# MVC-Fundamente, ORM, Routare

**Conf.dr. Cristian KEVORCHIAN**
Facultatea de Matematică și Informatică
ck@fmi.unibuc.ro

# Modele Arhitecturale

- Un **model arhitectural** este o soluție generală, aplicabilă unei probleme frecvent întâlnite în arhitecturarea unor soluții software plasate în diverse contexte, fie ele hardware, software sau de business.
- **Software design** este procesul de conversie a cerințelor utilizatorilor într-o formă adecvată, care ajută dezvoltatorul în scrierea, implementarea și integrarea pieselor software elaborate.
- Modelele arhitecturale sunt similare modelelor de design software, dar au un domeniu de aplicare mai larg.
- Modelele arhitecturale abordează diverse probleme legate de ingineria software, cum ar fi **limitările performanțelor hardware** ale calculatorului, disponibilitatea ridicată și minimizarea riscului în business. Unele modele arhitecturale au fost implementate în cadrul programelor software.

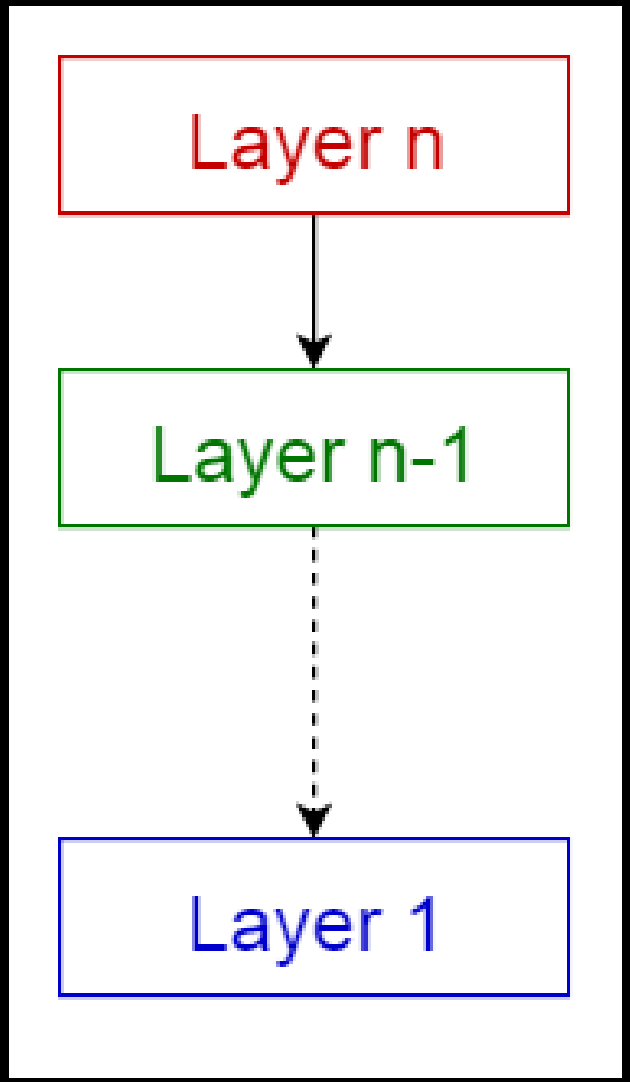
# Nivele de abstracție în software design

- În domeniul ingineriei software, proiectarea software reprezintă o componenta esențială, care poate fi divizată în:
  - **Design architectural**-procesul de definire a unei familii de componente hardware și software precum și a interfețelor acestora în scopul stabilirii cadrului de dezvoltarea a unui sistem informatic.
  - **Design de nivel-înalt(high-level)** – furnizează detalii arhitecturale care vor fi utilizate la dezvoltarea unui produs software. Schema asociată arhitecturii oferă o imagine de ansamblu a întregului sistem, identificând componentele principale care vor fi dezvoltate dar și modul de interfațare al acestora. Sunt utilizați, termeni non-tehnici ușor de înțeles de către administratorii sistemului.
  - **Design detaliat(low-level)**-designul detaliat expune designul logic, detaliat al fiecăreia dintre aceste componente destinat fiind dezvoltatorilor.

# Pattern architectural-Exemple

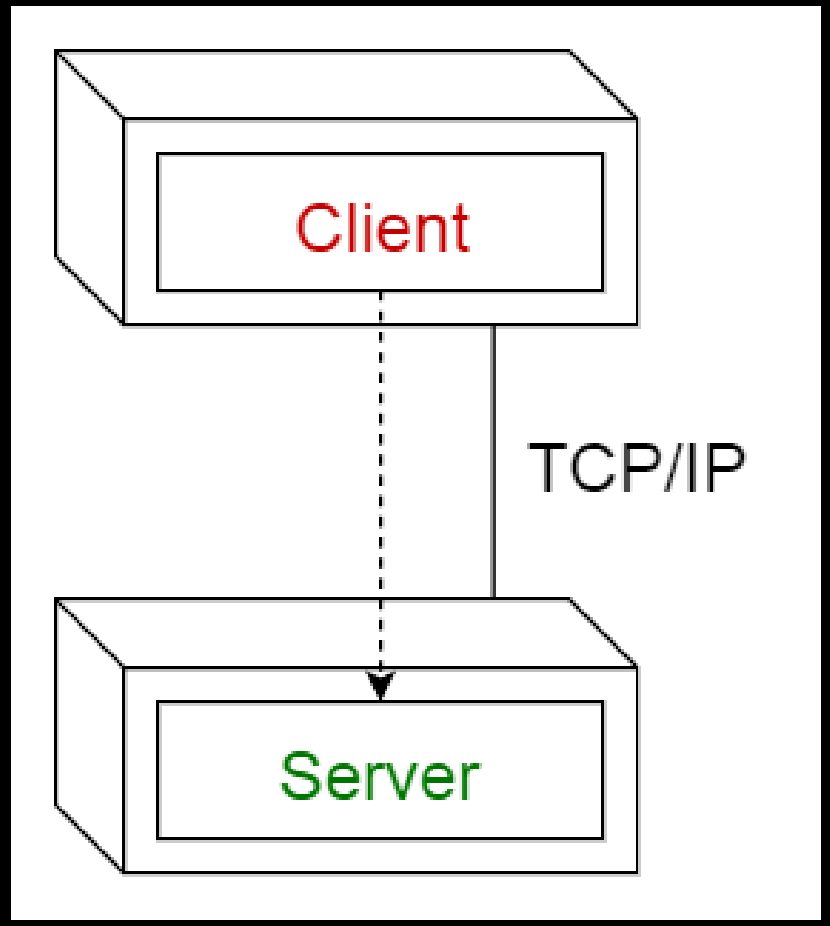
1. Modele stratificate(layered pattern)
2. Modele Client-Server
3. Modele Peer-to-peer
4. Modele Event-Bus
5. Modele Blackboard
6. Modele MVC(Model-view-controller)

# Modele Stratificate




- Acest model poate fi utilizat pentru structurarea programelor care pot fi descompuse în grupuri de sub-task-uri, fiecare dintre acestea fiind plasat la un anumit nivel de abstractizare, în care fiecare strat oferă servicii straturilor superioare.
- Cele mai des întâlnite 4 nivele ale unui sistem software sunt :
  - Nivelul-prezentare (cunoscut și ca nivelul UI)
  - Nivelul-aplicație (cunoscut și ca nivelul-serviciilor)
  - Nivelul business-logic(cunoscut și ca nivelul-domeniu)
  - Nivelul acces-la-date (cunoscut și ca nivelul-persistență)
- Larg utilizate în:
  - Aplicații desktop generale.
  - Aplicatii web pentru e-commerce.

# Aplicații client-server

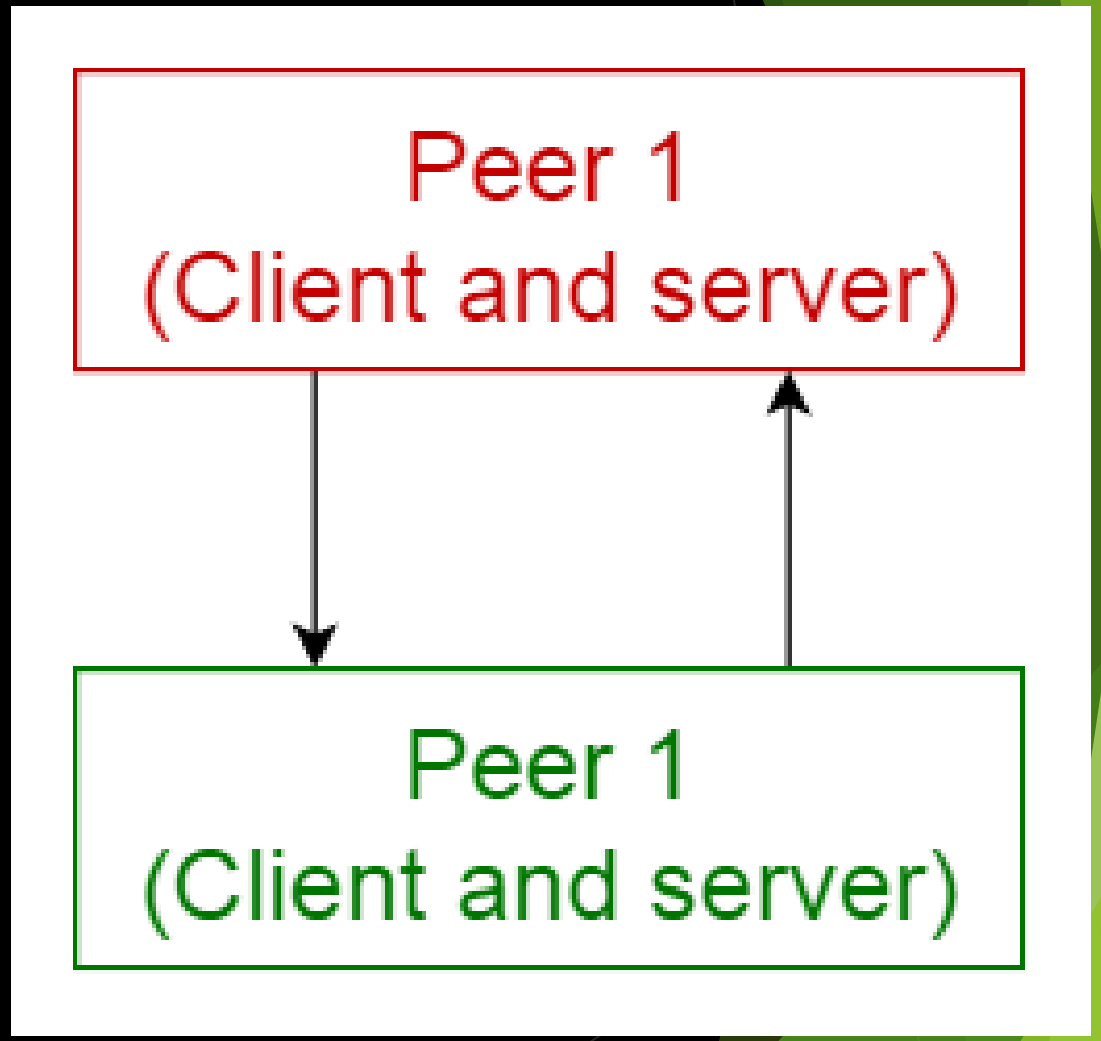



- Acest model este alcătuit din două părți; un server și mai mulți clienți. Componenta server va oferi servicii pentru mai multe componente ale clientului. Clienții solicită servicii de la server și serverul oferă servicii relevante clienților respectivi. Mai mult, serverul continuă să asculte cererile clienților.

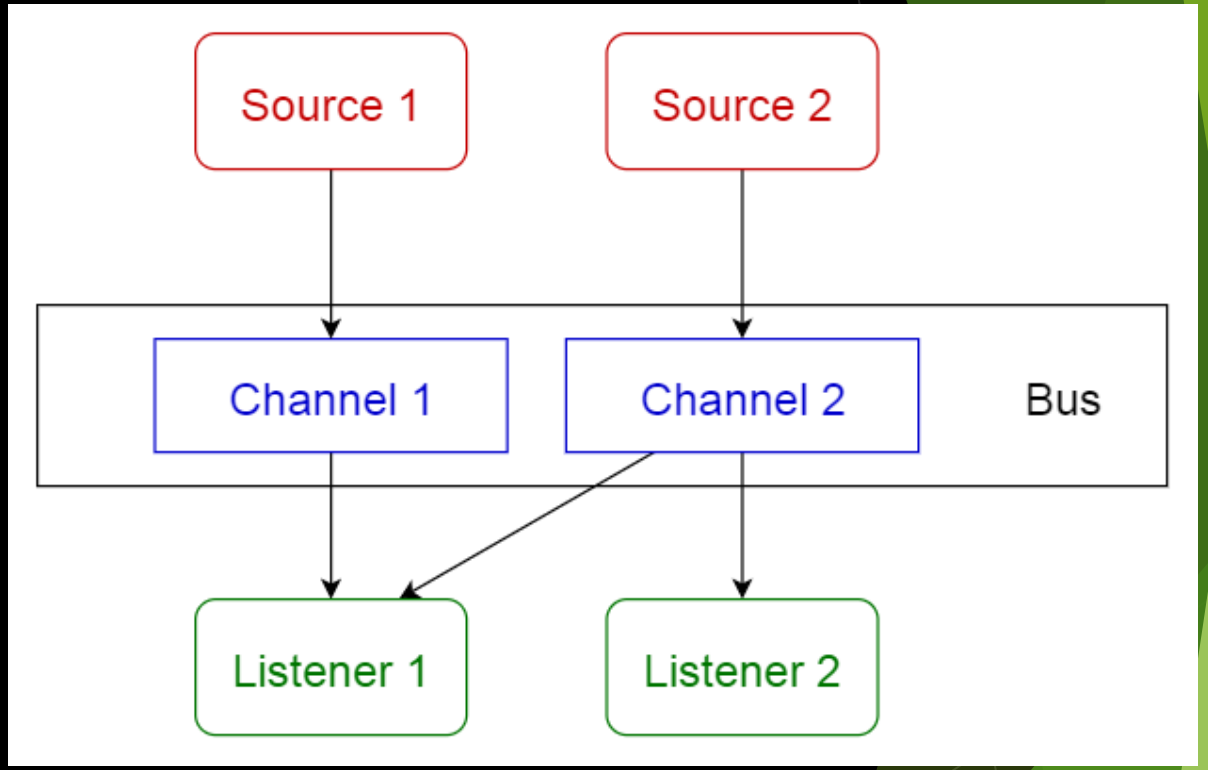
- Utilizare
  - Aplicații online cum ar fi e-mail, partajarea documentelor și servicii bancare.

# Modele P2P(Peer-to-Peer)

- Componentele individuale sunt cunoscute sub numele de P(peer).
- Statutul unui P poate fi unul dual, funcționânâd atât ca client(solicitând servicii de la alți P), cât și ca server(oferind servicii altor P).
- Un P poate acționa ca un client sau ca un server sau ca ambii și își poate schimba rolul dinamic.
- Utilizare
  - File-sharing networks such as **Gnutella** and **G2**
  - Protocol multimedia cum ar fi **P2PTV** si **PDTP**.

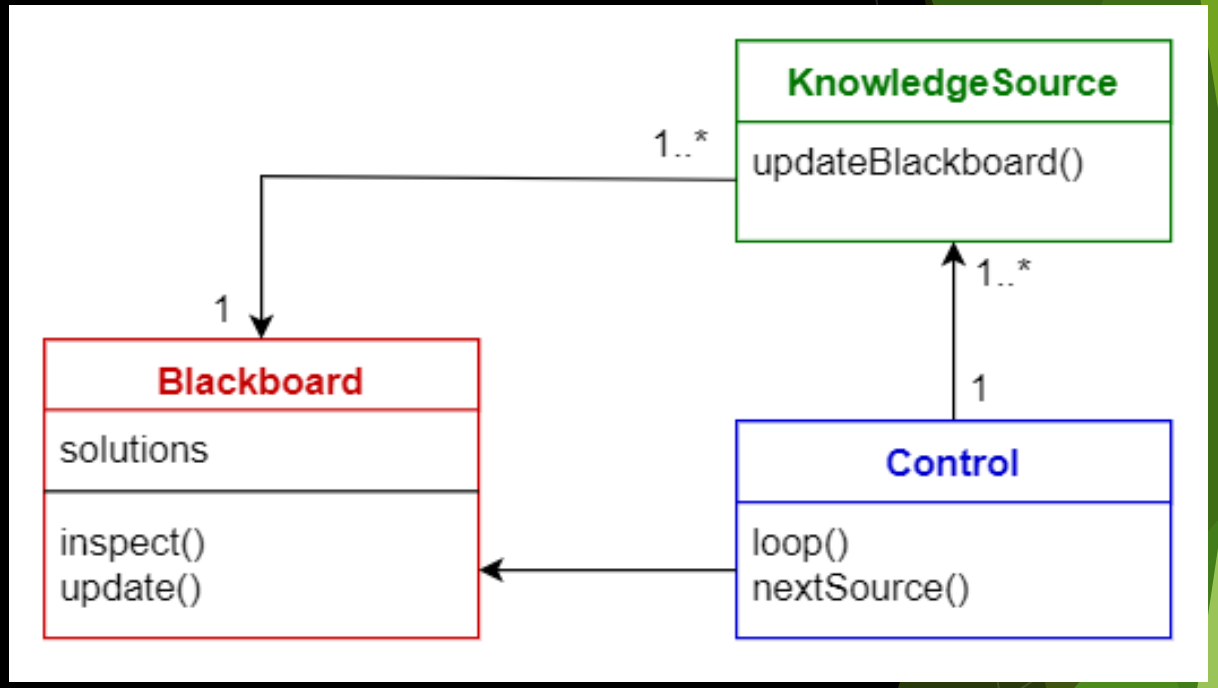


- Componenta centrală a modelului este **evenimentul,** care se identifică cu 4 componente majore:
  - sursă eveniment,
  - Destinatie(listener) de evenimente,
  - canal
  - magistrală de evenimente.
- Sursele publică mesaje către anumite canale dintr-o magistrală de evenimente. Destinatarii se abonează la anumite canale și sunt avertizați asupra mesajelor publicate(sisteme publish-subscriber) pe un canal la care s-au abonat anterior.
- Utilizare
  - Sisteme Android
  - Sisteme publish-subscriber



# Modele Blackboard

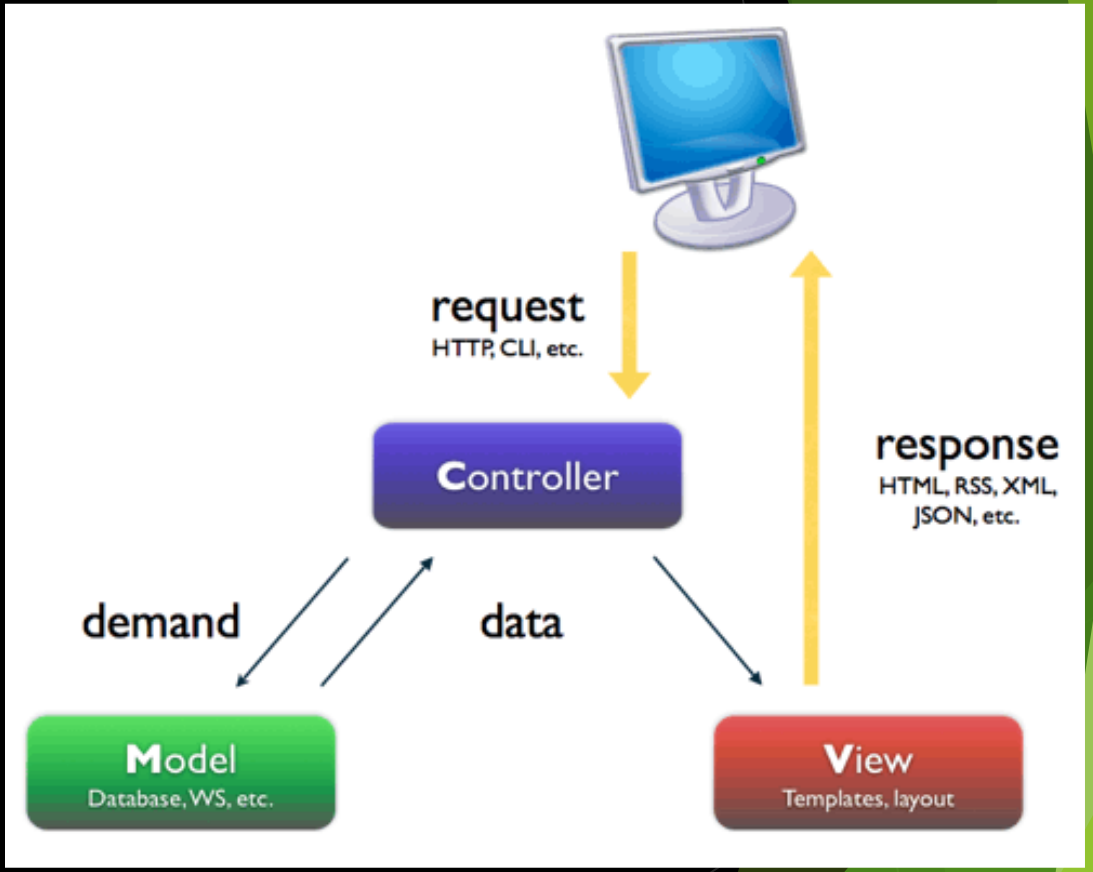
- În IS modelul blackboard este un model de design comportamental care oferă un cadru computational pentru proiectarea și implementarea sistemelor care integrează module specializate de mare complexitate, cu implementarea unor strategii de control nedeterministe.
- Modelul blackboard include trei componente principale:
  - **blackboard** - o memorie structurată și expusă global care conține obiecte din spațiul soluției
  - **sursă de cunoștințe** - module specializate cu reprezentare proprie
  - **componentă de control** - selectează, configurează și execută module.
- Toate componentele au acces la tablă. Componentele pot produce noi obiecte de date care sunt adăugate la tablă. Componentele caută anumite tipuri de date de pe tablă și pot găsi acestea conform modelului care se potrivește cu sursa de cunoștințe existentă.
- **Utilizare:** Recunoașterea vorbirii, Identificarea și monitorizarea autovehiculelor,Identificarea structurii proteinelor, Interpretarea semnalelor furnizate de sonare.



# Originea MVC

- Model-View-Controller este paradigma din spatele interfeței tradiționale pentru limbajul Smalltalk-80. (Interfața cu utilizatorul expusă de Smalltalk-80 MVC este interfața grafică originală cu ferestre care se suprapun, ulterior preluată de Macintosh și Windows.
- Framework-ul dezvoltat de **Trygve Reenskaug (Universitatea din Oslo) pentru implementarea MVC în Smalltalk-80 în 1978 reprezintă începuturile arhitecturii utilizată pe scară largă de peste 15-20 ani.**
- Smalltalk este un limbaj OO dezvoltat în spiritul ”simbiozei om-calculator” și dedicat segmentului educațional(ex:învățare constructivă) la Xerox PARC
- Smalltalk  s-a clasat pe poziția a doua la categoria "most loved programming language" în ”Stack Overflow Developer Survey” în 2017(Rust, Smalltalk,TypeScript,Swift,  Go, Python, Elixir, C#, Scala, Clojure,JavaScript, F#, Haskell, SQL, C++, Julia,Java,R,Ruby, C, PHP, Erlang,Dart,Common Lisp, Groovy

MVC este un pattern arhitectural care fundamentează interacțiunea aplicației cu utilizatorul ca fiind bazată pe trei roluri distincte, "Model", "View" și “Controler“ (Martin Fowler- **Patterns of Enterprise Application Architecture**).



# ”Model”

- Tehnologia de acces la date in contextul utilizarii .NET Framework este **Entity Framework**
- Entity Framework introduce o paradigmă de dezvoltare numita ”Code First”.Code First, permite crearea de obiecte in model prin scrierea de simple clase.(Clase POCO de la ”**P**lain-**O**ld **C**LR **O**bjects”
- O bază de date poate fi creată foarte ușor din clase, care permit, printr-o procedură rapidă dezvoltarea unui workflow.

# ”View”

- Controlorul transmite datele către ”View” prin intermediul unui dicționar numit ViewData. Acesta conține rezultatele împachetate, care sunt convertite în output-ul HTML.
- Ieșirea expusă de ”View”, este de obicei, HTML, însă poate fi JSON sau cod JavaScript.
- ”View” este un fisier aspx  care conține controale semnificative aplicațiilor ASP.NET. Un ”View” poate fi partajat între mai multe controller-e și poate fi un tip puternic sau partial. Un View parțial este analog controalellor din webform și este randat prin intermediul clasei **ViewUserControl**.
- Dacă un View nu este parțial ar putea fi un master page. Acest masterpage este similar cu masterpage din webforms; totuși,este localizat în directorul ViewShared. Un tip puternic View este izolat de Controller si datele pot fi transmise in două moduri distincte fie prin dicționarul ViewData fie prin-un ”object model”

# Controller

- Controller-ul implementează managementul evenimentelor
- Evenimentele pot fi preluate fie din interacțiunea aplicației cu utilizatorul fie prin procesele de sistem
- Furnizează legătura dintre View(interfața cu utilizatorul) și logica de business a aplicației(model).
- Controller-ul utilizează metodele modelului pentru a primi informații despre obiectele aplicației pentru a schimba status-ul obiectului și a informa View-ul despre această schimbare. Într-un sens controller-ul permite utilizatorului să faca modificări și să vadă rezultatele.

# Pe scurt...

- Rolurile pot fi rezumate astfel:
  1. "View" este componenta de prezentare a arhitecturii MVC.
  2. "Modelul" este componenta care răspunde de managementul datelor și le direcționează(gestionează mai multe surse de date) către "View" dar și de logica de business.
  3. „Controller" realizează managementul evenimentelor și extrage "Modelul„ de date corespunzător specificind template-ul "View“ care se asociaza solicitarii intorcind rezultatul. Un controller acceptă cererile și pregătește datele pentru răspuns
- MVC realizează separarea dintre prezentare și celelalte aspecte ale aplicației:
  - Separarea ”View”(”Presentation Logic”) de ”Model”(”Domain Logic”) reprezintă unul dintre cele mai importante principii de proiectare în software.
  - Separarea ”View” de ”Controller” este mai puțin importantă, așadar este recomandat să fie operată când demersul arhitectural o cere.

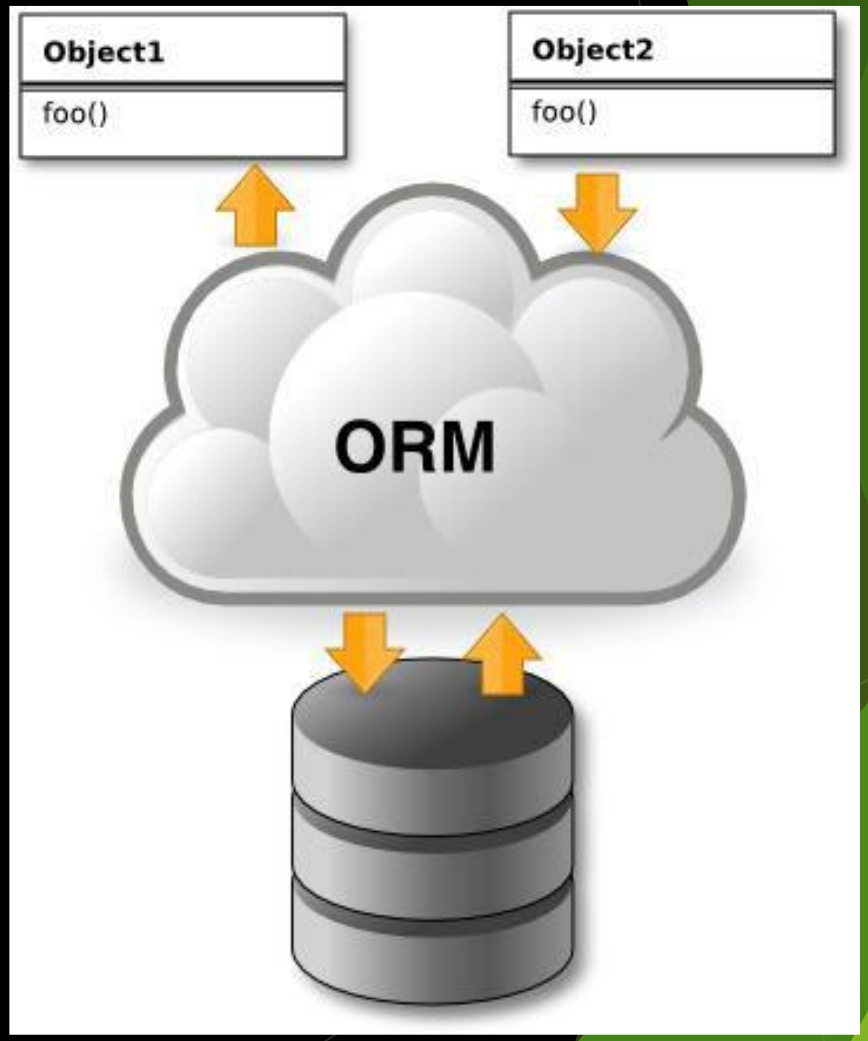
# Abstractizarea Bazei de Date

- Într-o paradigmă OOP aplicațiile sunt construite prin utilizarea de obiecte, care reflectă realitatea business-ului integrând date în comportamentul modelat.
- Modelul relațional, utilizat pentru stocarea datelor, utilizează tabele  și limbaje pentru manipularea datelor
- Diferite SGBD-uri includ facilități OOP dar nu sunt pe deplin compatibile (object-relational impedance mismatch)
- Programarea orientată pe obiecte se bazează pe concepte de programare confirmate de practică, în timp ce modelele relaționale urmează principiile matematice ale modelului relațional.
- Sistemele de mapare relație-obiect au fost dezvoltate tocmai pentru a rezolva aceasta problemă

# ORM(Object-Relational Mapping)

- Un sistem ORM reprezintă o modalitate de a corela datele dintr-o bază de date relațională cu clasele unui limbaj de programare, făcând posibil lucrul cu obiecte, din OOP, în contextul pentru a realiza managementuldatelordin SGBD.
- O platformă MVC trebuie să ofere dezvoltatorilor o modalitate de a interacționa cu un sistem de gestionare a bazelor de date pentru stocare și preluarea datelor. Un sistem ORM este integrat și face parte din această categorie de instrumente.
- Deoarece în arhitectura MVC **Modelul** este nivelul care interacționează cu datele, sistemul ORM se integrează acestuia. Practic, fiecare model este conectat la sistemele ORM și utilizează sistemul pentru a interacționa cu datele.

**Nota:Dezvoltatorii pot accesa resursele sistemului ORM fără a fi nevoie să utilizeze un model.**



# Routarea in MVC

- Utilizează un middleware de routare prin care se mapează URL-urile asociate cererilor de procesare pe acțiuni..
- Exemplu: Dorim ca orice utilizare a URL-ului:

**http://localhost/Controler/Actiune**

sa fie direcționată către **Controllerul** ”Controler” și să invoce actiunea “Actiune”. Acest lucru se defineste prin adăugarea în colecția de route prin utilizarea functiei “maproute”.

# Routarea in MVC

- O tabela de routare este creată in Global.asax(contine hadler-ul de evenimente pentru ciclul de viata al aplicațiilor ASP.NET)
- ASP.NET MVC invocă diferite clase controller(si diferite metode acțiuni în cadrul acesteia) depind de URL-ul receptionat. Logica de routare implicita utilizata de ASP.NET MVC utilizeaza un format de tipul:

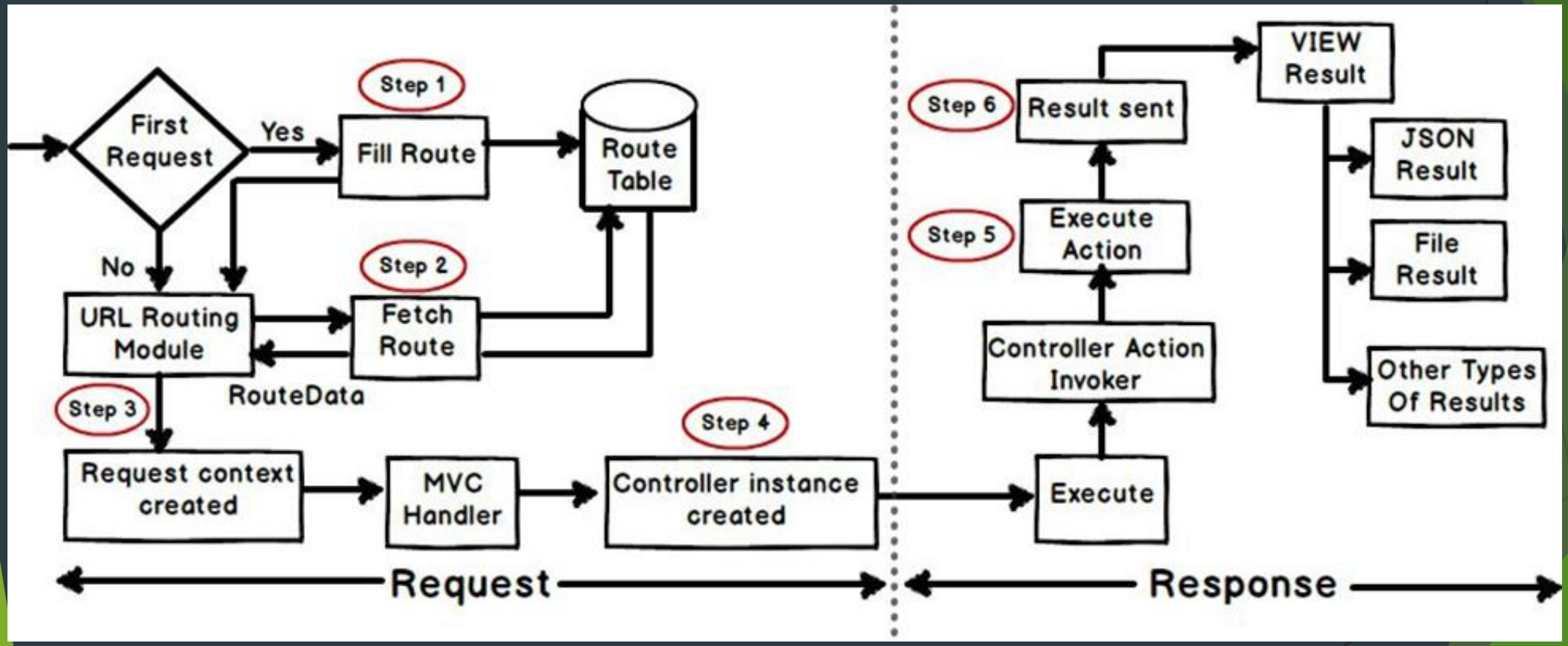
/[Controller]/[NumeleActiunii]/[Parametri]

setarea routarii se realizeaza in App_Start/RouteConfig.cs

# Ciclul de viata al aplicatiilor MVC(1)

- Aplicațiile web se caracterizează prin două etape de execuție:
  - Semantica solicitării
  - In functie de tipul cererii obținerea unui raspuns adecvat
- Ciclul de viata MVC implică
  - Crearea obiectului ”cerere”
  - Trimiterea raspunsului catre browser

# Ciclul de viata al applicatiilor MVC(2)



# Crearea obiectului ”cerere”

- Exista patru etape in crearea obiectului ”cerere”:
  - Pasul 1. Crearea routei: - cererile MVC sunt mapate la nivelul tabelelor de route, care, la rândul lor specifică care controller și ce acțiune urmează a fi efectuată. Astfel încât în cazul în care cererea este la prima solicitare, primul lucru este acela de a completa tabela de routare cu colecția de route. Completarea tabelei de routare se realizează în **Global.asax**.
  - Pasul 2. Extragerea routei: - În funcție de adresa URL trimisa "**UrlRoutingModule**" caută în tabela de routare pentru a crea obiectul "RouteData", care este însoțit de unele informații necesare identificării controllerului și acțiunii invocate.
  - Pasul 3. Solicitarea contextului creat: - Obiectul "RouteData" este utilizat la crearea obiectului "RequestContext".
  - Pasul 4. Instanta controller creată: - Cererea este transmisă către instanta clasei "MvcHandler" pentru a crea instanta clasei controller-ului. Odată ce obiectul asociat clasei controller aceasta apelează metoda "Execute" a clasei controller.

# Crearea obiectului ”Response”

- Aceasta fază a ciclului de viață MVC are două etape, ca în final să trimită răspunsul ca rezultat catre ”View”:
  - Pasul 5. Execuția acțiunii:- ”ControllerActionInvoker” determină care actiune se execută și trece la executarea ei.
  - Pasul 6. Trimiterea rezultatului:- Executarea metode asociate acțiunii și crearea tipului asociat rezultatului, poate fi văzut ca un: fisier rezultat, document JSON,etc.

# Comparatie MVC vs. Arhitectura 3-tire

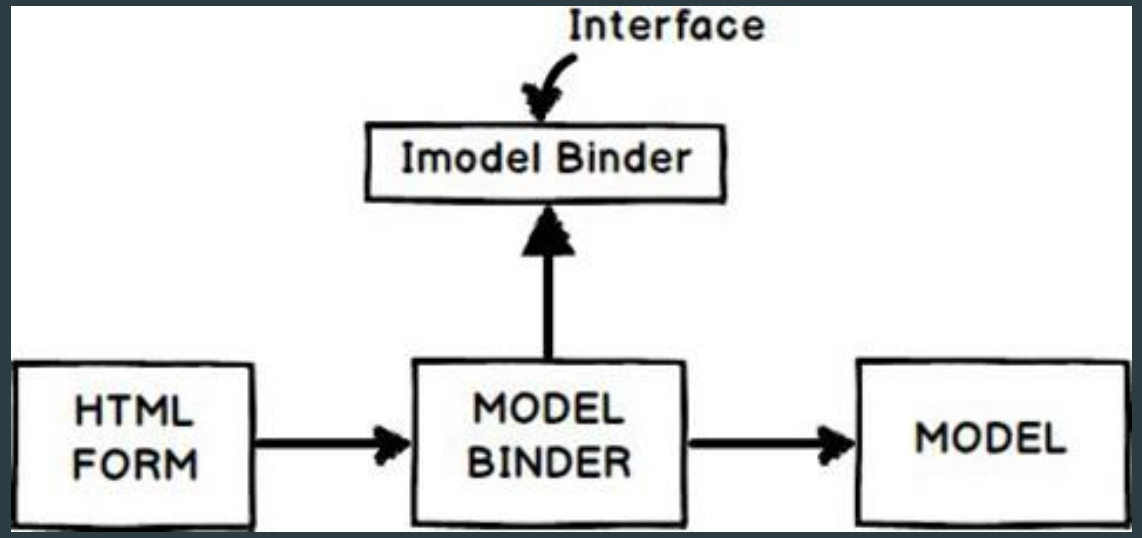
| Funcționalități | Arhitectura 3-tire | Arhitectura MVC |
|---|---|---|
| Aspecte legate de design | Interfata cu utilizatorul | View |
| Logica UI | Interfata cu utilizatorul | Controller |
| Logica de Business/Validare | 2-layer | Modelul |
| Cererea | UI(1-layer) | Controller |
| Accesul la date | DAL(Data Access Layer)3-layer | DAL |

# Beneficii ale utilizării MVC

- **Există doua mari avantaje în utilizarea MVC:**
  - Separarea funcționalităților se realizează pe măsură ce se deplasează din zona codului(legat de functionalitate) într-un fișier ce include o clasă separată. Prin mutarea codului(legat) într-un fișier separat ce include o clasă acesta se poate refolosi într-o mare măsură.
    • Testarea automata a IU este posibil, deoarece acum codul din spatele (codul ce interacționează cu IU) s-a mutat într-o clasă simplă .NET. Acest lucru ne oferă posibilitatea de a scrie teste unitare și automatiza testarea manuală.

# Importanta lucrului cu ”Model Binders” în MVC

Un ”Model Binder” pune în corespondență elemente ale form-ului HTML cu cele ale modelului.



Să considerăm următorul form:

```
<form id="frm1" method=post action="/Studenti/SubmitStudenti">
      Student cod:-<input name="SCod" type="text" />
      Student virsta:-<input name="SVirsta" type="text" />
       <input type=submit/>
<form>
```

Pentru clasa Student apartinînd modelului:

```
Public  class Student
{
    public string CodStudent {get; set;}
    public int32 VirstaStudent {get; set;}
}
```

```
public class StudentModelBinder : IModelBinder
    {
        public object BindModel(ControllerContext controllerContext,
ModelBindingContext bindingContext)
        {
            var request = controllerContext.HttpContext.Request;


            string CodStudent = request.Form.Get(“CStudent");
            int VirstaStudent = Convert.ToInt32(request.Form.Get(“SVirsta"));
            return new Student { Cod = CodStudent, Virsta = VirstaStudent };
        }
    }
```

# Surse de Documentare

https://www.asp.net/mvc/overview/getting-started/introduction/adding-a-controller