Eötvös Loránd Tudományegyetem
Informatikai Kar

Webes alkalmazások fejlesztése

8. előadás

Webszolgáltatások megvalósítása (ASP.NET WebAPI)

© 2016 Giachetta Roberto
groberto@inf.elte.hu
http://people.inf.elte.hu/groberto

Webszolgáltatások megvalósítása
A webszolgáltatás

- A *webszolgáltatás* (*web service*) olyan protokollok és szabályok gyűjteménye, amely lehetővé teszi alkalmazások közötti platform független adatcserét hálózaton keresztül
  - azaz a rendszernek egy, vagy szolgáltatója (*service provider*) biztosítja a funkcióknak olyan felületét, amelyet a fogyasztók (*service consumer*) elérhetnek
    - a fogyasztó lehet bármilyen alkalmazás, weblap, …
  - a kommunikációra számos protokollt és megoldást használhat, pl. *SOAP* (*Simple Object Access Protocol*) és *WSDL* (*Web Services Description Language*), vagy *REST*
  - lehetővé teszi a *szolgáltatásorientált architektúra* (*Service Oriented Architecture*, *SOA*) létrehozását

Webszolgáltatások megvalósítása
REST

- A *REST* (*Representational State Transfer*) egy szoftver architektúra típus, amely lehetővé teszi skálázható, nagy teljesítményű elosztott hálózati alkalmazások fejlesztésére
  - elsősorban HTTP alapon kommunikál alapvető HTTP utasítások (`GET`, `POST`, `PUT`, `DELETE`, …) segítségével
  - megszorításokat ad a rendszernek:
    - kliens-szerver modell,
    - egységes interfész,
    - állapotmentes kommunikáció,
    - kiegészíthetőség (*code on demand*), …
  - a támogató szoftverek a *RESTful alkalmazások*

Webszolgáltatások megvalósítása
ASP.NET WebAPI

- Az *ASP.NET WebAPI* egy RESTful alkalmazások fejlesztését lehetővé tevő keretrendszer
  - az MVC architektúrát valósítja meg, a tevékenységeket *vezérlők* felügyelik, amelyek adott erőforrásra és utasításra reagálnak
  - az adatokat alapértelmezetten *JSON* (*Javascript Object Notation*) formátumban továbbítja, de a kliens kérésének megfelelően automatikusan tudja a formátumot módosítani
  - könnyen integrálható az ASP.NET MVC webalkalmazásokkal
  - NuGet-ből telepíthető (*ASP.NET WebAPI* csomag)

Webszolgáltatások megvalósítása
JSON

- A JSON egy egyszerű formátum objektumok szöveges leképezése, pl.:

```
{ // objektum
  "id": 1234, // attribútum
  "group": "tool",
  "name": "hammer",
  "resposible": { "name" : "John" },
                // összetett attribútum
  "materials": [ // tömb attribútum
    { "name": "steel" },
    …
  ]
}
```


Webszolgáltatások megvalósítása
Vezérlők

- A vezérlőkben (`ApiController`) valósítjuk meg a HTTP akcióműveleteket (`Get`, `Post`, …)
  - a visszatérési érték a HTTP válasz törzsébe (*body*) kerül, ekkor egy `OK` (200) válasz készül
    - amennyiben nincs visszatérési érték (`void`), akkor egy `No Content` (204) válasz kerül kiküldésre
  - a műveletek feloldása az elérési útvonal leképezésének (`HttpRoute`) megfelelően történik, alapértelmezetten a `<domain>/api/<vezérlő>/<paraméterek>` formában
    - a műveletek csak korlátozottan túlterhelhetőek
    - az erőforrás címe mellett tartalmat is szolgáltathatunk, amit a kérés törzsébe helyezünk (`FromBody`)

Webszolgáltatások megvalósítása
Vezérlők

- Pl.:

```
public class ProductsController : ApiController {
   IList<string> products; // modell
   …
   // elérés GET /api/products/
   public IEnumerable<string> Get() {
      return products; // összes termék lekérése
   }

   // elérés: GET /api/products/1
   public string Get(int id) {
      return products[id];
      // adott termék lekérése
   }
```


Webszolgáltatások megvalósítása
Vezérlők

- Pl.:

```
   // elérés: POST /api/products/
   public void Post([FromBody] string product) {
      // meg kell adnunk, hogy a tartalom a
      // törzsben található
      products.Add(product);
   }

   // elérés: DELETE /api/products/1
   public void Delete(int id) {
      products.RemoveAt(id);
   }
}
```


Webszolgáltatások megvalósítása
Konfiguráció

- A WebAPI használatba vétele előtt az ASP.NET alkalmazást megfelelő konfigurációval (elsősorban az útvonal feloldás leírásával) kell ellátnunk
  - az útvonalelérés konfigurációját a **`WebApiConfig`** osztály **`Register`** művelete végzi (az **`App_Start`** könyvtárban) :

```
config.Routes.MapHttpRoute(
   name: "DefaultApi",
   routeTemplate: "api/{controller}/{id}", …
});
```

  - az **`Application_Start`** eseménykezelőben ezt a műveletet át kell adnunk a globális konfigurációnak:

```
GlobalConfiguration.Configure(
   WebApiConfig.Register);
```


Webszolgáltatások megvalósítása
Adatszolgáltatás

- A webszolgáltatás műveletei nem csak primitív típusokat, de összetett, *adatátviteli objektum*okat (*Data Transfer Object*, *DTO*) is közölhetnek
  - DTO bármilyen objektum lehet, ami szérializálható (az elvárt formátumban), azaz leképezhető primitív értékekből álló felépítésre
  - a felépítését úgy kell megválasztanunk, hogy az belső adatokat, illetve szükségtelen, vagy körkörös hivatkozásokat (pl. entitásobjektum esetén) ne tartalmazzon
  - visszaadhatunk egyedileg konfigurált HTTP üzenetet is (**`HttpResponseMessage`**), amelyet aszinkron módon is létrehozhatunk (**`IHttpActionResult`**)

Webszolgáltatások megvalósítása
Adatszolgáltatás

- Pl.:

```
public class Product { // DTO típus
   public Int32 Id { get; set; }
   public String Name { get; set; }
}

public class ProductsController : ApiController {
   …
   // elérés GET /api/products/
   public IEnumerable<Product> Get() {
      return products; // összes termék lekérése
   }
   …
}
```


Webszolgáltatások megvalósítása
Adatszolgáltatás

- Pl.:

```
public class ProductsController : ApiController {
   …
   // elérés GET /api/products/
   public HttpResponseMessage Get() {
      return new HttpResponseMessage()
      { // egyedileg összeállított üzenet
         StatusCode = HttpStatusCode.OK,
         Content = new ObjectContent<Product>(
           products,
           Configuration.Formatters.JsonFormatter)
        // megadjuk a kódot és a tartalmat
      };
}
```

Webszolgáltatások megvalósítása
Műveletek elérése

- A HTTP műveletek társítása megfeleltetése vezérlő műveleteknek lehet
  - automatikus, a név kezdőszelete alapján történik, pl.:

```
public class ProductsController : ApiController
{
    …
    // elérés: GET /api/products/1
    public Product GetSingleProduct(int id) { … }
    …
    // elérés: DELETE /api/products/1
    public void DeleteProduct(int id) { … }
    …
}
```


Webszolgáltatások megvalósítása
Műveletek elérése

- manuális, attribútumok segítségével megjelölve a HTTP műveletet (**`HttpGet`**, **`HttpPost`**, **`HttpDelete`**, …) és az elérési útvonalat (**`Route`**), pl.:

```
public class ProductsController : ApiController
{
    …
    // elérés: GET /api/myproducts/1
    [Route("api/myproducts/{id}")]
    [HttpGet]
    public Product FindProduct(int id) { … }
    …
}
```

- a manuális útvonal feloldást külön kell jeleznünk:

```
config.MapHttpAttributeRoutes();
```


Webszolgáltatások megvalósítása
Műveletek elérése

- az útvonal megjelölésénél lehetőségünk van
  - előtagot adni a vezérlőnek (**`RoutePrefix`**)
  - tetszőleges módon elhatárolni a paramétereket (további útvonal komponensek hozzáadásával)
  - megszorításokat adni a paraméterekre, úgymint típus (**`bool`**, **`datetime`**, **`decimal`**, **`double`**, **`float`**, **`guid`**, **`int`**, **`long`**), hosszúság (**`length`**, **`maxlength`**, **`maxlength`**), érték (**`min`**, **`max`**, **`range`** , **`values`**), alak (**`alpha`**, **`regex`**)
  - meghatározni a prioritást (**`RouteOrder`**), amennyiben több műveletre is illeszkedik az útvonal
- a típusmegjelölés lehetővé teszi a túlterhelést, mivel a típusnak megfelelő műveletet tudja futtatni a rendszer

Webszolgáltatások megvalósítása
Műveletek elérése

- pl.:

```
[RoutePrefix("api/myproducts")] // előtag
public class ProductsController : … {
    …
    // elérés: GET /api/myproducts/1
    [Route("{id:int:min(1)}")]
    public Product GetProduct(int id) { … }

    // elérés: GET /api/myproducts/tools/item/1
    [Route("{group:values(tools|machines)}/
              item/{id:int:min(1)}")]
    public Product GetProduct(string group,
                              int id) { … }
}
```

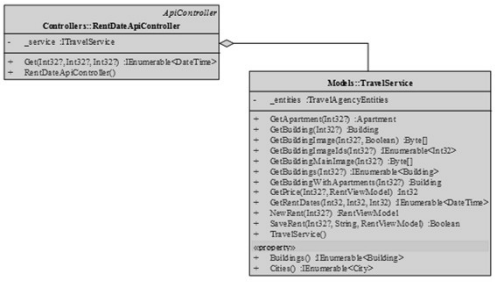

Webszolgáltatások megvalósítása
Példa

Feladat: Valósítsuk meg egy utazási ügynökség weblapját, amelyben apartmanok között böngészhetünk.

- könnyítsük meg a foglalást az által, hogy a dátumot egy naptár (**`glDatePicker`**) segítségével lehessen kiválasztani
- a dátumkiválasztó automatikusan lekérdezi (jQuery segítségével) a szolgáltatástól, mely napok szabadok
- a szolgáltatása vezérlője (**`RentDateApiController`**) egy műveletet biztosít (**`Get`**) a megfelelő napok lekérdezésére
- magát a lekérdezést a modell (**`TravelService`**) biztosítja (**`GetRentDates`**), amely adott egy hónap körüli

Webszolgáltatások megvalósítása
Példa

Tervezés:



Webszolgáltatások megvalósítása
Példa

Megvalósítás (`RentDateApiController.cs`):

```
[RoutePrefix("api/rentdate")]
   // útvonal feloldás megadása
public class RentDateApiController : ApiController
{
   …
   [Route("{apartmentId}/{year}/{month}")]
   // útvonal feloldás megadása
   public IEnumerable<DateTime> Get(
                     Int32? apartmentId,
                     Int32? year, Int32? month)
   {
      …
   }
}
```


Webszolgáltatások megvalósítása
Példa

Megvalósítás (`Rent/Index.cshtml`):

```
<script type="text/javascript">
    $(window).load(function () {
       …
       jQuery.getJSON("api/rentdate/" +
          @Model.Apartment.Id + "/" + year + "/" +
          month,
          function (data)
          {
             options.selectableDates =
                    parseDates(data);
          });
       …
    }
```


Webszolgáltatások megvalósítása
Visszajelzés és hibakezelés

- A vezérlő nem csupán az alapértelmezett, de tetszőleges HTTP kóddal tud válaszolni a kérésekre, amennyiben általános visszatérési típust specifikálunk (`IHttpActionResult`)
  - előre definiált visszatérési függvényekkel könnyedén megadhatjuk az eredményt: `Ok(<content>)`, `Created(<location>, <content>)`, `Redirect(<location>)`, `NotFound()`, `Unauthorized()`, `BadRequest(<message>)`, `Conflict()`, `InternalServerError(<exception>)`)
  - a megfelelő visszajelzés a hibakezelés szempontjából is fontos (amennyiben nem kezeljük le a műveletben dobott kivételeket, `INTERNAL SERVER ERROR` (500) üzenetet küld a szolgáltatás)

Webszolgáltatások megvalósítása
Visszajelzés és hibakezelés

- pl.:

```
public IHttpActionResult GetProduct(int id)
{
   try {
      …
      return Ok(product);
         // amennyiben sikeres volt a
         // lekérdezés, 200-as kód
   }
   catch {
      return NotFound();
         // ellenkező esetben 404-es kód
   }
}
```


Webszolgáltatások megvalósítása
Tesztelés

- A webszolgáltatások tesztelése elvégezhető
  - manuálisan, kliens oldalon, a kérések küldését biztosító program (böngésző) segítségével
  - automatikusan, kliens oldalon, a kérések küldését biztosító osztály (pl. `HttpClient`) segítségével
  - automatikusan, szerver oldalon, a vezérlő műveleteinek közvetlen tesztelésével
- A webszolgáltatás használata a célkörnyezetben (weblap, asztali alkalmazás, …) már *integrációs teszt*, amelyet csak a megfelelő *egységtesztek* végrehajtása után kezdeményezhetünk

Webszolgáltatások megvalósítása
Mock objektumok

- Amennyiben függőséggel rendelkező programegységet tesztelünk, a függőséget helyettesítjük annak szimulációjával, amit *mock objektum*nak nevezünk
  - megvalósítja a függőség interfészét, egyszerű, hibamentes funkcionalitással
  - használatukkal a teszt valóban a megadott programegység funkcionalitását ellenőrzi, nem befolyásolja a függőségben felmerülő esetleges hiba
- Mock objektumokat manuálisan is létrehozhatunk, vagy használhatunk erre alkalmas programcsomagot
  - pl. *NSubstitute*, *Moq* letölthetőek NuGet segítségével

Webszolgáltatások megvalósítása
Mock objektumok

- Pl. :

```
class DependencyMock : IDependency
   // mock objektum
{
   // egy egyszerű viselkedést adunk meg
   public Double Compute() { return 1; }
   public Boolean Check(Double value) {
      return value >= 1 && value <= 10;
   }
}
…
Dependant d = new Dependant(new DependencyMock());
   // a mock objektumot fecskendezzük be a függő
   // osztálynak
```


Webszolgáltatások megvalósítása
Mock objektumok

- *Moq* segítségével könnyen tudunk interfészekből mock objektumokat előállítani
  - a `Mock` generikus osztály segítségével példányosíthatjuk a szimulációt, amely az `Object` tulajdonsággal érhető el, és alapértelmezett viselkedést produkál, pl.:

```
Mock<IDependancy> mock =
   new Mock<IDependancy>();
   // a függőség mock objektuma
Dependant d = new Dependant(mock.Object);
   // azonnal felhasználható
```

  - a `Setup` művelettel beállíthatjuk bármely tagjának viselkedését (`Returns(…)`, `Throws(…)`, `Callback(…)`), a paraméterek szabályozhatóak (`It`)

Webszolgáltatások megvalósítása
Mock objektumok

- pl. :

```
mock.Setup(obj => obj.Compute()).Returns(1);
   // megadjuk a viselkedést, mindig 1-t ad
   // vissza
mock.Setup(obj =>
            obj.Check(It.IsInRange<Double>(0, 10,
                      Range.Inclusive)))
    .Returns(true);
mock.Setup(obj => obj.Check(It.IsAny<Double>())
    .Returns(false);
    // több eset a paraméter függvényében
…
```

- lehetőségünk van a hívások nyomkövetésére (`Verify(…)`)

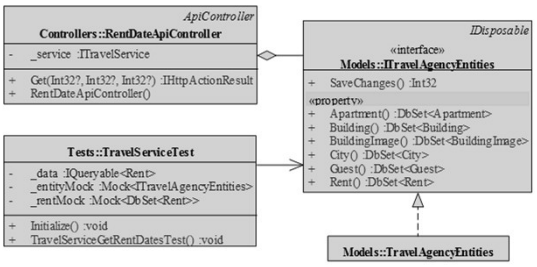
Webszolgáltatások megvalósítása
Példa

Feladat: Teszteljük az utazási ügynökség weblapját, azon belül pedig a szabad napokat lekérdező webszolgáltatást.

- egy külön tesztprojektben létrehozzuk a tesztkörnyezetet biztosító osztályt (`TravelServiceTest`), ezek belül pedig a `Get` művelet funkcionalitását teszteljük
- ehhez leválasztjuk az entitásmodell interfészét (`ITravelAgencyEntities`), amelyet szimulálunk a teszthez (ehhez a Moq programcsomagot használjuk)
- an entitásmodell mellett a foglalások gyűjteményét (`DbSet<Rent>`) is szimuláljuk, és az adatokat egy listában adjuk meg

Webszolgáltatások megvalósítása
Példa

Tervezés:



Webszolgáltatások megvalósítása
Példa

Megvalósítás (`TravelServiceTest.cs`):

```
[TestMethod]
public void TravelServiceGetRentDatesTest() {
   …
   // ellenőrzések júniusra
   DateTime[] result =
      service.GetRentDates(0, 2016, 06).ToArray();
   foreach (DateTime date in _data.
            Where(rent => rent.ApartmentId == 0).
            Select(rent => rent.StartDate))
      Assert.IsFalse(content.Contains(date));
   …
}
```