Dokumentacija projekta iz predmeta:

Cloud Computing u elektroenergetskim sistemima

1.Traženi listingi glavnih delova implementacije čuvanja primarne instance:

Proces čuvanja primarnih entiteta kreće iz samog kontrolera projekta web role kada se poziva writer instanca da bi proverila da li već postoji isti entitet, i zatim ako isti entitet ne postoji zahtev se prosleđuje prvoj ili trećoj instanci writera radi čuvanja.

Prvi listing predstavlja implementaciju Create metode u okviru Agencija kontrolera za primarne entitete.

```
public ActionResult Create(FormCollection collection)
        {
            //prvo provera postojanja preko readera,ide input radi azure load balancera
            NetTcpBinding readerbind = new NetTcpBinding();
            ChannelFactory<IReader> factory = new ChannelFactory<IReader>(readerbind, new
           EndpointAddress("net.tcp://localhost:10101/InputRequest"));
            IReader proxyreader = factory.CreateChannel();
            if (proxyreader.proveriDaLiPostojiDodavanje(collection["ime"],
collection["adresa"], collection["drzava"], collection["grad"]))//ako postoji sa istim
podacima ne moze se dodati
            {
                ViewBag.Error = "Doslo je do greske! Postoji vec isti entitet.";
                return View();
            }
            else//sve je uredu
            {

                NetTcpBinding binding = new NetTcpBinding();
                string internalEndpointName = "InternalRequest";
                List<RoleInstance> bratske_instance =
RoleEnvironment.Roles["WriterWorkerRole"].Instances.ToList(); //uzmemo sve writer
instance
                try
                {
                    // Konektuje se na prvi ili treci writer po principu ostatka pri
deljenju dvojkom
                    if (cnt % 2 == 0)//prva instanca
                    {
                        RoleInstanceEndpoint pristupna =
bratske_instance[0].InstanceEndpoints[internalEndpointName];
                        EndpointAddress endpoint = new
EndpointAddress(String.Format("net.tcp://{0}/{1}", pristupna.IPEndpoint.ToString(),
internalEndpointName));
```

```
                        IWriter proxy = new ChannelFactory<IWriter>(binding,
endpoint).CreateChannel();
                        //saljemo zahtev i dobijamo rezultat uspesnosti
                        bool zahtev = proxy.PrihvatiZahtevCreate(collection["ime"],
collection["adresa"], collection["drzava"], collection["grad"],IDGenerator.idAgencija);
                        IDGenerator.idAgencija++;
                        cnt++;
                    }
                    else//treca instanca
                    {
                        RoleInstanceEndpoint pristupna =
bratske_instance[2].InstanceEndpoints[internalEndpointName];
                        EndpointAddress endpoint = new
EndpointAddress(String.Format("net.tcp://{0}/{1}", pristupna.IPEndpoint.ToString(),
internalEndpointName));
                        IWriter proxy = new ChannelFactory<IWriter>(binding,
endpoint).CreateChannel();
                        bool zahtev = proxy.PrihvatiZahtevCreate(collection["ime"],
collection["adresa"], collection["drzava"], collection["grad"],IDGenerator.idAgencija);
                        IDGenerator.idAgencija++;
                        cnt++;
                    }




                    return RedirectToAction("Index");
                }
                catch(Exception e)
                {
                    throw e;
                }
            }
        }
```

Zatim sledi listing same metode writer worker role koja vrši upis podata u azure table storage.

```
public bool PrihvatiZahtevCreate(string ime, string adresa, string drzava, string
grad,int id)
        {
            Agencija temp = new Agencija(ime, adresa, grad, drzava, id);
            AgencijaProject_Data.AgencijaDataRepository repo = new
AgencijaProject_Data.AgencijaDataRepository();
            repo.AddAgencie(temp);
            LogujDogadjaj("[" + DateTime.Now.ToString() + "]:" + String.Format("Izvrseno
kreiranje novog entiteta Agencija sa sledecim podacima. Ime: {0}; Adresa:{1}; Drzava:{2};
Grad:{3}", ime, adresa, drzava, grad));
            return true;
        }
```

Kao sto vidimo metoda PrihvatiZahtevCreate kreira novi objekat agencije i zatim poziva metodu Add u repozitorijumu

koji služi za kontrolu nad azure tabelom. Sledi listing metode AddAgencie.

```
public void AddAgencie(Agencija newAgencie)
        {
            TableOperation insertOperation = TableOperation.Insert(newAgencie);
            _table.Execute(insertOperation);
        }
```

2.Traženi listing implementacije komunikacije Write instanci i Logger-a

Write instance prosleđuju logger instancama preko queue-a poruke o dešavanju u sistemu.

U svakoj metodi writer worker role koja menja, dodaje ili briše entitet na kraju njenog izvšavanja poziva se takozvana LogujDogadjaj(string logMessage) funkcija. Ta funkcija služi da doda poruku u red, sledi njen listing.

```
void LogujDogadjaj(string logMessage)
        {
            CloudQueue logQueue =
AgencijaProject_Data.QueueHelper.GetQueueReference("logqueue");

            logQueue.AddMessage(new CloudQueueMessage(logMessage));
        }
```

Navešćemo listing metode za kreiranje novih Agencija da bi pokazali primer log poruke.

```
public bool PrihvatiZahtevCreate(string ime, string adresa, string drzava, string
grad,int id)
        {
            Agencija temp = new Agencija(ime, adresa, grad, drzava, id);
            AgencijaProject_Data.AgencijaDataRepository repo = new
AgencijaProject_Data.AgencijaDataRepository();
            repo.AddAgencie(temp);
            LogujDogadjaj("[" + DateTime.Now.ToString() + "]:" + String.Format("Izvrseno
kreiranje novog entiteta Agencija sa sledecim podacima. Ime: {0}; Adresa:{1}; Drzava:{2};
Grad:{3}", ime, adresa, drzava, grad));
            return true;
        }
```

Dalji tok komunikacije odvija se na način da sam logger proverava na svakih 1 sekundu da li postoje poruke u redu tj. novi log događaji i ako postoji preuzima poruku iz reda, čuva je u blob storage i zatim uklanja iz reda.

Sledi listing samog koda zaduženog za preuzimanje iz reda i čuvanje u blob. (Run metoda u logger worker roli)

```
public override void Run()
        {
            Trace.TraceInformation("Logger is running");
            CloudQueue queue = QueueHelper.GetQueueReference("logqueue");

            while (true)
            {
                CloudQueueMessage message = queue.GetMessage();
                if (message != null)
                {
                    sacuvajUBlob(message);
                    queue.DeleteMessage(message);
                }
                Thread.Sleep(1000);
            }

        }
```

Dok funkcija sacuvajUBlob ima sledeću implementaciju.

```
void sacuvajUBlob(CloudQueueMessage message)
        {
            var storageAccount =
CloudStorageAccount.Parse(CloudConfigurationManager.GetSetting("DataConnectionString"));
            CloudBlobClient blobStorage = storageAccount.CreateCloudBlobClient();
            CloudBlobContainer container =
            blobStorage.GetContainerReference("loggerblob");
            CloudBlockBlob blob = container.GetBlockBlobReference("log");
            string content = "";
            if (blob.Exists())
            {
                content = blob.DownloadText() + "|";
            }
            content = content + message.AsString;
            blob.UploadText(content);

        }
```

Log događaji su međusobno razdvojeni uspravnom crtom.

Dalje je traženo da imamo konzolnu aplikaciju koja ispisuje sve događaje vezane za writer komponentu.

Konzolna aplikacija komunicira sa logger worker rolom putem input komunikacije jer je u specifikaciji zadatka traženo da azure load balancer brine o raspodeli posla. Ona proziva na svakih sekund logger worker rolu i traži da joj isporuči informacije ako je bilo novih događaja u sistemu.

Slede listinzi konzolne aplikacije.

```
static void Main(string[] args)
        {
            LoggerPooling();
        }

        static void LoggerPooling()
        {
            var binding = new NetTcpBinding();
            ChannelFactory<ILogger> factory = new ChannelFactory<ILogger>(binding, new
EndpointAddress("net.tcp://localhost:10100/InputRequest"));
            while (true)
            {
                ILogger proxy = factory.CreateChannel();
                string logovi = proxy.IzvuciLogoveIzBloba();

                if (logovi != "" && logovi != null)
                {
                    string[] redovi = logovi.Split('|');
                    foreach(string red in redovi)
                    {
                        Console.WriteLine(red);
                    }
                }
                Thread.Sleep(1000);
            }


        }
```

Kao što vidimo nakon prijema novonastalih logova oni se parsiraju i ispisiju na konzolu.

Sama metoda implementirana u logger worker roli „IzvuciLogoveIzBloba“ izgleda ovako.

```
public string IzvuciLogoveIzBloba()
        {
            var storageAccount =
CloudStorageAccount.Parse(CloudConfigurationManager.GetSetting("DataConnectionString"));
            CloudBlobClient blobStorage = storageAccount.CreateCloudBlobClient();
            CloudBlobContainer container =
            blobStorage.GetContainerReference("loggerblob");
            CloudBlockBlob blob = container.GetBlockBlobReference("log");
            string content = "";
            if (blob.Exists())
            {
                content = blob.DownloadText(); //preuzmemo celu log istoriju
            }
            string novi_logovi = "";
            novi_logovi = content.Split('*')[content.Split('*').Count() - 1]; // uzmemo
samo sve iza poslednje zvezdice jer to su jedini novi podaci
            if (novi_logovi != "")
            {
                content = content + "*";//obelezimo da smo sve do zvezdice poslali
```

```
            }
            blob.UploadText(content);
            return novi_logovi;
        }
```

Ova metoda se konektuje na blob i čita ceo log fajl zatim uzima samo ono iza zadnje zvezdice. To je odrađeno zbog zahteva u zadatku da logger worker rola isporučuje samo one logove o događajima koje ranije nije isporučio. Tako da ako ima novonastalih događaja on isporučuje informacije o njima konzolnoj aplikaciji „LoggerClientConsole“ i stavlja zvezdicu na kraj da bi obeležio da je to već poslao i uploaduje na blob ponovo ceo content.

3.Navesti nazive resursa u Cloud-u (nazive tabela, kontejnera i sl.), parametri u okviru Queue-a

Ime tabele je “AgencijaProjectTable”.

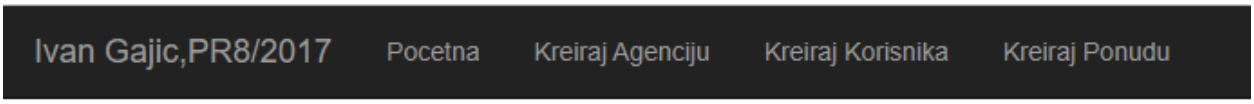
Ime queue je “logqueue”.

Ime bloba je “loggerblob”, ime bloka je “log”.

4. Opisati nekoliko test slučajeva koji se mogu izvršiti kako bi se svi aspekti sistema pokrili i testirali korak po korak npr jedan test za primarne instance:

-Primer za kreiranje primarnog entiteta tj. agencije:

Kreiranje započinje klikom na “Kreiraj agenciju” na web stranici aplikacije.



Zatim sledi popunjavanje input forme sa neophodnim podacima I klik na button “Create”.

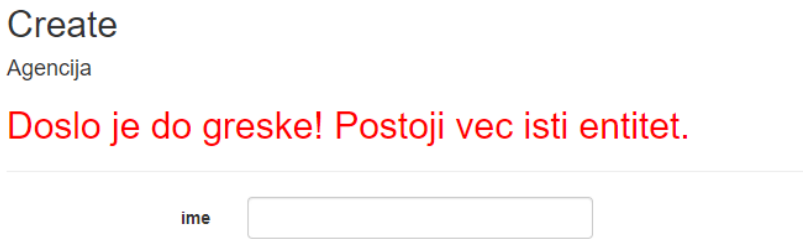
Nakon toga se poziva metoda Create u Agencija kontroleru I prosleđuju joj se vrednosti unete u formu.

Sledi njen listing radi lakšeg objašnjenja.

```
[HttpPost]
        public ActionResult Create(FormCollection collection)
        {
            //prvo provera postojanja preko readera,ide input radi azure load balancera
            NetTcpBinding readerbind = new NetTcpBinding();
            ChannelFactory<IReader> factory = new ChannelFactory<IReader>(readerbind, new
           EndpointAddress("net.tcp://localhost:10101/InputRequest"));
            IReader proxyreader = factory.CreateChannel();
            if (proxyreader.proveriDaLiPostojiDodavanje(collection["ime"],
collection["adresa"], collection["drzava"], collection["grad"]))//ako postoji sa istim
podacima ne moze se dodati
            {
                ViewBag.Error = "Doslo je do greske! Postoji vec isti entitet.";
                return View();
            }
            else//sve je uredu
            {

                NetTcpBinding binding = new NetTcpBinding();
                string internalEndpointName = "InternalRequest";
                List<RoleInstance> bratske_instance =
RoleEnvironment.Roles["WriterWorkerRole"].Instances.ToList(); //uzmemo sve writer
instance
                try
                {
                    // Konektuje se na prvi ili treci writer po principu ostatka pri
deljenju dvojkom
                    if (cnt % 2 == 0)//prva instanca
                    {
                        RoleInstanceEndpoint pristupna =
bratske_instance[0].InstanceEndpoints[internalEndpointName];
                        EndpointAddress endpoint = new
EndpointAddress(String.Format("net.tcp://{0}/{1}", pristupna.IPEndpoint.ToString(),
internalEndpointName));
                        IWriter proxy = new ChannelFactory<IWriter>(binding,
endpoint).CreateChannel();
                        //saljemo zahtev i dobijamo rezultat uspesnosti
                        bool zahtev = proxy.PrihvatiZahtevCreate(collection["ime"],
collection["adresa"], collection["drzava"], collection["grad"],IDGenerator.idAgencija);
                        IDGenerator.idAgencija++;
                        cnt++;
                    }
                    else//treca instanca
                    {
                        RoleInstanceEndpoint pristupna =
bratske_instance[2].InstanceEndpoints[internalEndpointName];
                        EndpointAddress endpoint = new
EndpointAddress(String.Format("net.tcp://{0}/{1}", pristupna.IPEndpoint.ToString(),
internalEndpointName));
                        IWriter proxy = new ChannelFactory<IWriter>(binding,
endpoint).CreateChannel();
                        bool zahtev = proxy.PrihvatiZahtevCreate(collection["ime"],
collection["adresa"], collection["drzava"], collection["grad"],IDGenerator.idAgencija);
                        IDGenerator.idAgencija++;
                        cnt++;
                    }
```

```
                return RedirectToAction("Index");
            }
            catch(Exception e)
            {
                throw e;
            }
        }
    }
```

Kao što vidimo na početku se konektuje na Reader preko input komunikacije. Koristi se input vid komunikacije radi aktivacije azure load balancer-a traženog u specifikaciji zadatka. Reader proverava da li postoji već isti entitet i ako postoji vraća grešku i korisniku se prikazuje ponovo stranica za kreiranje entiteta i ispisuje poruka o grešci.



U drugom slučaju ako je sve uredu i entitet već ne postoji konektuje se na prvu ili treću Writer instancu po principu deljenja brojača pristiglih zahteva sa dvojkom tako da će se u krug smenjivati prva i treća instanca Reader worker role. Ovde je korišćena internal komunikacija jer želimo mi da odlučujemo koja će instanca primiti zahtev bez mešanja azure load balancer-a.

Writer instanci se prosleđuju potrebni podaci za novi entitet i liniji

```
bool zahtev = proxy.PrihvatiZahtevCreate(collection["ime"], collection["adresa"],
collection["drzava"], collection["grad"],IDGenerator.idAgencija);
```

Sama metoda u writeru kreira nov objekat agencije i poziva AddAgencie metodu iz data repozitorijuma koji dalje brine o čuvanju entiteta u tabelu. Slede ta dva listinga.

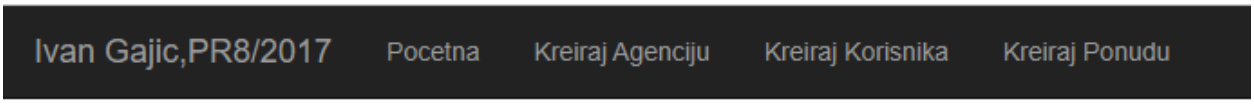
```
public bool PrihvatiZahtevCreate(string ime, string adresa, string drzava, string
grad,int id)
```

```
        {
            Agencija temp = new Agencija(ime, adresa, grad, drzava, id);
            AgencijaProject_Data.AgencijaDataRepository repo = new
AgencijaProject_Data.AgencijaDataRepository();
            repo.AddAgencie(temp);
            LogujDogadjaj("[" + DateTime.Now.ToString() + "]:" + String.Format("Izvrseno
kreiranje novog entiteta Agencija sa sledecim podacima. Ime: {0}; Adresa:{1}; Drzava:{2};
Grad:{3}", ime, adresa, drzava, grad));
            return true;
        }

public void AddAgencie(Agencija newAgencie)
        {
            TableOperation insertOperation = TableOperation.Insert(newAgencie);
            _table.Execute(insertOperation);
        }
```

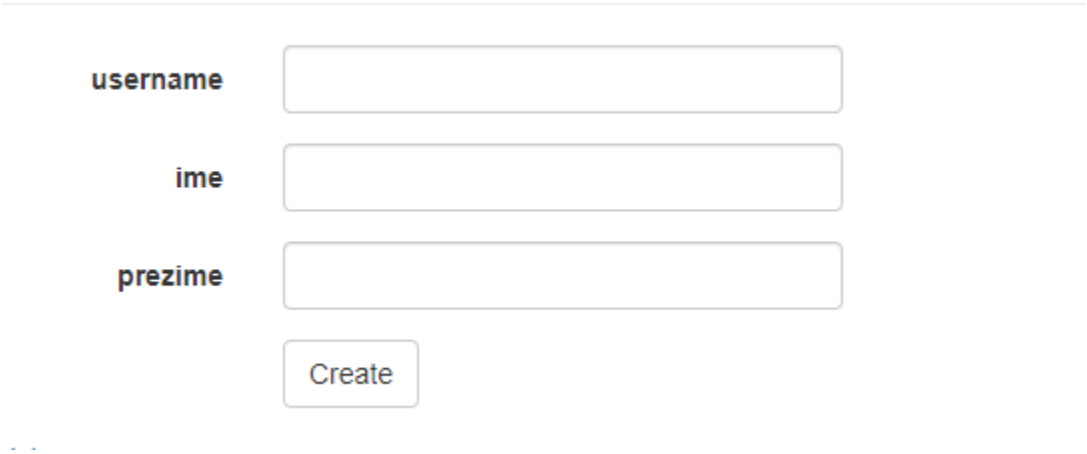
Time je završeno kreiranje primarnog entiteta tj. agencije.

-Primer za kreiranje sekundarnog entiteta tj. korisnika:

Kreiranje započinje klikom na “Kreiraj Korisnika” na web stranici aplikacije.



Zatim sledi popunjavanje input forme sa neophodnim podacima I klik na button “Create”.



Nakon toga se poziva metoda Create u Korisnik kontroleru I prosleđuju joj se vrednosti unete u formu.

Sledi njen listing radi lakšeg objašnjenja.

```
[HttpPost]
        public ActionResult Create(FormCollection collection)
        {
            //prvo provera postojanja preko readera,ide input radi azure load balancera
            NetTcpBinding readerbind = new NetTcpBinding();
            ChannelFactory<IReader> factory = new ChannelFactory<IReader>(readerbind, new
           EndpointAddress("net.tcp://localhost:10101/InputRequest"));
            IReader proxyreader = factory.CreateChannel();
            if (proxyreader.proveriDaLiPostojiDodavanjeKorisnik(collection["username"],
collection["ime"], collection["prezime"]))
            {
                ViewBag.Error = "Doslo je do greske! Postoji vec dati username.";
                return View();
            }
            else
            {
                NetTcpBinding binding = new NetTcpBinding();
                string internalEndpointName = "InternalRequest";
                List<RoleInstance> bratske_instance =
RoleEnvironment.Roles["WriterWorkerRole"].Instances.ToList(); //uzmemo sve writer
instance
                RoleInstance rola_dva = bratske_instance[1];//uzmemo samo drugu jer ona
radi sa secondary entitetima
                try
                {
                    RoleInstanceEndpoint pristupna =
rola_dva.InstanceEndpoints[internalEndpointName];
                    EndpointAddress endpoint = new
EndpointAddress(String.Format("net.tcp://{0}/{1}", pristupna.IPEndpoint.ToString(),
internalEndpointName));
                    IWriter proxy = new ChannelFactory<IWriter>(binding,
endpoint).CreateChannel();
                    proxy.PrihvatiZahtevCreateKorisnik(collection["username"],
collection["ime"], collection["prezime"], IDGenerator.idKorisnik);
                    IDGenerator.idKorisnik++;
                    return RedirectToAction("Index");
                }
                catch (Exception e)
                {
                    throw e;
                }
            }
        }
```

Kao što vidimo na početku se konektuje na Reader preko input komunikacije. Koristi se input vid komunikacije radi aktivacije azure load balancer-a traženog u specifikaciji zadatka. Reader proverava da li postoji već isti entitet i ako postoji vraća grešku i korisniku se prikazuje ponovo stranica za kreiranje entiteta i ispisuje poruka o grešci.

Create

Korisnik

Doslo je do greske! Postoji vec dati username.

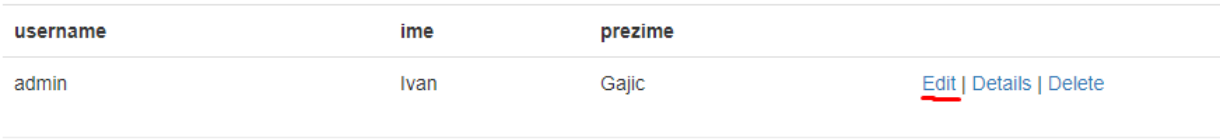
U drugom slučaju ako je sve uredu i entitet već ne postoji konektujemo se na drugu instancu writera. Tj instancu sa indeksom 1 jer je navedeno u specifikaciji zadatka da samo druga instanca writer worker role brine o sekundarnim entitetima. Pozivamo metodu writera „PrihvatiZahtevCreateKorisnik“ kojoj se prosleđuju potrebni parametri. Ta metoda kreira novi objekat tipa Korisnik i prosleđuje ga AddKorisnik metodi u data repozitorijumu koji čuva podatak u azure table storage. Slede 2 listinga za pomenute metode.

```
public bool PrihvatiZahtevCreateKorisnik(string username, string ime, string prezime, int
id)
        {
            Korisnik temp = new Korisnik(username, ime, prezime, id);
            AgencijaProject_Data.AgencijaDataRepository repo = new
AgencijaProject_Data.AgencijaDataRepository();
            repo.AddKorisnik(temp);
            //Trace.WriteLine("Invoked CreateKorisnik");
            LogujDogadjaj("[" + DateTime.Now.ToString() + "]:" + String.Format("Izvrseno
kreiranje novog entiteta Korisnik sa sledecim podacima. Username: {0}; Ime: {1}; Prezime:
{2}", username, ime, prezime));
            return true;
        }

public void AddKorisnik(Korisnik newKorisnik)
        {
            TableOperation insertOperation = TableOperation.Insert(newKorisnik);
            _table.Execute(insertOperation);
        }
```

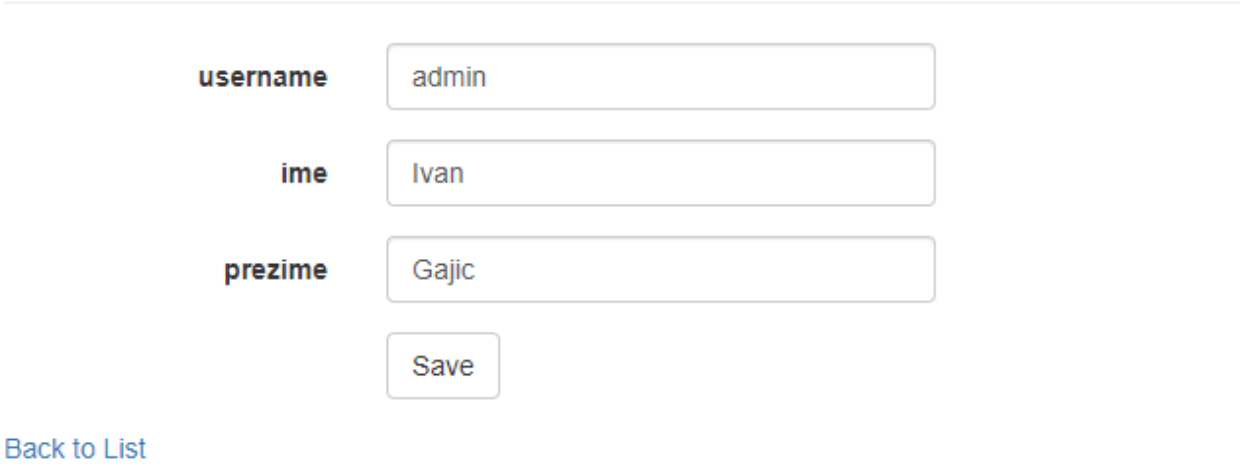
Time se završava proces kreiranja entiteta Korisnik koji pripada sekundarnoj grupi entiteta.

-Primer za modifikovanje sekundarnog entiteta tj. korisnika:

Ako smo izvršili gore navedeni test I dodali novog korisnika sada možemo modifikovati taj objekat. Na početnoj stranici web sajta kliknemo na “Izlistaj korisnike”. Zatim nam se otvara lista svih korisnika u sistemu. Biramo nekog od korisnika (u našem slučaju samo jedan postoji) I kliknemo na “Edit”.



Zatim nam se otvara nova stranica sa već popunjenom formom prethodnim podacima izabranog korisnika gde možemo izmeniti neki od podataka.



Klikon na “Save” button aktivira se metoda Edit u Korisnik kontroleru koja kroz parameter prima ID korisnika I vrednosti unešene u formu.

Sledi listing.

```
// POST: Korisnik/Edit/5
        [HttpPost]
        public ActionResult Edit(int id, FormCollection collection)
        {
            try
            {
                NetTcpBinding binding = new NetTcpBinding();
                string internalEndpointName = "InternalRequest";
                List<RoleInstance> bratske_instance =
RoleEnvironment.Roles["WriterWorkerRole"].Instances.ToList(); //uzmemo sve writer
instance
                RoleInstance drugaInstanca = bratske_instance[1];//samo druga jer je
secondary entity type
                RoleInstanceEndpoint pristupna =
drugaInstanca.InstanceEndpoints[internalEndpointName];
                EndpointAddress endpoint = new
EndpointAddress(String.Format("net.tcp://{0}/{1}", pristupna.IPEndpoint.ToString(),
internalEndpointName));
                IWriter proxy = new ChannelFactory<IWriter>(binding,
endpoint).CreateChannel();
                proxy.PrihvatiZahtevModifyKorisnik(collection["username"],
collection["ime"], collection["prezime"], id);

                return RedirectToAction("Index");
            }
```

```
            catch (Exception e)
            {
                throw e;
            }
        }
```

Kao što vidimo konektujemo se na writer instancu broj dva jer je ovo sekundarni entitet. I pozivamo metodu „PrihvatiZahtevModifyKorisnik“ prosleđujući joj neophodne podatke za objekat tipa Korisnik. Ta metoda kreira nov objekat korisnika i prosleđuje ga „ModifyKorisnik“ metodi u data repozitorijumu. Ta metoda pronalazi vec postojećeg korisnika u azure tabeli i modifikuje ga na osnovu pristiglih podataka. Slede listinzi tih metoda.

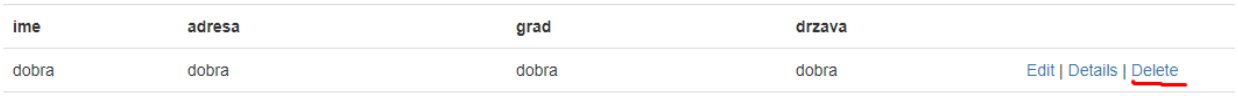
```
public void PrihvatiZahtevModifyKorisnik(string username, string ime, string prezime, int
id)
        {
            Korisnik temp = new Korisnik(username, ime, prezime, id);
            AgencijaProject_Data.AgencijaDataRepository repo = new
AgencijaProject_Data.AgencijaDataRepository();
            repo.ModifyKorisnik(temp);
            //Trace.WriteLine("Invoked Modify korisnik");
            LogujDogadjaj("[" + DateTime.Now.ToString() + "]:" + String.Format("Izvrsena
modifikacija entiteta Korisnik sa sledecim podacima. Username: {0}; Ime: {1}; Prezime:
{2}", username, ime, prezime));
        }



public void ModifyKorisnik(Korisnik newInfo)
        {
            TableOperation retrieveOperation =
            TableOperation.Retrieve<Korisnik>(newInfo.PartitionKey, newInfo.RowKey);
            var existingItem = _table.Execute(retrieveOperation);
            newInfo.ETag = existingItem.Etag;
            if (existingItem != null)
            {
                TableOperation replaceOperation =
                TableOperation.Replace(newInfo);
                _table.Execute(replaceOperation);
            }
        }
```

Ovim je završen test slučaj modifikacije sekundarnog entiteta.

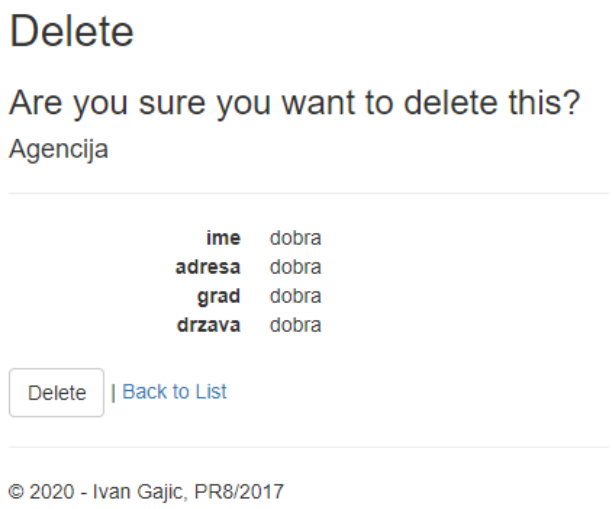
-Primer za brisanje primarnog entiteta tj. agencije:

Ako uzmemo u obzir da smo u prvom test slučaju kreirali agenciju. Sada klikom na "Izlistaj agencije" na početnoj strani dobijamo listu svih agencija, u našem slučaju jedne. Mi možemo nad njom vršiti sve CRUD operacije, a u ovoj dokumentaciji ćemo pokriti pored dodavanja, brisanje iste.

Brisanje započinje klikom na "Delete"



Zatim nam se otvara prozor potvrde brisanja.



Klikom na "Delete" button aktivira se metoda Delete u okviru Agencija kontrolera. Njoj se prosleđuju ID agencije I vrednosti polja.

Sledi listing metode.

```
public ActionResult Delete(int id, FormCollection collection)
        {
            try
            {
                NetTcpBinding binding = new NetTcpBinding();
                string internalEndpointName = "InternalRequest";
                List<RoleInstance> bratske_instance =
RoleEnvironment.Roles["WriterWorkerRole"].Instances.ToList(); //uzmemo sve writer
instance

                if (cnt % 2 == 0)//prva instanca konektovanje
                {
                    RoleInstanceEndpoint pristupna =
bratske_instance[0].InstanceEndpoints[internalEndpointName];
```

```
                    EndpointAddress endpoint = new
EndpointAddress(String.Format("net.tcp://{0}/{1}", pristupna.IPEndpoint.ToString(),
internalEndpointName));
                    IWriter proxy = new ChannelFactory<IWriter>(binding,
endpoint).CreateChannel();


                    proxy.PrihvatiZahtevBrisanje(collection["ime"], collection["adresa"],
collection["drzava"], collection["grad"], id);

                    cnt++;
                }
                else//treca instanca konektovanje
                {
                    RoleInstanceEndpoint pristupna =
bratske_instance[2].InstanceEndpoints[internalEndpointName];
                    EndpointAddress endpoint = new
EndpointAddress(String.Format("net.tcp://{0}/{1}", pristupna.IPEndpoint.ToString(),
internalEndpointName));
                    IWriter proxy = new ChannelFactory<IWriter>(binding,
endpoint).CreateChannel();
                    proxy.PrihvatiZahtevBrisanje(collection["ime"], collection["adresa"],
collection["drzava"], collection["grad"], id);

                    cnt++;
                }



                return RedirectToAction("Index");
            }
            catch (Exception e)
            {
                throw e;
            }
        }
```

Kao što vidimo konektujemo se na writer worker rolu i to samo prvu ili treću instancu po principu ostatka deljenja rednog broja zahteva sa dvojkom. Zatim se poziva metoda „PrihvatiZahtevBrisanje“ iz writer role i prosleđuju joj se podaci o entitetu.

Ta metoda kreira objekat Agencije na osnovu pristiglih podataka i prosledjuje ga metodi „DeleteAgencie“ u data repozitorijumu. „DeleteAgencie“ pronalazi entitet u tabeli i uklanja ga. Slede listinzi pomenutih metoda.

```
public void PrihvatiZahtevBrisanje(string ime, string adresa, string drzava, string grad,
int id)
        {
            Agencija temp = new Agencija(ime, adresa, grad, drzava, id);
```

```
            AgencijaProject_Data.AgencijaDataRepository repo = new
AgencijaProject_Data.AgencijaDataRepository();
            repo.DeleteAgencie(temp);
            LogujDogadjaj("["+DateTime.Now.ToString()+"]:" + String.Format("Izvrseno
brisanje agencije sa sledecim podacima. Ime: {0}; Adresa:{1}; Drzava:{2};
Grad:{3}",ime,adresa,drzava,grad));
        }

public void DeleteAgencie(Agencija deleteAgencie)
        {
            TableOperation retrieveOperation =
            TableOperation.Retrieve<Agencija>(deleteAgencie.PartitionKey,
deleteAgencie.RowKey);
            var existingItem = _table.Execute(retrieveOperation);
            deleteAgencie.ETag = existingItem.Etag;
            TableOperation deleteOp = TableOperation.Delete(deleteAgencie);
            _table.Execute(deleteOp);

        }
```

Time je završen proces brisanja agencije iz azure table storage-a.

Ovim test slučajevima je pokriven veći deo aplikacije. Nisu navedeni test slučajevi za rad sa ponudama jer u principu jedina je razlika što korisnik bira „Kreiraj Ponudu“, unosi njene neophodne podatke i submituje. Same metode rade potpuno isto kao i za „Korisnik“ objekte jer je „Ponuda“ takođe sekundarni tip entiteta. Konektuju se na reader putem input komunikacije, dok pri komunikaciji sa writer-om koriste internal komunikaciju i konektuju se samo na drugu instancu writer rola po specifikaciji zadatka.