MVVM
Model View View Model

Vastag Atila

2024

“Rend a lelke mindennek”, ahogy mondani szokás. Ez a mondás pedig hatványozottan igaz a kódolás esetében. Biztosan fel tudunk eleveníteni magunkban olyan eseteket, amikor bármiféle tervezés nélkül nekifogtunk egy újabb projektnek, majd amikor már huszadszor döbbentünk rá, hogy “ez így mégsem lesz jó”, mi magunk sem találjuk a kiutat a kódból. Nélkülöz mindenféle strukturáltságot, nincsen az elrendezésében logika, kaptunk egy spagetti kódot. Az ilyen esetek elkerülése érdekében érdemes megismerni valamilyen fejlesztési mintát: itt jön képbe az MVVM!


VIEW
felület
VIEWMODEL
a felülethez kötött
adatok
MODEL
adattípusok
osztályok

Az MVVM rövidítés a Model, View, ViewModel szavakból tevődik össze. A minta lényege, hogy alkalmazásunkat három, jól elkülöníthető részre bontsuk szét, melyek így egyrészt jobban átláthatóvá válnak, másrészt pedig a projektmenedzsment is egyszerűbbé válik. Miért érdekelné a felülettervezőt, hogy az alkalmazás a háttérben honnan kapja az adatot? És miért várjuk el tőle, hogy több száz soros kódokat hámozzon át ahhoz, hogy megtalálja, hol is írjuk át kódszinten a felületen lévő szövegeket?


VIEW
felület
VIEWMODEL
a felülethez kötött
adatok
MODEL
adattípusok
osztályok

Alkalmazásunkat tehát három logikai egységből fogjuk felépíteni. A **View** fogja tartalmazni a felületeinket, itt határozzuk meg a különböző képernyők megjelenését. A **Model** – szándékos a sorrend csere – tartalmazza a deklarációját az adategységeknek (osztályok, avagy class-ok), hogy pontosan milyen adattípusokkal is fog dolgozni az alkalmazásunk. A **ViewModel** pedig az utóbbi kettő összekötését oldja meg. Itt kerülnek példányosításra az objektumok, itt határozzuk meg a gyűjteményeket, melyeket valamilyen formában szeretnénk a felületen megjeleníteni.

Nézzünk egy egyszerű példát, mely most egy névjegyzék alkalmazás koncepcióterve lesz:

- **View**: Ez az egység fogja tartalmazni a felületünket. A felület (UI) pedig egy lista lesz nevekkel és telefonszámokkal.

- **Model**: A model egyetlen osztályt fog tartalmazni:

```
public class Contact
{
        public string Name { get; set; }
        public string PhoneNumber { get; set; }
}
```

- **ViewModel:** Itt pedig meghatározunk egy gyűjteményt, mely névjegyeket fog tartalmazni**:**

```
public class MainViewModel
{
        public List ContactList { get; set; }
        public MainViewModel()
        {
                this.ContactList = new List();
                this.ContactList.Add(new Contact() { Name = “John SMITH”, “0 800 1402 587” });
                this.ContactList.Add(new Contact() { Name = “Alan SMITH”, “0 800 1435 114” });
                this.ContactList.Add(new Contact() { Name = “Ted NELSON”, “0 800 0171 009” });
        }
}
```

MVVM-ben az adat két irányú. Ezt a két irányúságot a *Data Bindig* teszi lehetővé.

Ha a *Model* változik, erről értesíti a *ViewModel-t* és a *Data Bindig* változást idéz elő a *View*-ban.

Ha a *View*-ban változás történik (pl. egy szöveges dobozba véltoztatjuk meg a szöveget) arról a *ViewModel*-n keresztül értesül a *Model* is.