



**Eötvös Loránd Tudományegyetem  
Informatikai Kar**

# **Webes alkalmazások fejlesztése**

---

## **5. fejezet**

### **Adatbevitel és validáció (ASP.NET)**

---

**Giachetta Roberto**

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

# Adatbevitel és validáció

## Űrlapok

---

- Sok esetben szükséges, hogy a felhasználó adatokat vigyen fel a weblapokon (pl. felhasználónév/jelszó), ezt *űrlapok* (**form** elem) keretében teheti meg
  - az űrlapokban vezérlőket helyezünk el, amelyeknek tartalmát **POST** típusú kérésben tudjuk a szerverre küldeni
  - űrlapokat a **Html.BeginForm** művelettel tudunk létrehozni
    - egy **@using** blokkba helyezzük, ez megadja a hatókörét
    - az űrlapon belül beviteli mezőket (**input** elemeket) használunk, elküldéséhez pedig egy gombot (**submit** típusú **input** elemet)
    - a **value** attribútummal megadjuk, a modell mely értékeit (tulajdonságait) visszük be

# Adatbevitel és validáció

## Űrlapok

---

- pl.:

```
public class UserData {  
    // a nézetmodell típusa  
  
    public Int32 UserId { get; set; }  
        // felhasználó azonosítója  
    public String UserName { get; set; }  
        // felhasználónév  
    public String UserPass { get; set; }  
        // jelszó  
    public String Birthdate { get; set; }  
        // születési idő  
}
```

# Adatbevitel és validáció

## Űrlapok

---

- pl.:

```
@model UserData
...
@using (Html.BeginForm()) { // űrlap kezdete
    <div>Your name:
        <input name="userName"
            value="@Model.UserName" />
        @* szövegbeviteli mező, amelyben a modell
            UserName tulajdonságát állítjuk be *@
    </div>
    ... @* további adatbekérés *@
    <input type="submit" value="Login" />
    @* űrlap elküldő gomb *@
} // űrlap vége
```

# Adatbevitel és validáció

## Űrlapok

---

- Az űrlapok ugyanúgy egy akciót futtatnak, ám átadják ennek az akciónak a bevitt modell adatokat
  - alapértelmezetten ugyanazt az akciót futtatják, amely létrehozta a nézetüket, de ezt paraméterben megadhatjuk, illetve lehetőségünk van átirányításra, pl.:

```
@using (html.BeginForm("Index",  
                        "LoginController", ...)) ...
```
  - a vezérlőben megadhatjuk, hogy egy akció csak a **GET**, vagy **POST** kérésre hajtódjon végre (**HttpGet** és **HttpPost** attribútum), így szétválasztható a két működés
    - attribútumok nélkül egy műveletben kell a két állapotot kezelni (mivel ilyenkor túlterhelés nem engedélyezett)

# Adatbevitel és validáció

## Űrlapok

---

- pl.:

```
public class LoginController : Controller {
    [HttpGet] // ez fut le az oldal betöltésére
    public ActionResult Index() {
        return View(); // itt még csak üresen
                        // prezentáljuk a nézetet
    }

    [HttpPost] // ez fut le az űrlap elküldésére
    public ActionResult Index(UserData data) {
        // paraméterben megkapjuk az űrlapban
        // kitöltött modellt
        ...
    }
}
```

# Adatbevitel és validáció

## Adatbevitel űrlapokban

---

- Az űrlapon belül a beviteli mezőket műveletek segítségével is előállíthatjuk, pl.:  
`@Html.TextBox("userName", "@Model.UserName")`
- Az űrlapon belül a beviteli mezőket (erősen típusos nézetben) egy adott tulajdonságra is generálhatjuk, pl.:  
`@Html.TextBoxFor(m => m.UserName)`
- A következő beviteli mezőket használhatjuk:
  - szövegdoboz (**TextBox**), szövegmező (**TextArea**), jelszómező (**Password**)
  - kijelölő (**CheckBox**), rádiógomb (**RadioButton**), legördülő menü (**DropDownList**), lista (**ListBox**)

# Adatbevitel és validáció

## Adatbevitel űrlapokban

---

- pl.:

```
@model UserData
...
@using (Html.BeginForm()) { // űrlap kezdete
    <div>Your name:
        @Html.TextBoxFor(m => m.UserName)</div>
    <div>Your password:
        @Html.PasswordFor(m => m.UserPass)</div>
    @* a beviteli mezőket generáljuk a
        tulajdonságokhoz *@
    ...
    <input type="submit" value="Login" />
} // űrlap vége
```



# Adatbevitel és validáció

## Példa

*Feladat:* Valósítsuk meg az utazási ügynökség weblapjának foglalási funkcióját, azaz egy apartmant kiválasztva legyen lehetőség a foglaló adatival adott hetekre lefoglalni.

- felvesszünk egy új vezérlőt, amely a foglalásokat felügyeli (**RentController**),
  - a vezérlőben az **Index** művelet szolgálja ki a **GET** és **POST** kéréseket (paramétere az apartman, illetve utóbbinak a megadott adatok)
- A vezérlőhöz két nézetet vesszünk fel:
  - egyikben űrlapban megadjuk az adatokat (**Index**)
  - egy másikban visszajelezzük, hogy sikeres volt a foglalás (**Result**), és megadjuk a teljes összeget

# Adatbevitel és validáció

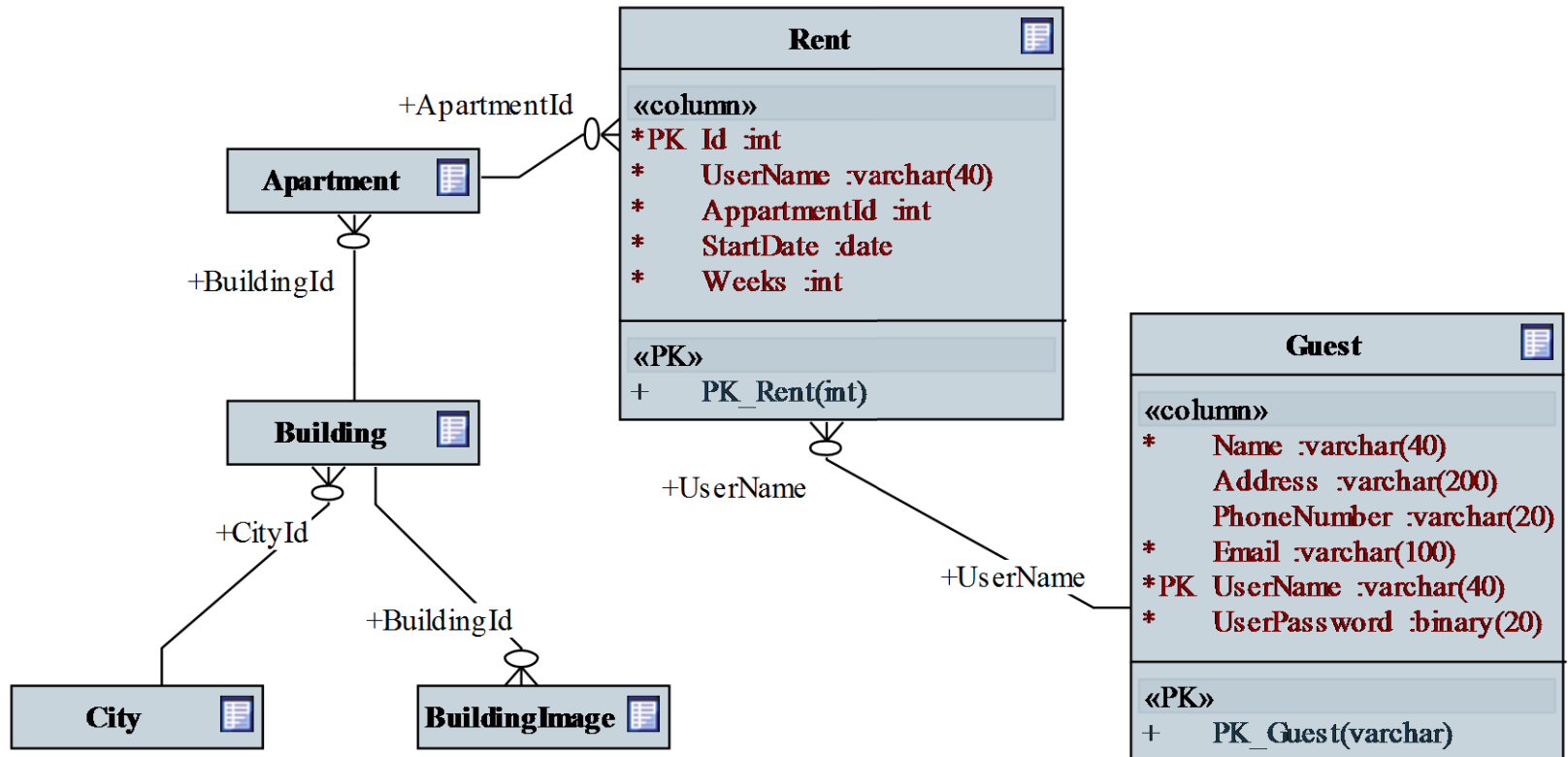
## Példa

- a foglalás és a foglaló adatait az adatbázisban két külön táblában tároljuk (**Rent**, **Guest**)
  - kiegészítjük az entitásmodell osztályait kényelmi funkciókkal (**Rent** típust az ütközésetektálással, **Apartment** típust a hét napjának lekérdezésével)
- az adatokat egyszerre szeretnénk bekérni, így létrehozunk egy nézetmodell osztályt (**RentViewModel**)
- minden megadott adatot ellenőrzünk, mielőtt mentenénk (pl. mezők kitöltöttsége, foglalási időpontok megfelelősége, illetve ütközése)
  - amíg nem hibátlan a kitöltés, visszairányítjuk a kitöltő oldalra (és megjelenítünk egy hibaüzenetet is)

# Adatbevitel és validáció

## Példa

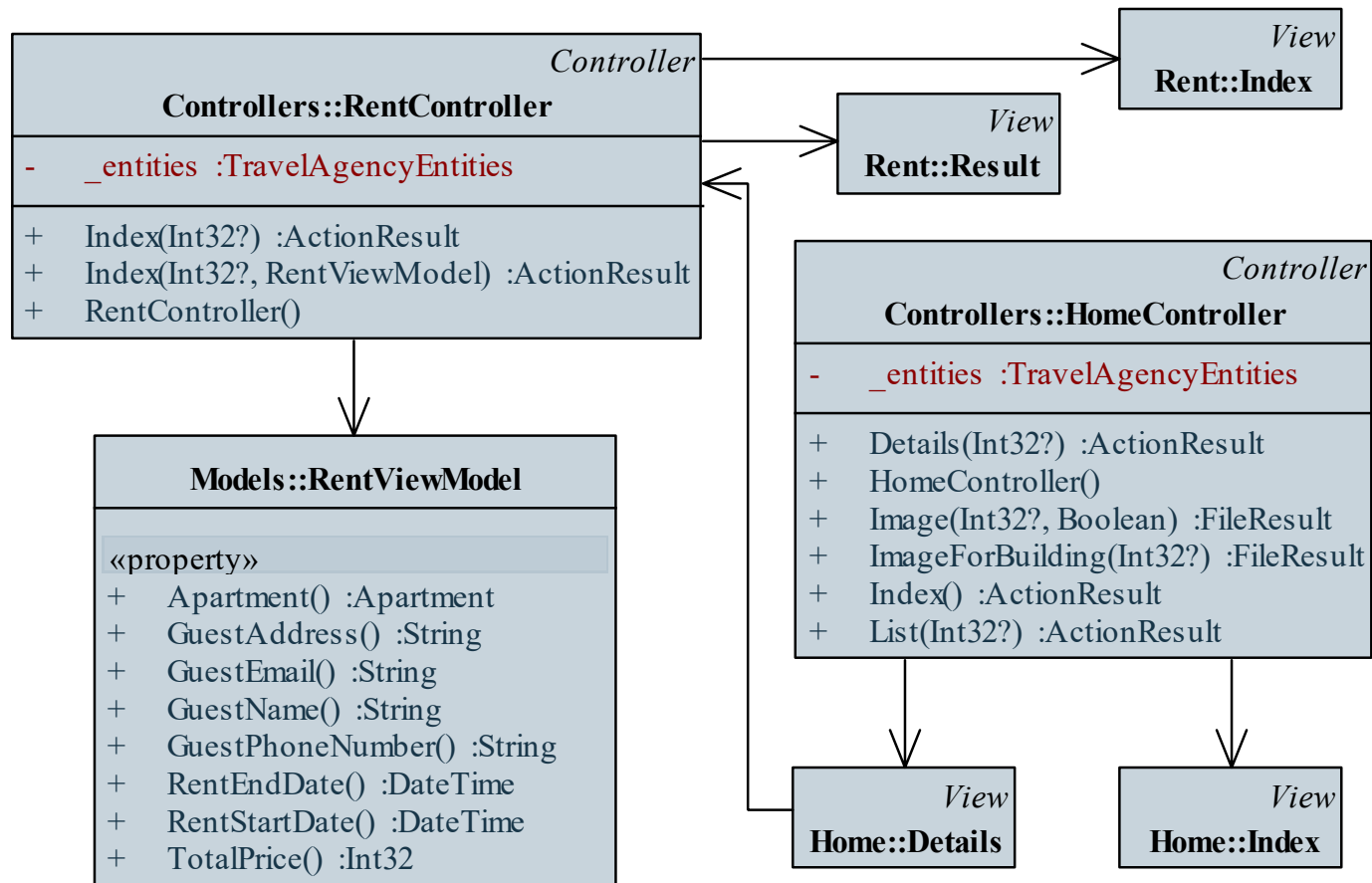
*Tervezés (adatbázis):*



# Adatbevétel és validáció

## Példa

Tervezés (alkalmazás):



# Adatbevitel és validáció

## Példa

---

*Megvalósítás (Rent/Index.cshtml):*

...

```
@* megjeleníthetünk üzeneteket *@
```

```
@if (ViewBag.Error != null) {
```

```
    <div ...>@ViewBag.Error</div>
```

```
}
```

...

```
@using (Html.BeginForm()) {
```

```
    // egy űrlapban töltjük ki a tartalmat
```

```
    @Html.TextBoxFor(rent => rent.CustomerName,
```

```
        new { size = "40" })
```

```
@* szövegbeviteli mezőt generálunk, amelynek
```

```
    megadjuk a méretét is *@
```

...

# Adatbevitel és validáció

## Példa

---

*Megvalósítás (RentController.cs):*

...

```
[HttpPost]
```

```
public ActionResult Index(Int32? apartmentId,  
                           RentInformation rent) {
```

...

```
// végrehajtjuk az ellenőrzéseket
```

```
if (String.IsNullOrEmpty(rent.CustomerName) ...) {
```

```
    ViewBag.Error = "Hiányosak a foglaló  
                    adatai!";
```

```
    return View("Index", rent);
```

```
}
```

...

# Adatbevétel és validáció

## Adatbevétel űrlapokban

---

- Amennyiben nem ismerjük előre a modelltulajdonság típusát, használhatunk dinamikusan generált elemeket:
  - az **Editor** művelet dinamikusan generálja a beviteli mezőt
  - a **Label** művelet címkét hoz létre a megadott tulajdonsághoz, míg a **Display** csak olvasható módon jeleníti meg a tartalmat
- Amennyiben nem egyenként szeretnénk bekérni a tartalmat, a teljes nézetmodell összes adatát megjeleníthetjük (**LabelForModel**, **DisplayForModel**), vagy szerkeszthetjük (**EditorForModel**)
  - ekkor célszerű annotációkkal felruházni a nézetmodellt

# Adatbevitel és validáció

## Adatbevitel űrlapokban

---

- pl.:

```
@model UserData // felhasználói adatok
...
@using (Html.BeginForm()) { // űrlap kezdete
    <div>@Html.LabelFor(m => m.UserName) :
        @Html.EditorFor(m => m.UserName)</div>
    /* a szerkesztő és a címke is dinamikus */
    ...
    <div>@Html.LabelFor(m => m.Birthdate) :
        @Html.EditorFor(m => m.Birthdate)</div>
    /* itt egy dátumbekérő fog megjelenni */

    <input type="submit" value="Login" />
} // űrlap vége
```



# Adatbevitel és validáció

## Adatbevitel űrlapokban

---

- pl.:

```
@model UserData // felhasználói adatok
...
@using (Html.BeginForm()) { // űrlap kezdete
    @Html.EditorForModel()
    @* a teljes modelltartalmat szerkeszthetővé
       tesszük *@

    <input type="submit" value="Login" />
    @* már csak a bejelentkező gombra van
       szükség *@
} // űrlap vége
```

# Adatbevitel és validáció

## Űrlapok modelljei

---

- A nézetmodellünket, és annak tulajdonságait számos *annotációval* (attribútummal) megjelölhetjük, amelyek az űrlapmezők dinamikus generálását befolyásolják
  - így a generáláskor jobban szabályozhatjuk az adatok megjelenítésének/bekérésének módját
  - pl.:
    - megjelenő címke (**Display**), illetve tartalom megjelenés formátuma (**DisplayFormat**)
    - elrejtés (**HiddenInput**)
    - specifikusabb adattípus (**DataType**), illetve beviteli mező specifikálása (**UIHint**)

# Adatbevitel és validáció

## Űrlapok modelljei

---

- pl.:

```
[DisplayName("User login:")]
    // megadjuk a bekérő lapnak a címszövegét
public class UserData {
    [HiddenInput(DisplayValue=false)]
        // ez a mező rejtett lesz, így nem
        // jelenik meg
    public Int32 UserId { get; set; }
        // azonosító

    [Display(Name="Your name:")]
        // a címkén a megadott szöveg jelenik meg
    public String UserName { get; set; }
        // felhasználónév
```

# Adatbevitel és validáció

## Űrlapok modelljei

---

```
[Display(Name="Your password:")]
[UIHint("Password")]
    // a szerkesztőmező egy jelszómező lesz
public String UserPass { get; set; }
    // jelszó

[Display(Name="Your birthday:")]
[DisplayFormat(DataStringFormat="yy.MM.dd")]
[DataType(DataType.Date)]
    // csak a dátum fog megjelenni, a megadott
    // formátumban
public DateTime Birthdate { get; set; }
    // születési dátum
}
```

# Adatbevitel és validáció

## Megjelenítő sablonok

---

- Megjelenítő (**DisplayFor**), illetve beviteli (**EditorFor**) mezőnek nem csak beépített elemeket, hanem általunk létrehozott parciális nézeteket is használhatunk, mint sablonokat (*display template*)
  - ezeket a **Views/Shared/DisplayTemplate** könyvtárba helyezzük
  - a tulajdonság ennek a nézetnek a modellje lesz
  - pl.:

```
[Display(Name="Your name:")]  
[UIHint("MyNameDisplay")]  
    // a MyNameDisplay.cshtml nézetet tölti be  
public String UserName { get; set; }
```

# Adatbevitel és validáció

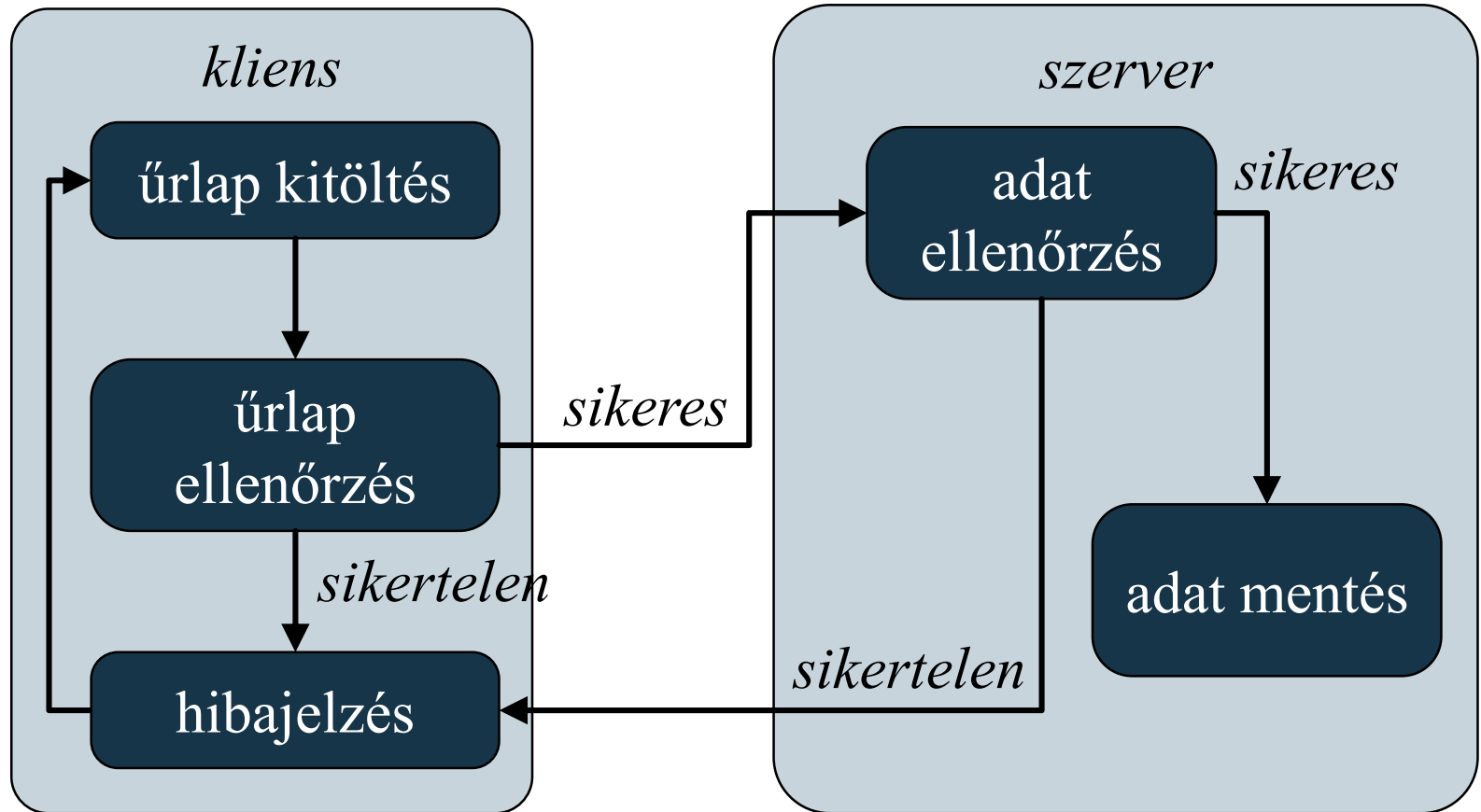
## Validáció

---

- A felhasználótól bekért adatokat mindig ellenőriznünk kell, ez a *modell validációja*, amely a következő lépésekből áll:
  1. a kliens oldalon, amint a felhasználó elküldi az űrlapot:
    - ellenőrizzük, hogy a szükséges adatokat megadták
    - ellenőrizzük, hogy a típusa és formátumuk helyes (pl. e-mail, dátum)
    - jelezzük a felhasználónak, ha bármilyen probléma van
  2. sikeres ellenőrzés esetén az adatok a szerverre kerülnek
  3. a szerver oldalon ismét lefutnak az ellenőrzések, immár *biztonsági ellenőrzéseket* is futtatva
  4. hiba esetén visszajelzünk a kliensnek, egyébként mentünk

# Adatbevitel és validáció

## Validáció



# Adatbevitel és validáció

## Validáció

---

- A validáció elvégezhető csak szerver oldalon, vagy kliens és szerver oldalon
  - a szerver oldali ellenőrzés mindenképpen szükséges, főleg támadások kivédése miatt
  - a validálás elvégezhető teljesen manuálisan, vagy használhatóak beépített eszközök
- Szerver oldalon a modell állapotát a vezérlőben a **ModelState** tulajdonságon keresztül tudjuk kezelni
  - az **IsValid** érték megadja, hogy minden szükséges érték megtalálható, és típusa megfelelő
  - az **AddModelError** művelettel jelezhetünk egy hibát



# Adatbevitel és validáció

## Validáció

---

- pl.:

```
public class LoginController : Controller {
    ...
    [HttpPost]
    public ActionResult Index(UserData data) {
        if (String.IsNullOrEmpty(data.UserName))
            // ha üresen hagyták a nevet
            ModelState.AddModelError("UserName",
                "User name is required!");
            // jelezzük a hibát a tulajdonságra
        ...
        if (ModelState.IsValid)
            ... // ha egyébként jók az adatok
    }
}
```

# Adatbevitel és validáció

## Validáció a nézetben

---

- A hibákat globálisan, vagy az egyes tulajdonságokra egyenként is megadhatjuk (előbbi esetben nem adjuk meg a tulajdonságot)
- A nézetben a hibajelzéseket jelezhetjük
  - egy tulajdonságra a `Html.ValidationMessageFor` művelet írja ki a jelzett hibaüzenetet
  - a teljes modellre `Html.ValidationSummary` művelet írja ki a hibaüzeneteket
    - paraméterben megadhatjuk, hogy az egyes tulajdonságok hibáit is kiírja, vagy csak azokat, amelyekhez nem adtunk meg tulajdonságot (`Html.ValidationSummary(true)`)

# Adatbevitel és validáció

## Validáció a nézetben

---

- pl.:

```
@using (Html.BeginForm()) {  
    <div>  
        @Html.ValidationSummary(true, "Errors:")  
        @* a globálisan jelzett hibák *@  
    </div>  
    <div>  
        @Html.LabelFor(m => m.UserName) :  
        @Html.EditorFor(m => m.UserName)  
        @Html.ValidationMessageFor(m =>  
            m.UserName)  
        @* a UserName-re jelzett hiba *@  
    </div>  
    ...  
}
```

# Adatbevitel és validáció

## Validáció a nézetmodellben

---

- Lehetőségünk van a nézetmodellen közvetlenül megadni ellenőrzési kritériumokat, automatizálva az ellenőrzést
  - a feltételeket tulajdonságonként szabályozhatjuk, és megadhatjuk a hibaüzenetet (**ErrorMessage**)
  - megadhatjuk a kötelező kitöltést (**Required**), szöveghosszt (**StringLength**), reguláris kifejezést (**RegularExpression**), intervallumot (**Range**), összehasonlítást más tulajdonsággal (**Compare**), illetve speciális formátumot (**Url**, **Phone**, **CreditCard**, **EmailAddress**, ...)
  - a **Validation** osztály **TryValidateObject** metódusával az ellenőrzés elvégezhető manuálisan is

# Adatbevitel és validáció

## Validáció a nézetmodellben

---

- pl.:

```
public class UserData {  
    ...  
    [Required(ErrorMessage="User name is  
                    required.")]  
    [StringLength(15, "User name cannot be  
                    longer, than 15 characters.")]  
    [RegularExpression("^ [a-z0-9_ -]{3,15}$",  
                    "User name has invalid characters.")]  
    // feltételek a felhasználónévre  
    public String UserName { get; set; }  
    // felhasználónév  
    ...  
}
```

# Adatbevitel és validáció

## Példa

---

*Feladat:* Valósítsuk meg az utazási ügynökség weblapjának foglalási funkcióját, azaz egy apartmant kiválasztva legyen lehetőség a foglaló adatival adott hetekre lefoglalni.

- a megjelenítéshez és validációhoz annotációkat használunk a `RentViewModel` osztályban
- a hét napja, illetve a tengerpart típusa speciális megjelenítést igényelnek több oldalon, ezért a korábbi lambda-kifejezéseket cseréljük le egy-egy parciális nézetre (`DayOfWeekDisplay`, `ShoreTypeDisplay`), és használjuk fel ezeket a megjelenítésre
  - ehhez a megfelelő entitásosztályokat is kiegészítjük

# Adatbevétel és validáció

## Példa

---

*Megvalósítás (RentViewModel.cs):*

```
public class RentViewModel {  
    ...  
    [Required(ErrorMessage =  
        "Az e-mail cím megadása kötelező.")]  
    [EmailAddress(ErrorMessage =  
        "Az e-mail cím nem megfelelő formátumú.")]  
    [DataType(DataType.EmailAddress)]  
    public String GuestEmail { get; set; }  
    ...  
}
```

# Adatbevitel és validáció

## Példa

---

*Megvalósítás (RentController.cs):*

```
public ActionResult Index(Int32? apartmentId,
                          RentViewModel rent) {
    ...
    // végrehajtjuk az ellenőrzéseket
    if (rent.RentStartDate < DateTime.Now +
        TimeSpan.FromDays(7) || ...)
        ModelState.AddModelError("RentStartDate", ...);
    ...
    if (!ModelState.IsValid)
        return View("Index", rent);
    ...
}
```



# Adatbevitel és validáció

## Példa

*Megvalósítás (Rent/Index.cshtml):*

...

```
<strong>Tengerpart típus: </strong>
```

```
@Html.DisplayFor(r => r.Apartment.Building.  
                    ShoreType)
```

```
@* meghívjuk az egyedi megjelenítőnket *@
```

...

```
@Html.EditorFor(rent => rent.GuestEmail,  
                new { htmlAttributes = new { size = "40" } })
```

```
@* a szerkesztőelemet a modell szabja meg, de így  
    befolyásolhatjuk a méretét *@
```

```
@Html.ValidationMessageFor(rent =>  
                             rent.GuestEmail);
```

...

# Adatbevitel és validáció

## Kliens oldali validáció

---

- A kliens oldali validációt Javascript segítségével végezzük
  - a legegyszerűbb a *jQuery Validation* programcsomag használata, amely automatikusan kezelni tudja a szerver oldali modellben lévő annotációkat, pl.:  
...  
`<script src="jquery-2.1.3.min.js">...`  
`<script src="jquery.validate.js">...`  
`<script src="jquery.validate.unobtrusive.js">...`  
...
    - a validáció az űrlap beküldése előtt, még kliens oldalon megtörténik (voltaképpen az annotációk megfelelő módon beépülnek a HTML vezérlőkbe)

# Adatbevitel és validáció

## Kliens oldali validáció

---

- a validáció nyelv specifikus formátumokat nem támogat (pontosabban csak az **en-US** nyelvi környezetet támogatja)
- nyelvfüggő elemek (pl. dátum) esetén használjuk a *jQuery Globalize* és *jQuery Validation Globalize* programcsomagokat, pl.:

...

```
<script src="globalize.js">...
```

```
<script src="globalize.culture.hu-HU.js">...
```

```
<script src="jquery.validate.globalize.min.js">...
```

```
<script type="text/javascript">
```

```
    $(function() { Globalize.culture("hu-HU"); });
```

```
    // nyelvi környezet beállítása
```

```
</script>
```

# Adatbevitel és validáció

## Biztonsági ellenőrzések

---

- A felhasználó által felvitt adatok kártékony információkat is tartalmazhatnak, ezért biztonsági szempontból is fontos a validálás, a legjellemzőbb támadások:
  - *SQL injekció*: a szerveren futó SQL utasításokat manipulálja
    - entitásmodell használata esetén nem fordulhat elő
  - *cross-site scripting (XSS)*: szkript kerül feltöltésre a szerverre, amelyet a kliens böngészője futtat
    - a bevitel eleve tiltja a HTML elemeket tartalmazó adatok feldolgozását, de ez kikapcsolható (`ValidateInput`)
    - az adatok tartalmát megjelenítéskor a `Html.Encode` utasítással kódolhatjuk, így biztosan nem kerül értelmezésre a szkript

# Adatbevitel és validáció

## Biztonsági ellenőrzések

---

- *cross-site request forgery (XSRF)*: a felhasználó átirányítása, és egy kérés végrehajtása a tudta nélkül
  - a felhasználó elküldi az űrlapot tartalommal (**POST**), anélkül, hogy megadta volna adatait
  - ez elkerülhető, ha megbizonyosodunk róla, hogy a kitöltést és a küldést is ugyanazon kliens végezte
  - ehhez az űrlapban elhelyezünk egy token (`Html.AntiForgeryToken()`), amely információkat közöl a klienssel
  - az akció végrehajtásakor lekérhetjük a token (`ValidateAntiForgeryToken` attribútum), ha a két érték egyezik, akkor nem volt támadás

# Adatbevitel és validáció

## Példa

*Feladat:* Valósítsuk meg az utazási ügynökség weblapjának foglalási funkcióját, azaz egy apartmant kiválasztva legyen lehetőség a foglaló adatival adott hetekre lefoglalni.

- elvégezzük kliens oldalon is a validációt, a dátum ellenőrzéséhez használjuk a nemzetközi csomagot (`_Layout.cshtml`)
- az aktuális nyelvi beállítást elkérhetjük a nézettől (`Culture` tulajdonság)
- az XSRF támadások ellen is védjük az oldalt a megfelelő helyeken (`Rent/Index.cshtml`, `RentController`)

# Adatbevitel és validáció

## Példa

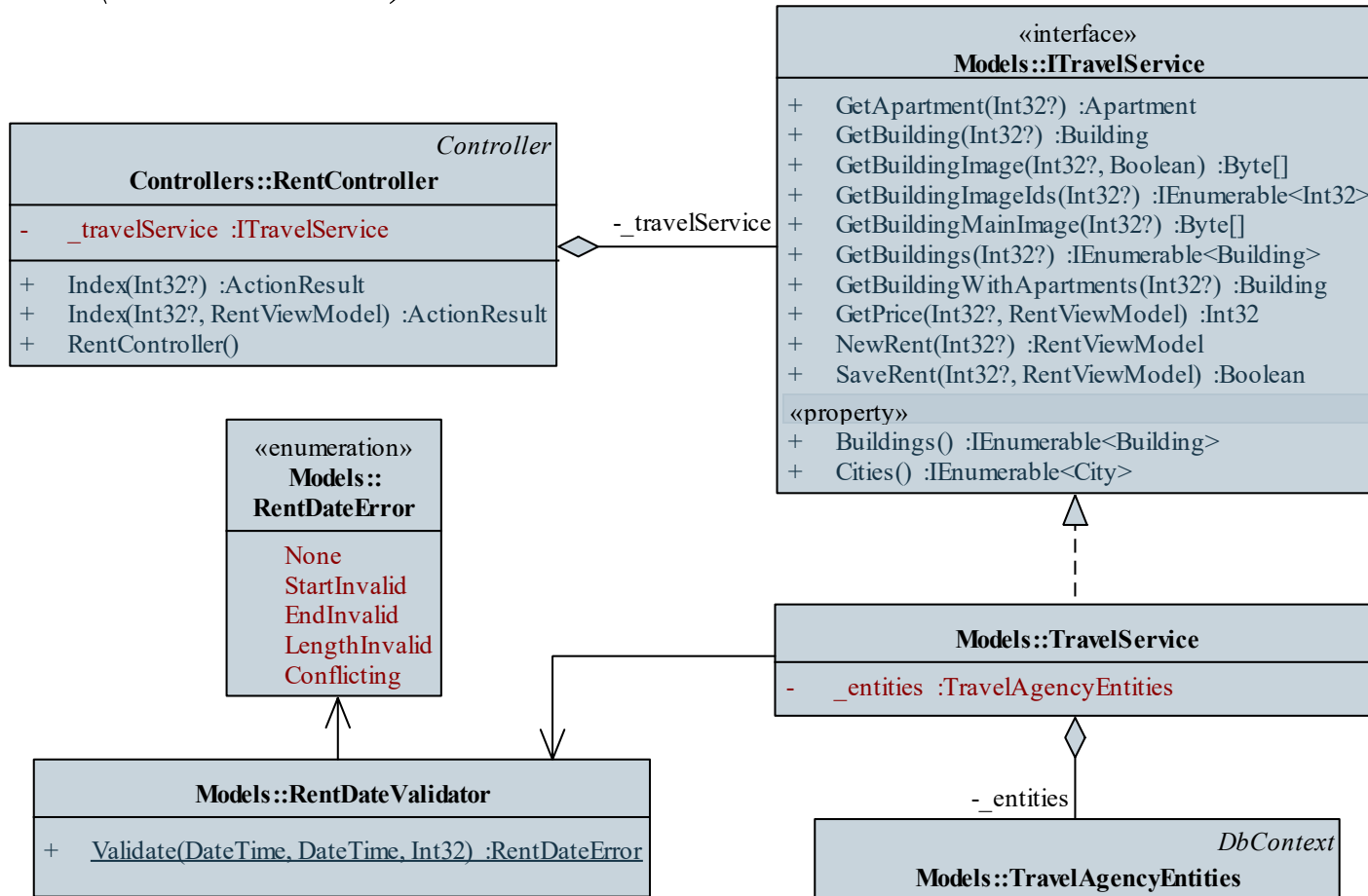
*Feladat:* Valósítsuk meg az utazási ügynökség weblapjának foglalási funkcióját, azaz egy apartmant kiválasztva legyen lehetőség a foglaló adatival adott hetekre lefoglalni.

- a vezérlőkben található üzleti logikát célszerű kihelyezni külön osztályba, legyen ez a **TravelService**
  - biztosítja a kapcsolatot a perzisztenciával, az adatok lekérdezését, a foglalás végrehajtását
  - az adatok ellenőrzését is elvégzi, a műveletek logikai értékkel, vagy hibakóddal térnek vissza (pl. **RentDateError**)
- egy segédosztály (**RentDateValidator**) ellenőrzi a dátumokat

# Adatbevétel és validáció

## Példa

Tervezés (alkalmazás):





# Adatbevitel és validáció

## Példa

---

*Megvalósítás (RentController.cs):*

```
public ActionResult Index(Int32? apartmentId) {  
    // létrehozunk egy foglalást csak az  
    // alapadatokkal (apartman, dátumok)  
    RentViewModel rent =  
        _travelService.NewRent(apartmentId);  
  
    if (rent == null)  
        // ha nem sikerül (nem volt azonosító)  
        return RedirectToAction("Index", "Home");  
        // visszairányítjuk a főoldalra  
  
    return View("Index", rent);  
}
```

# Adatbevitel és validáció

## Példa

---

*Megvalósítás (TravelService.cs):*

```
public Boolean SaveRent(Int32? apartmentId, ...) {  
    ...  
    // ellenőrizzük az annotációkat  
    if (!Validator.TryValidateObject(rent,  
        new ValidationContext(rent, ...), null))  
        return false;  
  
    // ellenőrizzük a dátumot  
    if (RentDateValidator.Validate(  
        rent.RentStartDate, rent.RentEndDate,  
        apartmentId.Value) != RentDateError.None)  
        return false;  
    ...  
}
```