebno je u *StudentsService_Data* projekat dodati klasu koja se bavi manipulacijom entiteta koristeći *DataContext*. Nazvati novu klasu *StudentDataRepository*.

• Modifikovati *Repository* klasu da bude *public*. Potom dodati dva polja s modifikatorima *private* za *CloudTable* i za *CloudStorageAccount*.

• *CloudStorageAccount* klasa se odnosi na podatke vezane za *WindowsAzure* nalog kao što su kredencijali za pristup servisima, postavljanje konfiguracije i metode za kreiranje *storage* servisa. Instalirani *SDK* nudi emulaciju *Storage* servisa, tako da je potrebno konfigurisati servis da koristi lokalni *storage* nalog umesto servisa koji se dobijaju uz pretplatu.



Slika 12 Konfiguracija za konekciju na Storage servis

• Da bi se konfigurisao *ConnectionString* kao na slici 12, potrebno je prvo otvoriti konfiguraciju za *web role* procese. Konfiguracija se nalazi u okviru *cloud* projekta, u direktorijumu *Roles*. Dvoklikom na *StudentsService_WebRole* se otvara konfiguracija. Kliknuti na *Settings*, i u ovkiru ovog prozora kliknuti na *AddSettings*. Pod *Name* kolonom upisati *DataConnectionString*, i selektovati *Type ConnectionString*. U okviru *Windows Azure web* portala, može se pronaći *connection string* namenjen pristupu servisima i prekopirati. U ovoj vežbi će se koristiti emulirani *storage* servis. Kliknuti na dugme sa tri tačke i otkačiti opciju *Windows Azure storage emulator*. Kliknuti na OK i u polju *Value* bi trebalo da se pojavi *UseDevelopmentStorage=true*.

• Kada se *ConnectionString* koji će koristiti emulirani *storage* konfiguriše, potrebno je refencirati se na emulirani servis. Dodati podrazumevani konstruktor u klasi *StudentDataRepository*. U okviru podrazumevanog konstruktora, inicijalizovati *StudentDataContext* koristeći *CloudStorageAccount* i kreirati sve neophodne tabele. *CloudStorageAccount* treba da koristi emulirano okruženje što je određeno konfiguracijom.

• U *Repository* klasi, neophodno je još dodati metode koje čine smisao same *Repository* klase. U ovom slučaju, to su metode za listanje svih studenata i za dodavanje studenta. Kasnije će

biti potrebe za proširenjem funkcionalnosti kao što je dodavanje metode za postavljanje novog linka ka *thumbnail* slici i slično. Implementirana *Repository* klasa data je u listingu 4.

• Dodati referencu na *Microsoft.WindowsAzure.Configuration* (ako već nije dodata) kako bi se mogao uključiti *Microsoft.WindowsAzure namespace* da bi klasa *CloudConfigurationManager* bila vidljiva. *CloudConfigurationManager* klasa omogućava čitanje vrednosti *XML* konfiguracionih datoteka.

Listing 4 – StudentRepository class

```
using Microsoft.WindowsAzure;
using Microsoft.WindowsAzure.Storage;
using Microsoft.WindowsAzure.Storage.Table;
using System;
using System.Linq;

namespace StudentService_Data
{
    public class StudentDataRepository
    {
        private CloudStorageAccount _storageAccount;
              private CloudTable _table;

        public StudentDataRepository()
        {
            _storageAccount =
CloudStorageAccount.Parse(CloudConfigurationManager.GetSetting("DataConnectionString"));
            CloudTableClient tableClient = new CloudTableClient(new
Uri(_storageAccount.TableEndpoint.AbsoluteUri), _storageAccount.Credentials);
            _table = tableClient.GetTableReference("StudentTable");
            _table.CreateIfNotExists();
        }

        public IQueryable<Student> RetrieveAllStudents()
        {
                    var results = from g in _table.CreateQuery<Student>()
                                  where g.PartitionKey == "Student"
                                  select g;
            return results;
        }

        public void AddStudent(Student newStudent)
        {
            // Samostalni rad: izmestiti tableName u konfiguraciju servisa.
                    TableOperation insertOperation = TableOperation.Insert(newStudent);
                    _table.Execute(insertOperation);
        }
    }
}
```

• U konstruktoru *StudentDataRepository* klase, vrši se inicijalizacija objekata *CloudStorageAccount* i *CloudTable*. *CloudStorageAccount* se inicijalizuje koristeći konfigurabilni *ConnectionString*. Na osnovu objekta *CloudStorageAccount*, inicijalizuje se *CloudTable*, koja će koristiti prosleđeni *storage* nalog.

• *RetrieveAllStudents* metoda preuzima sve studente iz tabele "*Student*". Koristi se *LINQ* upit za preuzimanje podataka.

• *AddStudent* metoda dodaje novog studenta u tabelu. Koristi se *TableOperation* i metoda *Insert*.

U ovom delu vežbi je napravljena biblioteka koja obezbeđuje rad sa *Table storage* servisom. Potrebno je dizajnirati *web role* prezentaciju u kojoj će postojati mogućnost pregleda studenata u tabelarnoj vizuelizaciji i koja će omogućiti dodavanje novih studenata.

3 Modifikacija *web role* za prikaz spiska studenata i dodavanje novih studenta.

U ovom delu vežbe 3, potrebno je napraviti dve *web* stranice u okviru postojeće *web role*. Početna stranica prikazuje sve sačuvane studente, dok druga stranica služi za dodavanje novog studenta. Prvo treba napraviti *Controller* koji prima *HTTP* zahteve i sprovodi poslovnu logiku *web* aplikacije.

• Napraviti *StudentController* klasu. U okviru *StudentService_WebRole* pozicionirati se na direktorijum *Controllers* i desnim klikom odabrati *Add/Controller*... U *Controller name* upisati *StudentController* i pod *Template* padajućim menijem odabrati *MVC 5 Empty Controller*. Kliknuti *Add*.

• Da bi u *web role* procesima moglo da se pristupa Student klasi, neophodno je referencirati biblioteku *StudentService_Data* iz *StudentService_WebRole*. Desni klik na *References* u *StudentService_WebRole* projektu, potom na *Add reference…* Kliknuti na tab *projects* u prozoru za dodavanje referenci. Potom kliknuti OK. Uraditi *Build* celog projekta.

• Potrebno je dodati *StudentDataRepository* polje u okviru klase *StudentController* kako bi se funkcionalnosti *StudentDataRepository* klase mogle koristiti u svim metodama kontrolera.

Listing 5 – StudentController.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;
using StudentsService_Data;
using Microsoft.WindowsAzure.StorageClient;
using Microsoft.WindowsAzure;
using System.Net;

namespace StudentService_WebRole.Controllers
{
    public class StudentController : Controller
    {
        StudentDataRepository repo = new StudentDataRepository();

        public ActionResult Index()
        {
```

```
            return View();
        }
    }
}
```

- Kako će se slike čuvati u okviru *BLOB* kontejnera, potrebno je dodati metodu za inicijalizaciju *BLOB* servisa koja bi se koristila pri pokretanju *WindowsAzure* servisa.

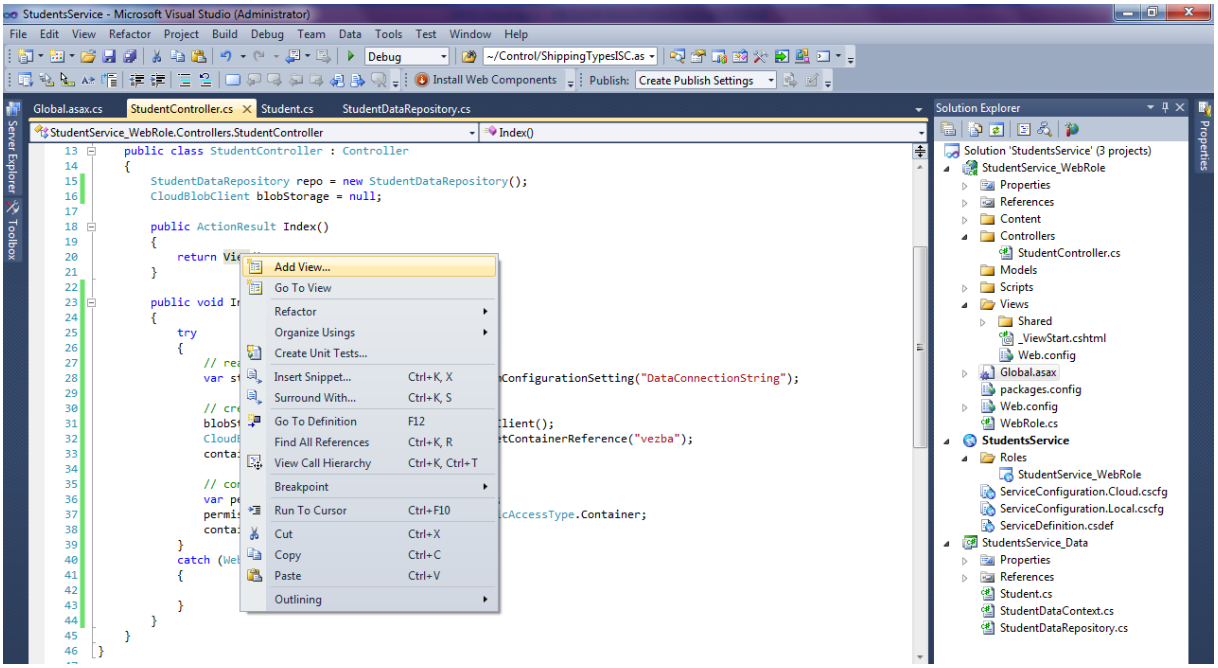
InitBlob metoda prvo iščitava *ConnectionString* koji je napravljen u prethodnom delu vežbe, što će omogućiti konektovanje *blob* klijenta na lokalni *storage* account. Potom, kreira se kontejner pod nazivom *„vezba“* ukoliko već ne postoji. Nakon kreiranja kontejnera, potrebno je dodeliti mu permisije tako da predstavlja javno dostupni kontejner. U okviru projekta *StudentService_WebRole*, dvokliknuti na *Global.asax* i pozvati inicijalizaciju *BLOB* servisa u okviru metode *Application_Start()* kao u listingu 6.

- Da bi se koristile klase iz *Windows Azure* biblioteka, potrebno je referencirati ih isto kao što je to učinjeno u biblioteci *StudentsService_Data*. Dodati *Microsoft.WindowsAzure.Storage* i *Microsoft.WindowsAzure.Configuration* reference (ako već nisu dodate). Nakon što su dodate reference, potrebno je uključiti odgovarajuće *namespace* u *Global.asax*.

Listing 6 – *Application_Start* metoda u okviru *Global.asax web role* servisa

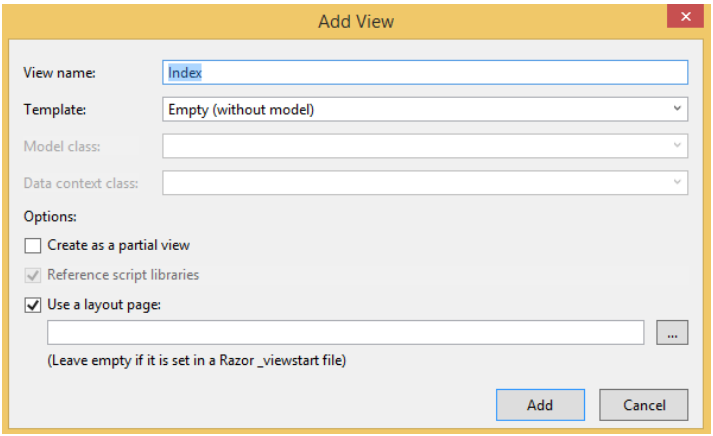
```
protected void Application_Start()
{
        AreaRegistration.RegisterAllAreas();
        InitBlobs();
        FilterConfig.RegisterGlobalFilters(GlobalFilters.Filters);
        RouteConfig.RegisterRoutes(RouteTable.Routes);
        BundleConfig.RegisterBundles(BundleTable.Bundles);
}

public void InitBlobs()
{
        try
        {
                // read account configuration settings
                var storageAccount =
        CloudStorageAccount.Parse(CloudConfigurationManager.GetSetting("DataConnectionStri
        ng"));

                // create blob container for images
                CloudBlobClient blobStorage = storageAccount.CreateCloudBlobClient();
                CloudBlobContainer container = blobStorage.GetContainerReference("vezba");
                container.CreateIfNotExists();

                // configure container for public access
                var permissions = container.GetPermissions();
                permissions.PublicAccess = BlobContainerPublicAccessType.Container;
                container.SetPermissions(permissions);
        }
        catch (WebException)
        {

        }
}
```

• Potrebno je dodati *view* koji će izlistati sve studente. U okviru Index() akcije u *StudentController* klasi, kliknuti desnim klikom na *return View()*, i potom kliknuti *Add View*… kao što je prikazano na slici 13.

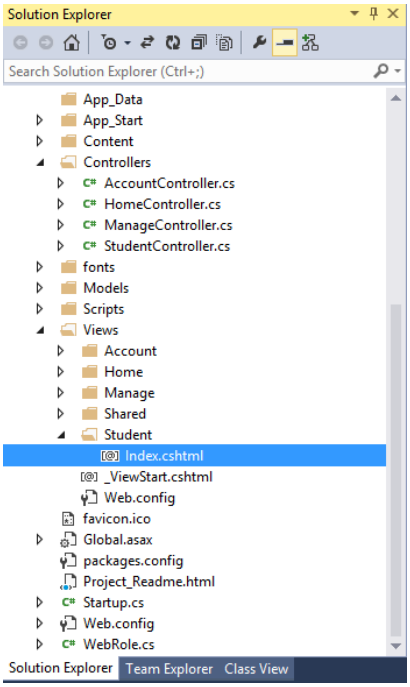


Slika 13 Dodavanje View za Index akciju.

U okviru *Add View* dijaloga, koji je prikazan na slici 15, nazvati *view* kao Index. Kliknuti *Add*. Pozicija *view*-a u okviru *StudentService_WebRole* projekta je prikazana na slici 14.



Slika 14 Kreiranje tipiziranog view-a



Slika 15 Pozicija view-a

- Da bi *MVC3 web role* servis prikazao listu studenata, klasu Student je neophodno proširiti sa podrazumevanim konstruktorom. Podrazumevani konstruktor ostaviti prazan: `public Student(){ }` Uraditi build projekta.
- Da bi prikaz studenata bio u skladu s potrebama *StudentsService* aplikacije, potrebno je izmeniti automatski izgenerisani *cshtml view* koji je generisan u prethodnim koracima. U okviru direktorijuma *Views/Student*, otvoriti Index.cshtml i potpuno izbrisati sadržaj ovog fajla. Potom, prekopirati sadržaj listinga 7. U odnosu na generisani sadržaj, izbačene su nepotrebne kolone u prikazu detalja studenta, i dodato je iscrtavanje slike studenta umesto prikaza *URL* informacije gde se slika nalazi.

Listing 7 – Student/Index.cshtml

```
@model IEnumerable<StudentService_Data.Student>
@{
    ViewBag.Title = "Index";
}
<h2>
    Index</h2>
<p>
    @Html.ActionLink("Create New", "Create")
</p>
<table>
    <tr>
        <th>
            Index
        </th>
        <th>
            Name
        </th>
        <th>
            LastName
        </th>
        <th>
            Photo
        </th>
        <th>
        </th>
    </tr>
    @foreach (var item in Model)
    {
        <tr>
            <td>
                @Html.DisplayFor(modelItem => item.RowKey)
            </td>
            <td>
                @Html.DisplayFor(modelItem => item.Name)
            </td>
            <td>
                @Html.DisplayFor(modelItem => item.LastName)
            </td>
            <td>
                <img src="@item.ThumbnailUrl" />
            </td>
            <td>
```

```
                @Html.ActionLink("Edit", "Edit", new { /* id=item.PrimaryKey */ })
                @Html.ActionLink("Details", "Details", new { /* id=item.PrimaryKey */ })
                @Html.ActionLink("Delete", "Delete", new { /* id=item.PrimaryKey */ })
            </td>
        </tr>
    }
</table>
```

• Potrebno je podesiti *StudentController* klasu kao podrazumevanu kada se pristupa *StudentsService web* aplikaciji. To se može podesiti izmenom u *Global.asax*, tako što se mapira podrazumevajuća ruta za *web* aplikaciju. Dovoljno je zameniti podrazumevani "*Home*" kontroler sa "*Student*" kontrolerom u okviru *RegisterRoutes* metode kao što je prikazano u listingu 8.

Listing 8 – RegisterRoutes metoda u Global.asax fajlu web role

```
public static void RegisterRoutes(RouteCollection routes)
        {
            routes.IgnoreRoute("{resource}.axd/{*pathInfo}");

            routes.MapRoute(
                "Default", // Route name
                "{controller}/{action}/{id}", // URL with parameters
                new { controller = "Student", action = "Index", id =
UrlParameter.Optional } // Parameter defaults
            );
        }
```

• Da bi se stranica popunila svim studentima koji postoje u *Table storage* servisu, potrebno je proslediti dobijene studente samom pogledu. To se može učiniti putem instance *StudentDataRepository* klase koja već postoji kao polje *StudentController* klase. Index akcija *StudentController* klase nakon poslednje izmene izgleda kao u listingu 9.

Listing 9 – Index akcija StudentController klase

```
public ActionResult Index()
{
    return View(repo.RetrieveAllStudents());
}
```

• Nakon ovog koraka, može se pokrenuti *cloud* projekat, pritisnuti F5 taster. Trebalo bi da se otvori stranica, ali kako još nije unet ni jedan student, lista studenata će biti prazna. Potrebno je napraviti stranicu koja će omogućiti dodavanje novog studenta. Prvobitno, neophodno je dodati akciju *Create* koja će vraćati *view* koji služi za unos podataka novog studenta. Listing 10 prikazuje akciju koju treba dodati u StudentController klasu.

Listing 10 – Create akcija StudentController klase

```
public ActionResult Create()
{
    return View("AddEntity");
}
```

• Potrebno je dodati *view* za kreiranje studenta, na isti način kao što je to pokazano na slici 13, s tim da desni klik treba da bude nad *return View* iskazom Create() akcije. Nakon desnog klika, kliknuti na *Add View*, pojavljuje se dijalog kao na slici 10. Podešavanja treba da budu identična kao na slici 15 osim što za naziv, treba upisati *AddEntity*.

• Nakon sprovodjenja ove procedure, trebalo bi da se u okviru *Views/Student* direktorijuma nalazi *view AddEntity.cshtml*. Kako bi ova stranica bila prilagodjena potrebama StudentsService aplikaciji i bila u mogućnosti da postavlja slike, sadržaj fajla *Views/Student/AddEntity.cshtml* treba da bude identičan kao u listingu 11.

Promena u odnosu na generisani sadržaj podrazumeva izbacivanje polja za unos nepotrebnih informacija, kao što su *PartitionKey*, *Timestamp* i linkovi ka slici. Umesto suvišnih polja, u formu za unos podataka je dodato postavljanje slike kako bi se slika mogla sačuvati u *BLOB* servisu.

Listing 11 – Student/AddEntity.cshtml

```
@model StudentService_Data.Student
@{
    ViewBag.Title = "CreateStudent";
}
<h2>Add Entity View</h2>
<script src="@Url.Content("~/Scripts/jquery.validate.min.js")"
type="text/javascript"></script>
<script src="@Url.Content("~/Scripts/jquery.validate.unobtrusive.min.js")"
type="text/javascript"></script>
@using (Html.BeginForm("AddEntity", "Student", FormMethod.Post, new { enctype =
"multipart/form-data" }))
{
    @Html.ValidationSummary(true)
    <fieldset>
        <legend>Student</legend>
        <div class="editor-label">
            @Html.LabelFor(model => model.RowKey)
        </div>
        <div class="editor-field">
            @Html.EditorFor(model => model.RowKey)
            @Html.ValidationMessageFor(model => model.RowKey)
        </div>
        <div class="editor-label">
            @Html.LabelFor(model => model.Name)
        </div>
        <div class="editor-field">
            @Html.EditorFor(model => model.Name)
            @Html.ValidationMessageFor(model => model.Name)
        </div>
        <div class="editor-label">
            @Html.LabelFor(model => model.LastName)
        </div>
        <div class="editor-field">
            @Html.EditorFor(model => model.LastName)
            @Html.ValidationMessageFor(model => model.LastName)
        </div>
        <input type="file" id="studentPicture" name="file" />
        <p>
            <input type="submit" value="Create" />
```

```
        </p>
    </fieldset>
}
<div>
    @Html.ActionLink("Back to List", "Index")
</div>
```

• *Student/AddEntity.cshtml* sadrži formu koja koristi *HTTP* post metodu za slanje podataka i slike *StudentController* objektu. Na osnovu poslatih podataka, *StudentController* treba da izvrši čuvanje slike u *BLOB storage*, i da sačuva entitet studenta u *table storage*. Nakon uspešnog dodavanja studenta, vrši se redirekcija na listu studenata. U klasu *StudentController* potrebno je dodati akciju iz listinga 12.

Listing 12 – *Create* Akcija za čuvanje slike i studenta u *BLOB* i *table* storage.

```
[HttpPost]
public ActionResult AddEntity(String RowKey, String Name, String LastName,
HttpPostedFileBase file)
{
        try
        {
                // kreiranje blob sadrzaja i kreiranje blob klijenta
                string uniqueBlobName = string.Format("image_{0}", RowKey);
                var storageAccount =
        CloudStorageAccount.Parse(CloudConfigurationManager.GetSetting("DataConnectionStri
        ng"));
                CloudBlobClient blobStorage = storageAccount.CreateCloudBlobClient();
                CloudBlobContainer container = blobStorage.GetContainerReference("vezba");
                CloudBlockBlob blob = container.GetBlockBlobReference(uniqueBlobName);
                blob.Properties.ContentType = file.ContentType;
        // postavljanje odabrane datoteke (slike) u blob servis koristeci blob klijent
                blob.UploadFromStream(file.InputStream);
                // upis studenta u table storage koristeci StudentDataRepository klasu
                Student entry = new Student(RowKey) { Name = Name, LastName = LastName,
        PhotoUrl = blob.Uri.ToString(), ThumbnailUrl = blob.Uri.ToString() };
                repo.AddStudent(entry);
                return RedirectToAction("Index");
        }
        catch
        {
                return View("AddEntity");
        }
}
```

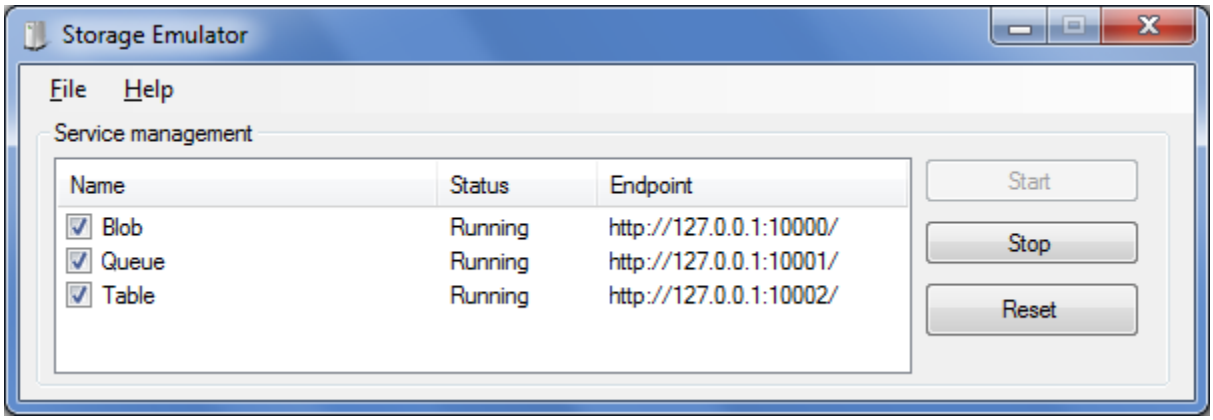
• Nakon ovog koraka, *StudentService* aplikacija ima mogućnost dodavanja studenata i njihovog listanja. U narednom delu vežbe, izneti su detalji korišćenja *Storage* emulatora i *Compute* emulatora.

4 Rad sa *Compute* emulatorom i *Storage* emulatorom

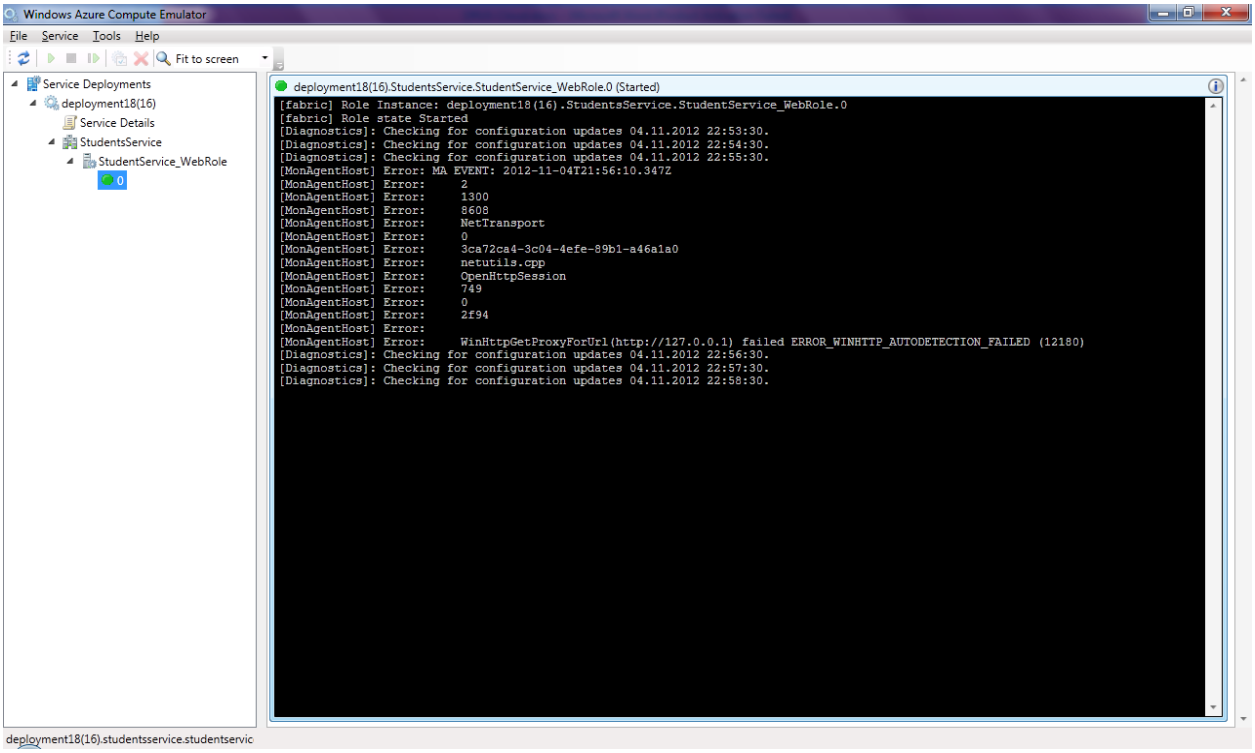
Nakon pokretanja *cloud* projekta, prvo se pokreću *Compute* emulator i *Storage* emulator. U okviru *system tray* (*notification area*) u *Windows OS* okruženju, koji se nalazi pored sata, pojavi se *Microsoft logo*, koji predstavlja *WindowsAzure* emulator. Desnim klikom na logo, dobija se pet opcija: *Shutdown Compute* emulator, *Shutdown Storage* emulator, *Show Compute UI*, *Show Storage UI* i *exit*.

Shutdown opcije će zaustaviti servise emulatora i aplikacije neće moći da se izvršavaju u emuliranom okruženju. Opcija *exit* će uraditi *shutdown* nad obe vrste emulator servisa. Opcije *Show Compute UI* i *Show Storage UI* prikazuju korisnički interfejs za ova dva servisa.

Storage UI je jednostavan interfejs koji je prikazan na slici 16. Interfejs prikazuje statuse servisa, nudi opcije pokretanja, restartovanja i zaustavljanja servisa. Takođe, pokazuje i *Endpoint* adresu na kojoj možemo pristupiti resursima. Na osnovu ovih adresa, moguće je pristupiti resursima i putem *browser*-a.



Slika 16 Storage emulator interfejs

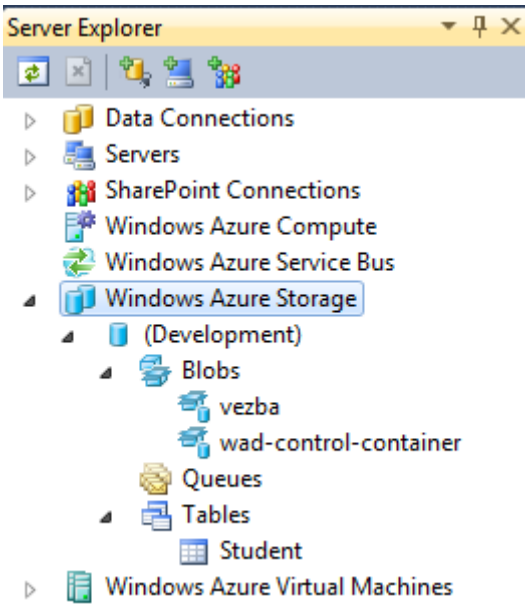


Slika 17 Compute emulator interfejs

Compute UI je češće korišćeni interfejs iz razloga što omogućava praćenje životnog ciklusa *WindowsAzure* servisa koji se izvršava u kontekstu emuliranog *compute* servisa. *Compute UI* je prikazan na slici 17.

Na osnovu stabla u levom delu interfejsa, moguće je izlistati sve zasebne kontekste izvršavanja, i okviru svakog postoje servisi i detalji servisa. U okviru servisa se uvek nalaze sve *web* i *worker* role. Proširenjem određene *role* možemo videti koliko je procesa pokrenuto. Na slici 17 se vidi da postoji samo jedan pokrenut *web role* proces. Klikom na određenu instancu, dobijaju se informacije o izvršavanju određene insance. Ovo je veoma pogodno u fazi rezvoja, jer je praktično omogućen konzolni prikaz za svaku instancu posebno. Uz pomoć *Trace* klase, moguće je ispisivati razne informacije na konzolu što olakšava proces razvijanja aplikacija.

WindowsAzure Storage servisu je moguće pristupiti direktno iz *Visual Studio* razvojnog okruženja. Iz menija *View*, kliknuti na *ServerExplorer* ukoliko već nije otvoren. Kada se otvori Server explorer, u njemu se nalazi *Windows Azure Storage* što je prikazano na slici 18.



Slika 18 Visual Studio Server Explorer

Iz *Visual Studio Server Explorer* komponente, može se vršiti navigacija po sve tri vrste servisa u okviru *Windows Azure Storage* servisa. Dvoklikom na tabele, moguće je izlistati sve entitete. Dvoklikom na *BLOB* kontejner, moguće je izlistati sve binarne objekte sadržane u okviru određenog kontejnera.

Samostalni rad:

1. Proširiti aplikaciju tako da se za studente čuvaju podaci JMBG i državljanstvo. Sačuvati novi entitet studenta. Potom pregledati tabelu Student.
2. Implementirati metodu za proveru da li student već postoji u tabeli. U okviru *StudentsRepository* klase, dodati metodu `bool Exists(string indexNo)`. Iskoristiti metodu za proveru pre dodavanja novog studenta. Ukoliko student već postoji, vratiti korisniku informaciju o grešci (kreirati novi *view* za ovu svrhu).
3. Implementirati akciju brisanja pojedinačnog studenta iz liste studenata klikom na link *delete*.
4. Kreirati entitet Nastavnik i čuvati entitete u tabeli Nastavnici. Entitet nastavnik je određen šifrom nastavnika, a potrebno je čuvati ime, prezime i predmet koji predaje.