Szoftverarchitektúrák

Webalkalmazás architektúrák

Miről lesz szó?

- Web alkalmazások
  - Miért fontosak?
  - Miért speciálisak?
  - Hogyan működnek?
- Alapvető felépítés
  - Szerveroldal
  - Kommunikáció
  - Kliensoldal

Miről lesz szó?

- Szerveroldal
  - Architektúra típusok:
    - rétegzett architektúra
    - MVC
    - Pipes and Filters
  - Átszövő vonatkozások (Crosscutting concerns)
  - Függőségek kezelése
  - Tesztelés
- Kommunikáció
  - RESTful
    - vs SOAP
- Kliensoldal (böngésző)
  - Separation of Concerns: HTML + JavaScript + CSS
  - Single-Page Application
  - MVC/MVP/MVVM

Bevezetés

Miről lesz szó?

- **Web alkalmazások**
  - **Miért fontosak?**
  - **Miért speciálisak?**
  - **Hogyan működnek?**
- Alapvető felépítés
  - Szerveroldal
  - Kommunikáció
  - Kliensoldal

Miért speciálisak?

- Nem kell telepíteni – elég a böngésző
  - Nincs upgrade
  - Platformfüggetlenség
- Kevés kliensoldali erőforrás használat (pl. diszk)
- **Elosztott architektúra**
- **Hálózatfüggőség**
- Nehezebb fejlesztés

Miért speciálisak?

- Alapvető működés:
  - Kliens-szerver architektúra
    - Webdokumentumok elérése
  - **Kliensoldal:**
    - **Böngészőben**
    - **Vékonykliens**
      - **HTML + Javascript (dinamikus tartalom) + CSS**
      - **Állapotmentesség**
  - HTTP
  - **Szerveroldal**
    - **Dokumentum előállítás**
    - **Session kezelés**
    - **Üzleti logika**

Szerveroldal

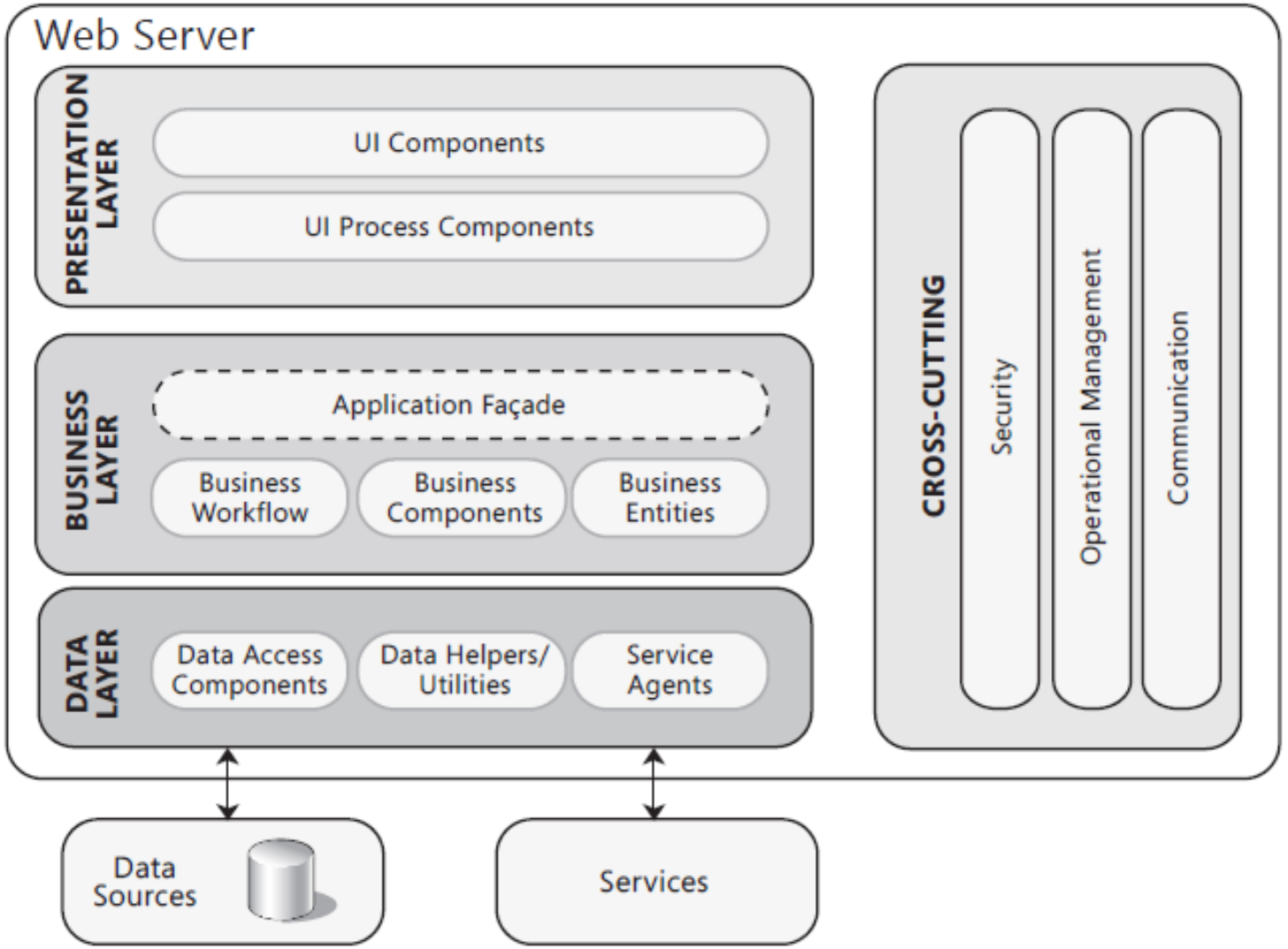
Miről lesz szó?

- Web alkalmazások
  - Miért fontosak?
  - Miért speciálisak?
  - Hogyan működnek?
- Alapvető felépítés
  - **Szerveroldal**
  - Kommunikáció
  - Kliensoldal

Szempontok

- Célok: biztonság, teljesítmény, komplexitás csökkentése
- Feladatok szétválasztása újrafelhasználható komponensekre
- Lazán csatolt komponensek (Facade minta alkalmazása)
- Pooling resources
- **Rétegzett architektúra**
- Átszövő vonatkozások (Crosscutting concerns)
  - Cache-elés
  - Felhasználó hitelesítése
  - Kivétel kezelés
- Kommunikáció felügyelete
  - Érzékeny adatok titkosítása
  - Adatok validálása minden réteg között
- Navigáció

Szerveroldal



Rétegelt architektúra

- Megjelenítési réteg (presentation layer)
  - Elosztott (szerver és kliens oldal)
  - UI komponensek szétválasztása
    - Újrafelhasználhatóság
  - Kliens oldali validáció **ismétlése** a szerveren

Rétegelt architektúra

- Üzleti logika (business logic layer)
  - Több lazán csatolt komponens
    - Adattovábbítás („üzeleti objektumok”)
  - Állapotmentes megvalósítás?
    - Teljesítménynövelés
    - Üzenet alapú interfész
    - Kritikus műveletek tranzakciókba foglalása

Rétegelt architektúra

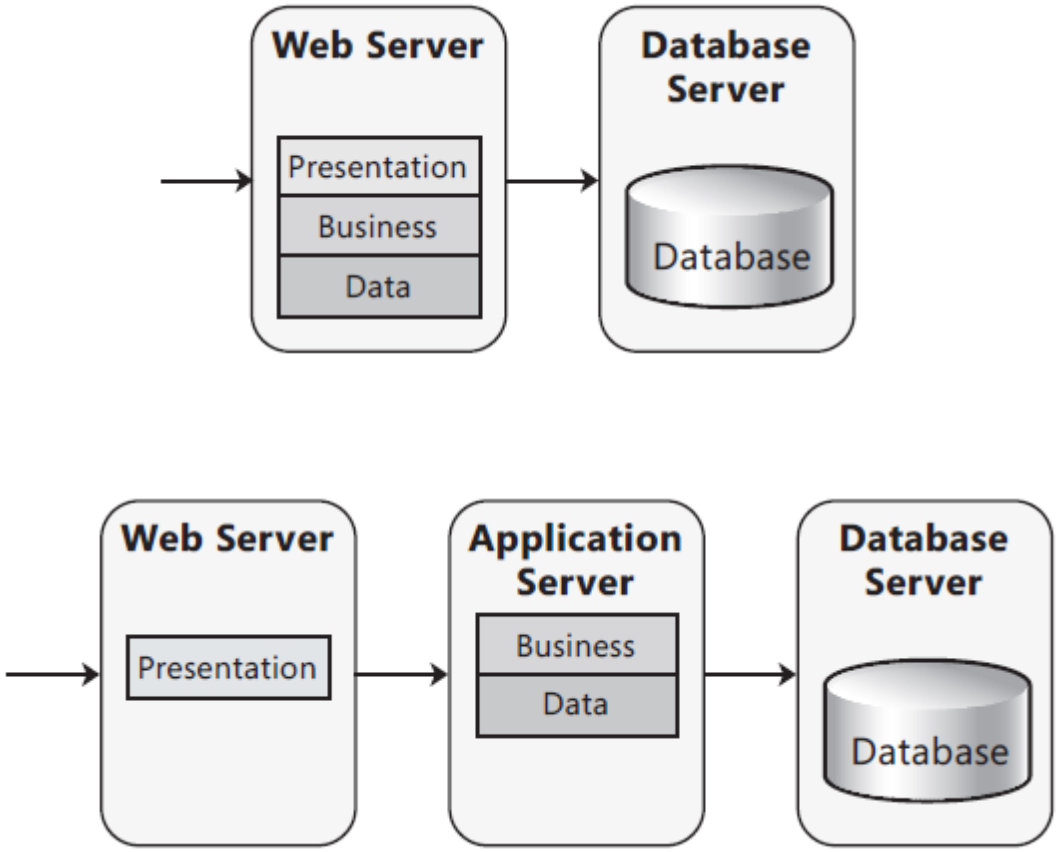
- Adatelérési réteg
  - Erőforrások kihasználása
    - Connection pooling
  - Batch operations
    - Kevesebbszer kelljen pl. adatbázishoz fordulni
  - Bonyolultabb exception kezelés

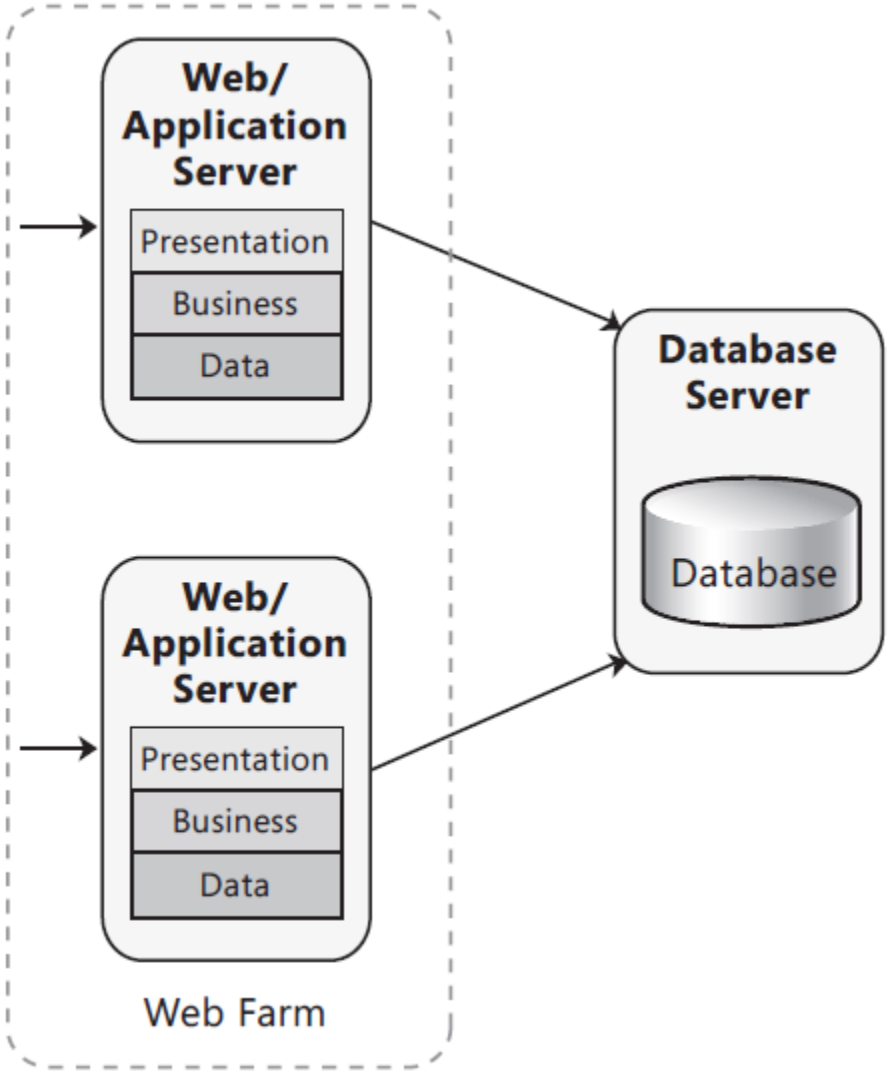
Rétegelt architektúra +1

- Szolgáltatás réteg (service layer)
  - Külön réteg, ha:
    - az üzleti réteget távolról elérhetővé akarjuk tenni, mert fizikailag másik gépre akartuk tenni, vagy
    - webserviceként publikussá akarjuk tenni.
  - Nem egy adott klienshez fejlesztünk, hanem önmagában használható kell legyen ez a réteg
  - Metódusok legyen tömörek, hogy ne kelljen sokszor fordulni a DBhez
  - Kommunikációs protokoll legyen általános

Rétegelt architektúra szempontok

- Telepítés





Crosscutting concerns

- Egy program olyan aspektusai, amelyek tipikusan sok komponenst érintenek
  - Nem lehet szétválasztani ezeket a program többi részétől
- Pl.
  - Authentikáció
  - Loggolás
  - Cache-elés
- → sok belső **függőség**

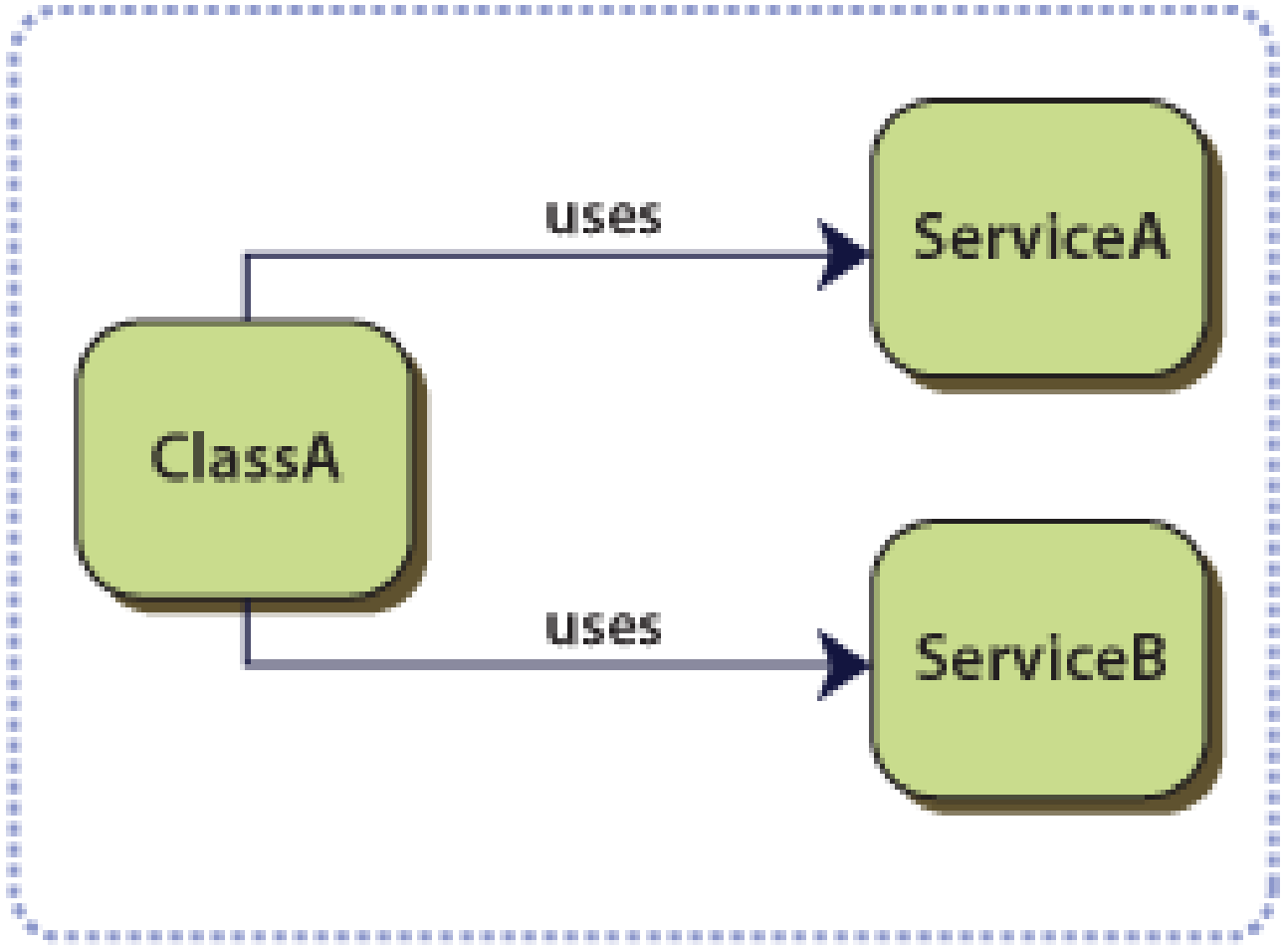
Probléma a függőségekkel

- Lazán csatolt komponensek (helyi komponensek, vagy szolgáltatások) egymásra hivatkozása
  - A függőségek változtatásához újra kell fordítani
  - Fordítási időben elérhetők kell legyenek a függőségek
  - Nehéz külön tesztelni
  - Ismételt kód a szoglátatások betöltésére
- Célok:
  - szétválasztani az osztályokat és a függőeségeiket
  - olyan osztályokat akarunk hazsnálni, amiknek az intefészét igen, de konkrét implementációját nem ismerjük fordítási időben
  - külön akarunk tesztelni
  - elválasztani a függőségek felderítését és betöltésének kódját, mert ez nem fontos a logika szempontjából.

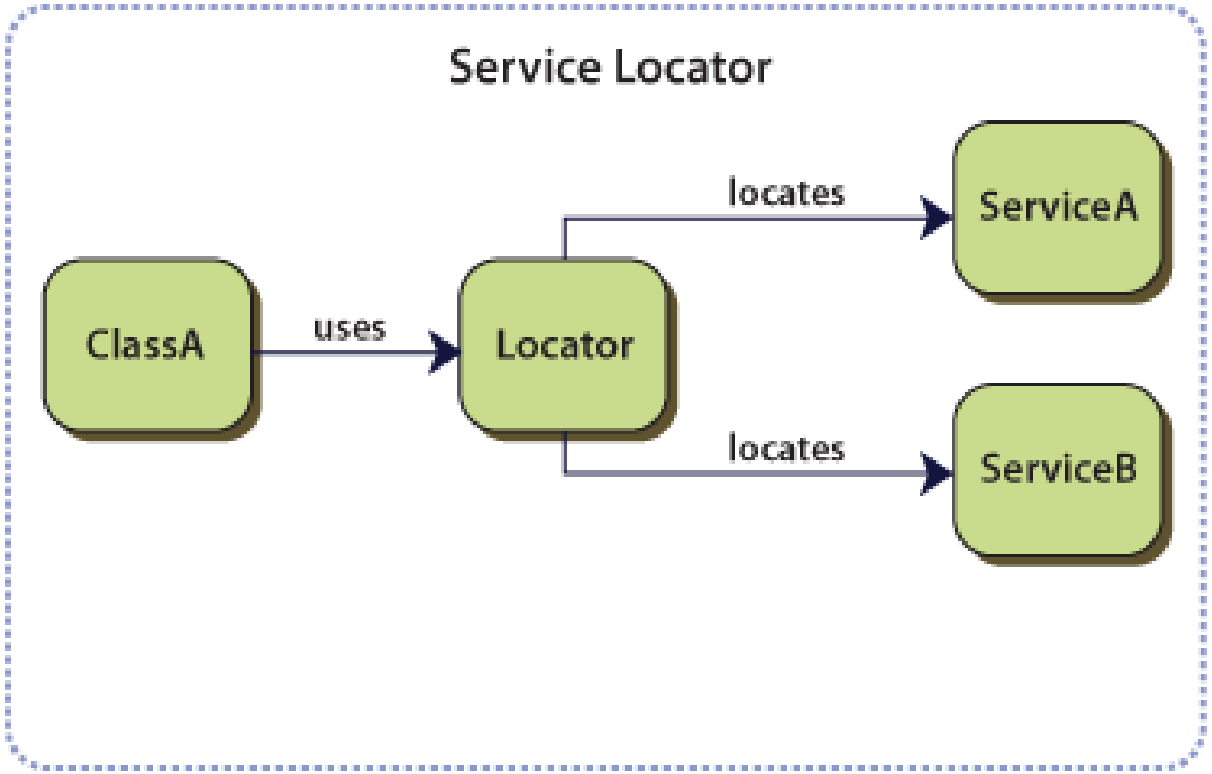
Függőségek kezelése

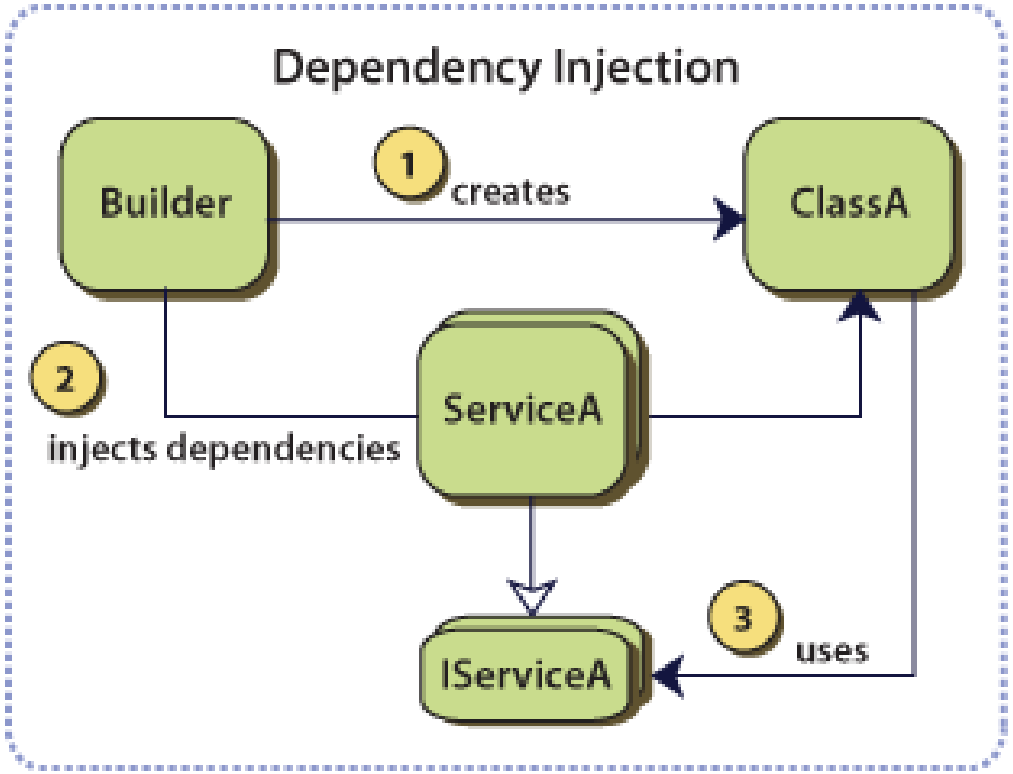
- Separation of concerns
  - Pl. HTML + JavaScript + CSS
- Inversion of Control
  - Dependency injection
  - Service Locator minta
- Programozási paradigmák
  - Aspektus Orientált Programozás (AOP)

Függőségek



Kétféle konkrét megoldás





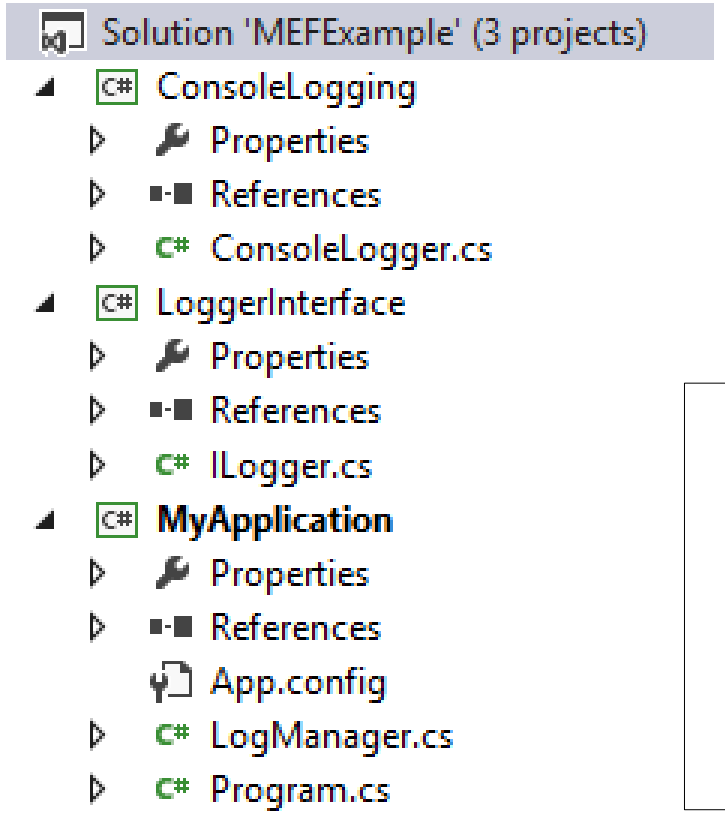
Dependency Injection

- A szolgáltatás, vagy komponens konkrét implementációjának betöltését egy külön objektum végzi
- 3 féle „stílus”:
  - Constructor injection
    - A hivatkozott ojecktumot a konstruktorban kapjuk meg
  - Setter/Property injection
    - A hivatkozott objektum beállítására setter függvényt/property készítünk
  - Interface injection
    - A beállítás egy interfészen keresztül történik

Microsoft Extensibility Framework

- IoC konténer alapú
- Dependency Injection megvalósítás
- Konfiguráció nélkül lehessen az alkalmazásunkat kiterjeszteni
- Interfész alapú függőségkezelés
  - Milyen interfész implementációra van szükség?
  - Milyen interfészt biztosítunk?

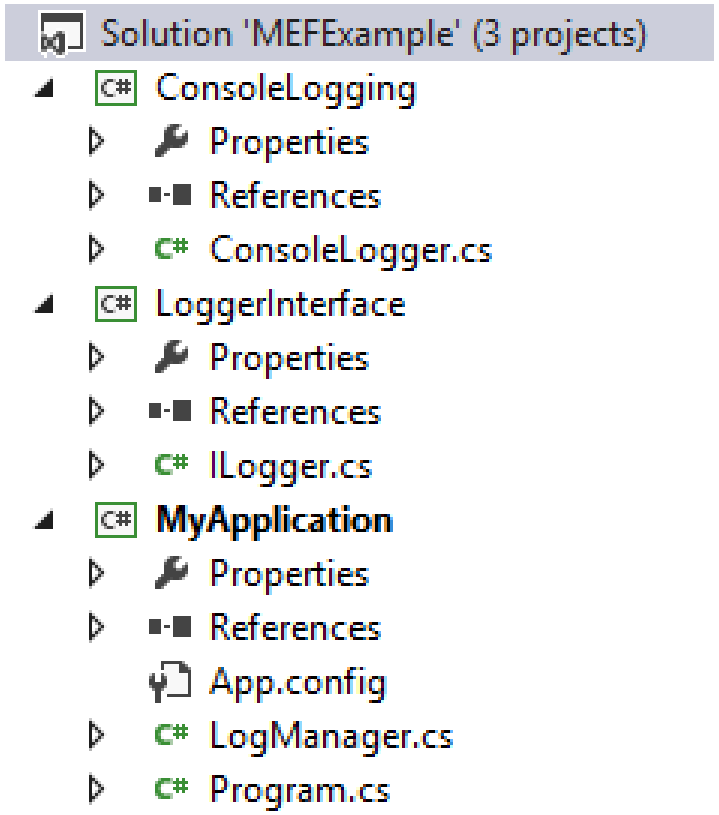
MEF – Példa alkalmazás




```
public interface ILogger
    {
        void Log(string textToLog);
    }
```

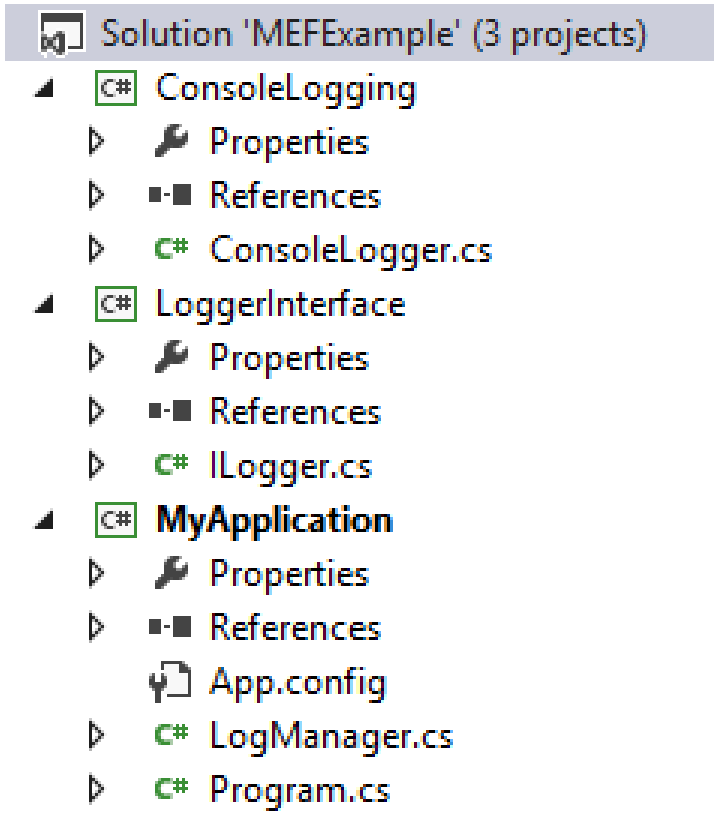
```
public class ConsoleLogger : ILogger
    {
        public void Log(string textToLog)
        {
            Console.WriteLine(textToLog);
        }
    }
```

MEF – Példa alkalmazás




```
public class LogManager
    {
        private static LogManager _Instance = null;
        public static LogManager Instance
        {
            get { return _Instance ??
                  (_Instance = new LogManager()); }
        }

        public void LogText(string textToLog)
        {
            //LOG
        }
    }
```

MEF – Példa alkalmazás




```
class Program
    {
        static void Main(string[] args)
        {
            //LOG
            //LogManager.LogText(…);
        }
    }
```

Probléma: a MyApplication-nek ismernie kell az implementációs osztályt

MEF működése

- System.ComponentModel.Composition assembly
- IoC konténer: ComponentContainer
  - Típusok betöltése és felismerése futásidőben
  - Hivatkozások kérése
  - Hivatkozások automatikus kitöltése
- Implementáció osztályok publikálják az intefészüket:
  - Export
- Függősek interfészét hivatkozzuk
  - Import

MEF – Export

A publikált intefész megjelölése

```
[Export(typeof(ILogger))]
public class ConsoleLogger : ILogger
    {
        public void Log(string textToLog)
        {
            Console.WriteLine(textToLog);
        }
    }
```

MEF – IoC konténe

```
public class LogManager
    {   //...
        private CompositionContainer container;

        private LogManager()
        {
            var catalogCollection = new AggregateCatalog();
            var catalog = new DirectoryCatalog(Environment.CurrentDirectory);
            catalogCollection.Catalogs.Add(catalog);

            container = new CompositionContainer(catalogCollection);

            try {
                this.container.ComposeParts(this);
            } catch (CompositionException compositionException) {
                //TODO
            }
        }
    }
```

Függőségek betöltésének koordinálása

Katalógus: ahol a függőségeket keressük

!

Függőségek betöltése futásidőben

MEF – Import

A hivatkozott függőség megjelölése

```
public class LogManager
    {
         //...

        [Import(typeof(ILogger))]
        public ILogger logger;

        public void LogText(string textToLog)
        {
            //LOG
            logger.Log(textToLog);
        }
    }
```

Sehol nem inicializáljuk a property-t kézzel!

MEF – Példa alkalmazás tesztelése

```
class Program
    {
        static void Main(string[] args)
        {
            LogManager.Instance.LogText("Program started");
            //...
            LogManager.Instance.LogText("Program finished");
        }
    }
```

MEF – Import

```
public class LogManager
    {
         //...

        [ImportMany(typeof (ILogger))]
        private IEnumerable<Lazy<ILogger>> loggers;

        public void LogText(string textToLog)
        {
            //LOG
            foreach (var logger in loggers)
                logger.Value.Log(textToLog);
        }
    }
```

Több példány, különböző implementációs osztállyal

Lazy: késleltett betöltés

Példányosítás az első lekérdezésnél

MEF - Metaadatok

- Metaadatok – Az egyes típusok, példányok testreszabására, azonosítására
  - Export metadata (kulcs érték pár)
  - Import:
    - Lazy<IMyInterface, IDictionary<string, object>>

    Vagy:
    - Lazy<IMyInterface, IMyMetaData

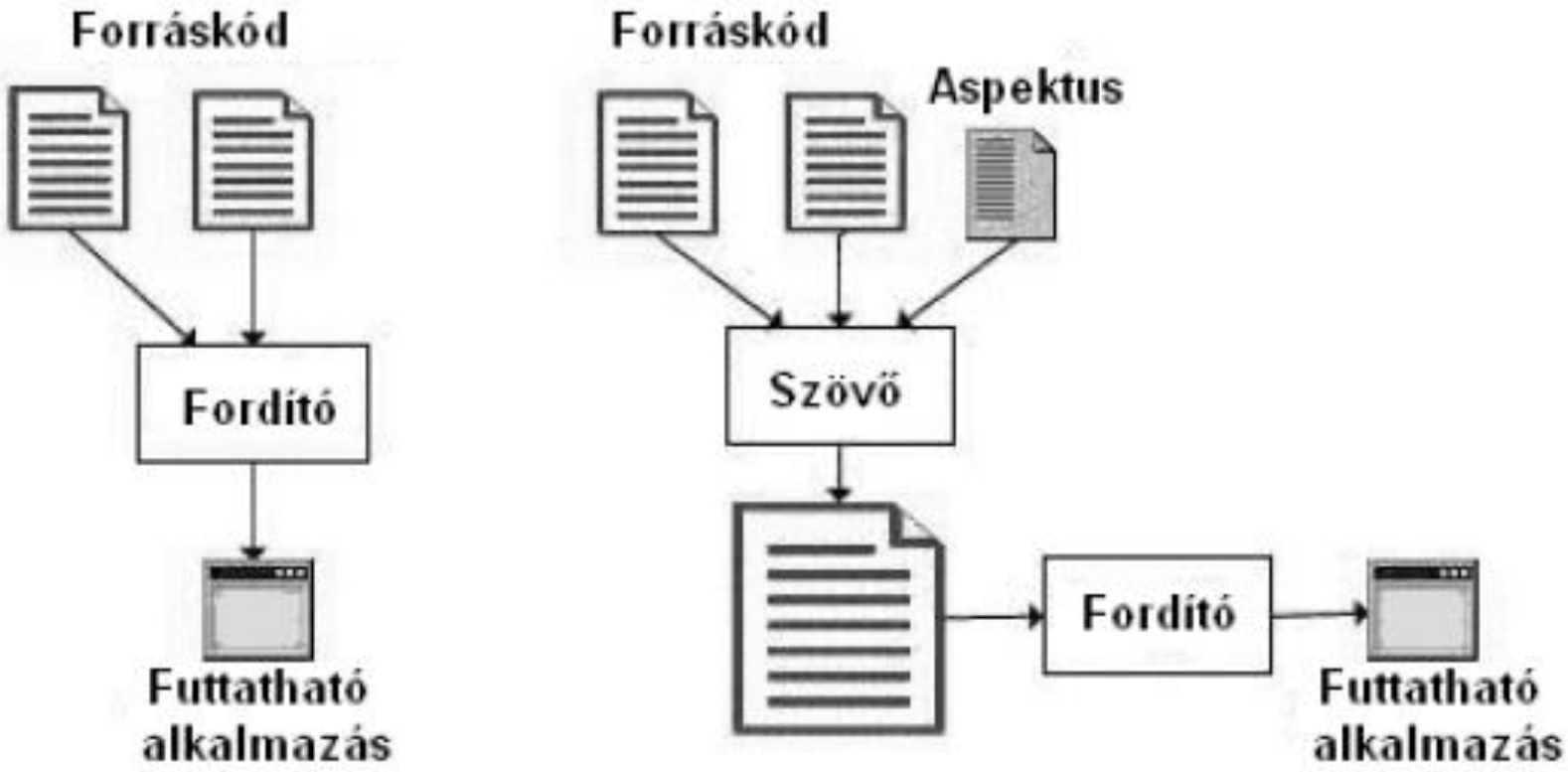
AOP

- Az objektum orientált paradigma továbbfejlesztése
- Főbb OO tulajdonságok:
  - Önálló entitások (objektumok): adatok és műveletek egységbezárása
  - A működést az objektumok közötti kommunikáció valósítja meg
  - → áttekinthető programok, biztonságosabb működés, kód újrafelhasználhatóság
- OO vs. más logikai rendező elvek:
  - Kódban elszórtan jelennek meg
  - → nehézkes karbantartás, nyomonkövetés
  - **Átszövő vonatkozások** (crosscutting concerns):
    - A program különböző egységein áthúzódó, de logikailag egybetartozó kódrészletek
    - Pl. naplózás

AOP

- a programkódokat aszerint osztja részekre, hogy azok milyen szempontból járulnak hozzá a program működéséhez
  - A szempontokat **aspektusok**ba tömörítjük
  - Az aspektusokat önállóan kódoljuk
  - Az aspektusszövő (**aspect weaver**) alkalmazás gondosodik a különböző kódrészletek egyesítéséről (futási, vagy fordítási időben)

AOP


Forráskód
Fordító
Futtatható alkalmazás
Forráskód
Aspektus
Szövő
Fordító
Futtatható alkalmazás

AOP

- **Csatlakozási pontok**:
  - Specifikálja, hogy egy átszövő aspektus adott kódrészlete hol jelenjen meg, hogy kerüljön meghívásra.
    - **Statikus**: a forráskód egy szövegéhez csatlakozik
    - **Dinamikus**: futási időben történő hozzárendelés

AOP vs. architektúra

- Miért fontos az AOP architekturális szempontból?
  - Az aspektusok átszövik az alkalmazás logikáját → megváltoztatják az architektúrát
  - Tervezési szinten megvalósítható az aspektus-orientált gondolkodás → modularitás
  - Modularitás:
    - A logikailag egy egységbe tartozó részek valóban egy fizikai egységbe tömörüljenek
    - Erős belső kohézió a modulon belül, lazább csatolás a modulok között
  - Kisebb redundancia

AOP

- Példák:
  - AspectJ,
  - HyperJ,
  - **PostSharp**

PostSharp

- .NET AOP kiterjesztés
- Kódrészletek fordításidejű transzformáció
  - Új kódrészletek beszúra
  - A régi kód kisebb átalakítása
- Fogalmak:
  - *Aspect*: egy kódrészlet transzformálását leíró objektum
    - A transzformáció eredményeként az erdeti kódból az aspektus metódusai is meghívásra kerülnek
  - *Advice*: az aspektus metódusai, amelyek meghívásra kerülnek
- Attribútum alapú megközelítés

PostSharp példa

```
[Serializable]
    public class LoggingAspect : OnMethodBoundaryAspect
    {
        public override void OnEntry(MethodExecutionArgs args)
        {
            Console.WriteLine("The {0} method has been entered.", args.Method.Name);
        }
    }
```

```
public class MyClass
    {
        [LoggingAspect]
        public string Greet(string name)
        {
            return "Hello " + name;
        }
    }
```

Egy lehetséges aspektus ősosztály, egy metódus transzformációját végrehajtók valósítanak

Aspektus alkalmazása

Egy Advice, ami meghívásra kerül a transzformált kódból

Transzformáland ó metódus

PostSharp

- `MethodExecutionArgs` paraméter
  - Argumentumok
  - Exception – az éppen dobott kivétel
  - FlowBehavior – Hogyan folytatódjon a végrehajtás (pl. Return, Continue)
  - Instance – az objektum, amin történik a végrehajtás
  - Method – Az aktuális metódus
  - ReturnValue – Az aktuális visszatérési érték

PostSharp példa

```
class Program
    {
        static void Main(string[] args)
        {
            var myClass = new MyClass();
            var ret = myClass.Greet("World");
            Console.WriteLine(ret);
        }
    }
```

```
The Greet method has been entered.
Hello World
```

PostSharp – Transzformált kód

```
public string Greet(string name)
{
    string text;
    MethodExecutionArgs args =
          new MethodExecutionArgs();
    //... args feltöltése
    MethodBase.LoggingAspect.OnEntry(args);
    try
    {
       text = "Hello " + name;
    }
    catch (Exception exception)
    {
    }
    finally
    {
       return text;
    }
}
```

Az eredeti függvény transzformáltja

Minden függvényhez létrejön egy Aspect objektum

Try – catch – finally beágyazás

PostSharp – további funkciók

- Kilépéskor meghívódó függvény
- Sikeres végrehajtás esetén meghívásra kerülő kód

```
public override void OnExit(MethodExecutionArgs args)
        {
            Console.WriteLine("The {0} method has exited", args.Method.Name);
            Console.WriteLine("The return value is : {0}", args.ReturnValue);
        }
```

```
public override void OnSuccess(MethodExecutionArgs args)
        {
            Console.WriteLine("The {0} method executed successfully.", args.Method.Name);
        }
```

PostSharp – további funkciók

```
public override void OnException(MethodExecutionArgs args)
{
    Console.WriteLine("An exception was thrown in {0}.", args.Method.Name);
    if (args.Exception.GetType() == typeof(DivideByZeroException))
    {
        //args.FlowBehavior = FlowBehavior.RethrowException;
        args.Exception = new DivideByZeroException("This was thrown from an aspect",
                                                   args.Exception);
        args.FlowBehavior = FlowBehavior.ThrowException;
    }
}
```

- Hiba esetén meghívásra kerülő függvény

PostSharp – további funkciók

- A MethodExecutionArgs további adatai

```
public override void OnEntry(MethodExecutionArgs args)
{
    var argValues = new StringBuilder();
    foreach (var argument in args.Arguments)
    {
       argValues.Append(argument.ToString()).Append(",");
    }
    Console.WriteLine("The {0} method was entered with the parameter values: {1}",
                      args.Method.Name, argValues.ToString());

     //args.FlowBehavior = FlowBehavior.Return;
}
```

PostSharp – további funkciók

- Állapotmegőrzés

```
[Serializable]
public class ExecutionDurationAspect : OnMethodBoundaryAspect
{
    public override void OnEntry(MethodExecutionArgs args)
    {
        args.MethodExecutionTag = Stopwatch.StartNew();
    }

    public override void OnExit(MethodExecutionArgs args)
    {
        var sw = (Stopwatch)args.MethodExecutionTag;
        sw.Stop();
        Console.WriteLine("{0} executed in {1} seconds", args.Method.Name,
                                                         sw.ElapsedMilliseconds / 1000);
    }
}
```

PostSharp – további askpektusok

- Aspektusok:
  - OnMethodBoundaryAspect
    - Try – catch – finally blokk
    - Advices: OnEntry, OnSuccess, OnException, OnExit
  - OnExceptionAspect
    - Try – catch blokk
    - OnException
  - MethodInterceptionAspect
    - OnInvoke
  - EventInterceptionAspect
    - OnAddHandler, OnRemoveHandler, OnInvokeHandler
  - LocationInterceptionAspect
    - OnGetValue, OnSetValue

Kommunikáció

Webszolgáltatás

- Webszolgáltatás
  - Elosztott rendszerek közötti kommunikációs protokoll
  - Hogyan (milyen formátumban) tudunk információt kérni egy szervertől?
- SOAP megközelítés
  - Simple Object Access Protocol
  - A webszolgáltatás funkciójához kapcsolódó szabványosított nyelvezet
- REST megközelítés
  - REpresentational State Transfer
  - Általános
  - Egyszerű
  - …Ahogyan a web működik

SOAP

- Szabványosított adatkommunikációs protokoll
- XML alapú üzenetek → nehézkes előállítás, de automatizálható
- Eredetileg régi, interneten nehezen használható technológiák (pl. DCOM, CORBA) lecserélésére
- Kiterjeszthető szabványos funkciókkal (pl. titkosítás)
- A szolgáltatás leírása (WSDL – Web Service Description Language) → XML előállítása automatizálható (proxy-k generálásával)
- Átviteli protokollok: **HTTP**, SMTP, vagy más…
- Alkalmazásfüggő nyelvezet

REST

- A világháló működésének egy absztrakciója
  - Egy architekturális stílus elosztott alkalmazásokhoz
    - Bonyolult kommunikációs protokollok helyett: egyszerű HTTP kérések
      - Minden CRUD műveletre
    - Erőforrások elérése és címzése URI alapján
  - Kommunikációs protokoll
    - Állapotmentes
    - Kliens-szerver
    - Cache-elést lehetővé tévő
    - Egyszerű alternatívája más kommunikációs protokolloknak pl. RPC, SOAP

REST

- REST megkötések
  1. Kliens-szerver felépítés
  2. Állapotmentesség
  3. Cache-elés lehetősége
  4. Egységes interfész (URI)
  5. (Rétegelt rendszer)
  6. [Code-on-demand] – opcionális
- Az architektúra céljai
  - Teljesítmény növelése, skálázhatóság
  - Egyszerűség, könnyű fejlesztés
- A kommunikáció tárgyai
  - Erőforrások – egyedi azonosítókkal (URI)
  - Metódusok (HTTP igék)
  - Média típusok (milyen típusú adat adunk vissza, pl. CSV, JSON, XML

REST vs SOAP

- Azok a web szolgáltatások, amik eleget tesznek a REST megkötéseinek: RESTful
  - A SOAP használata sok overhead-et jelenthet, pl. JavaScriptben
  - Nem kötődik XML-hez
  - URL gyakran elég a címzéshez
  - Csak HTTP felett
    - Kihasználja a HTTP eszközkészletet
- Jellemzők
  - Teljesítmény, skálázhatóság
  - Formátum
  - Kiegészítő szolgáltatások: hibakezelés, titkosítás, tranzakciókezelés
- Melyiket érdemes választani?

Kliensoldal

Kliensoldali programozás

- Separation of concerns
  - HTML dokumentum
  - JavaScript kód
  - Stílus: CSS

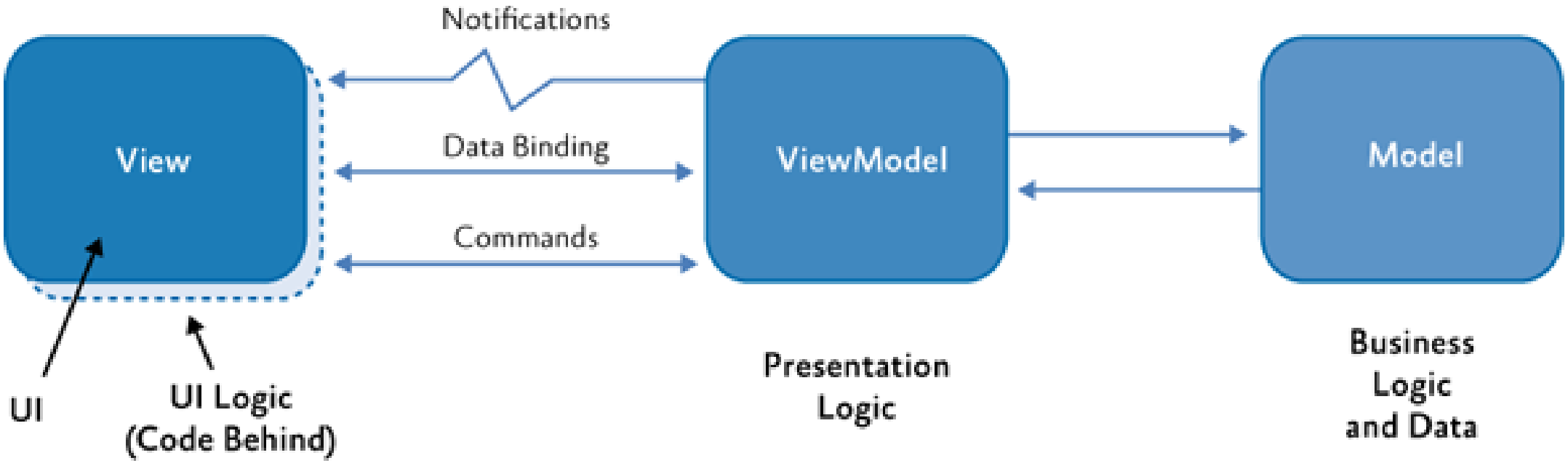
  → Sablon alapú megközelítések mind szerver, mind kliensoldalon
- Egyre összetettebb működés
  - REST → Code-on-demand
  - A JavaScript dinamikus nyelv
  - Nagy funkcionalitású JavaScript könyvtárak
  - A kliens oldali kód tervezést igényel
    - A hagyományos programozási környezetben megismert módszerek és minták megjelennek JavaScriptben

  → Jó ez?
- Megoldások
  - Statikus oldalak
  - AJAX hívások
  - SPA (Single Page Application)

MVVM

- Architekturális minta, MVC, MVP alapon
  - Cél: a megjelenítési, üzleti logika és a felhasználó felület szétválasztása
  - UI fejlesztés
  - Eredetileg: WPF-ben, Martin Fowler: Presentation Model
  - Eseményvezérelt programozási környezet
- Feladatok:
  - Adattárolás,
  - Üzleti logika - adatmanipuláció
  - Adatok megjelenítése (megjelenítési logika)
  - UI (vezérlők)
  - UI logika (vezérlőkön keresztül a felhasználói interakciók kezelése)

MVVM


Notifications
Data Binding
Commands
View
ViewModel
Model
UI
UI Logic
(Code Behind)
Presentation
Logic
Business
Logic
and Data

MVVM - View

- A felhasználó számára látható megjelenítés (UI)
- Hivatkozik a ViewModelre, mint adat kontextusra
  - A ViewModel adattagjait, műveleteit (command) kötjük hozzá
  - Testre szabhatja az adatkötést: validációval, konvertálással
  - Viselkedés testreszabása: animáció, tranzíciók
  - A túl bonyolult viselkedést lehet hagyományos kódban írni
- Az adatok formázása
- Aktív nézet: felhasználói interakciók kezelése
  - vs. passzív nézet: nincs tudomása a modellről, a controller kezeli
- Felhasználói események kezelése

MVVM - ViewModel

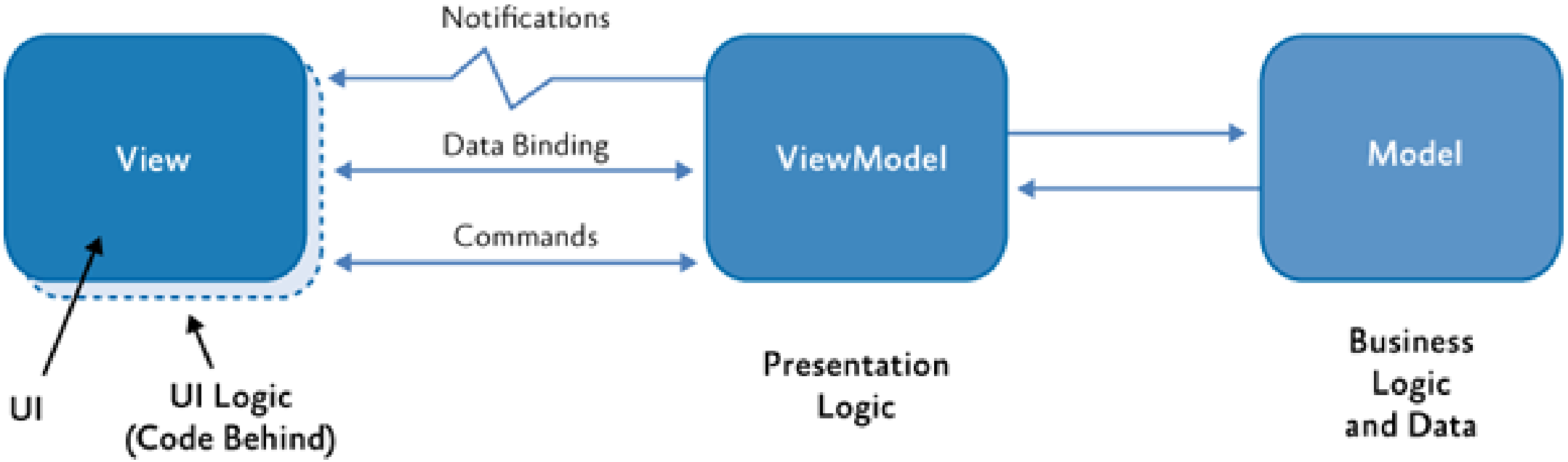
- Megjelenítési logika
- Adatok a nézetnek
- Nem tud a Viewról
- A View-t az adatkötésen keresztül, állapotváltozás értesítésen keresztül implicit módon változtatja
- A UI azokat a funkciókat nyújtja, amiket a ViewModel proeprty-ken és Commandokon keresztül definiál, a View ezeket csak rendereli
- Több modell osztályért is felelhet, de lehet 1-1 az összerendelés és akár ViewModel funkcionalitása (pl. adatkötés) lehet közvetlenül a modellben is
- Származtatott értékek definiálhat ahhoz, hogy a View megfelelően tudja megjeleníteni a modellt
  - Pl. összefűzhet két adatot
  - Bevezethet újabb logikai állapotokat, amik a UI kinézetét befolyásolhatják (pl. aszinkron hívás alatt)
- Művelet (command/action): olyan felhasználható által elérhető funkciók, amik a UI-on keresztül elérhetők a felhasználó számára
  - Nem fontos, hogy ehhez egy gomb lesz kint, vagy link, tehát a megjelenítés megint elválik az implementációtól.
- Önmagában tesztelhető a View-tól, modelltől függetlenül

MVVM – Model

- Szakterület-specifikus entitás (pl. User Account/name, email stb.)
- Információt tárolnak + üzleti logikát
  - Üzleti logika: az adatok kinyerése, menedzselése és annak biztosítása, hogy az üzleti szabályok, amik az adatok konzisztenciájára, validációjára vonatkoznak teljesüljenek
    - Perzisztálás
    - Cache
  - Újrafelhasználható entitás: nem tartalmaz Use-case, User specifikus adatot
- Nem kezelnek viselkedést
  - Kivéve: validáció
- Nem formázzák meg az információt a megjelenítéstől függően

Együttműködés

- Adatkötés
- Műveletek
- Validáció és hibakezelés



MVVM vs MVC

- Controller = ViewModel + Binder
- Binder – implicit elvárt a technológiától
  - Deklaratív adatkötés
  - Deklaratív műveletkötés

Knockout.js

- JavaScript MVVM keretrendszer
- Deklaratív adatkötés
  - Kétirányú: automatikus UI és model frissítés
- Származtatott értékek esetében a függőségek automatikus felderítése
- Sablonok használata

Knockout.js

A ViewModel egy tagváltozójához kötjük a belül található szöveget

```
<!-- VIEW -->
<p>First name: <strong data-bind="text: firstName"></strong></p>
<p>Last name: <strong data-bind="text: lastName"></strong></p>
```

```
// ViewModel + „model”
function AppViewModel() {
    this.firstName = "Bert";
    this.lastName = "Bertington";
}

// Activates knockout.js
ko.applyBindings(new AppViewModel());
```

ViewModel létrehozása

„Modell”: itt statikus adatok

MVVM rendszer indítása

First name: **Bert**
Last name: **Bertington**

Knockout.js – minden kód együtt

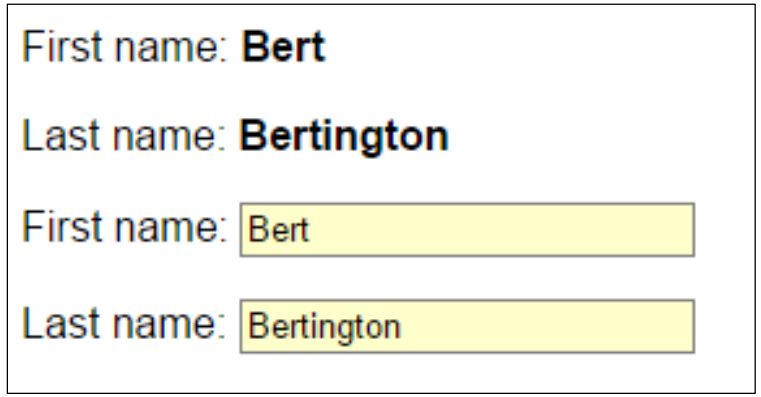
```
<html>
<head>
    <script type='text/javascript'
                src='http://ajax.aspnetcdn.com/ajax/knockout/knockout-3.0.0.js'></script>
</head>
<body>
    <h1>Knockout test</h1>
    <p>First name: <strong data-bind="text: firstName"></strong></p>
    <p>Last name: <strong data-bind="text: lastName"></strong></p>

    <script type='text/javascript'>
        function AppViewModel() {
            this.firstName = "Bert";
            this.lastName = "Bertington";
        }
        ko.applyBindings(new AppViewModel());
    </script>
</body>
</html>
```

Knockout.js

```
<p>First name: <strong data-bind="text: firstName"></strong></p>
<p>Last name: <strong data-bind="text: lastName"></strong></p>

<p>First name: <input data-bind="value: firstName" /></p>
<p>Last name: <input data-bind="value: lastName" /></p>
```

Kétirányú adatkötés

```
function AppViewModel() {
    this.firstName = ko.observable("Bert");
    this.lastName = ko.observable("Bertington");
}
ko.applyBindings(new AppViewModel());
```



Knockout.js

```
function AppViewModel() {
    this.firstName = ko.observable("Bert");
    this.lastName = ko.observable("Bertington");

    this.fullName = ko.computed(function() {
        return this.firstName() + " " + this.lastName();
    }, this);

    this.capitalizeLastName = function() {
        var currentVal = this.lastName();
        this.lastName(currentVal.toUpperCase());
    };
}

ko.applyBindings(new AppViewModel());
```

Változásértesítés az adat megváltozásáról

Származtatott adat (automatikus függőség követés)

Command: ahol elérhető a scope, vagyis az aktuális ViewModel

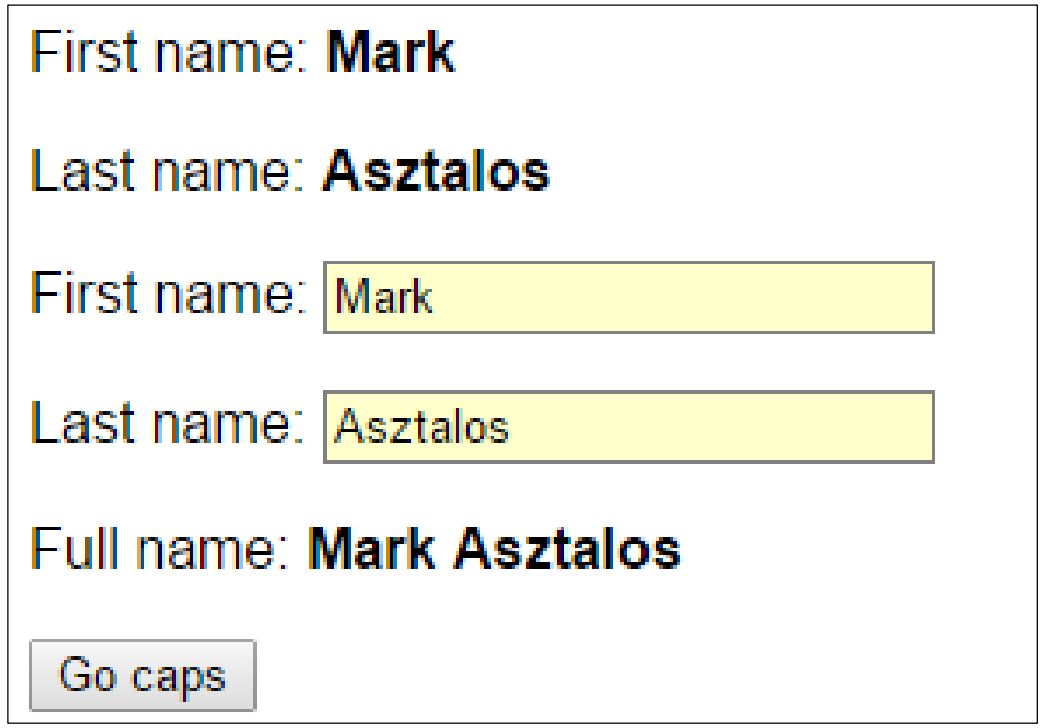
this.lastName = ko.observable(…) erdménye: függvényként érhető el a tárolt adat

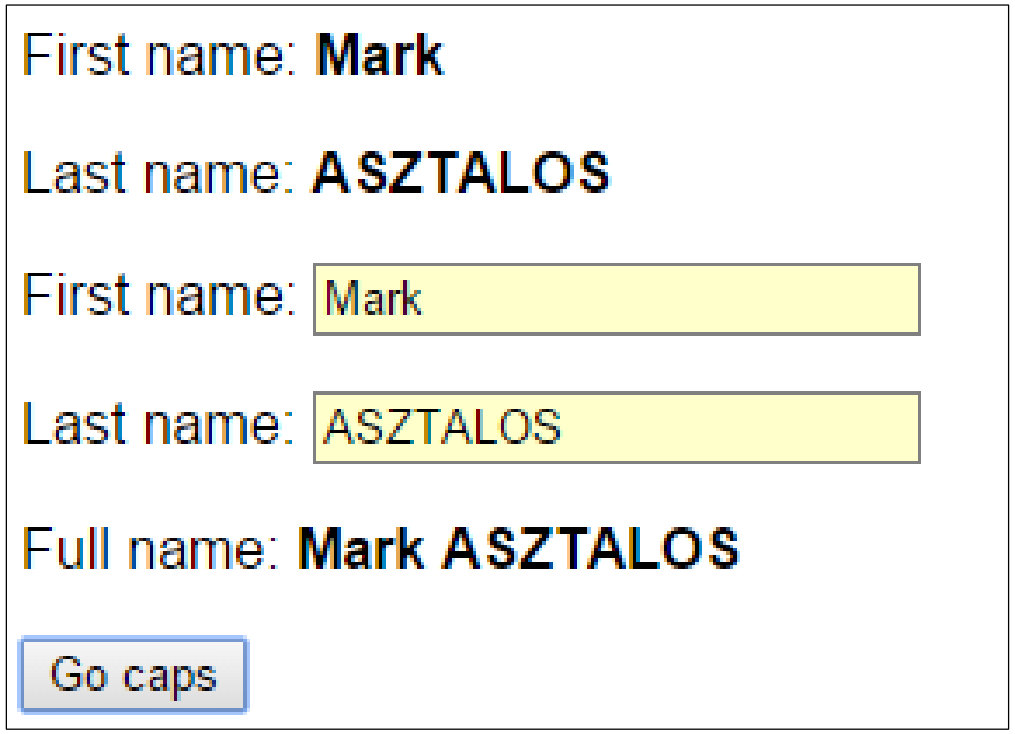
Knockout.js

```
<p>First name: <strong data-bind="text: firstName"></strong></p>
<p>Last name: <strong data-bind="text: lastName"></strong></p>

<p>First name: <input data-bind="value: firstName" /></p>
<p>Last name: <input data-bind="value: lastName" /></p>

<p>Full name: <strong data-bind="text: fullName"></strong></p>

<button data-bind="click: capitalizeLastName">Go caps</button>
```

Command kötése

Knockout.js


First name: Mark
Last name: Asztalos
First name: Mark
Last name: Asztalos
Full name: Mark Asztalos
Go caps


First name: Mark
Last name: ASZTALOS
First name: Mark
Last name: ASZTALOS
Full name: Mark ASZTALOS
Go caps

Knockout.js

- DEMO

Értékelés

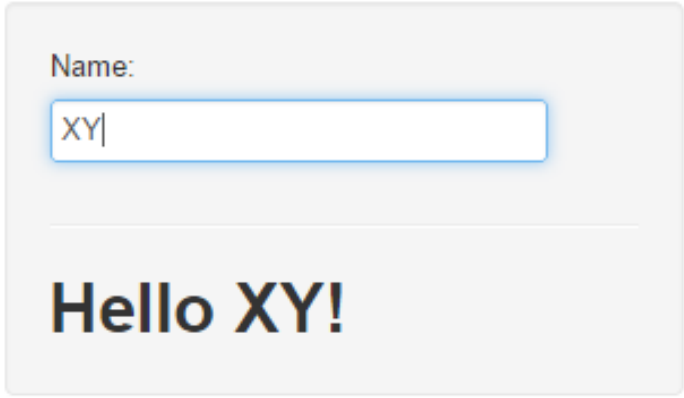
- Előnyök
  - Tesztelhetőség
  - Karbantarthatóság
  - Kód újrafelhasználás
  - Fejlesztők és UI designerek együttműködésének segítése

- Hátrányok
  - Egyszerű UI → overkill
  - Bonyolult UI → memória

AngularJS

- JavaScript MVVM/MVC keretrendszer
- Sablon alapú megközelítés
- Kétirányú adatkötés
- Vezérlők (controllers) írják le a DOM elemek viselkedését
- URL routing
- Parciális nézetek
- Kliens oldali validáció
- Egyszerűsített kommunikáció a szerverrel
- Direktívák – új HTML elemek
  - Újrafelhasználható komponensek
  - Komplexitás elrejteése
- Több AngularJS komponens beágyazása egy alkalmazásba
- Moduláris kialakítás
  - Kiterjeszthetőség
  - Dependency Injection
- Tesztelhetőség

AngularJS



Angular alkalmazás

JS könyvtár referálása

Kétirányú adatkötés

Egyirányú adatkötés (sablon alapú)

```
<!doctype html>
<html ng-app>
<head>
    <script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.3.2/angular.min.js"></script>
</head>
<body>
    <div>
        <label>Name:</label>
        <input type="text" ng-model="yourName" placeholder="Enter a name here">
        <hr>
        <h1>Hello {{yourName}}!</h1>
    </div>
</body>
```

AngularJS

```
<!doctype html>
<html lang="en" ng-app="phonecatApp">
<head>
  <meta charset="utf-8"><title>Google Phone Gallery</title>
  <link rel="stylesheet" href="bower_components/bootstrap/dist/css/bootstrap.css">
  <link rel="stylesheet" href="css/app.css">
  <script src="bower_components/angular/angular.js"></script>
  <script src="js/controllers.js"></script>
</head>
<body ng-controller="PhoneListCtrl">
  <ul>
    <li ng-repeat="phone in phones">
      <span>{{phone.name}}</span>
      <p>{{phone.snippet}}</p>
    </li>
  </ul>
</body>
</html>
```

Angular alkalmazás

Angular JS modell (scope) inicializálása

Adatkötés listához

Egyirányú adatkötés sablon alapon

AngularJS

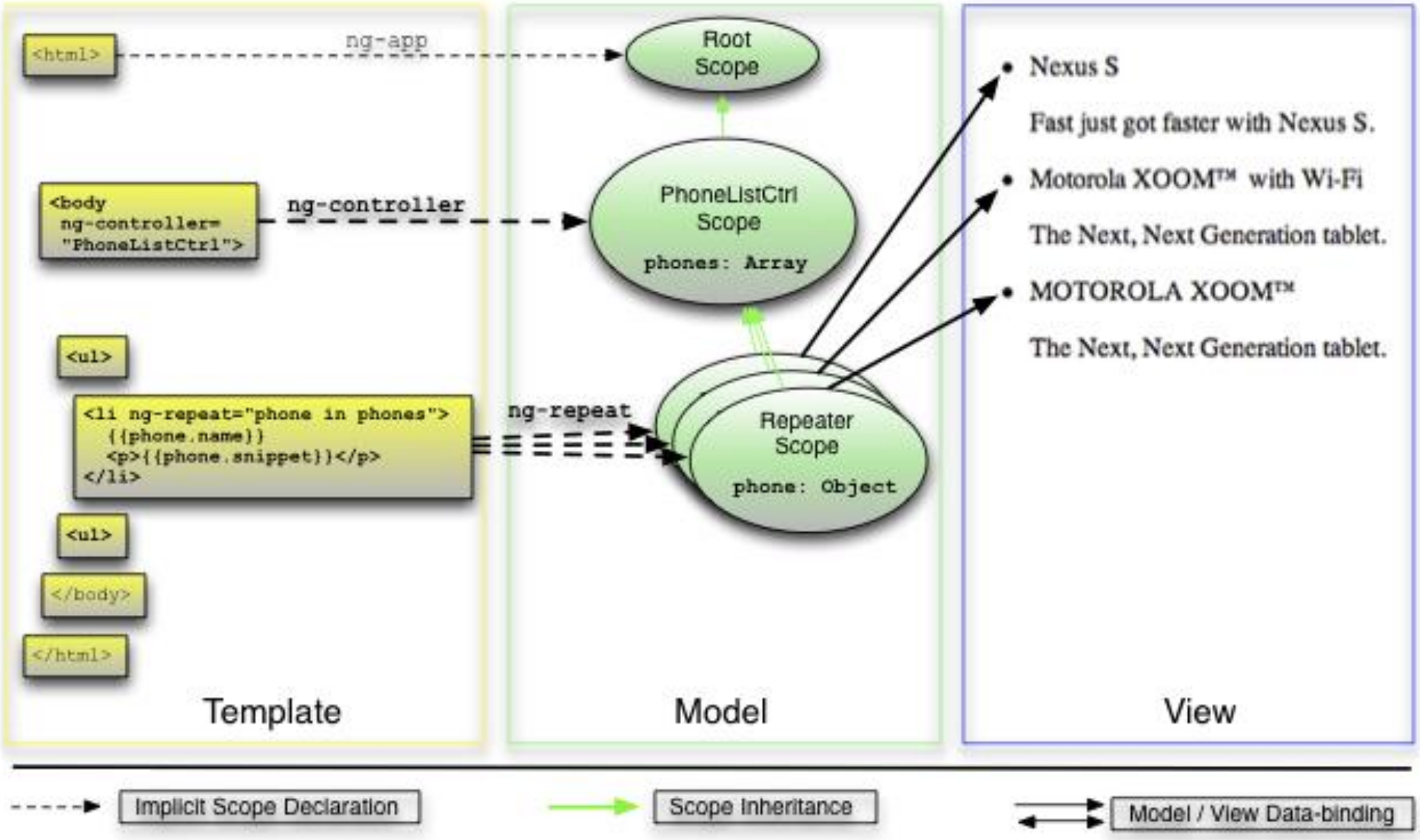
```
var phonecatApp = angular.module('phonecatApp', []);

phonecatApp.controller('PhoneListCtrl', function($scope) {
  $scope.phones = [
    {'name': 'Nexus S',
     'snippet': 'Fast just got faster with Nexus S.'},
    {'name': 'Motorola XOOM™ with Wi-Fi',
     'snippet': 'The Next, Next Generation tablet.'},
    {'name': 'MOTOROLA XOOM™',
     'snippet': 'The Next, Next Generation tablet.'}
  ];
});
```

Modul regisztrálása

Scope inicializálása

- Nexus S
  Fast just got faster with Nexus S.
- Motorola XOOM™ with Wi-Fi
  The Next, Next Generation tablet.
- MOTOROLA XOOM™
  The Next, Next Generation tablet.

AngularJS


<html>
ng-app
Root Scope
<body ng-controller= "PhoneListCtrl">
ng-controller
PhoneListCtrl Scope
phones: Array

<li ng-repeat="phone in phones">
{{phone.name}}
{{phone.snippet}}

ng-repeat
Repeater Scope
phone: Object

</body>
</html>
Nexus S
Fast just got faster with Nexus S.
Motorola XOOM™ with Wi-Fi
The Next, Next Generation tablet.
MOTOROLA XOOM™
The Next, Next Generation tablet.
Template
Model
View
Implicit Scope Declaration
Scope Inheritance
Model / View Data-binding

AngularJS

```
var phonecatApp = angular.module('phonecatApp', []);

phonecatApp.controller('PhoneListCtrl', ['$scope', '$http',
function($scope, $http) {
  $http.get('phones/phones.json').success(function(data) {
    $scope.phones = data;
  });

  $scope.orderProp = 'age';
}]);
```

Hivatkozott modulok (függőségek): $scope, $http

Még egy attribútum a modellben

- Adatok dinamikus betöltése
  - $http modul: egyszerű ajax hívások

AngularJS

```
<body ng-controller="PhoneListCtrl">
  <div class="container-fluid">
    <div class="row">
      <div class="col-md-2">
        Search: <input ng-model="query">
        Sort by:
        <select ng-model="orderProp">
          <option value="name">Alphabetical</option>
          <option value="age">Newest</option>
        </select>
      </div>
      <div class="col-md-10">
        <ul class="phones">
          <li ng-repeat="phone in phones | filter:query | orderBy:orderProp">
            <span>{{phone.name}}</span>
            <p>{{phone.snippet}}</p>
          </li>
        </ul>
      </div>
    </div>
  </div>
</body>
```

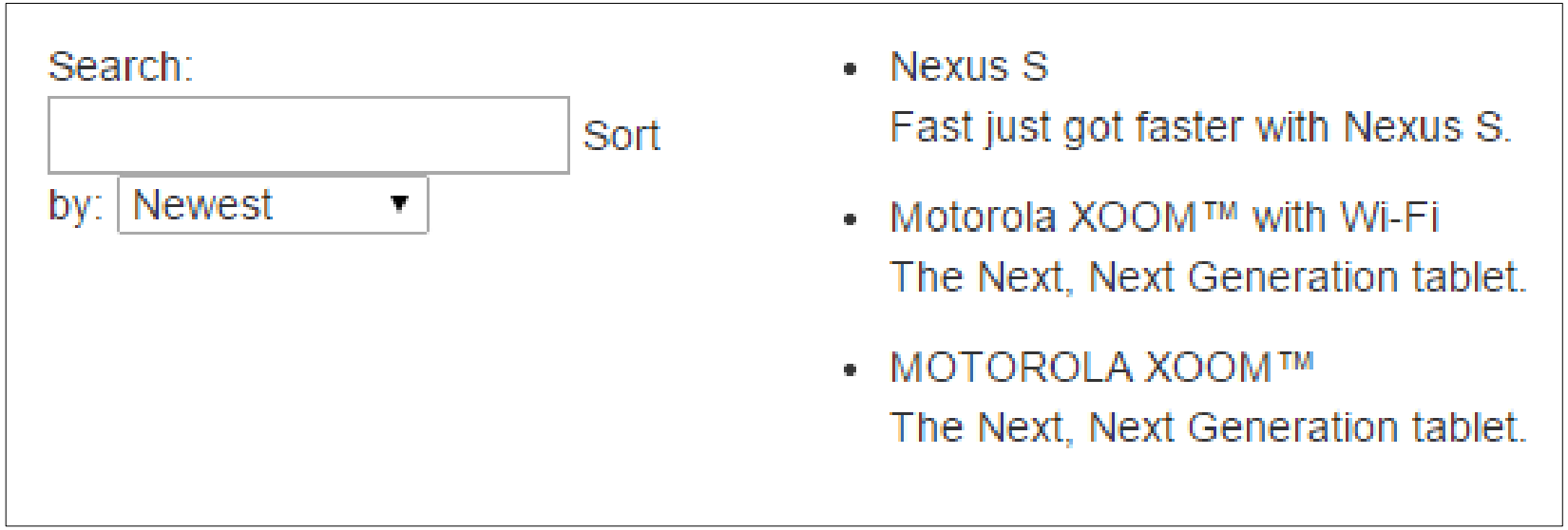
Kétirányú adatkötés

Kétirányú adatkötés

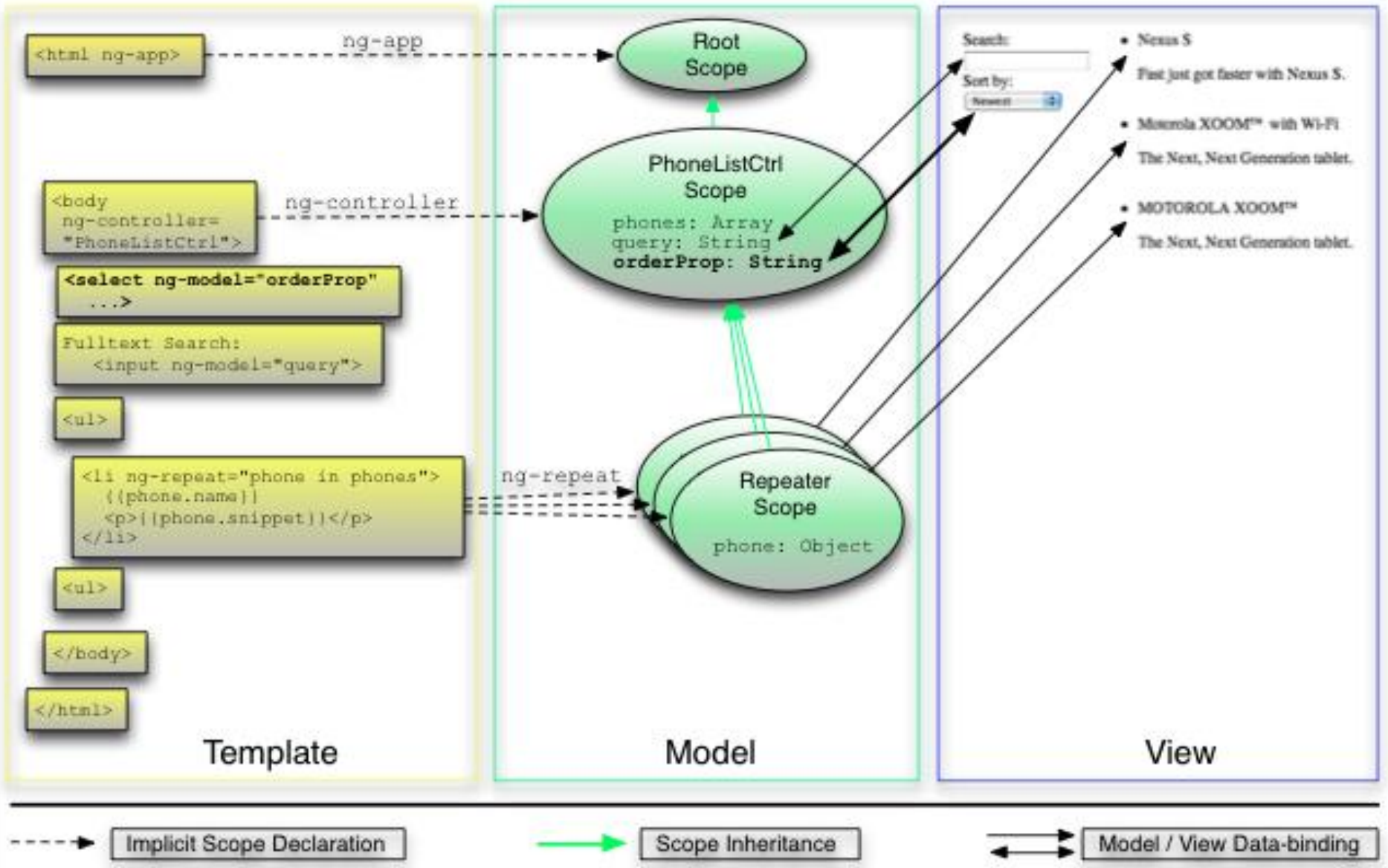
Kétirányú adatkötés egy gyűteményhez

Egyirányú adatkötés

AngularJS


Search:
Sort
by: Newest
Nexus S
Fast just got faster with Nexus S.
Motorola XOOM™ with Wi-Fi
The Next, Next Generation tablet.
MOTOROLA XOOM™
The Next, Next Generation tablet.

AngularJS


<html ng-app>
ng-app
Root Scope
<body ng-controller="PhoneListCtrl">
ng-controller
PhoneListCtrl Scope
phones: Array
query: String
orderProp: String
<select ng-model="orderProp" ...>
Fulltext Search: <input ng-model="query">

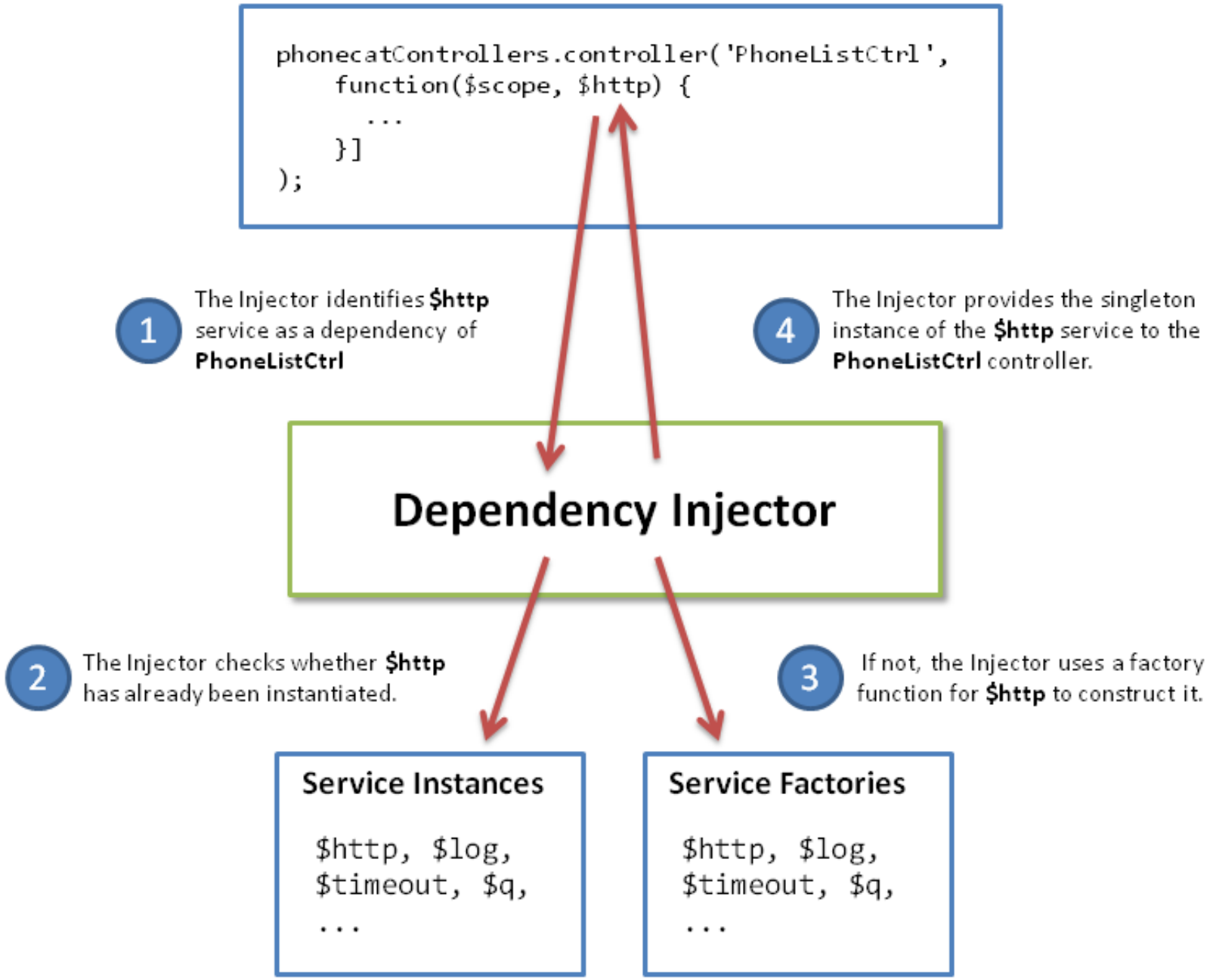
<li ng-repeat="phone in phones">
{{phone.name}}
{{phone.snippet}}

ng-repeat
Repeater Scope
phone: Object

</body>
</html>
Search:
Sort by:
Nexus S
Fast just got faster with Nexus S.
Motorola XOOM™ with Wi-Fi
The Next, Next Generation tablet.
MOTOROLA XOOM™
The Next, Next Generation tablet.
Template
Model
View
Implicit Scope Declaration
Scope Inheritance
Model / View Data-binding

AngularJS

- Dependency Injection
  - A függőséget jelentő modulok inicializálásról és betöltéséről a keretrendszer gondoskodik
  - Csak a nevüket kell megadni



AngularJS

- DEMO

Egyre bonyolultabb kliensoldali kódok…

- A kód komplexitása a szerveroldalról a kliensoldal felé mozog
  - Jó ez?
- Case Study: LinkedIn

  Leaving JSPs in the dust: moving LinkedIn to dust.js client-side templates

  https://engineering.linkedin.com/frontend/leaving-jsps-dust-moving-linkedin-dustjs-client-side-templates
  - Sok UI komponens → cél az újrafelhasználhatóság
  - Többféle szerveroldali technológia → nehéz a kódhordozhatóság
  - Cél: gyors, skálázható, gazdag UI a böngészőben

Case Study – LinkedIn