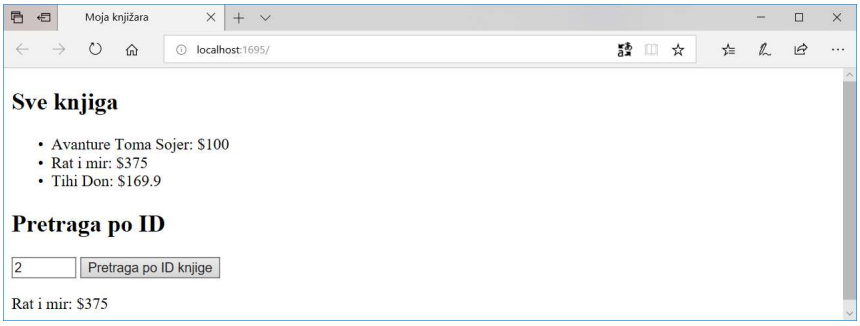
14. Web Api

HTTP je protokol namenjen ne samo za dostavljanje veb sadržaja u vidu stranica. HTTP je moćna platforma i za izgradnju API-ja koji nude usluge i podatke. HTTP je jednostavan, fleksibilan i sveprisutni. Gotovo svaka platforma ima biblioteke za HTTP. HTTP servisi mogu da ostvare širok spektar klijenata, uključujući pregledače, mobilne uređaje kao i tradicionalne desktop aplikacije.

ASP.NET Web API je radni okvir za izgradnju veb API-ja u .NET Framework. U nastavku pokazaćemo način kreiranja ASP.NET Web API koji formira listu knjiga.

Kreiranje Web API projekta

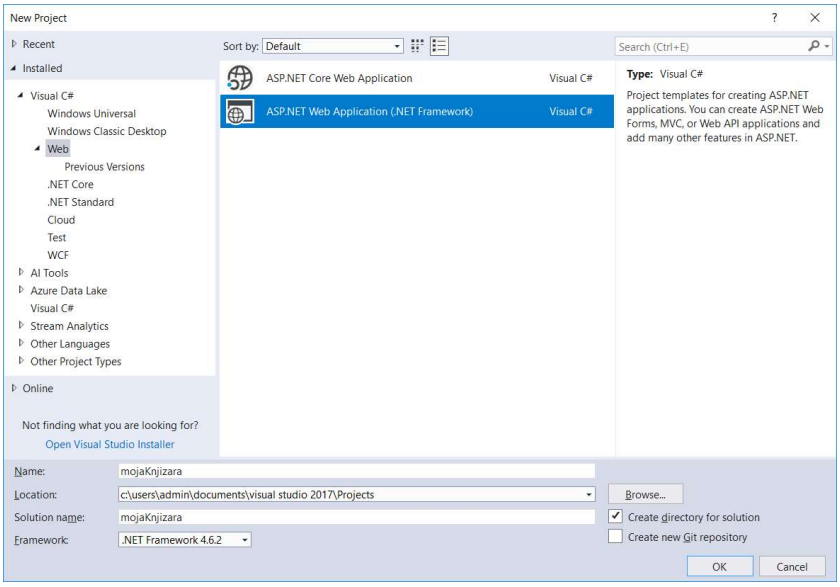
U ovom predavanju, koristićete `ASP.NET Web API` za kreiranje veb API koji daje listu knjiga. Veb stranica koristi jQuery pristupanje api funkcijama i vraćanje rezultata.



Slika 14.1. Prikaz stranice koju kreiramo primenom Web API

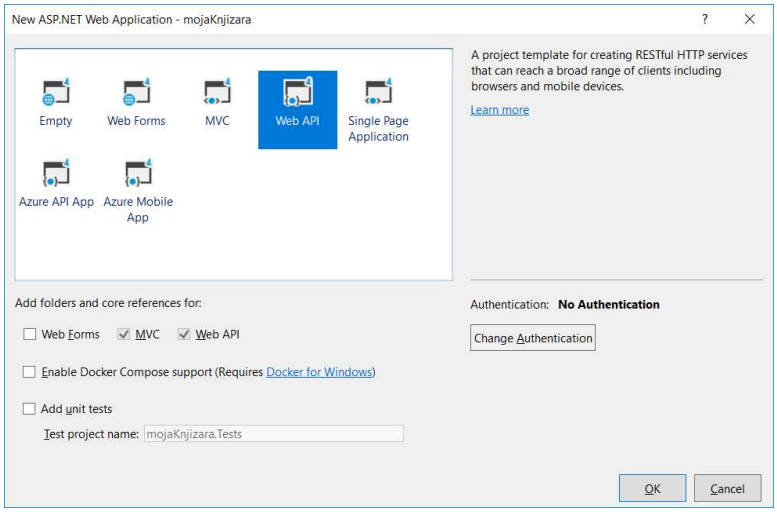
Pokrenimo Visual Studio i odaberimo **New Project**.

U delu `Templates` odaberimo `Installed Templates` zatim proširimo čvor **Visual C#**. Pod opcijom **Visual C#**, odaberimo **Web**. U listi šablona projekta odaberimo **ASP.NET Web Application**. Imenujmo projekat "mojaKnjizara" a zatim **OK**.



Slika 14.2. Formiranje novog projekta

U dijalogu **New ASP.NET Project** odaberite šablon **Empty**. Odaberite **Web API**. Klik na **OK**.



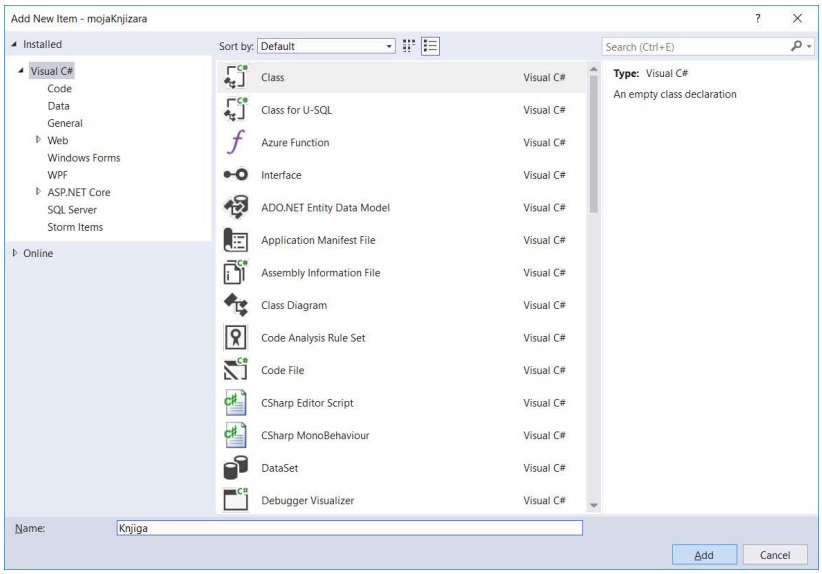
Slika 14.3. Izbor šablona za novi projekta

Dodavanje Modela

Model je objekat koji predstavlja podatke u aplikaciji. ASP.NET Web API može automatski da serijalizuje model u formate kao što je JSON, XML. A zatim se takvi podaci mogu ubaciti u telo poruke koja je HTTP odgovor. Većina klijenata može da analizira ili XML ili JSON. Štaviše, možete naznačiti format koji želite da se prihvati kroz zaglavlje HTTP zahteva za poruke.

Kreirajmo sada jednostavan model.

U prozoru `Solution Explorer`, desni-klik na folder Models, a zatim iz kontekstnog menija birati opciju **Add** a zatim odabrati **Class**.



Slika 14.4. Dodavanje nove klase za model

Imenujmo klasu "Knjiga". Zatim dodajmo sledeća svojstva ovoj klasi.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;

namespace mojaKnjizara.Models
{
    public class Knjiga
    {
        public int Id { get; set; }
        public string Naziv { get; set; }
        public string Kategorija { get; set; }
        public decimal Cena { get; set; }
    }

}
```

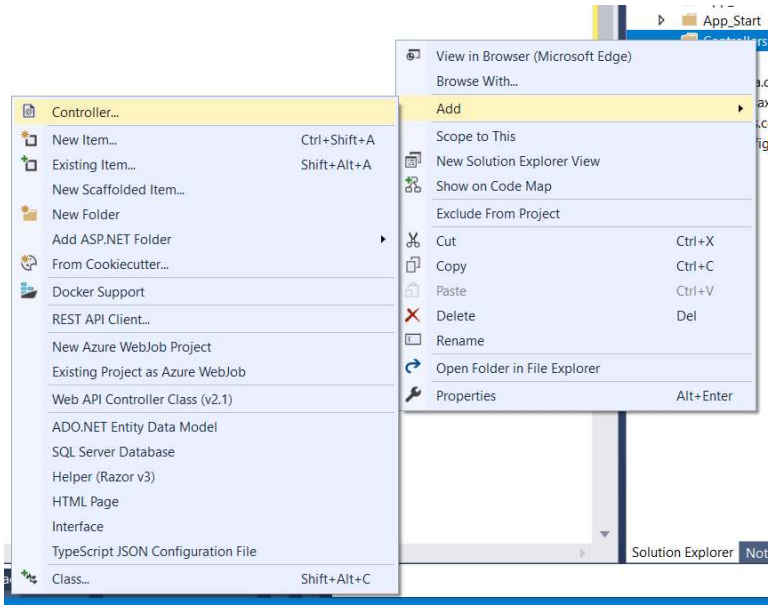
Dodavanje kontrolera

U Web API, kontroler je objekat koji rukuje HTTP zahtevom. Dodaćemo kontroler koji može da vrati listu proizvoda ili jedan proizvod a koji je zahtevan na osnovu prosleđenog Id podatka.

Napomena

Ako ste već koristili ASP.NET MVC, onda ste već bliski i koristili ste kontrolere. Web API kontroleri su slični MVC kontrolerima, s tim što nasleđuju klasu `ApiController` umesto klase `Controller`.

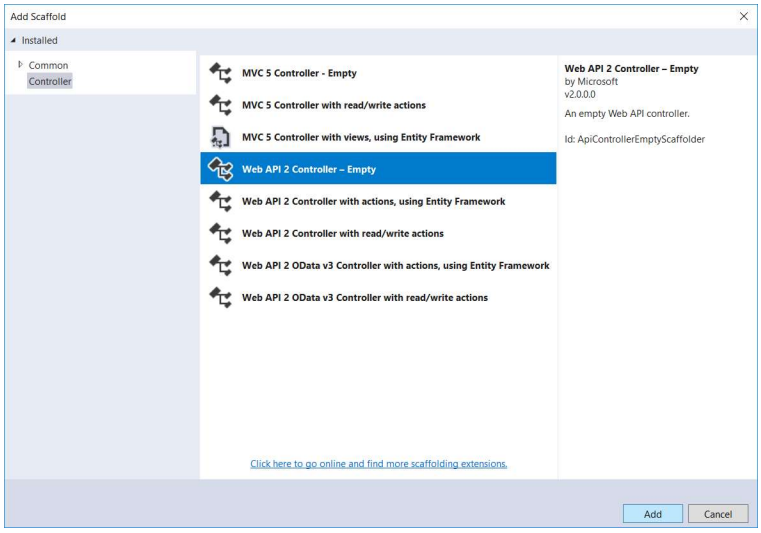
Korak 1. U prozoru `Solution Explorer`, desni-klik na Controllers folder, zatim odaberite **Add** a zatim odaberite **Controller**.



Slika 15.5. Dodavanje kontrolera

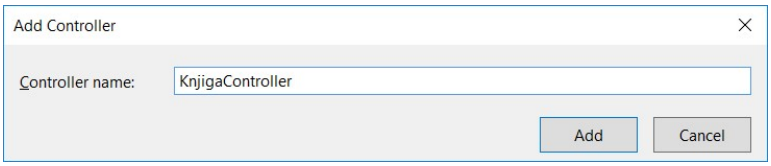
Napomena: Kreiranje WebApi kontrolera moguće je i izborom opcije Web Api Controller Class

Korak 2. Odaberite **Web API Controller - Empty**. a zatim **Add**.



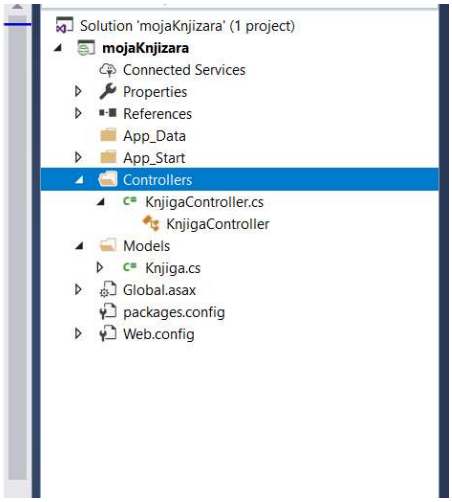
Slika 14.6. Izbor kontrolera

Korak 3. U dijalogu **`Add Controller`** imenujte kontroler **`KnjigaController`**. Odaberite **`Add`**.



Slika 14.7. Definisanje naziva kontrolera

Mehanizam **`scaffolding`** kreira jedan fajl naziva **`KnjigaController.cs`** u folderu **`Controllers`**.



Slika 14.8. Pogled na prozor `Solution Explorer`

Napomena: Nije neophodno da se kontroler postavi u folder koji se naziva Controllers. Izbor naziva foldera je pitanje konvencije i organizacije fajlova u projektu.

Ako ovaj fajl nije već automatski otvoren, dvostrukim-klikom na fajl on će se otvoriti za editovanje. Uredite kod na način kako je to prikazano u nastavku:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Net;
using System.Net.Http;
using System.Web.Http;
using mojaKnjizara.Models;

namespace mojaKnjizara.Controllers
{
  public class KnjigaController : ApiController
  {
    // Kreiranje podataka
    // Kasnije ovi podaci se preuzimaju iz neke baze
    Knjiga[] knjige = new Knjiga[]
    {
      new Knjiga { Id = 1, Naziv = "Avanture Toma Sojer",
Kategorija = "Dečiji", Cena = 100 },
```

```
      new Knjiga { Id = 2, Naziv = "Rat i mir", Kategorija
= "Klasika", Cena = 375 },
      new Knjiga { Id = 3, Naziv = "Tihi Don", Kategorija =
"Klasika", Cena = 169.9M }
    };

    [System.Web.Http.HttpGet]
    public IEnumerable<Knjiga> SveKnjige()
    {
      return knjige;
    }

    [System.Web.Http.HttpGet]
    public IHttpActionResult Knjiga(int id)
    {
      var knjiga = knjige.FirstOrDefault((p) => p.Id == id);
      if (knjiga == null)
      {
          return NotFound();
      }
      return Ok(knjiga);
    }
    [HttpGet]
    public IHttpActionResult Knjiga(int id)
    {
      var knjiga = knjige.FirstOrDefault((p) => p.Id == id);
      if (knjiga == null)
      {
          return NotFound();
      }
      return Ok(knjiga);
    }

    [HttpPost]
    public string Post()
    {
      var naslov =
HttpContext.Current.Request.Params["naslov"];
      var kategorija =
HttpContext.Current.Request.Params["kategorija"];
      return "OK";
    }
  }
}
```

Kontroler definiše dve metode koje vraćaju knjige:

- Metoda `SveKnjige` vraća celu listu knjiga kao neki `IEnumerable<Knjiga>` tip.
- Metod Knjiga pronalazi jednu knjigu iz liste knjiga koje postoje.

Svaki metod na kontroleru odgovara jednom ili više URL-ova:

U primeru se koristi pojednostavljen primer u kome se knjige formiraju u kontroleru tako što ih kreiramo neposredno u kodu. Umesto toga u realnom slučaju, to bi bilo preko pristupa nekoj bazi podataka ili drugom spoljnjem izvoru podataka odnosno dobavljanjem podataka iz nje.

Tabela 14.1. Primer povezivanja metode i URL-a

Metod kontrolera	URI
SveKnjige	/api/Knjiga
Knjiga	/api/Knjiga/*id*

Za metodu Knjiga argument id je u URI. Na primer, da bi se dobila knjiga čiji je `ID = 5`, odgovarajući URI je: `api/knjiga/5`.
Adresa na jedinstven način opisuje pristup do servisa i za jednu adresu može postojati jedan GET metod. Ukoliko metoda ima naziv `Get` onda nije neophodno da se postavlja dekorator `HttpGet` iznad naziva metode. Ako se koristi HttpGet onda se metodi koja prihvata Get zahtev treba dodati i dekorator za naziv akacije, na primer:

```
ActionName("sveknjige")]
```

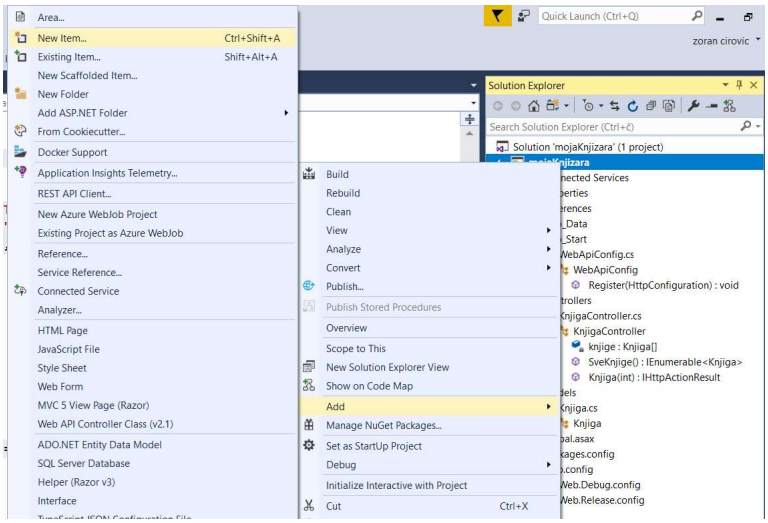
U ovom slučaju, adresa kojim se definiše Get zahtev bila bi:
`http://localhost:xxx/api/nazivKontrolera/sveknjige`

Veb API ima u kreiranom projektu podrazumevano definisan način rutiranja URI ka metodama, za više informacija pogledati: `https://docs.microsoft.com/en-us/aspnet/veb-api/overview/veb-api-routing-and-actions/routing-in-aspnet-veb-api`.

JavaScript / jQuery pozivi

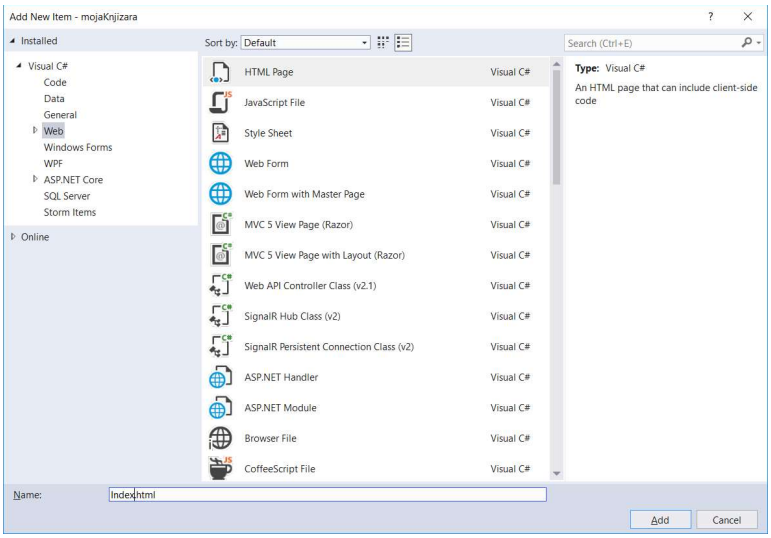
U ovom poglavlju videćemo kako da dodamo neku HTML stranicu koja koristi AJAX da bi pozvala veb API. Koristićemo jQuery kako bi napravili AJAX poziv a zatim ažurirali stranicu sa rezultatom tj knjigama.

Korak 1. U prozoru `Solution Explorer`, desni-klik na projekat a zatim odaberite **Add**, zatim odaberite **New Item**.



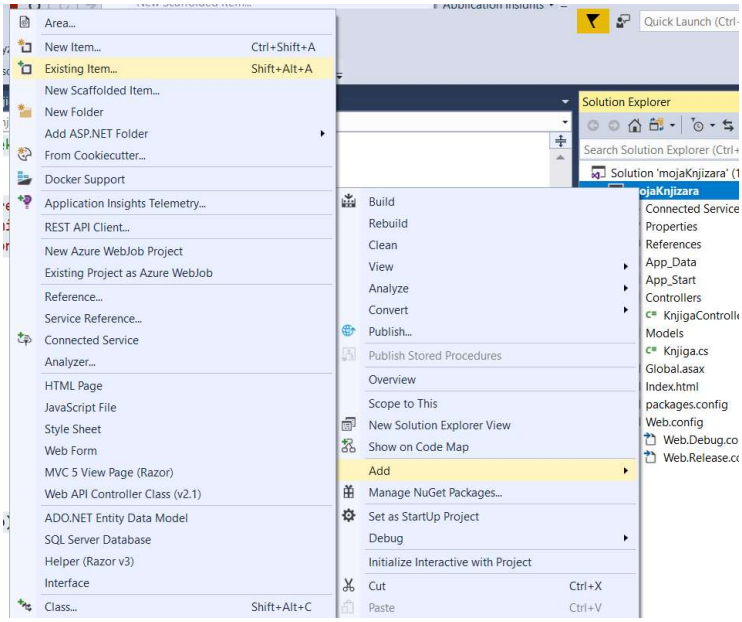
Slika 14.9. Dodavanje nove stavke

Korak 2. U dijalogu **Add New Item** odaberite čvor **Web** pod opcijom **Visual C#**, a zatim odaberite **HTML Page**. Nazovite stranicu `Index.html`.



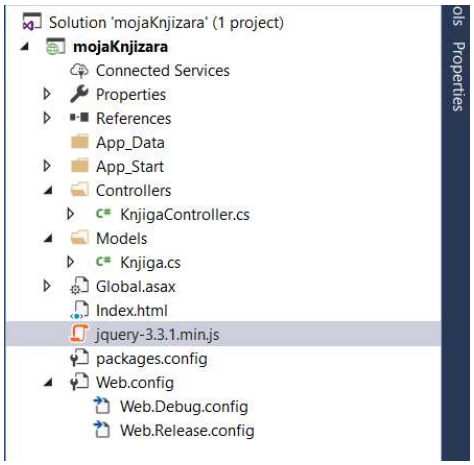
Slika 14.10. Izbor HTML stranice

Postoji više načina kako da se uključi biblioteka jQuery u veb stranicu. U ovom primeru korišćen je Microsoft Ajax CDN. Moguće je preuzeti celu biblioteku sa lokacije http://jquery.com/, a zatim je uključiti u projekat, kao na slici.



Slika 14.11. Uključivanje postojeće biblioteke projektu

Nakon dodavanja, u Project folderu pojavljuje se ubačeni `js` document, kao na slici:



Slika 14.12. `Solution Explorer` prikaz dodate biblioteke

A zatim isti možete koristiti i u aplikaciji:

```
<!--<script
src="https://ajax.aspnetcdn.com/ajax/jQuery/jquery-
3.3.1.min.js"></script>-->

<script src="jquery-3.3.1.min.js"></script>
```

Testiranje Get metoda

Get zahtev se može testirati neposredno preko web čitača unosom adrese metode u adres-bar čaitača. U stranicu index.html može se dodati kod za testiranje Get poziva.

Koristeći form element:

```
<form action="http://localhost:4713/api/knjiga" method="GET">
    <button id="sendGetSveKnjige1" type="submit">Get</button>
</form>
```

Koristeći Ajax – primer 1:

```
$("#btnGet").on('click', function () {
    alert("GET Request Sent");
```

```
    var request = $.ajax({
        url: "http://localhost:4713/api/knjiga",
        method: "GET",
        dataType: "json"
    }).done(function (msg) {
       console.log(msg);
   }).fail(function (jqXHR, textStatus) {
       console.log(jqXHR, textStatus);
   });
});
```

U prvom slučaju odgovor sistema je vidljiv na stranici, a u drugom se može ispitati u konzolnom prozoru (F12).

Primena u HTML stranicama

Lista

Dakle, da bi se dobila lista knjiga potrebno je da se pošalje HTTP GET zahtev na URI: "/api/knjiga". jQuery funkcija **`getJSON`** šalje AJAX zahtev. Za odgovor očekuje se niz JSON objekata. Funkcija ***done*** definiše povratnu funkciju tzv. **`callback`** koji se poziva kada je zahtev obavljen uspešno. U povratnoj metodi menjamo DOM elemente uključujući povratne informacije:

```
$(document).ready(function () {
    $.getJSON(uri)
        .done(function (data) {
            $.each(data, function (key, item) {
                $('<li>',    {    text:    formatItem(item)
}).appendTo($('#sveknjigeUL'));
            });
        });
});
```

Jedan podatak

Da bi se dobila knjiga po ID, potrebno je da korisnik učita ID I klikom na dugme pošalje zahtev tipa HTTP GET na adresu "/api/knjige/*id*", gde je *id* ID od knjige koja se traži:

```
function find() {
    var id = $('#knjigaId').val();
    $.getJSON(uri + '/' + id)
        .done(function (data) {
            $('#knjigaP').text(formatItem(data));
        })
        .fail(function (jqXHR, textStatus, err) {
            $('#knjigaP').text('Error: ' + err);
        });

}
```

Testiranje Post metoda

Post metodom klijent šalje podatke preko tela poruke. Serverska strana samo jednom čita podatke. Postoji nekoliko mogućih realizacija na serveskoj odnosno klijentskoj strani pogledajmo neke od njih.

Serverska metoda je bez argumenata. Ukoliko je naziv metode drugačiji od Post onda se dodaje dekorator `[HttpPost]` za ovu metodu.

```
public string Post()
{
    var naslov =
HttpContext.Current.Request.Params["naslov"];
    var kategorija =
HttpContext.Current.Request.Params["kategorija"];

    return "OK";
}
```

Ukoliko se šalje jedan podatak drugi će na prijemu biti `null`. Pogledajmo tri primera zahteva.

```
<form action=". . ./api/knjiga" method="POST">
    <div><input type="text" name="naziv"></div>
    <div><button type="submit">POST</button></div>
</form>
```

```
<form action=". . ./api/knjiga" method="POST">
    <div><input type="text" name="naziv"></div>
    <div><input type="text" name="kategorija"</div>
    <div><button type="submit">POST</button></div>
</form>
```

```
var knjigaZaSlanje = {
    "Id":"33",
    "Naziv":"Pesme",
    "Kategorija":"Poezija",
    "Cena":444.44
}
$("#post").on('click', function () {
    alert("POST Request Sent");
    var request = $.ajax({
        url: "http://localhost:4713/api/knjiga",
        method: "POST",
        data: knjigaZaSlanje,
        dataType: "xml"
    })
   .done(function (msg) {
       console.log(msg);
   })
   .fail(function (jqXHR, textStatus) {
       console.log(jqXHR, textStatus);
   });
});
```

Prvi se odnosi na slanje iz forme gde se šalje samo jedan podatak, drugi sa poslata dva podataka a treći preko Ajax-a. Sva tri slanja mogu se prihvatiti metodom koja je navedena.

Slanje kompleksnih tipova

Kompleksni objekti se mogu poslati preko forme ili koristeći Ajax. U oba slučaja možemo na serverskoj strani obezbediti prhvat taktvih objekata u

celosti. Naravno, da bi se primljeni podaci na serveru mogli prebaciti u objekat željenog tipa neophodno je da postoji deserijalizacija za taj objekata. Dakle, na serverskoj strani dovoljno je samo da se navede objekat koji se očekuje u prihvatnoj metodi.

```
public HttpResponseMessage Post(Knjiga k)
{
    if (ModelState.IsValid && k != null)
    {
        //// Create a 201 response.
        var response = new
HttpResponseMessage(HttpStatusCode.Created)
        {
            Content = new StringContent("OOKK")
        };
        return response;
    }
    else
    {
        return
Request.CreateResponse(HttpStatusCode.BadRequest);
    }
}
```

Ovakva metoda prihvata objekte koji imaju svojstva kojima je opisan objekat Knjiga. Na ovaj način se kreira očekivani objekat u celosti. Ukoliko nedostaju neke vrednosti objekat ima svojstva koja dobijaju podrazumevane vrenosti (npr. null ili 0)

Testiranja na klijenstoj strani koja su pokazana za slučaj metode rada sa metodom Post() mogu se koristiti i u ovom slučaju.

Slanje prostih tipova

Ako se šalje samo jedan podatak koristeći Post metodu i taj podatak mora biti pripremjen na strani klijenta u vidu objekta pri čemu se kao ključ ne navodi ništa: `{"": "Tom Sojer"}`. Na primer:

```
post2
.on('click', function () {
    alert("POST2 Request Sent");
```

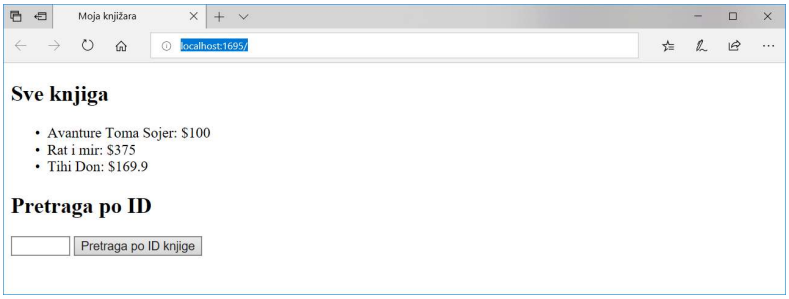
```
    var request = $.ajax({
        url: "http://localhost:4713/api/values",
        method: "POST",
        data: samoTekst, //var samoTekst={"":"Tom Sojer"};
        dataType: "json"
    })
    .done(function (msg) {
        console.log(msg);
    })
    .fail(function (jqXHR, textStatus) {
        console.log(jqXHR, textStatus);
    });
});
```

Za ovako poslate podatke koji su prosti postoji mala modifikacija u kodu:

```
public void Post([FromBody]string value)
```

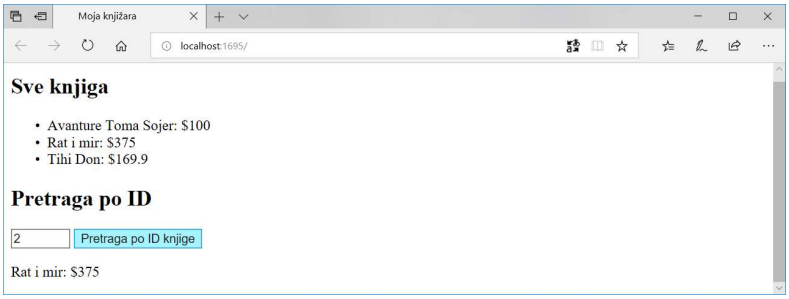
Testiranje aplikacije

Klikom na skraćenicu F5 pokreće se aplikacija sa mogućnošću debagovanja, **Debug mode**. Stranica izgleda na sledeći način:



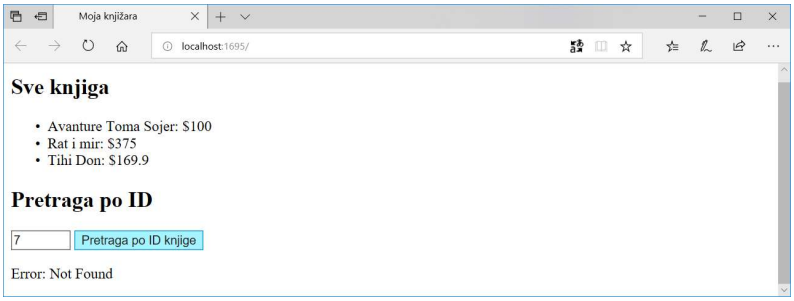
Slika 14.13. Prikaz početne stranice

Da bi se dobila knjiga po ID, unese se vrednost u polje klikne na dugme za pretragu:



Slika 14.14. Prikaz rezultata pretrage

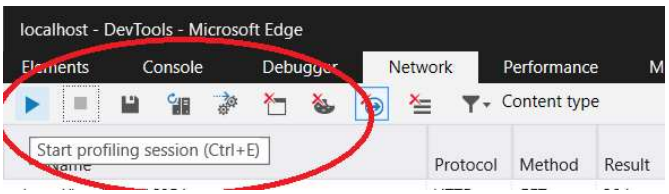
Ako se unese neispravan ID, server vraća HTTP grešku:



Slika 14.15. Prikaz pogrešnog zahteva za pretragu

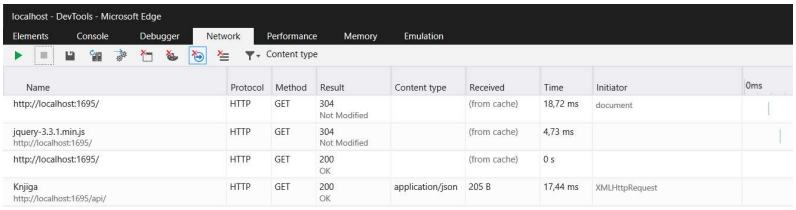
HTTP Request / Response

Kada radite sa HTTP servisima može biti veoma korisno da se vide HTTP zahtevi odnosno odgovori. To se može izvesti pomoću alatke za programere koja se pokreće tasterom **F12**. Odaberite karticu `Network`.



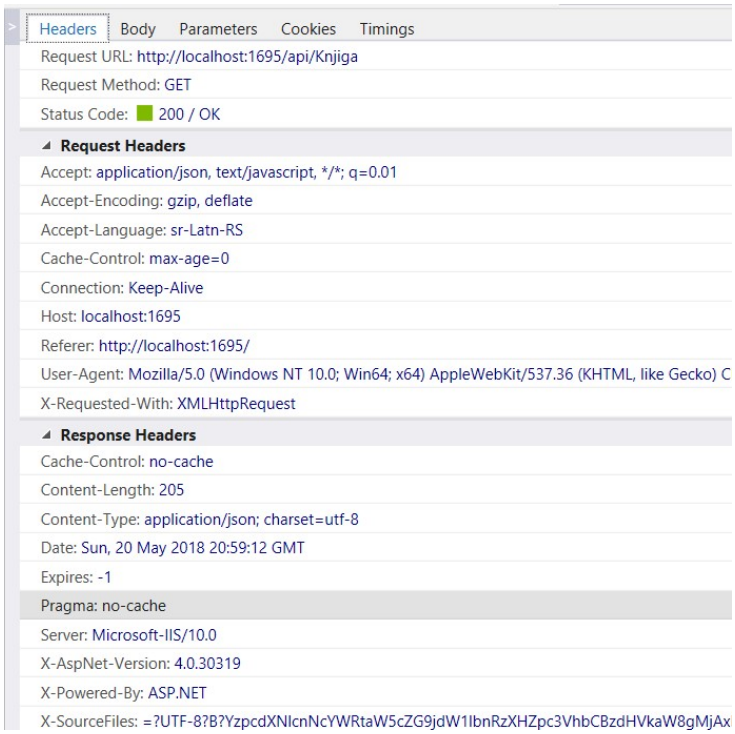
Slika 14.16. Kartice u veb čitaču korišćene za razvoj

zatim pritisnite dugme za „hvatanje“ poruka. Sada se vratite se na veb stranicu i pritisnite taster F5 da ponovo učitati veb stranicu. Veb čitač će izvršiti hvatanje HTTP saobraćaja između veb čitača i veb servera. Slika pokazuje mrežni saobraćaj za stranicu:



Slika 14.17. Network kartica

Nađite poziv za URI **`api/Knjiga/`**. Odaberite tu stavku, a zatim pogledajte detalje u desnom delu prozora. Među detaljima postoje tabovi koji prikazuju **`request`**/**`response`** zaglavlja i telo poruke.



Slika 14.18. Praćenje sadržaja poruka

Pitanja i zadaci za proveru znanja

1. Objasnite pojam Web Api aplikacija. Da li poznajete neki api?
2. Kreirajte novi projekat tipa Web Api. Da li možete birati MVC opciju pri kreiranju projekta?
3. Ubacite vaš model Radnik u ovaj projekat. Obratite pažnju na nazive kolona. Zašto?
4. Kreirati kontroler za podatke o radnicima.
5. Iz koje klase je izvedena klasa kontrolera?
6. Kreirajte metode koje vraćaju sve radnike odnosno jednog radnika.
7. Definišite URL za pozive kreiranih metoda.
8. Dodajte projektu HTML stranicu a zatim primenom JavaScript-a i jQuery jezika napišite forme za dobavljanje podataka preko vaših metoda.
9. Napraviti novi projekat koji je tipa ASP MVC aplikacije pa tom projektu dodajte kontroler i HTML stranicu kao u prethodnom slučaju.