



**Eötvös Loránd Tudományegyetem
Informatikai Kar**

Webes alkalmazások fejlesztése

10. fejezet

Szolgáltatás alapú rendszerek megvalósítása (ASP.NET WebAPI)

Giachetta Roberto

**A jegyzet az ELTE Informatikai Karának 2016. évi
jegyzetpályázatának támogatásával készült**

Szolgáltatás alapú rendszerek megvalósítása

Konfiguráció

- A .NET alkalmazások konfigurációját általában konfigurációs fájlban tároljuk (**app.config**, vagy **web.config**), amely számos paraméterét tartalmazhatja a működésnek, pl.:
 - a működés során változó beállítások (pl. szolgáltatás címe, adatbázis elérése)
 - az alkalmazás felépítéséhez szükséges adatok (pl. befecskendezett osztályok)
 - a platformmal kapcsolatos beállítások (pl. .NET verzió, csomagok)
 - a konfiguráció automatikusan átkerül a fordítási könyvtárba, és átveszi a futtatható állomány nevét

Szolgáltatás alapú rendszerek megvalósítása

Konfiguráció

- A fájl **appSettings** eleme tartalmazza az egyedi beállításokat (kulcs/érték párokként), amelyeket a programban lekérhetünk

- Pl.:

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <runtime>... <!-- platform -->
  <connectionStrings>... <!-- adatbázisok -->
  <appSettings>
    <!-- alkalmazás beállításai -->
    <add key="ServiceAddress"
        value="http://localhost:19243" />
    <!-- a szolgáltatás címe -->
    ...
  
```

Szolgáltatás alapú rendszerek megvalósítása

Konfiguráció elérése

- A konfigurációt kódban a **ConfigurationManager** osztály segítségével kezelhetjük
 - a beállításokat az **AppSettings** gyűjteményben találjuk (kulcs/érték párokként)
 - az értéket szöveggént kapjuk meg
 - amennyiben a konfiguráció nem található, vagy a beállítás nincs a konfigurációban **null** értéket kapunk
 - pl.:

```
String serviceAddress =  
    ConfigurationManager  
        .AppSettings["ServiceAddr"] ;
```

Szolgáltatás alapú rendszerek megvalósítása

Eseménynaplózás

- Összetett rendszereknél célszerű a tevékenységek követésére *eseménynaplót* vezetni (*event logging*)
 - célja, hogy megértsük a szoftver végrehajtási folyamatát, teljesítményét, könnyebben azonosítsuk a hibákat és a biztonságra veszélyes tevékenységeket
 - különösen fontos, ha nincs felhasználói interakció (pl. szolgáltatások)
 - a napló lehet egy fájl, adatbázis, vagy külső szolgáltatás, amely biztosítja a napló elemzését is (pl. LogStash)
 - a naplóbejegyzések rendelkeznek időponttal (*when*), hellyel (*where*), azonosítóval (*who*) és leírással (*what*)

Szolgáltatás alapú rendszerek megvalósítása

Eseménynaplózás

- A naplózott események köre az alkalmazás jellegétől függ, célszerű naplózni:
 - alkalmazásbeli hibák (pl. csatlakozás, konfiguráció, külső hívások, teljesítmény), váratlan események
 - magasabb kockázatú tevékenységek (pl. felhasználó azonosítás és hozzáférés, felhasználó hozzáadása/törlése, rendszerbeli folyamatok igénybevétele, konfiguráció változtatás)
 - validációs események (pl. bemenő adatok hibái)
 - eseménynaplózás tevékenységei (indítás, leállítás, szüneteltetés)

Szolgáltatás alapú rendszerek megvalósítása

Eseménynaplózás

- A bejegyzés szintje adja meg az üzenet fontosságát, célját, pl.:
 - *fatális* (**fatal**): olyan hibaesemény, amely miatt az alkalmazás összeomlott
 - *hiba* (**error**): olyan hibaesemény, amely után az alkalmazás folytatta munkáját (de keletkezhetett hibás adat)
 - *figyelmeztetés* (**warn**): esetleges mellékhatás, hibalehetőség
 - *információ* (**info**): egyéb információ
 - *tesztelés* (**debug**): a fejlesztéshez és teszteléshez használt információ
 - *nyomkövetés* (**trace**): a felmerült hiba pontos leírása

- Több programcsomag is elérhető, amely biztosítja az eseménynaplózást, az egyik legnépszerűbb az *NLog*
 - a naplózást a **Logger** osztály biztosítja, és annak szintnek megfelelő műveletei (**Info**, **Error**, ...)
 - az üzenetek mellett kivételek naplózását is megkönnyíti
 - a naplót adott névre, vagy osztályra hozhatjuk létre (`LogManager.GetCurrentClassLogger()`, `LogManager.GetLogger(<név>)`)
 - konfigurációs fájlban beállítható az naplózás módja, formája és szintje
 - alapértelmezetten az **NLog.config** fájl, de használjuk az alkalmazás konfigurációját is

Szolgáltatás alapú rendszerek megvalósítása

Eseménynaplózás

- Pl.:

```
Logger myLogger = LogManager.GetLogger("model");
myLogger.Info("Processing started.");
    // információ kiírása
try {
    ... // feldolgozás
    myLogger.Info("Processing finished.");
}
catch (Exception ex) {
    myLogger.Error("Processing aborted.");
    // hibajelzés
    myLogger.Trace("Exception occurred. ", ex);
    // kiírjuk a kivétel részleteit
}
```

Szolgáltatás alapú rendszerek megvalósítása

Eseménynaplózás

- Pl.:

```
<nlog ...>
```

```
  <targets>
```

```
    <target xsi:type="File" name="f"
```

```
      fileName="${basedir}/logs/${shortdate}.log"
```

```
      layout="${longdate} ${level} ${message}" />
```

```
    <!-- a naplózás a megadott fájlba történik a  
          megadott formátumban -->
```

```
  ...
```

```
<rules>
```

```
  <logger name="model" minlevel="Debug"
```

```
    writeTo="f" /> <!-- a modell naplója Debug  
          szintig írja a fenti fájlt -->
```

```
  ...
```

Szolgáltatás alapú rendszerek megvalósítása

Példa

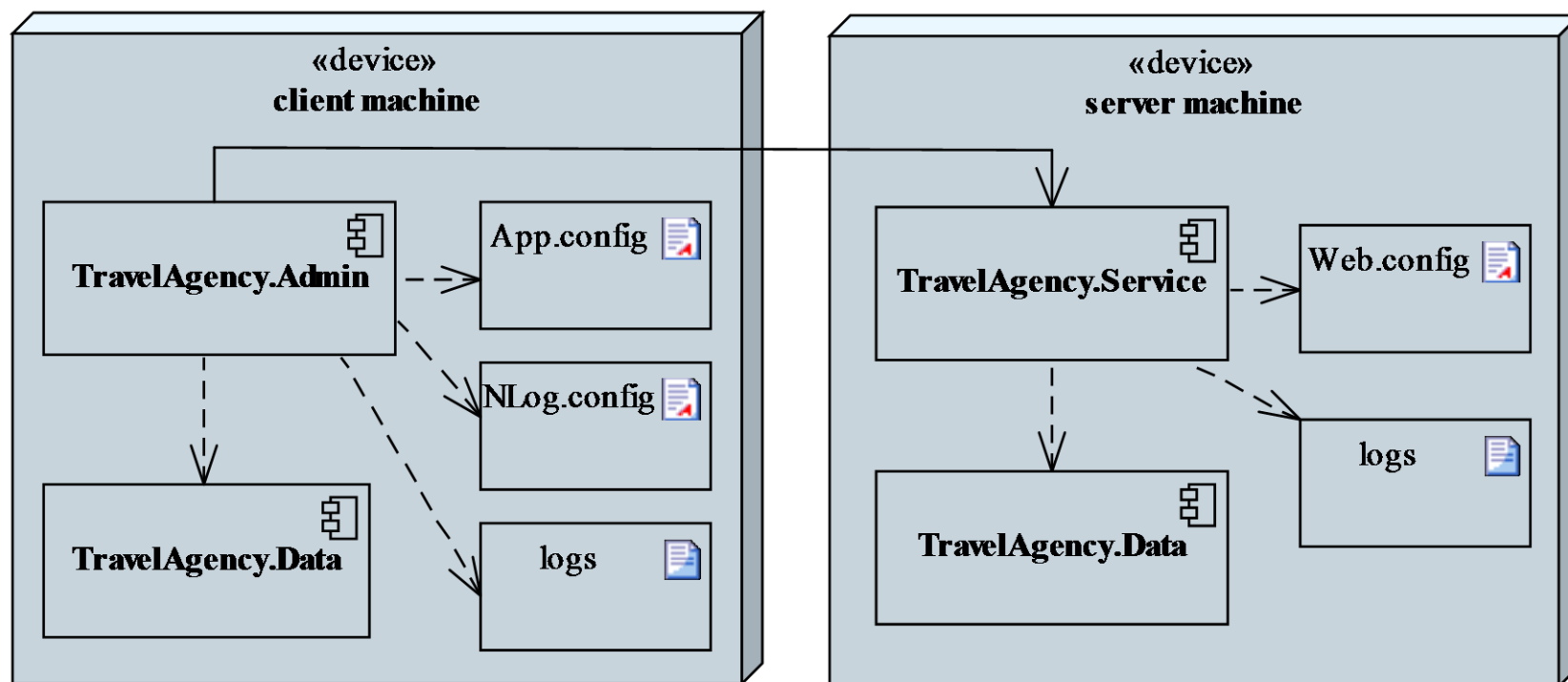
Feladat: Valósítsuk meg az utazási ügynökség épületeit karbantartó asztali alkalmazást.

- kliens oldalon kiemeljük a változtatható értékeket (szolgáltatás címe, képek méretezése) a konfigurációba
- kliens és szerver oldalon is bevezetünk eseménynaplózást (fájlba)
 - kliens oldalon a perzisztenciát naplózzuk, a végrehajtott kéréseket (**info**), az esetleges nem várt visszajelzéseket (**warning**), illetve a keletkezett kivételeket (**error**)
 - szolgáltatás oldalon a felhasználói funkciókat (pl. bejelentkezés), illetve szintén a kivételeket naplózzuk

Szolgáltatás alapú rendszerek megvalósítása

Példa

Tervezés (telepítés):



Szolgáltatás alapú rendszerek megvalósítása

Példa

Megvalósítás (App.xaml.cs):

...

```
String serviceAddress = ConfigurationManager.
```

```
AppSettings["ServiceAddress"];
```

```
// beállítás lekérdezése a konfigurációból
```

```
if (String.IsNullOrEmpty(serviceAddress)) { ... }
```

```
// ellenőrizzük, hogy sikerült-e beolvasni
```

```
_model = new TravelAgencyModel(
```

```
new TravelAgencyServicePersistence(
```

```
serviceAddress));
```

...

Szolgáltatás alapú rendszerek megvalósítása

Példa

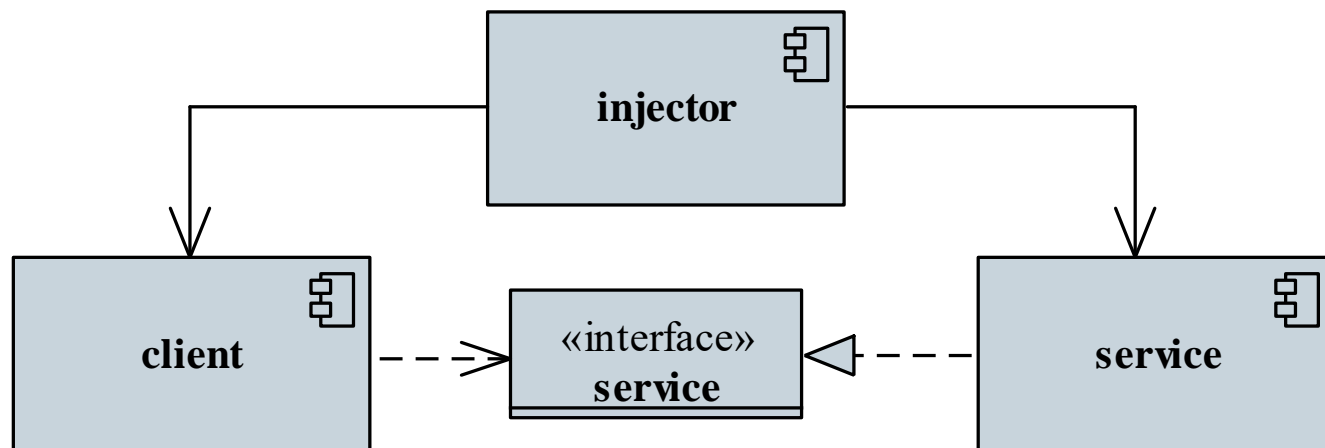
Megvalósítás (TravelAgencyServicePersistence.cs):

```
try {
    _log.Info("GET query on service " +
        _client.BaseAddress + ", path:
        api/buildings/");
    // információ kiírása az eseménynaplóba
    ...
}
catch (Exception ex) {
    _log.Error(ex, "GET query aborted with
        exception.");
    // hiba kiírása az eseménynaplóba a kivétel
    // tartalmával
}
```

Szolgáltatás alapú rendszerek megvalósítása

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



Szolgáltatás alapú rendszerek megvalósítása

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 komponens, amely lehetőséget ad szolgáltatások megvalósításának dinamikus (futási idejű) betöltésére
 - egy központi regisztráció, amelyet minden programkomponens elérhet, és felhasználhat

Szolgáltatás alapú rendszerek megvalósítása

IoC tároló

- a típusokat (elsősorban) interfész alapján azonosítja, és az interfészhez csatolja a megvalósító osztályt
- a tárolóba történő regisztrációkor (**Register**) megadjuk a szolgáltatás interfészét és megvalósításának típusát (vagy példányát)
- a szolgáltatást interfész alapján kérjük le (**Resolve**), ekkor példányosul a szolgáltatás
 - amennyiben a szolgáltatásnak függősége van, a tároló azt is példányosítja
- A *Unity* programcsomag egy általánosan használható IoC tárolót biztosít, amely lehetővé teszi a regisztráció konfigurációs fájlban történő elvégzését

Szolgáltatás alapú rendszerek megvalósítása

IoC tároló

- Pl. :

```
interface ICalculator // szolgáltatás interfésze
{
    Double Compute(Double value);
}
...
class LogCalculator : ICalculator
    // a szolgáltatás egy megvalósítása
{
    public Double Compute(Double value) {
        return Math.Log(value);
    }
}
```

Szolgáltatás alapú rendszerek megvalósítása

IoC tároló

```
interface IVisualization
{
    void PrintComputation();
}

class ConsoleVisualization : IVisualization {
    private ICalculator calculator; // függőség

    public ConsoleVisualization(ICalculator c) {
        calculator = c;
    } // konstruktor befecskendezés

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

Szolgáltatás alapú rendszerek megvalósítása

IoC tároló

- Pl. :

```
UnityContainer c = new UnityContainer();  
    // tároló példányosítása
```

```
c.RegisterType<ICalculator, LogCalculator>();  
c.RegisterType<IVisualization,  
    ConsoleVisualization>();  
    // szolgáltatások regisztrációja
```

```
IVisualization visualization =  
    c.Resolve<IVisualization>();  
    // szolgáltatás lekérése (példányosítással)  
    // egy ConsoleVisualization példányt, és benne  
    // egy LogCalculator példányt kapunk vissza
```

Szolgáltatás alapú rendszerek megvalósítása

IoC tároló megvalósítása

- A tároló beállítása elhelyezhető konfigurációs fájlban is a **unity** elemekben, pl.:

```
<configuration>
    ...
    <unity ...>
        <container>
            <register type="ICalculator"
                mapTo="LogCalculator" />
            ...
        </container>
    </unity>
</configuration>
```

- a konfiguráció a **LoadConfiguration()** művelettel tölthető be

Szolgáltatás alapú rendszerek megvalósítása

IoC tároló webes alkalmazásokban

- ASP.NET alkalmazások is támogatják IoC tárolók használatát függőségek kezelésére, és automatizálják a típusok feloldását
 - mivel számos típus példányosítását a rendszer felügyeli (pl. vezérlők), ezért a függőségek kezelése nem valósítható meg kódban történő átadással
 - a függőség kezelését az **IDependencyResolver** interfészt megvalósító típus biztosítja, amelynek példányát a **HttpConfiguration** típus **DependencyResolver** tulajdonságának kell átadnunk
 - elsőként az interfészt kell megvalósítanunk egy IoC tároló használatával
 - a **Register** műveletben beállíthatjuk a függőségeket

Szolgáltatás alapú rendszerek megvalósítása

IoC tároló webes alkalmazásokban

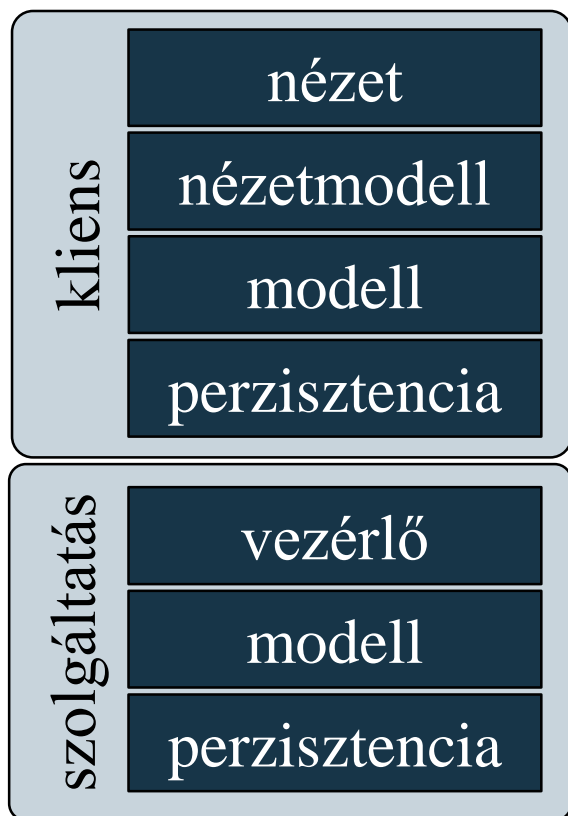
- Pl. :

```
class UnityResolver : IDependencyResolver { ... }  
    // függőségkezelő megvalósítása Unity tárolóval  
...  
public static void Register(... config)  
{  
    ...  
    UnityContainer container =  
        new UnityContainer();  
    ...  
    config.DependencyResolver =  
        new UnityResolver(container);  
    // a típusok feloldása már automatikus  
}
```

Szolgáltatás alapú rendszerek megvalósítása

Szolgáltatások tesztelése

- A szolgáltatás alapú rendszerek összetett struktúrájuknak köszönhetően számos komponensből, rétegből épülnek fel



- a felépítésből adódóan az egységtesztek mellett nagy hangsúlyt kap az *integrációs* és *rendszer*tesztek
 - több komponens együttes viselkedését ellenőrizzük (pl. modell-nézetmodell, modell-perzisztencia, perszisztencia-vezérlő)
 - ugyanakkor kizárjuk a külső tényezőket (pl. adatbázis, hálózat)

Szolgáltatás alapú rendszerek megvalósítása

Szolgáltatások tesztelése

- A szolgáltatás tesztelését célszerű felügyelt környezetben, a teszten belül elvégezni
 - mivel a szolgáltatás webszervert igényel, a Web API biztosít egy könnyűsúlyú webszervert (**HttpServer**), amely lehetővé teszi a szolgáltatás futtatását közvetlenül a memóriában, hálózati kapcsolat igénybevétele nélkül
 - a webszerver automatikusan csatlakoztatja a szolgáltatást (és vezérlőt), amennyiben hivatkozva van a projektben, csak a konfigurációt (**HttpConfiguration**) kell átadnunk
 - a kliens (**HttpClient**) példányosításakor átadhatjuk a szerveret, így minden kliensbeli kérés a memóriában hajtódik végre

Szolgáltatás alapú rendszerek megvalósítása

Szolgáltatások tesztelése

- Pl. :

```
HttpConfiguration config = ...;
WebApiConfig.Register(config);
    // betölthetjük közvetlenül a szolgáltatás
    // konfigurációját
HttpServer server = new HttpServer(config);
    // memóriabeli szerver létrehozása, a vezérlők
    // automatikusan betöltődnek
HttpClient client = new HttpClient(server);
    // kliens csatlakoztatása a szerverhez

... client.GetAsync(http://server/api/products);
    // a kérés a memóriában fut le
```

Szolgáltatás alapú rendszerek megvalósítása

Példa

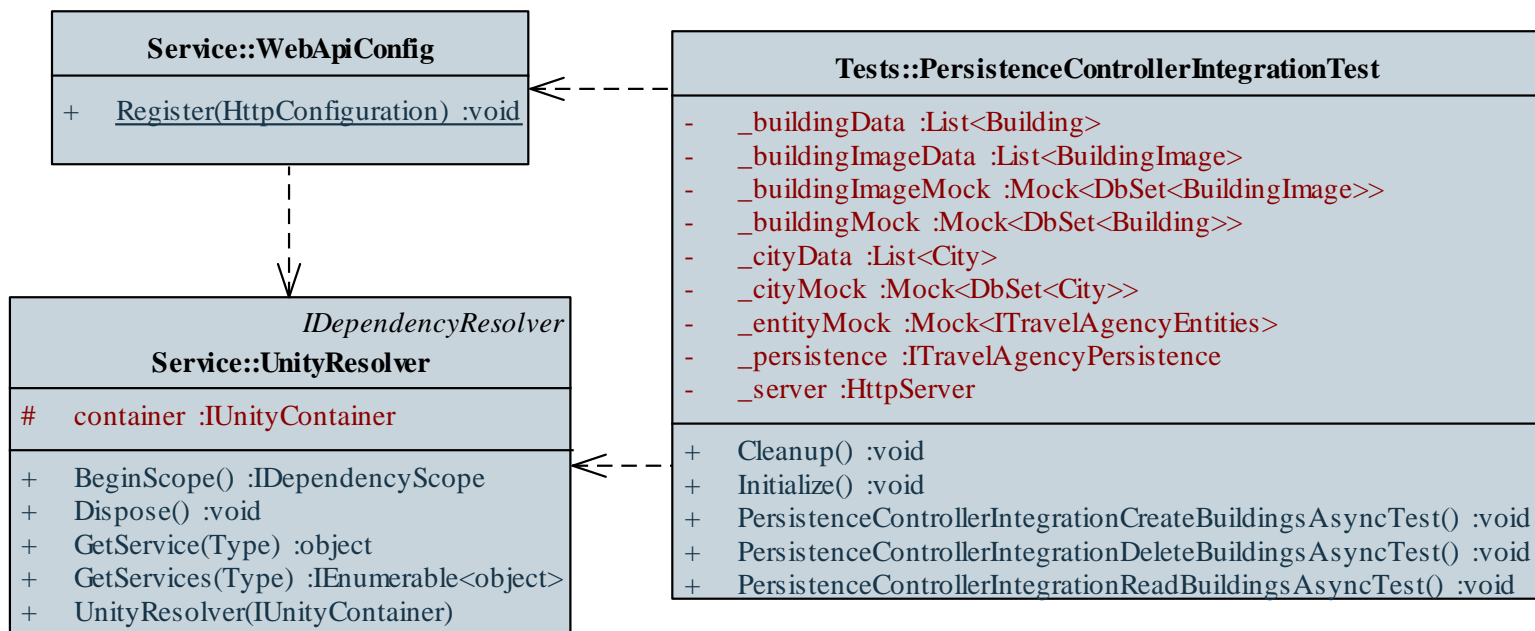
Feladat: Valósítsuk meg az utazási ügynökség épületeit karbantartó asztali alkalmazást.

- valósítsuk meg a függőségek (modell, perzisztencia) kezelését Unity tárolóval mindkét oldalon
 - a kliens esetén a konfigurációba helyezzük a felépítést, és az alkalmazás (**App**) tartalmazza a tárolót
 - a szerver esetén a konfigurációs művelet (**Register**) kezeli a tárolót, amit kódban állítunk össze
- készítsünk integrációs tesztet, amely a perzisztencia és a szolgáltatás (vezérlők) együttes működését ellenőrzi (**PersistenceControllerIntegrationTest**)
 - az entitásmodell tartalmát természetesen szimuláljuk

Szolgáltatás alapú rendszerek megvalósítása

Példa

Tervezés (tesztelés):



Szolgáltatás alapú rendszerek megvalósítása

Példa

Megvalósítás (WebApiConfig.cs):

...

```
// IoC tároló az adatbázis kezeléséhez
```

```
UnityContainer container = new UnityContainer();
```

```
container.RegisterType<ITravelAgencyEntities,  
                        TravelAgencyEntities>();
```

```
// az IoC tárolónkat fogja a rendszer használni a  
// függőségek feloldására
```

```
config.DependencyResolver =  
    new UnityResolver(container);
```

...

Szolgáltatás alapú rendszerek megvalósítása

Példa

Megvalósítás (`PersistenceControllerIntegrationTest.cs`):

...

```
// webszolgáltatás inicializációja
```

```
HttpConfiguration config =
```

```
    new HttpConfiguration();
```

```
WebApiConfig.Register(config);
```

```
// IoC tároló az adatbázis kezeléséhez
```

```
UnityContainer container = new UnityContainer();
```

```
container.RegisterInstance<ITravelAgencyEntities>(
    _entityMock.Object);
```

```
// itt egy példányt regisztrálunk be
```

Szolgáltatás alapú rendszerek megvalósítása

Példa

Megvalósítás (`PersistenceControllerIntegrationTest.cs`):

```
config.DependencyResolver =  
    new UnityResolver(container);  
  
_server = new HttpServer(config);  
    // memóriában futó HTTP szerver  
  
_persistence =  
    new TravelAgencyServicePersistence(  
        "http://server", _server);  
    // ehhez csatlakozik a kliens  
...
```