*2.3.1. Model-View-Controller (MVC) tervezési minta [16]*

A széles körben elterjedt MVC (magyarul: modell-nézet-vezérlés) tervezési mintát implementálja az aálkalmazásom, teljesen elválasztva egymástól a megjelenítést, az adatokat és működést. A *model* réteg jelképezi az adatokat és az azok közti logikát, a *view* (nézet) a felhasználói felületet és kimenteket (HTML, XML, JSON) foglalja magában, míg a *controller* (vezérlő) e két réteg közti kommunikációt vezérli. Például ha az alkalmazásunkat egy böngészőben és egy hordozható eszközön is futtatni szeretnénk, akkor csak új nézetekre van szükségünk.

A mintát Trygve Reenskaug írta le először 1979-ben, miután a Smalltalkon dolgozott a Xerox kutatói laborjában. Az eredeti megvalósítás részletesen a nagy hatású *Applications Programming in Smalltalk-80: How to use Model-View-Controller* [17] című tanulmányban olvasható.

Gyakori egy alkalmazás több rétegre való felbontása: megjelenítés (felhasználói felület), tartománylogika és adatelérés. Az MVC-ben a megjelenítés tovább bomlik nézetre és vezérlőre. Az MVC sokkal inkább meghatározza egy alkalmazás szerkezetét, mint az egy programtervezési mintára jellemző.

Mivel a webes alkalmazások esetében a kód pont erre a három részre osztható, ezért nagyon elterjedt ez a minta közöttük. A Yii esetében (és a legtöbb keretrendszernél is így van) az MVC minta meghatározza a rendszer mappaszerkezetét is. Ez a szeparáltság eleinte megszokást igényel attól a fejlesztőtől, aki korábban nem használta ezt a mintát, de később nagy gyakorlati haszna van: egységes lesz tőle az alkalmazás, valamint hibakeresésnél vagy új funkció írásánál egyértelműen tudni fogjuk, hogy a kód mely részéhez kell nyúlnunk.

„Habár az MVC-nek sok értelmezése létezik, a vezérlés menete általánosságban a következőképp működik:

1. A felhasználó valamilyen hatást gyakorol a felhasználói felületre (pl. megnyom egy gombot).

2. A vezérlő átveszi a bejövő eseményt a felhasználói felülettől, gyakran egy bejegyzett eseménykezelő vagy visszahívás útján.

3. A vezérlő kapcsolatot teremt a modellel, esetleg frissíti azt a felhasználó tevékenységének megfelelő módon (pl. a vezérlő frissíti a felhasználó kosarát). Az összetett vezérlőket gyakran alakítják ki az utasítás mintának megfelelően, a műveletek egységbezárásáért és a bővítés egyszerűsítéséért.

4. A nézet (közvetve) a modell alapján megfelelő felhasználói felületet hoz létre (pl. a nézet hozza létre a kosár tartalmát felsoroló képernyőt). A nézet a modellből nyeri az adatait. A modellnek nincs közvetlen tudomása a nézetről.

5. A felhasználói felület újabb eseményre vár, mely az elejéről kezdi a kört.” [16]

**4. ábra** Az MVC tervezési minta egy általános reprezentációja [16]



**Modell: „**Az alkalmazás által kezelt információk tartomány-specifikus ábrázolása. A tartománylogika jelentést ad a puszta adatnak.” [16]

Sok alkalmazás használ állandó tároló eljárásokat (pl. adatbázis) adatok tárolásához. Az MVC nem említi külön az adatelérési réteget, mert ezt beleérti a modellbe.

**View: „**Megjeleníti a modellt egy megfelelő alakban, mely alkalmas a felhasználói interakcióra, jellemzően egy felhasználói felületi elem képében, ilyenek például az űrlapok, menük, táblázatok stb. Különböző célokra különböző nézetek létezhetnek ugyanahhoz a modellhez.” [16] A Yii esetében a view-k általában egyszerű PHP scriptek, ahol a megjelenítési logikát kell megírnunk. A view neve megegyezik a PHP script fájl nevével, melyet a protected/views/ControllerID mappában kell elhelyezni, és a nézet megjelenítéséhez a render metódust kell hívni (erről később részletesebben is szót ejtek). A view script-ben a $this paranccsal a hívó Controller-t érjük el, további változókat a render metódusban adhatunk át, a következő módon: 16

$this->render('edit', array(

'var1'=>$value1,

'var2'=>$value2,

));

A fejlesztéskor egyedül itt találkozhatunk a HTML nyelvvel. Alapértelmezésként használhatjuk a Yii által felkínált sablonozó rendszert, de akár más ilyen eszközt is használhatunk. A szakdolgozatomban elkészített alkalmazás adminisztrátor moduljában megjelenítésnek az ExtJS-t használom.

A view-k mellett kell megemlítenem a *layout*-okat is, amik tulajdonképpen speciális view scriptek, és a több oldalon ismétlődő elemeket foglalják magukban. A gyakorlatban ez azt jelenti, hogy egy alap layout-ot hozunk létre, amiben deklaráljuk a *doctype*-ot, behúzzuk az alap CSS-eket, Javascript-eket, és a layout-on belül jelenítjük majd meg a különböző view-kat. Természetesen több layout-unk is lehet.

**Controller:** Az egész mintát a *controller* köti össze egy nagy egységgé. Az eseményeket, jellemzően felhasználói műveleteket dolgozza fel és válaszol rájuk, illetve a modellben történő változásokat is kiválthat. Yii esetében (és sok más keretrendszernél is) a controller-ek osztályok formájában jelennek meg és ezen osztályok metódusai az ún. action-ök. Az 5. ábra mutatja be, hogy valósul meg az MVC minta Yii esetében.

*2.3.2. Front Controller tervezési minta*

A Yii Framework a Front Controller mintát is implementálja, amely a kérelmek kezeléséért felelős: a felhasználótól érkező kérelmet feldolgozza, mely alapján meghatározza a megfelelő controller osztályt, amelynek átirányítja a kérést további feldolgozásra. Ahhoz, hogy részletesebben megértsük a Front Controller mintát, nézzük meg, mi történik a HTTP kérésekkel a *routing* és *dispatching* során [19]. A 6. ábra részletesen bemutatja ennek megvalósulását.

****

**6. ábra** Kérelem kezelése Yii Frameworkben [19]

Például, a felhasználó a következő címet írja be a böngészőjébe: http://www.example.com/index.php?r=post/show&id=1, akkor a fenti ábra lépései szerint a következők történnek:

**1.** A web szerver kezeli a kérelmet, futtatja az index.php belépő scriptet

**2.** A belépő script létrehozza a FC *singleton* (egyke) osztályt

**3.** A FC kinyeri a kérelemből a részletes információkat a *CHttpRequest* komponens segítségével

**4.** A *CUrlManager* komponens segítségével meghatározza a megfelelő controller-t és az action-t. A példában a post a PostController osztályra utal, a show a

18

showAction nevű metódusára, melyet egyértelműen meghatároz a controller osztály

**5.** Az FC példányosítja a kért controller osztályt, létrehozza és futtatja a controller-hez tartozó szűrőket (pl. hozzáférés szabályozás) és ha ezek engedik, futtatja az action-t

**6.** Az action kiolvassa az 1-es azonosítójú Post modellt az adatbázisból

**7.** Az action összeállítja a show nevű nézetet a lekért Post modell alapján

**8.** A view megjeleníti a Post modell attribútumait

**9.** A view futtathat *widget*-eket (komplex, de önálló, újrahasznosítható nézetelemek)

**10.** A view összeállítja az eredményt a layout-ba ágyazva

**11.** Az action befejezi a view készítését, és megjeleníti azt a felhasználónak