

PROGRAMOZÁSI TECHNOLÓGIÁK

6. óra – TERVEZÉSI MINTÁK

TÉMAKÖRÖK:

Tervezési Minták

- Minták típusai
 - Architekturális minták
 - Létrehozási tervezési minták
 - Szerkezeti tervezési minták
 - Viselkedési tervezési minták

Architekturális Minták

- MVC
- ASP.NET MVC
- MPV, MVVM
- Többrétegű architektúra

Létrehozási T. minták

- Singleton
- Prototype
- Factory Method
- Abstract Factory

Szerkezeti T. Minták

- Adapter
- Decorator
- Proxy

Viselkedési T. minták

- State
- Observer
- Template Method
- Strategy
- Visitor

TERVEZÉSI MINTÁK

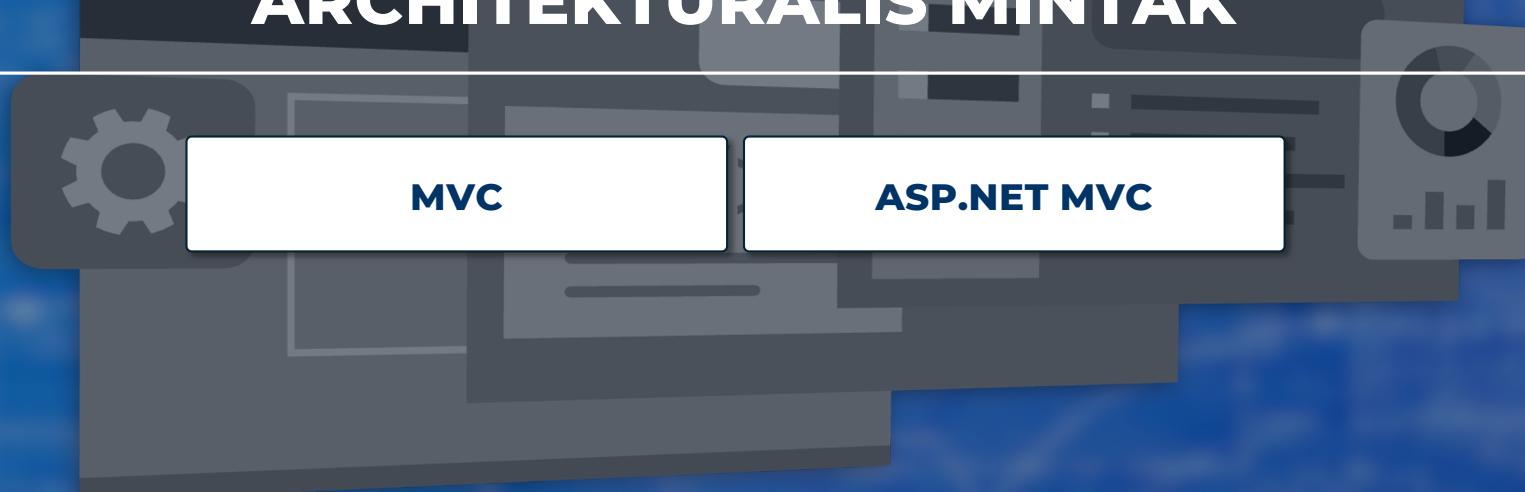
GOF-tól az Agile-ig

- Mire jók a tervezési minták?
Gyakori programozói feladatokat oldanak meg.
- Kik készítik őket?
Gyakorlott programozók, jól bevált best-practice-eket gyűjtenek össze nekünk.
- Mit adnak a tervezési minták?
Könnyen bővíthető, rugalmas osztályszerkezetet.
- Oké, de mi a trükk?
Tudnunk kell alkalmazni technikai osztályokat!

Típusok

- Architekturális minták
Magasszintű felépítési minták
- Létrehozási Tervezési minták
Objektumok létrehozását leíró minták
- Szerkezeti Tervezési minták
Osztály és Objektumkompozíciót leíró minták
- Viselkedési Tervezési minták
Objektum viselkedés és interakció minták

ARCHITEKTURÁLIS MINTÁK



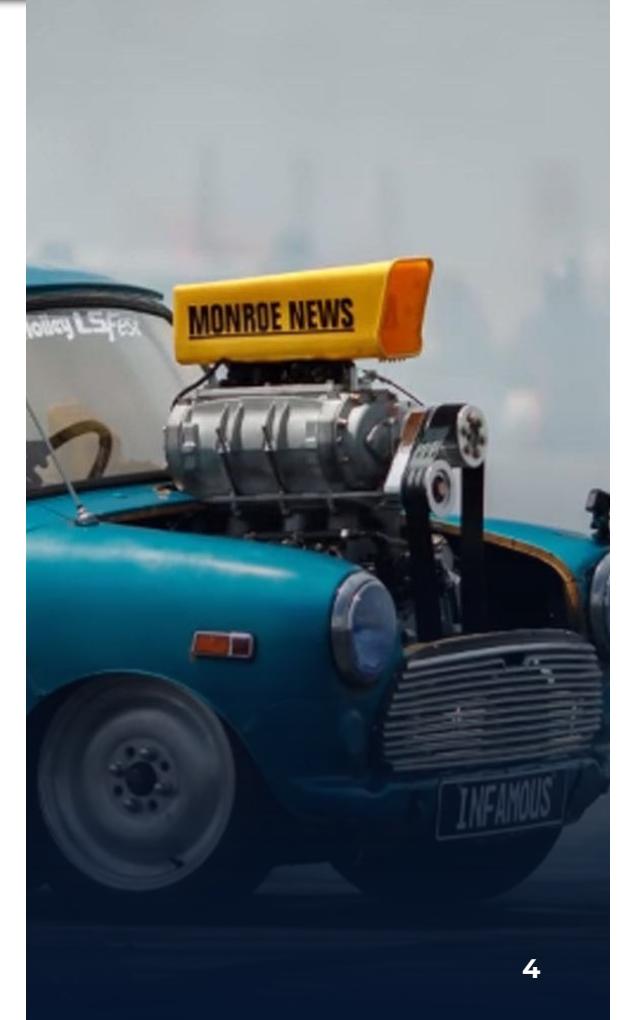
ARCHITEKTURÁLIS MINTÁK

Az Architektúra

- ─ A programnak azon része, amely általában nem változik
- ─ Komponensek, elrendezései és a közöttük lévő relációk
- ─ A teljes rendszer funkcionalitását, teljesítményét és üzemeltetési folyamatait formázza
- ─ Lecserélése olyan, mint egy motorcsere (másik típusra) egy autóban

Órán vizsgált minták

- ─ MVC – Model-View-Controller
- ─ ASP.NET Framework MVC
- ─ MVVM – Model View View-Model
- ─ MVP – Model View Presenter
- ─ Többrétegű Architektúra



ARCHITEKTURÁLIS MINTÁK

MVC – Model-View-Controller

Három fő komponense:

Model

- Az adatokat kezelő réteg
- Tárolás, Visszaolvasás valamint adatokon végzett műveletek függvényei.

View

- A UI megjelenítéséért, valamint Input továbbításáért felelős réteg
- Vezérlőelemek, felhasználónak szánt adatok és azok formázása

Controller

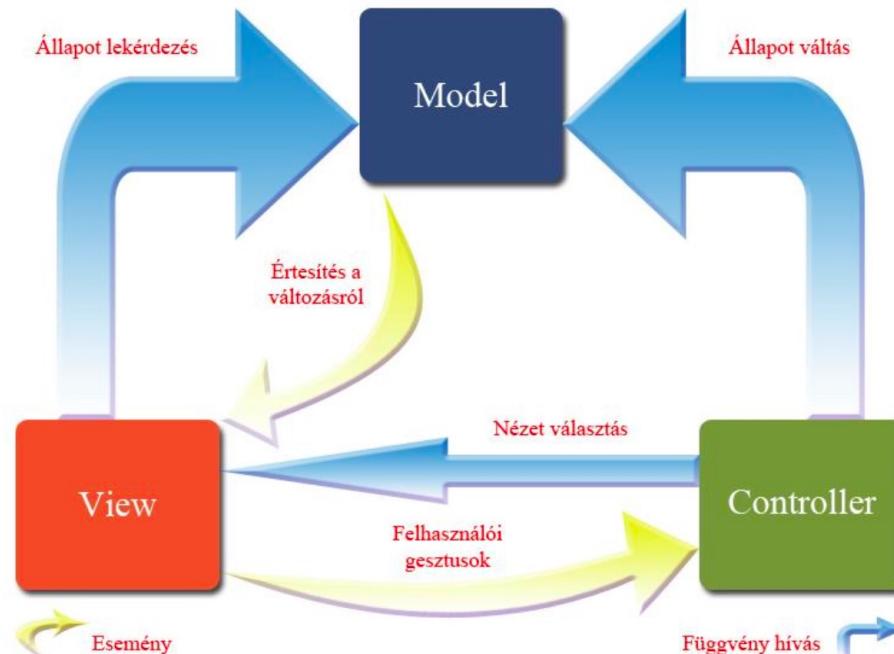
- Vezérlő réteg
- Vezérlőelemek eseményeire reagálva hívja meg a modell megfelelő függvényeit
- Adatok változásánál értesíti a View-t, hogy frissítsen
- Üzleti logika is itt helyezkedik el általában

Kérdés: Mi az üzleti logika?



ARCHITEKTURÁLIS MINTÁK

MVC – Model-View-Controller



14. ábra: Az MVC modell

MVC – Model-View-Controller

- Három egységre bontott alkalmazás
- Külön egység felelősségek
 - Megjelenítés
 - Adatok kezelése
 - Input kezelése
- Ha lecseréljük bármelyiket, a többi még maradhat

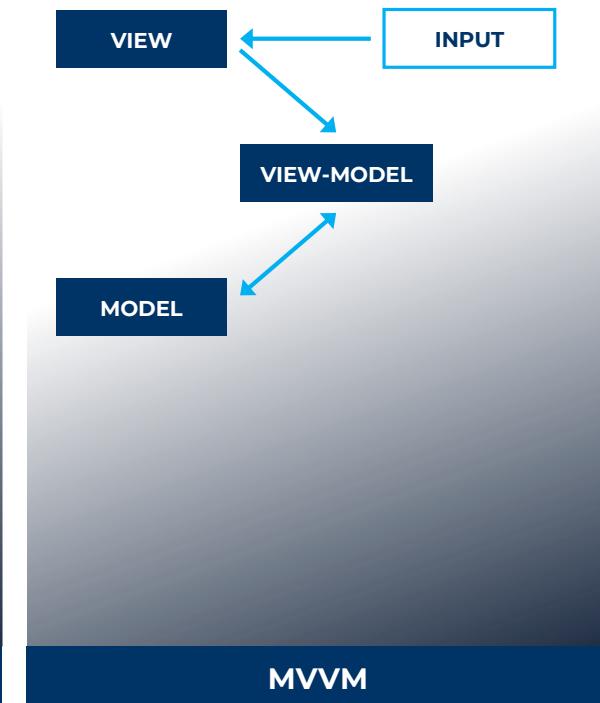
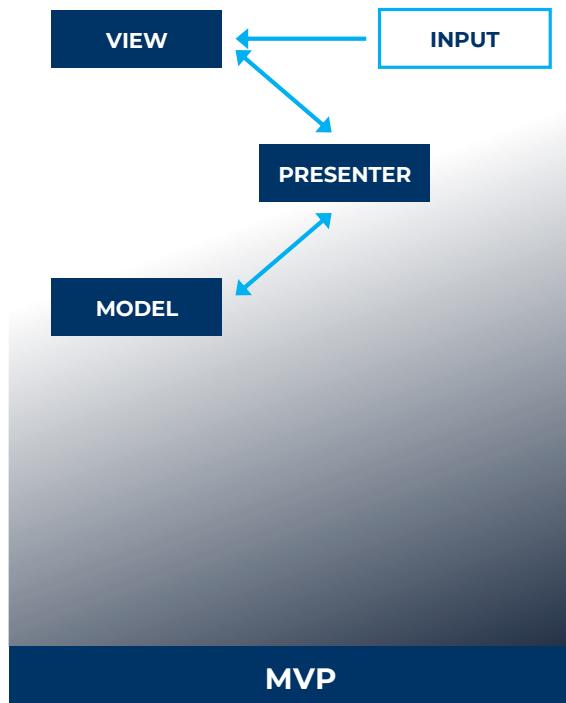
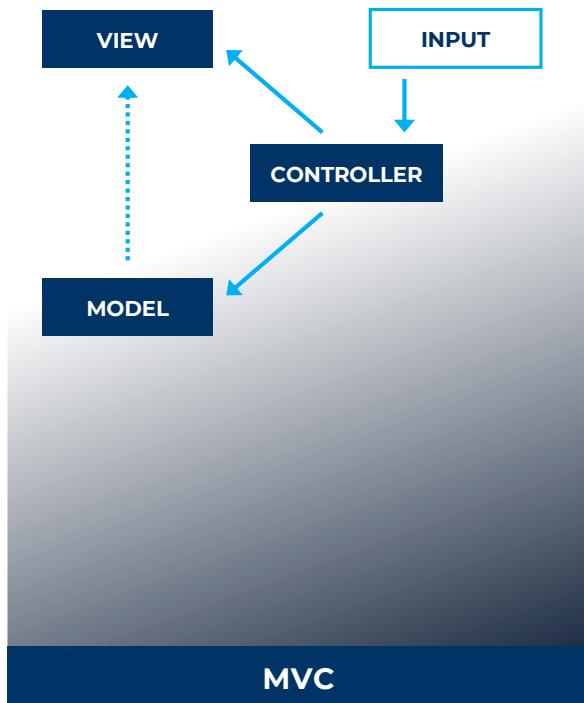
MVC újításai:

- Egy modellhez több nézet is tartozhat
- minden nézet ugyanannak a modellnek a belső állapotát jeleníti meg
- Bármelyik nézeten végzett input ugyanúgy eljut a Controllerhez
- Változás esetén a Controller értesíti a nézeteket

Kérdés: Milyen tanult best-practice-eket fedezhetünk itt fel?

ARCHITEKTURÁLIS MINTÁK

MVC – MVMM – MVP



ARCHITEKTURÁLIS MINTÁK

MVP, Megjelenítő

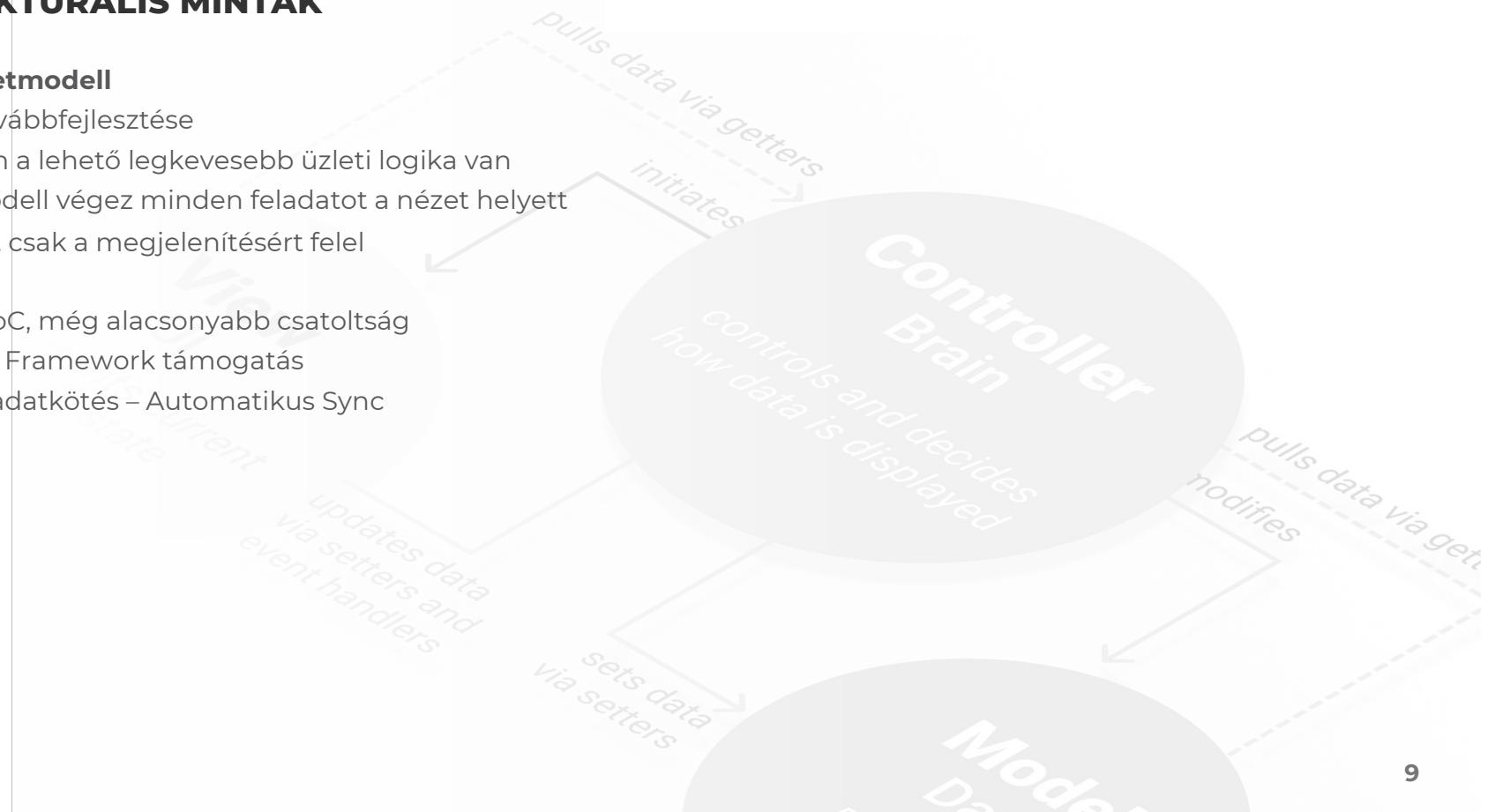
- Az MVC továbbfejlesztése
- A modell nem a nézetet, hanem a megjelenítőt értesíti változás esetén
- A Megjelenítő
 - Lekéri az adatokat a modellből
 - Feldolgozza
 - Megformázza
 - És visszaadja a Nézet számára
- Könnyebb tesztelhetőséget eredményez az MVC-nél.
 - Közvetítő szerepének köszönhetően a Presenterhez könnyebb Business Logic Egységek írni, GUI komponensek bevonása nélkül.
 - MVC-nél a Controller gyakran erős csatoltságban van a Nézzettel és Modellel, emiatt nehezebb elkülöníteni és tesztelni az üzleti logika működését



ARCHITEKTURÁLIS MINTÁK

MVVM, Nézetmodell

- Az MVP továbbfejlesztése
 - A nézetben a lehető legkevesebb üzleti logika van
 - A nézetmodell végez minden feladatot a nézet helyett
 - Így a nézet csak a megjelenítésért felel
-
- Erősebb SoC, még alacsonyabb csatoltság
 - Modern UI Framework támogatás
 - Beépített adatkötés – Automatikus Sync



ARCHITEKTURÁLIS MINTÁK

ASP.NET MVC Framework

- I Fejlesszünk MVC alapú webalkalmazásokat

Előnyök:

- I Integrált ASP.NET Core képességek pl.:
 - I Master page-ek és konzisztens elrendezések
 - I Beépített felhasználókezelés
 - I HTML5 Kompatibilitás
 - I Adatbázis csatlakozók
 - I ...

Nézzünk példákat: [Link](#)



ASP.NET

10

MVC KOMPONENSEK

KOMPONENSEK

LOGIKÁK

MVC KOMPONENSEK

Modell

- Az alkalmazás azon részei, amelyek az adatokat „szállító” logikát implementálják.
- A példában ez egy egyszerű boolean belső állapotot jelöl.
- Továbbá ha elvonatkoztatunk a modelltől, akkor a kapcsoló egy, a Controller felé menő INPUT is.

Vezérlő

- A felhasználói interakciót kezelik, dolgoznak a modellobjektumokkal és kiválasztják a megfelelő nézetet a megjelenítéshez.
- A példánkban egy egyszerű „NOT” inverter kapuként megfordítja az inputot és felkapcsolja a lámpát.

Nézet

- A felhasználói felület megjelenítő komponensei.
- Azokból az adatokból készül, amelyek a modellrétegből átjönnek.
- A példánkban egy bekapcsolt állás jön át, melyet a lámpa fényvel jelenít meg.



MVC KOMPONENTEK

Input logika

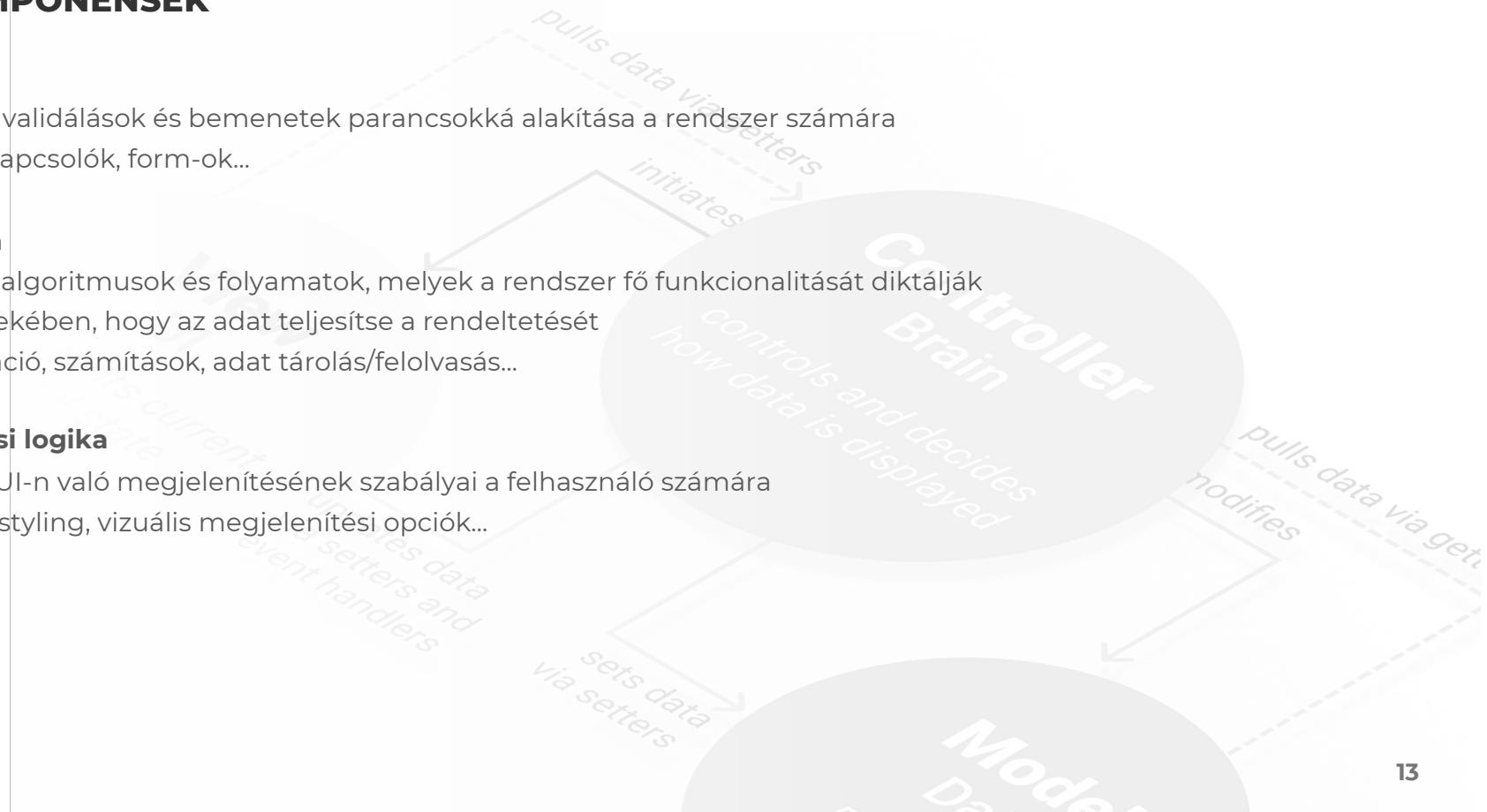
- User Input validálások és bemenetek parancsokká alakítása a rendszer számára
- Gombok, kapcsolók, form-ok...

Üzleti logika

- Szabályok, algoritmusok és folyamatok, melyek a rendszer fő funkcionálisát diktálják
- Annak érdekében, hogy az adat teljesítse a rendeltetését
- Adat validáció, számítások, adat tárolás/felolvasás...

Megjelenítési logika

- Az adatok UI-n való megjelenítésének szabályai a felhasználó számára
- Formázás, styling, vizuális megjelenítési opciók...



TÖBBRÉTEGŰ ARCHITEKTÚRA

RÉTEGEK

PÉLDÁK

TÖBBRÉTEGŰ ARCHITEKTÚRA

Rétegek

- A program jól elszeparált részei
- Akár külön gépen is futhatnak
- Interfészeken keresztül kommunikálnak (M2M és East-to-West kommunikáció)
- Mindig csak a felettük és alattuk levő réteggel kommunikálhatnak
- Nagyon laza csatoltság, csak interfészektől függ a rugalmasság

3 rétegű architektúra

- GUI
- Business Logic
- Database (vagy perzisztencia réteg)

Eltérések az MVC-től

- A GUI nem kommunikálhat az adatbázisréteggel
- Az MVC-nél a modell közvetlenül értesíti a nézeteket a változásokról

TÖBBRÉTEGŰ ARCHITEKTÚRA

Tipikus felépítés

- | Kliens
- | Alkalmazásszerver
- | Adatbázisszerver

A Kliens

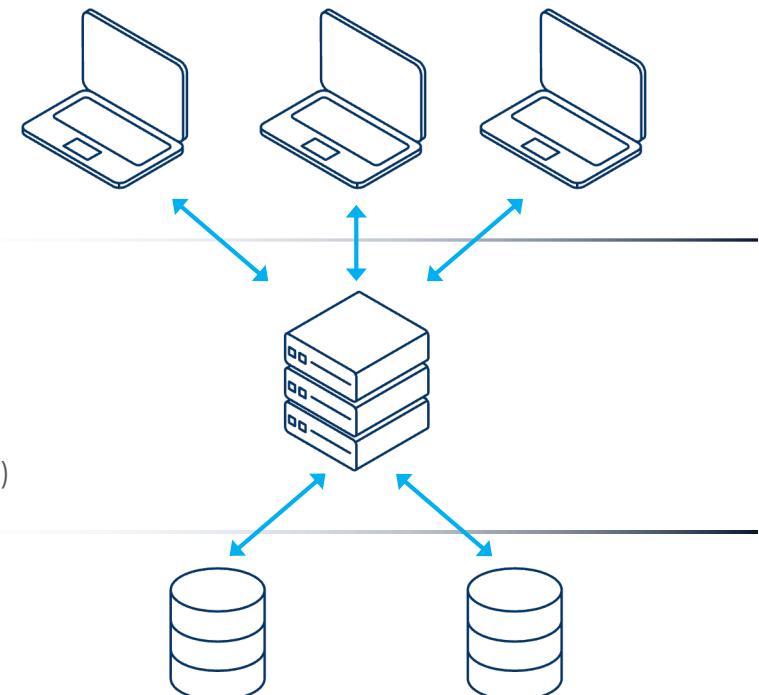
- | Lehet Vékony- vagy Vastagkliens

Alkalmazásszerver

- | Magát az alkalmazást futtató és ezáltal üzleti logikát kiszolgáló szerver(ek)

Adatbázisszerver

- | Adatok tárolását és felolvashatóságát biztosító szerver pl. MySQL



LÉTREHOZÁSI MINTÁK

SINGLETON

PROTOTYPE

FACTORY METHOD

ABSTRACT FACTORY

LÉTREHOZÁSI TERVEZÉSI MINTÁK

Gyártunk objektumokat!

- ─ Feladatuk, hogy megszűntessék a függőséget, melyet sok-sok „new” példány okoz.
- ─ Ahelyett, hogy mi magunk legyártunk manuálisan egy példányt (pl. new Kutya())
- ─ Egy létrehozási folyamatra bízzuk, hogy azt meghívva minden az aktuális követelményeknek megfelelő példányt kapjunk. (pl. kutyaGenerator.createKutya())



Az órán vizsgált létrehozási minták:

- ─ Singleton – Egyke
- ─ Prototype – Prototípus
- ─ Factory Method – Gyártómetódus
- ─ Abstract Factory – Absztrakt gyár

SINGLETON

Az Egyke

| Gyakran kell úgy megírnunk egy osztályt, hogy abból csak egyetlen példány keletkezhessen.

Kérdés: Milyen eseteket ismer, ahol Singletonra van szükség?

Nincs Konstruktur, van Osztályszintű Metódus

| Az osztályból a konstruktőrrel készíthetünk példányt
| A konstruktur metódusait példányosításkor hívjuk meg
| Ha van publikus konstruktur, akkor akárhány példányt készíthetünk
| Ha nincs konstruktur... **Akkor?**

Osztályszintű metódus

| Akkor is hívhatók, ha nincs példány.
| A Singlentonnak 1 getInstance metódusa van, ami mindenkinél ugyanazt a példányt adja vissza
| Ezt a példányt is létre kell hozni, de az osztály részét képező metódus hívja csak a privát konstruktort

THREAD-SAFETY

Szálbiztos

- | A Thread-safety lehetőséget biztosít megosztott erőforrások vagy adatstruktúrák egyidejű elérésére és módosítására
- | A Thread-safety megelőzi, hogy „versengés” alakuljon ki, károsodjon az adat vagy szinkronizálási hibákba fussunk

Thread-safe és Singleton

- | Gyakran összeér a két fogalom, sőt van Thread-safe Singleton
- | De alapvetően más funkciót látnak el

Thread-safe Singleton

- | Egy osztálynak csak egy példánya van
- | És globálisan hozzáférhető pontként szolgál több szál számára akár egyidejűleg is

PROTOTYPE

A Prototípus létrehozási minta

- █ Egy prototípust használ
- █ A prototípust klónozva gyárt objektumokat
- █ A klónoknak saját memória címmel rendelkeznek, így a klón megváltoztatása nem hat ki az eredeti prototípusra
- █ A klónozás: A klónozottal teljesen megegyező objektumot hozunk létre, a belső állapota is ugyanaz
- █ A klón részben, vagy teljesen független a klónozott objektumtól
- █ A prototípus létrehozási minta klónozási formája: Deep copy

Deep copy

- █ Nem csak az objektumból, hanem a benne foglalt objektumokról is másolatot hozunk létre
- █ A prototípusban foglalt objektumok teljesen klónozásra kerülnek, nem referenciaként jelennek meg
- █ Teljes függetlenség a klónozott objektumtól

Shallow copy

- █ Az eredeti objektum referencia típusú mezőit csak másoljuk, azok ugyanoda mutatnak
- █ Csak részben független a klónozott objektumtól

FACTORY METHOD

Gyártómetódus

- Sok-sok new utasítás kiadása helyett készítsünk gyárat!
- Create-el kezdjük: pl.: petGenerator.createKutya(); petGenerator.createMacska()
- A gyártómetódus a nevében leírt terméket adja vissza pl: Kutya, Macska

Kérdés: Miért jobb ez, mint egy new Kutya() vagy new Macska()?

Kérdés: Milyen OCP megfelelősséget láthatunk itt?

Gyártás

- Az ősben lévő gyártómetódus írja le a gyártás algoritmusát
- A gyermek osztály szabja meg, hogy pontosan mit kell gyártani

Algoritmus lépések

- A gyártás közös lépsei: Konkrét metódusok az ősben
- A gyártás kötelező változó lépései: Az ősben absztrakt metódusok, a gyermek írja felül a termék konstruktornak meghívásával
- A gyártás opcionális lépései: Hook metódusok az ősben, ezeket felülírhatjuk

Kérdés: Mit tanultunk a Hook-rol?

Példa: MS Office felületén

ABSTRACT FACTORY

Absztrakt és gyár?

- Olyan objektumok létrehozására jó, amelyek képesek egymással interakcióba lépni, együttműködni
- Több létrehozásra alaklamos (create method) metódust tartalmaznak
- Egyszerre több dolgot gyártunk és azoknak interoperábilisnek kell lenniük egymással

KÖSZÖNÖM A FIGYELMET!

6. óra – TERVEZÉSI MINTÁK