



**Eötvös Loránd Tudományegyetem  
Informatikai Kar**

## **Webes alkalmazások fejlesztése**

---

### **7. előadás**

## **Webszolgáltatások megvalósítása (ASP.NET Core)**

---

**Cserép Máté**

**[mcserep@inf.elte.hu](mailto:mcserep@inf.elte.hu)**

**<http://mcserep.web.elte.hu>**

# 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 szolgáltatója (*service provider*) biztosítja a funkcióknak olyan felületét, amelyet a fogyasztók (*service consumer*) elérhetnek
    - a fogyasztó lehet bármilyen alkalmazás, weblap, stb.
  - 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,
    - réteges felépítés,
    - gyorsítótárazhatóság
    - kiegészíthetőség (*code on demand*)
  - a támogató szoftverek a *RESTful alkalmazások*

# Webszolgáltatások megvalósítása

## ASP.NET WebAPI

---

- Az *ASP.NET Core WebAPI* keretrendszer lehetővé teszi a RESTful alkalmazások fejlesztését
  - MVC architektúrában megvalósítva, 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 *ASP.NET Core MVC* webalkalmazásokkal
  - NuGet-ből telepíthető (`Microsoft.AspNet.WebApi.Core` csomag)

# Webszolgáltatások megvalósítása

## JSON

---

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

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

# Webszolgáltatások megvalósítása

## Vezérlők

---

- A vezérlőkben (**Controller** osztályból származtatva) 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 (**Route**) megfelelően történik, alapértelmezetten a **<domain>/api/<vezérlő>/<paraméterek>** formában
    - a műveletek csak korlátozottan túlterhelhetőek
    - az erőforrás címe mellett tartalmat is szolgáltatathatunk, amit a kérés törzsébe helyezünk (**FromBody**)

# Webszolgáltatások megvalósítása

## Vezérlők

---

- Pl.:

```
[Route ("api/myproducts")]
public class ProductsController : Controller {
    IList<string> products; // modell
    ...
    // elérés GET /api/myproducts/
    [HttpGet]
    public IEnumerable<string> Get() {
        return products; // összes termék lekérése
    }
    // elérés: GET /api/myproducts/1
    [HttpGet("{id}")]
    public string Get([FromRoute] int id) {
        return products[id]; // adott termék lekérése
    }
}
```

# Webszolgáltatások megvalósítása

## Vezérlők

---

- Pl.:

```
// elérés: POST /api/myproducts/  
[HttpPost]  
public void Post([FromBody] string product) {  
    // meg kell adnunk, hogy a tartalom a  
    // törzsben található  
    products.Add(product);  
}  
  
// elérés: DELETE /api/myproducts/1  
[HttpDelete("{id}")]  
public void Delete([FromRoute] int id) {  
    products.RemoveAt(id);  
}  
}
```



# Webszolgáltatások megvalósítása

## Konfiguráció

---

- A Web API használatba vétele előtt az ASP.NET Core alkalmazást megfelelő konfigurációval (elsősorban az útvonal feloldás leírásával) kell ellátnunk
  - az útvonalelérés konfigurációját vezérlő és akció szintű *routing* attribútumokkal konfigurálhatjuk.

- pl.:

```
[Route("api/products")]
public class ProductsController : Controller {
    // elérés GET /api/products/1
    [HttpGet("{id}")]
    public string Get() {
        // ...
    }
}
```

# Webszolgáltatások megvalósítása

## Adatszolgáltatás

---

- A webszolgáltatás műveletei nem csak primitív típusokat, de összetett, *adatátviteli objektumokat* (*Data Transfer Object*, *DTO*) is közölhetnek
  - DTO bármilyen objektum lehet, ami szerializálható (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 (**ContentResult**), amelyet aszinkron módon is létrehozhatunk (**Task<IActionResult>**)

# Webszolgáltatások megvalósítása

## Adatszolgáltatás

---

- Pl.:

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

```
[Route("api/products")]
public class ProductsController : Controller {
    // elérés GET /api/products/
    [HttpGet]
    public IEnumerable<Product> Get() {
        return products; // összes termék lekérése
    }
    ...
}
```

---

# Webszolgáltatások megvalósítása

## Adatszolgáltatás

---

- Pl.:

```
[Route("api/myproducts")]
public class ProductsController : Controller {
    ...
    // elérés GET /api/products/
    public IActionResult Get() {
        return new ContentResult()
        { // egyedileg összeállított üzenet
            StatusCode = HttpStatusCode.OK,
            Content = JsonConvert.SerializeObject(
                products); // string
        // megadjuk a kódot és a tartalmat
        };
    }
}
```

# Webszolgáltatások megvalósítása

## Műveletek elérése

---

- A HTTP műveletek megfeleltetése vezérlő műveleteknek lehet

- automatikus, a név kezdőszelete alapján történik, pl.:

```
[Route("api/products")]
```

```
public class ProductsController : Controller {
```

```
...
```

```
// elérés: GET /api/products
```

```
public Product GetAllProduct() { ... }
```

```
...
```

```
// elérés: DELETE /api/products
```

```
public void DeleteAllProducts() { ... }
```

```
...
```

```
}
```

# Webszolgáltatások megvalósítása

## Műveletek elérése

---

- manuális, attribútumok segítségével megjelölve az elérési útvonalat (**Route**) vagy egyben akár a kívánt HTTP műveletet (**HttpGet**, **HttpPost**, **HttpDelete**, ...) is, pl.:

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

# Webszolgáltatások megvalósítása

## Műveletek elérése

---

- vagy kombinálható a vezérlő és az akció *routing* szabálya:  

```
[Route("api/myproducts")]  
public class ProductsController : Controller {  
    ...  
    // elérés: GET /api/myproducts/1  
    [HttpGet("{id}")]  
    public Product FindProduct(int id) { ... }  
    ...  
}
```
- az attribútum alapú útvonal feloldást a `Startup.cs` fájlban regisztráljuk: `app.UseMvc();`

# Webszolgáltatások megvalósítása

## Műveletek elérése

---

- az útvonal megjelölésénél lehetőségünk van
  - előtagot adni a vezérlő szintjén (**Route**)
  - 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 (**Order** tulajdonság), 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.:

```
[Route("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

## Visszajelzés és hibakezelés

---

- A vezérlő nem csupán az alapértelmezett, de tetszőleges HTTP kóddal tud válaszolni a kérésekre, amennyiben általános visszatérési típust specifikálunk (**IActionResult**)
  - 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 IActionResult 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, így böngésző vagy célszoftver (pl. *Postman*, *Insomnia*, *Fiddler*) segítségével
  - automatikusan, kliens oldalon, a kérések küldését biztosító osztály (pl. `HttpClient`) segítségével
  - automatikusan, szerver oldalon, a vezérlő műveleteinek közvetlen tesztelésével