



Eötvös Loránd Tudományegyetem
Informatikai Kar

Webes alkalmazások fejlesztése

7. előadás

Webszolgáltatások megvalósítása (ASP.NET Core)

Cserép Máté

mcserep@inf.elte.hu

<https://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 MVC* 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* keretrendszerben készített weblapokkal
 - fejlesztéshez a **Microsoft.NET.Sdk.Web** SDK csomag használható, mint az eddig MVC webalkalmazásoknál

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

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

Webszolgáltatások megvalósítása

Vezérlők

- A vezérlőkben (**Controller** osztályból származtatva) valósítjuk meg a HTTP akcióműveleteket (**Get**, **Post**, ...)
 - sikeres végrehajtáskor 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, jellemzően 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/myproducts")]
public class ProductsController : Controller {
    // elérés GET /api/myproducts/1
    [HttpGet("{id}")]
    public string Get(int id) {
        // ...
    }
}
```

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/myproducts")]
public class ProductsController : Controller {
    // elérés GET /api/myproducts/
    [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/myproducts")]
```

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

```
...
```

```
// elérés: GET /api/myproducts
```

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

```
...
```

```
// elérés: DELETE /api/myproducts
```

```
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 : Controller {  
    ...  
    // 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) { ... }`
`...`
`}`
- a vezérlő osztályok használatát **Program** osztályban regisztráljuk: `builder.Services.AddControllers()`
 - a weblapok esetén a nézetek használatát is regisztráltuk:
`builder.Services.AddControllersWithViews()`

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 (**`ActionResult`**)
 - előre definiált visszatérési függvényekkel könnyedén megadhatjuk az eredményt: **`Ok(<content>)`**, **`Created(<location>, <content>)`**, **`Redirect(<location>)`**, **`NotFound()`**, **`Forbidden()`**, **`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

ApiController attribútum

- Webszolgáltatások API vezérlőit célszerű ellátni az **ApiController** attribútummal, amely számos konfigurációt elvégez az osztály és akciói tekintetében. Fontosabbak:
 - a *routing szabályok* csak a **Routing** attribútummal adhatóak meg, a **Program** osztály **app.UseEndpoints()** eljárásában megadott szabályok nem kerülnek alkalmazásra
 - cserélhető **app.MapControllers()** eljárás hívásra
 - a nézetmodell automatikus validációja és 400 (*Bad request*) hibaüzenet visszaadása sikertelenség esetén.
 - Mintha minden akció tartalmazná az alábbi ellenőrzést:


```
if (!ModelState.IsValid)
{
    return BadRequest(ModelState);
}
```

Webszolgáltatások megvalósítása

ApiController attribútum

- a paraméterek forrásának implicit meghatározása az akciómetódusok esetében, az alábbiak szerint:
 - **[FromBody]** alkalmazása komplex típusokra, néhány kivétellel (pl. **IFormCollection**, **CancellationToken**).
 - **[FromForm]** alkalmazása **IFormFile** és **IFormFileCollection** típusú paraméterekre.
 - **[FromRoute]** alkalmazása az útvonal szabályban egyező névvel definiált paraméterekre.
 - **[FromQuery]** minden további paraméterre.
- pl.:

```
[ApiController]  
[Route("api/[controller]")]  
public class ProductsController : Controller
```

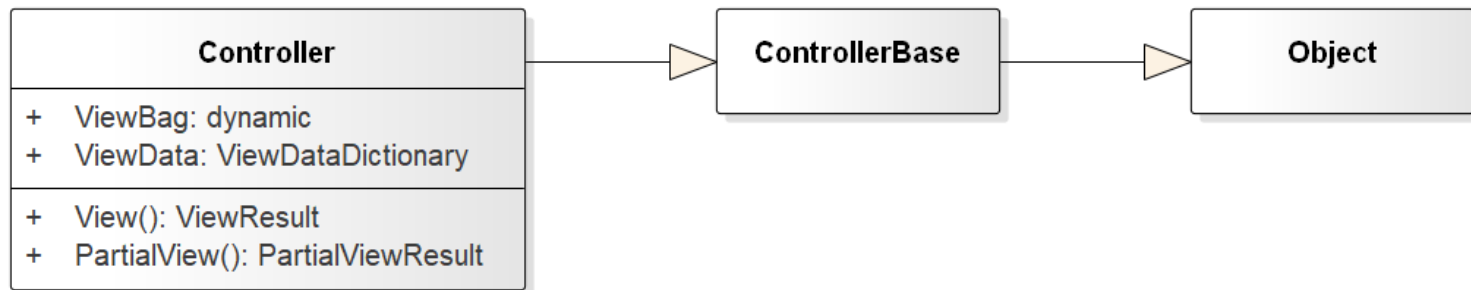


vezérlő nevének automatikus behelyettesítése

Webszolgáltatások megvalósítása

ControllerBase őssztály

- API vezérlő osztályainkat a **Controller** osztály helyett a **ControllerBase** osztályból is származtathatjuk.
 - A **ControllerBase** osztály a **Controller** osztály őse, amely a nézetek támogatását leszámítva rendelkezik az összes szükséges funkcionalitással.



- Amennyiben egyetlen vezérlő osztály szolgál ki nézeteket és API kéréseket is, a **Controller** osztályból érdemes származtatni.

Webszolgáltatások megvalósítása

Tesztelés

- A webszolgáltatások tesztelése elvégezhető
 - manuálisan, kliens oldalon, a kérések küldését biztosító program, így böngésző vagy célszoftver (pl. *Postman*, *Insomnia*, *Fiddler*) segítségével
 - manuálisan, egy tesztelő webes vagy asztali kliens alkalmazás generálásával az API-hoz (pl. *Swagger UI*)
 - automatikusan, kliens oldalon, a kérések küldését biztosító osztály (pl. `HttpClient`) segítségével
 - automatikusan, szerver oldalon, a vezérlő műveleteinek közvetlen tesztelésével

Webszolgáltatások megvalósítása

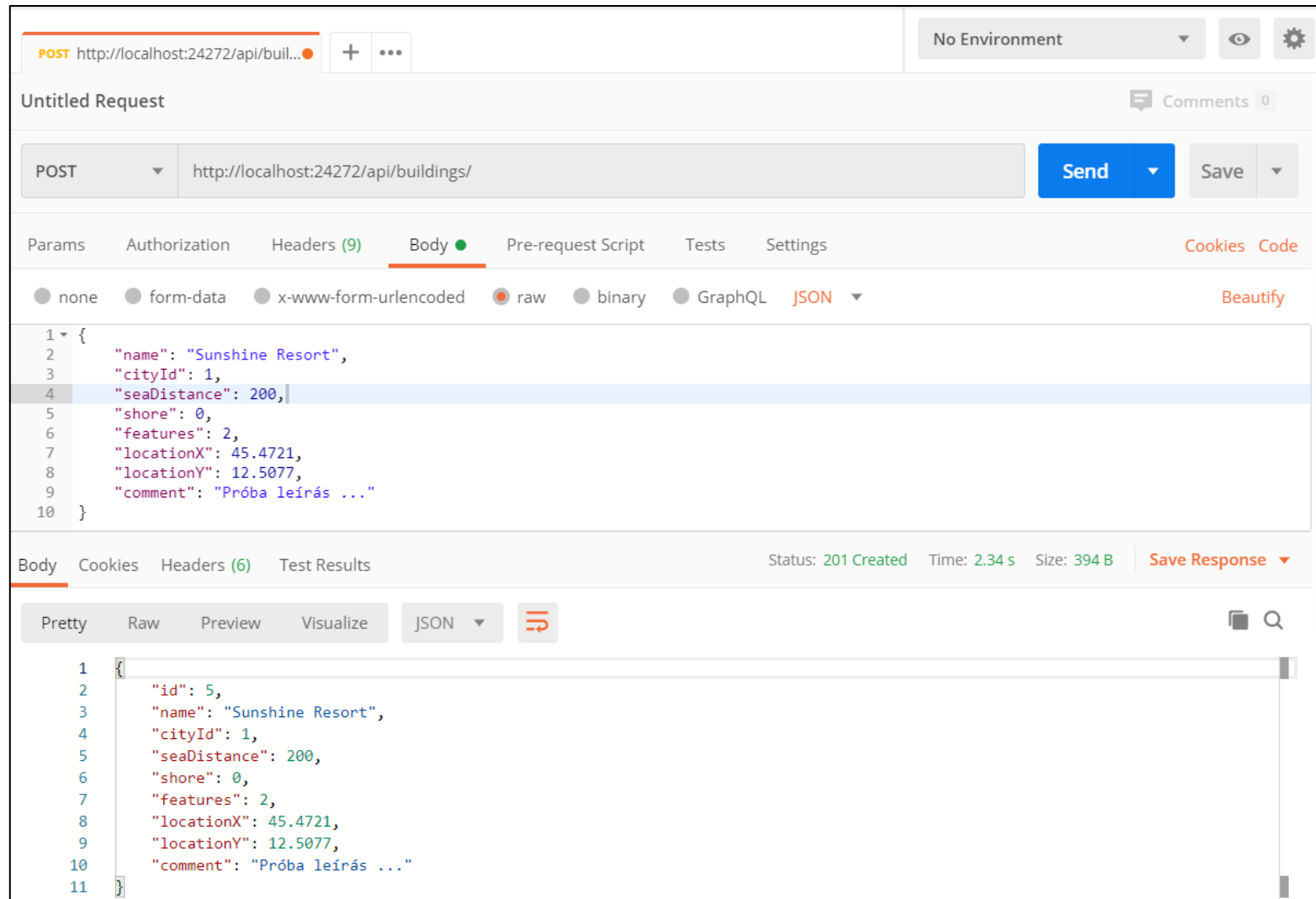
Postman használata - GET

The screenshot shows the Postman application interface. At the top, there's a tab for the current request, labeled 'GET http://localhost:24272/api/build...'. To the right, there's a dropdown for 'No Environment' and icons for eye and settings. Below this, the title 'Untitled Request' is displayed, along with a 'Comments' section showing 0 comments. The main area shows the request method 'GET' and the URL 'http://localhost:24272/api/buildings/2'. There are 'Send' and 'Save' buttons. Below the URL bar, there are tabs for 'Params', 'Authorization', 'Headers (7)', 'Body', 'Pre-request Script', 'Tests', and 'Settings'. The 'Params' tab is active, showing 'Query Params' with a table with columns 'KEY', 'VALUE', and 'DESCRIPTION'. The table has one row with 'Key' and 'Value'. To the right of the table are 'Bulk Edit' and 'Code' buttons. Below the table, there are tabs for 'Body', 'Cookies', 'Headers (5)', and 'Test Results'. The 'Body' tab is active, showing the response status '200 OK', time '22 ms', and size '705 B'. There is a 'Save Response' button. At the bottom, there are tabs for 'Pretty', 'Raw', 'Preview', and 'Visualize'. The 'Pretty' tab is active, showing the response in JSON format. The JSON object contains the following data:

```
{
  "id": 2,
  "name": "Petra bungaló",
  "cityId": 1,
  "seaDistance": 100,
  "shore": 0,
  "features": 0,
  "locationX": 45.4591,
  "locationY": 12.5068,
  "comment": "Szállás bungalóban 100m strandtól (homokos tengerpart). A területen található étterem, bár, játszótér. Éjszakai nyugalmat betartani. Ott-tartózkodás szombattól szombatig. A szállásslátszolgáltató beszél is csehül. Parkoló - nem őrzött, ingyenes. Busz 100m. Vonat (állomás San Dona di Piave) 25km. Repülőtér (Venezia - Marco Polo) 40km. Cavallino - központ.)"
```


Webszolgáltatások megvalósítása

Postman használata - POST



The screenshot displays the Postman interface for a POST request. The request URL is `http://localhost:24272/api/buildings/`. The body is formatted as JSON, containing the following data:

```
1 {
2   "name": "Sunshine Resort",
3   "cityId": 1,
4   "seaDistance": 200,
5   "shore": 0,
6   "features": 2,
7   "locationX": 45.4721,
8   "locationY": 12.5077,
9   "comment": "Próba leírás ..."
10 }
```

The response status is **201 Created**, with a response time of **2.34 s** and a size of **394 B**. The response body is also in JSON format:

```
1 {
2   "id": 5,
3   "name": "Sunshine Resort",
4   "cityId": 1,
5   "seaDistance": 200,
6   "shore": 0,
7   "features": 2,
8   "locationX": 45.4721,
9   "locationY": 12.5077,
10  "comment": "Próba leírás ..."
11 }
```

Webszolgáltatások megvalósítása

OpenAPI specifikáció

- Az *OpenAPI specifikáció* (korábbi nevében *Swagger specifikáció*) egy olyan nyílt formátum, amelyben JSON vagy YAML leíró nyelven definiálhatjuk a webszolgáltatásunk interfészét
 - végpontokat (és támogatott HTTP műveleteket)
 - bemeneti és kimeneti paramétereket
 - alkalmazott autentikációs módszereket
 - egyéb leíró információkat (pl. licence)
- A webszolgáltatást leíró OpenAPI formátumú specifikáció egy egységes interfészt nyújt, amely alapján a szolgáltatók és a fogyasztók is implementálhatók.

Webszolgáltatások megvalósítása

OpenAPI specifikáció

- Pl.

```
{ "openapi": "3.0.1",  
  "paths": {  
    "/api/products/{id}": {  
      "get": {  
        "parameters": [  
          {  
            "name": "id",  
            "in": "path",  
            "required": true,  
            "schema": {  
              "type": "integer",  
              "format": "int32"  
            }  
          }  
        ],  
        "responses": {  
          "404": {  
            ...  
          },  
          "200": {  
            ...  
          }  
        }  
      }  
    }  
  }  
}
```

...

Webszolgáltatások megvalósítása

Swagger

- A *Swagger* az OpenAPI-hoz nyújt (részben ingyenes, részben kereskedelmi) implementációs eszközöket, a legelterjedtebben használt ezen a területen. Fontosabb eszközei:
 - *Swagger Editor*: webes szerkesztő alkalmazás az OpenAPI specifikáció egyszerűbb elkészítésre
 - *Swagger UI*: webes API tesztelő felület a fejlesztett webszolgáltatáshoz.
 - *Swagger Codegen*: kód generátor, amely a webszolgáltatás OpenAPI specifikációja alapján elkészíti számos támogatott nyelven az implementáció vázát, az eljárások csonkjait.

Webszolgáltatások megvalósítása

Swagger UI

- Az ASP.NET Core MVC keretrendszerben elkészített webalkalmazásunkhoz egyszerűen generálhatunk OpenAPI specifikációt, és Swagger UI tesztelő felületet is.
 - Ehhez a **Swashbuckle.AspNetCore** vagy az **NSwag** NuGet csomagokat is használhatjuk.
- A **Swashbuckle.AspNetCore** csomag esetén a **Program** osztályban a **WebApplicationBuilder** objektumon regisztrálhatjuk az ehhez szükséges generátort.
 - Pl.:

```
builder.Services.AddEndpointsApiExplorer();
builder.Services.AddSwaggerGen(c => {
    c.SwaggerDoc("v1", new OpenApiInfo {
        Title = "Product Manager", Version = "v1"});
});
```

Webszolgáltatások megvalósítása

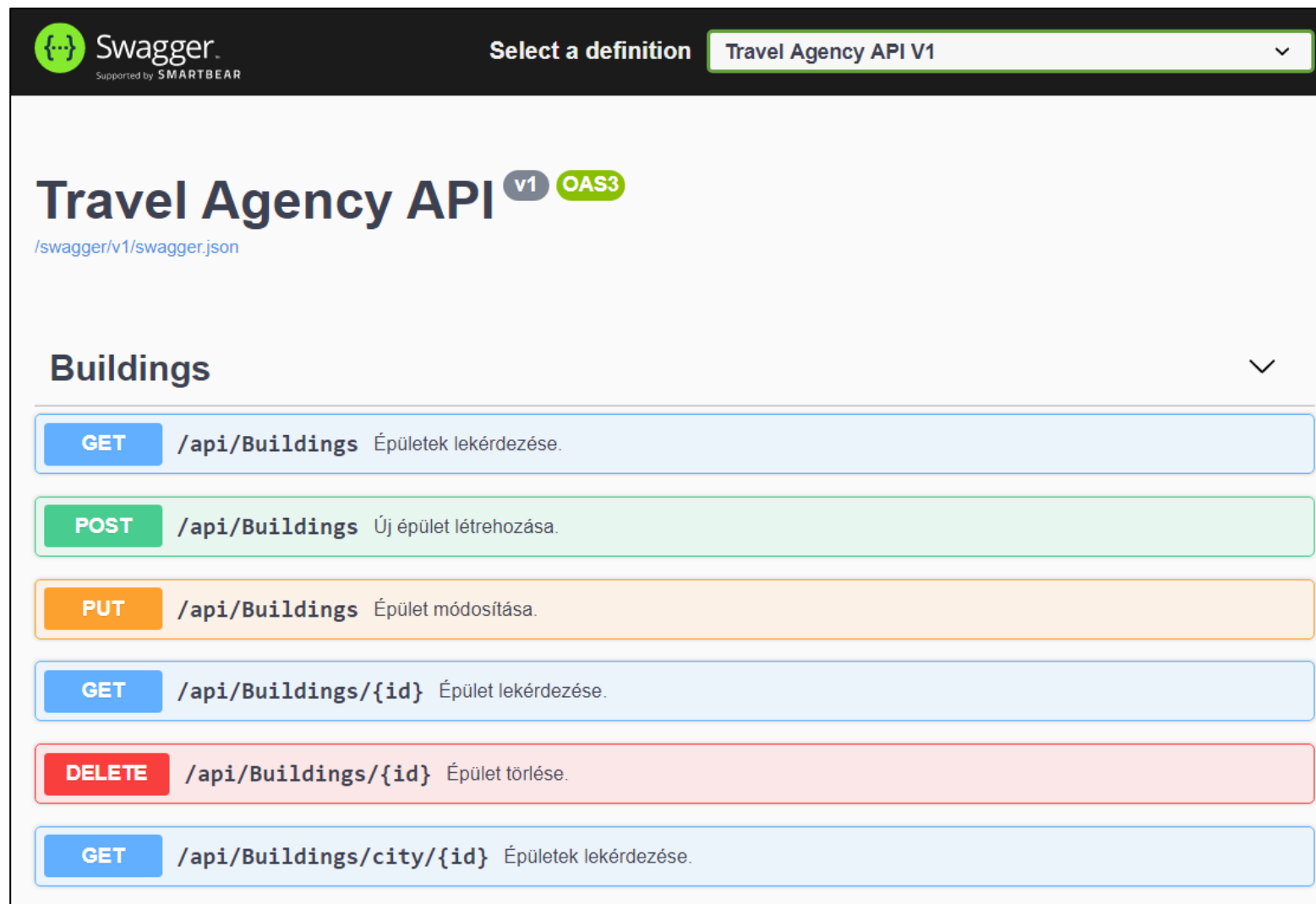
Swagger UI

- Ezt követően a **Program** osztályban a **WebApplication** objektumon engedélyezhetjük a Swagger használatát.
 - Pl.:

```
// Swagger használata (JSON végpontok)
app.UseSwagger();
// Swagger UI használata (HTML végpontok)
app.UseSwaggerUI(c => {
    // a JSON végpont megadása
    c.SwaggerEndpoint(
        "/swagger/v1/swagger.json",
        "Product Manager API V1"); });
```
- Generált URL-ek:
<http://<domain>/swagger/index.html>
<http://<domain>/swagger/v1/swagger.json>

Webszolgáltatások megvalósítása

Swagger UI



The screenshot displays the Swagger UI for the 'Travel Agency API V1'. The interface has a dark header with the Swagger logo and 'Supported by SMARTBEAR' on the left. On the right, there's a 'Select a definition' dropdown menu currently showing 'Travel Agency API V1'. Below the header, the main title 'Travel Agency API' is shown with 'v1' and 'OAS3' badges. A link '/swagger/v1/swagger.json' is provided. A section titled 'Buildings' is expanded, revealing a list of API endpoints:

- GET** `/api/Buildings` Épületek lekérdezése.
- POST** `/api/Buildings` Új épület létrehozása.
- PUT** `/api/Buildings` Épület módosítása.
- GET** `/api/Buildings/{id}` Épület lekérdezése.
- DELETE** `/api/Buildings/{id}` Épület törlése.
- GET** `/api/Buildings/city/{id}` Épületek lekérdezése.

Webszolgáltatások megvalósítása

Swagger UI

GET

/api/Buildings/{id} Épület lekérdezése.

Cancel

Parameters

| Name | Description |
|-------------------------------------------|-------------------|
| id * required integer (path) | Épület azonosító. |

Execute

Clear

Responses

Curl

curl -X GET "http://localhost:24272/api/Buildings/2" -H "accept: text/plain"

Request URL

http://localhost:24272/api/Buildings/2

Server response

| Code | Details |
|------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 200 | <div><div>Response body</div><div><pre>{ "id": 2, "name": "Petra bungaló", "cityId": 1, "seadistance": 100, "shore": 0, "features": 0, "locationX": 45.4591, "locationY": 12.5068, "comment": "Szállás bungalóban 100m strandtól (homokos tengerpart). A területen található étterem, bár, játszótér. Éjszakai nyugalmat betartani. Ott-tartózkodás szombattól szombatig. A szállásslátogató beszél is csehül. Parkoló - nem őrzött, ingyenes. Busz 100m. Vonat (állomás San Dona di Piave) 25km. Repülőtér (Venezia - Marco Polo) 40km. Cavallino - központ.'" }</pre></div><div>Download</div></div> |

Webszolgáltatások felhasználása

Példa

Feladat: Valósítsuk meg az utazási ügynökség épületeit karbantartó webszolgáltatást.

- a megoldást egy *ASP.NET Core* web API szolgáltatásként (**TravelAgency.Service**) valósítsuk meg
- az adatátvitelhez külön típust hozunk létre (**BuildingDTO**), és egy külön osztálykönyvtárba helyezzük el (**TravelAgency.Data**)
 - így később könnyen megosztható lesz egy kliens asztali projekttel, csökkentve redundanciát és egyszerűsítve a program továbbfejlesztését
- a szolgáltatáshoz generáljunk *Swagger UI* webes felületet a (manuális) tesztelés támogatására

Webszolgáltatások felhasználása

Példa

Tervezés (szolgáltatás):

- egy vezérlő (**BuildingsController**) biztosítja a CRUD műveleteket
 - hozzáadásnál visszaküldjük a hozzáadott épületet
 - módosításnál és törlésnél ellenőrizzük a kapott azonosítót

