

Napredne tehnike i principi razvoja softvera
Razvoj web servisa i serverske strane web aplikacija

TEMA 2:

REST KONTROLERI

MSc Ivan Vasiljević, ivan.vasiljevic@hotmail.com

MSc Mladen Kanostrevac, mkanostrevac@gmail.com



Pregled

- REST (step-by-step)
 - Napredni elementi kreiranja REST servisa u ASP.NET Web API-ju
 - Metode
 - Parametri
 - Postman
- Debugging
 - Breakpoints
- Zadaci

REST KONTROLERI

Napredni elementi kreiranja REST servisa u ASP.NET Web API-ju

Metode

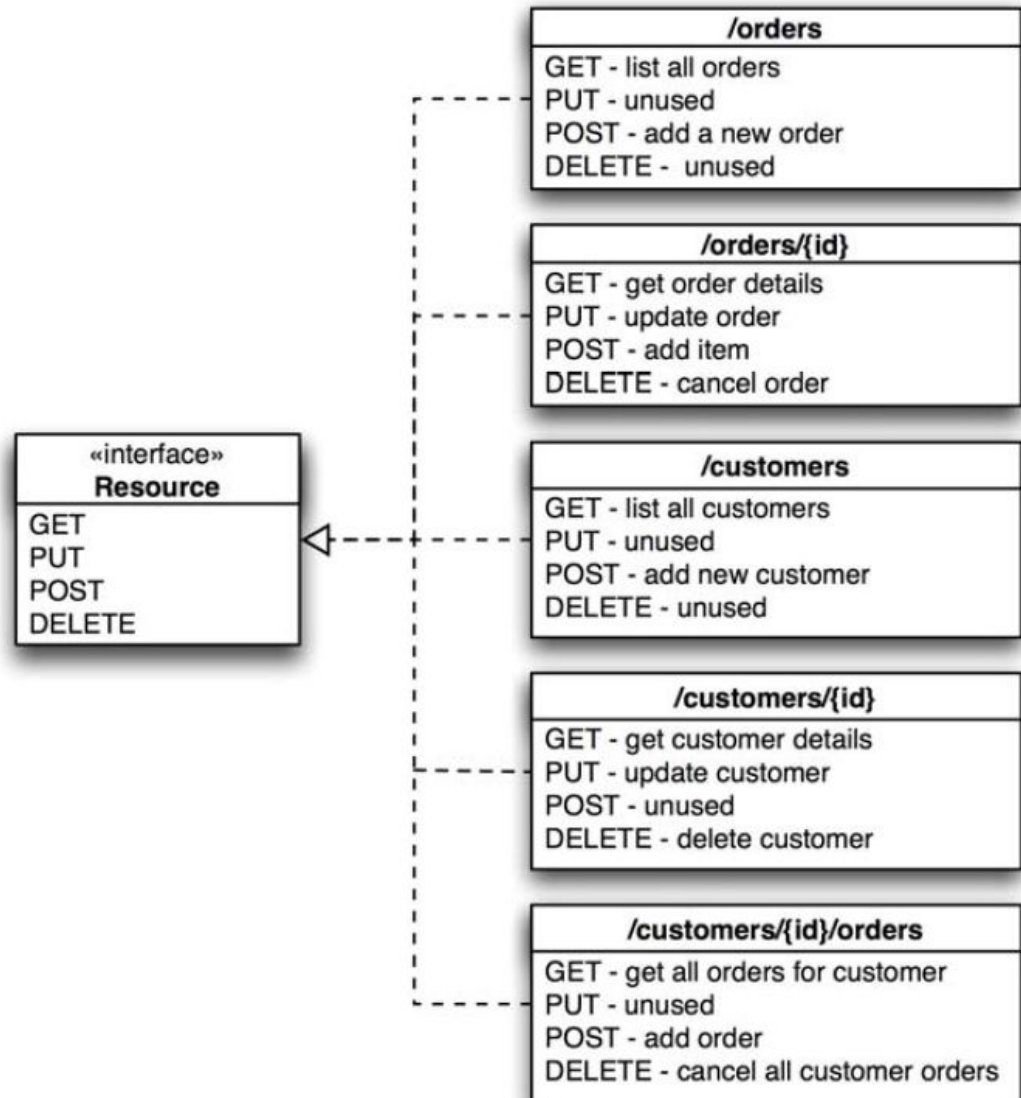
Parametri

REST – standardne metode

- Koristiti isključivo standardne HTTP metode za rukovanje resursima
 - GET, POST, PUT, DELETE ...
 - bez uvođenja novih metoda specifičnih za trenutni projekat
 - generički klijenti mogu samo da implementiraju standardne HTTP metode i da poznaju URI resursa
- Primeri:
 - dobavljanje klijenta sa ID-jem 1234
 - GET na `http://example.com/customers/1234`
 - brisanje klijenta sa ID-jem 1234
 - DELETE na `http://example.com/customers/1234`
 - dodavanje novog klijenta sa ID-jem 4321
 - POST na `http://example.com/customers/1234`
 - u telu zahteva specificirati vrednosti za sva polja klijenta

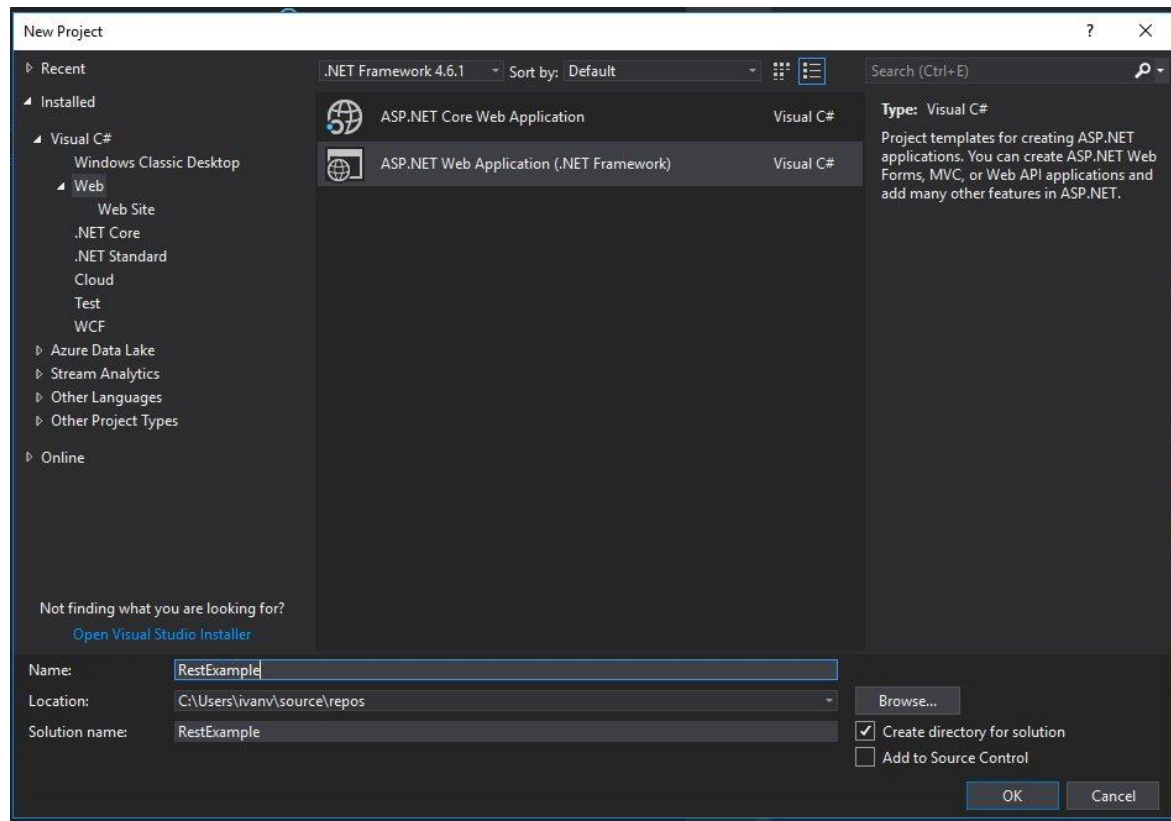
REST – standardne metode

- Standardni interfejs
 - npr. četiri metode
 - implementiraju ga svi servisi



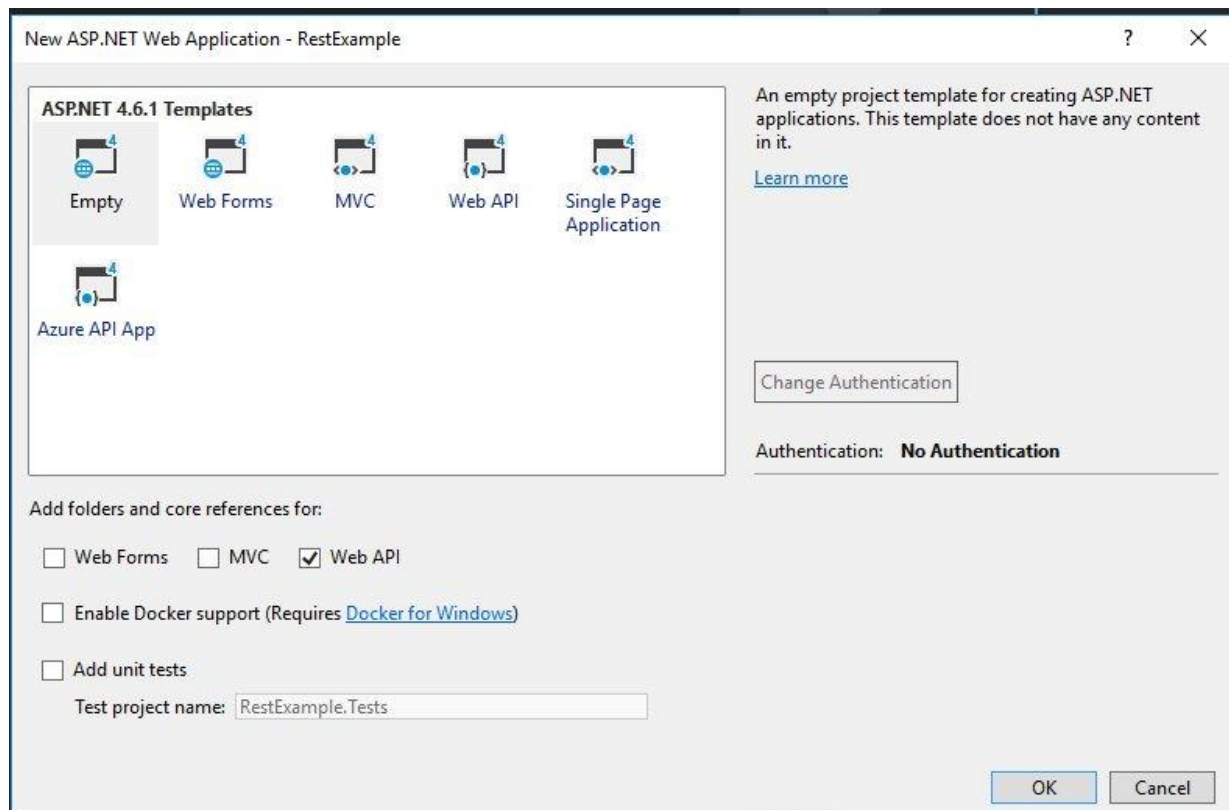
VS projekat

- Kreirati novi ASP.NET Web Application projekat
 - File -> New -> Project
ASP.NET Web Application(.NET Framework) project
 - **na prvom ekranu popuniti kao što je dato na slici**
 - nakon popunjavanja
pritisnuti dugme OK



VS projekat

- Kreirati novi ASP.NET Web Application projekat
 - na drugom ekranu:
 - odabrati **Empty project template**
 - selektovat **Web API**
 - pritisnuti **OK**



VS projekat – REST Kontroler

- desni klik na folder Controllers
 - Add -> Controller
 - odabrati Web API 2 Controller with read/write actions
 - nazvati je *BankClientRestController*
- *ApiController*
 - klasa koju kontroler treba da nasledi
 - sadrži korisne metode za kontrolera
- metode se zovu po HTTP metodi koju obrađuju
 - /help - *GetHelp*
 - ovo može da se promeni
- *FromBody* - označava da će parametar biti prosleđen u telu HTTP zahteva

VS projekat – REST kontroler

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

namespace RestExample.Controllers
{
    public class BankClientRestController : ApiController
    {
        // GET: api/BankClientRest
        public IEnumerable<string> Get()
        {
            return new string[] { "value1", "value2" };
        }

        // GET: api/BankClientRest/5
        public string Get(int id)
        {
            return "value";
        }

        // POST: api/BankClientRest
        public void Post([FromBody]string value)
        {
        }

        // PUT: api/BankClientRest/5
        public void Put(int id, [FromBody]string value)
        {
        }

        // DELETE: api/BankClientRest/5
        public void Delete(int id)
        {
        }
    }
}
```

Primer 1

- Editovati REST *endpoint*
 - Koji vraća listu koja sadrži klijente banke
 - putanja **/api/bankclientrest**
- desni klik na postojeći folder Models
 - Add -> Class
 - kreirati novu klasu u folderu
 - nazvati ja *BankClientModel*
- Klasa *BankClientModel* sadrži attribute:
 - *Id, name, surname, email*

Primer 1

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

namespace RestExample.Models
{
    public class BankClientModel
    {
        public int Id { get; set; }

        public string Name { get; set; }

        public string Surname { get; set; }

        public string Email { get; set; }
    }
}
```

Primer 1

- U klasi *BankClientModel* kreirati prazan konstruktor i konstruktore sa parametrima
- U klasi *BankClientRestController* izmeniti metodu koja vraća listu klijenata banke

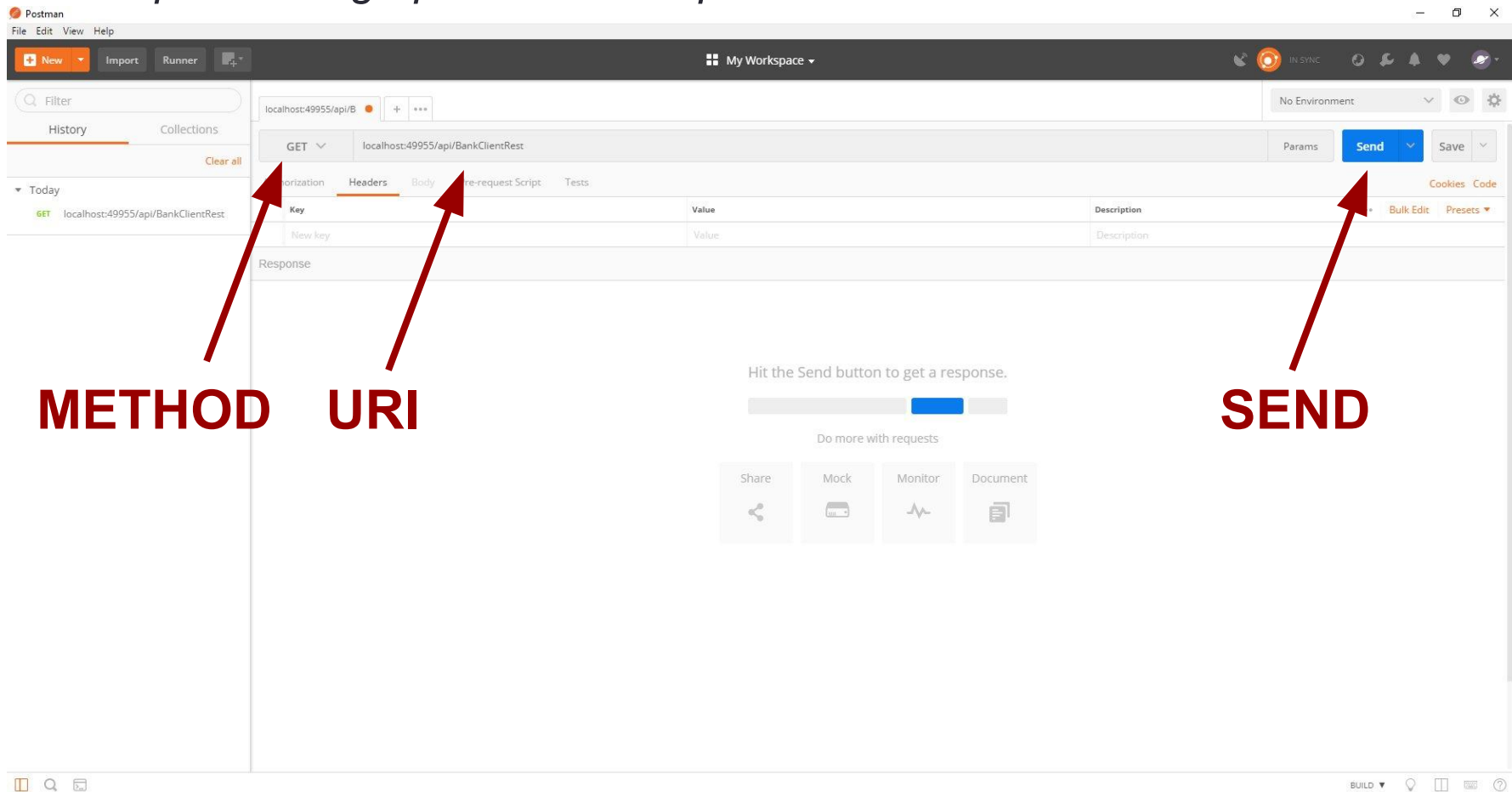
Primer 1

```
public IEnumerable<BankClientModel> Get()
{
    List<BankClientModel> clients = new List<BankClientModel>();
    clients.Add(new BankClientModel(1, "Ivan", "Vasiljevic",
                                     "ivan.vasiljevic@hotmail.com"));
    clients.Add(new BankClientModel(2, "Mladen", "Kanostrevac",
                                     "mkanostrevac@gmail.com"));

    return clients;
}
```

Primer 1 - testiranje

- Pokrenuti *Postman* aplikaciju
 - <https://www.getpostman.com/postman>



Primer 2

- Kreirati REST *endpoint*
 - Koji vraća klijenta banke po vrednosti *id* klijenta
 - putanja **/api/bankclientrest/{clientId}**
 - ukoliko klijent sa prosleđenom *id* vrednosti ne postoji vratiti objekat sa null vrednošću
- U ASP.NET Web API-ju postoje dva načina da se definiše ruta:
 - bazirano na konvenciji (do sada korištena)
 - bazirano na atributima

Routing

- Bazirano na konvenciji
 - koriste se šabloni za rute (route templates)
 - definiše se šablon
 - pravila za rutiranje se primenjuju konzistentno
 - teško je pružiti podršku za neke česte šablone kod definisanja rute

Convention based

- Izgenerisan *template*

```
config.Routes.MapHttpRoute(  
    name: "DefaultApi",  
    routeTemplate: "api/{controller}/{id}",  
    defaults: new { id = RouteParameter.Optional }  
);
```

- name - predstavlja jedinstveno ime šablona
- routeTemplate - šablon koji se popunjava
 - dva rezervisane reči koje se koriste u šablona
 - controller - predstavlja ime kontrolera
 - action - ime akcije, gde je običaj u Web API-ju da se on ne koristi
- defaults - za svaku rezervisanu reč možemo definisati podrazumevane vrednosti
 - RouteParameter.Optional - označava da je parametar opcioni

Convention based

- constraints - ograničenja
 - constraints: new { id = @"\d+" } // Only matches if "id" is one or more digits.
- Primer

```
routes.MapHttpRoute(  
    name: "DefaultApi",  
    routeTemplate: "api/{controller}/{category}/{id}",  
    defaults: new { category = "all", id =  
RouteParameter.Optional }  
);
```

Attribute based

- moramo omogućiti rutiranje zasnovano na atributima

```
using System.Web.Http;

namespace WebApplication
{
    public static class WebApiConfig
    {
        public static void Register(HttpConfiguration config)
        {
            // Web API routes
            config.MapHttpAttributeRoutes();

            // Other Web API configuration not shown.
        }
    }
}
```

- za svaku metodu u kontroleru koja koristi rutiranje atributima moramo dodati atribut koji HTTP metodu koristi i kako izgleda ruta

Attribute based

- podržane HTTP metode:
 - DELETE - anotacija `HttpDelete`
 - GET - anotacija `HttpGet`
 - HEAD - anotacija `HttpHead`
 - OPTIONS - anotacija `HttpOptions`
 - PATCH - anotacija `HttpPatch`
 - POST - anotacija `HttpPost`
 - PUT - anotacija `HttpPut`
- atribut `Route` služi za definisanje šablona za metodu koja će obraditi *request*
 - prima jedan parametar URI template
 - u templatu se mogu definisati variabilni parametri pomoću `{}`
 - `[Route("customers/{customerId}/orders")]`

Attribute based

- RoutePrefix atribut može se koristiti na kontroleru
- omogućava da se definiše URI template za kontroler

```
[RoutePrefix("api/books")]
public class BooksController : ApiController
{
    // GET api/books
    [Route("")]
    public IEnumerable<Book> Get() { ... }

    // GET api/books/5
    [Route("{id:int}")]
    public Book Get(int id) { ... }

    // POST api/books
    [Route("")]
    public HttpResponseMessage Post(Book book) { ... }
}
```

Attribute based

- Ako želimo da ignorišemo RoutePrefix koristimo '~' u templatu

```
[RoutePrefix("api/books")]
public class BooksController : ApiController
{
    // GET /api/authors/1/books
    [Route("~/api/authors/{authorId:int}/books")]
    public IEnumerable<Book> GetByAuthor(int authorId) { ... }

    // ...
}
```

Attribute based

- parametrima u ruti možemo definisati ograničenja
- sintaksa je {varijabla:ograničenje}
- podržani tipovi
 - alpha
 - bool
 - datetime
 - decimal
 - double
 - float
 - guid
 - int
 - length
 - long
 - max
 - maxlength
 - min
 - minlength
 - range
 - regex

Primer 2

// GET: api/BankClientRest/5

```
public BankClientModel Get(int id)
{
    if (id == 1)
    {
        return new BankClientModel(1, "Ivan", "Vasiljevic",
                                   "ivan.vasiljevic@hotmail.com");
    } else if (id == 2)
    {
        return new BankClientModel(2, "Mladen", "Kanostrevac",
                                   "mkanostrevac@gmail.com");
    }

    return new BankClientModel();
}
```


Primer 2 - testiranje

The image shows a REST client interface with several components. Red arrows point from labels below to specific parts of the interface:

- METHOD** points to the **GET** dropdown menu.
- URI** points to the text `localhost:49955/api/BankClientRest/1`.
- PATHPARAM** points to the `1` in the URI, which is highlighted with a red box.
- SEND** points to the blue **Send** button.

The interface also includes tabs for **Authorization**, **Headers** (which is selected), **Body**, **Pre-request script**, and **Tests**. Below the tabs is a table with columns **Key**, **Value**, **Description**, and **...**. On the right side of the table are buttons for **Build** and **Edit**.

Primer 2

- Kreirati metodu *GetDB* unutar kontrolera *BankClientRestController* koja vraća listu klijenata banke.
 - Metoda treba da bude vidljiva samo unutar klase
- Izmeniti metodu *Get(int)* tako da pronalazi klijenta u listi klijenata na osnovu prosleđene *id* vrednosti
- Izmeniti metodu *Get()* tako da vraća sve klijente iz liste klijenata

Primer 2

```
private List<BankClientModel> GetDb()
{
    List<BankClientModel> clients = new List<BankClientModel>();
    clients.Add(new BankClientModel(1, "Ivan", "Vasiljevic", "ivan.vasiljevic@hotmail.com"));
    clients.Add(new BankClientModel(2, "Mladen", "Kanostrevac",
                                     "mladen.kanostrevac@example.com"));

    return clients;
}

// GET: api/BankClientRest
public IEnumerable<BankClientModel> Get()
{
    return GetDb();
}

// GET: api/BankClientRest/5
public BankClientModel Get(int id)
{
    return GetDb().FirstOrDefault(x => x.Id == id);
}
```

Primer 3

- Kreirati REST *endpoint*
 - Koji omogućuje dodavanje novog klijenta
 - putanja **/api/bankclientrest/**
 - Metoda ispisuje ime i prezime klijenta koji je prosleđen i vraća poruku „*New client added*“
- FromBody
 - označava da metoda prihvata parametar koji je deo tela (*body*) web zahteva
- ime metode mora da počinje sa Post ili da metoda ima anotaciju HttpPost

Primer 3

```
// POST: api/BankClientRest
public string Post([FromBody]BankClientModel client)
{
    Debug.WriteLine($"{client.Name} {client.Surname}, email is: {client.Email}");
    return "New client added";
}
```

Primer 3 - testiranje

The screenshot displays a REST client interface for testing an API endpoint. The request is a POST to `localhost:49955/api/BankClientRest/3`. The request body is a JSON object:

```
{
  "Id": 3,
  "Name": "Aco",
  "Surname": "Vasiljevic",
  "Email": "aco.vasiljevic@hotmail.com"
}
```

Red arrows highlight the following elements:

- BODY**: Points to the 'Body' tab in the request configuration.
- TYPE**: Points to the 'raw' radio button, indicating the body is raw text.
- JSON / TEXT**: Points to the 'JSON (application/json)' dropdown menu.

The response status is **200 OK**, with a time of **286 ms** and a size of **402 B**. The response body is `"New client added"`.

Primer 4

- Kreirati REST *endpoint*
 - Koji omogućuje izmenu postojećeg klijenta
 - putanja **/api/BankClientRest/{clientId}**
 - Ukoliko je prosleđen id sa vrednošću 1 metoda vraća podatke o klijentu sa izmenjenim vrednostima. U suprotnom vraća null vrednost.
- ime metoda mora da počinje sa Put ili metoda ima atribut HttpPut

Primer 4

```
// PUT: api/BankClientRest/5
public BankClientModel Put(int id, [FromBody]BankClientModel client)
{
    BankClientModel bcb = new BankClientModel(1, "Ivan", "Vasiljevic",
"ivan.vasiljevic@hotmail.com");
    if (id == 1)
    {
        bcb.Name = client.Name;
        return bcb;
    }
    else
    {
        return null;
    }
}
```


Primer 5

- Kreirati REST *endpoint*
 - Koji omogućuje brisanje postojećeg klijenta
 - putanja **/api/BankClientRest/{clientId}**
 - Ukoliko je prosleđen id sa postojećom vrednosti metoda vraća podatke o klijentu koji je obrisao. U suprotnom vraća objekat sa null vrednostima.
- metoda mora da se počinje sa Delete ili da ima atribut **HttpDelete**

Primer 5

// DELETE: api/BankClientRest/5

```
public BankClientModel Delete(int id)
{
    BankClientModel bcb = GetDb().FirstOrDefault(x => x.Id == id);
    if (bcb != null)
    {
        GetDb().Remove(bcb);
        return bcb;
    }

    return new BankClientModel();
}
```

Primer 6

- Kreirati REST *endpoint*
 - Koji vraća klijenta na osnovu vrednosti *name* i *surname*
 - putanja **/api/BankClientRest/**
 - Ukoliko je prosleđen zahtev sa vašim imenom i prezimenom vratiti objekat koji sadrži vaše ime i prezime. U suprotnom vraća objekat sa null vrednostima.
- FromUri
 - označava da metoda prihvata parametar (*Query parameter*) koji je deo zahteva

Primer 6

```
// GET: api/BankClientRest?name=Ivan&surename=Vasiljevic
public BankClientModel Get([FromUri]string name, [FromUri]string
surename)
{
    if (name == "Ivan" && surename == "Vasiljevic")
    {
        return new BankClientModel(1, "Ivan", "Vasiljevic",
                                "ivan.vasiljevic@hotmail.com");
    }
    else
    {
        return new BankClientModel();
    }
}
```

Primer 6 - testiranje

The screenshot shows a REST client interface with a GET request to `localhost:49955/api/BankClientRest?name=Ivan&surename=Vasiljevic`. The request parameters are listed in a table, and the response body is shown in JSON format.

Request Parameters (Params):

Key	Value	Description
<input checked="" type="checkbox"/> name	Ivan	
<input checked="" type="checkbox"/> surname	Vasiljevic	
New key	Value	Description

Request Headers (Headers (1)):

Key	Value	Description
<input checked="" type="checkbox"/> Content-Type	application/json	
	Value	Description

Response Body (JSON):

```
{
  "Id": 1,
  "Name": "Ivan",
  "Surname": "Vasiljevic",
  "Email": "ivan.vasiljevic@hotmail.com"
}
```

Annotations:

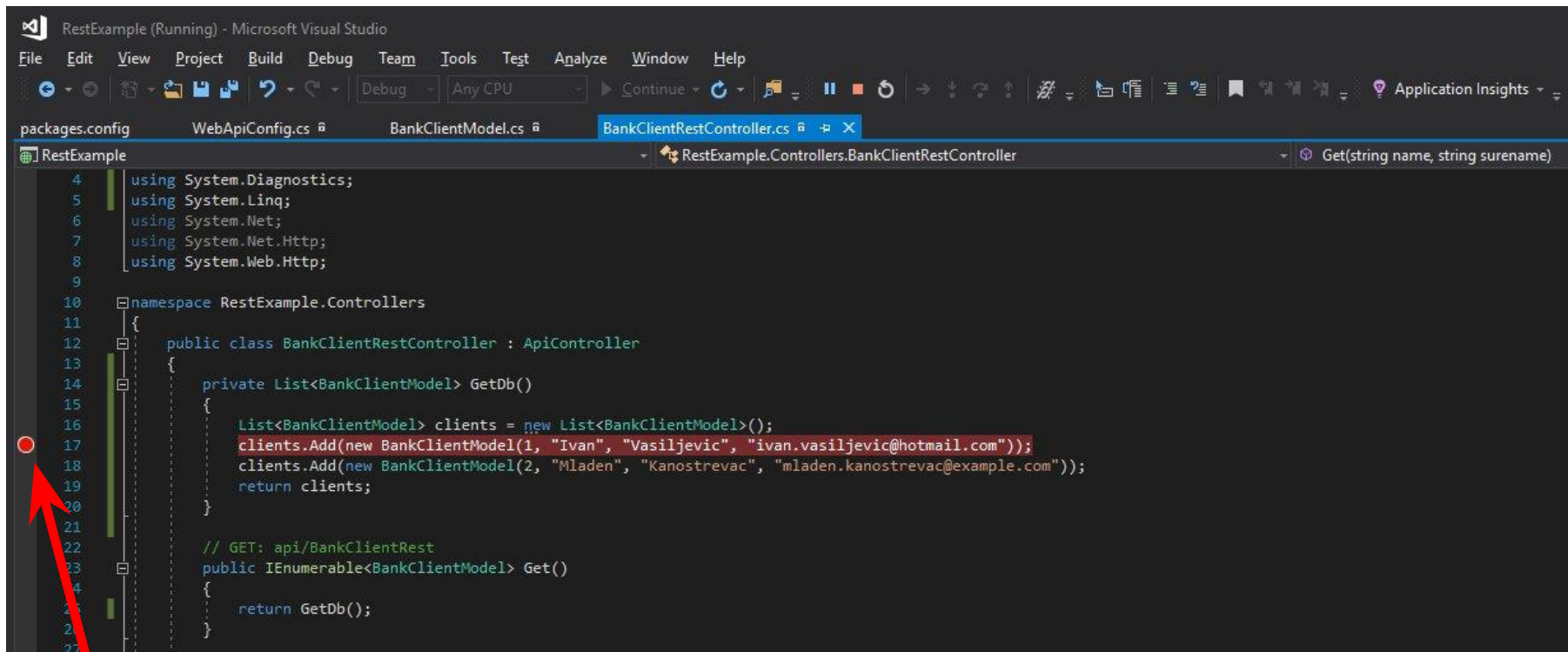
- QUERYPARAM**: Points to the 'name' and 'surname' parameters in the Params table.
- PARAMS**: Points to the 'Params' tab header.

DEBUGGING

Breakpoints

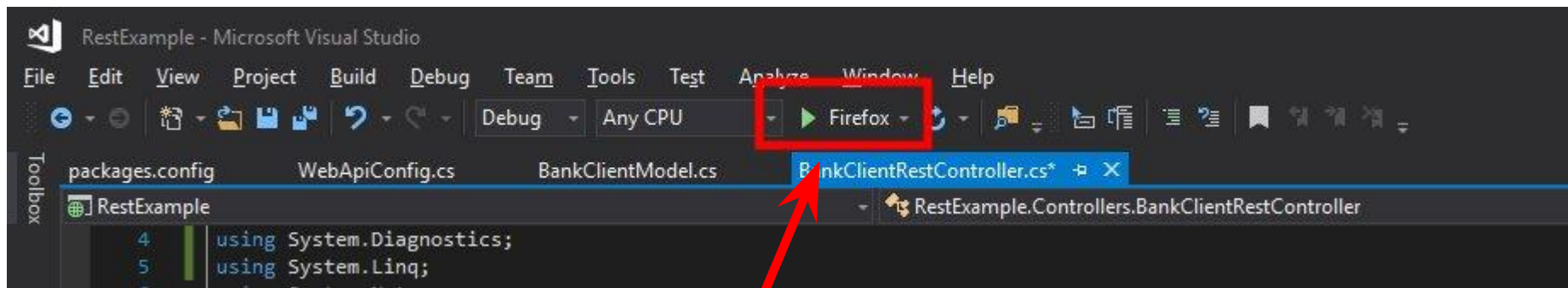
Tačka prekida

- Zadati tačku prekida na odgovorajućem mestu u programu
- Levi klik mišem



Debugging

- Pokrenuti VS projekat
 - **F5**
 - desni klik miša na projekat Debug -> Start new instance
 - Toolbar



Debugging

The screenshot displays the Microsoft Visual Studio IDE in a debugging session. The main window shows the source code of `BankClientRestController.cs` with a breakpoint set at line 37. The code defines a REST API for managing bank clients. The `Get` method is currently executing, and the debugger has paused at the `if` statement condition.

```
13 private List<BankClientModel> GetDb()
14 {
15     List<BankClientModel> clients = new List<BankClientModel>();
16     clients.Add(new BankClientModel(1, "Ivan", "Vasiljevic", "ivan.vasiljevic@hotmail.com"));
17     clients.Add(new BankClientModel(2, "Mladen", "Kanostrvac", "mladen.kanostrvac@example.com"));
18     return clients;
19 }
20
21 // GET: api/BankClientRest
22 public IEnumerable<BankClientModel> Get()
23 {
24     return GetDb();
25 }
26
27 // GET: api/BankClientRest/5
28 public BankClientModel Get(int id)
29 {
30     return GetDb().FirstOrDefault(x => x.Id == id);
31 }
32
33 // GET: api/BankClientRest?name=Ivan&surename=Vasiljevic
34 public BankClientModel Get([FromUri]string name, [FromUri]string surename)
35 {
36     if (name == "Ivan" && surename == "Vasiljevic")
37     {
38         return new BankClientModel(1, "Ivan", "Vasiljevic", "ivan.vasiljevic@hotmail.com");
39     }
40     else
41     {
42         return new BankClientModel();
43     }
44 }
```

The **Locals** window at the bottom left shows the current state of the program:

Name	Value	Type
this	[RestExample.Controllers.BankClientRestController]	RestExample.Controllers.B
name	"Ivan"	string
surename	"Vasiljevic"	string

The **Call Stack** window at the bottom right shows the current call stack:

Name	Lang
RestExample.dllRestExample.Controllers.BankClientRestController.Get(string name, string surename) Line 37	C#
[External Code]	

The **Diagnostics Tools** window on the right provides performance metrics for the current session:

- Diagnostics session: 6 seconds (6.583 s selected)
- Events: 10s
- Process Memory (MB): 64
- CPU (% of all processors): 100

The **Summary** tab in the Diagnostics Tools window shows the following data:

Summary	Events	Memory Usage	CPU Usage
Events	Show Events (1 of 1)	Take Snapshot	Record CPU Profile

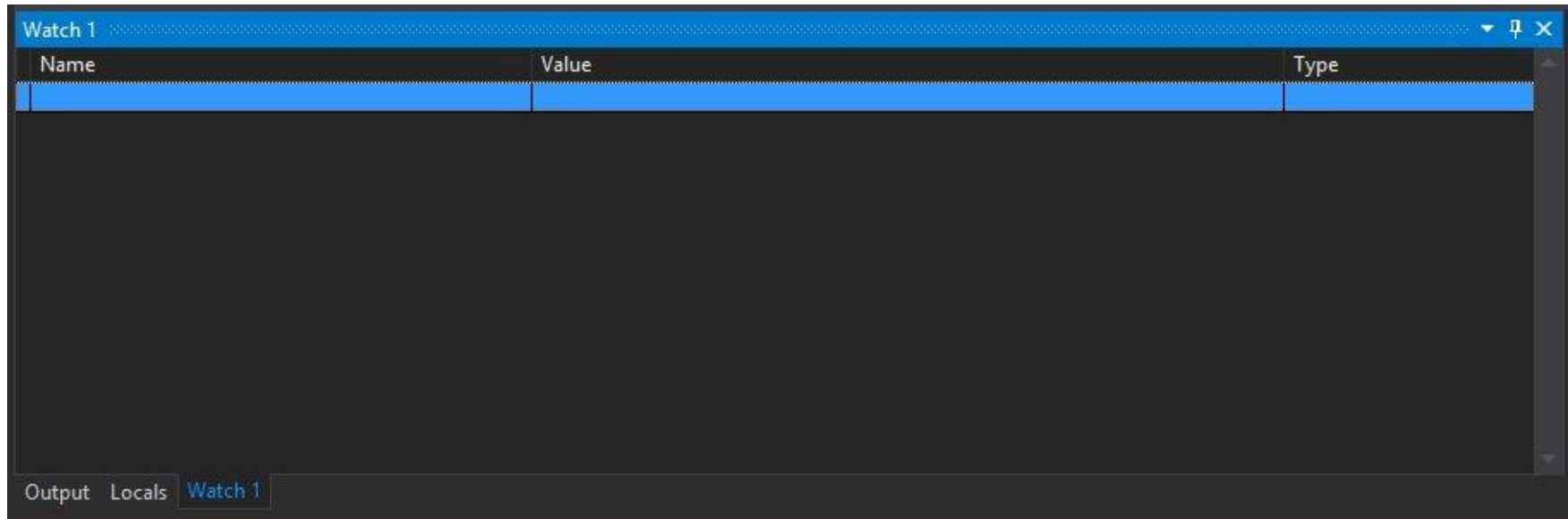
Debugging



- Nastavi izvršavanje programa
- Pauziraj izvršavanje programa
- Zaustavi izvršavanje programa
- Uđi u pozivajuću metodu
- Pređi na narednu naredbu u programu
- Povratak u metodu iz koje je pozvana

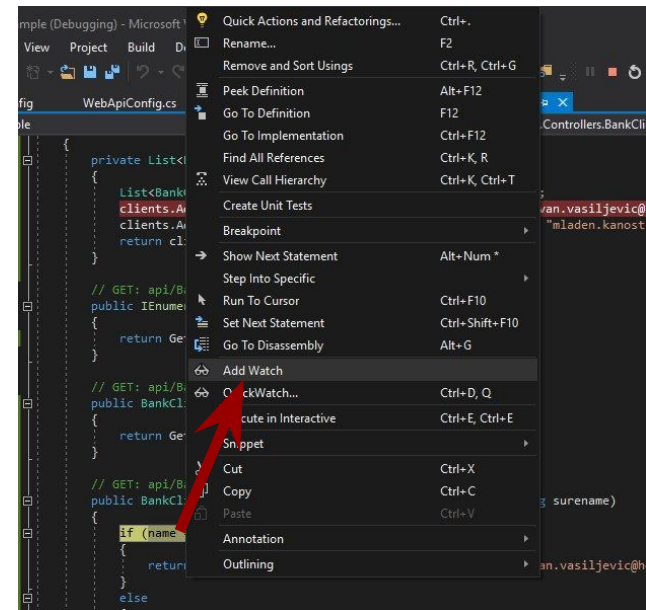
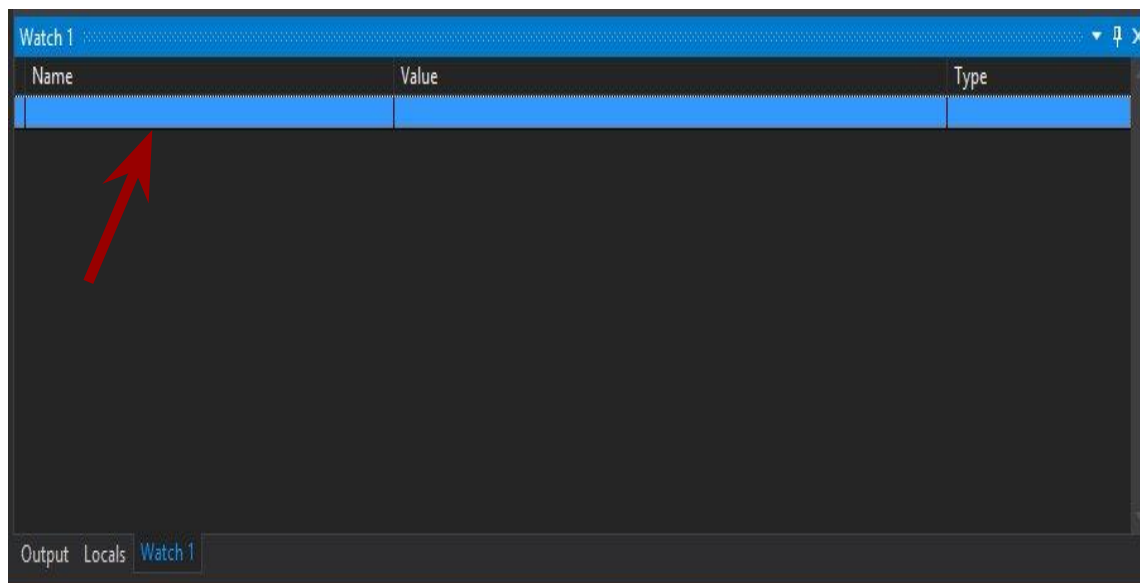
Debugging

- Po potrebi dodati prozor za nadgledanje vrednosti promenljivih
 - Debug -> Windows -> Watch -> Watch 1

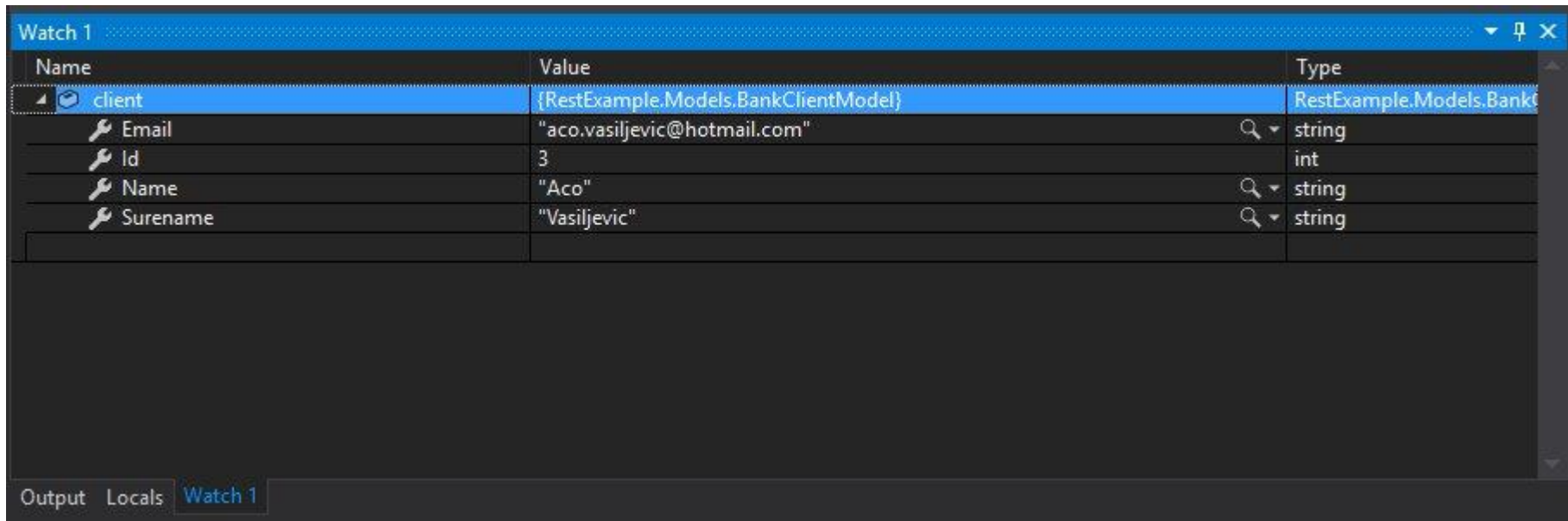


Debugging

- Po potrebi dodati prozor za nadgledanje vrednosti promenljivih
 - Window -> Show View -> Expressions
 - Dodati promenjive / izraze:
 - selektovati promenljivu -> kliknuti desni taster miša -> Add Watch
 - dvoklik na prazan red u tabeli na prozoru Watch 1



Debugging



The screenshot shows the 'Watch' window in a debugger. The window title is 'Watch 1'. It contains a table with three columns: 'Name', 'Value', and 'Type'. The first row is expanded, showing the 'client' object. The object is of type 'RestExample.Models.BankClientModel' and has four properties: 'Email' (string), 'Id' (int), 'Name' (string), and 'Surname' (string). The 'Email' property value is 'aco.vasiljevic@hotmail.com', 'Id' is 3, 'Name' is 'Aco', and 'Surname' is 'Vasiljevic'. The 'Watch' window is the active tab in the bottom pane, with 'Output' and 'Locals' also visible.

Name	Value	Type
client	{RestExample.Models.BankClientModel}	RestExample.Models.Bankt
Email	"aco.vasiljevic@hotmail.com"	string
Id	3	int
Name	"Aco"	string
Surname	"Vasiljevic"	string

Output Locals **Watch 1**

ZADACI

Zadatak 1

- Kreirati sledeće REST *endpointe*
 - 1.1 endpoint koji iz liste klijenata banke uzima samo email adrese svih klijenata i vraća listu email adresa
 - putanja **/api/clients/emails**
 - 1.2 endpoint koji vraća listu koja sadrži imena klijenata, čije ime počinje na slovo koje je prosleđeno kao parametar
 - putanja **/api/clients/{firstLetter}**
 - 1.3 endpoint koji vraća listu koja sadrži imena i prezimena klijenata, čije ime počinje na slovo koje je prosleđeno kao parametar i čije prezime počinje na slovo koje je prosleđeno kao parametar
 - putanja **/api/clients/firstLetters**
 - 1.4 endpoint koji vraća listu koja sadrži imena klijenata, koja su sortirana u redosledu koji je prosleđen kao parameter
 - putanja **/api/clients/sort/{order}**

Zadatak 2

- Kreirati sledeće REST *endpoints*
 - 2.1 endpoint koji u listi klijenata banke, svakom klijentu, postavlja polje bonitet na 'P' (pozitivan) ako je klijent mlađi od 65 godina ili 'N' negativan ako je klijent stariji od 65 godina
 - putanja **/api/clients/bonitet**
 - u klasu BankClientModel dodati atribut datum rođenja i bonitet
 - 2.2 endpoint koji briše klijenta iz liste klijenata ukoliko klijent nema jednu od vrednosti: ime, prezime, email
 - putanja **/api/clients/delete**
 - 2.3 endpoint koji vraća ukupan broj klijenata u listi klijenata koji imaju manje od broja godina koje je prosleđeno kao parametar
 - putanja **/clients/countLess/{years}**
 - 2.4 endpoint koji prosečan broj godina klijenata iz liste klijenata
 - putanja **/api/clients/averageYears**

Zadatak 3

- Kreirati sledeće REST *endpoints*
 - 3.1 endpoint koji omogućuje izmenu mesta stanovanja klijenta
 - putanja **/api/clients/changelocation/{clientId}**
 - u klasu BankClientBean dodati atribut grad
 - novu vrednost mesta stanovanja proslediti kao FromUri
 - 3.2 endpoint koji vraća klijente banke koji žive u gradu koji je prosleđen kao parametar
 - putanja **/api/clients/from/{city}**