UNIVERZA V LJUBLJANI
FAKULTETA ZA MATEMATIKO IN FIZIKO

Matematika – Praktična matematika (VSŠ)

Aleksandra Sandra Perko

Uporaba tehnike AJAX pri gradnji spletnih strani

Diplomska naloga

Ljubljana, 2008

DIPLOMSKA NALOGA :
FAKULTETA ZA MATEMATIKO IN FIZIKO

Zahvala

Hvala mentorju za pomoč in napotke pri pripravi diplomske naloge.
Iskrena hvala Mateju za izjemno podporo, pomoč in razumevanje.
Hvala mami za vse.
Hvala Tomu za pomoč pri oblikovanju grafičnih elementov.

Kazalo vsebine

DIPLOMSKA NALOGA :
FAKULTETA ZA MATEMATIKO IN FIZIKO

Program diplomske naloge

V diplomski nalogi predstavite sestavljanje spletnih aplikacij s pomočjo tehnike AJAX.

Osnovna literatura:

- Brett McLaughlin: *Head Rush Ajax*, O'Reilly Media, Inc., 2006

Mentor:
Mag. Matija Lokar

DIPLOMSKA NALOGA :
FAKULTETA ZA MATEMATIKO IN FIZIKO

Povzetek

Diplomska naloga zajema predstavitev tehnike AJAX.

V uvodu, ki je tudi prvo poglavje, na kratko predstavimo tehnologije, ki jih tehnika AJAX združuje.
Poglavja, ki sledijo, so namenjena podrobnejšemu vpogledu v tehniko AJAX in v princip njenega delovanja.
V drugem poglavju, "Spletne aplikacije nove generacije", se tako osredotočimo na razlike med spletnimi aplikacijami, ki sledijo klasičnemu modelu in tistimi, ki sledijo modelu AJAX. To poglavje zajema tudi nekaj zgledov, kjer so nakazane prednosti modela AJAX.
Tretje poglavje, "Primerjava klasičnega modela in modela AJAX", zajema razvoj enostavne spletne aplikacije po obeh modelih. Spletno aplikacijo smo najprej razvili po klasičnem modelu, nato pa smo skozi poglavje prikazali prehod na model AJAX.
V četrtem poglavju, "Zahteva POST", smo spletno aplikacijo, ki smo jo razvili v tretjem poglavju, nadgradili.
Peto poglavje, "Knjižnice", je namenjeno pregledu nekaterih knjižnic, katerih uporaba nam lahko zelo olajša delo. Poglavje je namenjeno predvsem kot vzpodbuda za nadaljnje raziskovanje in razmišljanje bralca o možnostih, ki jih ponuja tehnika AJAX.

Math. Subj. Class. (2000): 68N01, 68N19

Ključne besede: AJAX, tehnika programiranja, interaktivne spletne aplikacije, asinhrona komunikacija, zahteva HTTP, XMLHttpRequest

Keywords: AJAX, web development techniques, iteractive web applications, rich internet applications, asynhronous communication, HTTP request, XMLHttpRequest

1.Uvod

Diplomska naloga, ki je pred vami, predstavlja uporabo tehnike AJAX pri gradnji spletnih aplikacij[1]. V diplomski nalogi smo dali poudarek predvsem razlagi osnovnega principa delovanja tehnike AJAX. Po poglavjih, ki sledijo, bomo tako spoznali razloge za razvoj tehnike AJAX, njene lastnosti in prednosti.
Pri razumevanju nam bodo ves čas v pomoč praktični primeri.

Da pa bi lahko začeli z vpogledom v tehniko AJAX, si moramo najprej razjasniti nekaj pojmov, katerih razumevanje je pomembno za poglavja, ki sledijo.

JavaScript
JavaScript je skriptni programski jezik. Z razvojem jezika JavaScript je začelo podjetje Netscape Communications za potrebe svojega spletnega brskalnika[2] Navigator. Netscape je bil prvi izdelovalec, ki je poleg podjetja Sun, v svoj spletni brskalnik vgradil podporo izvajanju javanske kode.
Skriptni jezik JavaScript je nekaj drugega kot programski jezik java. Z jezikom JavaScript dokumenti HTML pridobijo na dinamiki in interaktivnosti. O pomenu teh izrazov bomo več povedali v nadaljevanju.
Programska koda, napisana v jeziku JavaScript, se ne prevaja v vmesno obliko, ampak se vedno prenese na odjemalca v obliki izvorne kode. Izvajanje kode JavaScript je naloga spletnega brskalnika. To pomeni, da za razvoj kode JavaScript ne potrebujemo posebnih razvojnih orodij. Za razvoj nam zadošča enostaven urejevalnik besedila in spletni brskalnik.

PHP
PHP ali "PHP: Hypertext Preprocessor" je skriptni programski jezik, ki se izvaja na strani strežnika[3] in nam omogoča razvoj dinamičnih in interaktivnih spletnih aplikacij. Strežnik kot rezultat izvajanja kode PHP vrne spletno stran. Sama koda PHP uporabniku ni vidna. Uporabnik vidi le rezultat obdelave, ki se je zgodila na strežniku. Datoteke PHP imajo končnico php, php3 ali phtml.

DOM
Z jezikom JavaScript lahko dostopamo do elementov, ki so del spletne aplikacije in so uporabniku vidni preko spletnega brskalnika. Te elemente lahko tudi spreminjamo. Notranjo strukturo, torej pregled in povezave med elementi strani, nam omogoča DOM ali Document Object Model. DOM definira objekte in lastnosti vseh elementov dokumenta HTML in metode, ki nam omogočajo dostop do teh elementov in njihovih lastnosti.

DOM celoten dokument HTML vidi kot podatkovno strukturo drevo. Vsak element, atribut in tekst dokumenta HTML vidi kot vozlišče.
Oglejmo si pravila, kaj so vozlišča v DOM-u:
- Vsaka značka HTML je vozlišče.
- Vsebina znotraj značke HTML je vozlišče.
- Vsak atribut značke HTML je vozlišče.
- Komentarji so vozlišča.

Za boljšo predstavo si poglejmo primer enostavnega dokumenta HTML in pripadajoče drevo DOM.

```
1.<html>
2.   <head>
3.      <title>DOM</title>
4.   </head>
5.   <body>
6.      <a href="http://www.google.com/">povezava</a>
7.      <h1>primer 1</h1>
8.   </body>
9.</html>
```

Datoteka 1: Enostaven primer dokumenta HTML

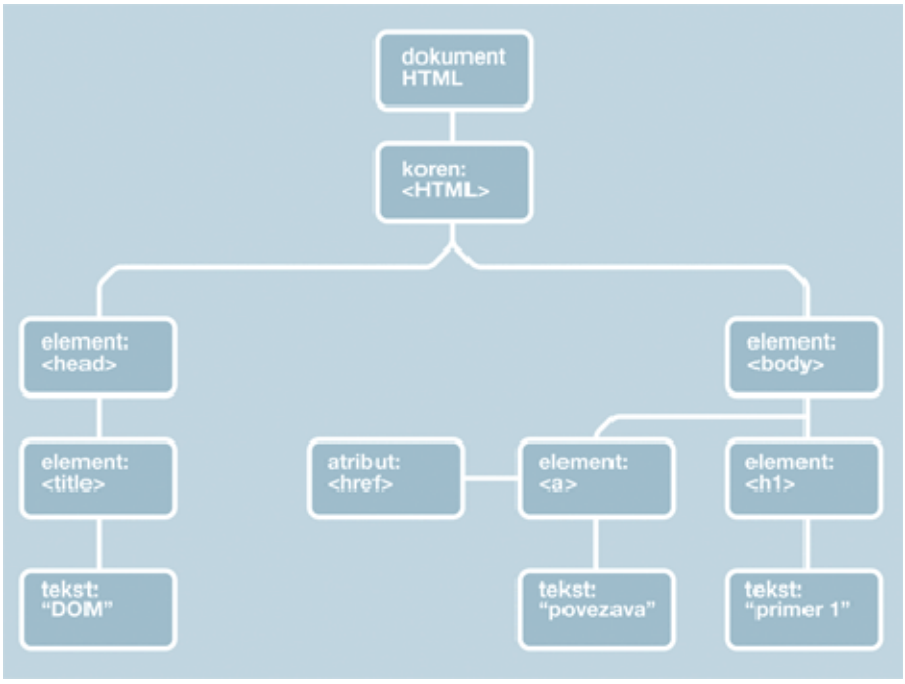
---

1 *Spletna aplikacija* je aplikacija, ki je uporabniku dostopna preko spletnega brskalnika2. Med spletne aplikacije štejemo tudi spletne strani.

2 *(Spletni) brskalnik* je računalniški program, ki omogoča pregledovanje vsebin po svetovnem spletu. Primeri spletnih brskalnikov so Internet Explorer (IE), Mozilla Firefox, Safari, Opera, Konqueror in drugi.

3 *Strežnik* je računalnik, ki je povezan s svetovnim spletom in na katerem so shranjene vse datoteke posamezne spletne strani. Njegova naloga je posredovanje podatkov spletnemu brskalniku. Podatke lahko pred posredovanjem tudi ustrezno obdela.

Dokument HTML, naveden v datoteki Datoteka 1, se po pravilih DOM-a prevede v drevesno podatkovno strukturo, ki je grafično prikazana na sliki Slika 1.



Slika 1: Drevo DOM

Iz grafičnega prikaza na sliki Slika 1 je razvidno, da je koren drevesa, t.j. vozlišče brez predhodnika, značka <html>. Vsa ostala vozlišča so znotraj značke <html>.
Vozlišče <head> vsebuje vozlišče <title>. Vozlišče <body> vsebuje vozlišči <h1> in <a>. Vozlišče <a> ima dva sinova, t.j. svoj atribut (href) in vsebino ("povezava").

DOM nam definira, kako dokumente HTML prevedemo v drevo. S pomočjo funkcij, ki jih definira DOM, lahko nato z uporabo jezika JavaScript dostopamo do vozlišč tega drevesa.
Vsako vozlišče je predstavljeno kot objekt, ki ima svoje lastnosti in metode (funkcije).

Če v drevesu DOM, ki predstavlja naš dokument HTML, karkoli spremenimo, se to takoj odrazi tudi v prikazu v spletnem brskalniku. Zato uporabniku ni potrebno s posebnim ukazom strani osveževati. To prednost, osveževanje podatkov brez zahteve po osveževanju celotne spletne strani, bomo v enem od prihodnjih poglavij uporabili tudi sami.

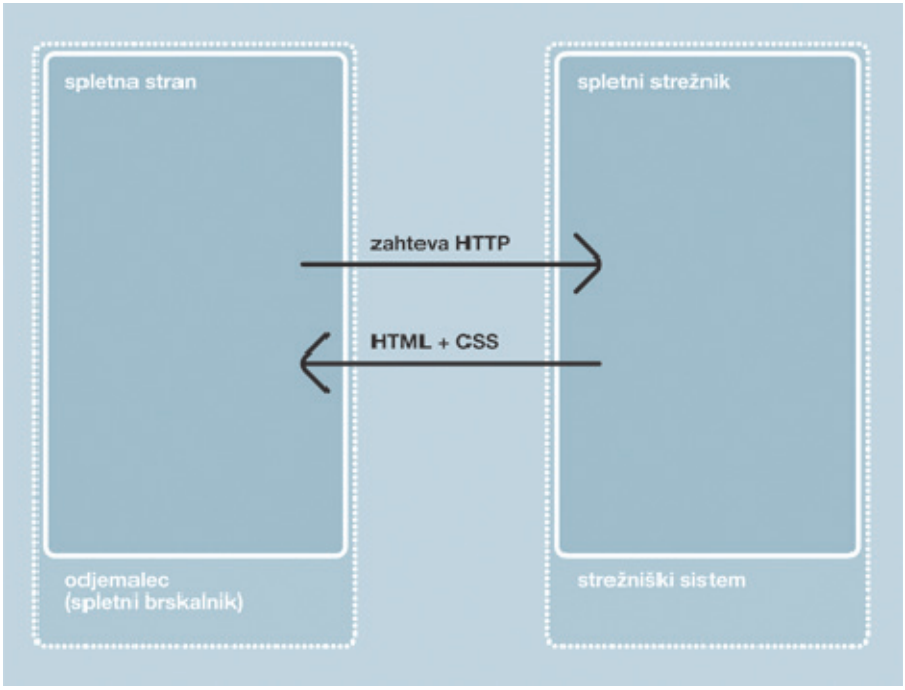
2.Spletne aplikacije nove generacije

V programskem jeziku JavaScript je že vrsto let na razpolago objekt XMLHttpRequest, ki s svojimi lastnostmi postavlja osnovo za gradnjo zanimivih spletnih aplikacij. Vendar so se šele v zadnjem času pojavile spletne aplikacije, podprte z jezikom JavaScript, ki izrabljajo možnosti, ki jih ta objekt XMLHttpRequest nudi. Do povečanega zanimanja za ta objekt je prišlo, ko so se pojavile spletne aplikacije podjetja Google, kot na primer Suggest, Maps in Gmail, ki uporabljajo zmožnosti objekta XMLHttpRequest. Uporaba tega objekta je tudi osnova glavnih funkcionalnosti zelo priljubljene spletne strani za izmenjavo slik Flickr.

Zakaj je objekt XMLHttpRequest tako poseben? Na kratko zato, ker omogoča rešitev ene največjih neprijetnosti spletnih aplikacij – počasnega odzivanja spletne aplikacije na dejanja uporabnika.

2.1.Klasični model spletne aplikacije

Pred pojavom spletnih aplikacij, ki izrabljajo možnosti, ki jih nudi objekt XMLHttpRequest, so spletne aplikacije v splošnem delovale na način, ki ga bomo imenovali klasični model spletne aplikacije. Na sliki Slika 2 je klasični model spletnih aplikacij predstavljen grafično.



Slika 2: Klasičen model spletne aplikacije

Pri spletnih aplikacijah, ki sledijo klasičnemu modelu, dejanja uporabnika sprožijo pošiljanje zahteve na strežnik preko protokola HTTP[4]. Primer dejanja uporabnika je klik na gumb, ki je del spletne aplikacije. Ker se zahteva pošlje preko protokola HTTP, jo imenujemo tudi zahteva HTTP.

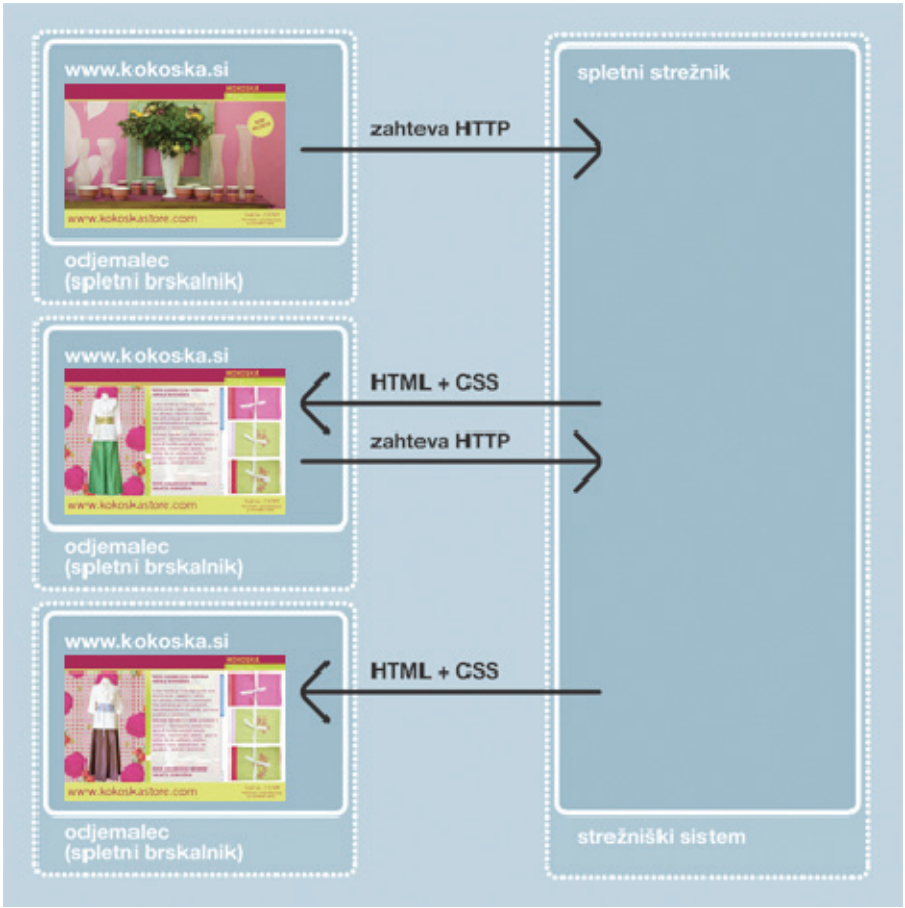
Zahtevo strežniku pošlje spletni brskalnik, ki ga uporabnik uporablja za prikaz spletne aplikacije. Strežnik prejeto zahtevo obdela in spletnemu brskalniku vrne odgovor.

Vsebino in strukturo spletne aplikacije podamo v obliki datoteke HTML, vizualna podoba spletne aplikacije pa je v obliki datoteke CSS. Odgovor strežnika na zahtevo je običajno nova spletna aplikacija. Zato bomo v nadaljevanju, kadar bo strežnik kot odgovor na zahtevo vrnil novo spletno aplikacijo, zapisali, da je odgovor oblike HTML+CSS.

[4] *HTTP* (HyperText Transform Protocol) določa način prenosa informacij po svetovnem spletu.

Pri klasičnem modelu spletnih aplikacij je izmenjava podatkov med spletnim brskalnikom in strežnikom zaporedna. Za zaporedno izmenjavo podatkov pravimo tudi, da je sinhrona. To pomeni, da mora spletni brskalnik preden pošlje novo zahtevo vedno počakati, da strežnik vrne odgovor na zahtevo, ki jo trenutno obdeluje. Če čaka spletni brskalnik, čaka tudi uporabnik, ki za prikaz spletne aplikacije ta spletni brskalnik uporablja. Iz tega sledi, da je spletna aplikacija, ki sledi klasičnemu modelu, slabo odzivna. Ker spletni brskalnik pred pošiljanjem nove zahteve vedno počaka na odgovor na zahtevo, ki jo strežnik trenutno obdeluje, bo strežnik istočasno vedno obdeloval le eno zahtevo.
Odgovor strežnika je v primeru klasičnega modela spletnih aplikacij vedno v obliki HTML+CSS. To pomeni, da spletni brskalnik prejeti odgovor strežnika vedno prikaže kot novo spletno aplikacijo. V nekaterih primerih to ni smiselno. Velikokrat se spletna aplikacija, ki je uporabniku vidna pred pošiljanjem zahteve in tista, ki jo spletni brskalnik prejme in prikaže kot odgovor na zahtevo, razlikujeta le za malenkost. V takih primerih bi bilo bolj smiselno osvežiti le tiste dele spletne aplikacije, kjer so se podatki dejansko spremenili. Za spletno aplikacijo, ki se osvežuje le z novimi podatki in ne v celoti, pravimo, da je dinamična. Dinamike pri klasičnem modelu spletnih aplikacij ne moremo doseči. Razlog za to je oblika, v kateri strežnik pošlje odgovor na našo zahtevo. Kot smo že navedli, je odgovor vedno v obliki HTML+CSS, kar pa lahko spletni brskalnik prikaže le kot povsem novo spletno aplikacijo.

Za boljše razumevanje si na primeru spletne aplikacije, ki se nahaja na naslovu http://www.kokoska.si/, poglejmo, kako deluje klasičen model spletne aplikacije. Iz grafičnega prikaza na sliki Slika 3 sta razvidni dve glavni pomanjkljivosti klasičnega modela spletnih aplikacij, to sta slaba odzivnost in pomanjkanje dinamike.



Slika 3: Primer spletne aplikacije po klasičnem modelu

Iz grafičnega prikaza na sliki Slika 3 je razvidno, da vsako dejanje zahteva/odgovor pomeni, da se celotna spletna aplikacija prikaže na novo. To se zgodi tudi, če se je spremenilo le malo vsebine. Uporabnik je med posameznimi dejanji prisiljen čakati, še posebej, če je stran precej zapletena in mora brskalnik opraviti veliko dela. Če je med stanjem 1 in stanjem 2 pošiljanje celotne nove strani in njen prikaz bilo še nekako smiselno, saj sta spletni strani povsem drugačni, pa se je med stanjem 2 in 3 spremenil le delček strani (glej levi del grafičnega prikaza na sliki Slika 3). Zato bi bilo bolj smiselno osvežiti le del strani, kjer se sprememba vsebine dejansko zgodi.

2.2. Spletna aplikacija po modelu AJAX

Če želimo, da je naša spletna aplikacija odzivna in dinamična, ji klasični model spletnih aplikacij ne ustreza.

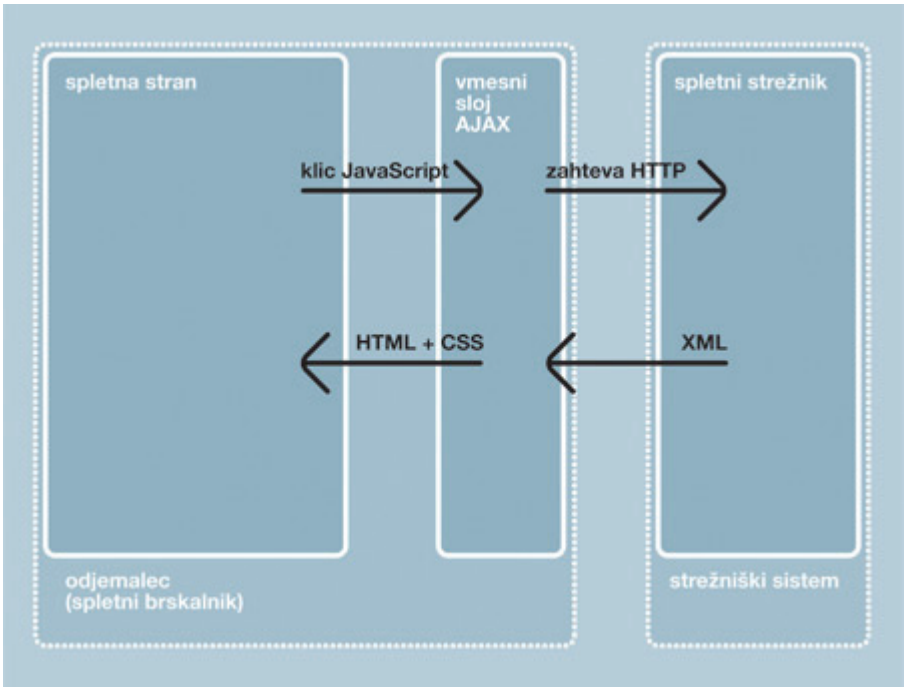
Tega so se začeli zavedati tudi spletni razvijalci. Zato so leta 2005, kot izboljšavo klasičnega modela spletnih aplikacij, predstavili spletno tehniko ali model, imenovan AJAX.

AJAX je kratica za "Asinhroni JavaScript in XML" in predstavlja tehniko ustvarjanja spletnih aplikacij. AJAX sam po sebi ni tehnologija, temveč le način združevanja večih neodvisnih tehnologij.

Tehnologije, ki jih AJAX združuje, so največkrat:

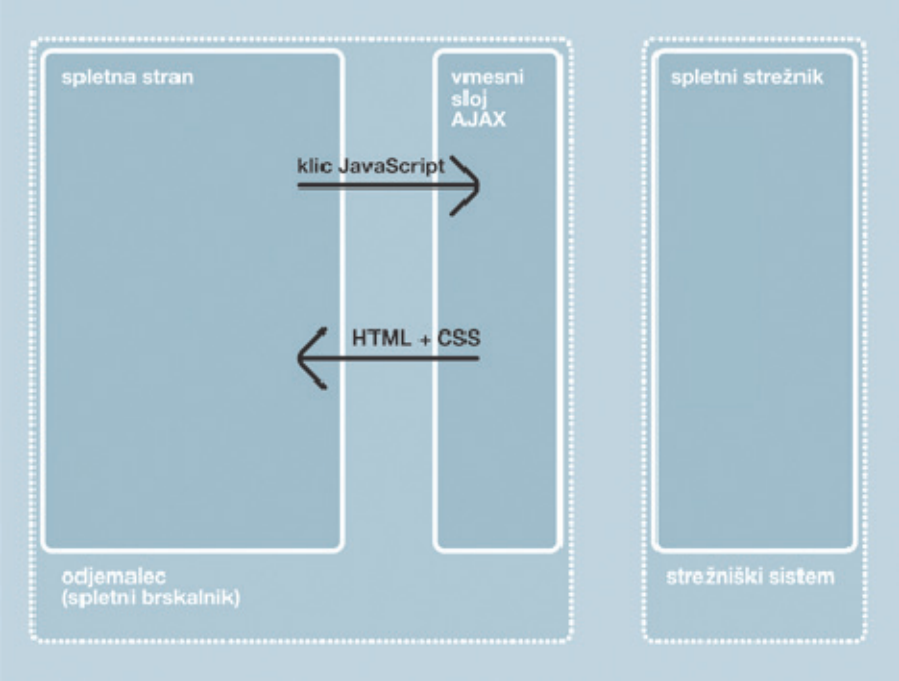
- (X)HTML[5] in CSS za obliko in prikaz vsebin.
- DOM v povezavi z jezikom JavaScript za dinamičen prikaz informacij in interakcijo z njimi.
- Objekt XMLHttpRequest za asinhrono izmenjavo podatkov med spletnim brskalnikom in strežnikom.

Grafični prikaz na sliki Slika 4 prikazuje proces komunikacije in prenos podatkov med spletnim brskalnikom in strežnikom po modelu AJAX.



Slika 4: Model AJAX, komunikacija s strežnikom je potrebna

[5] *XHTML* ali razširjeni HTML (Extensible Hyper Markup Language) je označevalni jezik za objavljanje vsebine na svetovnem spletu. Kot pove že samo ime, je to razširitev jezika HTML.



Slika 5: Model AJAX, komunikacija s strežnikom ni potrebna

Iz grafičnega prikaza na sliki Slika 4 in sliki Slika 5 je razvidno, da je pri modelu AJAX med spletnim brskalnikom in strežnikom vmesni sloj, ki ga bomo imenovali vmesni sloj AJAX. Vsaka zahteva, ki bi jo pri klasičnem modelu spletnih aplikacij spletna aplikacija poslala direktno strežniku, gre pri modelu AJAX preko vmesnega sloja AJAX.

Uporabnik z akcijami znotraj spletne aplikacije sproži klic funkcije v jeziku JavaScript. Primer akcije bi bil na primer klik na gumb. Funkcija, ki se kliče, se nahaja v vmesnem sloju AJAX. Funkcija JavaScript se nato izvrši na strani odjemalca, t.j. spletnega brskalnika, s katerim uporabnik dostopa do spletne aplikacije. Vmesni sloj analizira funkcijo. Če je v funkciji zapisano, da za odgovor na zahtevo ni potrebna interakcija s strežnikom, se podatki zahteve ne posredujejo strežniku (Slika 5). V tem primeru vmesni sloj AJAX s pomočjo funkcije JavaScript sam dinamično osveži vsebino znotraj spletne aplikacije. Primer osvežitve vsebine je sprememba barve ozadja ali pa položaj grafičnih elementov. V tem primeru se vse akcije opravijo na strani odjemalca, saj ni nikakršne komunikacije s strežnikom.

Kadar pa za odgovor na zahtevo potrebujemo podatke s strežnika, vmesni sloj AJAX, prav tako s pomočjo funkcij JavaScript, pošlje zahtevo strežniku (Slika 4). Strežnik zahtevo obdela. Če med obdelavo odgovor na spletni strani ni potreben takoj (npr. uporabnik izpolnjuje drugo polje vnosnega obrazca), uporabnik lahko nadaljuje z delom in ne čaka na odgovor strežnika. Kot odgovor na zahtevo strežnik lahko pošlje le podatke, ki so dejanski odgovor na zahtevo in ne več celotne spletne strani v obliki HTML+CSS. Spomnimo se, da je pri klasičnem modelu spletnih aplikacij strežnik kot odgovor vedno poslal celotno novo spletno stran v obliki HTML+CSS. Pri modelu AJAX je lahko odgovor na zahtevo v obliki navadnega besedila, (X)HTML-ja, XML-ja[6], JSON-a[7]. Ko vmesni sloj AJAX prejme odgovor na zahtevo, s pomočjo jezika JavaScript in metod DOM-a dinamično osveži vsebino znotraj spletne aplikacije.

Na ta način model AJAX omogoča, da komunikacijo med spletnim brskalnikom in strežnikom postavimo v ozadje. Z uporabo modela AJAX omogočimo asinhrono (torej ne nujno zaporedno) izvajanje obdelav na strežniku in brskalniku. Za uporabnika to pomeni, da lahko izvaja akcije znotraj spletne aplikacije, istočasno pa v ozadju poteka komunikacija med spletnim brskalnikom in strežnikom. S tem postane spletna aplikacija bolj odzivna, saj uporabniku med izvajanjem posameznih akcij ni potrebno več čakati.

V poglavju, ki sledi, si bomo komunikacijo med odjemalcem (spletnim brskalnikom) in strežnikom pogledali

---

[6] *XML* (Extensible Markup Language) je označevalni jezik, podoben jeziku HTML.

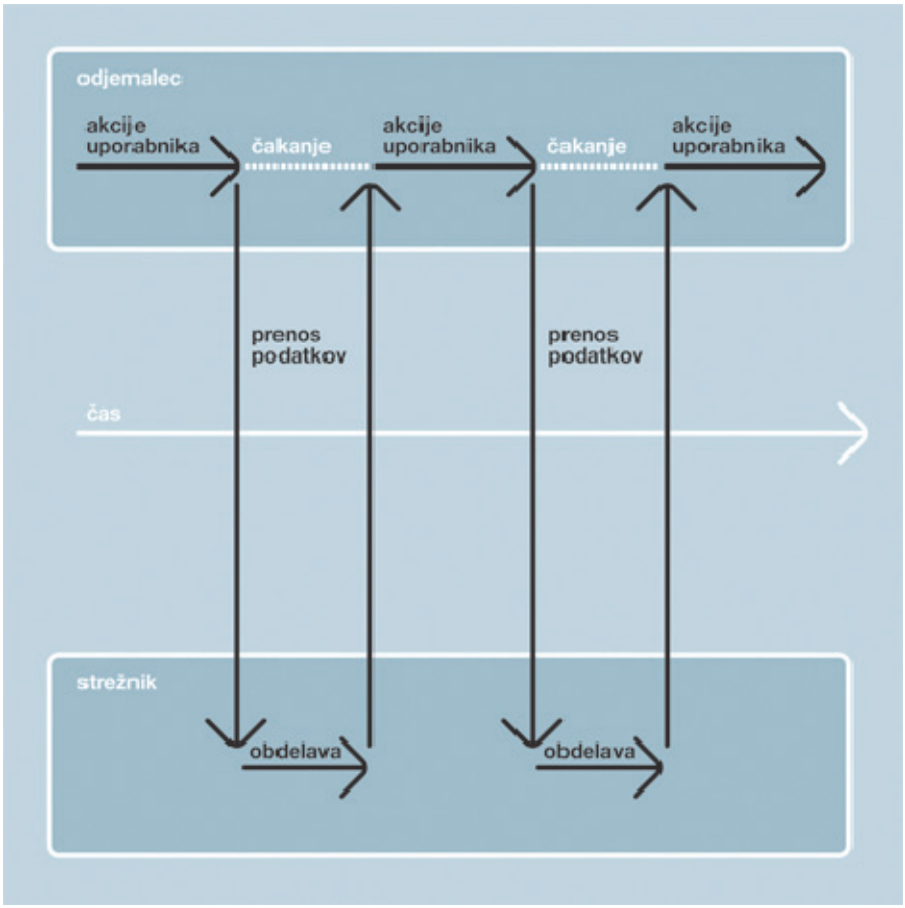
[7] *JSON* (JavaScript Object Notation) je preprost format za izmenjavo podatkov.

na konkretnem primeru

2.3. Sinhrona in asinhrona komunikacija

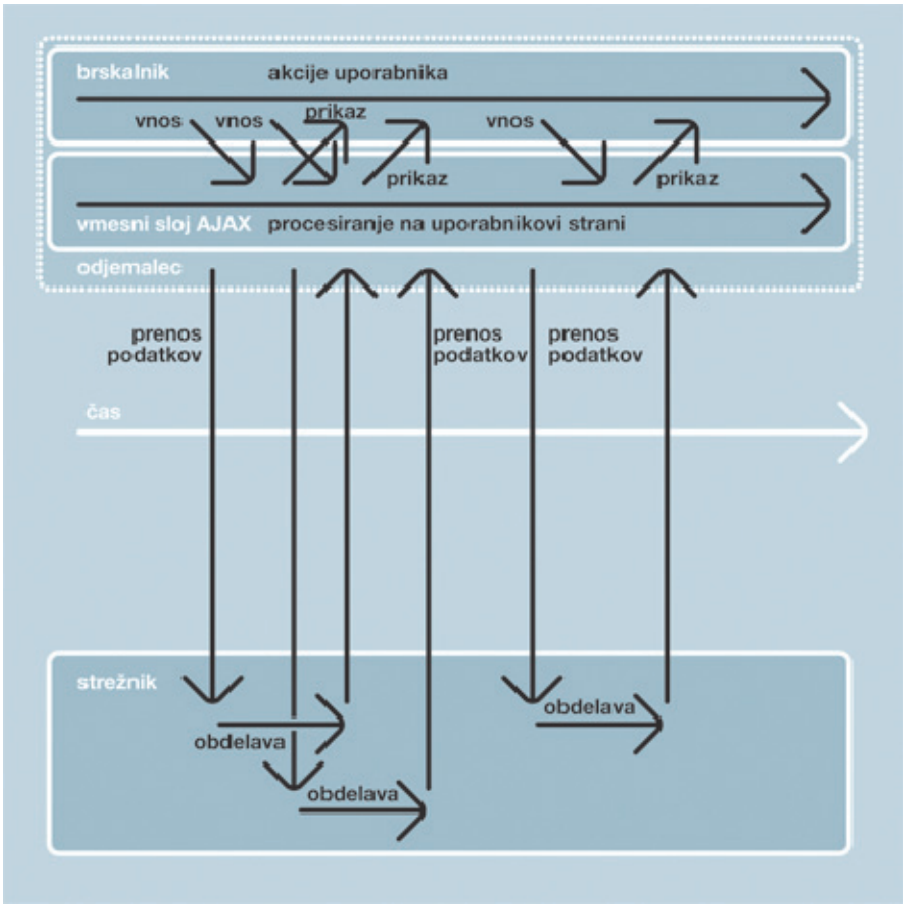
Da bi bolje razumeli razliko med sinhrono in asinhrono komunikacijo, si poglejmo grafični prikaz obeh.

Najprej si poglejmo grafični prikaz sinhrone komunikacije. Iz grafičnega prikaza na sliki Slika 6 je razvidno, da mora odjemalec (spletni brskalnik) preden pošlje strežniku novo zahtevo HTTP, vedno počakati, da dobi odgovor na predhodno poslano zahtevo HTTP. Za uporabnika to pomeni, da mora po vsaki akciji na spletni strani, ki sproži pošiljanje zahteve HTTP strežniku, čakati na odgovor strežnika. Šele, ko spletni brskalnik prejme odgovor strežnika, lahko uporabnik nadaljuje z izvajanjem naslednje akcije.



Slika 6: Sinhrona komunikacija

Asinhrona komunikacija, grafično predstavljena na sliki Slika 7, uporabniku omogoča izvajanje akcij brez vmesnega čakanja na odgovor strežnika. Denimo, da uporabnik izvede akcijo 1. Akcija 1 povzroči, da spletni brskalnik pošlje strežniku zahtevo HTTP 1. Še preden spletni brskalnik prejme odgovor akcije 1, uporabnik izvede akcijo 2. Akcija 2 sproži, da spletni brskalnik pošlje strežniku zahtevo HTTP 2. Pošiljanje zahtev HTTP prevzame vmesni sloj AJAX, ki je na strani odjemalca (spletnega brskalnika). Na ta način je uporabniku omogočeno nemoteno delo, saj mu med posameznimi akcijami ni potrebno čakati.



Slika 7: Asinhrona komunikacija

2.4.Nekaj zgledov

Oglejmo si nekaj zgledov spletnih strani, kjer se uporablja model AJAX. Pri vsakem primeru bomo na kratko opisali, kaj smo kot uporabniki s tehnologijo AJAX pridobili.

1. Primer: Kokoškastore
URL spletne strani: https://www.kokoskastore.com/
Kratek opis spletne strani: Na navedenem naslovu se nahaja spletna trgovina izdelkov za dom blagovne znamke Kokoška. Spletna trgovina uporabniku nudi pregled izdelkov in njihov nakup. Na sliki Slika 8 je grafični prikaz spletne strani.

Slika 8: https://www.kokoskastore.com/

Zgoraj navedena spletna stran s pomočjo modela AJAX dinamično posodablja podatke na strani. Zaradi tega je stran hitro odzivna.
Kot primer si poglejmo akcijo, ko uporabnik doda izdelek iz nabora v košarico.

1.korak: Iz nabora izdelkov si izberemo izdelek, ki ga želimo dodati v košarico.

Slika 9: Izberemo izdelek, ki ga želimo kupiti

2.korak: Kliknemo na gumb z napisom "DODAJ V KOŠARICO". Pojavi se okno, ki nam sporoči, da je bil izdelek dodan v košarico.

Slika 10: Dodamo izdelek v košarico

Prvi in drugi korak sta potekala sinhrono. Ker je spletna aplikacija narejena po modelu AJAX, strežnik vrne le odgovor o tem, ali je bil izdelek uspešno dodan v košarico ali ne. Spletni brskalnik nato prikaže le odgovor strežnika, informacijo o tem ali je bil izdelek uspešno dodan v košarico, ostali podatki na strani pa ostanejo nespremenjeni. Če bi bila spletna aplikacija narejena po klasičnem modelu, bi strežnik kot odgovor vrnil novo spletno stran. Čeprav bi se nova spletna stran od prvotne razlikovala le za malenkost, bi moral spletni brskalnik vseeno, za prikaz informacije o tem ali je bil izdelek uspešno dodan v košarico, ponovno naložiti celotno spletno stran, ki bi vsebovala odgovor.

3.korak: Ko uporabnik potrdi sporočilo pojavnega okna, se pojavno okno zapre. Nato se dinamično osvežita dve področji spletne strani: del, kjer je košarica uporabnika in del, ki prikazuje stanje zaloge za izdelek, ki smo ga dodali v košarico. 3. korak je grafično predstavljen na sliki Slika 11.

Slika 11: Osvežimo podatke na spletni strani

Ker je spletna stran bogata z grafičnimi vsebinami, je dinamično osveževanje spletne strani dobrodošlo. V primeru, da osveževanje spletne strani ne bi bilo dinamično, bi moral spletni brskalnik poleg nove vsebine, ponovno naložiti tudi veliko grafično bogate vsebine, ki pa se ni spremenila.

2. primer: Fitness
URL spletne strani: http://www.fitness.com/
Kratek opis spletne strani: Na navedenem naslovu URL se nahaja vse v zvezi s fitnes vadbo. Spletna stran uporabniku nudi informacije iz področja vaj, načrtovanja vadbe, prehrane in še mnogo drugega. Na sliki Slika 12 je grafični prikaz spletne strani.

Slika 12: http://www.fitness.com/

Na spletni strani so na voljo tudi pripomočki, npr. računalo za izračun indeksa BMI. (Op. Indeks BMI, angl. body mass index, je razmerje med telesno težo in višino.) Spletna stran in računalo sta grafično prikazana na sliki Slika 13.

Slika 13: Računalo za izračun indeksa BMI

V vnosna polja uporabnik vnese podatke o svoji teži v kilogramih in višini v centimetrih. Klik na povezavo "Calculate" sproži obdelavo vnesenih podatkov in izračun indeksa BMI. Ker je formula za izračun indeksa BMI stalna, komunikacija s strežnikom ni potrebna in računanje opravi vmesni sloj AJAX. Vmesni sloj AJAX nam po izračunu dinamično osveži stran z rezultatom izračuna.
Prikaz rezultatov, ki jih vrne računalo, je grafično predstavljen na sliki Slika 14.

Slika 14: Prikaz rezultata izračuna indeksa BMI

Na ta način ni prišlo do nepotrebne komunikacije s strežnikom in do osveževanja celotne spletne strani. Zaradi tehnike AJAX ta spletna aplikacija pridobi predvsem na hitrosti.

3. primer: Gmail
URL spletne strani: http://mail.google.com/
Kratek opis spletne strani: Na navedenem naslovu URL se nahaja spletna pošta podjetja Google. Spletna stran uporabniku omogoča prejemanje in pošiljanje elektronske pošte in pogovore z osebami z našega imenika, ki so prav tako imetniki Gmail računa. Na sliki Slika 15 je grafični prikaz spletne strani Gmail, ko smo se že prijavili v sistem.

Slika 15: http://mail.google.com/

Kot smo omenili že v uvodu 2. poglavja, je prav pojav spletne pošte Gmail pripomogel k temu, da je prišlo do povečanega zanimanja za tehniko AJAX.

V nadaljevanju si bomo pogledali eno izmed funkcionalnosti spletne aplikacije Gmail, ki jo omogoča tehnika AJAX.

Večina spletnih aplikacij za upravljanje z elektronsko pošto pri ustvarjanju novega elektronskega sporočila nudi možnost izbire naslovnika iz uporabnikovega imenika elektronskih naslovov. Lista elektronskih naslovov imenika se največkrat prikaže v novem pojavnem oknu. Pojavno okno z imenikom se prikaže, ko uporabnik klikne na gumb za iskanje, ki je največkrat poleg polja za vnos elektronskega naslova. V imeniku imamo več kot le nekaj oseb. Določeno osebo lahko poiščemo preko iskalnika ali tako, da kliknemo na določeno črko in potem aplikacija prikaže le naslovnike, katerih ime se začne s to črko. V tem primeru se moramo potem prebiti še skozi okrnjeno listo naslovnikov in najti pravega, kateremu želimo poslati elektronsko sporočilo. Tako vsakič naredimo najmanj naslednje štiri korake:

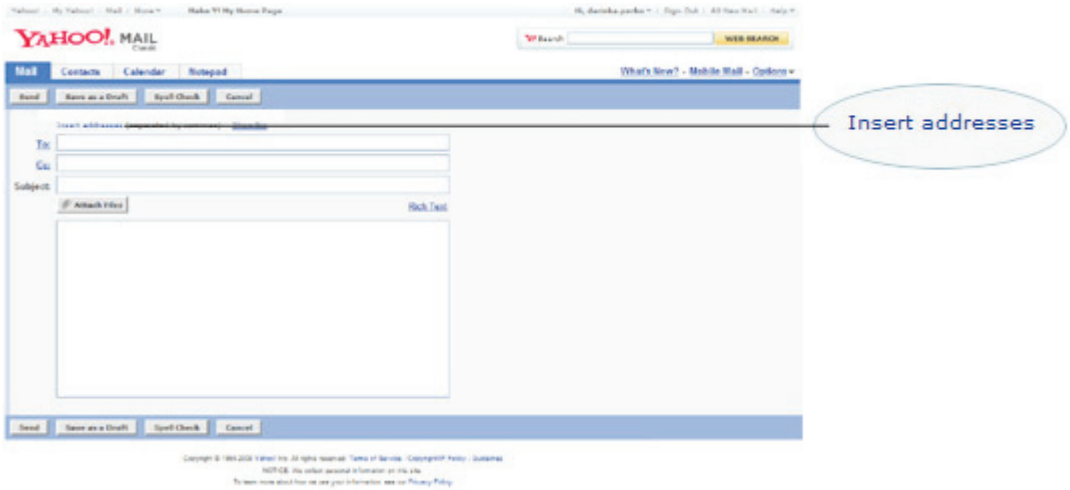
1.korak: Kliknemo na gumb za prikaz imenika.

2.korak: Vpišemo iskani niz ali kliknemo na določeno črko, da skrčimo nabor elektronskih naslovov.

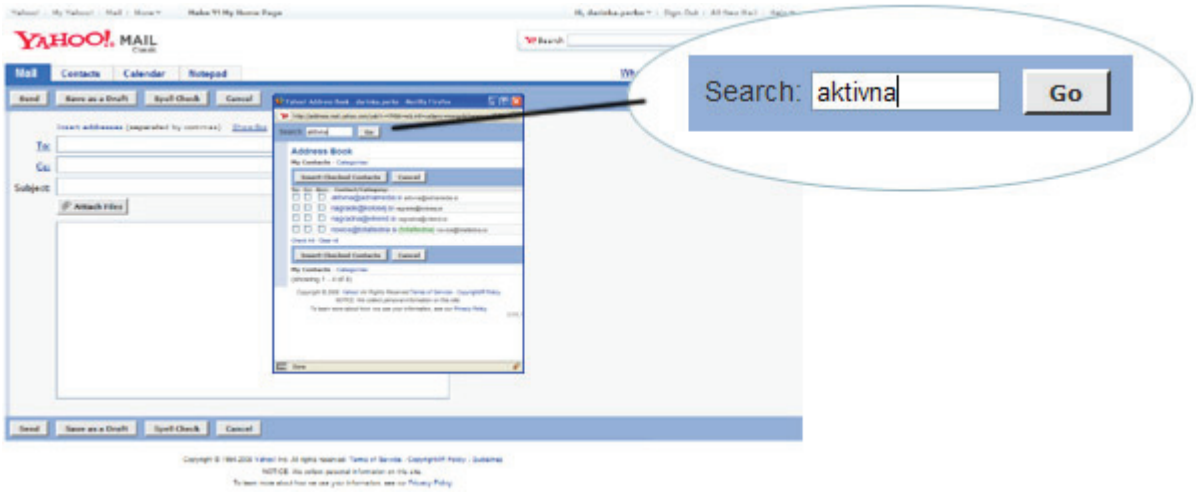
3.korak: Kliknemo na gumb za iskanje.

4.korak: Preletimo seznam elektronskih naslovov, ki jih spletna aplikacija vrne kot rezultat, glede na naše kriterije. Izberemo naslovnika.

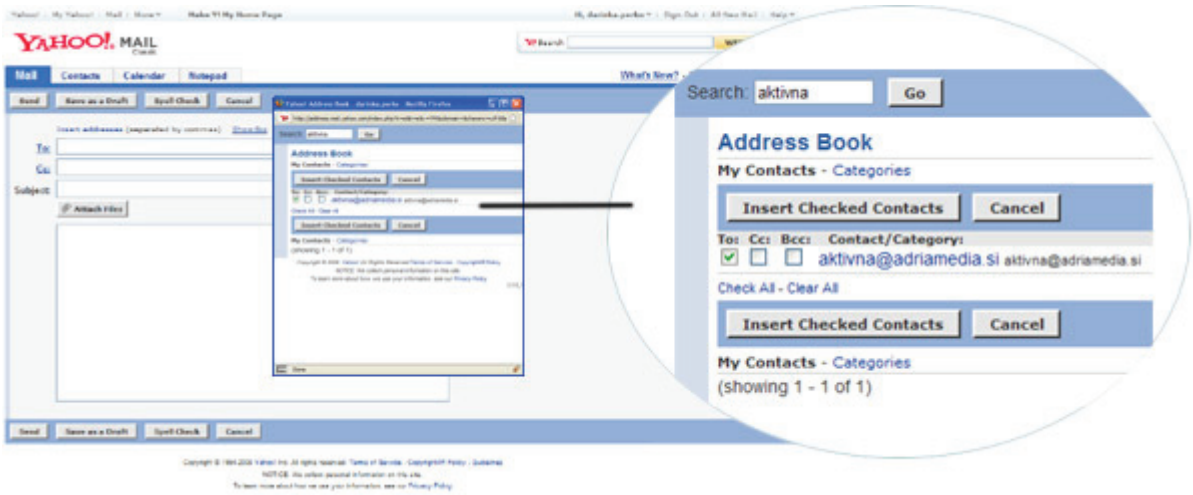
Sledi grafični prikaz pravkar opisanih štirih korakov dodajanja elektronskega naslova iz našega imenika med naslovnika pri kreiranju novega elektronskega sporočila. Aplikacija, ki jo bomo pri tem uporabili, se imenuje Yahoo! Mail, podjetja Yahoo. Ta uporablja klasičen model.



Slika 16: 1. korak, Yahoo! Mail



Slika 17: 2. in 3. korak, Yahoo! Mail



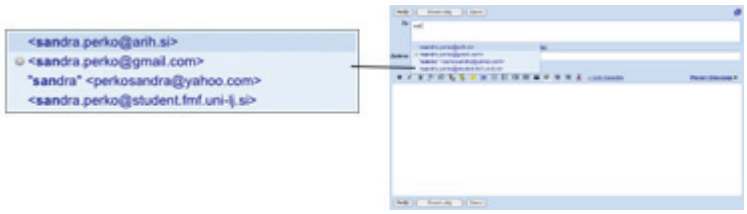
Slika 18: 4. korak, Yahoo! Mail

Spletni aplikaciji Gmail je te štiri korake uspelo strniti na dva koraka in sicer:

1.korak: V polje za vpis naslovnika začnemo vpisovati ime ali elektronski naslov (četudi le ugibamo).

2.korak: Spletna aplikacija ob vsaki spremembi v polju za vpis naslovnika, ki jo izvedemo, primerja vpisano s podatki v našem imeniku. Spletna aplikacija nato v obliki dinamičnega seznama ponudi naslovnike, ki se po imenu ali elektronskem naslovu ujemajo z vpisanim nizom. Če v tem seznamu najdemo željen elektronski naslov, ga lahko, tako da kliknemo nanj, dodamo med naslovnike. Če v seznamu ne najdemo želenega elektronskega naslova, nadaljujemo z vnosom.

Sledi grafični prikaz pravkar opisanega 1. in 2. koraka.



Slika 19: 1. korak, Gmail

Slika 20: 2. korak, Gmail

Spletni aplikaciji Gmail je uspelo zmanjšati število korakov s pomočjo tehnike AJAX. Ko uporabnik začne vpisovati niz znakov v vnosno polje, vmesni sloj AJAX pošlje zahtevo HTTP strežniku z nizom, ki ga je vnesel uporabnik. Strežnik primerja prejeti niz s podatki v imeniku uporabnika. Strežnik nato vrne naslovnike iz imenika uporabnika, ki temu nizu ustrezajo. Vmesni sloj AJAX nato poskrbi za dinamičen prikaz odgovora strežnika.

Te akcije se izvedejo vsakič, ko uporabnik vnese nov znak, oziroma kakorkoli spremeni podatke v vnosnem polju.

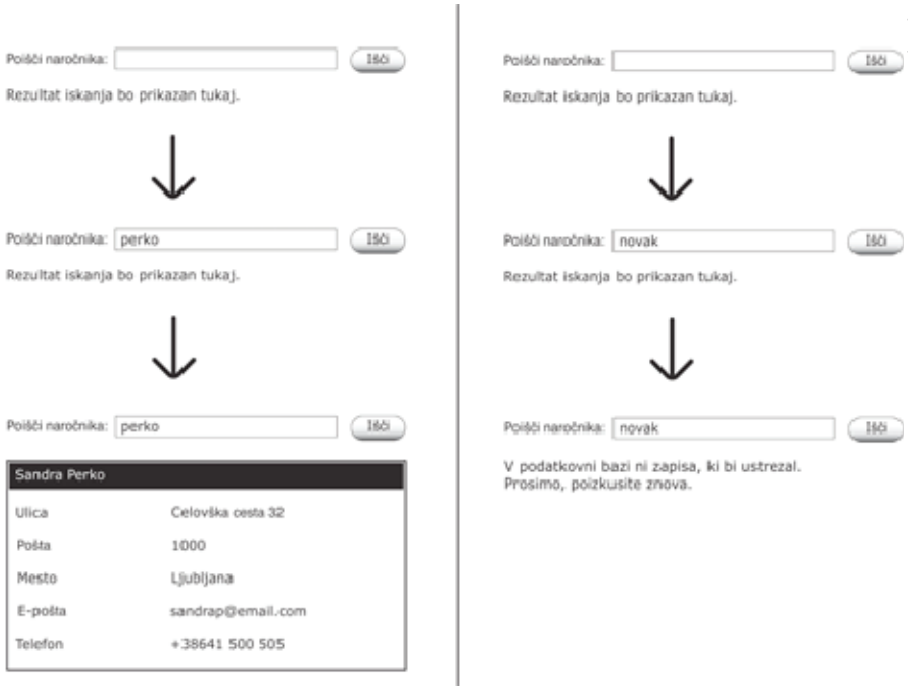
V našem primeru smo vnesli niz "sa" in spletna aplikacija nam vrne štiri vpise iz imenika, ki ustrezajo temu nizu znakov. Pri vsakem od teh vpisov je tudi poudarjen del (ime ali e-naslov), kjer se pojavi ujemanje z vnesenim nizom "sa" (glej grafični prikaz na sliki Slika 19).

3.Primerjava klasičnega modela in modela AJAX

Spletne aplikacije, ki uporabljajo tehniko AJAX, so navadno kompleksne in vsebujejo cel kup različnih elementov in prijemov. Pri tem se prikaže vsa prednost asinhrone komunikacije. Vendar zaradi kompleksnosti za začetno spoznavanje z osnovami tehnike AJAX niso najprimernejše. Kot primer bomo zato predstavili enostavno spletno aplikacijo, enostaven iskalnik, ki omogoča iskanje po bazi podatkov o naročnikih. Ta enostavna spletna aplikacija bi bila lahko del bolj kompleksne, grafično bogate spletne aplikacije. Zaradi enostavnosti iskalnika, ki ga bomo vzeli kot primer, spletna aplikacija s tehniko AJAX bistveno ne pridobi. Tehniko AJAX bomo v tem primeru uporabili le zato, da model AJAX bolj nazorno predstavimo.
Kot primer si torej poglejmo enostaven iskalnik, ki omogoča iskanje po bazi podatkov o naročnikih. Baza podatkov se imenuje primeri in ima le eno tabelo, imenovano narocniki. Polja tabele narocniki so predstavljena na sliki Slika 21.

Slika 21: Podatkovna tabela "narocniki"

Uporabnik v vnosno polje iskalnika vnese niz znakov, t.j. iskani niz, in klikne na gumb "Išči". Iskalnik iskani niz primerja s podatki v tabeli naročnikov narocniki. Če najde vpis, ki bi ustrezal iskanemu nizu, vrne podatke naročnika, ki ustreza iskanemu nizu. V nasprotnem primeru iskalnik uporabnika obvesti, da iskanega niza ni bilo moč najti in ga povabi k ponovnemu iskanju. Potek pravkar opisanih akcij je grafično prikazan na sliki Slika 22.



Slika 22: Potek iskanja

Najprej si poglejmo, kako bi takšno aplikacijo napisali po klasičnem modelu.

3.1. Klasični model

Vsebino in strukturo iskalnika bomo podali v obliki datoteke HTML, imenovane test.html. Vizualno podobo iskalnika bomo podali v obliki CSS, ki bo vključen neposredno v dokument HTML, t.j. datoteko test.html. Dokument test.html spletni brskalnik uporabniku prikaže kot spletno stran.

Sledi koda datoteke test.html.

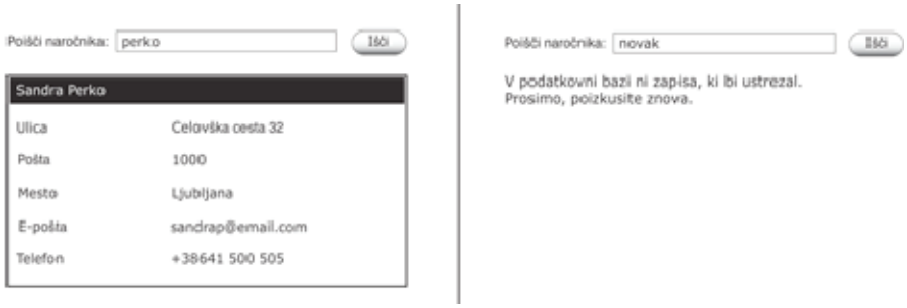
```
1.<html>
2.<head>
3.<title>Iskalnik po podatkovni bazi naročnikov</title>
4.</head>
5.<body style="margin: 100px auto; font-family: Verdana; font-size: 12px; color: #00000;">
6.<!-- začetek obrazca -->
7.<!-- ob potrditvi obrazca se kliče datoteka poisciNarocnika.php -->
8.<!-- poisciNarocnika.php se nahaja in izvrši na strežniku -->
9.    <form name="obrazec" method="GET" action="poisciNarocnika.php">
10.        Poišči naročnika:
11.        <!-- polje za vnos iskanega niza -->
12.        <input id="poisciNarocnika" name="poisciNarocnika" value="" type="text" size="30" />
13.        <!-- gumb za potrditev obrazca -->
14.        <input value="Išči" type="submit" />
15.    </form>
16.    <!-- konec obrazca -->
17.    <p>
18.    <!-- tu bo prikazan rezultat iskanja -->
19.        <div id="rezultatIskanja">
20.            Rezultat iskanja bo prikazan tukaj.
21.        </div>
22.    </p>
23.</body>
24.</html>
```

Datoteka 2: test.html

Dokument test.html spletni brskalnik uporabniku prikaže kot preprost obrazec z enim vnosnim poljem (Slika 23).

Slika 23: Enostaven iskalnik

Ko uporabnik v vnosno polje vpiše iskani niz in klikne na gumb "Išči", ta akcija sproži pošiljanje zahteve HTTP na strežnik. S tem se pove, da naj se izvede koda datoteke poisciNarocnika.php, ki se nahaja na strežniku. Dokumentu poisciNarocnika.php se posredujejo tudi podatki, ki jih je uporabnik vpisal v vnosno polje obrazca. Način, kako so ti podatki posredovani, v tem trenutku ni pomemben. Koda na datoteki poisciNarocnika.php se izvrši na strežniku. Strežnik spletnemu brskalniku kot rezultat vrne novo spletno stran v obliki HTML+CSS. Brskalnik jo prikaže, kot je vidno na grafičnem prikazu na sliki Slika 24.



Slika 24: Enostaven iskalnik, rezultat iskanja

Čeprav za našo zgodbo koda datoteke poisciNarocnika.php pravzaprav ni pomembna, jo vseeno navedimo. Na ta način bomo lažje tudi razumeli spremembe, ki jih bomo naredili v nadaljevanju.

```
1.<?php
2.// povezava s podatkovno bazo
3.mysql_connect("localhost", "root", "");
4.mysql_select_db("primeri");
5.// na ta način povemo MySQL-lu kakšne podatke naj pričakuje
6.mysql_query("SET NAMES 'utf8'");
7.mysql_query("SET CHARACTER SET utf8");
8.// dobimo podatke, ki so bili vneseni v vnosno polje obrazca
9.// vrednost, ki jo dobimo, spravimo v spremenljivko $poisciNarocnika
10.if (isset($_REQUEST["poisciNarocnika"])) {
11.    $poisciNarocnika = $_REQUEST["poisciNarocnika"];
12.} else {
13.    $poisciNarocnika = '';
14.}
15.// spremenljivka $rezultatIskanja bo vsebovala rezultat iskanja
16.$rezultatIskanja = '';
17.if ($poisciNarocnika == '') {
18.    // če je iskani niz prazen, ne naredimo nič
19.} else {
20.    // v nasprotem primeru, ga primerjamo s podatki v podatkovni bazi
21.    $sql = mysql_query("SELECT * FROM narocniki WHERE ime LIKE '%".$poisciNarocnika."%' OR priimek LIKE
'%".$poisciNarocnika."%' OR eposta LIKE '%".$poisciNarocnika."%' OR telefon LIKE '%".$poisciNarocnika."%' OR ulica LIKE
'%".$poisciNarocnika."%' OR posta LIKE '%".$poisciNarocnika."%' OR mesto LIKE '%".$poisciNarocnika."%' ORDER BY priimek ASC");
22.    // spremenljivka $sqlRezultat vsebuje rezultat iskanja
23.    // $sqlRezultat bo vseboval nekaj 'lepotnih' popravkov pri izpisu
24.    while ($narocnik = mysql_fetch_object($sql)) {
25.        // za večjo preglednost, bomo rezultat predstavili kot tabelo
26.        $sqlRezultat .= '<tr>';
27.        $sqlRezultat .= '<td colspan="2" style="background-color: #000000; color: #FFFFFF; font-weight: bold;">';
28.        $sqlRezultat .= $narocnik->ime.' '.$narocnik->priimek;
29.        $sqlRezultat .= '</td>';
30.        $sqlRezultat .= '</tr>';
31.        $sqlRezultat .= '<tr><td>Ulica</td>';
32.        $sqlRezultat .= '<td>'.$narocnik->ulica.'</td></tr>';
33.        $sqlRezultat .= '<tr><td>Pošta</td>';
34.        $sqlRezultat .= '<td>'.$narocnik->posta.'</td></tr>';
35.        $sqlRezultat .= '<tr><td>Mesto</td>';
36.        $sqlRezultat .= '<td>'.$narocnik->mesto.'</td></tr>';
37.        $sqlRezultat .= '<tr><td>E-pošta</td>';
38.        $sqlRezultat .= '<td>';
```

```
39.       $sqlRezultat .= '<a href="mailto:'.$narocnik->eposta.'">';
40.       $sqlRezultat .= $narocnik->eposta;
41.       $sqlRezultat .= '</a>';
42.       $sqlRezultat .= '</td></tr>';
43.       $sqlRezultat .= '<tr><td>Telefon</td>';
44.       $sqlRezultat .= '<td>'.$narocnik->telefon.'</td></tr>';
45.    }
46.    if ($sqlRezultat == '') {
47.       // če v podatkovni bazi nismo našli nobenega ustreznega zapisa,
48.       // moramo to sporočiti uporabniku
49.       // sporočilo uporabniku vsebuje spremenljivka $rezultatIskanja
50.       $rezultatIskanja = 'V podatkovni bazi ni zapisa, ki bi ustrezal.';
51.       $rezultatIskanja .= '<br />';
52.       $rezultatIskanja .= 'Prosimo, poizkusite znova.';
53.    } else {
54.       // v nasprotnem primeru še malo popravimo format izpisa rezultata
55.       // spremenljivka $sqlRezultat nosi vrstice tabele
56.       // za pravilen izpis moramo dodati še začetek in konec tabele
57.       $rezultatIskanja = '<table style="border: 1px solid #000000; font-family: Arial, Verdana; font-size: 12px; width: 500px;"
cellpadding="5" cellspacing="5">';
58.       $rezultatIskanja .= $sqlRezultat;
59.       $rezultatIskanja .= '</table>';
60.    }
61.}
62.?>
63.<html>
64.<head>
65.<title>Iskalnik po podatkovni bazi naročnikov</title>
66.</head>
67.<body style="margin: 100px auto; font-family: Verdana; font-size: 12px; color: #00000;">
68.<!-- zacetek obrazca -->
69.<!-- ob potrditvi obrazca se kliče datoteka poisciNarocnika.php -->
70.<!-- poisciNarocnika.php se nahaja in izvrši na strežniku -->
71.    <form name="obrazec" method="GET" action="poisciNarocnika.php">
72.       Poišči naročnika:
73.       <!-- polje za vnos iskanega niza -->
74.       <!-- polje za vnos iskanega niza ima sedaj vrednost zadnjega iskanega niza -->
75.       <input id="poisciNarocnika" name="poisciNarocnika" value="<?php echo $poisciNarocnika; ?>" type="text" size="30" />
76.       <!-- gumb za potrditev obrazca -->
77.       <input value="Išči" type="submit" />
78.    </form>
79.    <!-- konec obrazca -->
80.    <p>
81.    <!-- tu bo prikazan rezultat iskanja -->
82.       <div id="rezultatIskanja">
83.<?php
84.          // izpišemo rezultat iskanja
85.          echo $rezultatIskanja;
86.?>
87.       </div>
88.    </p>
89.</body>
90.</html>
```

Datoteka 3: poisciNarocnika.php

Za boljše razumevanje zgoraj navedene kode navedimo nekaj komentarjev o sami vsebini te datoteke. Za boljšo preglednost in lažje sledenje bomo kodo komentirali kakor si sledijo vrstice, saj se v tem vrstnem redu koda dejansko tudi izvaja.

Vrstica 11-14:

Dokument poisciNarocnika.php dobi vrednost iskanega niza. Če ta vrednost dokumentu poisciNarocnika.php iz kakršnega koli razloga ni posredovana, poisciNarocnika.php privzame, da ima iskani niz vrednost praznega niza.

Vrstica 15-45:

Če iskani niz ni prazen, njegovo vsebino primerjamo s podatki, ki se nahajajo v podatkovni tabeli narocniki. Primerjavo izvedemo med vsebinami vseh polj podatkovne tabele narocniki. S "primerjavo" v danem primeru mislimo, da je iskani niz vsebovan v zapisu polj podatkovne tabele narocniki. Če najdemo zapis, ki ustreza iskanem nizu, s tem najdemo naročnika, ki ustreza. Ker je ustreznih naročnikov lahko več, bomo rezultate ob primerjanju s posameznimi zapisi sproti dodajali vrednosti spremenljivke sqlRezultat. Spremenljivka sqlRezultat predstavlja niz, ki je z vsakim naročnikom, ki ustreza iskanemu nizu, daljši.

Podatke o naročniku, ki ji bomo dodali nizu, ki ga predstavlja spremenljivka sqlRezultat, pred tem še formatiramo. S tem dosežemo bolj pregleden izpis rezultata.

Vrstica 46-59:

V tem delu kode oblikujemo končni rezultat iskanja, katerega vrednost nosi spremenljivka rezultatIskanja. Najprej preverimo, kako uspešno je bilo naše iskanje po bazi podatkov. Če med zapisi v bazi podatkov nismo našli naročnika, ki bi ustrezal našemu iskanemu nizu, oblikujemo odgovor, s katerim to uporabniku tudi sporočimo. V nasprotnem primeru našo tabelo naročnikov, ki ustrezajo, še malo popravimo in s tem zagotovimo pravilnost kode HTML.

Vrstica 62-89:

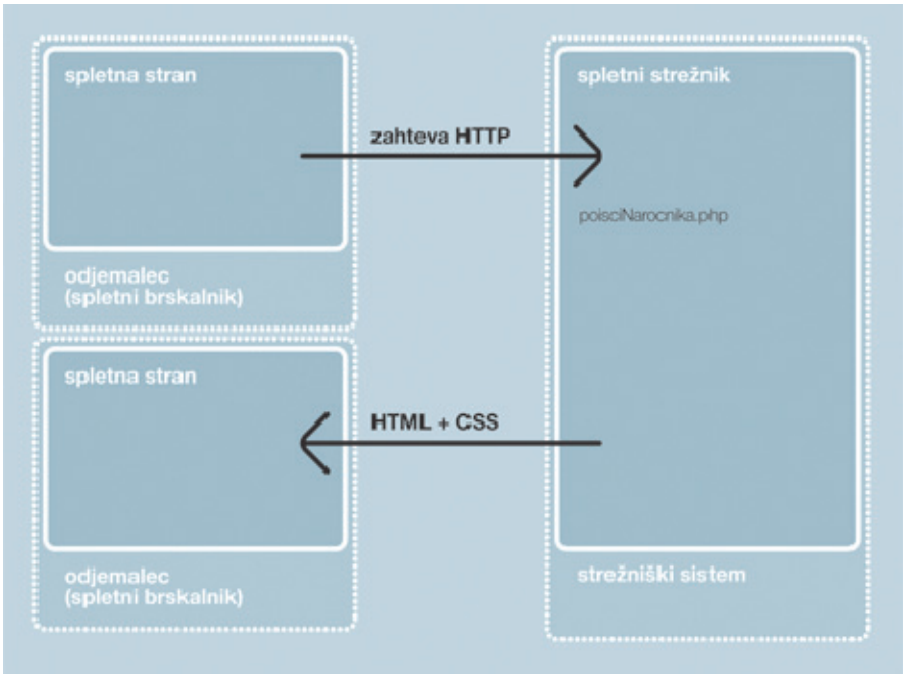
Odgovor (HTML+CSS) bo vseboval celotno stran z vnosnim obrazcem, kakršen je bil prej (za morebitno nadaljnje iskanje). Spremenili pa bomo del strani, kjer se prikažejo rezultati.

Vrstica 82-85:

Kot rezultat iskanja prikažemo vrednost spremenljivke rezultatIskanja.

V tem primeru koda v dokumentu poisciNarocnika.php opravi komunikacijo z bazo podatkov in posodobi vsebino spletne aplikacije z rezultatom iskanja. Rezultat iskanja nam strežnik vrne v obliki HTML+CSS, kar spletni brskalnik uporabniku prikaže kot novo spletno stran.

Grafični prikaz izvajanja aplikacije z uporabo klasičnega modela na našem primeru spletnega iskalnika je torej tak, kot prikazuje slika Slika 25.



Slika 25: Primer spletne aplikacije po klasičnem modelu

Iz grafičnega prikaza na sliki Slika 22 je razvidno, da se rezultat iskanja pojavi samo na določenem območju spletne strani, t.j. znotraj elementa div, ki nosi id z vrednostjo rezultatIskanja. Spletni brskalnik pa kot odgovor strežnika vedno dobi celotno stran v obliki HTML+CSS. Bolj smiselno bi bilo, da bi spletni brskalnik v odgovor dobil le rezultat iskanja, saj ostale vsebine ostajajo nespremenjene. Potem bi z dobljenim odgovorom strežnika posodobil tisti del spletne strani, kjer je predviden prikaz rezultata, t.j. znotraj elementa div z id vrednostjo rezultatIskanja. Na ta način bi spletna aplikacija postala bolj

dinamična.

Iz navedenega v razdelku o razlagi DOM-a lahko sklepamo, da je osveževanje delov spletne strani mogoče tudi pri klasičnem modelu s spreminjanjem drevesa DOM, ki pripada dokumentu HTML test.html. Vendar to drži le za vsebine, ki so statične narave in za kreiranje katere ne potrebujemo sodelovanje strežnika. Če za podatke, ki jih želimo prikazati, potrebujemo sodelovanje strežnika, pa klasični model ne podpira osveževanja le delov spletne strani. Razlog za to tiči v obliki odgovora, v kateri strežnik posreduje podatke spletnemu brskalniku pri klasičnem modelu spletnih aplikacij. Odgovor je v tem primeru vedno v obliki HTML+CSS, ki pa ne more biti prikazan drugače, kot popolnoma nova spletna stran.

Naša spletna aplikacija ima še eno pomanjkljivost. Ko se na strežniku izvaja koda poisciNarocnika.php, uporabnik ne more izvajati drugih akcij. Uporabnik mora počakati na konec izvajanja kode in s tem na odgovor strežnika na zahtevo. Vmesno čakanje uporabnika med posameznimi akcijami pa z drugimi besedami pomeni slabo odzivnost spletne aplikacije.

Za tako enostavno aplikacijo seveda to čakanje ni moteče, saj med tem uporabnik nima početi kaj drugega. Prav tako je prikaz strani enostaven in ga brskalnik opravi hitro. Zato se komentarja na temo dinamike in odzivnosti morda ne zdita smiselna. Če pa našo enostavno spletno aplikacijo umestimo v bolj kompleksno spletno aplikacijo, pa vprašanji dinamike in odzivnosti postaneta še kako pomembni.

V prejšnjem poglavju smo omenili, da lahko s spremembo modela omogočimo dinamično osveževanje spletne strani in boljšo odzivnost. Tak model smo imenovali model AJAX.

Kot smo že uvodoma navedli, je naš primer zelo enostaven in s prehodom s klasičnega na model AJAX ne bi bistveno pridobili. Enostaven primer smo uporabili le zato, da se lahko bolj osredotočimo na delovanje samega modela AJAX in ne toliko na delovanje spletne aplikacije.

V nadaljevanju si bomo pogledali, kako bi pravkar po klasičnem modelu narejeno spletno aplikacijo, spremenili v spletno aplikacijo po modelu AJAX.

3.2. Model AJAX

Problem klasičnega modela je, da se ob vsaki zahtevi (klik na gumb, vnos v vnosno polje …) vsakič izvede pošiljanja zahteve HTTP in s tem komunikacija med spletnim brskalnikom in strežnikom. Do pošiljanja zahteve HTTP pride tudi takrat, ko za to ni nobene potrebe. Pri klasičnem modelu je zahtevo HTTP pošiljal neposredno spletni brskalnik, pri modelu AJAX to nalogo prevzame JavaScript. Da pa bi poslali zahtevo strežniku z uporabo jezika JavaScript, moramo uporabiti objekt, ki nam omogoča to funkcionalnost.
Podjetje Microsoft je za potrebe svojega brskalnika Internet Explorer (v nadaljevanju IE) 5.0 razvilo razred XMLHttp s tako funkcionalnostjo. XMLHttp je podrazred razreda ActiveX, ki nam omogoča, da iz ene aplikacije kličemo objekte, definirane v drugi aplikaciji. Pri tem pa sta obe aplikaciji hkrati aktivni. Prva aplikacija se obnaša kot klient (spletni brskalnik), druga kot strežnik.

Poslovna politika Microsofta je bila, da je razvil razred XMLHttp le za potrebe svojega spletnega brskalnika IE. Če so ostala podjetja s svojimi spletnimi brskalniki želela konkurirati spletnemu brskalniku IE, so morala razviti svoj razred, ki bi omogočal podobne funkcionalnosti, kot jih omogoča razred XMLHttp. Tako se je kasneje pojavil razred XMLHttpRequest, ki ima enake funkcionalnosti kot razred XMLHttp.

Tako imamo sedaj dva razreda, razred XMLHttp in razred XMLHttpRequest, ki omogočata isto funkcionalnost, t.j. omogočata pošiljanje zahtev HTTP strežniku v jeziku JavaScript.

Da bi uporabo modela AJAX lažje razumeli, razdelimo dogajanje na tri korake. Posamezne korake bomo predstavili v podpoglavjih, ki sledijo.

3.2.1. *1. korak: Ustvarimo objekt za pošiljanje zahteve HTTP*

Spletne aplikacije običajno razvijamo tako, da so podprte na vseh spletnih brskalnikih. To povedano z drugimi besedami pomeni, da prikaz in delovanje spletne aplikacije ne sme biti odvisen od spletnega brskalnika, s katerim do spletne aplikacije dostopamo.

Posledica zgoraj opisanega razvoja je, da imamo dva razreda, ki v jeziku JavaScript omogočata pošiljanje zahtev HTTP strežniku. Če torej želimo zagotoviti delovanje spletne aplikacije, kjer v jeziku JavaScript pošiljamo zahteve HTTP, ne glede na s strani uporabnika uporabljeni spletni brskalnik, moramo omogočiti pošiljanje zahteve strežniku s pomočjo obeh razredov. Vsa koda JavaScript se namreč izvede na strani odjemalca, t.j. na strani spletnega brskalnika.
Zato je prvi korak ta, da uporabimo ustrezno implementacijo in ustvarimo nov objekt zahteva iz ustreznega razreda. Kateri razred bomo uporabili, se odločimo glede na spletni brskalnik, s katerim uporabnik dostopa do spletne aplikacije,

Vse akcije v zvezi z zahtevami HTTP bomo s pomočjo funkcij JavaScript nato izvajali preko tega objekta zahteva. S tem bomo omogočili, da bo koda od tu naprej ne glede na različne spletne brskalnike enaka.

Datoteka ustvariZahtevo.js vsebuje funkcijo ustvariZahtevo(), ki glede na uporabljeni spletni brskalnik ustvari ustrezen objekt za pošiljanje zahteve HTTP.

```
1.// spremenljivka zahteva predstavlja objekt za pošiljanje zahteve HTTP
2.var zahteva = null;
3.function ustvariZahtevo() {
4.    try {
5.        // poskušamo ustvariti nov objekt za pošiljanje zahteve HTTP
6.        zahteva = new XMLHttpRequest();// če ni Microsoftov brskalnik
7.    } catch (trymicrosoft) {
8.        try {
9.            // poskušamo ustvariti objekt na način, ki ustreza starejšim verzijam
10.           // brskalnika IE
11.           zahteva = new ActiveXObject("Msxm12.XMLHTTP");
12.       } catch (othermicrosoft) { //
13.           try {
14.               // poskušamo ustvariti objekt na način, ki ustreza novejšim verzijam
15.               // brskalnika IE
16.               zahteva = new ActiveXObject("Microsoft.XMLHTTP");
17.           } catch (failed) {
18.               // če gre kaj narobe, spremenljivko zahteva nastavimo na vrednost null
19.               zahteva = null;
20.           }
21.       }
22.   }
23.   // če ima spremenljivka zahteva vrednost null,
24.   // vemo, da je pri kreiranju objekta prišlo do napake
25.   if (zahteva == null) {
26.       // uporabnika s sporočilom v pojavnem oknu opozorimo, da je prišlo do napake
27.       alert ("Prišlo je do napake pri kreiranju objekta za pošiljanje zahteve HTTP!");
28.   }
29.}
```

Datoteka 4: ustvariZahtevo.js

Funkcija JavaScript ustvariZahtevo() poizkuša ustvariti objekt zahteva, ki ga uporabljamo za pošiljanje zahteve HTTP strežniku. Predstavnik katerega razreda bo objekt zahteva, je odvisno od spletnega brskalnika, s katerim dostopamo do spletne aplikacije. Če pri kreiranju objekta zahteva ni prišlo do napake, je funkcija ustvariZahtevo() ustvarila nov objekt zahteva, s katerim lahko pošiljamo zahteve HTTP. V nasprotnem primeru objekt zahteva ne obstaja in uporabniku javimo napako.

S prvim korakom smo s funkcijo ustvariZahtevo() v JavaScriptu ustvarili objekt, s katerim lahko s pomočjo jezika JavaScript strežniku pošljemo zahtevo HTTP.

Sedaj lahko nadaljujemo z naslednjim korakom. V njem bomo zahtevo HTTP dejansko poslali strežniku.

3.2.2. *2. korak: strežniku pošljemo zahtevo HTTP*

Ta korak bomo razdelili na več podkorakov:

2.1. Navedemo ime datoteke s kodo, ki jo bo spletni brskalnik poklical in se nahaja na strežniku. Koda klicane datoteke se bo izvršila na strežniku.

2.2. Navedemo funkcijo v JavaScriptu, ki jo bo spletni brskalnik poklical, ko bo na našo zahtevo prejel podatke oz. odgovor strežnika. Ta funkcija bo nato tudi osvežila podatke na spletni strani.
2.3. Objektu, ki smo ga kreirali v 1. koraku, določimo lastnosti, ki smo jih definirali v korakih 2.1 in 2.2.
2.4. Strežniku pošljemo zahtevo HTTP.

Vse naštete podkorake, pa tudi 1. korak, smo strnili v funkciji posljiZahtevo(), ki je del datoteke posljiZahtevo.js.

```
1.function posljiZahtevo() {
2.    // ustvarimo nov objekt za pošiljanje zahteve HTTP strežniku
3.    // objekt imenujeno zahteva (glej 1. korak)
4.    ustvariZahtevo();
5.
6.    // iz vnosnega polja z imenom posiciNarocnika poberemo vrednost iskanega niza
7.    var isci = document.getElementById('poisciNarocnika').value;
8.
9.    // na strežniku bomo izvedli program na datoteki poisciNarocnika-ajax.php
10.    // tej  podamo še vrednost parametra poisciNarocnika, ki je vsebina spremenljivke isci
11.    var url = "poisciNarocnika-ajax.php?poisciNarocnika="+isci;
12.
13.    // request.open izvede povezavo s strežnikom
14.    // "GET" definira, kako bomo poslali podatke strežniku
15.    // Vrednost spremenljivke url definira, kaj se bo na strežniku izvršilo.
16.    // Zadnji parameter definira vrsto komunikacije med spletnim brskalnikom in strežnikom.
17.    // V tem primeru je komunikacija asinhrona, kar opišemo z vrednostjo true
18.    zahteva.open("GET", url, true);
19.
20.    // Ko strežnik konča z obdelavo zahteve, naj se izvrši funkcija  posodobiStran().
21.    zahteva.onreadystatechange = posodobiStran;
22.
23.    // Zahtevo pošljemo strežniku brez dodatnih parametrov.
24.    zahteva.send(null);
25.}
```

Datoteka 5: posljiZahtevo.js

Za boljše razumevanje funkcije JavaScript posljiZahtevo() ji namenimo nekaj komentarjev. Komentarje bomo podali le za vrstice, za katere že v kodo umeščeni komentarji morda ne zadoščajo za razumevanje.

Vrstica 7:

Jezik JavaScript lahko preko funkcij, ki jih določa DOM dostopa do lastnosti in vrednosti elementov spletne strani. Tukaj smo za dostop uporabili kombinacijo dveh funkcij.

Klic funkcije document.getElementById('poisciNarocnika') nam vrne element, ki se nahaja znotraj celotne spletne aplikacije in ima atribut id enak poisciNarocnika. Po pravilih jezika HTML je lahko znotraj celotne spletne aplikacije le en tak element, ker mora biti atribut id enoličen.

Z `.value` dobimo vrednost tega elementa. V našem primeru element z atributom id poisciNarocnika predstavlja vnosno polje obrazca, v katerega uporabnik vnese iskani niz. Vrednost elementa HTML, ki predstavlja vnosno polje, je enaka kar vsebini vnosnega polja. V našem primeru je torej vsebina vnosnega polja iskani niz, ki ga uporabnik vpiše v naš enostaven iskalnik. Z izrazom document.getElementById('poisciNarocnika').value tako dobimo niz, ki ga je kot iskalni niz uporabnik vnesel v iskalnik.

Vrstica 11:

Spremenljivka url vsebuje URL(spletni naslov), kamor naj se pošlje naša zahteva HTTP.

Prvi del naslova je ime datoteke, ki se nahaja in izvrši na strežniku. Drugi del naslova so podatki, ki se posredujejo datoteki, navedeni v prvem delu. Navedeni so v obliki imeParametra=vrednostParametra in med seboj ločeni z znakom '&'. Prvi in drugi del URL pa sta med seboj ločena z znakom '?'.

Za naš primer je torej prvi del URL 'poisciNarocnika-ajax.php', drugi del pa 'poisciNarocnika='+isci. Spremenljivka isci vsebuje vrednost iskanega niza. Na ta način datoteki poisciNarocnika-ajax.php posredujemo vsebino iskanega niza, ki ga je uporabnik vnesel v iskalnik.

Vrstica 18:

Z ukazom zahteva.open pošljemo zahtevo strežniku. V komentarju v vrstici 14 smo navedli, da `"GET"` definira metodo, kako bomo poslali podatke strežniku. Kadar uporabimo metodo GET, bodo podatki, ki jih bomo posredovali, kar del naslova podanega s spremenljivko `url`. Kako podatke vključimo v URL, smo navedli pri predhodnem komentarju.

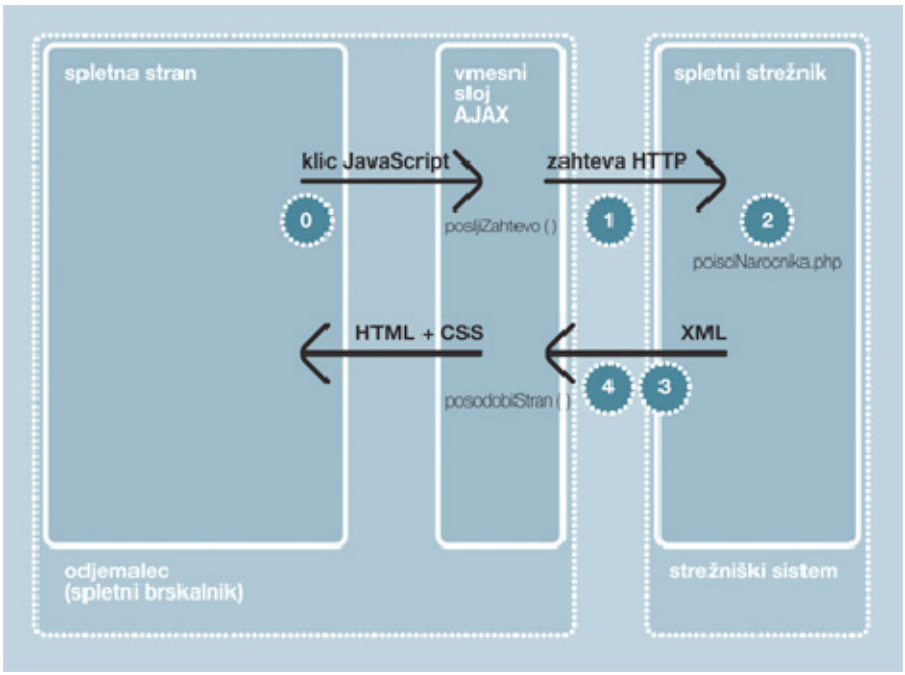
V delu komentarja k temu ukazu (komentar v vrstici 16) smo navedli, da zadnji parameter tipa boolean definira vrsto komunikacije med spletnim brskalnikom in strežnikom. Če ima zadnji parameter vrednost true, je komunikacija asinhrona. V nasprotnem primeru je komunikacija sinhrona (razlago sinhrone in asinhrone komunikacije smo navedli na strani 12). Iz tega sledi, da so spletne aplikacije, ki sledijo modelu AJAX, lahko tudi sinhrone. Sinhrono komunikacijo med spletnim brskalnikom in strežnikom uporabimo takrat, ko mora biti dogajanje nujno zaporedno, ali pa ko želimo imeti več kontrole nad akcijami uporabnika in se nam zdi smiselno omejiti dogajanje na strežniku na le eno zahtevo na enkrat.

Vrstica 21:

Objekt zahteva, ki predstavlja našo zahtevo HTTP, ima tako imenovano lastnost ready state. Spletni brskalnik ima dostop do te lastnosti, ki spletnemu brskalniku pove, v katerem koraku izvajanja je objekt zahteva. Lastnost ready state posameznega objekta lahko zavzame pet možnih vrednosti. Te so razložene v nadaljevanju.

Lastnost zahteve onreadystatechange spletnemu brskalniku pove, katero funkcijo v JavaScriptu onreadystatechange mora klicati vsakič, ko se vrednost lastnosti ready state objekta spremeni. Z zahteva.onreadystatechange=posodobiStran torej povemo, da se bo ob spremembi vrednost lastnosti ready state objekta zahteva tako klicala funkcija posodobiStran(). Sintaksa zapoveduje, da tukaj pri imenu funkcije JavaScript izpustimo oklepaje in pišemo le posodobiStran, namesto posodobiStran(), kar bi bilo morda bolj naravno.

Koraki spreminjanja vrednosti lastnosti ready state so grafično prikazani na sliki Slika 26.



Slika 26: Koraki spreminjanja stanja objekta zahteva

Posamezno stanje nam pove, v katerem stadiju izvajanja je zahteva HTTP. Vrednost stanja objekta, ki predstavlja zahtevo, lahko spreminja le spletni brskalnik. Spletni brskalnik vrednost stanja objekta zahteva nastavi glede na stanje komunikacije s strežnikom.

Korak 0:

Ustvari se objekt za pošiljanje zahteve HTTP. Objekt je v stanju 0.

Prehod iz koraka 0 v korak 1 se zgodi, ko se s strežnikom vzpostavi povezava HTTP. V našem zgledu se to zgodi takrat, ko funkcija posljiZahtevo() na objektu zahteva izvrši funkcijo open() (zahteva.open("GET", url, true)) in s tem vzpostavi povezavo HTTP s strežnikom.

Korak 1:

Ko je povezava HTTP s strežnikom vzpostavljena, se zahteva HTTP, ki jo predstavlja objekt zahteva, pošlje strežniku. Objekt zahteva je v koraku 1 v stanju 1.

Korak 2:

Strežnik prejme zahtevo HTTP in jo posreduje ustrezni datoteki, ki se nahaja in izvrši na strežniku. V našem primeru je to poisciNarocnika-ajax.php. Ko poisciNarocnika-ajax.php prejme zahtevo, začne z izvrševanjem kode. Objekt zahteva preide v stanje 2.

Korak 3:

Podatki za odgovor strežnika na zahtevo HTTP se shranjujejo v objektu zahteva. Ker obdelava še ni končana, v tem koraku podatki odgovora na zahtevo HTTP še niso dostopni za uporabo kodi JavaScript na odjemalčevi strani. Kakor hitro se začnejo podatki shranjevati v objekt zahteva, njegovo stanje postane enako 3.

Korak 4:

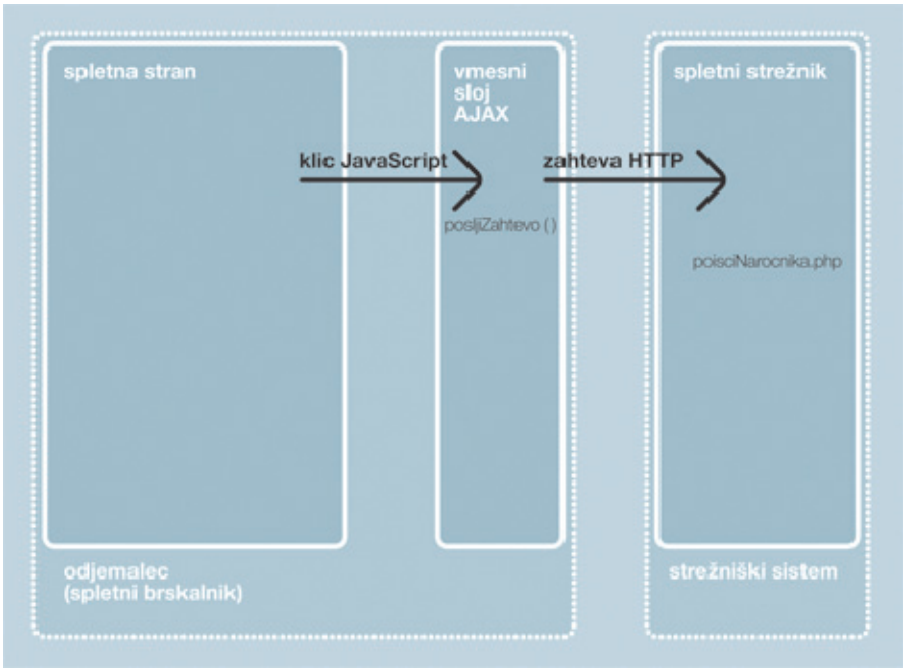
Strežnik je zaključil z obdelovanjem naše zahteve HTTP in njegov odgovor je shranjen v objektu zahteva. Objekt zahteva preide v stanje 4, oziroma ready state dobi vrednost 4. Koda JavaScript na odjemalčevi strani lahko uporabi podatke, ki jih je vrnil strežnik.

Stanja 0, 1, 2, 3 in 4 tvorijo "življenjsko" pot objekta zahteva, ki predstavlja zahtevo HTTP. Taka pot se izvrši vsakič, ko na spletni strani pride do akcije, ki sproži pošiljanje zahteve HTTP strežniku. Vsaka nova pot pomeni kreiranje novega objekta zahteva za pošiljanje zahtev HTTP strežniku.

Iz grafičnega prikaza na sliki Slika 26 in pripadajočih komentarjev sledi, da je podatke odgovora strežnika na zahtevo HTTP možno uporabiti le, ko je objekt zahteva v stanju 4. Takrat je strežnik zagotovo že končal z obdelovanjem zahteve in so podatki, ki jih je vrnil, končni.

Lastnost ready state objekta zahteva nam torej pove, v kateri fazi izvajanja je zahteva HTTP, ki jo predstavlja določen objekt zahteva. To bi pri posameznih akcijah utegnilo zanimati tudi uporabnika spletne aplikacije. Uporabniku lahko, še preden strežnik konča z obdelovanjem zahteve HTTP, ki jo predstavlja objekt zahteva, s pomočjo lastnosti ready state objekta zahteva sporočimo, v kateri fazi izvajanje je zahteva HTTP. Tako uporabnik dobi informacijo o tem, kaj se dogaja med tem, ko na spletni strani izvaja nove akcije in še vedno pričakuje rezultat akcije, ki je sprožila pošiljanje zahteve HTTP. Na ta način uporabniku sporočimo, kaj se dogaja v ozadju, ko poteka komunikacija med vmesnim slojem AJAX in strežnikom. Pri spletnih aplikacijah klasičnega modela je uporabniku jasno, da strežnik še ni zaključil z obdelovanjem zahteve, saj je prisiljen čakati in ne more izvajati novih akcij. Pri spletnih aplikacijah modela AJAX pa je komunikacija med odjemalcem in strežnikom premaknjena v ozadje in tako uporabniku nevidna. Tako uporabnik ne more vedeti, da je zahteva HTTP v obdelavi, razen, če mu tega ne sporočimo.

Celoten 2. korak, pošiljanje zahteve HTTP strežniku, je grafično prikazan na sliki Slika 27.



Slika 27: Model AJAX, pošiljanje zahteve HTTP

Povzemimo zgodbo do sedaj.

Ko uporabnik v vnosno polje obrazca vnese iskani niz in klikne na gumb za potrditev obrazca, spletni brskalnik pokliče funkcijo posljiZahtevo(). Funkcija posljiZahtevo() najprej s pomočjo funkcije ustvariZahtevo() kreira nov objekt, imenovan zahteva. Ta bo uporabljen za pošiljanje zahtev HTTP strežniku. Funkcija posljiZahtevo() nato z uporabo tega objekta strežniku pošlje zahtevo HTTP. Strežnik zahtevo HTTP posreduje datoteki poisciNarocnika-ajax.php, ki se nahaja in izvrši na strežniku.

Do tega trenutka smo se osredotočali le na dogajanje na odjemalčevi strani. Sedaj pa nas bo zanimalo, kaj se zgodi, ko zahteva HTTP enkrat pride do datoteke poisciNarocnika-ajax.php, ki se nahaja na strežniku. Da bi vedeli kaj se zgodi, si moramo najprej ogledati vsebino datoteke poisciNarocnika-ajax.php.

Še prej pa spomnimo na eno od prednosti modela AJAX. Ta je da strežniku pri odgovoru na zahtevo HTTP ni več potrebno pošiljati celotne nove spletne strani. Pri klasičnem modelu se je na strežniku izvršila koda datoteke poisciNarocnika.php, ki je vračala celotno spletno stran v obliki HTML+CSS. Kodo, ki je v datoteki poisciNarocnika.php smo preoblikovali tako, da sedaj vrača le rezultat iskanja in ne več celotne spletne strani. Preimenovali smo jo v poisciNarocnika-ajax.php. Kompletni začetek (prvih 61 vrstic) je koda identična prvotni. Zato si oglejmo le del tisti del kode poisciNarocnika-ajax.php, kjer smo kodo PHP spremenili.

```
62. // od 1. do vključno 61. vrstice je koda identična kodi prvotne datoteke,
63. // datoteki poisciNarocnika.php
64.
65. // izpišemo rezultat iskanja
66.    echo $rezultatIskanja;
67. // konec datoteke poisciNarocnika-ajax.php
68. ?>
```

Datoteka 6: poisciNarocnika-ajax.php

Kot je razvidno iz vsebine datoteka poisciNarocnika-ajax.php, nam koda te datoteke vrne le vrednost spremenljivke rezultatIskanja. Rezultat iskanja le bodisi tabela s podatki naročnika ali večih naročnikov, ki ustrezajo iskanemu nizu bodisi niz znakov s sporočilom o neuspešnem iskanju. Struktura in vsebina tabele je podana v jeziku HTML, izgled tabele pa z jezikom CSS, ki je umeščen direktno v kodo HTML. V odgovor na zahtevo HTTP bo tako strežnik vrnil le del strani v obliki HTML+CSS, ki predstavlja dejanski rezultat iskanja. S tem smo zmanjšali količino podatkov, ki se prenašajo s strani strežnika proti spletnemu brskalniku.

Do tega trenutka smo ustvarili objekt zahteva za pošiljanje zahtev HTTP strežniku, objektu zahteva

določili parametre zahteve HTTP in jo poslali strežniku. Strežnik je zahtevo posredoval datoteki poisciNarocnika-ajax.php. V tem trenutku se na strežniku izvaja koda, zapisana v datoteki poisciNarocnika-ajax.php, kar pomeni, da strežnik zahtevo obdeluje. Naslednji logični korak je, da definiramo akcije, ki se bodo izvršile, ko strežnik zaključi z obdelovanjem naše zahteve HTTP.

3.2.3. *3. korak: prejmemo odgovor strežnika in osvežimo podatke na spletni strani*

Ko strežnik konča z obdelovanjem naše zahteve HTTP, odgovor na zahtevo shrani v objekt zahteva. Takrat pri objektu zahteva lastnost ready state dobi vrednost 4. Sedaj funkcije JavaScript na odjemalčevi strani lahko varno uporabijo podatke, ki jih vrne strežnik kot odgovor na zahtevo HTTP.
Za osvežitev strani poskrbi funkcija JavaScript, ki smo jo navedli v funkciji posljiZahtevo(), t.j. funkcija posodobiStran() (glej korak 2.3). V funkciji posljiZahtevo() smo navedli, da se funkcija posodobiStran() kliče vsakič, ko se spremeni stanje objekta zahteva. Torej bomo preden bomo izvajali nadaljnje akcije, morali v funkciji posodobiStran() prej preveriti, kakšno je stanje objekta zahteva. V nadaljevanju sledi koda funkcije posodobiStran(), ki smo jo vključili v datoteko posodobiStran.js.

```
1.function posodobiStran() {
2.    if (zahteva.readyState == 4) { // če je strežnik že vrnil odgovor
3.        if (zahteva.status == 200) { // strežnik je sporočil, da je njegov odgovor v redu
4.            // dobimo odgovor strežnika
5.            var odgovorStreznika = zahteva.responseText;
6.
7.            // posodobimo stran z vsebino odgovora strežnika
8.            document.getElementById("rezultatIskanja").innerHTML = odgovorStreznika;
9.        } else {
10.            alert ('Napaka pri vračanju rezultatov! Strežnik je vrnil kodo' +
11.              zahteva.status + '.');
12.        }
13.    }
14.}
```

Datoteka 7: posodobiStran.js

Za boljše razumevanje funkcije JavaScript posodobiStran() ji poleg v komentarjev v kodi namenimo še nekaj komentarjev.

Vrstica 2:

Na ta način zagotovimo, da s podatki operiramo šele, ko je strežnik zaključil s svojim delom in je varno uporabiti podatke, ki jih vrne.

Vrstica 3:

Na tem mestu preverimo vrednost lastnosti status objekta zahteva. Lastnost status nam pove, kaj se je zgodilo z zahtevo HTTP med izvajanjem na strežniku. Na tem mestu že vemo, da je strežnik zaključil z obdelovanjem zahteve HTTP (glej komentar za vrstico 2). Vendar to še ne pomeni, da je bila obdelava zahteve uspešna. Strežnik v lastnost status objekta zahteva shrani kodo oz. številko, ki predstavlja končno stanje obdelave zahteve HTTP. Bolj pogoste kode stanj obdelave in njihovi pomeni so zbrani v naslednji tabeli. Znak x v kodah predstavlja poljubno števko.

Koda	Opis
1xx	Informativno sporočilo.
2xx	Uspeh.
200	Obdelava zahteve HTTP uspešno izvedena.
3xx	Preusmeritev.
301	Strežnik ni mogel posredovati zahteve HTTP navedeni datoteki, ker se datoteka ne nahaja več na danem URL naslovu.
4xx	Napaka odjemalca.
400	Nekaj je narobe s sintakso zahteve HTTP in je strežnik ni razumel.
401	Za izvršitev zahteve HTTP nimamo pravic.
403	Strežnik je zavrnil izvršitev zahteve.
404	Strežnik ni našel navedene datoteke, na katero je naslovljena zahteva HTTP.
5xx	Napaka strežnika.
500	Prišlo je do notranja napaka strežnika in zahteva ni bila uspešno obdelana.

Tabela 1: Bolj pogoste možne kode stanj objekta zahteva

Iz tabele Tabela 1 sledi, da je strežnik uspešno zaključil z obdelovanjem zahteve HTTP, ko ima objekt zahteva vrednost lastnosti status enako 200.

Preden uporabimo rezultate, ki jih je vrnil strežnik, moramo vedno preveriti dve stvari. Tako to, ali je strežnik zaključil z obdelovanjem zahteve HTTP, kot tudi ali je bila obdelava zahteve na strežniku uspešno izvedena.

Vrstica 5:

Na ta način odgovor strežnika shranimo v spremenljivko. Kot smo že omenili strežnik odgovor shrani v sam objekt. To zapiše v lastnost responseText. V našem primeru bo spremenljivka odgovorStreznika kot vrednost hranila rezultat iskanja iskanega niza po tabeli naročnikov.

Vrstica 8:

Vsak element HTML ima lastnost imenovano innerHTML, ki predstavlja vsebino med začetno in končno značko[8].

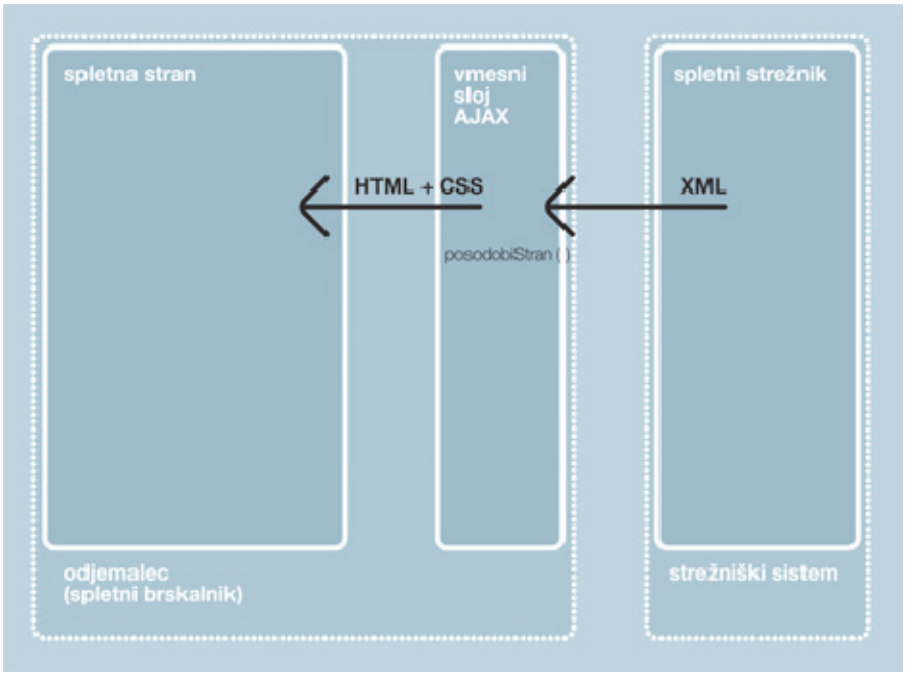
Pomen izraza document.getElementById("rezultatIskanja") smo že spoznali. Izraz document.getElementById("rezultatIskanja").innerHTML torej predstavlja vsebino med začetno in končno značko tistega elementa HTML, ki ima id enak rezultatIskanja. Kot vsebino tega elementa, dodelimo vrednost spremenljivke odgovorStreznika. Tako smo z odgovorom strežnika osvežili le del spletne strani. Tako osveževanje smo imenovali dinamično.

Vrstice od 9 do 11:

Če strežnik ni uspešno zaključil z obdelovanjem zahteve, uporabniku to sporočimo s sporočilom v pojavnem oknu.

Za boljše razumevanje poteka koraka 3 si na oglejmo še njegov grafičen prikaz.

[8] Izjema so elementi HTML, ki so definirani z enojno značko (npr. <img>).



Slika 28: Pošiljanje odgovora strežnika in osveževanje spletne strani

Na sliki Slika 28 je grafično prikazano, da strežnik odgovor pošlje v obliki XML. V 1. poglavju smo navedli, da pri modelu AJAX lahko strežnik odgovor pošlje v različnih formatih. Na sliki Slika 28 je format XML naveden zgolj zato, da poudarimo, da pri modelu AJAX strežniku ni potrebno več pošiljati odgovora le v obliki HTML+CSS.

Povzemimo ponovno celotno dogajanje do tega trenutka. Ustvarili smo objekt zahteva za pošiljanje zahtev HTTP strežniku. Objektu zahteva smo določili parametre zahteve HTTP in jo poslali strežniku. Za vse navedeno poskrbi funkcija JavaScript posljiZahtevo(). Strežnik je zahtevo posredoval datoteki poisciNarocnika-ajax.php. Skripta na datoteki poisciNarocnika-ajax.php se je izvedla. Strežnik je vrnil odgovor na zahtevo HTTP. Funkcija JavaScript posodobiStran() je s podatki, dobljenimi kot odgovor strežnika, osvežila podatke na spletni strani.

3.2.4. *Celotna spletna aplikacija*

Zdi se, da je s tem naše delo končano. Vendar je potrebno narediti še zelo pomemben del, ki poveže vse skupaj. To je sama spletna aplikacija, ki bo vključevala vse zgoraj naštete funkcije JavaScript. Vsebino in strukturo spletne aplikacije podamo v obliki datoteke HTML. Vizualna podoba spletne aplikacije pa je v obliki CSS. Ker je oblikovanja malo, bomo stile vključili direktno v dokument HTML. V nadaljevanju sledi koda datoteke test-ajax.html, ki jo spletni brskalnik uporabniku prikaže kot spletno aplikacijo.

```
1.<html>
2.   <head>
3.      <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
4.      <title>Iskalnik po podatkovni bazi naročnikov, model AJAX</title>
5.      <script type="text/javascript" src="ustvariZahtevo.js"></script>
6.      <script type="text/javascript" src="posljiZahtevo.js"></script>
7.      <script type="text/javascript" src="posodobiStran.js"></script>
8.   </head>
9.   <body style=“margin: 100px auto; font-family: Verdana; font-size: 12px; color: #00000;“>
10.      <form name="obrazec">
11.         Poišči naročnika:
12.         <input id="poisciNarocnika" name="poisciNarocnika" value="" type="text" size="30" />
13.         <input value="Išči" type="button" onclick="posljiZahtevo();" />
14.      </form>
15.      <p>
16.         <div id="rezultatIskanja">
17.            Rezultat iskanja bo prikazan tukaj.
18.         </div>
19.      </p>
20.   </body>
21.</html>
```

Datoteka 8: test-ajax.html

Če primerjamo vsebino datoteke test.html, ki je pripadala spletni aplikaciji po klasičnem modelu in vsebino datoteke test-ajax.html, opazimo nekaj sprememb.

Vrstice 5, 6 in 7:

Na ta način spletnemu brskalniku omogočimo dostop do datotek JavaScript in s tem do funkcij, ki jih vsebujejo. Vse datoteke JavaScript vsebujejo le po eno funkcijo. Funkcije posameznih datotek bi lahko združili v eno datoteko JavaScript. Ločevanja v posamezne datoteke JavaScript smo se poslužili le zaradi lažjega sledenja po korakih modela AJAX.

Vrstica 10:

Sedaj ne želimo več, da spletna aplikacija pošlje podatke strežniku. Pošiljanje podatkov smo prepustili vmesnemu sloju AJAX. Ob potrditvi obrazca (kliku na gumb) se mora opraviti klic funkcije JavaScript posljiZahtevo(), ki opravi pošiljanje podatkov strežniku. Funkcija posljiZahtevo() je del vmesnega sloja AJAX. Da bi omogočili pošiljanje podatkov strežniku preko vmesnega sloja AJAX, znački form ne smemo več nastaviti lastnosti method.

Vrstica 13:

S tem, ko se v spletnem brskalniku naloži spletna aplikacija, se naložijo tudi vse funkcije JavaScript, ki smo jih vključili v glavi datoteke HTML (v predelu med značkama head). Vendar se s tem izvajanje tako vključene kode JavaScript ne začne. Izvajanje kode JavaScript sproži dogodek. V našem primeru je to klik na gumb iskalnika. Da pa bi lahko gumb iskalnika povezali z dogodkom, mora biti lastnost type elementa input, ki nam predstavlja gumb iskalnika naše spletne aplikacije, nastavljena na vrednost button. V datoteki test.html je bila lastnost type nastavljena na vrednost submit. To je pomenilo, da je ob kliku na ta gumb spletna aplikacija podatke obrazca avtomatsko posredovala strežniku. Pri modelu AJAX pa želimo, da posredovanje podatkov in komunikacijo s strežnikom opravi vmesni sloj AJAX. Zato smo vrednost type elementa input, ki predstavlja gumb iskalnika, sedaj nastavili na vrednost button. Elementu smo dodali še dogodek, t.j. onclik="posljiZahtevo();". S tem smo določili, da se ob kliku na ta element HTML opravi klic funkcije JavaScript posljiZahtevo().

Vrstica 16:

Elementu HTML div smo dodali lastnost id z vrednostjo rezultatIskanja. Preko te lastnosti id, bomo v funkciji JavaScript posodobiStran() dostopali do vsebine znotraj tega elementa.

S tem smo zaključili "preobrazbo" spletne aplikacije iz klasičnega modela v model AJAX. Kot smo že omenili, dana spletna aplikacija z modelom AJAX zaradi svoje preproste narave pri sami funkcionalnosti ne pridobi bistveno. Enostavno spletno aplikacijo smo vzeli le kot primer predstavitve modela AJAX.

V nadaljevanju bomo našo spletno aplikacijo po modelu AJAX še nadgradili in si ogledali dodatne možnosti.

4.Zahteva POST

V prejšnjem poglavju smo naredili enostavno spletno aplikacijo, iskalnik po bazi naročnikov in pri razvoju uporabili model AJAX.

Komunikacija med spletnim brskalnikom in strežnikom je bila asinhrona. Spletni brskalnik je strežniku podatke posredoval preko metode GET (glej komentar za vrstico 11 datoteke poslјiZahtevo.js). Podatke pa lahko pošiljamo tudi preko metode POST. V nadaljevanju si bomo podrobneje ogledali razliko med metodama GET in POST.

V predhodnem poglavju smo zahtevo HTTP poslali strežniku preko metode GET. Pošiljanje zahteve HTTP je izvedel vmesni sloj AJAX. V našem primeru je za to poskrbela funkcija posljiZahtevo(). Pri metodi GET strežnik dodatne podatke, ki jih potrebuje za izvršitev zahteve, prejme kot del naslova. Spomnimo se, da prvi del naslova (URL) strežniku pove, kateri datoteki s kodo naj posreduje zahtevo HTTP in podatke. Drugi del naslova pa so morebitni podatki, ki jih datoteka s kodo, ki se nahaja na naslovu iz prvega dela, potrebuje za izvršitev zahteve. Prvi in drugi del sta med seboj ločena z znakom '**?**'.

Spletni brskalnik in strežnik imata omejitve pri dolžini URL naslovov. Dolžina običajno ne sme biti večja od 2000 znakov. To se na prvi pogled morda zdi veliko, a npr. bi bolj kompleksnih iskanjih bi nam lahko zmanjkalo prostora za vse parametre. Če bi URL naslov presegel dovoljeno mejo, strežnik ne bo prejel vseh podatkov. V tem primeru se lahko zgodi dvoje. Prva možnost je, da bo strežnik prejel le podatke, ki so v okviru dovoljene meje, kar je čez mejo, pa ignoriral. Druga možnost pa je, da bo vrnil napako odjemalca. To v našem primeru pomeni, da bi vrednost lastnosti status objekta, ki predstavlja zahtevo HTTP, postala enaka 400 (glej tabelo Tabela 1 v prejšnjem poglavju).

Seveda v veliko primerih ni sprejemljiva nobena od teh dveh možnosti. Najti moramo način, kako brez omejitve dolžine prenesti podatke od odjemalca (spletnega brskalnika) do strežnika.

To nam omogoča metoda POST. Pri metodi POST odjemalec pošlje strežniku podatke ločeno od URL naslova. Podatke pred pošiljanjem zakodira. Če zahtevo pošljemo po metodi POST, to zahtevo imenujemo tudi zahteva POST. Ko strežnik prejme zahtevo POST, mora najprej poznati naravo podatkov, ki jih je prejel. Šele nato lahko prejete podatke odkodira in jih posreduje datoteki s kodo, ki se nahaja na URL naslovu, ki je naveden v zahtevi POST.

Informacijo o naravi podatkov, ki jih strežnik prejme skupaj z zahtevo HTTP, strežniku sporoči spletni brskalnik. To informacijo spletni brskalnik shrani v lastnost objekta zahteva, ki predstavlja zahtevo HTTP. Ta lastnost se imenuje request header. Ima več parametrov. Parameter, ki vsebuje informacijo o naravi podatkov, ki se pošiljajo strežniku, se imenuje Content-Type.

Oglejmo si primer vrednosti lastnosti request header.

```
1.Hypertext Transfer Protocol
2.    POST /poisciNarocnika.php HTTP/1.1
3.        Request Method: POST
4.        Request URI: /poisciNarocnika.php
5.        Request Version: HTTP/1.1
6.    Host: localhost
7.    Keep-Alive: 300
8.    Connection: keep-alive
9.    Content-Type:
10.        application/x-www-form-urlencoded
11.    Content-lenght: 121
```

Vsebina 1: Primer vrednosti lastnosti request header

V tem primeru so podatki torej tipa application/x-www-form-urlencoded.

Nekatere parametre lastnosti request header nastavi spletni brskalnik sam (npr. Host, Keep-Alive ...), nekatere pa moramo nastaviti sami. Tak parameter je tudi že prej omenjeni Content-Type, ki nosi informacijo o naravi podatkov, ki jih pošiljamo strežniku. Da bo strežnik lahko prejete podatke pravilno odkodiral in jih posredoval ustrezni datoteki, bomo torej morali nastaviti vrednost parametra Content-Type lastnosti request header objekta zahteva.

Poglejmo si, kako to naredimo v naši spletni aplikaciji.

Nadgradili bomo del, ki je odgovoren za pošiljanje zahteve HTTP strežniku. To je funkcija posljiZahtevo(), ki se nahaja v datoteki posljiZahtevo.js. Spremeniti moramo metodo pošiljanja zahteve HTTP iz GET v POST.

Oglejmo si spremenjeno funkcijo.

```
1.function posljiZahtevo() {
2.    // ustvarimo nov objekt za pošiljanje zahteve HTTP strežniku
3.    // objekt imenujemo zahteva (glej 1. korak)
4.    ustvariZahtevo();
5.
6.    // v isci shranimo iskani niz
7.    var isci = document.getElementById('poisciNarocnika').value;
8.
9.    // navedemo ime datoteke s programom, ki ga želimo izvršiti na strežniku
10.    var url = "poisciNarocnika-ajax.php";
11.
12.    // request.open izvede povezavo s strežnikom
13.    // "POST" definira, kako bomo poslali podatke strežniku
14.    // Vrednost spremenljivke url definira, kaj se bo na strežniku izvršilo.
15.    // Zadnji parameter definira vrsto komunikacije med spletnim brskalnikom
16.    //in strežnikom.
17.    // V tem primeru je komunikacija asinhrona. Parameter je spremenljivka tipa boolean.
18.    zahteva.open("POST", url, true);
19.
20.    // Ko strežnik konča z obdelavo zahteve, naj se izvrši
21.// funkcija posodobiStran().
22.    zahteva.onreadystatechange = posodobiStran;
23.
24.    // Strežniku povemo kakšne podatke naj pričakuje.
25.    zahteva.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");
26.
27.    // Zahtevo pošljemo strežniku z dodatnimi podatki.
28.    zahteva.send("poisciNarocnika="+isci);
29.}
```

Datoteka 9: posljiZahtevo.js

Oglejmo si še določene podrobnosti o spremembah, ki smo jih opravili.

Vrstica 10:

Ker bomo zahtevo HTTP poslali preko metode POST, odstranimo podatke, ki smo jih pri metodi GET pošiljali kot del URL naslova. Spremenljivka `url` sedaj vsebuje le naslov datoteke s kodo, ki se nahaja na strežniku, kamor bo zahteva HTTP posredovana. Spremenljivka `url` vsebuje le prvi del URL naslova, ki ga je vsebovala, preden smo izvedli popravke na funkciji posljiZahtevo().

Vrstica 17:

Na tem mestu navedemo, da bomo zahtevo HTTP poslali preko metode POST.

Vrstica 24:

Kadarkoli želimo strežniku povedati kaj o sami zahtevi HTTP, to storimo preko lastnosti request header. Z nastavitvijo vrednosti parametra Content-Type strežniku povemo, kakšna bo narava podatkov, ki mu bodo poslani z zahtevo.

Ukaz zahteva.setRequestHeader(parameter_ime, parameter_vrednost) nastavi parameter z imenom parameter_ime na vrednost parameter_vrednost. V danem primeru smo parameter "Content-Type" nastavili na vrednost "application/x-www-form-urlencoded". To pomeni, da bo zapis podatkov urejen po parih ime/vrednost.

Lastnost request header objekta zahteva nastavimo, preden zahtevo pošljemo strežniku.

Vrstica 27:

Pri opisu zahteve POST smo rekli, da se podatki pošljejo ločeno od URL naslova datoteke s kodo, ki naj zahtevo obdela. Z ukazom zahteva.send("poisciNarocnika="+isci) pošljemo na strežnik poleg zahteve

HTTP še podatek, t.j. vrednost iskanega niza. Podatki so, kot smo določili z ustrezno nastavitvijo Content-Type, posredovani v obliki urejenih parov, t.j. ime=vrednost.

Opišimo, kaj se dogaja, ko se ta funkcija izvede. Ko strežnik prejme zahtevo HTTP, pogleda vrednost lastnosti request header objekta zahteva. Iz vrednosti parametra Content-Type te lastnosti razbere podatek o tipu podatkov, ki jih je prejel z zahtevo HTTP. Če prepozna tip podatkov, prejete podatke odkodira in jih posreduje naprej programu na ustrezni datoteki. V nasprotnem primeru vrne napako.

Strežnik lahko spletnemu brskalniku poleg samega odgovora na zahtevo HTTP posreduje še nekaj dodatnih informacij o samem odgovoru. Dodatne informacije so koristne predvsem v primerih, ko pride do napake pri obdelovanju zahteve HTTP. V takih primerih je vsekakor bolje na voljo imeti več informacij.

Dodatne informacije v zvezi z odgovorom strežnik shrani v vrednost lastnosti response header objekta, ki predstavlja zahtevo HTTP, na katero strežnik pošilja odgovor. Lastnost response header ima, podobno kot lastnost request header, več parametrov.

Oglejmo si primer vrednosti lastnosti response header.

```
1.HTTP/1.1 Bad Request
2.    Request Version: HTTP/1.1
3.    Response Code: 400
4.Date: Fri, 27 Sept 2008 23:57:00 GMT
5.Server: Apache
6.X-Powered-By: PHP/4.4.9
7.StatusNapake: Iskani niz je prazen.
8.Connection: close
9.Transfer-Encoding: chunked
10.Content-Type: text/html
```

Vsebina 2: Primer vrednosti lastnosti response header

Vrstica 3:

Vrednost parametra Response Code nam pove odgovor strežnika. Iz tega lahko razberemo, ali je je bilo z obdelavo zahteve vse v redu (koda je 200), oziroma ali je prišlo do napake. V našem primeru je ta vrednost nastavljena na 400. Glede na opisu, ki smo ga navedli v tabeli Tabela 1, to pomeni, da je nekaj narobe s sintakso zahteve HTTP, ki jo je prejel strežnik.

Vrstica 7:

Vrednost parametra StatusNapake nam pove, kakšen je status objekta zahteva, ko je vrednost Response Code (glej komentar za vrstico 3) enaka 400. Ker smo parameter StatusNapake definirali sami in v splošnem ni prisoten (kako to storimo, si bomo ogledali v razlagi kode nadgrajene datoteke poisciNarocnika-ajax.php), se vrednost parametra StatusNapake nastavi le, ko je vrednost Response Code enaka 400. V vseh ostalih primerih ta parameter ne obstaja.

V naslednjem koraku bomo datoteko poisciNarocnika-ajax.php, ki se nahaja na strežniku in obdela zahtevo HTTP, nadgradili, da bo poleg samega odgovora na zahtevo HTTP poslala še nekaj informacij v zvezi z odgovorom. Uporabniku želimo sporočiti, če je kliknil na gumb "Išči", v iskalno polje pa ni vpisal ničesar. Seveda bi lahko to napako prestregli že na strani odjemalca (v brskalniku), a želimo pokazati, kako to storimo na strežniški strani.

Sledi popravljena vsebina dokumenta poisciNarocnika-ajax.php. Čeprav se prejšnja in sedanja verzija dokumenta poisciNarocnika-ajax.php razlikujeta le v treh vrsticah (dodali smo vrstice 19, 20 in 21), bomo za lažje sledenje navedli celotno vsebino sedanje verzije dokumenta poisciNarocnika-ajax.php.

```
1.<?php
2.// povezava s podatkovno bazo
3.mysql_connect("localhost", "root", "");
4.mysql_select_db("primeri");
5.// na ta način povemo MySQL-lu kakšne podatke naj pričakuje
6.mysql_query("SET NAMES 'utf8'");
7.mysql_query("SET CHARACTER SET utf8");
8.// dobimo podatke, ki so bili vneseni v vnosno polje obrazca
9.// vrednost, ki jo dobimo spravimo v spremenljivko $poisciNarocnika
```

```
10.if (isset($_REQUEST["poisciNarocnika"])) {
11.    $poisciNarocnika = $_REQUEST["poisciNarocnika"];
12.} else {
13.    $poisciNarocnika = '';
14.}
15.// spremenljivka $rezultatIskanja bo nosila vrednost rezultata iskanja
16.$rezultatIskanja = '';
17.if ($poisciNarocnika == '') {
18.    // če je iskani niz prazen
19.    header("StatusNapake: Iskani niz je prazen.", true, 400);
20.    echo " ";
21.    exit;
22.} else {
23.    // v nasprotem primeru, ga primerjamo s podatki v podatkovni bazi
24.    $sql = mysql_query("SELECT * FROM narocniki WHERE ime LIKE '%".$poisciNarocnika."%' OR priimek LIKE
'%".$poisciNarocnika."%' OR eposta LIKE '%".$poisciNarocnika."%' OR telefon LIKE '%".$poisciNarocnika."%' OR ulica LIKE
'%".$poisciNarocnika."%' OR posta LIKE '%".$poisciNarocnika."%' OR mesto LIKE '%".$poisciNarocnika."%' ORDER BY priimek ASC");
25.    // spremenljivka $sqlRezultat nosi vrednost rezultata iskanja
26.    // $sqlRezultat bo vseboval nekaj 'lepotnih' popravkov pri izpisu
27.    while ($narocnik = mysql_fetch_object($sql)) {
28.        // za večjo preglednost, bomo rezultat predstavili kot tabelo
29.        $sqlRezultat .= '<tr>';
30.        $sqlRezultat .= '<td colspan="2" style="background-color: #000000; color: #FFFFFF; font-weight: bold;">';
31.        $sqlRezultat .= $narocnik->ime.' '.$narocnik->priimek;
32.        $sqlRezultat .= '</td>';
33.        $sqlRezultat .= '</tr>';
34.        $sqlRezultat .= '<tr><td>Ulica</td>';
35.        $sqlRezultat .= '<td>'.$narocnik->ulica.'</td></tr>';
36.        $sqlRezultat .= '<tr><td>Pošta</td>';
37.        $sqlRezultat .= '<td>'.$narocnik->posta.'</td></tr>';
38.        $sqlRezultat .= '<tr><td>Mesto</td>';
39.        $sqlRezultat .= '<td>'.$narocnik->mesto.'</td></tr>';
40.        $sqlRezultat .= '<tr><td>E-pošta</td>';
41.        $sqlRezultat .= '<td>';
42.        $sqlRezultat .= '<a href="mailto:'.$narocnik->eposta.'">';
43.        $sqlRezultat .= $narocnik->eposta;
44.        $sqlRezultat .= '</a>';
45.        $sqlRezultat .= '</td></tr>';
46.        $sqlRezultat .= '<tr><td>Telefon</td>';
47.        $sqlRezultat .= '<td>'.$narocnik->telefon.'</td></tr>';
48.    }
49.    if ($sqlRezultat == '') {
50.        // če v podatkovni bazi nismo našli ujemanja,
51.        // moramo to sporočiti uporabniku
52.        // sporočilo uporabniku nosi spremenljivka $rezultatIskanja
53.        $rezultatIskanja = 'V podatkovni bazi ni zapisa, ki bi ustrezal.';
54.        $rezultatIskanja .= '<br />';
55.        $rezultatIskanja .= 'Prosimo, poizkusite znova.';
56.    } else {
57.        // v nasprotnem primeru še malo popravimo format izpisa rezultata
58.        // spremenljivka $sqlRezultat nosi vrstice tabele
59.        // za pravilen izpis moramo dodati še začetek in konec tabele
60.        $rezultatIskanja = '<table style=“border: 1px solid #000000; font-family: Arial, Verdana; font-size: 12px; width: 500px;“
cellpadding=“5“ cellspacing=“5“>';
61.        $rezultatIskanja .= $sqlRezultat;
62.        $rezultatIskanja .= '</table>';
63.    }
64.}
65.// izpišemo rezultat iskanja
66.   echo $rezultatIskanja;
67.// konec datoteke poisciNarocnika-ajax.php
68.?>
```

Datoteka 10: poisciNarocnika-ajax.php

Kot smo omenili, smo dodali le vrstice 19, 20 in 21. Te vrstice se izvedejo, če uporabnik vnese prazen iskalni niz. Ostala vsebina dokumenta poisciNarocnika-ajax.php je ostala nespremenjena.

Vrstica 19:

"StatusNapake" postane ime parametra lastnosti response header objekta zahteva. Vsebina, ki je navedena za "StatusNapake:" postane del sporočila, ki je vrnjeno kot vrednost lastnosti response header. V našem primeru je to vsebina "Iskani niz je prazen.". Parameter StatusNapake smo definirali sami in v splošnem ni del vrednosti lastnosti response header.

Parameter tipa boolean z vrednostjo true pomeni, da bo vrednost parametra StatusNapake nadomestila že

obstoječo, če le ta parameter obstaja. To pomeni, da če bi lastnost response header že vsebovala parameter z imenom StatusNapake, bi se vrednost tega parametra nadomestila z vrednostjo, ki smo jo definirali mi.

Zadnji parameter nastavi vrednost lastnosti status objekta zahteva. Nastavili smo jo na 400. Spomnimo se, da ta lastnost pove, kaj se je med izvajanjem na strežniku zgodilo z zahtevo HTTP, ki jo predstavlja objekt zahteva. Če je vrednost enaka 400, se zahteva HTTP ni uspešno izvršila, saj je sintaksa zahteve HTTP napačna.

Vrstica 20:
Nekateri spletni brskalniki, na primer Safari, javijo napako, da je koda lastnosti status objekta zahteva nedefinirana, če strežnik kot odgovor na zahtevo ne vrne ničesar. Zato takrat, ko je prišlo do napake (ki jo javimo s tem, da v vrstici 19 lastnost status nastavimo na 400), kot odgovor vrnemo prazen niz. Odgovor strežnika ni dejanski odgovor na zahtevo, ampak le način, da zagotovimo, da bo spletni brskalnik uporabniku javil pravilno kodo vrednosti lastnosti status objekta zahteva.

Ostala vsebina se ni spremenila.

Sedaj je potrebno nadgraditi vsebino funkcije posodobiStran(), ki prejme odgovor strežnika. Funkcijo posodobiStran() želimo nadgraditi tako, da bo uporabniku v primeru napake znala prikazati dodatne podatke, ki jih, poleg odgovora na zahtevo HTTP, vrne strežnik.

```
1.function posodobiStran() {
2.    if (zahteva.readyState == 4) {
3.        if (zahteva.status == 200) {
4.            // dobimo odgovor strežnika
5.            var odgovorStreznika = zahteva.responseText;
6.
7.            // posodobimo stran z vsebino odgovora strežnika
8.            document.getElementById("rezultatIskanja").innerHTML = odgovorStreznika;
9.        } else {
10.            var sporocilo = zahteva.getResponseHeader("StatusNapake");
11.            if (sporocilo.length == null) {
12.                alert ('Napaka pri vračanju rezultatov! Strežnik je vrnil kodo' + zahteva.status + '.');
13.            } else {
14.                alert (sporocilo);
15.            }
16.        }
17.    }
18.}
```

Datoteka 11: posodobiStran.js

Vrstica 9:
Če med odgovarjanjem na zahtevo HTTP, ki jo predstavlja objekt zahteva, pride do napake, strežnik vrne kodo, s katero javi napako. Napako smo javili tudi s tem, ko smo v primeru uporabe praznega iskalnega niza v 19. vrstici poisciNarocnika-ajax.php kodo nastavili na 400. Takrat vrednost lastnosti status objekta zahteva ne bo enaka 200. Zato se bo, bodisi ko pride do "prave" napake, bodisi v primeru uporabe praznega niza, koda funkcije posodobiStran() izvršila v tem delu.

Vrstica 10:

Na ta način dobimo vrednost parametra StatusNapake lastnosti response header objekta zahteva. Če je pri obdelavi zahteve prišlo do napake iz drugega vzroka (ne zaradi uporabe praznega niza), te lastnosti seveda ni. Takrat metoda getResponseHeader za odgovor vrne vrednost null.

Vrstica 11 do 13:
Če parameter StatusNapake lastnosti response header objekta zahteva ni nastavljen, izpišemo sporočilo napake.

Vrstica 13 do 15:

Če je strežnik oz. koda datoteke na strežniku, v našem primeru poisciNarocnika-ajax.php, nastavil parameter StatusNapake lastnosti response header objekta zahteva, potem prikažimo vrednost parametra StatusNapake uporabniku. V našem primeru je vrednost parametra Status enaka "Iskani niz je prazen.". Torej se bo uporabniku v pojavnem oknu izpisalo "Iskani niz je prazen.".

Z danimi popravki bo naša spletna aplikacija ob napakah bolje obveščala uporabnika.

V tem poglavju smo razvoj naše enostavne spletne aplikacije, iskalnika po podatkovni bazi naročnikov, pripeljali do konca.
Še enkrat pa poudarimo, da smo namenoma izbrali zelo enostaven primer in bi v praksi seveda zadevo izvedli drugače. Kljub temu pa ta enostavni primer dobro ponazarja osnovne prijeme, ki jih uporabljamo pri razvoju spletnih aplikacij po modelu AJAX.

V nadaljevanju sledi kratek pregled obstoječih knjižnic, ki nam lahko olajšajo delo pri razvoju spletnih aplikacij po modelu AJAX.

5.Knjižnice

V tem poglavju bomo naredili kratek pregled knjižnic, ki podpirajo tudi spletne aplikacijah narejene po modelu AJAX. Namen tega poglavja je predvsem vzbuditi razmišljanje in zanimanje za možnosti, ki nam jih ponuja tehnika AJAX in ne natančen pregled posameznih knjižnic.

Knjižnica je skupek objektov in funkcij zapisanih v jeziku JavaScript in shranjenih v datoteki ali v več njih. Namenjena je gradnji interaktivnih spletnih aplikacij. Za delo s posamezno knjižnico si na strežnik prenesemo njene datoteke JavaScript. Datoteke na strežnik prenesemo zato, da bodo dostopne tudi uporabnikom naše spletne aplikacije. V nasprotnem primeru bi si moral vsak uporabnik naše spletne aplikacije na svoj računalnik prenesti datoteke knjižnice, ki jo naša spletna aplikacija uporablja ali pa spletna aplikacija ne bi ustrezno delovala.

Datoteke JavaScript posamezne knjižnice vključimo v našo spletno aplikacijo kot smo vključili datoteke JavaScript za delo z objektom zahteva v datoteki test-ajax.html. Namen knjižnic je predvsem to, da nam olajšajo delo in prihranijo čas. Včasih se zgodi, da v določeni knjižnici že obstaja objekt s funkcionalnostmi, ki jih potrebujemo za našo spletno aplikacijo. V tem primeru je smiselno razmisliti o uporabi knjižnice, ki vsebuje ta objekt in s tem prihraniti čas, ki bi ga sicer namenili samostojnemu razvoju.

Čeprav so knjižnice le skupek med seboj povezanih datotek JavaScript, pa moramo biti pri njihovi uporabi previdni. Priporočljivo je namreč uporabljati le eno knjižnico naenkrat, da ne pride do neprijetnosti pri prekrivanju metod. Zato se, preden se odločimo za uporabo posamezne knjižnice, poglobimo v to, kaj nam nudi.

Kako hitro se posamezna knjižnica razvija (nastajajo nove različice) je odvisno predvsem od njene uporabnosti, razširjenosti in podpore, ki jo je deležna.

V nadaljevanju sledi kratek opis nekaterih knjižnic. Za več informacij o posamezni knjižnici obiščite spletno stran, katere naslov URL je naveden poleg vsake knjižnice.

- *Dojo*

  Začetni avtor: Alex Russell (2004)

  Spletna stran: http://dojotoolkit.org/

  Knjižnica Dojo nam nudi veliko različnih funkcionalnosti, ki jih lahko uporabljamo pri gradnji spletnih aplikacij po modelu AJAX. Knjižnica Dojo vsebuje že pripravljene vizualne gradnike, ki jih lahko enostavno uporabimo pri gradnji svojih spletnih aplikacij. Primer takega vizualnega gradnika bi bil koledar, ki se odziva na klike uporabnika.

  Sledi grafični prikaz demonstrativnega primera funkcionalnosti animacije, ki jo spletni aplikaciji omogoča Dojo.

Slika 29: Dojo, animacija

  Naslov URL, kjer se nahaja primer: http://dojotoolkit.org/demos/dojo-moj-oe.

  Začetno stanje elementov spletne aplikacije je prikazano na levi polovici slike Slika 29. Ko uporabnik odkljuka možnost "Gravity" (op. gravitacija), grafični elementi "krogi" spremenijo položaj na način, kot da bi na njih deloval učinek gravitacije. Končno stanje spremembe položaja je prikazano na desni polovici slike Slika 29. Prehod med stanjema, ki ju prikazuje slika Slika 29, je zvezen. Torej gre za

animacijo.

- *Mochikit*

  Spletna stran: http://www.mochikit.com/

  Mochikit predstavlja skupek dobro dokumentiranih in testiranih knjižnic, katerih uporaba spletnim aplikacijam omogoča asinhrono upravljanje z nalogami, upravljanje s časom, formatiranje nizov, uporabo funkcionalnosti "povleci in spusti", preproste vizualne učinke in še mnogo drugega.

  Sledi grafični prikaz demonstrativnega primera, ki jo spletni aplikaciji omogoča Mochikit.

  Slika 30: Mochikit, zaobljeni robovi

  Naslov URL, kjer se nahaja primer: http://www.mochikit.com/examples/rounded_corners/index.html.

  Mochikit v tem primeru spletni aplikaciji omogoča enostavno kreiranje elementov z zaobljenimi robovi. V splošnem bi to reševali na drugačen, bolj kompliciran način.

- *Prototype*

  Začetni avtor: Sam Stephenson (2004)

  Spletna stran: http://prototypejs.org/

  Prototype je namenjen podpori gradnji dinamičnih spletnih aplikacij. Vsebuje objekte za delo z zahtevo HTTP. Vsebuje tudi razširitve nekaterih objektov, ki jih najdemo v jeziku JavaScript. Tako nam omogoča dodatne funkcionalnosti in nam olajša delo. Primer takega objekta je Array, ki ga uporabljamo za delo s tabelami.

  Prototype kot sam ne nudi nobenih vizualnih učinkov in podobno, zato bomo v tem primeru izpustili grafični prikaz demonstracijskega primera. Vizualne učinke nam omogočajo knjižnice, ki razširjajo knjižnico Prototype. Več o tem sledi v nadaljevanju.

Kot smo že omenili, obstajajo tudi knjižnice, ki razširjajo že obstoječe knjižnice. Za uporabo le teh potrebujemo tudi datoteke, ki jih vsebuje primarna knjižnica, t.j. knjižnica, ki smo jo razširili.

V nadaljevanju sledita dva primera takih knjižnic.

- *OpenRico*

  Začetna avtorja: Bill Scott, Darren James (2005)

  Spletna stran: http://openrico.org/

  Primarna knjižnica: Prototype

  Knjižnica OpenRico nam nudi veliko različnih funkcionalnosti, ki jih lahko uporabljamo pri gradnji spletnih aplikacij po modelu AJAX. OpenRico nudi podporo za področje uporabniških vmesnikov in nam med drugim omogoča preproste animacije, uporabo učinka "povleci in spusti", različne vizualne gradnike in podobno. OpenRico uporablja nekatere objekte in funkcije iz knjižnice Prototype.

  Sledi grafični prikaz demonstrativnega primera, ki jo spletni aplikaciji omogoča OpenRico.

Slika 31: OpenRico, učinek "povleci in spusti"

Naslov URL, kjer se nahaja primer: http://demos.openrico.org/demos/drag_and_drop_simple.

OpenRico v tem primeru spletni aplikaciji omogoča uporabo učinka "povleci in spusti". Uporabnik lahko z miško "prime" element, ki vsebuje napis "Drag Me" (op. v prevodu "Povleci me") in ga prenese v element z napisom "Drop On Me" (op. v prevodu "Spusti name") ali v element z napisom "Drop On Me 2".

- *Scriptaculous*

  Začetni avtor: Thomas Fuchs (2004)

  Spletna stran: http://script.aculo.us/

  Primarna knjižnica: Prototype

  Knjižnica Scriptaculous je namenjena predvsem bogatenju spletnih aplikacij z učinki animacije in gradnji uporabniških vmesnikov. Scriptaculous uporablja nekatere objekte in funkcije iz knjižnice Prototype.

  Sledi grafični prikaz demonstrativnega primera, ki jo spletni aplikaciji omogoča Scriptaculous.

Slika 32: Scriptaculous, naključna menjava vrstnega reda

Naslov URL, kjer se nahaja primer: http://github.com/madrobby/scriptaculous/wikis/list-morph-demo.

Na sliki Slika 32 je grafični prikaz stanj elementov seznama, ki ga sestavljajo nizi "Lorem", "dolor", "ipsum", "sit" in "amet". Ko uporabnik klikne na enega izmed elementov seznama, se vrstni red elementov naključno spremeni. Pri menjavi vrstnega reda elementov pride do animacije.

Vse navedene knjižnice so na voljo brezplačno. Za boljšo predstavo o tem kaj nudijo, priporočamo ogled njihovih spletnih strani, kjer so na vpogled primeri uporabe in demonstracijske spletne aplikacije.

Za primerjavo med nekaterimi funkcionalnostmi, ki jih nudijo oz. ne nudijo navedene knjižnice, sledi primerjalna tabela Tabela 2.

Funkcionalnost \ Knjižnica	Dojo	Mochikit	OpenRico *	Scriptaculous *
Učinek "povleci in spusti"	da	da	da	da
Preprosti vizualni učinki	da	da	da	da
Animacija / napredni vizualni učinki	da	da	da	da
Operiranje z dogodki	da	da		da
Orodja za urejanje obogatenega besedila	da			
Orodja za generiranje kode HTML	da			da

Tabela 2: Primerjalna tabela knjižnic

Iz tabele Tabela 2 smo izpustili knjižnico Prototype, saj v glavnem nudi le osnovo za razširjeni knjižnici, OpenRico in Scriptaculous. Ker knjižnici OpenRico in Scriptaculous navedene funkcionalnosti podpirata le s podporo knjižnice Prototype, smo ju v tabeli Tabela 2 označili z znakom "*".

Podatki v tabeli Tabela 2 nam nudijo le grobo primerjavo med posameznimi knjižnicami. Lahko se namreč zgodi, da dve knjižnici nudita podporo enake funkcionalnosti. Vendar je lahko implementacija te funkcionalnosti v našo spletno aplikacijo v eni izmed teh knjižnic lažja.

S tem zaključujemo kratek pregled knjižnic.

6.Zaključek

Ob pisanju diplomske naloge smo prišli do zaključka, da gre razvoj spletnih aplikacij v smeri izboljšanja uporabniške izkušnje. Tej smeri zagotovo sledi tudi tehnika AJAX.

Ena izmed prednosti tehnike AJAX je tudi v tem, da ima široko podporo okolja spletnih razvijalcev in nekaterih velikih podjetij (na primer podjetja Google). Rezultat te podpore je veliko število prosto dostopnih knjižnic, ki razvijalcem močno olajšajo delo. Ker je razvoj spletnih aplikacij s pomočjo teh knjižnic lažji, se vse več spletnih razvijalcev pri odločitvi med klasičnim modelom ali modelom AJAX, odloči za slednjega. Tako nastaja vedno več spletnih strani narejenih po modelu AJAX.

Na podlagi vsega navedenega lahko sklepamo, da bo tehnika AJAX aktualna še vsaj nekaj časa. Vendar pa je internet "živ organizem" in bo na vpogled v prihodnost vloge tehnike AJAX pri razvoju spletnih aplikacij potrebno še počakati.

7.Literatura in spletni viri

7.1.Literatura

- Brett McLaughlin: *Head Rush Ajax*, O'Reilly Media, Inc., 2006
- Dave Crane, Eric Pascarello in Darren James: *Ajax in action*, Manning Publications Co., 2006
- Lee Babin: *Beginning Ajax with PHP: From Novice to Professional*, Apress, 2007

7.2.Spletni viri

Vsi spletni viri so bili zadnjič dostopani 2. 11. 2008.

Spletne strani z vsebinami o tehniki AJAX
- http://www.ajaxmatters.com/
- http://ajaxian.com/
- http://www.w3schools.com/ajax/default.asp

Definicije vrednosti lastnosti status objekta zahteva
- http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html

Definicije metod za prenos podatkov
- http://www.w3.org/Protocols/rfc2616/rfc2616-sec9.html

Definicija lastnosti objekta zahteva request header
- http://www.w3.org/Protocols/HTTP/HTRQ_Headers.html

Definicija lastnosti objekta zahteva response header
- http://www.w3.org/Protocols/rfc2616/rfc2616-sec6.html

Knjižnice
- http://dojotoolkit.org/
- http://www.mochikit.com/
- http://www.prototypejs.org/
- http://openrico.org/
- http://script.aculo.us/
- http://chandlerproject.org/Projects/AjaxLibraries