Curs 10

Servicii Web

API web

- Scop - transferul de date, independent de limbaj sau framework-ul folosit
- ASP.NET Core suporta crearea de servicii REST, cunoscute si sub denumirea de API-uri web
- Pentru a gestiona cereri, un API web utilizeaza controale
- Controalele intr-un API web sunt clase care deriva din clasa ControllerBase

Representational **S**tate **T**ransfer

- Introdus de Roy Fielding

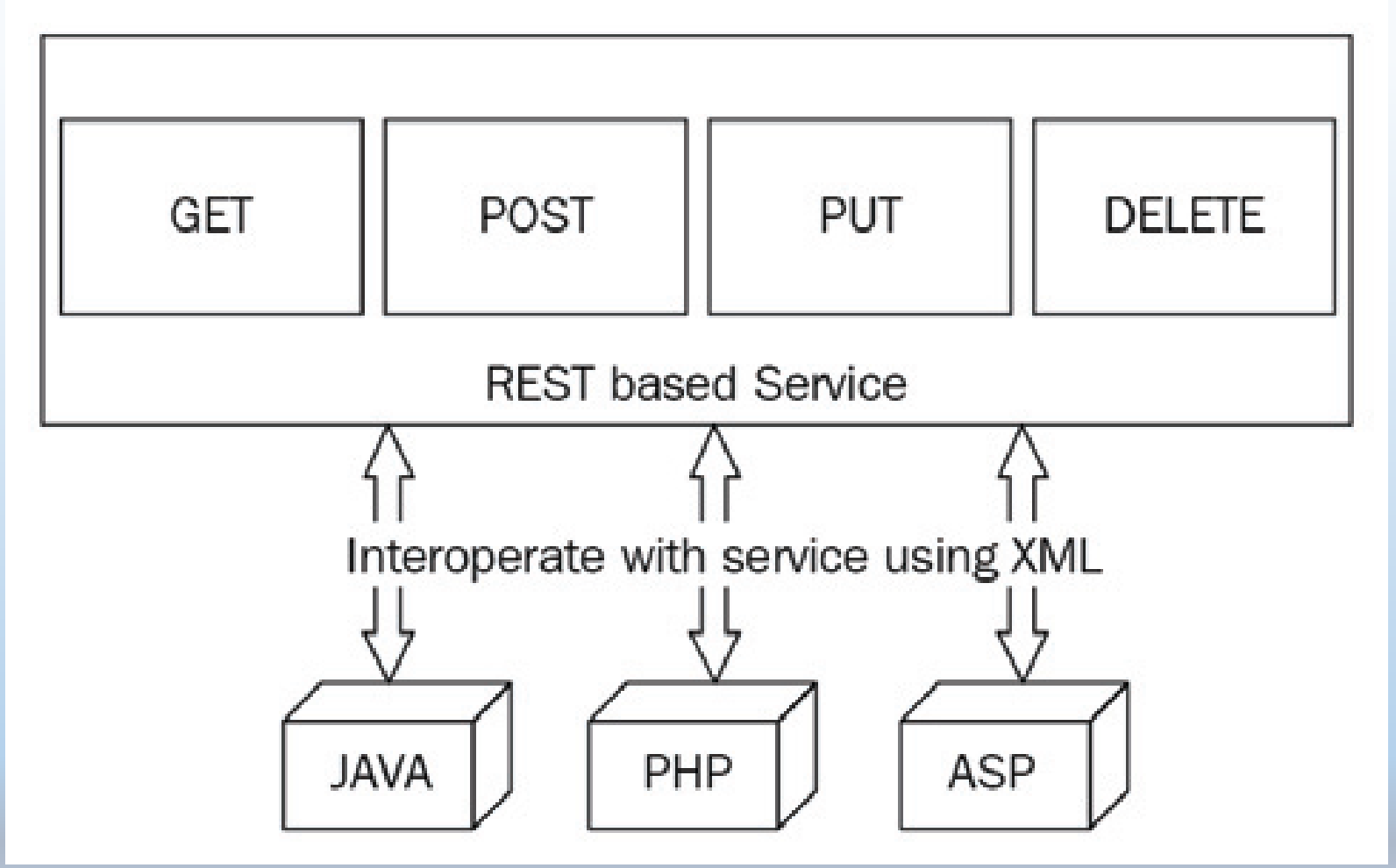
- Architectural Styles and the Design of Network-based Software Architectures, 2000

https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm

REST

- Un stil arhitectural care defineste un set de reguli pentru a construi servicii web
- Stil architectural = concept cu principii predefinite
- Implementare principilor care reprezinta componente in aplicatie
- Implementarea REST va fi diferita de la dezvoltator la dezvoltator – nu exista un stil de implementare fixat
- Diferit de pattern-uri arhitecturale precum MVC, care reprezinta concepte si implementari concrete. MVC este un pattern arhitectural avand o structura fixata care defineste cum componentele interactioneaza unele cu altele si nu pot fi implementate diferit

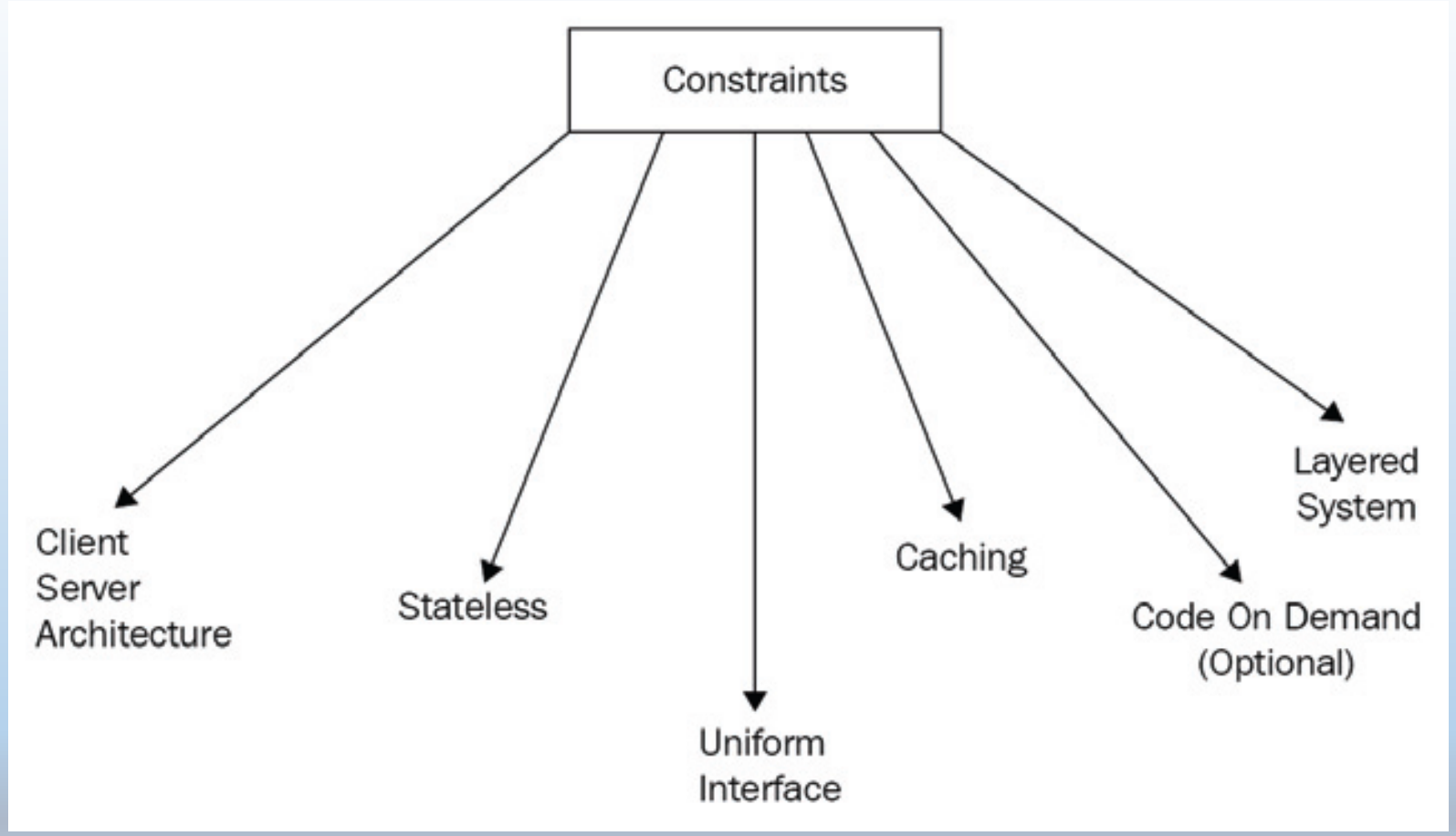
Serviciu REST



Caracteristici REST

- Raspunsul trimis de la server ca urmare a cererii clientului este o resursa intr-un anumit format
- Cele mai comune formate sunt: .json, .xml, .pdf, .doc
- REST este stateless- starea sistemului este intodeauna diferita : cand se primeste o cerere la server, cererea este gestionata si apoi uitata, asftel ca urmatoarea cerere nu depinde de starea celei anterioare.
- Fiecare cerere este gestionata de server in mod independent
- Cererile sunt realizate folosind o conexiune HTTP, fiecare continand un URI (Uniform Resource Identifier) pentru localizarea resursei solicitate

Constrangeri REST



1. Arhitectura client-server

- Clientul (consumatorul serviciului) nu trebuie sa cunoasca detalii despre cum serverul proceseaza datele si le stocheaza in baza de date

- Serverul nu trebuie sa depinda de implementarea clientului, in special de UI
- Clientul si serverul nu sunt una si aceasi entitate si fiecare poate exista in lipsa celuilalt

- Serviciul cand interactioneaza cu clientii, ofera suficiente informatii: cum sa fie consumat, ce operatii se pot efectua la utilizarea lui
- Clientul si serverul pot fi complet decuplate daca adera la toate constrangerile REST

2. Stateless

- Un serviciu REST nu mentine starea aplicatiei =>Stateless
- O cerere intr-un serviciu REST nu depinde de cererile anterioare. Serviciul trateaza fiecare cerere independent
- Fiecare cerere de la client către server trebuie să conțină toate informațiile necesare pentru înțelegerea și finalizarea cererii.
- Serverul nu poate utiliza orice informații de context stocate anterior pe server. Din acest motiv, aplicația client trebuie să păstreze în întregime starea sesiunii.

3. Caching

- Caching trebuie sa fie aplicat cand este posibil, iar resursele trebuie sa se autodeclare cacheable.
- Daca sunt cacheable, serverul stie durata pentru care raspunsul este valid
- Daca clientul are acces la un raspuns din cache valid, nu se mai executa cererea si se utilizeaza copia din cache
- Specificare explicita a unei resurse cachable se face in headerul Cache-Control unde se poate seta si durata valabilitatii copiei
- Imbunatatirea performantei pe partea de client si o mai buna scalabilitate pe partea de server deoarece se reduce incarcarea

4. Code on demand (optional)

- De cele mai multe ori serviciul returneaza reprezentari statice de resurse - .json, .xml
- Serviciul poate returna cod executabil pentru a extinde functionalitatea aplicatiei

5. System multi nivel

- Fiecare nivel este restrictionat sa acceseze doar urmatorul nivel din ierarhie
- Deployment-ul API-urilor pe serverul A, stocarea datelor pe serverul B si cererile de autentificare pe serverul C – clientul nu stie unde e conectat
- Arhitectura bazata pe layere – ajuta la o mai buna gestionare a complexitatii si imbunatateste mentenabilitatea codului.

6. Interfata uniforma

- Decuplarea clientului de serviciu
- REST este definit de patru constrangeri pentru interfata:
- **Identificarea resurselor:** Un URI este utilizat pentru a identifica o resursa; Interfața trebuie să identifice în mod unic fiecare resursă implicată în interacțiunea dintre client și server
- **Manipularea resurselor prin reprezentari:** Cand un client detine o resursa are suficienta informatie pentru a modifica sau sterge resursa; Resursele ar trebui să aibă reprezentări uniforme în răspunsul serverului. Consumatorii API ar trebui să folosească aceste reprezentări pentru a modifica starea resursei de pe server.
- **Mesaje auto-descriptive:** Mesajele trimise trebuie sa contina suficienta informatie despre datele procesate. Fiecare reprezentare a resursei ar trebui să conțină suficiente informații pentru a descrie modul de procesare a mesajului. De asemenea, ar trebui să furnizeze informații despre acțiunile suplimentare pe care clientul le poate efectua asupra resursei
- **Hypermedia ca si motor al starii aplicatiei:** Reprezentarea returnata de serviciu poate sa contina link-uri catre alte resurse; Clientul ar trebui să aibă doar URI inițial al aplicației. Aplicația client ar trebui să conducă în mod dinamic toate celelalte resurse și interacțiuni prin utilizarea hyperlinkurilor.
- REST definește o interfață uniformă pentru interacțiunile dintre clienți și servere. De exemplu, API-urile REST bazate pe HTTP folosesc metodele standard HTTP (GET, POST, PUT, DELETE etc.) și URI-urile (Uniform Resource Identifiers) pentru a identifica resursele

Metode

Metoda	Operatia realizata pe server	Tipul Metodei
GET	Citeste/Incarca o resursa	Safe
PUT	Insereaza sau actualizeaza o resursa daca aceasta exista	Idempotent
POST	Insereaza sau actualizeaza o resursa daca aceasta exista	Nonidempotent
DELETE	Sterge o resursa	Idempotent
OPTIONS	Afiseaza o lista cu operatiile permise pentru o resursa	Safe
HEAD	Returneaza doar Headerul fara body pentru cererea respectiva	Safe

Tipuri de metode

- *Safe* – operatia executata nu afecteaza valoarea initiala a resursei
- GET, OPTIONS, HEAD – doar incarca sau citesc resursa
- *idempotent* (poate fi repetata) – operatia executata are acelasi rezultat indiferent de cate ori o executam
- DELETE si PUT opereaza pe o resursa specifica, operatia putand fi repetata

POST vs. PUT

- Ambele pot sa adauge sau sa actualizeze o resursa
- POST este nonidempotent, nu este repetabil – va crea de fiecare data o noua resursa daca nu se trimite URI-ul exact
- PUT valideaza existenta resursei prima data, iar daca exista o actualizeaza, in caz contrar o creaza

- PUT https://localhost:44327/api/shoplists- nu merge pt ca nu are ID-ul resursei specificat
- PUT https://localhost:44327/api/shoplists/5 - create/update

- POST https://localhost:44327/api/shoplists- creeaza o noua resursa; la apeluri ulteriorare – inregistrari duplicat, creaza o noua resursa cu aceleasi date
- POST https://localhost:44327/api/shoplists/18- update

- PUT creaza sau actualizeaza o resursa pentru un URI specificat
- PUT si POST se comporta la fel daca resursa exista deja
- POST fara specificarea ID-ului creaza o resursa de fiecare data cand se executa

Avantaje REST

- Independenta fata de o platforma sau un limbaj de programare
- Metode standardizate prin utilizarea HTTP
- Nu pastreaza starea clientului pe server
- Suporta caching
- Accesibil tuturor tipurilor de aplicatii client: Mobile, web, desktop

Dezavantaje REST

- Daca standardele nu sunt aplicate corect vor exista dificultati pentru clientul care consuma serviciul
- Securitatea este o preocupare, daca nu exista procese care sa restrictioneze accesul la resurse

Coduri de stare

- Cand serverul returneaza raspunsul returneaza si codul de stare pentru a informa clientul despre cum s-a executat cererea pe server

- **200 OK** Raspuns standard pentru cereri HTTP executate cu success
- **201 CREATED** Raspuns standard pentru o cerere cand o resursa a fost creata cu success

- **204 NO CONTENT** Raspuns standard pentru cereri executate cu success dar nu s-a returnat nimic in body
- **400 BAD REQUEST** Cererea nu poate fi procesata datorita sintatxei gresite, marimii prea mari sau alt motiv
- **403 FORBIDDEN** Clientul nu are permisiunea sa acceseze resursa solicitata
- **404 NOT FOUND** Resursa nu exista
- **500 INTERNAL SERVER ERROR** Cand apare o eroare sau o exceptie la procesarea codului pe server

Coduri de stare implicite

- GET: Returneaza **200 OK**
- POST: Returneaza **201 CREATED**
- PUT: Returneaza **200 OK**
- DELETE: Returneaza **204 NO CONTENT** daca operatia a esuat

RESTful APIs

- Un serviciu web care respecta principiile REST se numeste RESTful API
- In mod uzual serviiciile web RESTful utilizeaza mesaje JSON pentru a returna date clientului.
- JSON – format de date de tip text de dimensiuni reduse – data-interchange

```
{
 "id":1
  "amount": 8.50,
  "date":"2020-05-09T00:00:00Z",
  "description": "tea"
}
```

Maparea URI

- Cand un Web API primeste o cerere ruteaza cererea respectiva la o actiune
- Actiunile – metode in clasa Controller
- Gaseste URI pe baza specificarilor de rutare definite astfel:
  - Pentru a gasi controllerul adauga “controller” la valoarea variabilei {controller}-shoplists
  - Pentru a gasi actiunea cauta in actiunile definite in controller care sunt marcate cu acelasi atribut HTTP
  - {id} este mapat cu parametrul unei actiuni.
- https://192.168.1.9:45455/api/shoplists/{0}

Atribute

- [ApiController] – aplicat la controale specifice

```
[ApiController]
[Route("[controller]")]
public class WeatherForecastController : ControllerBase
```

[ApiController] – controale multiple

- Cream o clasa de baza

```
[ApiController] public class MyControllerBase :
ControllerBase { }

[Route("[controller]")]
public class PetsController : MyControllerBase
```

Atributul [Route]

- Cand utilizam atributul [ApiController] este necesar atributul [Route]

```
[ApiController]
[Route("[api/controller]")]
public class WeatherForecastController : ControllerBase
```

Action methods sunt inaccesibile prin rutele conventionale specificate in Program.cs

Tipul de return

- ActionResult

Mai multe tipuri de ActionResult sunt posibile intr-o actiune.

Tipurile ActionResult reprezinta diferite coduri HTTP status:

- BadRequestResult(400) / BadRequest())
- NotFoundResult(404)
- OkObjectResult(201)

Clasa HttpClient si HttpResponseMessage

- Clasa HttpClient este utilizata pentru a trimite si primi cereri folosind protocolul HTTP
- Ofera functionalitati pentru a trimite cereri HTTP si a primi raspunsuri HTTP de la un URI
- Clasa HttpResponseMessage reprezinta un mesaj de tip raspuns primit de la un serviciu web in urma realizarii unei cereri HTTP
- Contine informatii despre raspuns, incluzand codul de stare
- Continutul poate fi citit cu metoda ReadAsStringAsync

GET

```
[Produces("application/json")]
[ApiController] //indica faptul ca clasa va fi utilizata pentru raspunsuri HTTP
[Route("[controller]")] //conventie de nume - la runtime se va concatena cu numele textului din URL
 public class ShopListsController : ControllerBase
  {
  [HttpGet("{id}")] //identifica o actiune asociata cu metoda GET
     public async Task<ActionResult<ShopList>> GetShopList(int id)
     {
         var shopList = await _context.ShopList.FindAsync(id);
         if (shopList == null)
         {
             return NotFound();
         }
         return shopList;
     }
```

POST

```
[HttpPost]
        public async Task<ActionResult<ShopList>>
PostShopList(ShopList shopList)
        {
            _context.ShopList.Add(shopList);
            await _context.SaveChangesAsync();


            return CreatedAtAction("GetShopList", new { id =
shopList.ID }, shopList); // returneaza codul de stare 201


        }
```

PUT

```
[HttpPut("{id}")] //primeste ID-ul din URL

        public async Task<IActionResult> PutShopList(int id, ShopList shopList)
        {
            if (id != shopList.ID)
            {
                return BadRequest();
            }

            _context.Entry(shopList).State = EntityState.Modified;

            try
            {
                await _context.SaveChangesAsync();
}
            catch (DbUpdateConcurrencyException)
            {
                if (!ShopListExists(id))
                {
                    return NotFound();
                }
                else
                {
                    throw;
                }
            }
 return NoContent();
        }
```

DELETE

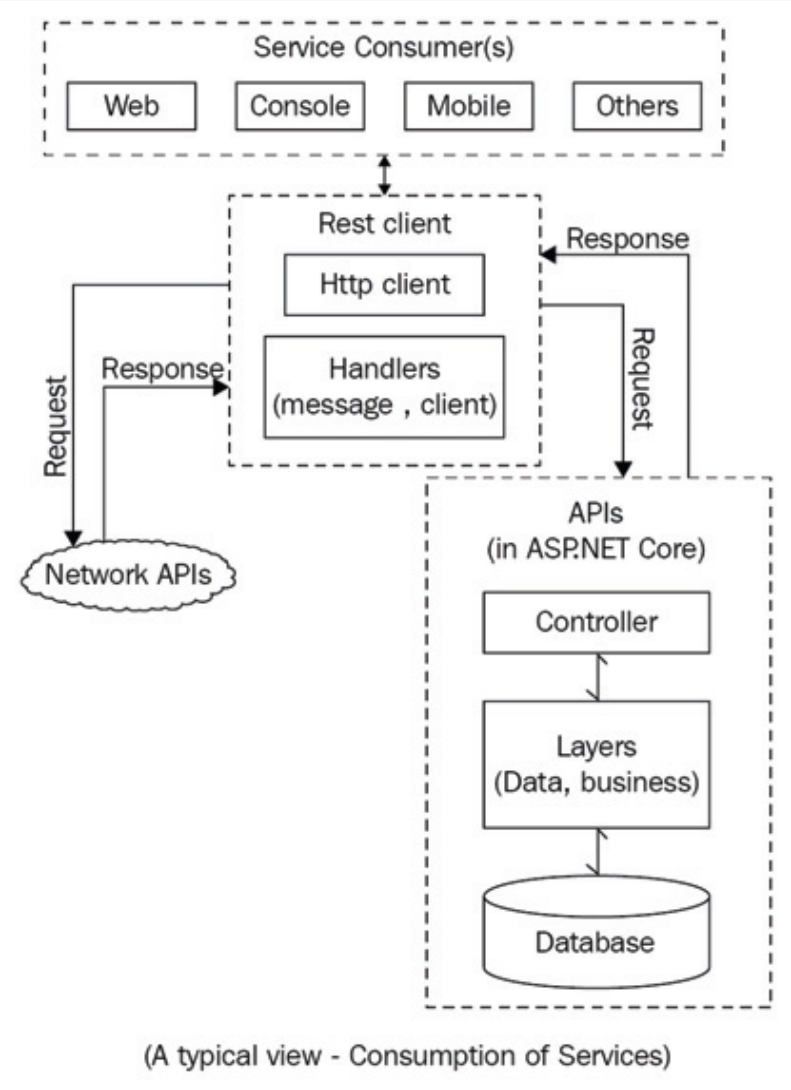
```
[HttpDelete("{id}")]
        public async Task<ActionResult<ShopList>> DeleteShopList(int id)
        {
            var shopList = await _context.ShopList.FindAsync(id);
            if (shopList == null)
            {
                return NotFound();
            }

            _context.ShopList.Remove(shopList);
            await _context.SaveChangesAsync();

            return shopList;
        }
```

Consumarea serviciilor web

- Serviciile REST pot sa fie sau nu parte a unei aplicatii Web
- O aplicatie Web poate sa apeleze sau sa consume servicii web externe sau servicii care fac parte din aceasi aplicatie
- Programul care permite interactiunea (cerere,raspuns) intre serviciu si aplicatia care consuma acel serviciu se numeste *client*



(A typical view - Consumption of Services)