



**Eötvös Loránd Tudományegyetem
Informatikai Kar**

Webes alkalmazások fejlesztése

5. előadás

Állapotfenntartás (ASP.NET Core)

Cserép Máté

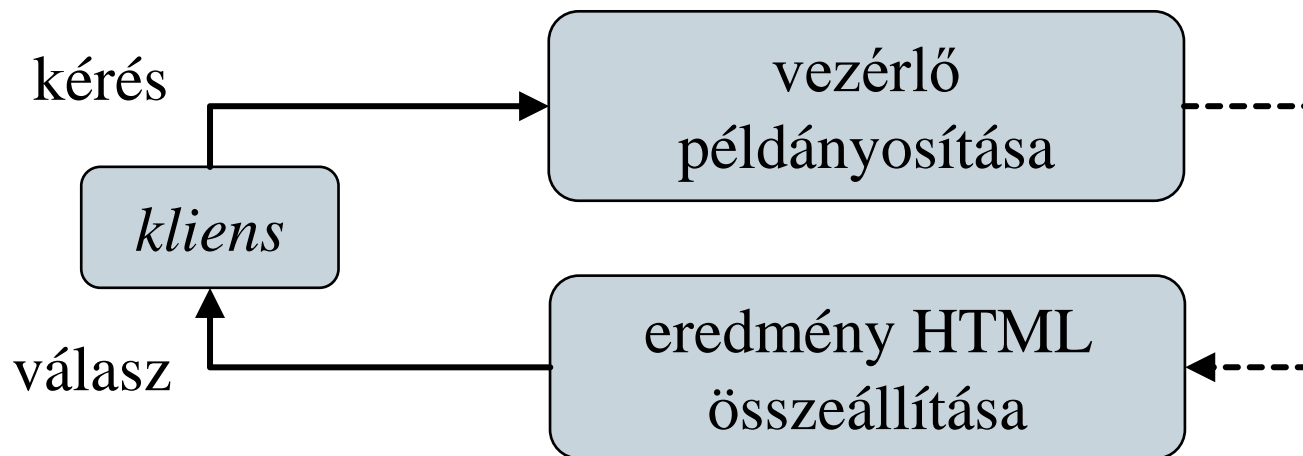
mcserep@inf.elte.hu

<https://mcserep.web.elte.hu>

Állapotfenntartás

A HTTP protokoll

- A HTTP protokoll a kérés/válasz paradigmára épül, vagyis a kliens elküld egy kérést, amelyre a szerver (alkalmazás) válaszol
 - a kérések egymástól függetlenül kerülnek kiszolgálásra
 - minden kiszolgáláshoz külön objektumok jönnek létre, amelyek előállítják a választ, majd megsemmisülnek



Állapotfenntartás

Eszközök

- Két kérés között sokszor szeretnénk megőrizni az állapotot
 - pl. a felhasználó bejelentkeztetése, az űrlap mezők kitöltései
 - objektumokkal erre nincs lehetőség (a mezők megsemmisülnek), osztályszinten pedig nincs garancia a megőrzésre
- Az állapotot a szerver speciális eszközökkel tudja fenntartani
 - kliens oldalon: elérési útvonal, weblap értékei (űrlapmezők, rejtett mezők), *sütik*
 - szerver oldalon:
 - egy kliensre: *munkamenet*
 - minden kliensre: *alkalmazás*

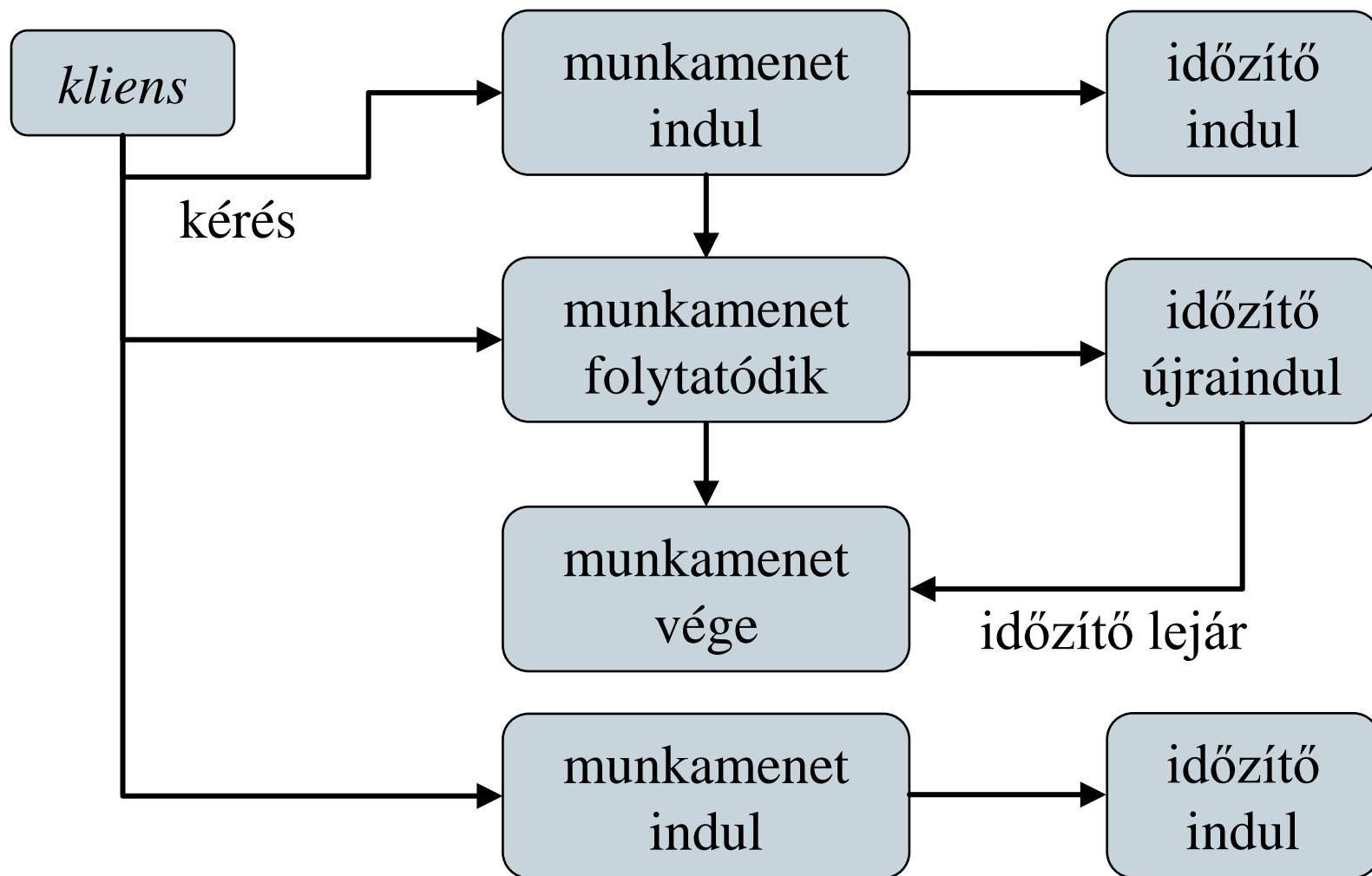
Állapotfenntartás

Munkamenet állapotok

- A munkamenet (*session*) egy kliens weblapon történő tartózkodása, és közben végrehajtott tevékenységei
 - minden kliens rendelkezik (pontosan) egy saját munkafolyamattal a szerveren
 - automatikusan elindul, amikor a kliens először kérést küld a szerverre
 - automatikusan végződik, amikor a kliens egy megadott ideig nem intéz kérést, ezt egy időzítő felügyeli, amely minden kéréssel újraindul (*session timeout*)
 - a klienst a kérés paraméterei (IP cím, böngésző, süti, ...) alapján azonosítja, ami megghamisítható (*session hijacking*)

Állapotfenntartás

Munkamenet állapotok



Állapotfenntartás

Munkamenet állapotok

- A munkamenetet a szerver oldalon a **Program** osztályban kell konfigurálni és hozzáadni a webalkalmazáshoz:
 - adjuk hozzá a munkamenet kezelést a szolgáltatásokhoz a **WebApplicationBuilder** objektumon keresztül:
`builder.Services.AddSession()` ;
 - Itt paraméterezhetjük is a munkamenet működését, pl. az **IdleTimeout** tulajdonsággal állítható a munkamenet lejáratí ideje (alapértelmezetten 20 perc)
 - a **WebApplication** példány `app.UseSession()` ;
utasításával használhatjuk a beállított munkamenet szolgáltatást.

- A munkamenetek a szerver oldalon kerülnek tárolásra.
 - Megadhatjuk a munkamenetek tárolásához használt *cache*-t is, használhatunk pl. memóriabeli tárolást:
`builder.Services.AddMemoryCache();`
 - .NET Core 3 óta ezt alapértelmezetten engedélyezi a
`builder.Services.AddControllersWithViews()`
eljárás meghívása.
 - Nagyobb forgalmú alkalmazásoknál használhatunk elosztott memóriabeli tárolást a terhelés csökkentése érdekében.
`builder.Services.AddDistributedMemoryCache();`
 - Támogatott a relációs adatbázis és a *Redis* cache is.
`AddDistributedSqlServerCache();`
`AddDistributedRedisCache();`

Állapotfenntartás

Munkamenet állapotok

- A munkamenethez szerver oldalon bármikor hozzáférhetünk a vezérlő/nézet **Session** tulajdonságán keresztül, vagy máshol a **HttpContext.Session** tulajdonságon keresztül
 - ebben kulcs/érték párokként elhelyeztünk az adott kliensre vonatkozó adatokat, amelyeket a szerver a memóriában tárol (a munkafolyamat megszűnéséig), pl.:
`Session.SetString("myKey", myString);`
`Session.SetInt32("myKey", myInteger)`
`Session.Set("myKey", myByteArray);`
 - a munkamenet azonosítója a **Session.Id** tulajdonsággal kérhető le
 - az adatokhoz a kliens nem férhet hozzá

Állapotfenntartás

Munkamenet állapotok

- Komplex típus munkamenetben történő tárolása is megvalósítható kiterjesztő metódusok (*extension methods*) által, például *JSON* szerializáció révén:

```
public static class SessionExtensions {  
    public static void Set<T>(this ISession session,  
                              string key, T value) {  
        session.SetString(key,  
                           JsonConvert.SerializeObject(value));  
    }  
    public static T Get<T>(this ISession session,  
                           string key) {  
        var value = session.GetString(key);  
        return value == null ? default(T) :  
            JsonConvert.DeserializeObject<T>(value);  
    }  
};
```

- Az *adatlopás* elkerülése végett fontos, azonosításra szolgáló adatokat mindig kódoltan tároljuk az adatbázisban
 - a kódoláshoz egyirányú kódoló algoritmusokat használunk (pl. *MD5*, *SHA1*, *SHA512*), amelyek nem fejthetők vissza, viszont azonosításra használhatóak
 - a kódoló eljárások a **System.Security.Cryptography** névtérben helyezkednek el
 - a kódolás előtt és/vagy után célszerű megsózni a jelszót (*password salt*), azaz tegyünk bele extra karaktereket és byte-okat, hogy megnehezítsük a jelszó visszakeresését
 - a só lehet fix, véletlenszerű, vagy időfüggő, ilyenkor magát a sót is eltárolhatjuk az adatbázisban

Állapotfenntartás

Adattitkosítás

- Pl.:

```
String pwdText = ... // jelszó szöveges alakja
SHA512 coder = SHA512.Create();
    // SHA512 kódoló objektum
```

```
Byte[] pwdBytes = coder.ComputeHash(
    Encoding.UTF8.GetBytes(pwdText));
    // kódolás végrehajtása a szövegből kiolvasott
    // UTF8 értékeken, az eredmény 160 bites lesz
```

```
Byte[] storedBytes = ...
    // kinyerjük az eltárolt kódolt jelszót
if (pwdBytes.SequenceEquals(storedBytes)) { ... }
    // ha a kettő megegyezik, jó a jelszó
```

Feladat: Valósítsuk az utazási ügynökség weblapjának felhasználó kezelési funkcióját.

- a felhasználók regisztrálhatnak, és adataikat foglaláskor automatikusan kitölti a weblap
 - a regisztráció nem kötelező, az újonnan megadott adatok a korábbiak szerint mentődnek (automatikusan generált felhasználónévvel)
- egy új vezérlőben (**AccountController**) kezeljük a regisztráció (**Register**), bejelentkezés (**Login**) és kijelentkezés (**Logout**) funkciókat
 - a regisztráció és a bejelentkezés megfelelő nézeteket kapnak, űrlapokkal

Állapotfenntartás

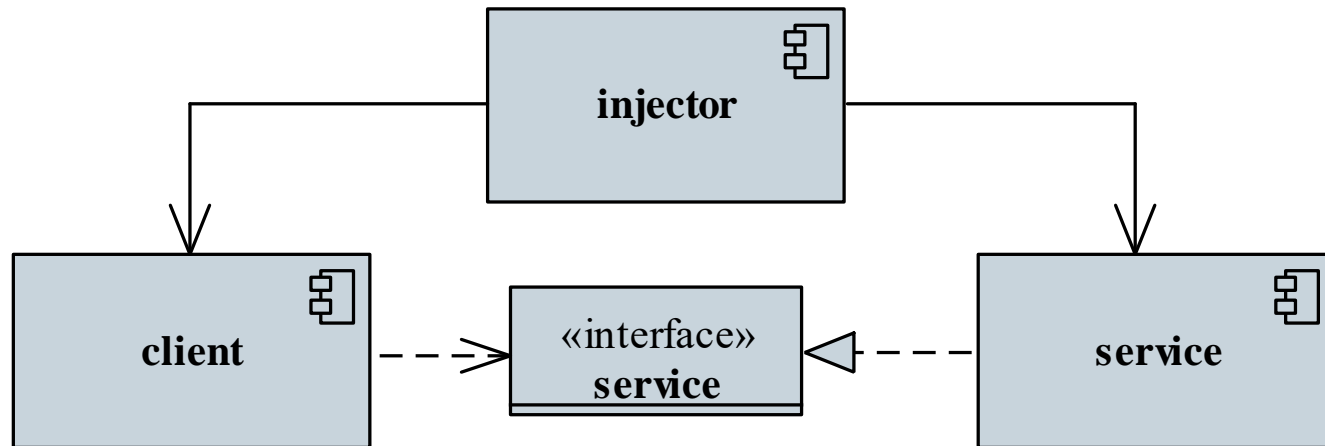
Példa

- a funkciókat az **AccountService** osztály hajtja végre, amely megvalósítja az **IAccountService** interfészt
 - a bejelentkezés, kijelentkezés és regisztráció mellett lekérhetjük egy adott vendég adatait (**GetGuest**), és létrehozhatunk vendéget regisztráció (felhasználói adatok) nélkül (**Create**)
- a nézetmodell bővül a bejelentkezés (**LoginViewModel**), illetve a regisztráció (**RegistrationViewModel**) adataival
 - mivel több adat közös a foglалás és a regisztráció között, egy őssosztályba (**GuestViewModel**) általánosítunk

Állapotfenntartás

Függőség befecskendezés

- A végrehajtás során egy réteg (*client*) által használt szolgáltatás (*service*) egy, az adott körülmények függvényében alkalmazható megvalósítása kerül alkalmazásra
- A szolgáltatás konkrét példánya meghatározható függőség befecskendezés segítségével, amely során egy külső programkomponens (*injector*) állapítja meg a függőséget



Állapotfenntartás

Függőség befecskendezés

- A szolgáltatások befecskendezése szükségessé teszi a megvalósítás statikus (fordítási időben) történő ismeretét, ez korlátozza a program hasznosítását
 - nem változtatható a megvalósítás futás közben, noha a körülmények változhatnak
 - nem bővíthető a program újabb megvalósítással
- Az *IoC tároló* (*IoC container*) egy olyan *Inversion of Control* paradigmájú komponens, amely lehetőséget ad szolgáltatások megvalósításának dinamikus (futási idejű) betöltésére
 - egy központi regisztráció, amelyet minden programkomponens elérhet, és felhasználhat

- a típusokat (elsősorban) interfész alapján azonosítja, és az interfészhez csatolja a megvalósító osztályt
- a tárolóba történő regisztrációkor megadjuk a szolgáltatás interfészét és megvalósításának típusát (vagy példányát)
- a szolgáltatást interfész alapján kérjük le, ekkor példányosul a szolgáltatás vagy kapunk egy már létező példányt
 - amennyiben a szolgáltatásnak függősége van, a tároló azt is példányosítja
- Az *ASP.NET Core* keretrendszer egy általánosan használható IoC tárolót biztosít, lehetővé téve a szolgáltatások regisztrációját, lekérést és konstruktoron keresztüli befecskendezését (*constructor injection*)

Állapotfenntartás

Szolgáltatások regisztrációja az ASP.NET Core IoC tárolójába

- A szolgáltatásokat a **Program** osztályban regisztráljuk, egyben megadjuk a szolgáltatás példányok élettartamát, amely lehet:
 - **transient**: minden kérésre új példányosítás
 - **scoped**: egy oldallekéréshez egy példányosítás
 - **singleton**: alkalmazás szinten egyetlen példányosítás a *Singleton* (egyke) tervezési mintát követve
- Pl.: **builder.Services**
.AddTransient<IMyService, MyService>() ;
- Az **AddDbContext** és a **Configure** eljárások a szolgáltatás regisztráció speciális esetei adatbázis kontextus és alkalmazás konfigurációs osztályok regisztrálására.

Állapotfenntartás

Szolgáltatások lekérése az ASP.NET Core IoC tárolójából

- A regisztrált szolgáltatások egy megfelelő példányát lekérhetjük manuálisan a *service provider* **GetService** vagy a **GetRequiredService** metódusával, utóbbi kivételt vált ki sikertelenség esetén.
 - pl.:
`app.Services.GetService<IMyService>() ;`
 - Ezt alkalmaztuk a **DbInitializer** osztály esetében is.
- Kényelmesebb megoldást nyújt a szolgáltatások konstruktoron keresztüli befecskendezése, ilyenkor a szolgáltatások lekérése a keretrendszer által automatikus.
 - Ezt alkalmaztuk többek között a vezérlőkbe a modell szolgáltatás osztályok befecskendezésekor is.

Állapotfenntartás

Alkalmazás állapotok

- Az egész web alkalmazásra vonatkozó, globális információkat egy tetszőleges osztályban tárolhatunk, amelyet egyszer, az alkalmazás teljes élettartamára, *singleton* módon példányosítunk.
 - pl.:
`builder.Services`
`.AddSingleton<IMyApplicationState,`
`MyApplicationState>() ;`
- Amennyiben csak egyetlen implementáció létezik, nem szükséges interfészt kiemelni az osztályból.
 - pl.:
`builder.Services`
`.AddSingleton<MyApplicationState>() ;`

Állapotfenntartás

Alkalmazás állapotok

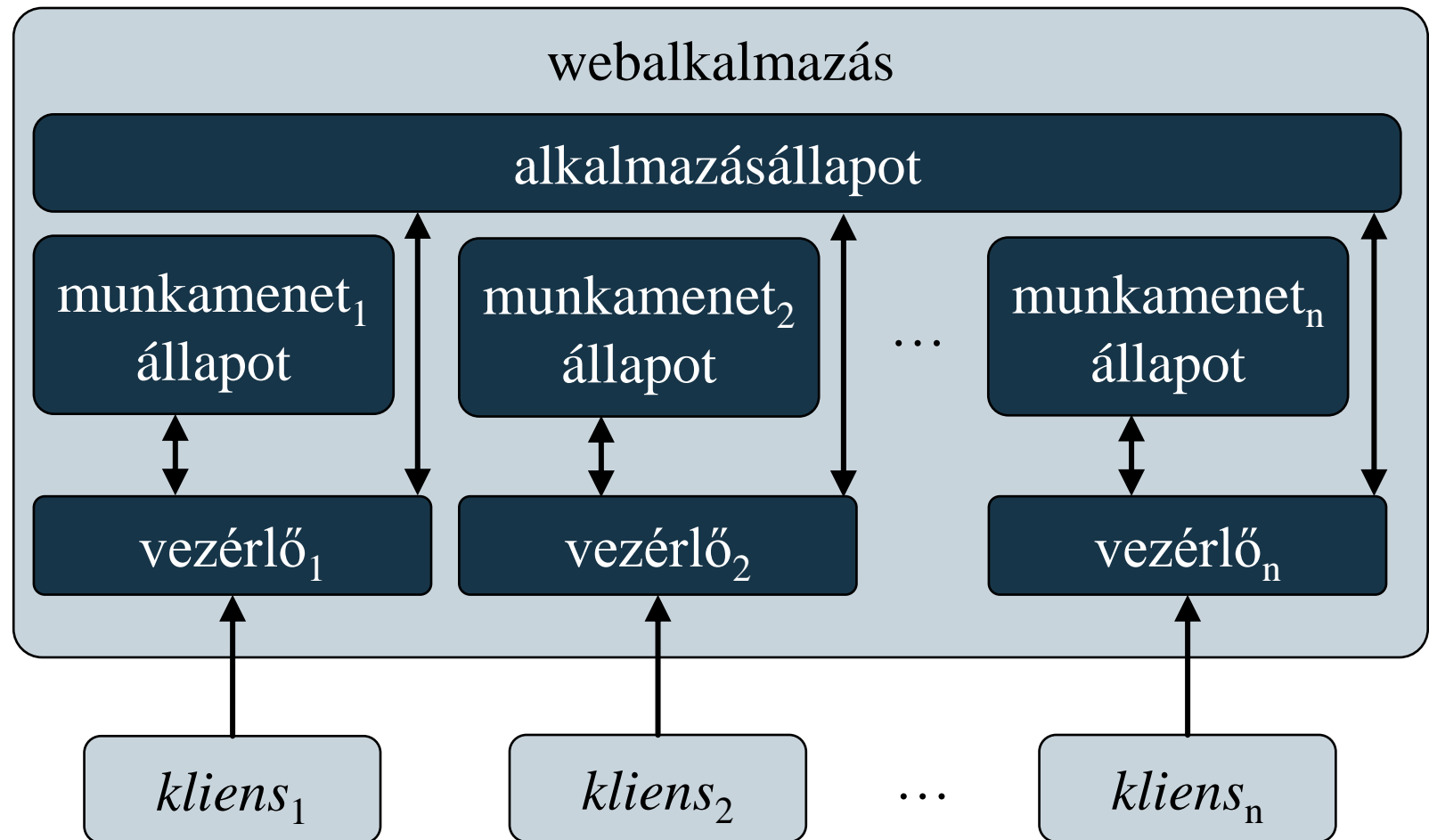
- Az alkalmazás állapotát reprezentáló osztályból lekérhetünk egy példányt konstruktoron keresztüli befecskendezéssel.

```
public class HomeController {  
    MyApplicationState _state;  
  
    public HomeController(MyApplicationState state)  
    {  
        _state = state;  
    }  
    // akciók ...  
};
```

- Mivel egy *singleton* objektumhoz párhuzamosan többen hozzáférhetnek, ezért kezelni kell az adatok esetleges konkurens módosítását.

Állapotfenntartás

Alkalmazás állapotok



- A HTTP *süti* olyan információgyűjtemény, amelyet a kliens eltárol egy fájlban, így az oldal későbbi látogatása során a felhasználóra vonatkozó adatok abból visszatölthetők
 - a sütik adott webcímre vonatkoznak, rendelkeznek névvel valamint egy tárolt szöveges értékkel, pl.:
`HttpContext.Response.Cookies`
`.Append("MyCookie", value, options);`
 - megadhatunk lejáratot, amely elteltével a süti törlődik, pl.:
`var options = new CookieOptions {`
`Expires = DateTime.Now.AddDays(10)`
`};`
 - az adott webcímhez tartozó sütiket a böngésző automatikusan továbbítja a kérésben

- Sütiket a vezérlőben kezelhetjük
 - a kéréssel küldött sütit a **Request.Cookies** gyűjteményben találjuk, pl.:
String value = Request.Cookies["MyCookie"] ;
 - a válaszhoz sütit a **Response.Cookies** gyűjteménybe helyezhetjük, pl.:
Response.Cookies.Append("MyCookie", value) ;
 - sütit úgy törölhetünk, hogy az érvényességét lejárt időpontra állítjuk (és így a böngésző kitörli), vagy:
Response.Cookies.Delete("MyCookie") ;
- A munkafolyamatok klienseinek beazonosításához is sütit használunk, ez a munkafolyamat süti (neve alapértelmezetten **.AspNetCore.Session**)

- Pl.:

```
[HttpPost]
public IActionResult LoginUser(UserData user) {
    ...
    if (user.RemberMe) {
        // ha kérte az azonosító megjegyzését
        Response.Cookies.Append(
            "challenge", user.Challenge,
            new CookieOptions {
                Expires = DateTime.Today.AddDays(365)
            });
        // egyedi azonosítót elküldünk a kliensnek
    }
    return View(...);
}
```


Állapotfenntartás

Sütik

```
[HttpGet]
public IActionResult LoginUser() {
    UserData user = ...
    // amikor legközelebb betölti az oldalt

    if (Request.Cookies.ContainsKey("challenge")) {
        // és megjegyeztette az azonosítót

        user = LoadUserByChallenge(
            Request.Cookies["challenge"]);
        // beállítjuk előre az azonosítót
    }
    return View(user);
}
```

Állapotfenntartás

Sütik biztonságos kezelése

- Mivel a sütik szolgáltatják a kliens oldali információ tárolás (és benne a munkamenet tárolás) alapját, különösen figyelni kell a biztonságukra
 - felhasználói adatokat (különösen jelszavakat) direkt módon ne tároljuk sütiben
 - a sütik tartalmát kódolhatjuk, vagy helyettesíthetjük speciális azonosítókkal
 - szabályozható, hogy kliens oldali szkriptek ne férjenek hozzá a sütihez (**HttpOnly**)
 - szabályozható, hogy csak biztonságos (TSL/SSL) kapcsolat esetén továbbítódjanak (**Secure**)

- Amennyiben sütiket használunk a felhasználó azonosítására, különös tekintettel kell lennünk a biztonságra
 - az információt osszuk el több sütibe
 - jelszavak helyett használjunk egyedi azonosítókat, *tokeneket* (**Guid**), amelyeket mindkét oldalon eltárolunk
 - az azonosítót cserélhetjük minden bejelentkezéssel
 - a felhasználói azonosítók mellett tárolhatunk felhasználó-specifikus információkat
 - a **HttpContext** és a **Request** tulajdonság számos információt tartalmaz a kliensről
(**Connection.RemoteIpAddress**, **Request.Host**, ...),
amik szintén elmenthetők (kódolva) a sütibe

Állapotfenntartás

Állapotkezelés szolgáltatásokban

- A **HttpContext** tulajdonság a vezérlőkben (**Controller** osztály leszármazottai) érhető el.
- Egyéb osztályban, pl. egy modell szolgáltatásban egy **IHttpContextAccessor** objektum befecskendezésével férhetünk hozzá.

- Pl.:

```
public class SomeService {  
    private readonly HttpContext _httpContext;  
    public SomeService(IHttpContextAccessor  
                        accessor) {  
        _httpContext = accessor.HttpContext;  
    }  
}
```

Állapotfenntartás

Állapotkezelés szolgáltatásokban

- Az alapértelmezett `HttpContextAccessor` típus általi implementáció regisztrációját is el kell végeznünk az IoC tárolóba a `Program` osztályban:
 - `builder.Services`
 `.AddSingleton<IHttpContextAccessor,`
 `HttpContextAccessor>() ;`
 - Ekvivalens módon használhatjuk az `AddHttpContextAccessor()` kiterjesztő metódust is.

Feladat: Valósítsuk meg az utazási ügynökség weblapjának felhasználó kezelési funkcióját.

- lehetőséget adunk a felhasználónak a bejelentkezés megjegyzésére
 - az azonosító megjegyzéséhez sütit használunk, amelyet bejelentkezést követően továbbítunk a felhasználónak (a sütiben a felhasználónevet tároljuk)
- a felületen megjelenítjük az oldalt böngésző, bejelentkezett felhasználók számát, ezt alkalmazás állapotban tároljuk

Feladat: Valósítsuk meg az utazási ügynökség weblapjának felhasználó kezelési funkcióját.

- a munkafolyamat és az alkalmazásállapot kezelését áthárítjuk az **AccountService** osztályra, így a vezérlő mentesül az állapotkezeléstől
 - a konstruktor ellenőrzi a sütit, és tölti be a munkafolyamatba
 - tulajdonságok segítségével kérdezzük le az aktuális felhasználót (**CurrentUserName**), illetve a felhasználók számát (**UserCount**)
- az alkalmazás szintű állapotot az **ApplicationState** osztály *singleton* példányában tároljuk