Univerza v Ljubljani
Fakulteta za računalništvo
in informatiko

Ajax

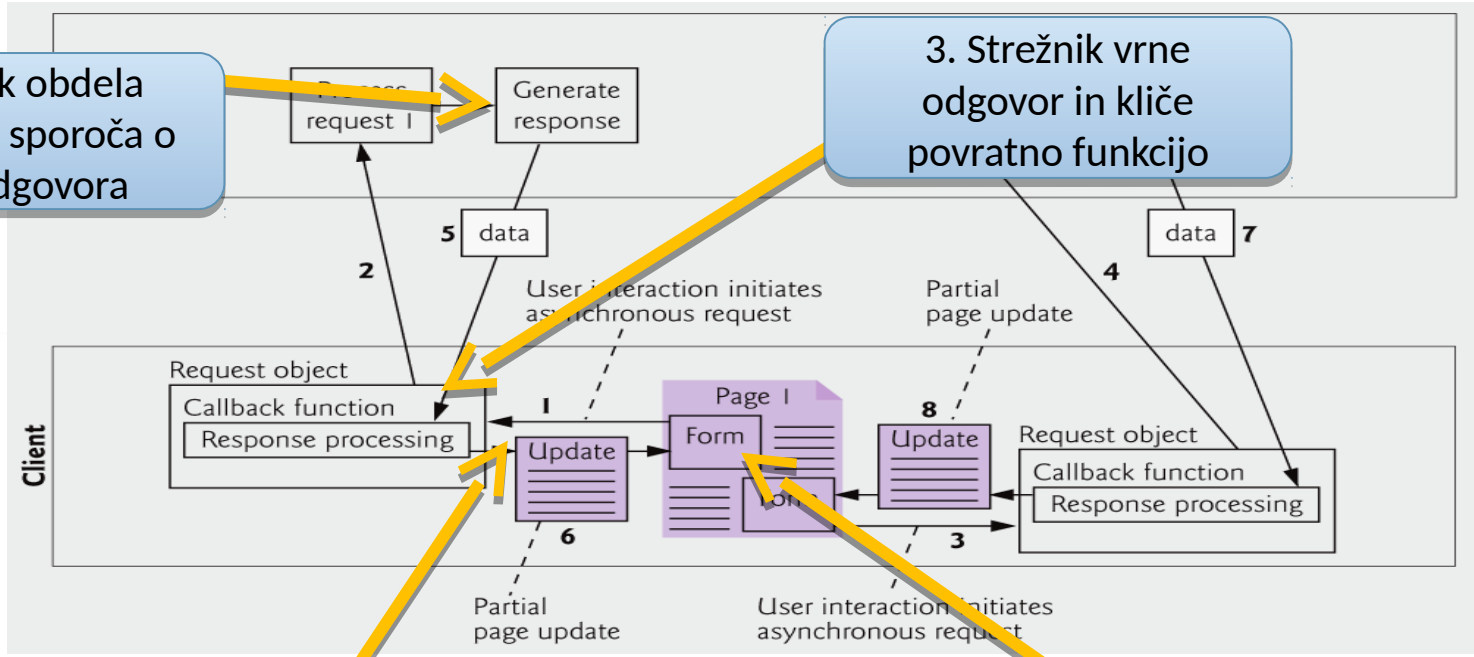
Uvod

- potreba po asinhronih zahtevkih, ki omogočajo spletnim aplikacijam večjo odzivnost
- porast potreb po "**Rich Internet Applications**" (RIA): bogate spletne aplikacije, aplikacije, ki imajo bogat uporabniški vmesnik in odzivnost podobno namiznim aplikacijam
- Asinhronous JavaScript And XML (Ajax) omogoča:
  - izvedbo asinhronih zahtevkov
  - posodabljanje le delov dokumenta namesto prenosa celotnega dokumenta
  - je kombinacija tehnologij na strani odjemalca(JavaScript, XML, HTML, DOM, CSS)
- zahtevke Ajax izvedemo z:
  - objektom `XMLHttpRequest`
  - JavaScript orodji (jQuery, Dojo, Prototype)

Primer aplikacije z Ajax - 4 faze

2. Strežnik obdela zahtevek in sporoča o stanju odgovora

3. Strežnik vrne odgovor in kliče povratno funkcijo



1. Aplikacija izvede asinhroni zahtevek

4. Klient izvede posodobitev strani

Primer aplikacije z Ajax

- spletna aplikacija
  - omogoča vnos podatkov v obrazec in registrira dogodek, ki se proži ob zapustitvi polja za vnos poštne številke
  - krmilnik dogodka `onblur` vsebuje programsko kodo za izvedbo asinhrone zahteve

FAZA 1: Izvedba asinhrone zahteve

- za izvedbo zahteve se uporabi objekt `XMLHttpRequest()`
  starejši Microsoft brskalniki lahko uporabljajo tudi druge ActiveX komponente
- `var xhr = new XMLHttpRequest();`
- lastnosti in metode objekta `XMLHttpRequest()`:

Lastnost	Opis
onreadystatechange	Hrani naziv funkcije za povratni klic – krmilnik, ki se izvede ob odgovoru strežnika.
readyState	Hrani stanje o tem, kaj se dogaja z zahtevkom. Vrednosti: 0 – neinicializiran, 1 – zahtevek se nalaga, 2 – zahtevek naložen, 3 – prejemanje podatkov od strežnika, 4 – zahtevek zaključen
responseText	Besedilo (rezultat), ki ga vrne strežnik.
responseXML	XML dokument (rezultat), ki ga vrne strežnik, če je zahtevan XML
status	HTTP koda statusa zahtevka (npr. 200, 404, 500)
statusText	Besedilni opis statusa zahtevka.

Metoda	Opis
open	Inicializira zahtevek, poda metodo (GET/POST) in URL
send	Pošlje zahtevek strežniku.
setRequestHeader	Nastavi glavo zahtevka.
getResponseHeader	Vrne glavo zahtevka.
getAllResponseHeaders	Vrne polje glav, ki so pred vsebino odgovorov.
abort	Prekliče zahtevek.

FAZA 1: Izvedba asinhrone zahteve

- v lastnost `onreadystatechange` shranimo naslov odzivne funkcije (callback), ki se bo klicala ob prejemu odgovora (= rokovalnik dogodka)
  `xhr.onreadystatechange = receivePlace;`
- za izdelan zahtevek kličemo funkciji
  - `xhr.open()`, ki ji podamo vrsto metode (GET/POST) in URL
  - `xhr.send()`, ki pošlje zahtevek

```
function getPlace(zip) {
   var xhr = new XMLHttpRequest();
   xhr.onreadystatechange = function () {
      if (xhr.readyState == 4 && xhr.status == 200) {
         var result = xhr.responseText;
         var place = result.split(', ');
         document.getElementById("city").value = place[0];
         document.getElementById("state").value = place[1];
      }
   }
   xhr.open("GET", "getCityState.php?zip=" + zip);
   xhr.send(null);
}
```

primerek zahtevka

registracija povratne funkcije

naslov zahtevka

pošiljanje zahtevka

FAZI 2 in 3: Obdelava in odgovor strežnika

- strežnik izvede obdelavo (npr. php skripta) in vrne odgovor:
  - glavo HTTP (header,) ki vsebuje tip MIME: text/plain, text/html ali text/xml
  - odgovor generira glede na poslane podatke
  - vrne/izpiše odgovor
- ob povratnem klicu preverimo, ali je bil zahtevek uspešno izveden:
  `if (xhr.readyState == 4 && xhr.status == 200)`
- možne vrednosti za `readyState` in `status`:

readyState	
0	neinicializiran
1	zahtevek se nalaganje
2	zahtevek je naložen
3	prejemanje podatkov od strežnika

status	
1xx	informacija
2xx	uspešno (200 – OK)
3xx	preusmeritev
4xx	napaka na strani odjemalca
5xx	napaka na strani strežnika

FAZA 4: Uporaba odgovora za posodobitev

- možne vrste odgovorov:

1. **besedilo** (atribut `responseText`); za nestrukturirane podatke
2. **HTML** (atribut `responseText`); primer uporabe, značka div v originalnem dokumentu, JavaScript koda interpretira vrnjen tekst zahtevka Ajax:

```
var divDom = document.getElementById("replaceable_list");
divDom.innerHTML = xhr.responseText;
```

3. **XML** (atribut `responseXML`)
   Rezultat vpišemo z metodami nad DOM ali preslikavo XSLT
4. **JSON** (atribut `responseText`)

```
<header> 2007 US Champion/Runnerup - football </header>
<list_item> New York Giants </list_item>
<list_item> New England Patriots </list_item>
```

Odgovor tipa JSON

- JSON - JavaScript Object Notation
- Odgovor JSON vrne strežnik v lastnosti `responseText`
  - predstavitev objektov v obliki nizov s seznami ime:vrednost in polji
  - vrednost je lahko število, niz, boolean, polje ([...]), objekt ({...}) ali pa null

```
{"employees" :  [
                {"name" : "Dew, Dawn", "address" : "1222 Wet Lane"},
                {"name" : "Do, Dick", "address" : "332 Doer Road"},
                {"name" : "Deau, Donna", "address" : "222 Donne Street"} ] }
```

- tekstovni opis objekta pretvorimo v objekt JavaScript z metodo JSON.parse (in eval()!)

```
var response = xhr.responseText;
var myObj = JSON.parse(response);
var address2 = myObj.employees[1].address;
```

- bolje kot XML: manj besedila, razčlenitev hitrejša kot razčlenitev XML ali transformacija XSLT, enostavno za branje/pisanje in razčlenjevanje/tvorjenje

Pomoč: knjižnica DOJO

- prostodostopna knjižnica modulov za poenostavitev rabe jezika JavaScript, vključujoč zahtevke Ajax, animacijo vizualnih efektov, povleci-spusti elementov dokumenta in rokovanje z dogodki
- olajša nekatere kompatibilnostne probleme brskalnikov
- za vključitev je potrebna referenca na lokalno kopijo ali na CDN , podobno kot za jQuery

```
<script type = "text/javascript" src = "dojo/dojo.js"></script>
```

- izvedba asinhronih zahtevkov
  - metodi `dojo.xhrGet` in `dojo.xhrPost`
  - potrebna sta vsaj dva parametra: url (naslov strežnika) in load (ime povratne funkcije)

```
dojo.xhrGet( {url: "getColors.php" + "?size=" + size,
        load: buildMenu
});
```

Pomoč: JQuery Ajax

- podobno kot z Dojo lahko delamo tudi z JQuery
- asinhroni zahtevek + polnjenje kontrole: `$(selektor).load(URL, callback)`

```
$("button").click(function(){
   $("#div1").load("demo_test.txt");
});
```

- asinhroni zahtevek GET: `$.get(URL,callback)`

```
$("button").click(function(){
    $.get("demo_test.asp“ + “aData=” +    someData,
          function(data,status){
                ...
          });
});
```

- asinhroni zahtevek POST: `$.post(URL,data,callback)`

Varnostni pomisleki

1. kompleksne aplikacije zahtevajo več kode

2. več (delov) skript - več zahtevkov strežniku - večja ranljivost

3. možnost napadov kot sta:
   - "cross site scripting" – injeciranje skript v spletne strani
   - "code injection" – injeciranje kode, ki spremeni potek izvajanja

4. možnost nelegalnih asinhronih zahtevkov strežniku

5. dovoljeno je, da se asinhroni zahtevki izvedejo samo na strežniku, ki je posredoval glavno stran dokumenta

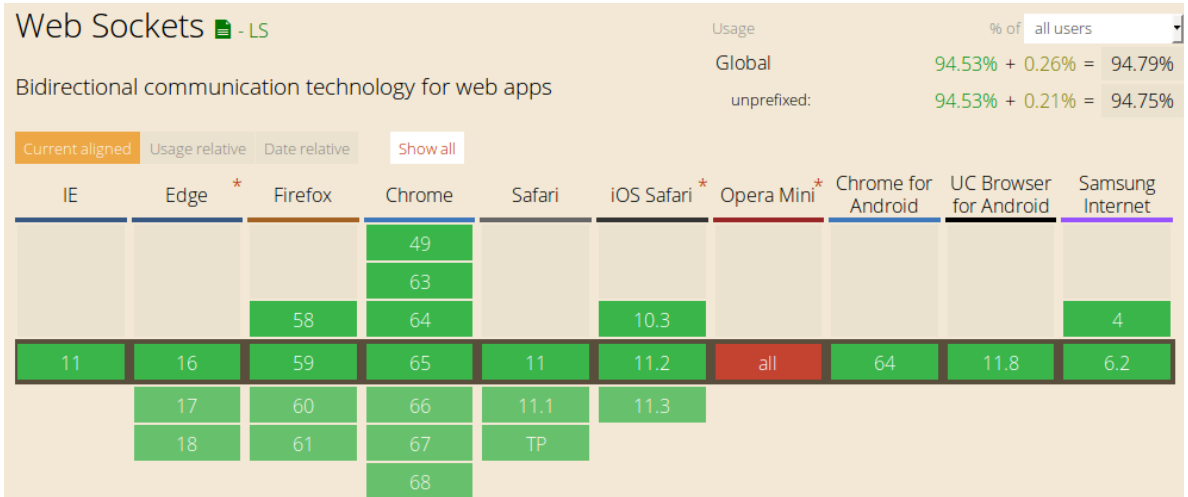
WebSockets

Uvod

- WebSocket je računalniški komunikacijski **protokol**, ki omogoča dvosmerno komunikacijo
- WebSocket je neodvisen protokol temelječ na TCP, standardiziran leta 2011
- narejen za uporabo v spletnih brskalnikih in spletnih strežnikih, vendar ga lahko uporablja katerakoli strežniška ali odjemalska aplikacija
- WebSocket rokovanje razumejo strežniki kot zahtevo za nadgradnjo
  - odjemalec zahteva WebSocket povezavo z uporabo **Upgrade: WebSocket** in **Connection: Upgrade** polj glave
  - če strežnik razume protokol, odgovori z istima poljema glave in konča rokovanje
  - po rokovanju se lahko začne prenos podatkov
- WebSocket omogoča realnočasovno komunikacijo na in od strežnika
- omogoča vzpostavljanje dolgo-časovnih povezav preko TCP vtičnic
- omogoča takojšnje razpošiljanje sporočil z malo redundance

Uvod

- protokol WebSocket podpirajo vsi najpomembnejši brskalniki
- protokol WebSocket zahteva, da tudi aplikacije na strani strežnika razumejo ta protokol in zagotavljajo podatke
- strežniki, ki podpirajo protokol, imajo lahko hkrati odprtih več povezav z različnimi odjemalci v kateremkoli trenutku



Ajax proti WebSocket

- WebSocket ne naredij Ajax nepotrebnega
- obe tehnologiji se dopolnjujeta
- Ajax je tehnologija izbire ko so potrebne kratke povezave, na primer kratki kklici spletnih storitev
- WebSocket naj bi se uporabil ko je potrebna realno-časovna funkcionalnost, saj omogočajo nizko-latentno, dvo-smerno komunikacijo preko enega samega kanala

Uporaba WebSocket-a na odjemalcu

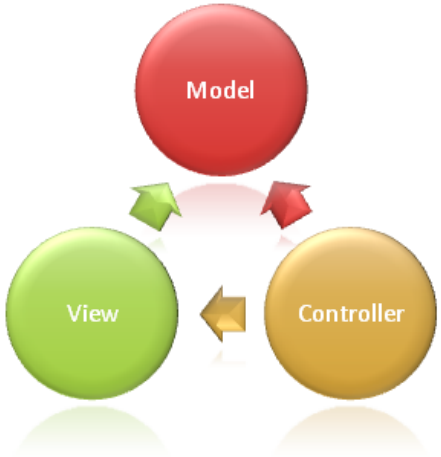
```
<script type="text/javascript">
var socket;
function init() {
  var host = "ws://127.0.0.1:9000/echobot";
  try {
    socket = new WebSocket(host);
    log('WebSocket - status '+socket.readyState);
    socket.onopen = function(msg) {log("Welcome");};
    socket.onmessage = function(msg) {
      log("Received: "+msg.data.length);
      var myData = JSON.parse(msg.data);
      width=myData[0]; height=myData[1];
      log("Size"+myData.length+" "+width+" "+height);
      imgData = rContext.createImageData(width, height);
      binary = imgData.data;
      rContext.fillStyle="orange";
      rContext.fillRect(0, 0, width, height);
      for (var i=2;i<myData.length;i++)
        {binary[i-2]=myData[i];}
      rContext.putImageData(imgData, 10, 10);
      log("Pixels: ");};
    socket.onclose=function(msg) {log("Disconnected");};
  }catch(ex){log(ex);}
  $("msg").focus();
}
function send() {
   var w=300, h=200;
   var img = context.getImageData(100,60,w, h);
   var binary = new Array(img.data.length+2);
   binary[0]=w;
   binary[1]=h;
   for (var i=0;i<img.data.length;i++)
     {binary[i+2]=img.data[i];}
   try {socket.send(JSON.stringify(binary));}
   catch(ex) {log(ex); }
}
function quit(){
   if (socket != null) {
      socket.close();
      socket=null;
   }
}
function reconnect() {quit();init();}
function $(id){ return document.getElementById(id); }
function log(msg){ $("log").innerHTML+="<br>"+msg;}
….
</script>
```

Uporaba WebSocket-a na odjemalcu

```
<body onload="init()">
    <h3>WebSocket v2.00</h3>
    <div id="log"></div>
    <button onclick="quit()">Prekini</button>
    <button onclick="reconnect()">Povezi</button>
    <button id="slikaj">Posnemi sliko</button>
    <button id="poslji">Poslji sliko</button><p />
    <video id="video" width="480"
        height="320" autoplay></video>
    <canvas id="slika" width="480" height="320"></canvas>
    <canvas id="prejeta" width="480" height="320"></canvas><p />
    <script>
      window.addEventListener("DOMContentLoaded",
                              doVideo, false);
      document.getElementById("slikaj").addEventListener
                              ("click", paintCapture);
      document.getElementById("poslji").addEventListener
                              ("click",send);
    </script>
  </body>
```

```
function openStream(stream){
  video.src=window.URL.createObjectURL(stream);
  video.play();};
function errorFunction(error){
  console.log("Napaka pri zajemu: ", error);}
function paintCapture() {
  context.drawImage(video, 0, 0, 480, 320);}
function doVideo() {
  canvas = document.getElementById("slika"),
  context = canvas.getContext("2d"),
  rCanvas = document.getElementById("prejeta"),
  rContext = rCanvas.getContext("2d"),
  video = document.getElementById("video"),
  videoObj = { "video": true, "audio": false  };
  navigator.getUserMedia = navigator.getUserMedia||
        navigator.webkitGetUserMedia ||
        navigator.mozGetUserMedia ||
        navigator.msGetUserMedia;
  if(navigator.getUserMedia) {
       navigator.getUserMedia(videoObj,
                    openStream, errorFunction );

    }
}
```



MVC

MVC – vzorec

- Model (model)
  - podatki
  - metode za dostopanje in spreminjanje podatkov in stanja
  - ni povezan z vmesnikom oziroma predstavitvijo
  - pogosto je shranjen na neki lokaciji
- Pogled (view)
  - prikaže vsebino modela uporabniku v ustreznem vmesniku
  - omogoča uporabniku upravljanje podatkov
  - ko se model spremeni se mora pogled prilagoditi
- Kontroler (controller)
  - posrednik med modelom in pogledom
  - prevede uporabnikove akcije (interakcija s pogledom) v operacije na modelu
  - posodobi model, ko uporabnik upravlja s pogledom
  - primeri uporabnikovih akcij: pritiski na gumb, izbira v meniju, ...

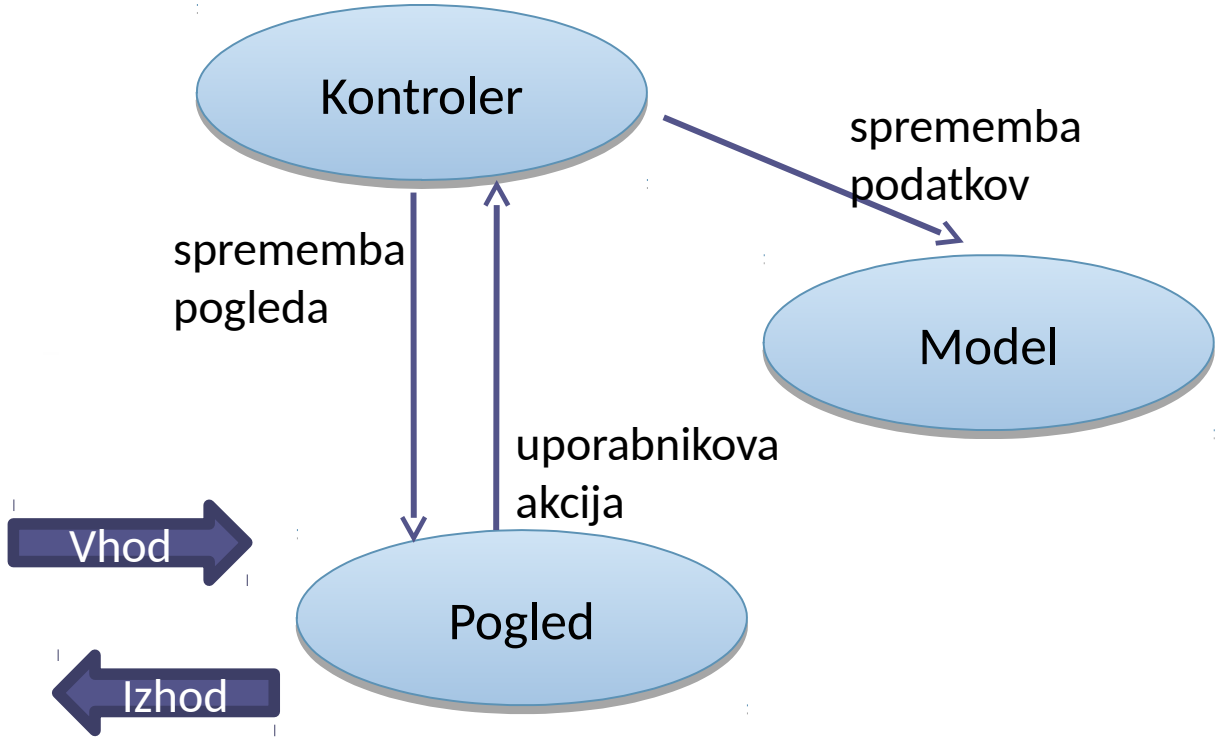
MVC – motivacija za pristop

- osnovne komponente vsake interaktivne aplikacije:
  - podatki, s katerimi se rokuje
  - uporabniški vmesnik, ki služi za rokovanje s podatki
- podatki so logično neodvisni od načina predstavitve uporabniku
  - prikaz naj bi bilo mogoče načrtovati ločeno
- primer takega pristopa: porazdelitev ocen pri predmetu
  - mogoča je predstavitev s stolpičnim in/ali tortnim grafikonom
- prvotno se pojavi pri razvoju uporabniških vmesnikov za namizne računalnike (1979), zelo primeren pristop tudi pri razvoju spletnih aplikacij

Zakaj uporaba MVC

- dobra ločitev predstavitve, podatkov in poslovne logije
  - omogoča lažje vzdrževanje aplikacije
  - omogoča izogibanje enemu razredu/objektu/..., ki bi postoril vse
- ločevanje delov aplikacije omogoča tudi ponovno uporabljivost
  - z zmanjševanjem odvisnosti posameznik komponent vmesnika je lažje vzeti neko napisano komponento in jo ponovno uporabiti kje drugje
- omogoča zmanjševanje kopleksnosti posameznih delov aplikacije
  - kompleksnot je porazdeljena med različne dele, vsak del je tako zase manj kompleksen
- povečana fleksibilnost
  - vsak del aplikacije je lažji za prilagoditev kakršnim koli spremembam
- omogoča ločen razvoj, testiranje in vzdrževanje vsakega izmed delov aplikacije

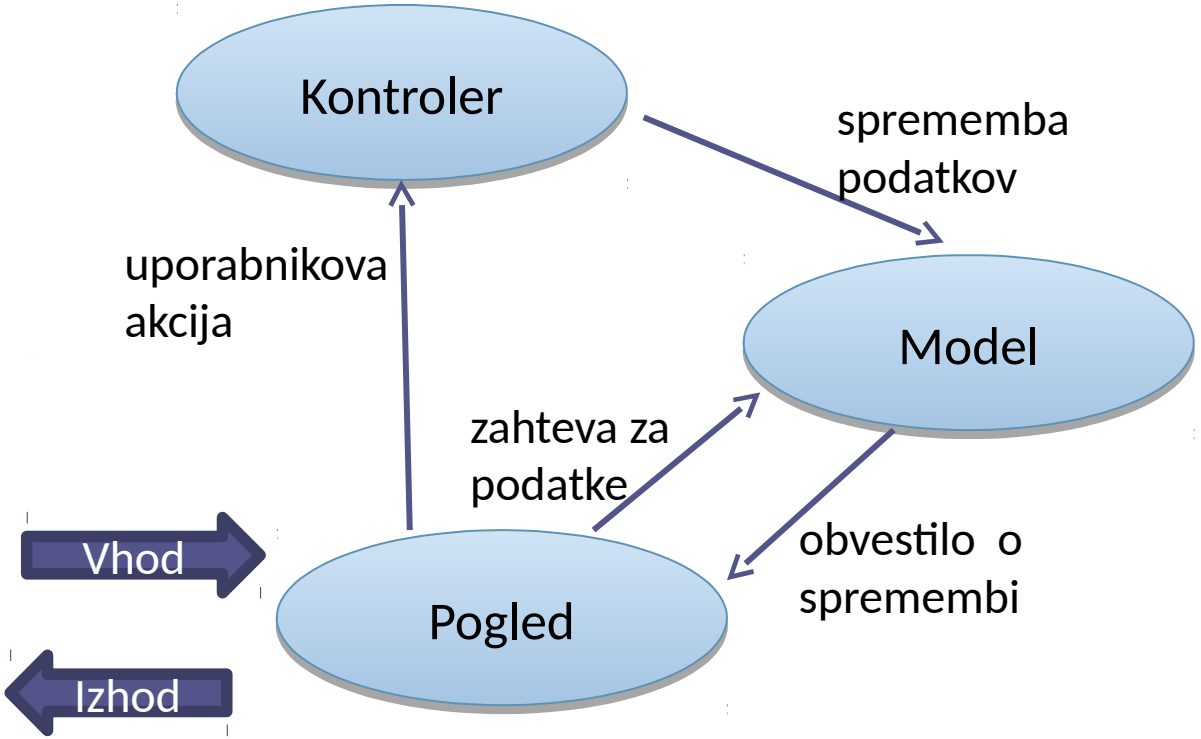
Osnovna interakcija v MVC



Osnovna interakcija v MVC

- uporabnik izvaja interakcijo z uporabniškim vmesnikom (pogled)
- kontroler prejme vhod od uporabniškega vmesnika
- kontroler spremeni model glede na uporabnikovo akcijo
- model procesira posodobitve, ki jih je zahteval kontroler, pogosto je to branje/pisnje v podatkovno bazo
- kontroler posodobi ali spremeni pogled s posodobitvami modela
- pogled se prikaže in ponovno čaka na uporabnikovo interakcijo

Razširjena interakcija v MVC



Razširjena interakcija v MVC

- objekt je obveščen o spremembah drugega objekta
- v razširjeni interakciji je pogled opazovalec modela
- asinhrone spremembe modela
  - model se lahko spreminja neodvisno od uporabnika
  - z modelom povezan pogled mora biti obveščen o spremembi, zato, da ve, da se mora prilagoditi
- model ima lahko različne poglede
  - vendar ima pogled lahko samo en model
  - vsi pogledi se morajo prilagodit ob spremembi modela