



**Eötvös Loránd Tudományegyetem  
Informatikai Kar**

## **Webes alkalmazások fejlesztése**

---

### **9. előadás**

## **Webszolgáltatások felhasználása (ASP.NET WebAPI)**

---

**Cserép Máté**  
**[mcserep@inf.elte.hu](mailto:mcserep@inf.elte.hu)**  
**<http://mcserep.web.elte.hu>**

Készült Giachetta Roberto jegyzete alapján  
<http://www.inf.elte.hu/karunkrol/digitkonyv/>

# Webszolgáltatások felhasználása

## A webszolgáltatás

---

- A webszolgáltatások lehetővé teszik az alkalmazások közötti platformfüggetlen adatcserét
  - a legelterjedtebb modell a *REST (Representational State Transfer)*, amely HTTP protokoll segítségével biztosítja a kommunikációt
  - a szolgáltató megvalósítható *ASP.NET WebAPI* alapon, a kliens tetszőleges alkalmazás lehet
  - a műveletek nem csak primitív típusokat, de összetett, *adatátviteli objektumokat (Data Transfer Object, DTO)* is közölhetnek
  - az objektumelvű adatok továbbítására legelterjedtebb a *JSON (Javascript Object Notation)* formátum

# Webszolgáltatások felhasználása

## A kliens

---

- Az ASP.NET alapú webszolgáltatásokhoz fogyasztóként az *ASP.NET WebAPI Client* csomag segítségével férhetünk hozzá
- A `HttpClient` típus biztosítja a kapcsolatot HTTP alapú szolgáltatásokhoz
  - mivel a hálózati kommunikáció időigényes, aszinkron függvények segítségével biztosítja a HTTP utasítások futtatását (`GetAsync`, `PutAsync`, ...)
  - az utasítások eredménye tartalmazza a választ (`HttpResponseMessage`)
  - amennyiben a művelet sikeres (`IsSuccessStatusCode`), akkor feldolgozhatjuk a tartalmat (`Content`)

# Webszolgáltatások felhasználása

## A kliens

---

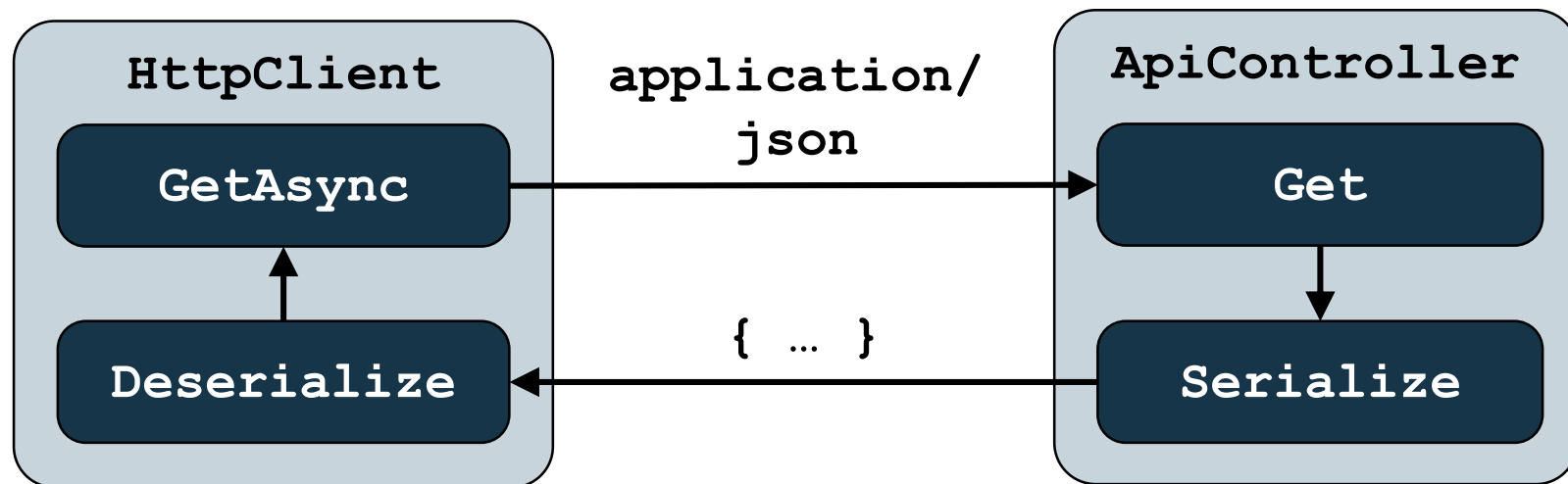
- Pl.:

```
using (HttpClient client = new HttpClient())
    // kliens példányosítása
{
    HttpResponseMessage response =
        await client.GetAsync("http://...");
    // kérés aszinkron végrehajtása
    if (response.IsSuccessStatusCode)
    {
        HttpContent content = response.Content;
        ... // tartalom feldolgozása
    }
} // kliens megsemmisítése
```

# Webszolgáltatások felhasználása

## A kliens adatkezelése

- A tartalom kezelése objektumorientált módon történik, az adattovábbítás formátumát a rendszer kezeli (ahogy a szolgáltató esetén is)
  - az adatátviteli objektum átalakításának (*szerializáció*) és visszaállításának (*deszerializáció*) módja a HTTP fejlécinformációk szerint történik



# Webszolgáltatások felhasználása

## A kliens adatkezelése

---

- ehhez szükséges, hogy az adatátviteli objektum típusa a kliens és szerver oldalon is megegyezzen (pl. egyazon osztálykönyvtárból vannak meghivatkozva)
- olvasás esetén csak az adat típusát kell megadnunk, pl.:

```
if (response.IsSuccessStatusCode) {  
    Product myProduct = await  
        response.Content.ReadAsAsync<Product>();  
    ... // tartalom feldolgozása  
}
```
- ugyanakkor lehetőségünk van a tartalmat szöveges (`ReadAsStringAsync`), vagy bináris formátumban (`ReadAsByteArrayAsync`), illetve adatfolyamként (`ReadAsStreamAsync`) kezelni

# Webszolgáltatások felhasználása

## A kliens adatkezelése

---

- a szerializációt végző típus (**MediaTypeFormatter**) specializálható, így egyedi üzenetformátumok is kialakíthatóak
- amennyiben tartalmat is küldünk (pl. **POST**, **DELETE**),
  - megadhatjuk annak formátumát, pl.:

```
client.PostAsync("http://...", product,  
    new JsonMediaTypeFormatter());  
// az adatot JSON formátumra szerializálja
```
  - vagy közvetlenül a megfelelő formátumú adatküldést is hívhatjuk, pl.:

```
client.PostAsJsonAsync("http://...", product);  
// az adatot JSON formátumra szerializálja
```

# Webszolgáltatások felhasználása

## A kliens konfigurációja

---

- A kliensben konfigurálható
  - a címek előtagja (**BaseAddress**),
  - a kommunikáció időkorlátja (**Timeout**),
  - az elküldött üzenetek fejlécének tulajdonságai (**DefaultRequestHeaders**), és azon belül
  - a tartalom formátuma (**DefaultRequestHeaders.Accept**),  
pl.:

```
client.DefaultRequestHeaders.Accept.Clear();
client.DefaultRequestHeaders.Accept.Add(
    new MediaTypeWithQualityHeaderValue(
        "application/json"));
// a kliens csak a JSON formátumot fogadja el
```



# Webszolgáltatások felhasználása

## Alapvető adatkezelési műveletek

---

- A szolgáltatás sok esetben alapvető adatkezelési műveleteket biztosít, ezek a *CRUD* műveletek
  - létrehozás (*Create*), olvasás (*Read*), módosítás (*Update*), törlés (*Delete*)
  - a műveleteknek adott a HTTP megfelelője (létrehozás: **POST**, olvasás: **GET**, módosítás: **PUT**, törlés: **DELETE**)
    - a válasz kódja létrehozás esetén **CREATED** (201), többi művelet esetén **OK** (200), vagy **NO CONTENT** (204)
    - amennyiben a művelet nem azonnal hajtódik végre, **ACCEPTED** (202) állapotot jelezhetünk
  - egy RESTful szolgáltatásban a műveleteknek ehhez a sémához kell alkalmazkodnia

# Webszolgáltatások felhasználása

## Kliens oldali adatkezelés

---

- A kliens oldali adatkezelést kétféleképpen valósíthatjuk meg:
  - *szinkron módon*: a kliens és a szerver állapota mindig megegyezik
  - *aszinkron módon*: a kliens és a szerver állapota eltér, és manuálisan szinkronizálható (mentés, betöltés, frissítés)
- Az aszinkron adatkezelés előnyös, ha a változtatásainkat nem egyenként, hanem csoportosan szeretnénk elmenteni
  - ehhez kliens oldalon követnünk kell a változásokat *állapotjelzőkkel (flag)*, és megjelölnünk, milyen változtatásokat történtek az adatokon (új, módosított, törölt)

# Webszolgáltatások felhasználása

## Példa

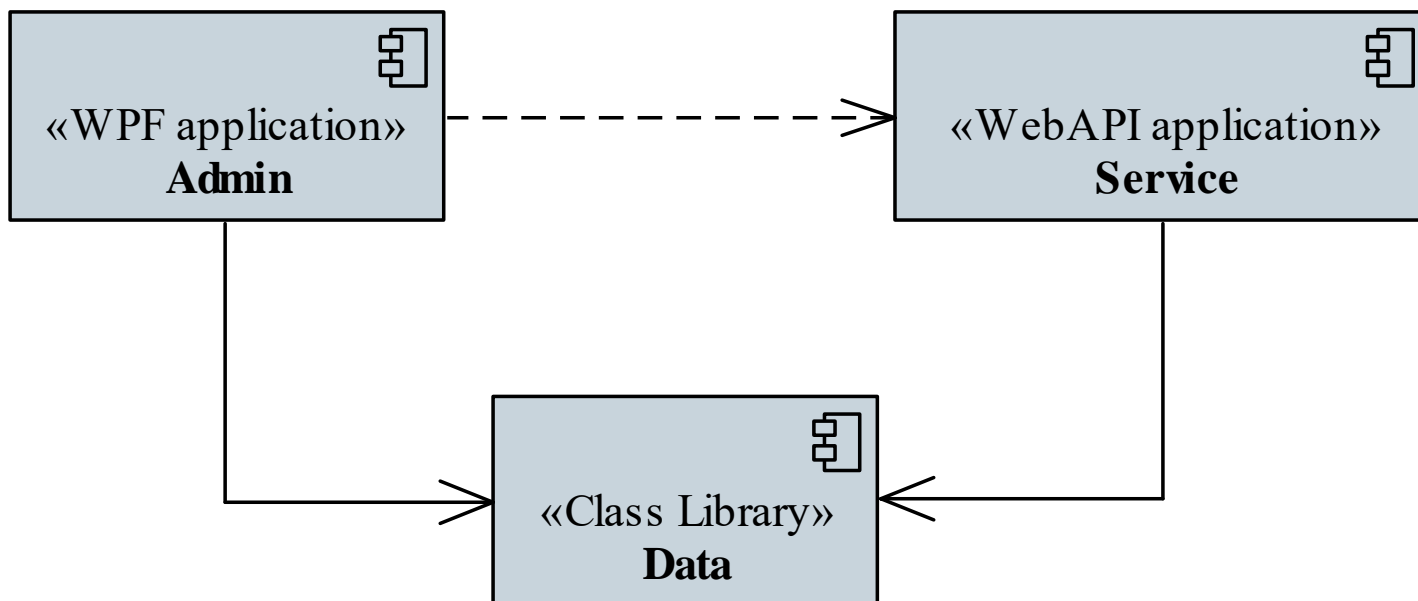
*Feladat:* Valósítsuk meg az utazási ügynökség épületeit karbantartó asztali alkalmazást.

- a kliens egy WPF alkalmazás lesz (**TravelAgency.Admin**), amely egy WebAPI szolgáltatáshoz (**TravelAgency.Service**) fog csatlakozni
- a kliens alkalmazást MVVM architektúrában készítjük el, ahol a perzisztencia (**TravelAgencyServicePersistence**) biztosítja a hálózati kommunikációt
- az adatátvitelhez külön típust hozunk létre (**BuildingDTO**), és egy külön osztálykönyvtárba helyezzük el (**TravelAgency.Data**), amely megosztásra kerül mindkét projekt számára

# Webszolgáltatások felhasználása

## Példa

*Tervezés (architektúra):*

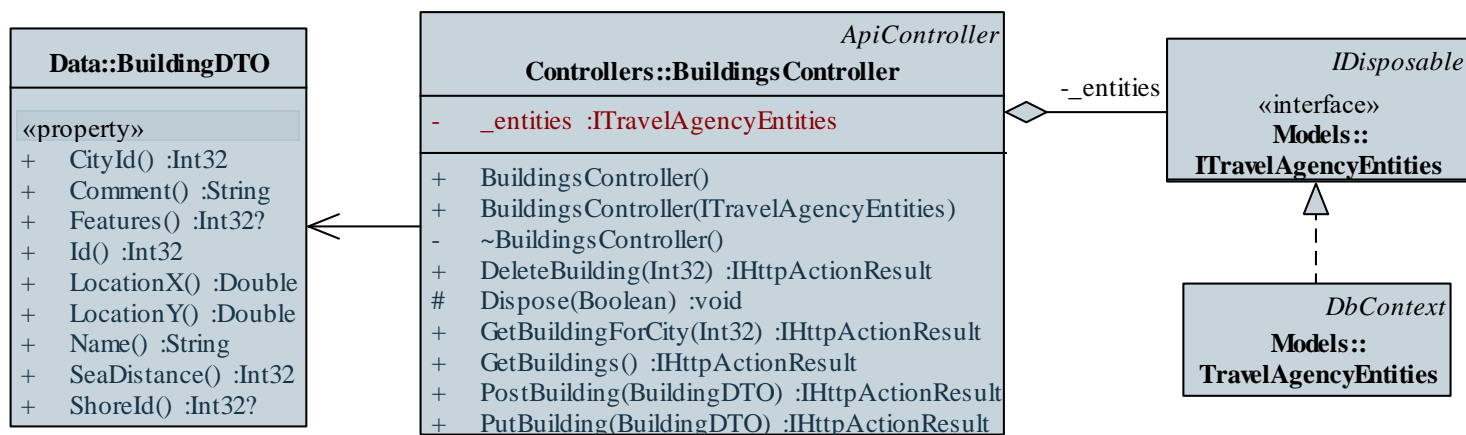


# Webszolgáltatások felhasználása

## Példa

*Tervezés (szolgáltatás):*

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



# Webszolgáltatások felhasználása

## Példa

---

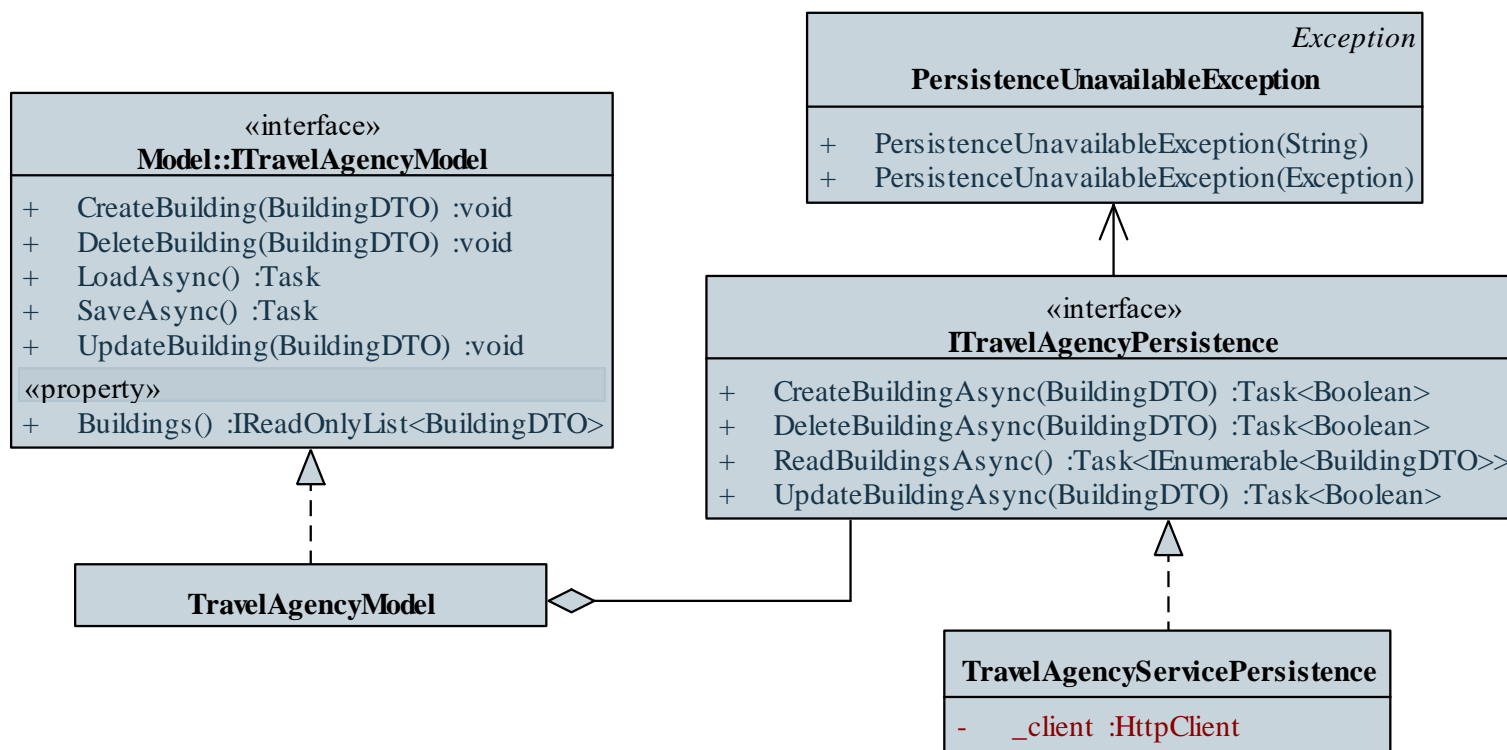
*Tervezés (kliens):*

- a kliens aszinkron adatkezelést biztosít, a modell (**ITravelAgencyModel**) felügyeli a kliensbeli állapotot állapotjelzőkkel (**DataFlag**), ez alapján tudjuk mentéskor a megfelelő műveletet elvégezni
- a perzisztencia (**ITravelAgencyPersistence**) feladata az adatok betöltése, mentése és konvertálása aszinkron műveletekkel
- az új épületeknek létrehozunk egy ideiglenes azonosítót (a megkülönböztetés végett), amely helyett a szerver visszaad egy végleges azonosítót

# Webszolgáltatások felhasználása

## Példa

Tervezés (kliens):



# Webszolgáltatások felhasználása

## Példa

*Megvalósítás (TravelAgencyModel.cs):*

```
public async Task SaveAsync() {  
    ...  
    // az állapotjelzőnek megfelelő műveletet  
    // végezzük el  
    switch (_buildingFlags[building])  
    {  
        case DataFlag.Create:  
            result = await _persistence  
                .CreateBuildingAsync(building);  
            break;  
        case DataFlag.Delete:  
            ...  
    }  
}
```



# Webszolgáltatások felhasználása

## Példa

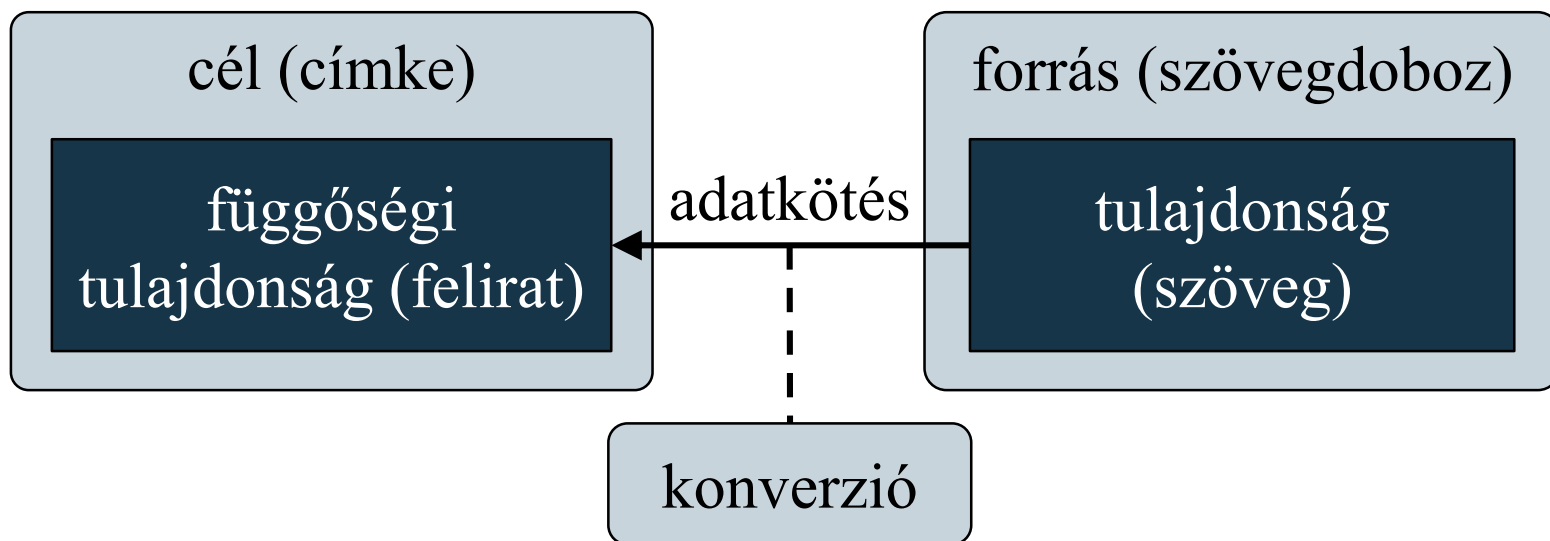
*Megvalósítás (TravelAgencyPersistence.cs):*

```
public async Task<Boolean> CreateBuildingAsync (...) {  
    ...  
    HttpResponseMessage response = await  
        _client.PostAsJsonAsync("api/buildings/",  
                                building);  
    // az értékeket azonnal JSON formátumra  
    // alakítjuk  
    building.Id = (await response.Content  
        .ReadAsAsync<BuildingDTO>()).Id;  
    // a válaszüzenetben megkapjuk a végleges  
    // azonosítót  
    ...  
}
```

# Webszolgáltatások felhasználása

## Adatkonverzió

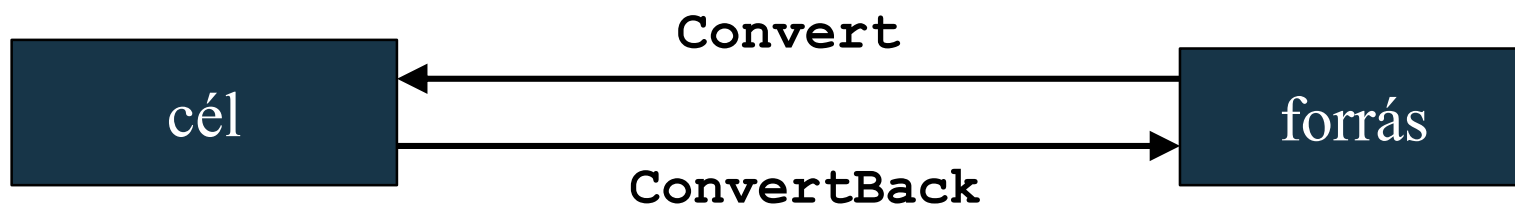
- Az adatkötés során lehetőségünk van átalakítani (konvertálni) az adatot a megjelenítés és a kötött tartalom között
  - vannak alapértelmezett átalakítások (pl. szöveg/szám)
  - alkalmazhatunk egyedi konverziót (az `IValueConverter` interfész megvalósításával)



# Webszolgáltatások felhasználása

## Adatkonverzió

- Az `IValueConverter` interfész biztosítja a `Convert` és `ConvertBack` műveleteket, amelyek elvégzik a transzformációt
  - az átalakítás paraméterezhető (`ConverterParameter`)
  - figyelembe veszi a nyelvi környezetet (`ConverterCulture`)



- A konverziót a kötésnél adjuk meg, általában erőforrásból betöltve:  
`{Binding Path=..., Converter=...,  
ConverterParameter=..., ConverterCulture=...}`

# Webszolgáltatások felhasználása

## Adatkonverzió

---

- Pl.:

```
class StringToIntConverter : IValueConverter
    // egyszerű szám-szöveg átalakító
{
    public object Convert(object value, ...) {
        return value.ToString();
    } // átalakítás szöveggé

    public object ConvertBack(object value, ...) {
        return Convert.ToInt32(value);
    }
    // átalakítás számmá
}
```

# Webszolgáltatások felhasználása

## Adatkonverzió

---

- Pl.:

```
<Window ... xmlns:local="clr-namespace:MyApp">
  <Window.Resources>
    <local:StringToIntConverter
      x:Key="converter" />
    <!-- az átalakító, mint erőforrás -->
  </Window.Resources>
  ...
  <TextBox Text="{Binding Path=...,
    Converter={StaticResource converter}
  }" />
  <!-- szövegdoboz, amely az adatkötéshez
    felhasználja az átalakítót -->
  ...
```

# Webszolgáltatások felhasználása

## Adatkonverzió hibakezelése

---

- Az átalakítás során hiba léphet fel (pl. a megadott szöveg nem konvertálható számmá), amelyet megfelelően kell kezelnünk
  - a konvertáláskor nem keletkezhetsz kivétel
  - amennyiben a cél értéket nem tudjuk létrehozni (a **Convert** műveletben), akkor jelezzük, hogy nem kell végrehajtani a kötetést (**Binding.DoNothing**)
  - amennyiben a forrás tulajdonságot nem tudjuk beállítani (a **ConvertBack** műveletben), akkor visszaadjuk a beállítatlan függőségi értéket (**DependencyProperty.UnsetValue**)
  - a beállítási hiba azonnal jelentkezik a felületen is (alapértelmezetten piros keretben)

# Webszolgáltatások felhasználása

## Adatkonverzió hibakezelése

---

- Pl.:

```
class StringToIntConverter : IValueConverter
    // egyszerű szám-szöveg átalakító
{
    ...
    public object ConvertBack(object value, ...) {
        try {
            return Convert.ToInt32(value);
        } catch { // elfogjuk a kivételt
            return DependencyProperty.UnsetValue;
        } // jelezzük a sikertelen beállítást
    } // átalakítás számmá
}
```

# Webszolgáltatások felhasználása

## Ellenőrzések adatkonverzióval

---

- A hibakezeléssel egybekötött átalakító használható ellenőrzések végrehajtására is
  - nem is szükséges konvertálnia a tartalmat, csupán ellenőrzi az adat meglétét, formátumát

Pl.:

```
class EmailCheckConverter : IValueConverter
    // e-mail formátum ellenőrző átalakító
{
    public object Convert (object value, ...) {
        return value;
    } // nem végzünk semmilyen átalakítást
```



# Webszolgáltatások felhasználása

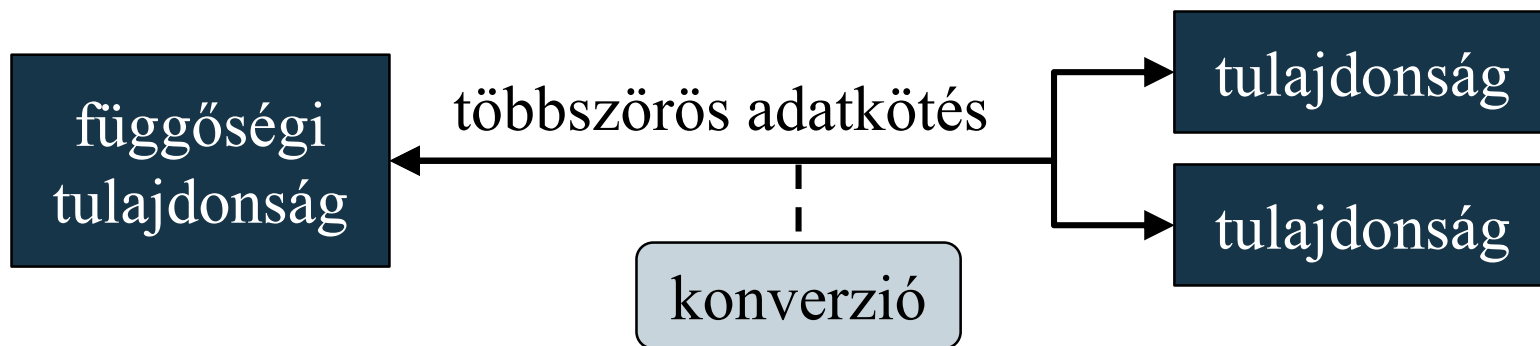
## Ellenőrzések adatkonverzióval

```
public object ConvertBack(object value, ...) {
    if (value == null ||
        !Regex.IsMatch(value.ToString(),
            @"^([0-9a-zA-Z]([-.\w]*[0-9a-zA-Z])*\@[0-9a-zA-Z]([-.\w]*[0-9a-zA-Z])\.)+[a-zA-Z]{2,9})$")) {
        // ha nem egyezik az e-mail formátum
        // reguláris kifejezésével
        return DependencyProperty.UnsetValue;
        // akkor jelezzük a hibát
    }
    return value;
    // különben nem csinálunk semmit
}
```

# Webszolgáltatások felhasználása

## Többszörös kötés és konverzió

- Lehetőségünk van egy függőségi tulajdonságra több tulajdonságot is kötni (**MultiBinding**)
  - több egyszerű kötés (**Binding**) gyűjteménye
  - csak megfelelő konverzióval (**IMultiValueConverter**) jeleníthetők meg az adatok
    - tömbként fogadja (a kötés sorrendjében) az adatokat, és ugyanebben a sorrendben kell visszaadnia



# Webszolgáltatások felhasználása

## Többszörös kötés és konverzió

---

- Pl.:

```
<TextBlock>
```

```
  <TextBlock.Text>
```

```
    <!-- szöveg összetett megadása -->
```

```
    <MultiBinding Converter="...">
```

```
      <!-- többszörös kötés átalakítóval -->
```

```
      <Binding Path="..." />
```

```
      <Binding Path="..." />
```

```
      <!-- tetszőleges sok kötést adunk meg -->
```

```
    </MultiBinding>
```

```
  </TextBlock.Text>
```

```
</TextBlock>
```

# Webszolgáltatások felhasználása

## Többszörös kötés és konverzió

---

- Pl.:

```
class MyMultiConverter : IMultiValueConverter {
    public object Convert(object[] values, ...) {
        // egy tömbben kapjuk meg az értékeket, a
        // megadott kötések sorrendjében
        ...
    }
    public object[] ConvertBack(object value, ...) {
        // egy tömbben szolgáltatjuk vissza az
        // eredményt, ismét a megadott sorrendben
        ...
    }
}
```

# Webszolgáltatások felhasználása

## Példa

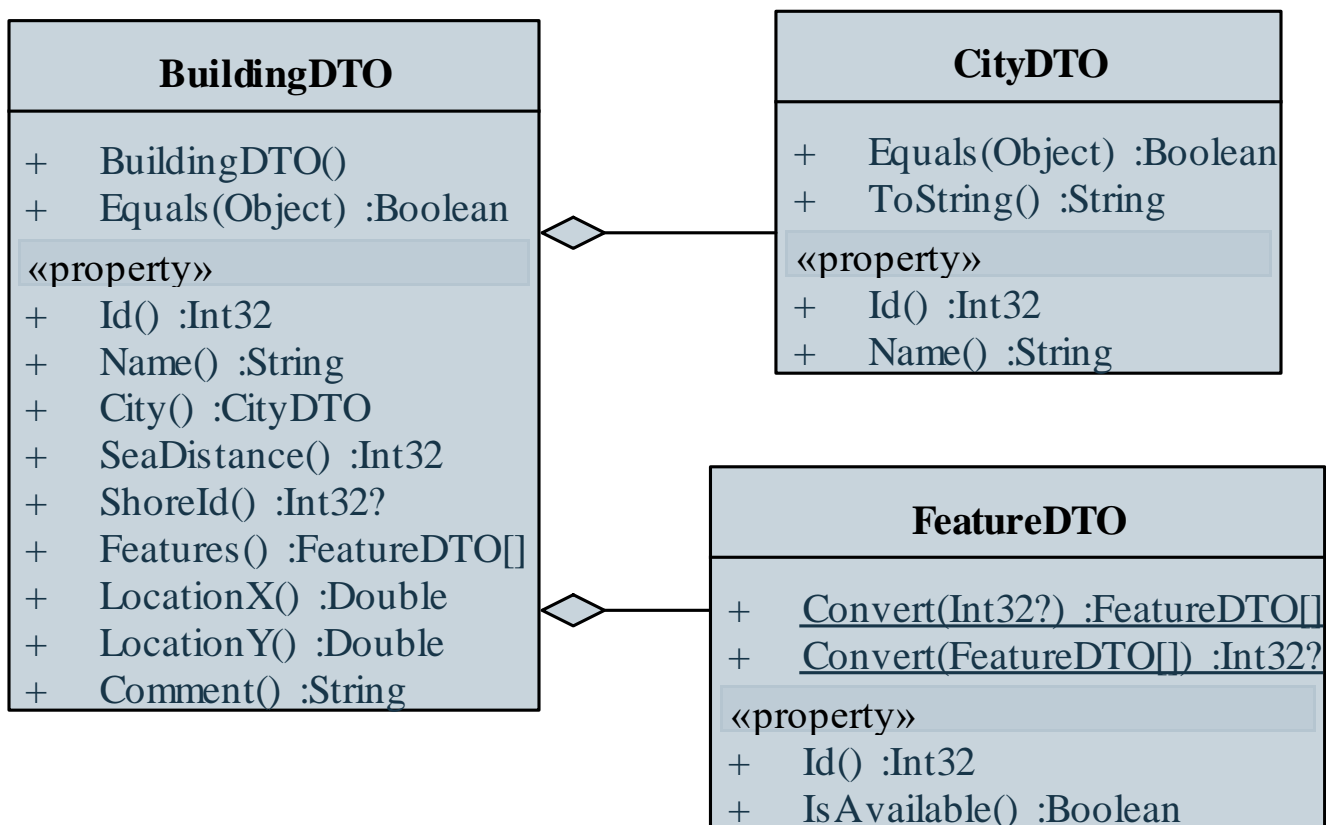
*Feladat:* Valósítsuk meg az utazási ügynökség épületeit karbantartó asztali alkalmazást.

- a jobb megjelenítés érdekében használjunk adatkonverziót a kliens oldalán, szükség lesz:
  - a tengerpart távolság átalakítására (**SeaDistanceConverter**)
  - a tengerpart típus átalakítására (**ShorteTypeConverter**)
  - a jellemzők átalakítására (**FeatureConverter**, **FeatureDisplayConverter**)
- módosítjuk az adatátviteli típust (**BuildingDTO**) is, hogy az kifejezőbb legyen a megjelenítés számára, és felvesszünk két további segédtypust (**CityDTO**, **FeatureDTO**)

# Webszolgáltatások felhasználása

## Példa

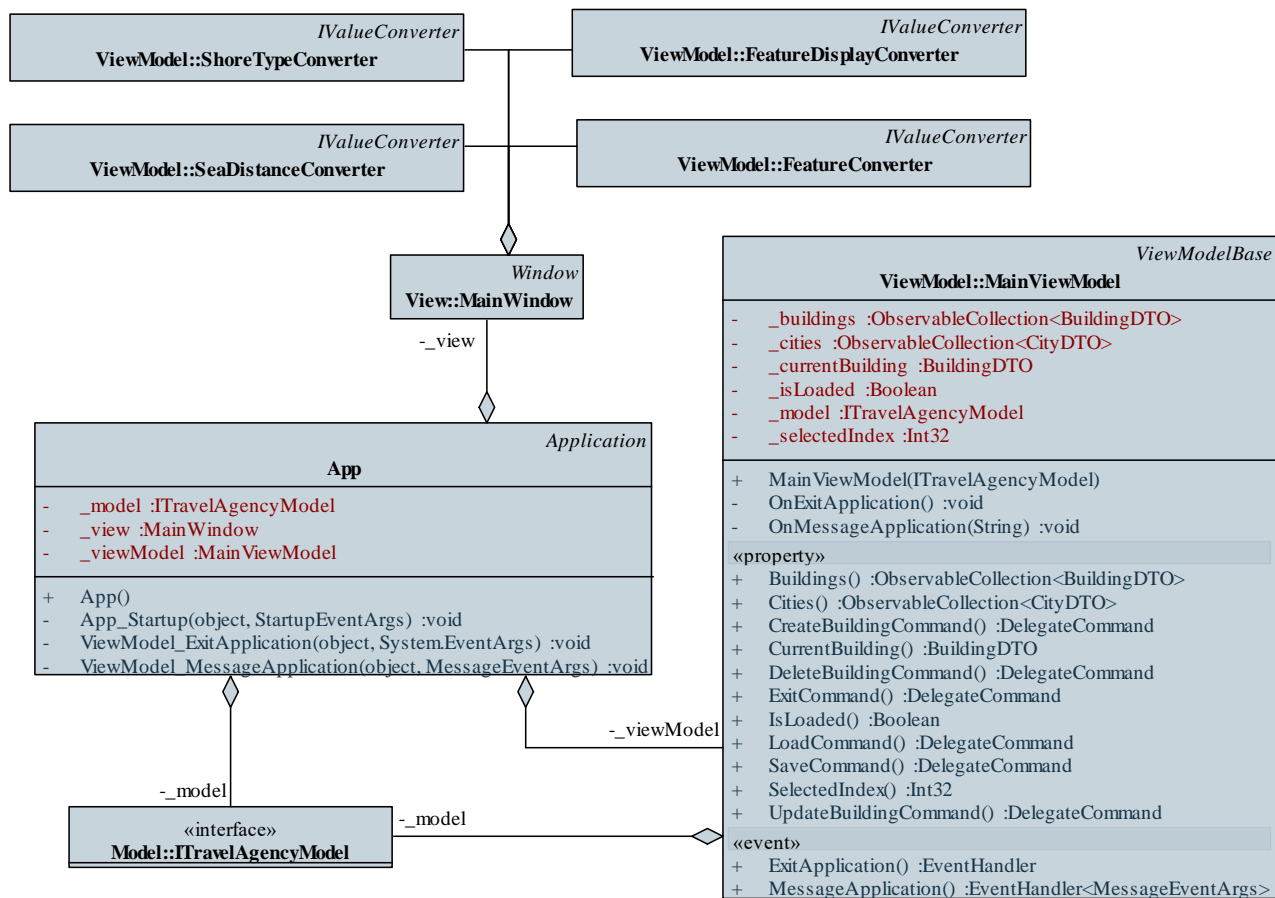
*Tervezés (adatátvitel):*



# Webszolgáltatások felhasználása

## Példa

Tervezés (kliens):



# Webszolgáltatások felhasználása

## Példa

---

*Megvalósítás (ShoreTypeConverter.cs):*

```
public Object Convert(Object value, ...) {
    ... // ellenőrizzük az értéket
    ... // ellenőrizzük a paramétert

    List<String> shoreNames =
        (parameter as IEnumerable<String>).ToList();
    Int32 index = (Int32)value;

    if (index < 0 || index >= shoreNames.Count)
        return Binding.DoNothing;

    return shoreNames[index];
}
```



# Webszolgáltatások felhasználása

## Példa

*Megvalósítás (MainWindow.xaml):*

...

```
<x:Array ... x:Key="shoreTypeArray">  
    <system:String>homokos</system:String>...
```

```
</x:Array>
```

```
<viewModel:ShoreTypeConverter  
    x:Key="shoreTypeConverter" />
```

...

```
<DataGridTextColumn Header="Tengerpart"  
    Binding="{Binding ShoreId,  
    Converter={StaticResource shoreTypeConverter},  
    ConverterParameter={StaticResource  
        shoreTypeArray}}" />
```

...

# Webszolgáltatások felhasználása

## Adatok titkosítása memóriában

---

- A teljes biztonsághoz a memóriában lévő kényes tartalmat is titkosítani kell, mivel az is potenciális támadási felület
  - lehetőség szerint csak a feldolgozás időtartamára szerepeljen titkosítatlan információ a memóriában, egyébként kódolva tároljuk
- Szövegek titkosított kezelésére szolgál a **SecureString** típus, amely alapvetően titkosítva tárolja a jelszót, és csak lekéréskor (**ToString**) dekódolja
  - a grafikus felületen a jelszavak titkosított bekérését a **PasswordBox** biztosítja, a **Password** tulajdonság feloldja a titkosítást (amely nem függőségi tulajdonság, így nem köthető)

# Webszolgáltatások felhasználása

## Adatok titkosítása memóriában

---

- Pl.:

```
<Button Content="Bejelentkezés"  
        Command="{Binding LoginCommand}"  
        CommandParameter="{Binding  
                            ElementName=passwordBox}" .../>
```

```
<!-- magát a jelszóbekérőt adjuk át -->
```

...

```
LoginCommand = new DelegateCommand(param =>  
{  
    _model.Login(UserName,  
                  (param as PasswordBox).Password);  
    // kiolvassuk a titkosított jelszót  
    ...  
}
```

# Webszolgáltatások felhasználása

## Adatok betöltése az adatbázisba

---

- A szolgáltatás adatbázis struktúrája és adatai létrehozhatónak kód segítségével (*Entity Framework Code First*)
  - adatok betöltéséhez engedélyeznünk kell adatmigrációt a NuGet csomagkezelő konzolban (**Enable-Migrations**), ezzel automatikusan létrejön egy betöltő függvény (**Configuration.Seed**), amely a paraméterben megadott adatbázisba (**context**) feltölti az adatokat
  - az ASP.NET Identity (EF) adatbázis is kód alapú, így a felhasználók és csoportok létrehozhatóak ezzel a módszerrel
    - mivel a migrációs eszköz osztályokra keres, mindenképpen szükséges az **IdentityContext** származtatása

# Webszolgáltatások felhasználása

## Adatok betöltése az adatbázisba

---

- Pl.:

```
internal sealed class Configuration : ... {
    protected override
        void Seed(MyDbContext context)
    {
        // adatbázis feltöltése adatokkal
        context.Customer.Add(new Customer
        {
            Name = "John Doe" ...
        });
        ...
        context.SaveChanges();
    }
}
```

# Webszolgáltatások felhasználása

## Adatbázis migrációja

---

- A csomagkezelő konzol **Add-Migration** utasításával további adatbázis migrációt leíró osztályok generálhatóak (**DbMigration** leszármaztatásával), amelyek **Up** és **Down** metódusaikkal adják meg a migrációt *CodeFirst* módon.
- új migrációk konzolon az **Update-Database** paranccsal vagy automatikusan alkalmazhatóak (**Configuration.AutomaticMigrationsEnabled**).
- Az inicializáló eljárás (**Configuration.Seed**) minden adatbázis frissítéskor végrehajtásra kerül.

# Webszolgáltatások felhasználása

## Példa

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

- adjunk lehetőséget képek megtekintésére, hozzáadására, törlésére
  - a képet fájlból töltjük be, majd átméretezzük (kis és nagy méretben, PNG formátumban)
  - a képeket egyedi azonosítóval látjuk el, valamint az épület azonosítójával
- a biztonság növelésére az adatkezelést autentikációhoz kötjük (ASP.NET Identity segítségével), így a felhasználónak előbb be kell jelentkezniük az alkalmazásba

# Webszolgáltatások felhasználása

## Példa

---

### *Tervezés:*

- létrehozunk egy vezérlőt (**BuildingImageController**), valamint egy adatátviteli típust (**ImageDTO**) a képkezeléshez
- a képeket alapvetően byte tömbként kezeljük, a szolgáltatás nem is ismeri azok képi tartalmát
- a képbetöltést egy segédtípusban (**ImageHandler**) végezzük
- a képek megjelenítéséhez átalakítást végzünk (**BuildingImageConverter**), ami **BitmapImage** típusra alakítja a tömböt, ezeket Image vezérlővel jelenítjük meg
- külön nézetbe szervezzük az épület adatainak megadását (**BuldingEditorWindow**)



# Webszolgáltatások felhasználása

## Példa

---

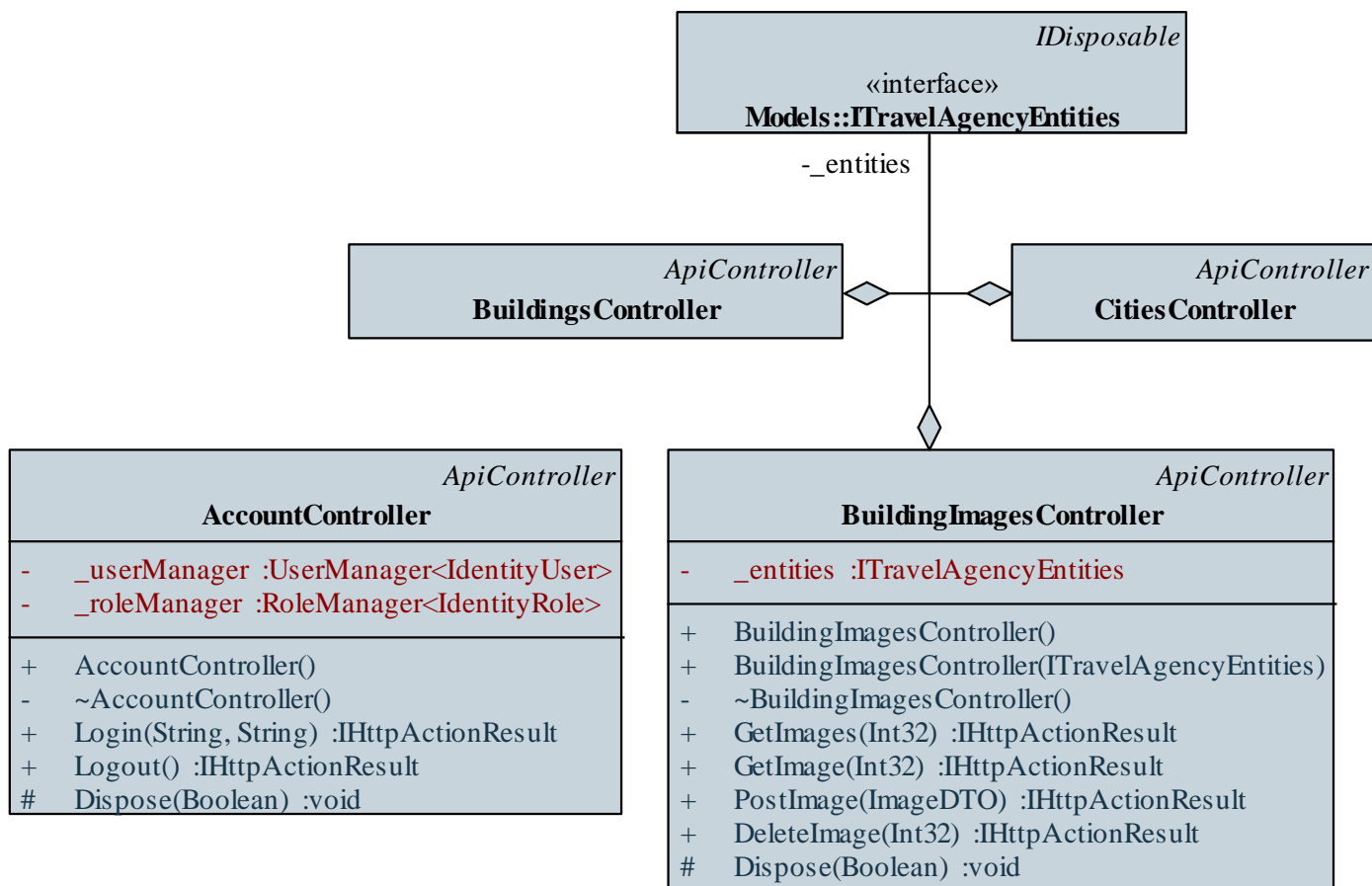
### *Tervezés:*

- létrehozunk egy vezérlőt a felhasználó-kezeléshez (**AccountController**), ebben lehetőséget adunk bejelentkezésre (**Login**) és kijelentkezésre (**Logout**)
- a szolgáltatásban attribútum (**Authorize**) segítségével korlátozzuk az akciófüggvényekhez való hozzáférést (csak a rendszergazda csoportban lévő felhasználókra)
- kliens oldalon megjelenik a két új művelet a modellben (**LoginAsync**, **LogoutAsync**)
- a bejelentkezéshez egy külön nézetet (**LoginWindow**), valamint nézetmodellt (**LoginViewModel**) hozunk létre

# Webszolgáltatások felhasználása

## Példa

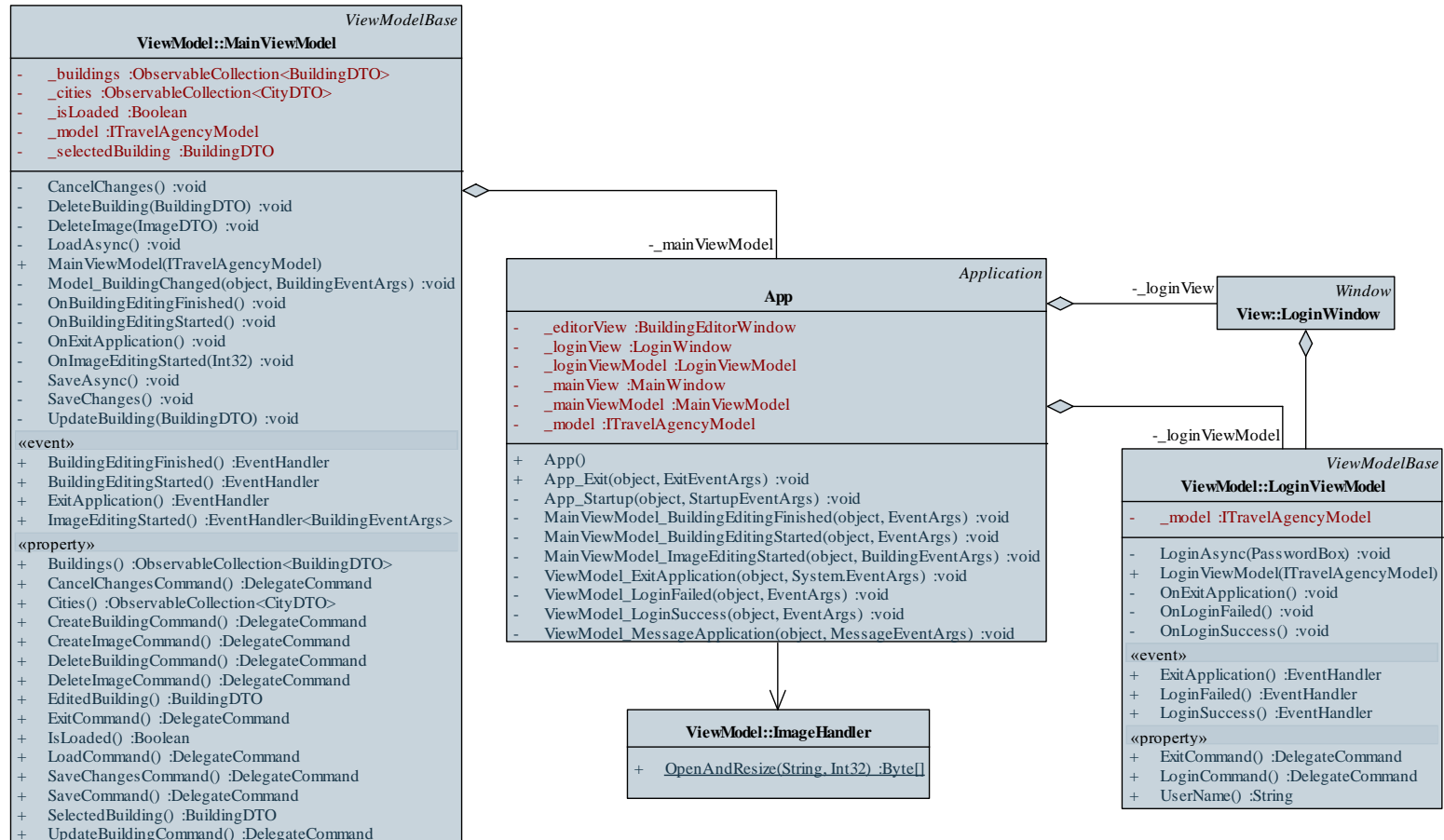
*Tervezés (szolgáltatás):*



# Webszolgáltatások felhasználása

## Példa

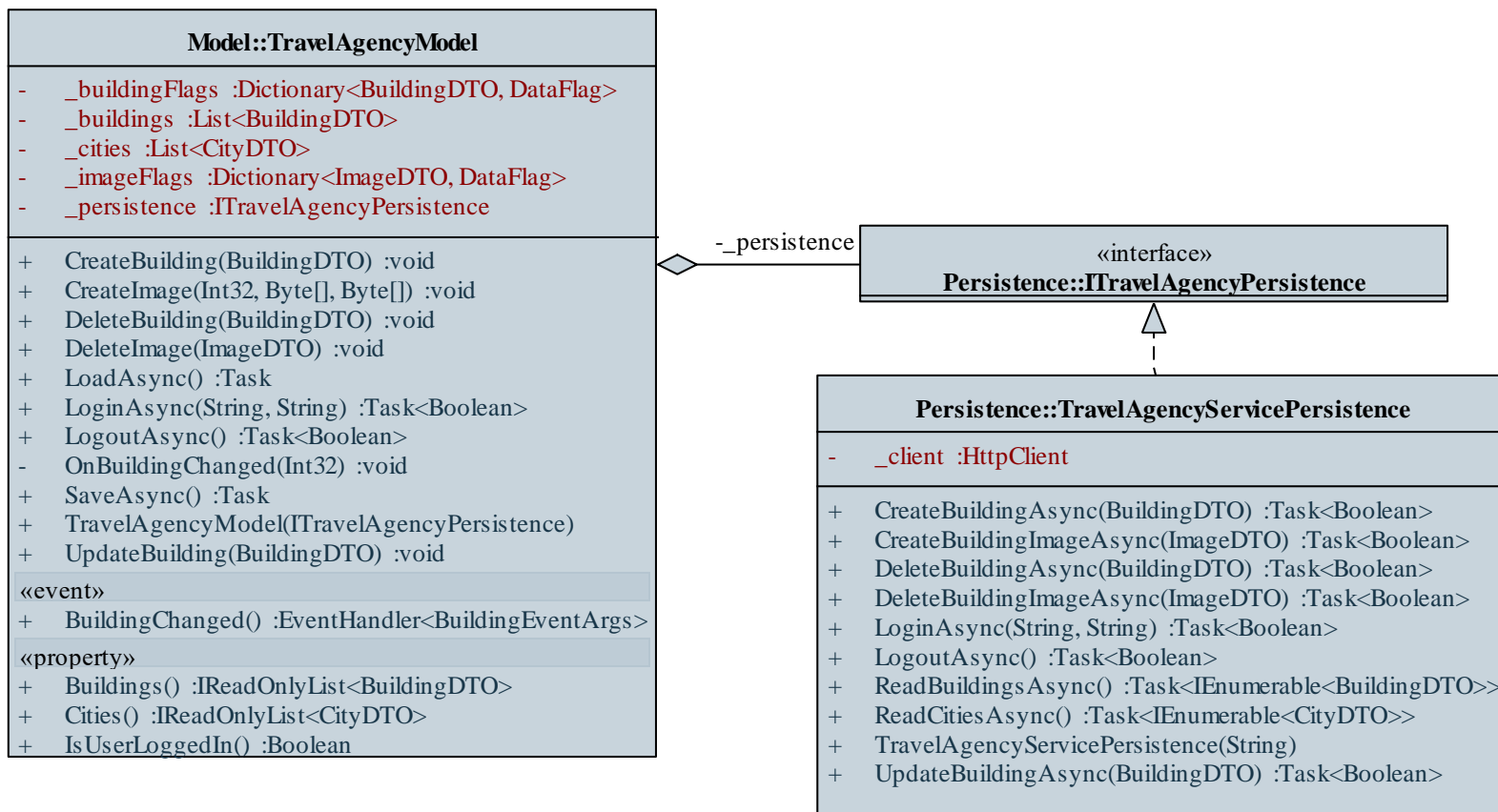
### Tervezés (kliens):



# Webszolgáltatások felhasználása

## Példa

*Tervezés (kliens):*



# Webszolgáltatások felhasználása

## Példa

---

*Megvalósítás* (BuildingImagesController.cs):

```
[Authorize(Roles = "administrator")]
    // csak bejelentkezett adminisztrátoroknak
public IHttpActionResult PostImage([FromBody]
    ImageDTO image)
{
    ...
    _entities.SaveChanges();
    return Created(Request.RequestUri +
        image.Id.ToString(), image.Id);
    // csak az azonosítót küldjük vissza
    ...
}
```

# Webszolgáltatások felhasználása

## Példa

*Megvalósítás (ImageHandler.cs):*

```
public static Byte[] OpenAndResize(String path,
                                    Int32 height)
{
    BitmapImage image = new BitmapImage();
    // kép betöltése
    image.BeginInit();
    image.UriSource = new Uri(path);
    image.DecodePixelHeight = height;
    // megadott méretre
    image.EndInit();
}
```

# Webszolgáltatások felhasználása

## Példa

*Megvalósítás (ImageHandler.cs):*

```
PngBitmapEncoder encoder =  
    new PngBitmapEncoder();  
    // átalakítás PNG formátumra  
encoder.Frames.Add(BitmapFrame.Create(image));  
  
using (MemoryStream stream =  
    new MemoryStream())  
    // átalakítás byte-tömbre  
    {  
        encoder.Save(stream);  
        return stream.ToArray();  
    }  
}
```