REST servisi

Zoran Ćirović

Uvod

- Web servisi su programske komponente koje omogućavaju komunikaciju među udaljenim računarima. Web servisi predstavljaju osnovu za servisno orijentisane arhitekture (SOA). Okosnica SOA arhitekture tj. distribiranog sistema su javne metode – procedure koje čine servise i postavljaju se na servere.
- Druga vrsta arhitekture je resursno orijentisana arhitektura (ROA) za koju se ne kreiraju procedure već resursi kojima se pristupa.

Zoran Ćirović

- REST je arhitektura. Nekada se koristi i naziv RESTful obično za primenu ove arhitekture.
- Poreklo naziva: *Engl. Representational state*
- Omogućuje komunikaciju na Internetu primenom svih najpoznatijih programskih jezika kao što su C/C++, C#, Jave, Ruby, PHP i sličnih kao i preko najpoznatijih operativnih sistema: Microsoft Windowsa, Mac OSa, Linuxa itd.
- Obezbeđuje odličan rad sa velikim brojem klijenata (engl. scalability)
  Primena na veliki broj programerskih problema kao što su dohvatanje podataka na serveru, dodavanje, promena ili brisanje postojećih podatka i nezavisnost od tehnologije, razlog je zašto ovaj protokol koriste mnoge velike tvrtke kao što su Google (GoogleMaps[1], GoogleSearch[2]), Microsoft (Azure[3], Shrepoint[4]) za komunikaciju na Internetu
- ZASNIVA SE NA KORIŠĆENJU POSTOJEĆIH PROTOKOLA (pre svih http protokola)

Karakteristike

- Principi REST tehnologije su:
  - Jedinstveni interfejs (zasnovan na resursima. Resurs ima identifikator, URI. Format može biti XML, JSON, HTML,…),
  - Nema stanja (osnovni princip RESTa),
  - Keširanje - klijent može keširati resurse, a server može da da informaciju u vezi keširanja resursa,
  - Klijent-Server - klijent vidi URI resursa, dakle postoji razdvojenost od servera,
  - Slojeviti sistem - Klijent (ni)malo poznaje server. Koriste se slojevi poput proxija ili drugih medju servera),
  - Kod na zahtev - Server može proširiti funkcionalnost klijenta slanjem koda. To može biti JS kod.

Osnovna pravila

- Nekoliko pravila koji se poštuju kada se vrši pristup podacima preko resursa:
  - Informacije su statičke strane, fajlovi, delovi fajla, podaci iz baze i slično.
  - Resurs je informacija kojoj se pristupa udaljeno. Svaki resurs ima sopstveni URL ili URI koji ga identifikuje.
  - Pristup do resursa se definiše pomoću HTTP protokola.
  - Svaki poziv označava jednu akciju koja čita, menja ili briše podatke. Poziv ne označava pristup do neke stranice.

Primer 1 - 1

- Neka je potrebno omogućiti udaljeno: kreiranje, brisanje, promenu i čitanje podataka o knjigama. Dakle, radi se o zahtevu da se sve 4 operacije mogu izvesti udaljeno tzv. kompletna CRUD kontrola.
- U standardnim web servisima bi bilo potrebno kreirati skup metoda za pristup proizvodima. Na primer:
  - NovaKnjiga(int id, ..)
  - IzmenaKnjige(int id, string naslov, ...)
  - BrisanjeKnjigu(int id)
  - ListaKnjiga(string kriterijumPretrage)

Primer 1 - 2

- REST uvodi promenu razmišljanja u odnsu na operacije.
- Umesto skupa operacija, za svaku operaciju koje smo pozivali u slučaju XML servisa, sada se vrši pristup resursima. Po HTTP protokolu postoji nekoliko metoda kojima se može zahtevati resurs po URLu. Najčešći metodi su GET metod kojim se čita određena strana i POST kojim se šalju podaci serveru. Postoje još neke metodi kao što su PUT, DELETE ili HEAD ali se retko koriste.
- Servis je npr. na adresi: *http://www.viser.edu.rs/knjizara/knjiga*
- Umesto različitih URLova za svaku operaciju, u REST servisima se primenjuju različite metode HTTP protokola za različite namene.
- Pogledajmo koje su to HTTP metode.

HTTP/1.1 metode

1. OPTIONS - Predstavlja zahtev o opcijama komunikacije. Ovom metoda klijent može da utvrdi opcije vezane za resurse
2. GET - Zahteva se prikaz tj. vrši se čitanje određenog resursa. Koristi se samo za čitanje podataka, što znači da se smatra sigurnim.
3. HEAD - Identična kao i GET, s tim što server ne vraća telo poruke u odgovoru. Ova metoda može da se koristi za prikupljanje meta podataka zapisanih u zaglavlju odgovora, bez prebacivanja kompletnog sadržaja.
4. POST - Koristi se za kreiranje novih resursa.
5. PUT - Ova metoda se najčešće koristi za ažuriranje, zamenu ili kreiranje novog prikaza resursa.
6. DELETE - Ova metoda se koristi za brisanje resursa.
7. TRACE - Vraća primljene zahteve tako da klijent može da vidi da li je napravljena neka izmena od strane posrednih servera.
8. CONNECT - Konvertuje konekciju u transparentni TCP/IP tunel. Obično se koristi da olakša SSL enkriptovanu komunikaciju kroz neenkriptovani HTTP proksi.

HTTP statusni kodovi

HTTP definiše paket standardnih statusnih kodova koje preciziraju rezultat obrade zahteva.

Statusni kodovi su organizovani u opsege koji imaju određeno značenje.

Postoji 5 tipova HTTP status kodova:

- Informational (1xx)
- Success (2xx)
- Redirection (3xx)
- Client errors (4xx)
- Server errors (5xx)

Na primer, statusni kodovi na 200 znače da je „uspešno“ a 400 da je zadat loš zahtev.

Opseg	Opis
100	Informacioni
200	Uspešan
201	Napravljen
202	Prihvaćen
300	Preusmeravanje
304	neizmenjen
400	Greška klijenta
402	Plaćanje obavezno
404	Nije pronađen
405	Nije dozvoljen
500	Greška na serveru
501	Ne izvršavaj

Primer 1 - 3

- Dakle, pošto REST servisi koriste HTTP metode kako bi na istom URL prepoznali da li klijent želi da pročita, kreira, ažurira ili obriše neki podatak, pretpostavimo da se URL: http://www.viser.edu.rs/knjizara/knjiga/28 koristi da bi se pristupilo resursu knjiga sa ID=28.
- Format URLa nije standardizovan i zavisi od realizacije konkretnog REST servisa, tj. za istu namenu a drugačiji URL može biti: http://www.viser.edu.rs/knjizara/knjiga(28)
- http://www.viser.edu.rs/knjizara/knjiga?id=28
- Pogledajmo jedan primer upotrebe različitih metoda za realizaciju CRUD operacija.

Primer 1 - 4

- Čitanje
- GET: vraćanje podatka o knjizi id=28

- Ažuriranje
- PUT: šalju se podaci o knjizi i vrši se ažuriranje knjige id=28

- Kreiranje
- POST: šalju se podaci o novoj knjizi

- Brisanje
- DELETE: briše se knjiga id=28.

- A sada i kompletan skup pratećih URL adresa za odgovarajuće realizacije.

Primer 1 – 4 kompletan

Metod	URI	Opis
GET	/knjige	Vraća listu knjiga.
GET	/knjige/42	Vraća knjigu id=42.
DELETE	/knjige/42	Briše knjigu 42.
POST	/knjige	Kreira jedan knjigu .
PUT	/knjige/42	Ažurira knjigu 42.
PATCH	/knjiga/42	Delimično ažuriranje.
OPTIONS	/knjiga/42	Vraća listu operacija raspoloživih za resurs.
HEAD	/knjiga/42	Vraća samo HTTP header.

- Napomena: Format u kome se podaci vraćaju nije unapred definisan i zavisi od realizacije na samom servisu. Može biti XML, JSON ili bilo neki drugi format.

Primer javnih REST servisa

- REST servisi mogu biti javno dostupni. Na primer
- http://code.google.com/more/
- https://developers.facebook.com/
- https://dev.twitter.com/rest/public
- https://affiliate-program.amazon.com/gp/advertising/api/detail/main.html
- https://www.publicapis.com/ (lista mnogih javnio dostupnih servisa)

- Primer javnog servisa namenjenog učenju:
- **The Gate to the Service**
  http://www.thomas-bayer.com/sqlrest/
- **Customer # 18**
  http://www.thomas-bayer.com/sqlrest/CUSTOMER/18/
- **List of invoices**
  http://www.thomas-bayer.com/sqlrest/INVOICE/

Primeri odziva javnih servisa

`http://www.thomas-bayer.com/sqlrest/`

```
<?xml version="1.0"?>
<resource xmlns:xlink="http://www.w3.org/1999/xlink">
  <CUSTOMERList xlink:href="http://www.thomas-bayer.com/sqlrest/CUSTOMER/">CUSTOMER</CUSTOMERList>
  <INVOICEList xlink:href="http://www.thomas-bayer.com/sqlrest/INVOICE/">INVOICE</INVOICEList>
  <ITEMList xlink:href="http://www.thomas-bayer.com/sqlrest/ITEM/">ITEM</ITEMList>
  <PRODUCTList xlink:href="http://www.thomas-ayer.com/sqlrest/PRODUCT/">PRODUCT</PRODUCTList>
</resource>
```

`http://www.thomas-bayer.com/sqlrest/CUSTOMER/18/`

```
<?xml version="1.0"?>
<CUSTOMER xmlns:xlink="http://www.w3.org/1999/xlink">
  <ID>18</ID>
  <FIRSTNAME>Sylvia</FIRSTNAME>
  <LASTNAME>Fuller</LASTNAME>
  <STREET>158 - 20th Ave.</STREET>
  <CITY>Paris</CITY>
</CUSTOMER>
```

Prednosti u odnosu na SOAP servise

- REST web servisi imaju nekoliko prednosti nad standardnim web servisima.
- Jednostavniji su.
  - Nema složenog formatiranja parametara i rezultata. Klijenti koji pozivaju REST servise ne moraju da formatiraju zahteve po SOAP specifikaciji i ne moraju da parsiraju SOAP odgovor kako bi iz njega izvukli rezultat. Jednostavni klijenti kao što su JavaScript Ajax direktno pozivaju REST servise i prihvataju rezultate.
- Podaci koji se vraćaju mogu biti u formatu koji odgovara klijentu.
  - Na primer JavaScript kod može dobiti podatke u JSON formatu koji lako može da pročita, RSS čitač u RSS-XML formatu koji može da prikaže. S obzirom da je format u kome se vraćaju podaci fleksibilan, različiti klijenti mogu da zatraže podatke u formatu koji im najviše odgovara što je dosta lakše od SOAP formata koji iako je standardizovan mora da se parsira.

Određivanje resursa – Razmatranje sigurnosti

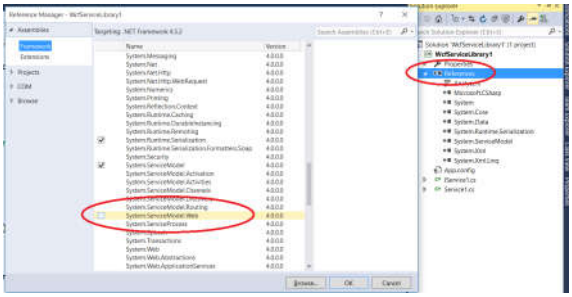
- Naš servis zahteva opciju autentifikacije korisnika, zato moramo izvršiti autorizaciju resursa i metoda kojima se može pristupati. Na primer, samo ulogovani korisnici mogu pristupiti njihovim nalozima i raditi sa njima. Takođe, samo ulogovani korisnici mogu kreirati nove bookmark objekte i raditi sa njima. U slučaju da neki neautorizovan korisnik pokuša pristupiti ovim resursima servis treba da vrati 401.
- HTTP dolazi sa nekim ugrađenim mehanizmima autentifikacija od kojih je mehanizam osnovne autentifikacije najpopularniji. Najpopularniji je jer je veoma lak za realizaciju i ima široku podršku, ali je i najmanje siguran, jer se u ovom slučaju lozinke šalju u tekstualnom formatu preko mreže i postaju lak plen za presretače i dekodovanje.
- Jedan način za rešenje je primena SSL (HTTPS) za sav HTTP saobraćaj u kome se očekuje osnovna autentifikacija, pošto bi u ovom slučaju lozinka bila prenošena kroz zaštićen kanal.
- Drugi pristup je koristeći digitalnu autentifikaciju (drugu šemu autentifikacije ugrađenu u HTTP). Ova šema sprečava prisluškivanje tako što nikada ne šalje lozinku preko veze. Umesto toga, algoritam zahteva slanje *hash* vrednosti izračunate na osnovu lozinke i neke (jedne ili više) drugih vrednosti koje su poznate samo klijentu i serveru. Server može da, na osnovu dobijene poruke, izračuna *hash* vrednost i tako proveri da li je klijent vlasnik te poruke sa lozinkom.
- Ova zaštita je mnogo bolja od osnovne, ali je i dalje ranjiva na pokušaje koristeći rečnik ili tzv napad „brutalnom silom“.

- Sledeći pristup je implementacije sopstvene autentifikacione šeme oko „Autorizacionog“ zaglavlja. Mnoge od ovih šema koriste tvz. *Hash Message Authentication Code* (HMAC) pristup u kome server omogućava da klijent sa nekim korisničkim id i tajnim ključem koristeći neku drugu tehniku da prenese tajni ključ (na primer koristeći e-mail šalje se user id i tajni ključ). Zatim klijent koristi ovaj ključ za šifrovanje.
- Za ovaj pristup servis mora definisati algoritam za klijeta pri prijavi. Na primer, mora definisati kako se poruka koristi za šifrovanje tj koji delovi treba da budu uključeni u HMAC potpis. Ovo je važno jer se procedura provere obavlja identično na obe strane. Pošto klijent generiše HMAC hash on uključije i Autorizaciono zaglavlje sa user id, na primer
- Authorization: skonnard:uCMfSzkjue+HSDygYB5aEg==
- Pošto servis prihvati Autorizaciono zaglavle i pošto odvoji user id odnosno hash vrednost, može zatim odrediti ne samo korisnika već i primeniti isti HMAC algoritam na poruku. Ako je izračunata hash vrednost odgovara onoj u poruci zna se sa sigurnošću da se klijent vlasnik tajne poruke i da je validan korisnik. Takođe smo sigurni da niko nije mogao da izmeni deo poruke. Možemo u poruku uključiti timestamp takođe. Tada servis može odbaciti zastarele poruke iz razloga dodatne sigurnosti.
- HMAC je najbolji algoritam ako je generisanje slučajnosti bazirano na dovoljnoj dugoj pseudoslučajnoj sekvenci odnosno

Izrada REST servisa pomoću WCF servisa

Povezivanje REST servisa sa WCF modelom servisa

- Od verzije 3.5 WCF sadrži biblioteku System.ServiceModel.Web.dll koja nudi više klasa koje omogućavaju lako korišćenje "Web-based" modela za izradu REST servisa. Pogledati sliku.



- Pošto se biblioteka uključi u Reference projekta, potrebno je u postojeći WCF servis dodati mapiranje servisnog interfejsa tj metoda dodajući „Web" atribute tj. definišući koja metoda odgovara kojem URI.

Modelovanje resursa

- Prvi način modelovanja je preko obrade poruka request/response.
- Međutim, obično se koriste već pripremljene klase koje obavljaju serijalizaciju u tipične formate resursa. Tada se mogu primeniti neke serijalizacije neophodne za ugovore o podacima.
- Na primer:
- *DataContractSerializer* je veoma efikasan ali podržava mali podskup schema.
- *XmlSerializer*, složeniji i napredniji.
- *DataContractJsonSerializer* ili *Newtonsoft* – json

- Pre verzije .NET3.5 uz svaki podatak trebalo je da stoji [DataContract] odnosno [DataMember] kako bi se serijalizacija uspešno obavila. Sada to za ovako zapisane podatke nije neophodno. U ovom primeru je to urađeno samo za Bookmarks koristeći [CollectionDataContract], ali se može primeniti i na ostale elemente ako je potrebno neko složenije definisaje serijalizacije.

```
public class User
{
    public Uri Id { get; set; }
    public string Username { get; set; }
    public string Name { get; set; }
    public string Email { get; set; }
    public Uri Bookmarks { get; set; }
}
public class UserProfile
{
    public Uri Id { get; set; }
    public string Name { get; set; }
    public Uri Bookmarks { get; set; }
}
public class Bookmark
{
    public Uri Id { get; set; }
    public string Title { get; set; }
    public Uri Url { get; set; }
    public string User { get; set; }
    public Uri UserLink { get; set; }
    public string Tags { get; set; }
    public bool Public { get; set; }
    public DateTime LastModified { get; set; }
}
[CollectionDataContract]//nije neophodno
public class Bookmarks : List<Bookmark>
{
    public Bookmarks() { }
    public Bookmarks(List<Bookmark> bookmarks) : base(bookmarks) { }
}
```

Definisanje HTTP interfejsa

- WCF definiše više atributa kojima se definišu HTTP interfejsi koje servis podržava. Napomena: Već je poznato je da atribute [ServiceContract] odnosno [OperationContract] koriste i SOAP i REST servisi.
- Novi atributi za definisanje HTTP metoda su:
  - [WebGet]
  - [WebInvoke]

  Pažnja: Zbog značajne razlike u primeni GET metode od svih ostalih postoje dva različita atributa, mada WebInvoke može da se koristi i za Get zahteve.

  Osnovno što se definiše pri upotrebi ovih atributa je UriTemplate uz metod koji treba da obradi zahtev. Takođe, kroz ovaj atribut mogu se mapirati i promenljive jednostavnim korišćenjem istog imena na oba mesta.
- Na sličan način može da se primeni [WebInvoke] atribut na druge metode.
- Međutim, pošto se ovaj atribut odnosi na sve preostale HTTP metode, neophodno je definisati korišćenu metodu.

```
[WebGet(UriTemplate = "?tag={tag}")]
[OperationContract]
Bookmarks GetPublicBookmarks(string tag);

[WebGet(UriTemplate = "{username}?tag={tag}")]
[OperationContract]
Bookmarks GetUserPublicBookmarks(string username, string tag);

[WebGet(UriTemplate = "users/{username}/bookmarks?tag={tag}")]
[OperationContract]
Bookmarks GetUserBookmarks(string username, string tag);

[WebGet(UriTemplate = "users/{username}/profile")]
[OperationContract]
UserProfile GetUserProfile(string username);

[WebGet(UriTemplate = "users/{username}")]
[OperationContract]
User GetUser(string username);

[WebGet(UriTemplate = "users/{username}/bookmarks/{bookmark_id}")]
[OperationContract]
Bookmark GetBookmark(string username, string bookmark_id);
```

```
[WebInvoke(Method = "PUT", UriTemplate = "users/{username}")]
[OperationContract]
void PutUserAccount(string username, User user);

[WebInvoke(Method = "DELETE", UriTemplate = "users/{username}")]
[OperationContract]
void DeleteUserAccount(string username);

[WebInvoke(Method = "POST", UriTemplate = "users/{username}/bookmarks")]
[OperationContract]
void PostBookmark(string username, Bookmark newValue);

[WebInvoke(Method = "PUT", UriTemplate =
"users/{username}/bookmarks/{id}")]
[OperationContract]
void PutBookmark(string username, string id, Bookmark bm);

[WebInvoke(Method = "DELETE", UriTemplate =
"users/{username}/bookmarks/{id}")]
[OperationContract]
void DeleteBookmark(string username, string id);
```

Hostovanje i konfigurisanje

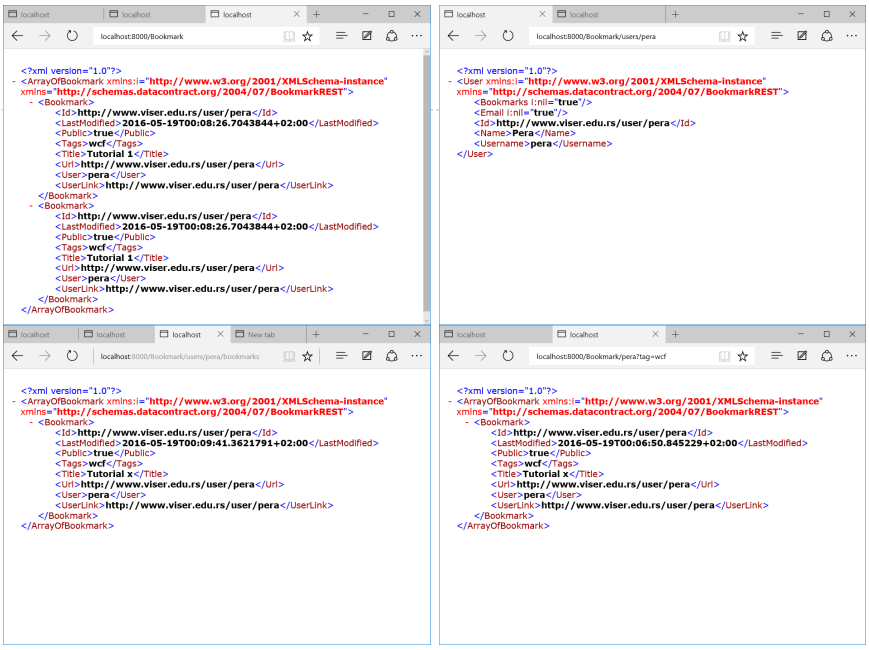
- Pri hostovanju RESTful WCF servisa postoje dve ključne komponente koje treba konfigurisati da bi se omogućilo "Web" ponašanje tokom izvršavanja. A) Krajnja tačka treba da ima svojstvo povezivanja prilagođeno za RESTful servise, a to je – WebHttpBinding.
- B) Potrebno je dodatno opisati ponašanje (*behavior*) "Web" krajnje tačke sa WebHttpBehavior. Novo ponašanje definiše da WCF ne koristi SOAP XML već obične XML ili JSON poruke, naravno uz upotrebu atributa [WebGet] odnosno [WebInvoke] i odgovarajućih UriTemplates objekata.
- Hostovanje u konzolnoj ili windows aplikaciji sa primenom konfiguracionog fajla za ostala podešavanja može se izvesti na sledeći način:

```
Uri adresa = new Uri("http://localhost:8000/Bookmark");
ServiceHost serviceHost = new
ServiceHost(typeof(BookmarkService), adresa);
serviceHost.Open();
```

- U ovom primeru, specificira se bazna URI za servis pri stvaranju objekta tipa ServiceHost. Ista adresa se koristi i kao adresa krajnje tačke, osim ako se posebno specificira.

Prateći konfiguracioni fajl

```
<configuration>
  <system.serviceModel>
    <services>
      <service name="BookmarkService">
        <endpoint binding="webHttpBinding" contract="BookmarkREST.BookmarkService"
                behaviorConfiguration="webHttp"/>
      </service>
    </services>
    <behaviors>
      <endpointBehaviors>
        <behavior name="webHttp">
          <webHttp/>
        </behavior>
      </endpointBehaviors>
    </behaviors>
  </system.serviceModel>
<configuration>
```



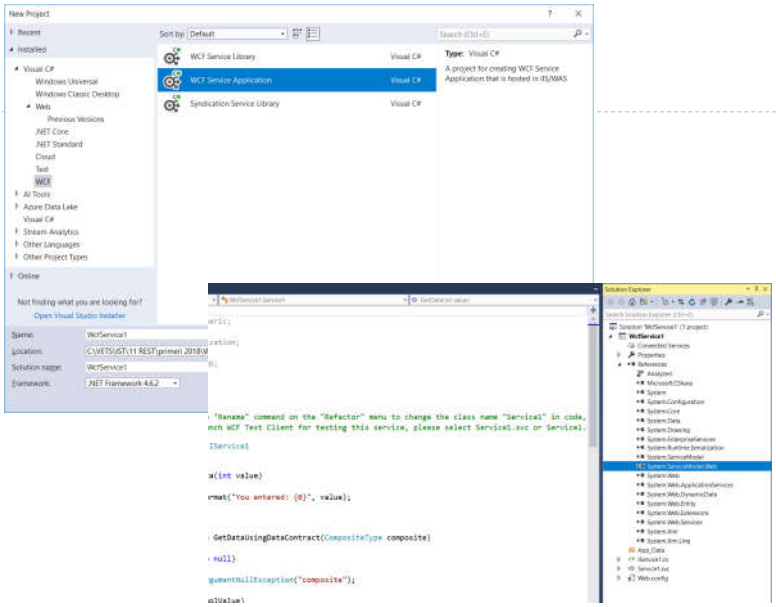
Šta se događa iza scene? IHttpHandler

- Moguće je pratiti sve poruke i upravljati istim ako RESTful servis implementira interfejs IHttpHandler. Tada je neophodno uraditi sve potrebno koristeći samo jednu ulaznu tačku tj. ProcessRequest – za obradu dolazećih HTTP zahteva, a to znači:
  - Ispitati dolazeći URI i identifikovati resurs.
  - Izdvojiti sve promenljive iz URI.
  - Izvršiti autentifikaciju korisnika.
  - Odrediti HTTP metod.
  - Ako postoji pročitati podatke o resursu.
  - Izvršiti odgovarajući logiku na servisnoj strani.
  - Generisati odgovarajući HTTP odgovor uključujući odgovarajći statusni kod.

Upotreba REST servisa

Uvod

- Da bi pokazali upotrebu servisa potrebno je napraviti jedan jednostavan REST servis. Ako koristite Visual Studio, napravite web aplikaciju u kojoj je dovoljno dodati novi WCF servis (Add > New Item > Web > WCF Service) i biće kreirana dva fajla: Interfejs kojim se opisuju metode REST servisa Implementacija u kojoj se nalazi kod koji realizuje operacije. Pod pretpostavkom da je napravljen WCF servis Service1, interfejs bi bio generisan u fajlu IService1.cs i izgledao bi kao u sledećem listingu:
- `[OperationContract]`
- `[WebInvoke(Method ="GET", RequestFormat = WebMessageFormat.Xml, ResponseFormat = WebMessageFormat.Xml, UriTemplate ="GetData/{value}")]`
- `string GetData(string value);`
- Obratite pažnju da je ulazni argument promenjenu string. Ovo je urađeno iz razloga testiranja servisa sa drugim formatom koji je prilagođen samo stringovima.
- Na kraju obavezno se mora prilagodti konfigurcioni fajl. Bez ovih izmena servis neće biti RESTful.

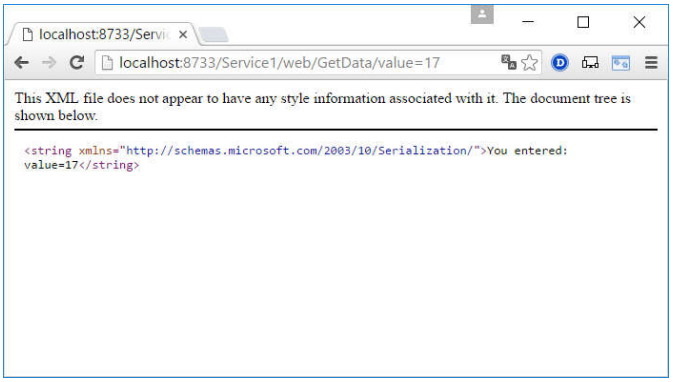


- U slučaju korišćenja standardnog WCF servisa tj šablona „Wcf Service Library“ neophodno je uraditi izmene na konfiguracionom fajlu:
  - Dodata je nova krajnja tačka pored postojeće. Nova krajnja tačka je prilagođena radu sa web servisima.
  - Zatim je dodata konfiguracija za web servis.

```
***
<endpoint address="web" binding="webHttpBinding"
contract="WcfServiceLibrary1.IService1" behaviorConfiguration="aaa" >
</endpoint>
<endpoint address="wcf" binding="basicHttpBinding"
contract="WcfServiceLibrary1.IService1">
</endpoint>
***
<endpointBehaviors>
     <behavior name="aaa">
       <webHttp/>
     </behavior>
</endpointBehaviors>
***
```

- U ovom primeru je napravljen interfejs koji opisuje web servis koji će vratiti detalje na osnovu prosledjene vrednosti metodom GetData. Samo ime metode je beznačajno - ona se ne poziva po imenu nego po URI šablonu /GetData/{value}. Metoda će vratiti podatke o proizvodu u JSON formatu kada se otvori neki url kao na primer /WCFtestServis.svc/GetData/17 .
- Pošto metoda vraća podatke ona je podešena tako da reaguje na HTTP GET metod. WCF Servisi omogućavaju još dosta prilagođavanja servisa. Više detalja o implementaciji možete naći u tekstovima Create RESTful WCF Service API: Step By Step Guide ili CREATE RESTful WCF Service API Using POST: Step By Step Guide. Tamo možete naći detaljnija uputstva o tome kako se može napraviti REST pomoću WCF servisa.
- Napomena: Pored klasičnih REST servisa postoje i takozvani OData servisi koji pored jednostavnih upita omogućavaju i složenije upite. OData servisi se mogu posmatrati kao specijalan slučaj naprednih REST servisa. Više informacija o OData servisima možete naći u artiklu OData services. Primer OData servisa je Netflix web servis predstavljen u prethodnoj sekciji.

- REST servis možemo testirati iz web čitača, kako je prikazano na slici, ili pomoću specijalizovanih programa kao na primer Postman
- Rad sa xml porukama je na strani klijenta u okviru njegovih poziva funkcija.
- Sada ćemo da razmotrimo još jedan format.



Zaključak

- REST servisi predstavljaju veoma primenljivu alternativu standardnim SOAP servisima. Osnovna prednost REST servisa je u slučajevima kada treba omogućiti punu kontrolu nad informacijama, ili kada treba pojednostaviti protokol komunikacije sa servisom kako bi i jednostavni klijenti kao JavaSkript Ajax pozivi mogli da uzimaju podatke sa servisa i dinamičkiih ih prikazuju. REST servisi su sigurno nešto na šta ćete naići tokom rada u web aplikacijama tako da je korisno znati šta mogu da pruže. Ovo je samo kratak pregled osnovnih koncepata ali REST servisi pružaju mnogo više tako da je korisno istražiti ih.

Kompletan primer/servis - 1

```
[WebGet(UriTemplate = "init", ResponseFormat = WebMessageFormat.Json)]
[OperationContract]
string Init();

[WebInvoke(ResponseFormat = WebMessageFormat.Json, RequestFormat = WebMessageFormat.Json, Method = "GET", UriTemplate = "GetData/{value}")]
[OperationContract]
string GetData(string value);

[WebInvoke(Method = "POST", ResponseFormat = WebMessageFormat.Json, UriTemplate = "InitPost")]
[OperationContract]
string InitPost(Stream podaci);

[WebInvoke(Method = "POST", ResponseFormat = WebMessageFormat.Xml, UriTemplate = "InitPostXml")]
[OperationContract]
string InitPostXml(Stream podaci);


[WebInvoke(Method = "POST", UriTemplate = "PostDataUsingDataContract1", ResponseFormat = WebMessageFormat.Json, RequestFormat = WebMessageFormat.Json)]
[OperationContract]
bool PostDataUsingDataContract1(CompositeType2 c);
```

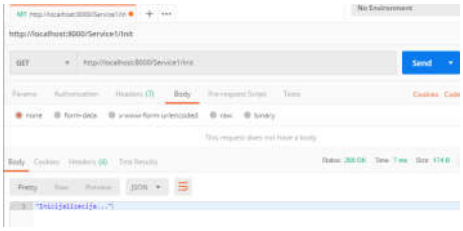
Kompletan primer/servis - 2

```
[WebInvoke(Method = "POST", UriTemplate = "PostDataUsingDataContract2", ResponseFormat =
WebMessageFormat.Json, RequestFormat = WebMessageFormat.Json)]
[OperationContract]
string PostDataUsingDataContract2(CompositeType2 c);


[WebInvoke(Method = "POST", UriTemplate = "PostDataUsingDataContract3", ResponseFormat =
WebMessageFormat.Json, RequestFormat = WebMessageFormat.Json)]
[OperationContract]
CompositeType PostDataUsingDataContract3(CompositeType c);

[WebGet(UriTemplate = "PostDataContract",ResponseFormat =WebMessageFormat.Xml)]
[OperationContract]
CompositeType2 PostDataContract();

[WebInvoke(ResponseFormat = WebMessageFormat.Json, RequestFormat = WebMessageFormat.Json, Method =
"PUT", UriTemplate = "PutData/{value}")]
[OperationContract]
string PutData(string value);

[WebInvoke(ResponseFormat = WebMessageFormat.Json, RequestFormat = WebMessageFormat.Json, Method =
"DELETE", UriTemplate = "DeleteData/{value}")]
[OperationContract]
string DeleteData(string value);
```

Windows klijent i testiranje preko Postman okruženja

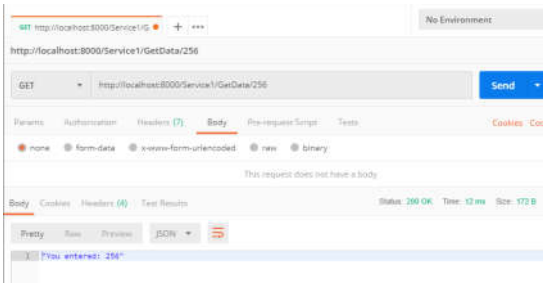
Kompletan primer/klijent - Init

```
using System.Runtime.Serialization;
using System.Runtime.Serialization.Json;

WebClient webClient = new WebClient();
string serviceURL = string.Format("http://localhost:8000/Service1/Init");
byte[] data = webClient.DownloadData(serviceURL);
Stream stream = new MemoryStream(data);

StreamReader reader = new StreamReader(stream);
string text = reader.ReadToEnd();
```



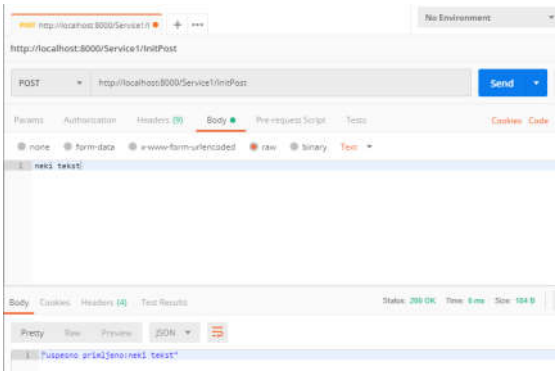
Kompletan primer/klijent - GetData

```
WebClient webClient = new WebClient();
string serviceURL = string.Format("http://localhost:8000/Service1/GetData/{0}", val);
byte[] data = webClient.DownloadData(serviceURL);
Stream stream = new MemoryStream(data);

StreamReader reader = new StreamReader(stream);
string text = reader.ReadToEnd();
```



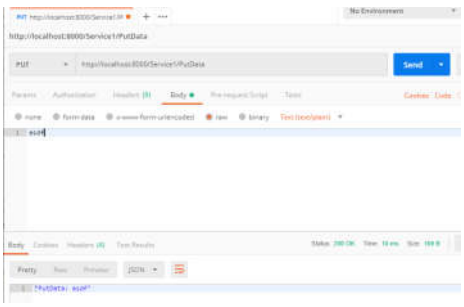
Kompletan primer/klijent - InitPost

```
WebClient webClient = new WebClient();
webClient.Headers["Content-type"] = "application/json";
webClient.Encoding = Encoding.UTF8;
string rez = webClient.UploadString("http://localhost:8000/Service1/InitPost", podaciStr);
```



Kompletan primer/klijent - PutData

```
WebClient webClient = new WebClient();
string serviceURL = string.Format("http://localhost:8000/Service1/PutData", val);
string rezStr = webClient.UploadString(serviceURL, "PUT", val);
```



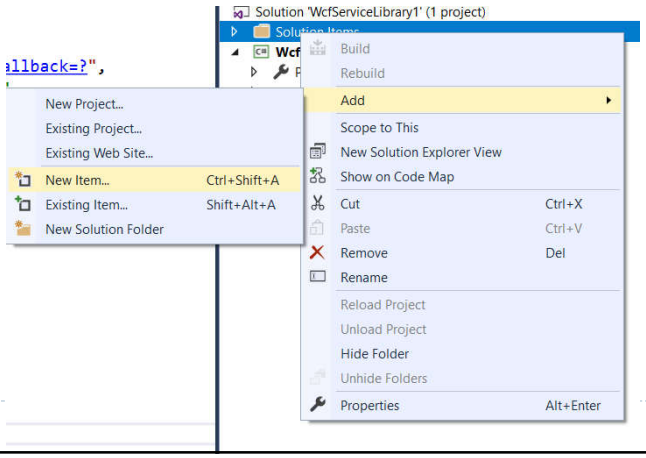
Dodavanje web klijenta

Pristup od strane web klijenta

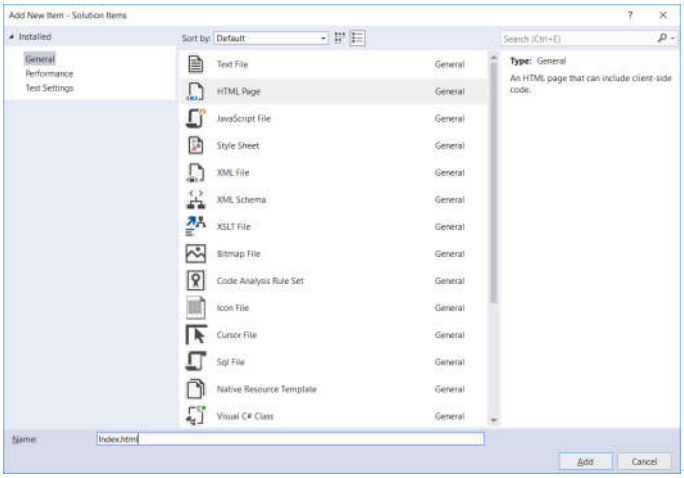
- Servis koji se pokreće pri testiranjima, pokreće se na localhost-u na odgovarajućem portu a uz pomoć pomoćnog IIS servera.
- Projekat servisa odnosno sam servis nije predviđen da sadrži HTML stranice. HTML stranice koje bi se koristile za testiranje moraju biti posebno napisane van ovako napisanog projekta.
- U praksi, servisi mogu biti hostovani od web servera i deo su neke veće web aplikacije, a HTML stranice najčešće same pripadaju istoj aplikaciji odnosno istom domenu.
- Napomena: Pristup servisima van istog domena nisu omogućeni od strane severa bez posebnog naznačavanja.

Postupak dodavanja HTML stranice kao klijentske aplikacije

- 1. Postojećem Solution-u, a van projekta servisa, dodajmo HTML dokument.

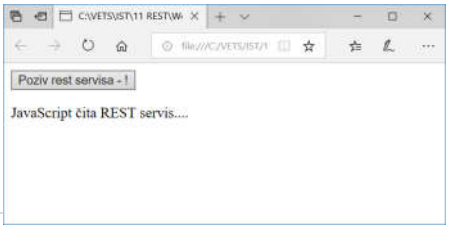


- 2. Zatim odaberimo tip dokumenta i njegov naziv:

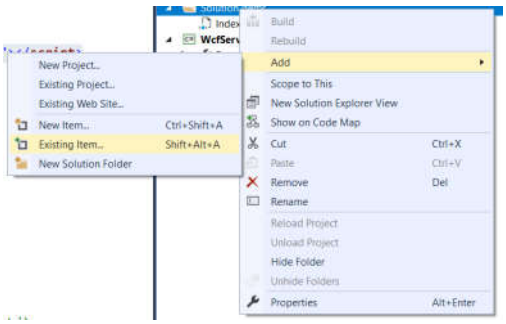


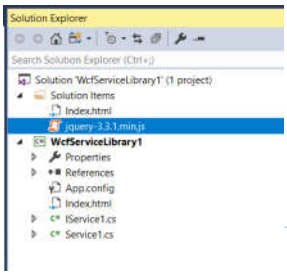
- Dodati HTML kod za stranicu. Neka to bude samo jedno dugme preko koga se obraćamo nekom servisi i ispisujem dobijene podatke.

```
<body>
    <button id="actionArgJSON">Poziv rest servisa - !</button>
    <div id="info"></div>

    <p id="demo">JavaScript čita REST servis....</p>

</body>
```

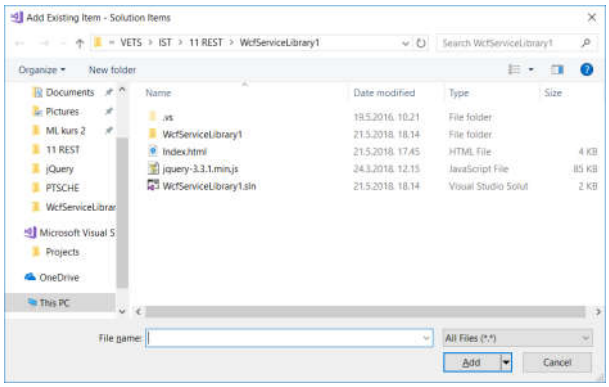


- Zatim uključimo jQuery biblioteku. Pretpostavimo da smo je prethodno preuzeli i smestili negde u našem fajl sistemu. Sledi:
- Nakon toga se pojavljuje u prozoru Solution Ex.





- Fajl koji se uključuje potrebno je da bude na određenoj lokaciji, u ovom slučaju na istoj lokaciji gde i Index.html, pogledati sliku.



- Zatim dodati odgovarajući JS Ajax kod:

```
$(document).ready(function () {
    $('#actionArgJSON').click(function () {
        $.ajax({
            cache: false,
            type: "GET",
            dataType: "json",
            contentType: "application/json; charset=utf-8",
            data: "222",
            url: "http://localhost:8733/Service1/web/GetDataString/15?callback=?",
            //url: "http://localhost:8733/Service1/web/GetDataString/15",
            success: function (primam) {
                alert(JSON.stringify(primam));
            },
            error: function (status) {
                alert(status);
                $('#info').html('<p>Dogodila se greška</p>');
            }
        });
    });
}
```

- Ili odgovarajući jQuery ajax poziv:

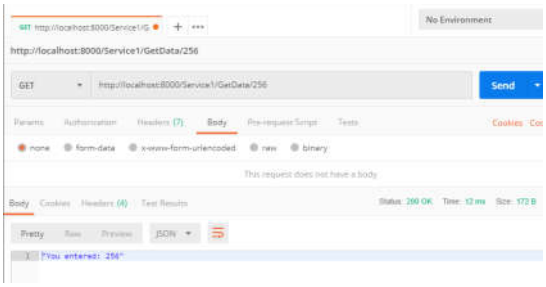
```
$(document).ready(function () {
    $('#actionArgJSON').click(function () {
        $.getJSON('http://localhost:8733/Service1/web/GetDataString/17?callback=?',
                  'nebitno', function (primam) {
        //$.getJSON('http://localhost:8733/Service1/web/GetDataString/17',
                    'nebitno', function (primam) {
            alert(JSON.stringify(primam));
        });

}
)
```

Kompletan primer/klijent - Init

```
$(document).ready(function () {
    $('#init').click(function () {
            $.getJSON('http://localhost:8000/Service1/Init',"", function (primljeno) {
                alert(primljeno);
        });

    })
})
```

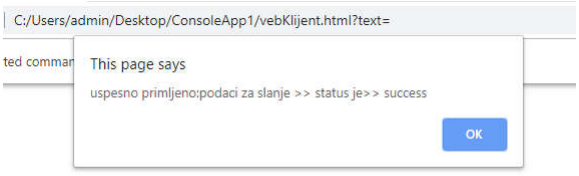
Kompletan primer/klijent - GetData

```
$(document).ready(function () {
    $('#init').click(function () {
        $.get('http://localhost:8000/Service1/GetData/256',"", function (primljeno) {
            alert(primljeno);
        });

    })
})
```



Kompletan primer/klijent - InitPost

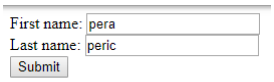
```
$(document).ready(function () {
    $('#init').click(function () {
        $.post('http://localhost:8000/Service1/InitPost', "podaci za slanje", function (primljeno,
status) {
            alert(`${primljeno} >> status je>> ${status }`);
        });

    })
})
```



Kompletan primer/klijent - PutData

```
$(document).ready(function () {
    $('#init').click(function () {
        $.ajax({
            cache: false,
            type: "PUT",
            contentType: "text; charset=utf-8",
            data: "66",
            url: "http://localhost:8000/Service1/PutData",
            success: function (primam) {
                alert(primam); },
            error: function (status) {
                alert("dogodila se greska:" + status); }
        });
    });
})
```

Slanje sa web forme




```
<form action="http://localhost:8000/Service1/InitPost" method="post">
    First name: <input type="text" name="fname"><br>
    Last name: <input type="text" name="lname"><br>
    <input type="submit" value="Submit">
</form>
```

```
"uspesno primljeno:fname=pera&lname=peric"
```

Izdvanje posebnih podataka

```
//ukljuciti biblioteku System.Web
public string InitPost(Stream podaci)
{
    StreamReader reader = new StreamReader(podaci);

    NameValueCollection nvc = HttpUtility.ParseQueryString(reader.ReadToEnd());

    string rez = "ime=" + nvc["fname"] + " prezime=" + nvc["lname"];

    return rez;
}
```

```
"ime=pera prezime=peric"
```

Važno

- Napomena1: Obratiti pažnju na način kako je urađen poziv. Način je drugačiji u odnosu na očekivani koji je komentarisan.
- Što se tiče klasičnog načina primene AJAX metoda u HTML stranicama, možemo videti već u naredom delu koji se tiče web api projekata u okviru web aplikacija.
- Napomena2: Primenom Chrome web čitača moguće je uz kreiranje pomoćnog foldera, na primer: c:\chrome, ukoliko već ne postoji. Zatim preko *run* komande pokrenuti chrome sa opcijama, na primer:
- "C:\Program Files (x86)\Google\Chrome\Application\chrome.exe" --disable-web-security --user-data-dir="c:\chrome"

Dodatak

Dodatak-Određivanje resursa - 1

Razmatrajmo slučaj realizacije REST servisa koji treba da radi sa bookmark objektima. Neka imamo ili formiramo najpre listu operacija odnosno listu koja obezbeđuje podatke i šalje potrebne podatke

createUserAccount	Kreiranje novog korisničkog naloga
getUserAccount	Vraćanje detalja o nalogu
updateUserAccount	Ažuriranje korisničkog naloga
deleteUserAccount	Brisanje korisničkog naloga
getUserProfile	Vraćanje javnih podataka korisničkog profila
createBookmark	Kreiranje nove Creates a new bookmark for the authenticated user
updateBookmark	...
deleteBookmark	
getBookmark	
getUserBookmarks	
getUserPublicBookmarks	
getPublicBookmarks	

Određivanje resursa - 2

- Zatim je važno prepoznati objekte tj. resurse nad kojima ćemo imati 4 operacije koje treba da obezbede sve što je potrebno u navedenim metodama.
- Na osnovu navedenih operacija objekti mogu biti:

1. Users
2. Bookmarks

- U okviru ovih objekata potrebno je obezbediti rad sa:

a) Pojedinačnim nalogom
b) Pojedninačnim profilom
c) Pojedinačnim bookmark-om.
d) Kolekcijom privatnih bookmark objekata.
e) Korisničkom kolekcijom javnih bookmark objekata.
f) Kolekcijom svih javnih bookmark objekata.

Određivanje resursa - 3

- Pošto su definisani resursi i potrebne operacije nad njima, sledi definisanje URI identifikatora za svaki. (Pošto su resursi na WEBu onda se koristi URI)
- Neka se servis za bookmark objekte hostuje na `http.testadr.rs/rest/pr2/bookmarkservice` onda se, na primer, ista adresa može koristiti za vraćanje svih javnih bookmark objekata. Pošto takvih objekata može biti jako puno, treba omogućiti filtriranje dodavanjem nastavka na osnovni URI: `?tag={tag}`
- Vrednost u vitičastim zagradama je promenljiva dok je ostatak URI fiksan. Na primer, ako se filtrira bookmark naziva `wcf`, tada je URI: `http.testadr.rs/rest/pr2/bookmarkservice?tag={wcf}`

Određivanje resursa - 4

- Razumno je pretpostaviti da korisnički podaci mogu biti javni i privatni. U slučaju da pristupamo javnim podacima URI možemo da modifikujemo tako da posle adrese servisa stoji identifikator korisnika tj. korisničko ime. Dakle, za javni pristup URI bi bio `{username}?tag={tag}`
- Zadržan je isti koncept naveden za prethodni slučaj, tj. ako na kraju stoji tag onda se podaci dodatno filtriraju po osnovu tog tag-a.

- Na primer:
- `.../bookmarkservice/pperica?tag={wcf}`
- vraća javne bookmark objekte. `pperica` predstavlja korisničko ime, a ceo URI se odnosi na bookmark objekte korisnika korisničkog imena *pperica* a koji su označeni (filtrirani) sa `wcf`

Određivanje resursa - 5

- Osim javnih, razmotrimo i privatne podatke, tj. podatke koji su dostupni samo korisniku. Pravila bi trebalo da budu što generalnija. Dakle, na isti način, da obezbede filtriranje po tag-u, ali i dobavljanje podataka o profilu korisnika. Jedan izbor bi bio da se na kraj URI servisa postavi sufiks `users` a nakon njega korisničko ime korisnika - `username`. Ovaj URI se razlikuje po nastavku `users` koji sledi iza URI adrese servisa i koji ukazuje na privatne podatke. Dakle:
- `/users/{username}` što se odnosi na privatne podatke korisnika, a u nastavku URI slede sufiksi koji identifikuju koji se podaci za tog korisnika očekuju:
  - `/users/{username}/profile`
  - `/users/{username}/bookmarks`
  - `/users/{username}/bookmarks?tag={tag}`
  - `/users/{username}/bookmarks/{id}`

Određivanje resursa - Konačna šema

`users/{username}`
- PUT - createUserAccount
- GET - getUserAccount
- PUT - updateUserAccount
- DELETE - deleteUserAccount

`users/{username}/profile`
- GET – getUserProfile

`users/{username}/bookmarks`
- POST-createBookmark

`users/{username}/bookmarks/{id}`
- PUT – updateBookmark
- DELETE – deleteBookmark
- GET – getBookmark

`users/{username}/bookmarks?tag={tag}`
- GET - getUserBookmarks

Javni pristup:

`{username}?tag={tag}`
- GET - getUserPublicBookmarks

`?tag={tag}`
- GET - getPublicBookmarks

Primer: Testiranje Uri, UriTemplate

Primer:

```
static void Main(string[] args){
    Uri baseUri = new Uri("http://contoso.com/bookmarkservice");
    UriTemplate uriTemplate = new UriTemplate("users/{username}/bookmarks/{id}");
    Uri newBookmarkUri = uriTemplate.BindByPosition(baseUri, "skonnard", "123");
    Console.WriteLine(newBookmarkUri.ToString());
    UriTemplateMatch match = uriTemplate.Match(baseUri, newBookmarkUri);
    System.Diagnostics.Debug.Assert(match != null);
    Console.WriteLine(match.BoundVariables["username"]);
    Console.WriteLine(match.BoundVariables["id"]);
}
```

- Klasa UriTemplateTable poseduje mehanizam za upravljanjem kolekcije UriTemplate. Ovom klasom se omogućava primena metode Match koja nalazi sve šablone koji odgovaraju ponuđenom Uri. Moguće je koristiti i metodu MatchSingle za testiranje samo jednog UriTemplate objekta iz tabele.
- WCF “Web” način programiranja olakšava mapiranje objekta UriTemplate u odnosu na korišćene potpise metoda pomoću atributa [WebGet] odnosno [WebInvoke].