SKRIPTA IZ PREDMETA

INTERNET TEHNOLOGIJE

AJAX(Asynchronous JavaScript And XMl)

SADRŽAJ

1 Uvod

AJAX je skraćenica od “Asynchronous JavaScript and XML”. Termin AJAX prvi put je upotrebljen u februaru 2005. godine, kada je Džesi Džejms Garet (Jesse James Garret), dizajner informacionih sistema i direktor kompanije Adaptive Path, pokušao da nađe odgovarajuću skraćenicu za grupu tehnologija koju je predlagao svom klijentu. Tehnologije potrebne za postojanje AJAX -a su postojale i ranijih godina ali je glavni razlog za slabiju upotrebu svih tehnika korišćenih u AJAX-u, nedostatak podrške od strane web čitača. Mnogi su godinama pre nje koristile slične principe, poput Microsoftovog Remote Scriptinga ili veoma raširenog DHTML-a. Najbitnija stvar koja je uticala na brz razvoj AJAX-a jeste trenutak na tržištu u kojem su veliki igrači Internet industrije želeli da težište korišćenja računara prenesu sa desktopa na web stranice, za šta im je bila potrebna upravo tehnologija.

1.1 Potreba za AJAX tehnologijama

Web aplikacije imaju brojne prednosti u odnosu na desktop aplikacije. Web aplikacije dosežu do većeg broja ljudi, lakše se razvijaju, implementiraju i održavaju.

Najveći nedostatak Web aplikacija u odnosu na desktop aplikacije je taj što su desktop aplikacije „bogatije” u smislu funkcija i informacija koje nude. Međutim, taj nedostatak je u velikoj meri otklonjen uz pomoć AJAX tehnologije.

Web aplikacije su one aplikacije čijom funkcionalnošću se upravlja preko web servera i dostavljaju se korisnicima preko mreže kao što je internet ili intranet. I pored mnogostrukih prednosti, klasične web aplikacije imaju određene neostatke, kada je u pitanju interakcija sa korisnicima:

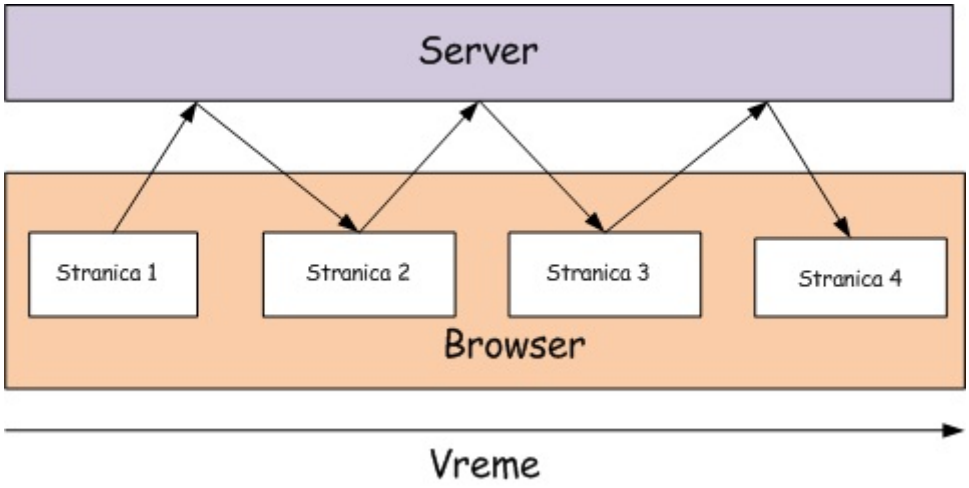
- Spor odgovor na zahteve
- Gubitak sadržaja prilikom ponovnog učitavanja stranice
- Gubitak informacija na ekranu
- Gubitak pozicije skrolovanja
- Bez trenutnog odgovora na korisničke akcije
- Korisnik mora da čeka na učitavanje sledeće strane
- Klikni, čekaj, refresh princip rada na stranici
- Stranica se ponovno učitaca sa servera za sve akcije korisnika
- Sinhroni način rada.

Pre svega, svaki put kada se učitava nova stranica, postoji određeno vreme kada aplikacija «stoji». Veoma često se prilikom malih izmena ili zahteva za delovima stranice, ponovo učitava cela stranica, iako je najveći deo nove stranice u potpunosti identičan sa prethodnom.

Da bi se shvatio koncept AJAX tehnologija, neophodno je objasniti razliku izmenu sinhronog i asinhronog prenosa podataka između klijenta i servera.

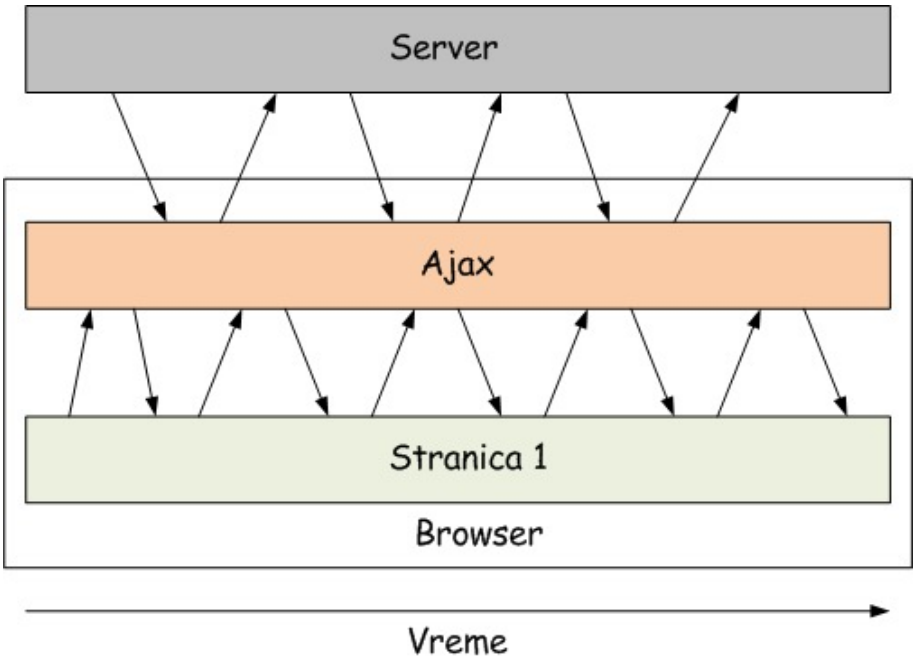
U sinhronom načinu rada procesi se izvode sekvencijalno. Izmedju dva procesa postoji jaz koji je jednak vremenu izvodjenja drugog procesa. Dakle, osnovni proces komunikacije izmedju klijenata i servera može se opisati na sledeći način: klijent šalje zahtev ka serveru, podaci se prenose ka serveru, server obrađuje podatke I zatim ih vraća klijentu. Ono što je

primetno sa slike dole je da postoji vremenski interval izmedju pravljenja zahteva i odgovora na isti. Za vreme tog intervala klijent čeka, bez mogućnosti da pravi nove zateve ili nastavi svoje korišćenje web aplikacije. Može se zaključiti da kod sinhronog prenosa podataka ne postoji mogućnost za eliminisanje čekanja zato što se procesi obavljaju jedan za drugim, čak i ukoliko za to nema potrebe.



Slika 1 Sinhroni prenos podataka

Kod asinhronog prenosa podataka izmedju klijenata i servera karakteristično je da se korisničko korišćenje web aplikacije nikada ne prekida. Za razliku od sinhronog režima rada, ovde korisnik može da nastavi korišćenje aplikacije nezavisno od toga hoće li sistem odgovoriti na njegov zahtev neposredno ili nešto kasnije. Ono što je uočljivo na prvi pogled su neprekidne linije koje označavaju aktivnosti korisnika i AJAX-ovog engine-a.



Slika 2 Asinhroni prenos podataka

Klasične aplikacije funkcionišu tako što šalju zahtev, server ih obrađuje i konačno vraća odgovor. Međutim, sve vreme rada servera i aplikcaija je «zamrznuta». Kada server kompletira rad, kao odgovoro korisnik dobija novu stranicu.

Sa asinhronom komunikacijom u pozadini se odvija komunikacija, a korisnik uopšte ne zna da se tako nešto dešava.

1.2 AJAX

AJAX predstavlja web razvojnu tehniku za kreiranje interaktivnih web sadržaja. AJAX tehnologija se zasniva na razmeni malih količina podataka sa serverom, kako korisnik ne bi stalno morao da osvežava stranicu. Na taj način se omogućava potpuna interaktivnost, efikasnost i poboljšava funkcionisanje stranice uopšte. AJAX nije tehnologija sam po sebi, već termin koji se odnosi na korišćenje grupe tehnologija.

AJAX obuhvata:

- prezentaciju baziranu na standardima koristeći XHTML i CSS;
- dinamički prikaz i interakciju preko Document Object Model-a;
- razmenu i manipulaciju nad podacima koristeći XML i XSLT;
- asinhrono prikupljanje podataka uz pomoć XMLHttpRequest-a koji čini jezgro;
- i JavaScript koji povezuje sve u celinu.

Ideja koja se krije iza AJAX-a jeste da se stranica na kojoj se odvija web aplikacija učita samo jednom, a da se svaka dalja komunikacija sa serverom sakrije od očiju korisnika i obavlja bez ponovnog učitavanja čitave stranice. Svaki prenos podataka između servera i klijenta (u slučaju AJAX-a to je browser) vrši se u pozadini. Jasno je da je ovo nemoguće izvesti statičkim HTML-om, pa tu na scenu stupa JavaScript. JavaScript je zadužen za komuniciranje sa serverom – slanje HTTP zahteva, prijem podataka sa servera i njihov prikaz na stranici, a i interakcija sa korisnikom postiže se korišćenjem ovog jezika.

Neki od osnovnih principa na kojima bi trebalo da se zasnivaju AJAX aplikacije:

- Minimalan protok - AJAX aplikacije bi trebalo da šalju i primaju sa servera što je moguće manje informacija
- AJAX aplikacije predstavljaju korisniku drugačije modele interakcije od tradicionalnih web aplikacija. Nasuprot “klik – čekaj“ prirodi standardnih web aplikacija, AJAX predstavlja modele interakcije koji su slični desktop aplikacijama. Bez obzira koji se model interakcije koristi, najbitnije je obezbediti konzistentnost kako bi korisnik znao šta treba sledeće da uradi.
- Izbegavaju se nepotrebni elementi na strani kao što su animacije ili delovi koji trepere.
- Izbegava se preuzimanje cele strane - celokupna komunikacija sa serverom, nakon učitavanja inicijalne web strane treba da bude zadatak AJAX engine –a

AJAX omogućava različite funkcionalnosti:

- Validacija podatka u realnom vremenu - Podaci na formama kao što su: user IDs, serial numbers, postal codes, i ostali koji zahtevaju validaciju od strane servera, mogu biti provereni pre submit- ovanja forme.
- Autocompletion i autosuggest - Određeni delovi podataka koji se unose od strane korisnika(posetioca): e-mail adresa, ime, grad i ostali se mogu automatski kompletirati, tj. dovršiti odmah nakon što se unesu samo početni delovi.

- Učitavanje na zahtev - U skladu sa iniciranim događajem od strane korisnika, HTML strana može povući veću količinu podataka u pozadini i omogućiti učitavanje kompletne strane mnogo brže.
- Sofisticiran korisnički interfejs, kontrole i efekti – Kontrole kao što su meniji, tabele podatka, tekst editori, kalendari, status bar-ovi I sl. Omogućavaju korisnicima bolju interakciju, bez potrebe za učitavanjem kompletne stranice.
- Osvežavanje podataka - HTML stranice povlače podatke sa servera i omogućavaju najsvežije podatke kao što su rezultati, cene akcija, vreme ili neke druge podatke iz specifičnih aplikacija.
- Delimičan submit – bez potrebe za učitavanjem cele forme.
- Stranica kao aplikacija – web stranice postaju slične desktop aplikacijama
- Iscrtavanje grafika – izveštaja na osnovu asinhronih upita nad bazom

Može se zaključiti da, za razliku od klasičnih web aplikacija, AJAX onemogućava korisničkom interfejsu direktnu komunikaciju sa serverom. Na taj način je moguće ostvariti n-slojnu arhitekturu u punom smislu te reči (u potpunosti je odvojen korisnički interfejs od ostatka sistema). AJAX engine je odgovoran za tu komunikaciju i on na određen način i ima odgovornost kontrolera korisničkog interfejsa, jer poziva server u skladu sa akcijama korisnika. HTTP request prenosi zahteve do servera dok se oni sa servera vraćaju u odgovarajućem obliku. AJAX engine može da ima u potpunosti ulogu kontrolera korisničkog interfejsa ukoliko se, kao što je i preporučljivo, unutar njega implementiraju i funkcije koje će da obrađuju korisničke akcije (pritisak miša, dugmeta i sl.). Zahtevi prema, kao i odgovori od strane servera, se ne moraju poklapati sa korisničkim zahtevima, već se odvijaju u bilo kom pogodnom trenutku. Browser se “ne zamrzava“ i ne čeka na odgovor servera. Umesto toga, korisnik je neprestano u mogućnosti da skroluje, pritiska miša, kuca na stranici.

AJAX aplikacija eliminiše start-stop-start-stop prirodu interakcije sa Webom uvodeći posrednika – AJAX engine – izmedju korisnika i servera. Izgledalo je da će se dodavanjem sloja samo usporiti aplikacija, ali to uopšte nije slučaj. Umesto učitavanja web strane, na startu sesije, browser učitava AJAX engine – napisanu u JavaScriptu i obično sklonjenu u neki skriveni frejm. AJAX engine je odgovoran za prenos podataka u oba smera, prikazivanje interfejsa koji korisnik vidi i komunikaciju sa serverom u ime korisnika. AJAX engine dozvoljava korisniku da se njegova interakcija sa aplikacijom odigrava asinhrono – nezavisno od komunikacije sa serverom.

Svaka korisnička akcija koja bi generisala HTTP zahtev, umesto toga dobija formu JavaSript poziva AJAX engine. Svaki odgovor na korisničku akciju koji ne zahteva odgovor od servera, kao što je prosta validacija podataka, izmena podataka u memoriji, čak i neka navigacija – engine vraća sama. Upotrebom AJAX-a mnoge od akcija, koje su karakteristične za desktop aplikacije, postaju dostupne i u web aplikacijama (*real-time* validacija podatka, *drag and drop*, i sl.)

Kao ilustracija, može se uzeti primer validacije forme. Neka je u pitanju forma u na kojoj korisnik pokušava da se registruje unoseći podatke. Da bi se utvrdila validnost podataka (npr. da li uneti grad postoji), moguće je izvršiti validaciju na dva načina. Prvo rešenje je izvršiti validaciju na korisničkoj strani, što otpada jer je veoma često za tako nešto neophodno preneti ogromnu količinu podataka na klijenta. Druga solucija je da korisnik submit-uje formu i čeka na odgovor. Međutim, AJAX omogućava boljebalansiranje protoka infomracija između klijentske i serverske strane Klijent i server komuniciraju u pozadini, dok korisnik radi na stranici. Istovremeno, dok korisnik kuca ime države, AJAX u pozadini poziva

server i učitava sve gradove iz te države, bez bilo kakve potrebe za ponovnim učitavanjem cele stranice. Korisnik dobija odgovor, automatski bez bilo kakvog «stajanja» aplikacije.

2 Dodavanje AJAX funkcionalnosti

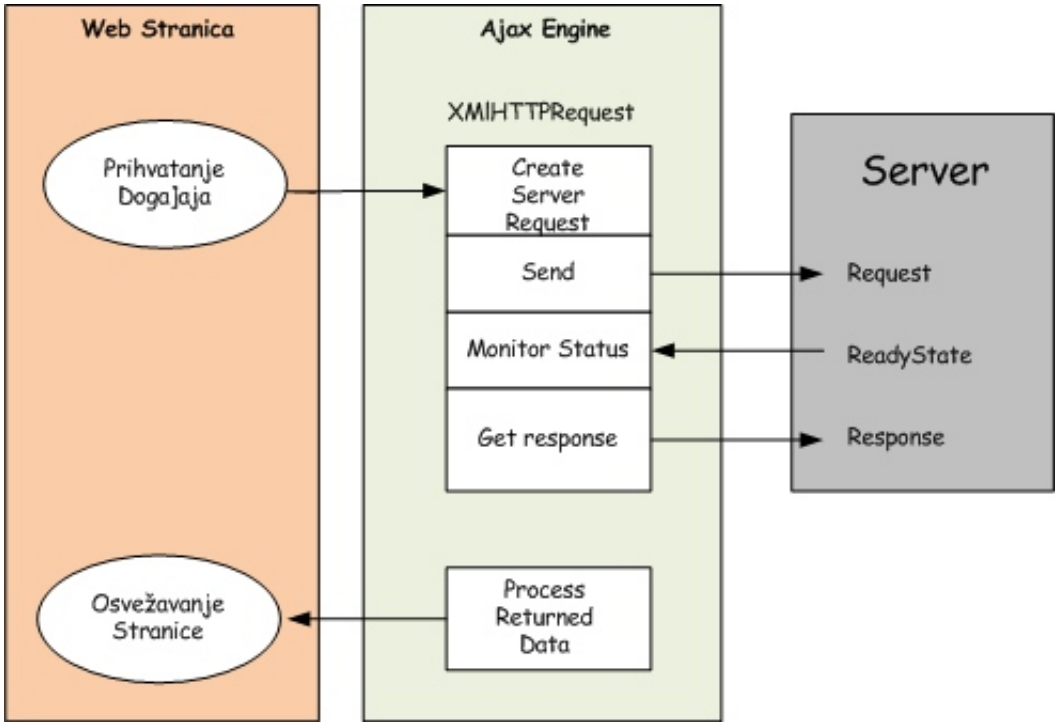
Prilikom dodavanja AJAX funkcionalnsoti u aplikaciji, neophodno je definisati koji događaj na strani ili korisnička akcija će biti odgovorna za pokretanje asinronog poziva i HTTP zahteva.

U AJAX aplikaciji, asinroni HTTP zahtev se kreira pomoćo objekta XMLHTTPRequest.

Nakon što se pošalje zahtev, neophodno je odrediti metod ili funkciju koja će pratiti stanje i prihvatiti odgovor sa servera.

Konačno, nakon pristizanja odgovora sa servera, dolazi do promena na stranici korisnika (ispis teksta, popunjavanje polja, promena položaja elemenata i sl.)

Na slici 3 je opisano kako funkcioniše AJAX aplikacija.



Slika 3 Funkcionisanje AJAX aplikacije

U klasičnim JavaScript aplikacijama, ako je potrebno preuzeti informacije iz baze podataka ili sa Web servera, ili poslati informacije serveru, mora se napraviti HTML forma i koristiti GET ili POST metode. Korisnik mora da klikne na „Submit“ dugme da bi poslao ili preuzeo informacije sa servera, a pri tom je neophodno da čeka odgovor servera. Tek nakon odgovora, nova Web strana se učitava u korisnikov browser.

Baš zbog toga što server vraća rezutate u vidu nove web strane svaki put kada korisnik klikne na određeno dugme, klasične web aplikacije su sporije i manje user-friendly. Pomoću

AJAX-a, JavaScript uspostavlja direktnu komunikaciju sa serverom preko JavaScript XMLHttpRequest objekta i tako AJAX aplikacije postaju znatno brže. Sa HTTP zahtevom, browser može da pošalje zahtev serveru i primi odgovor od servera bez ponovnog učitavanja (reload, refresh) Web strane. Korisnik pri tom i dalje ostaje na istoj strani, i ne primećuje da se u pozadini odvija browser-server komunikacija.

XMLHttpRequest je objekat koji omogućava JavaScript- u da načini asinhroni HTTP zahtev prema serveru. Različiti browser-i koriste različite metode za stvaranje XMLHttpRequest objekta. Internet Explorer koristi ActiveXObject[1] metodu, dok ostali browser-i koriste ugrađenu (built-in) JavaScript metodu XMLHttpRequest.

Pri kreiranju ovog objekta, da bi se omogućila kompatibilnost s različitim browser-ima, u JavaScript kodu, najčešće se koristi „try-catch“ blok. U nastavku, dat je prikaz JavaScript koda u kome se kreiraju pomenute metode u okviru „try-catch“ bloka.

```
<html>
<body>
<script type="text/javascript">
function ajaxFunction()
  {
  var xmlHttp;
  try
    {
    // Firefox, Opera 8.0+, Safari
    xmlHttp=new XMLHttpRequest();
    }
  catch (e)
    {
    // Internet Explorer
    try
      {
      xmlHttp=new ActiveXObject("Msxml2.XMLHTTP");
      }
    catch (e)
      {
      try
        {
        xmlHttp=new ActiveXObject("Microsoft.XMLHTTP");
        }
      catch (e)
        {
        alert("Your browser does not support AJAX!");
        return false;
        }
      }
    }
  }
```

Kao što se iz koda vidi, prvo je potrebno definisati promenljivu, koja je u ovom slučaju nazvana *xmlHttp* koja čuva XMLHttpRequest() i ActiveXObject() metode. Zatim se kreira prva metoda naredbom *XMLHttp=new XMLHttpRequest().* Ova metoda se koristi u *Firefox, Opera* i *Safari* browser-ima. Ako korisnikov pretraživač ne uspe da interpretira ovu naredbu, onda se definiše ActiveXObject() metoda naredbom *xmlHttp=new ActiveXObject(“Msxml2.XMLHTTP”)* koja je „čitljiva“ *Internet Explorer 6.0*+ verziji. Ako i ova metoda nije razumljiva od strane korisnikovog

[1] ActiveX je Microsoft technology za omogućavanje korišćenja active objects u aplikacijia

pretraživača, onda se naredbom *xmlHttp=new AcriveXObject("Microsoft.XMLHTTP")* pravi još jedna metoda koja je podržana od strane *Internet Explorer 5.5*+ verzije. Ako ni jedna od ove tri metode ne prikaže rezultate, onda je korisnikov pretraživač veoma star (outdated) i korisnik će videti poruku na ekranu da njegov *Browser* ne podržava AJAX.
Na sličan način se može koristiti i objekat *window:*

```
function getXMLHTTPRequest()
{
var request = false;
if(window.XMLHTTPRequest)
   {
   request = new XMLHTTPRequest();
   } else {
   if(window.ActiveXObject)
     {
     try
         {
         request = new ActiveXObject("Msml2.XMLHTTP");
         }
     catch(err1)
         {
         try
              {
                 request = new ActiveXObject("Microsoft.XMLHTTP");
              }
         catch(err2)
              {
              request = false;
              }
         }
     }
   }
return request;
}
```

Pri pravljenju AJAX aplikacije, najbolja praksa je da se uvek otkuca ovaj kod, da bi aplikacija bila čitljiva u većini *Browser*-a.

Postoje tri bitna svojstva (properties) XMLHttpRequest objekta:

- **onreadystatechange;**
- **readyState;**
- **responseText.**

Atribut	Opis
Onreadystatechange	Obrađuje događaj koji se ostvaruje svaki put kada se promeni stanje objekta. Prilikom svake promene stanja poziva odgovarajući deo koda koji vrši obradu.
Readystate	Status objekta, integer
Response Text	String verzija podatka vraćenog sa servera
ResponseXML	DOM kompatibilan document objekat vraćen sa servera

Status	Numerička vrednost vraćena sa servera, kao što je 404 za Not Found ili 200 za OK
Status Text	String poruka za status kod

Tabela 1. Pregled osnovnih atributa XMLHttpRequest objekta

Nakon što browser pošalje određeni zahtev serveru, potrebno je definisati funkciju koja može da primi podatke koji su poslati browser-u sa servera. ***Onreadystatechange*** svojstvo čuva funkciju koja obrađuje odgovor servera. Sledeći kod definiše praznu funkciju i postavlja ***onreadystatechange*** svojstvo u isto vreme:

```
xmlHttp.onreadystatechange=function()
{
// neki kod
}
```

readyState svojstvo čuva status odgovora servera. Svaki put kada se ***readyState*** svojstvo promeni, *onreadystatechange* funkcija će biti izvršena. Prikaz mogućih vrednosti *readyState* svojstva, dat je u tabeli 5.

Stanje	Opis
0	Zahtev je inicijalizovan
1	Zahtev je postavljen
2	Zahtev je poslat
3	Zahtev se obrađuje
4	Zahtev je kompletiran (završen)

Tabela 2. Moguće vrednosti readystate svojstva

Atribut ***readystate*** predstavlja trenutno stanje objekta. Preko ovog objekta se prate stanja, i u skladu sa tim preuzimaju odredjene akcije. Naravno, najbitnije je stanje 4 koje ukazuje na to da se izvršila transakcija.

Sledeći kod prikazuje upotrebu onreadystatechange i readyState svojstva gde se testira da li je zahtev koji je upućen serveru završen.

```
xmlHttp.onreadystatechange=function()
{
If (xmlHttp.readyState==4)
// zahtev je kompletiran
}
}
```

Pristup podacima koji su vraćeni sa servera omogućava da uz pomoć atributa ***Response Text*** ili ***ResponseXML***. Atribut ***Response Text*** vraća podatak u string formatu, međutim, mnogo veće mogućnosti pružaju podaci u XML obliku koje obezbeđuje atribut ***ResponseXML***. Sledeći kod prikazuje da ako je zahtev završen, preuzima se tekuće vreme sa servera.

```
xmlHttp.onreadystatechange=function()
{
If(xmlHttp.readyState==4)
{
document.time.value=xmlHttp.responseText;
```

```
}
}
```

Da bi se poslao zahtev serveru, koriste se dve metode: open() i send(). Open() metoda uzima tri argumenta. Prvi argument definiše način slanja zahteva serveru (GET ili POST). Drugi argument definiše URL (Uniform Resource Locator) skripte serverske strane, dok treći argument ukazuje na to da bi zahtevom trebalo upravljati asinhronim prenosom podataka.

GET:

```
xmlHttp.open("GET", "http://localhost/ajax/test.php?param1=x&param2=y", true);
xmlHttp.onreadystatechange = handleRequestStateChange;
xmlHttp.send(null);
```

POST:

```
xmlHttp.open("POST", "http://localhost/ajax/test.php", true);
xmlHttp.onreadystatechange = handleRequestStateChange;
xmlHttp.send("param1=x&param2=y");
```

Dva prikazana koda imaju isti efekat. U praksi, GET metoda može da pomogne prilikom debug – ovanja, jer se jasno može ispratiti zahtev. POST metoda se koristi u slučajevima kada se prenosi više od 512 bajta.

Send() metoda sprovodi sam proces slanja zahteva serveru i ona najčešće prima argument null. Uz pretpostavku da se HTML fajl i fajl skripte serverske strane nalaze u istom direktorijumu, dve opisane metode bi mogle izledati ovako:

```
xmlHttp.open(“GET”, “vreme.php”, true);
xmlHttp.send(null);
```

U nastavku je dat primer AJAX koda u kojem korisnik u text box unosi svoje ime. Čim korisnik unese svoje ime, u drugom text box-u, se prikazuje vreme koje je preuzeto iz skripte serverske strane. Treba obratiti pažnju da ne postoji „submit“ dugme na koje korisnik klikne, da bi se generisalo vreme.

2.1.1 ajaxPrimer.html

```
<html>
<body>
<script type="text/javascript">
function ajaxFunction()
  {
  var xmlHttp;
  try
    {
    // Firefox, Opera 8.0+, Safari
    xmlHttp=new XMLHttpRequest();
    }
  catch (e)
    {
    // Internet Explorer
    try
      {
```

```
      xmlHttp=new ActiveXObject("Msxml2.XMLHTTP");
      }
    catch (e)
      {
      try
        {
        xmlHttp=new ActiveXObject("Microsoft.XMLHTTP");
        }
      catch (e)
        {
        alert("Your browser does not support AJAX!");
        return false;
        }
      }
    }
    xmlHttp.onreadystatechange=function()
      {
      if(xmlHttp.readyState==4)
        {
        document.mojaForma.vreme.value=xmlHttp.responseText;
        }
      }
    xmlHttp.open("GET","vreme.php",true);
    xmlHttp.send(null);
  }
</script>
<form name="mojaForma">
Ime: <input type="text" onkeyup="ajaxFunction();" name="ime" />
</br>
Vreme se ispisuje bez ponovnog ucitavanja stranice
</br>
Vreme: <input type="text" name="vreme" />
</form>
</body>
</html>
```

Kod fajla ***vreme.php***, tj. kod skripte serverske strane izgleda ovako:

```
<?php
 echo date(" G:i:s", time());

?>
```

Ako se pogleda kod u fajlu ***ajaxPrimer.html***, može se videti da je prvo napisan kod koji omogućava kompatibilnost AJAX aplikacije sa različitim browser-ima. Metode open() i send() šalju zahtev serveru. Open() metoda uzima kao argumente metodu GET, naziv skripte serverske strane (***vreme.php***) i argument true. Posle se definiše ***onreadystatechange*** svojstvo i proverava se da li ***readyState*** svojstvo ima vrednost 4, tj. da li je zahtev završen. Ako je zahtev završen, onda ***responseText*** svojstvo preuzima podatake sa servera i prikazuje te podatke na ekranu.

2.1.2 Problem sa keširanjem stranica

Skoro svi browser- i održavaju takozvani *keš* posećenih stranica – lokalnu kopiju sadržaja stranice skladištene na hard disku browser-ovog kompjutera. Kada se zahteva ta stranica, browser prvo pokušava da je pročita iz njenog *keša,* a tek kasnije šalje novi HTTP zahtev. Ovo je posebno značajan problem sa *Inernet Explorer*- om. Iako, pomenuto keširanje može izgledati kao prednost u kontekstu brzine učitavanja stranica, kod pisanja AJAX aplikacija

nastaje poteškoća. AJAX kontaktira sa serverom, ne čita podatke iz *keša*. Kada se kreira asinhroni zahtev ka serveru, svaki put mora biti formiran nov HTTP zahtev. U slučaju GET metode, u zahtevu se na kraju dodaje još jedan parametar. Ako jedan deo URL zahteva svaki put generiše drugu vrednost, to "zavara" browser, tako da svaki put šalje HTTP zahtev na drugu adresu, a ne poziva keširanu stranicu. Koristi se Math objekat i njegova funkcija *random()*. Kod PUT metode ovaj problem ne postoji.

```
function callAjax() {

var ime = 'ITEH';
// kreiraj zahtev za server
var url = "myserverscript.php?imekorisnika=" + ime;
// generiši slučajan broj
var myRandom=parseInt(Math.random()*99999999);
// kreiraj zahtev za asinhronim pozivom
myRequest.open("GET", url + "&rand=" + myRandom, true);
// kreiraj funkciju za obradu odgovora sa servera
myRequest.onreadystatechange = responseAjax;
// pošalji zahtev
myRequest.send(null);
}
```

3 PRIMERI KORIŠĆENJA AJAXA IZ PRAKSE[2]

3.1 Google Suggest

Interfejs Google-ovog AJAX rešenja je jednostavno klon glavnog interfejsa koji se sastoji od textbox-a u koji se unosi termin koji se traži. Sve izgleda isto dok ne počne da se kuca u textbox-u. Kako se unose jedno slovo Google Suggest traži sugestiju sa servera i prikazuje predlog, tj.listu termina koje korisnici možda traže. Svaka sugestija se prikazuje sa brojem rezultata koji su dostupni za dati pojam. Pošto se server poziva asinhrono nakon unošenja svakog simbola, lista predloga se dinamički menja bez bilo kakvog čekanja ili ponovnog učitavanja stranice.

2

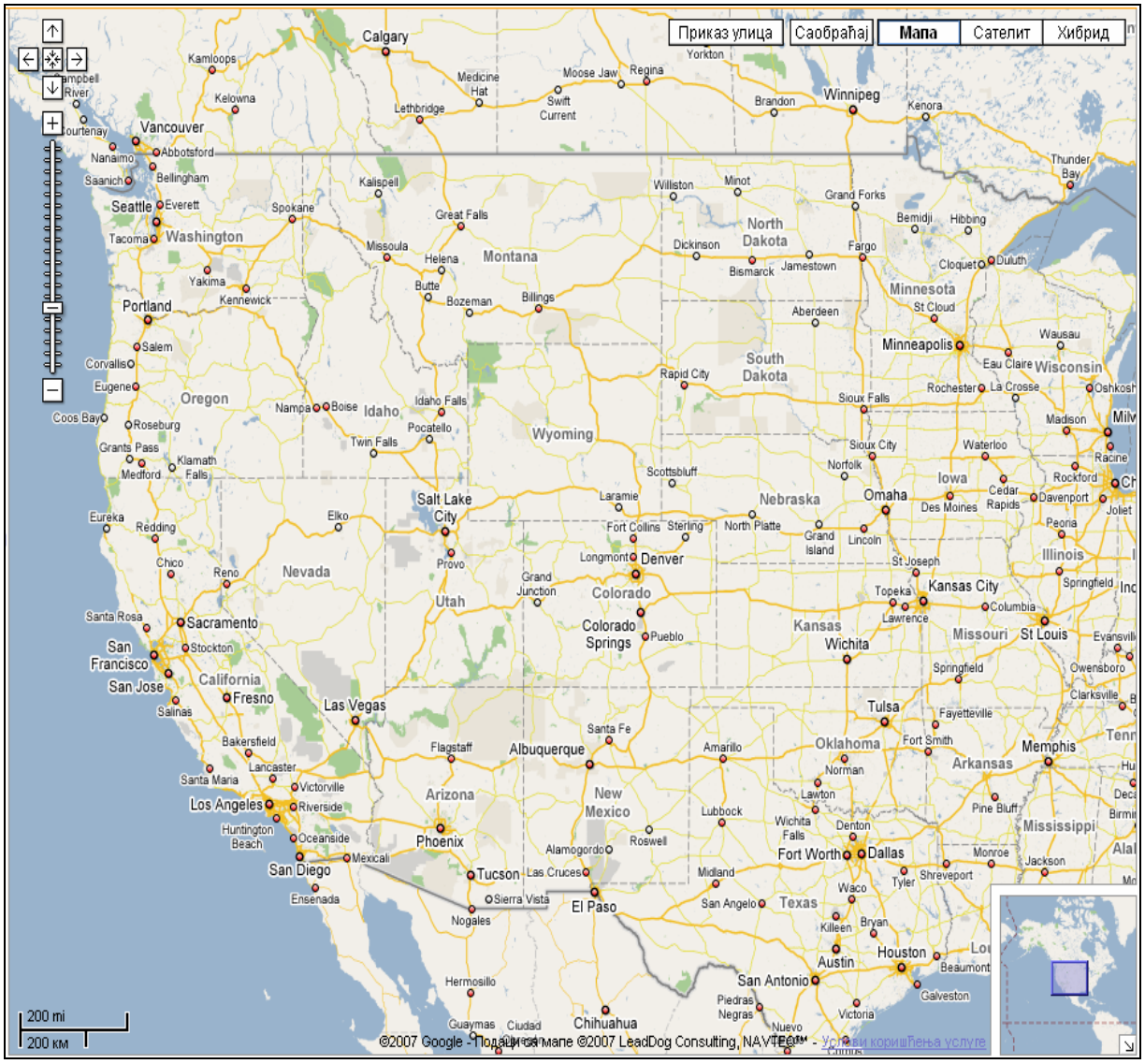
Slika 4 Google suggest

3.2 Gmail

Gmail predstavlja e–mail servis kompanije Google. Kada se uloguje na Gmail, engine korisničkog interfejsa je učitava u jedan od nekoliko iframe-ova koje aplikacija koristi. Svi naredni zahtevi ka serveru se obavljaju kroz ovaj interfejs koristeći XMLHttpRequest objekat. Podaci koji se prenose su u okviru JavaSkript koda, koji služi za brzo izvršavanje kada se jednom preuzme uz pomoć čitača. Zahtevi određuju šta će se sledeće prikazati na interfejsu.

3.3 Google Maps

Google Maps omogućava kretanje kroz mape gradova u različitim smerovima bez ponovnog učitavanja glavne stranice. Celokupna mapa je podeljena na niz slika koje su povezane tako da daju neprekidnu sliku. Kad se zatraži nova putanju na mapi, klijent - server aplikacija se učitava pomoću skrivenog frejma. Podaci se vraćaju u XML formatu i prosledjuju se JavaSkript funkciji (AJAX engine) da ih obradi. Svaki put kad se klikne na GMarker da otvori Info Window ustvari se šalje AJAX zahtev ka serveru. Treba uočiti da se ne osveži ceo ekran.



Slika 5 Google Maps

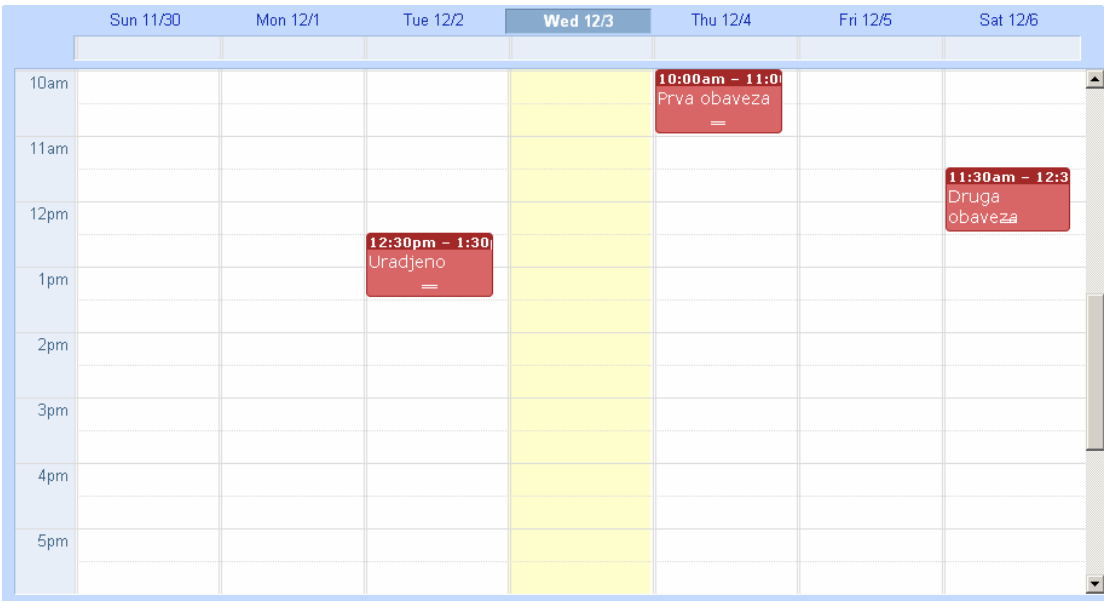
3.4 Ostali primeri

Treba istaći činjenicu da danas postoje milioni web aplikacija i web sajtova koji sadrža različite tipove AJAX funkcionalnosti. AJAX je jedan od centralnih koncepata prihvaćenih u Web 2.0 standardu.

Na sledećoj slici je prikazan primer drag and drop funkcionalnosti na webu

Slika 6 Drag and drop

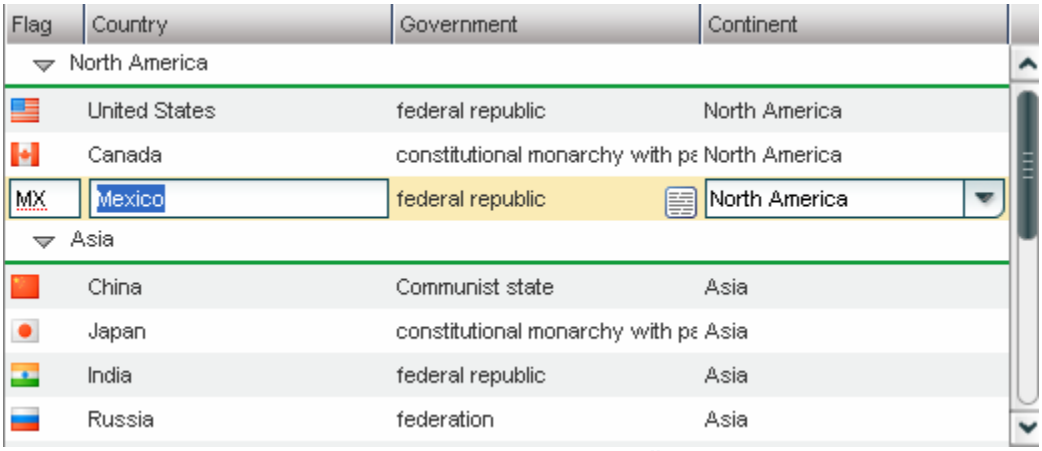
Ova funkcionalnost je svojstvena desktop aplikacijama, ali pojavom AJAX tehnologija, omogućena je i u okviru web aplikacija. Jedan od primera za gde je moguće obavljati drag and dropa operacije je i Google calendar.



Slika 7 Google Calendar

Validacija forme o kojoj je bilo reči se može videti na velikom broju sajtova prilikom registracije i sličnih aktivnosti

Asinhrono brisanje, ubacivanje i ažuriranje delova stranice ili redova u koloni, bez ponovnog učitavanja čitave stranice, je jedan od najbitnijih zahteva od strane korisnika. Ove funkcionalnsoti su na primer implementirane na sajtovima poput www.facebook.com, www.wordpress.com i sl. Na slici je prikazan jedan od primera.



Slika 8 Primer za update i delete reda bez učitavanja nove stranice

Pored navedenih primera u ovoj oblasti, između ostalog pogledati na:

http://www.kayak.com/
http://www.metak.com/
http://www.planplus.co.yu/
http://www.xscores.com/
http://livesearch.goatstone.com/