Универзитет
Сингидунум




Младен Веиновић
Александар Јевремовић

# ИНТЕРНЕТ ТЕХНОЛОГИЈЕ

Београд, 2020.

УНИВЕРЗИТЕТ СИНГИДУНУМ

ФАКУЛТЕТ ЗА ИНФОРМАТИКУ И РАЧУНАРСТВО

Александар Јевремовић
Младен Веиновић

# ИНТЕРНЕТ ТЕХНОЛОГИЈЕ

Четврто издање

Београд, 2020.

**ИНТЕРНЕТ ТЕХНОЛОГИЈЕ**

*Аутори:*
др Александар Јевремовић
др Младен Веиновић

*Рецензенти:*
др Милан Милосављевић
др Бранко Ковачевић
др Горан Шимић

*Издавач:*
УНИВЕРЗИТЕТ СИНГИДУНУМ
Београд, Данијелова 32
www.singidunum.ac.rs

*За издавача:*
др Милован Станишић

*Техничка обрада:*
Александар Јевремовић

*Дизајн корица:*
Александар Михајловић

*Година издања:*
2020.

*Тираж:*
800 примерака

*Штампа:*
Caligraph, Београд

ISBN: 978-86-7912-648-1

Copyright:
© 2020. Univerzitet Singidunum
Izdavač zadržava sva prava.
Reprodukcija pojedinih delova ili celine ove publikacije nije dozvoljeno.

# Садржај

*1. Интернет мрежа и технологије ... 1*
*1.1. Комуникациони модел Интернет мреже ... 2*
*1.1.1. Нивои рада мрежних уређаја ... 4*
*1.1.2. Пакетски пренос и инкапсулација ... 5*
*1.1.3. Технологије за приступ Интернет мрежи ... 7*
*1.1.4. Међумрежно повезивање ... 9*
*1.1.5. Транспортовање података ... 12*
*1.1.6. Кориснички сервиси ... 14*
*1.2. Адресовање на Интернету ... 15*
*1.2.1. Адресовање на нивоу локалне мреже ... 15*
*1.2.2. Међумрежно адресовање ... 17*
*1.2.3. Систем доменских имена ... 18*
*1.2.3.1. Историјат проблема и решења ... 18*
*1.2.3.2. Теорија рада система доменских имена ... 21*
*1.2.3.3. Кеширање код система доменских имена ... 24*
*1.2.3.4. Типови записа ... 26*
*1.2.3.5. Процес регистрације домена ... 26*
*1.2.4. Униформни локатори ресурса ... 27*
*1.2.5. Адресовање електронске поште ... 29*
*1.3. Мрежно програмирање ... 31*
*1.3.1. Софтверске архитектуре ... 31*
*1.3.1.1. Клијент-сервер архитектура ... 34*
*1.3.1.1.1. Итеративна и конкурентна обрада захтева ... 36*
*1.3.1.2. Архитектура равноправних чланова ... 37*
*2. Веб и Веб технологије ... 39*
*2.1. Клијент-сервер Веб модел ... 39*
*2.2. Трослојна архитектура ... 41*
*2.3. Сајтови, презентације, апликације и сервиси ... 42*
*2.4. Развојна и продукциона окружења ... 44*
*2.4.1. Развојно окружење ... 45*
*2.4.2. Продукционо окружење ... 47*
*2.4.2.1. Дељени сервер ... 48*
*2.4.2.1.1. Безбедносни аспекат ... 52*
*2.4.2.2. Виртуални приватни сервери ... 54*
*2.4.2.3. Удомљавање сервера ... 56*
*2.4.2.4. Изнајмљивање брзе везе са Интернетом ... 58*
*2.4.2.5. Мреже за испоруку садржаја ... 59*
*2.4.3. Доступност ... 60*

*2.4.4. Управљање оптерећењем и скалабилност ... 61*
*2.4.4.1. Пример вертикалног скалирања ... 63*
*3. Језик за означавање хипер-текста ... 65*
*3.1. Структура и садржај хипертекстуалних докумената ... 65*
*3.1.1. Хијерархијска структура HTML докумената ... 67*
*3.2. Повезивање докумената путем хипер-веза ... 68*
*3.3. Уређивање текста ... 71*
*3.3.1. Параграфи и основни текст ... 72*
*3.3.2. Заглавља ... 75*
*3.3.3. Додавање слика у документе ... 75*
*3.3.4. Унос специјалних симбола ... 76*
*3.3.5. Нумерисане и ненумерисане листе ... 77*
*3.3.6. Табеле ... 79*
*3.4. Формулари ... 81*
*3.4.1. Поља за унос текста од стране корисника ... 82*
*3.4.2. Контроле за избор предефинисаних вредности ... 83*
*3.4.3. Дугмад ... 86*
*3.4.4. Слање фајлова путем формулара ... 87*
*3.4.5. Означавање и груписање контрола ... 88*
*3.4.6. Слање садржаја формулара ... 89*
*4. Програмирање на страни сервера ... 91*
*4.1. Програмски језик PHP ... 92*
*4.1.1. Извршно окружење ... 93*
*4.1.2. Конфигурациони фајлови ... 94*
*4.1.3. Управљање грешкама ... 95*
*4.2. Основе програмирања ... 96*
*4.2.1. Променљиве и константе ... 96*
*4.2.1.1. Видљивост променљивих ... 98*
*4.2.2. Употреба наводника ... 99*
*4.2.3. Низови и матрице ... 100*
*4.2.4. Структуре гранања ... 102*
*4.2.4.1. If/else структура гранања ... 102*
*4.2.4.2. Switch/case структура гранања ... 103*
*4.2.5. Цикличне структуре ... 104*
*4.2.6. Кориснички дефинисане функције ... 105*
*4.2.7. Објектно оријентисано програмирање ... 106*
*4.2.8. Разлагање апликације на више фајлова ... 107*
*4.2.9. Коментари ... 107*
*4.2.10. Правила кодовања ... 108*
*4.3. Комуникација са окружењем ... 109*

*4.3.1. Обрада података из формулара* ... *111*
*4.3.2. Рад са фајловима* ... *112*
*4.3.2.1. Руковање фајловима из формулара* ... *114*
*4.3.3. Рад се базама података* ... *115*
*4.3.3.1. Повезивање са сервером базе података* ... *115*
*4.3.3.2. Додавање података у базу* ... *116*
*4.3.3.3. Учитавање података из базе* ... *118*
*4.3.3.4. Измена података у бази* ... *121*
*4.3.3.5. Уклањање података из базе* ... *125*
*4.3.4. Рад са колачићима* ... *126*
*4.3.5. Управљање сесијом и сесијском меморијом* ... *127*
*4.4. Контрола приступа апликацији* ... *129*
*4.4.1. Контрола приступа на нивоу директоријума* ... *129*
*4.4.2. Контрола приступа на нивоу апликације* ... *131*
*4.5. Раздвајање логике и приказа* ... *135*
*4.6. Рад са графичким садржајем* ... *140*
*4.7. Рад са електронском поштом* ... *143*
*4.8. Серијализација података и Веб сервиси* ... *145*
*4.8.1. Серијализација* ... *146*
*4.8.2. XML* ... *148*
*4.8.3. JSON* ... *150*
*4.8.4. SOAP* ... *151*
*4.9. Отклањање грешака и повећање перформанси* ... *153*
*4.9.1. Профилисање* ... *154*
*4.9.2. Кеширање података из базе података* ... *157*
*4.9.3. Кеширање операционог кода* ... *159*
*4.9.4. Заштита изворног кода* ... *161*
*5. Литература* ... *163*

# 1. Интернет мрежа и технологије

Интернет мрежа представља јединствени комуникациони систем данашњице, као и јединствени комуникациони систем познат општој историји људске цивилизације. Ни један други систем се не може похвалити таквом географском покривеношћу, брзином еволуције, као ни разнородношћу технологија и типова комуникационих садржаја као што је то случај код ове мреже.

Интернет мрежи приступа сваке године све већи проценат становника на нашој планети, као и све већи број уређаја. Процењује се[1] да је крајем 2011. године приступ глобалној мрежи имало преко 2,2 милијарде људи. Првих 50 милиона корисника Интернет мрежа је остварила већ након пет година од свог настанка. За остваривање истог броја корисника медију као што је радио било је потребно тридесет осам година а телевизији тринаест.

Капацитет Интернет мреже као медија такође бележи значајан раст сваке године. Брзина којом корисници приступају Интернету је у последње две деценије расла за око 50% сваке године. Тај раст се директно одражава и на мултимедијално богатство садржаја који се том мрежом преносе, односно којима се путем ње може приступити.

Један од значајних чинилаца успеха Интернет мреже је и могућност њеног асиметричног и асинхроног развоја на свим нивоима. На пример, крајњи корисници Интернета у развијеним земљама су у могућности да себи приуште везе и до хиљаду пута веће брзине него што су то у могућности корисници у земљама у развоју. То их, међутим, не ограничава да међусобно комуницирају путем сервиса са мултимедијално ограниченијим садржајима. Са друге стране, слојевити дизајн комуникационог модела Интернет мреже омогућава унапређивање технологија на једном слоју без опасности по нарушивање компатибилности са актуелним технологијама на осталим слојевима.

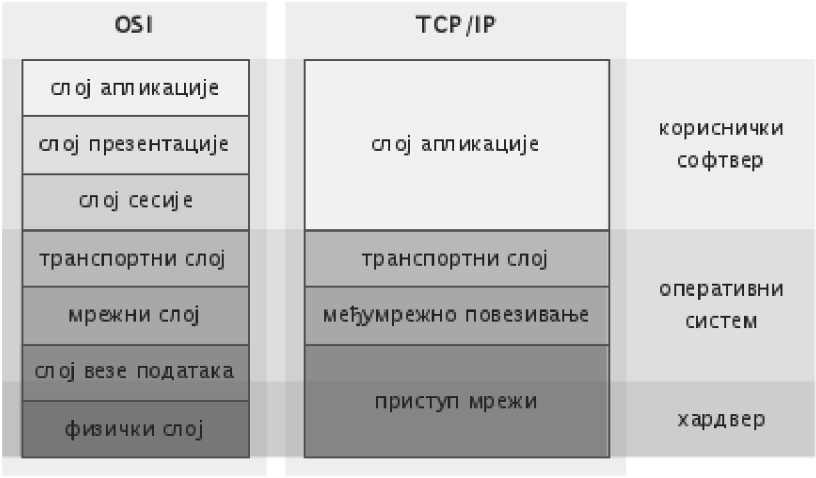
Са инжењерске тачке гледишта најзначајнију карактеристику Интернет мреже чини њена способност да омогући готово тренутну комуникацију између најразличитијих уређаја, који се могу налазити на различитим крајевима света. Функционисање Интернет мреже је засновано на великом броју технологија. Главни инжењерски изазов је управо организација ових технологија за омогућавање комуникације. Технологије које користи Интернет мрежа организоване су по *TCP/IP* комуникационом моделу, познатом и под називом Интернет комуникациони модел.

1 http://www.internetworldstats.com

## 1.1. Комуникациони модел Интернет мреже

Свака комуникација - не само рачунарска - може се посматрати у виду слојева. На пример, уколико једна особа жели да нешто саопшти другој, она на највишем нивоу има апстрактну поруку, ближу потреби за саопштењем нечега него уобличеној поруци. Она, затим, бира одговарајући облик за формулисање поруке - вербални, музички, сликовни... Након формулисања поруке изабира се најприкладнији начин кодовања поруке и медиј за њен пренос - на пример кодовање музичке поруке у нотном систему и запис на папиру или њено извођење на одређеном музичком инструменту.

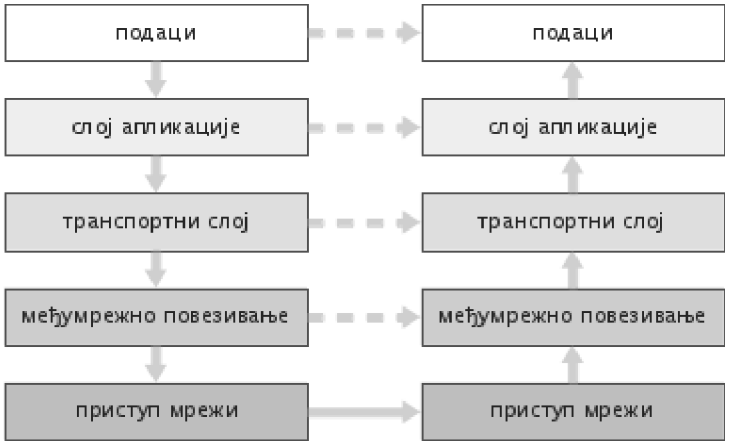
Слојевитост код комуникације у рачунарским мрежама је стандардизована. Формални стандард за рачунарске телекомуникације дефинисала је Међународна организација за стандардизацију (енгл. *International Organization for Standardization, ISO*) у виду референтног модела за отворено повезивање система (енгл. *Open Systems Interconnection, OSI*). Овај модел дели рачунарску мрежну комуникацију на седам слојева у зависности од специфичног задатка који се обавља: физички слој, слој везе података, мрежни слој, транспортни слој, слој сесије, слој презентације и слој апликације.




*Слика 1. Референтни комуникациони модели и нивои имплементације*

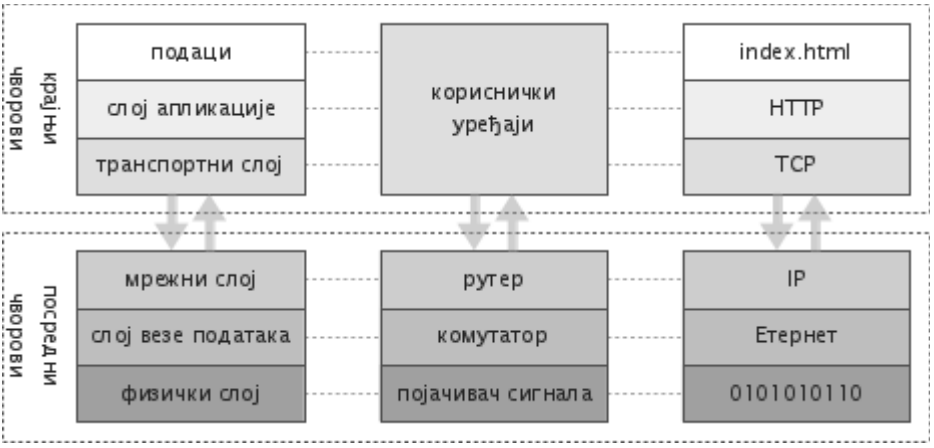
С обзиром да детаљност коју нуди, подела *OSI* модела најчешће није потребна. У пракси се за опис комункације често користи једноставнији модел, познат под називом ***TCP/IP*** **скуп протокола** (енгл. *TCP/IP Protocol Suite*) или **Интернет модел**. Овај модел не залази у детаље комуникације унутар корисничког сервиса (апликације), те слојеве сесије и презентације интегрише у сам слој

апликације. Додатно, унутар њега се не врши подела телекомуникационе инфраструктуре на физички слој и слој везе података већ су они обједињени у функционалну целину задужену за приступ рачунарској мрежи.




*Слика 2. Вертикална и хоризонтална комуникација слојева*

Подела рачунарске комуникације на слојеве омогућава фокусирање на издвојени задатак слоја при развоју телекомуникационог протокола или медија, односно изузимање из вида детаља осталих слојева и апстракцију до нивоа комуникације са имплементацијом решења на другој комуникационој страни. Оваква комуникација, комуникација између страна које комуницирају, назива се **хоризонтална комуникација**.




*Слика 3. Паралела између комуникационих слојева, уређаја и протокола*

Са друге стране, сваки слој добија задатке (податке које треба пренети) од

слојева виших нивоа, а за њихово извршавање користи услуге слојева нижих нивоа. Оваква комуникација - комуникација између различитих слојева унутар једног рачунара - назива се **вертикална комуникација**. Вертикална комуникација је неодвојива од принципа инкапсулације и декапсулације пакета.

Подела рачунарске комуникације на више нивоа омогућава виши ниво специјализације при развоју мрежних уређаја, односно уређаја који омогућавају рачунарске телекомуникације. За одређивање нивоа на коме ради одређени уређај узима се највиши слој *OSI* модела чије је јединице података уређај у стању да разуме, уз подразумевање чињенице да разуме и све ниже слојеве.

На пример, у уређаје који раде на првом, физичком слоју (тзв. *L1* уређаји) спадају појачивачи дигиталних сигнала (разводници и рипитери), у уређаје који раде на слоју везе (тзв. *L2* уређаји) комутатори, а у уређаје који раде на трећем, мрежном слоју (тзв. *L3* уређаји) рутери. Размевање протокола транспортног слоја је обично директно повезано са њиховом употребом од стране протокола на слоју апликације а у уређаје виших слојева углавном спадају крајњи, кориснички уређаји или специјализовани мрежни уређаји.

## 1.1.1. Нивои рада мрежних уређаја

У пракси се често користе ознаке *L1*, *L2* и *L3* за означавање нивоа рада мрежних уређаја. Ове ознаке се односе на највиши слој (ниво) *OSI* референтног модела који уређај разуме. У складу са тим, на првом, физичком нивоу (*L1*) функционишу рипитери и разводници. На слоју везе података (*L2*) функционишу мрежни мост, комутатор и модем. На мрежном слоју (*L3*) функционишу сви виши уређаји - рутер, рачунар, заштитни зид...



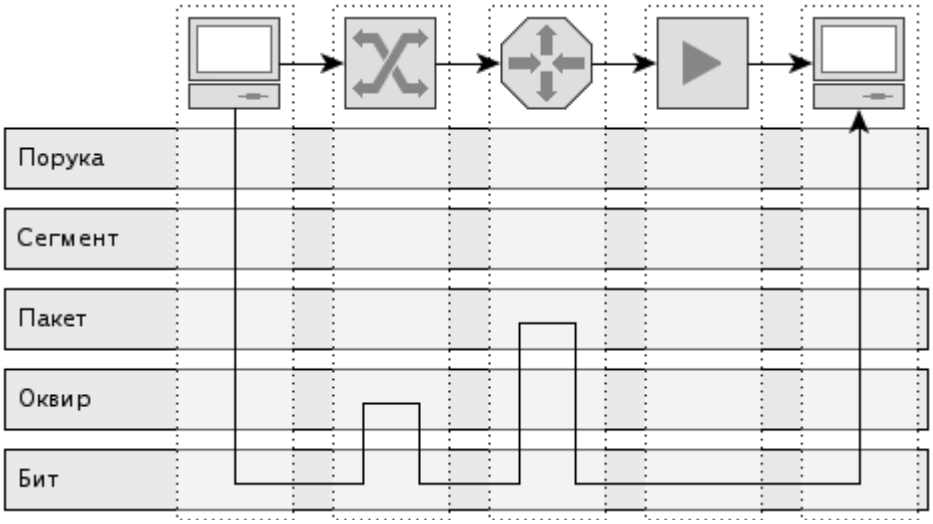

*Слика 1. Нивои рада мрежних уређаја.*

Треба имати у виду да се мрежни слој - *L3* - често узима за највиши слој при одређивању нивоа на коме ради уређај, а да у пракси велики број *L3* уређаја има подршку и за протоколе виших слојева. На пример, већина рутера има могућност коришћења протокола за динамичко рутирање који функционишу на апликативном, седмом нивоу. Рачунари такође подразумевано користе сервисе на апликативном слоју. Постоје заштитни зидови који заиста разумеју само протоколе мрежног нивоа, али и заштитни зидови који могу да анализирају и протоколе транспортног и апликативног нивоа (прокси сервери).

Такође, постоје и комутатори који подржавају одређене функције мрежног слоја. У њих првенствено спадају основне функције рутирања за потребе повезивања различитих виртуалних локалних мрежа. Такви комутатори се називају *L3* комутаторима (енгл. *L3 switch*).

## 1.1.2. Пакетски пренос и инкапсулација

Подаци се у рачунарским мрежама реализованим по Интернет моделу размењују кроз пакетски пренос. Пакетски пренос података подразумева поделу веће количине података у мање јединице - пакете - које се засебно преносе у оквиру комуникације. Пакети представљају основне јединице података протокола а у зависности од комуникационог слоја на коме се протокол налази називају се оквирима (слој везе), пакетима (мрежни слој) или сегментима (транспортни слој).

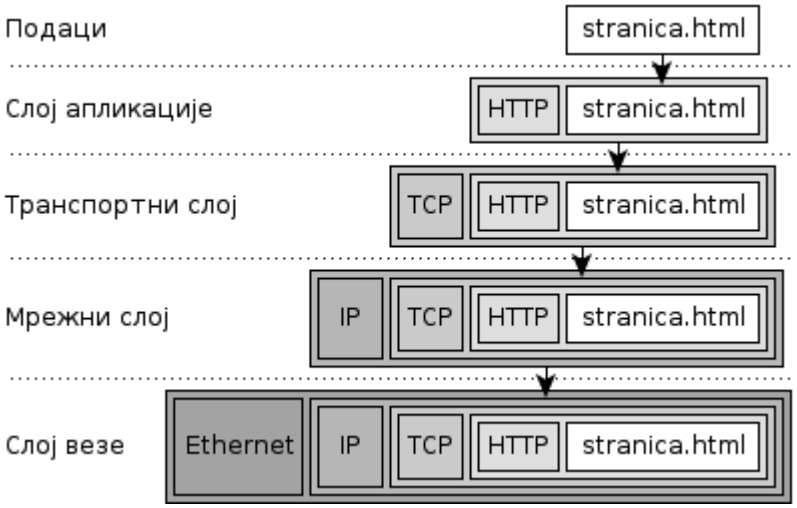



*Слика 1. Могуће су вишеструке инкапсулације/декапсулације током преноса*

Хоризонтална пакетска комуникација подразумева превођење битова пакета у одговарајуће сигнале коришћене технологије за повезивање и на такав начин њихов пренос до коначног одредишта или првог следећег посредника у комуникацији. Уколико је у питању комуникација путем посредника, пакети се могу једноставно прослеђивати следећем посреднику/одредишту или преводити у пакете различите телекомуникационе технологије.

Када је у питању вертикална комуникација, односно комуникација између различитих протокола на различитим комуникационим слојевима, неодвојив део пакетског преноса је и процес **инкапсулације**. Процес инкапсулације подразумева прављење пакета протокола на одређеном слоју чији садржај чини пакет добијен од протокола вишег нивоа. Овај процес се понавља за сваки комуникациони слој чији се протоколи користе.

Процес инкапсулације пакета се извршава на страни пошиљаоца, а њему инверзан процес је процес **декапсулације.** Декапсулација се извршава на страни примаоца а њоме се из пакета протокола нижих слојева издвајају пакети протокола виших слојева и достављају тим протоколима на даљу обраду.




*Слика 2. Инкапсулација у пакете протокола различитих слојева*

Комплетан процес инкапсулације одвија се на страни пошиљаоца а комплетан процес декапсулације на страни примаоца података. Међутим, оба процеса се делимично могу извршавати и на посредницима у комуникацији, у зависности од тога на којим нивоима ти посредници функционишу. На пример, рутер ће из примљених оквира издвојити пакете мрежног слоја (декапсулација) да би на основу података у њиховом заглављу донео одлуку о прослеђивању. Након

доношења те одлуке, рутер ће пакет мрежног слоја упаковати у одговарајући оквир (инкапсулација) за преношење до следећег посредника или коначног одредишта.

## 1.1.3. Технологије за приступ Интернет мрежи

Данас постоји велики број различитих технологија за повезивање рачунара у рачунарске мреже, односно технологија на физичком слоју и слоју везе података *OSI* референтног модела. Ове технологије се могу груписати по различитим критеријумима, а основне поделе се праве по типу медија који се користи за пренос података и по географској површини коју покривају.




*Слика 1. Популарне технологије за приступ Интернету и WAN повезивање*

Основна подела рачунарских телекомуникационих технологија по типу медија који користе је на бежичне и жичне. Унутар ове поделе даља подела се врши по носиоцу сигнала: електрични импулси, светлосни импулси, радио-таласи.

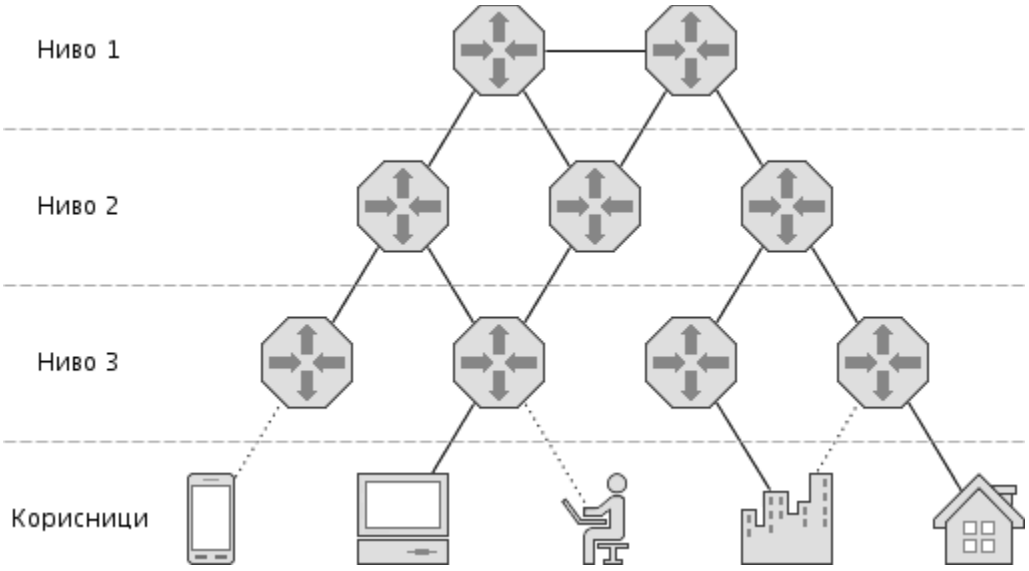
Следећа значајна карактеристика рачунарских телекомуникационих технологија односи се на величину простора на коме се путем њих може извршити повезивање рачунарских система. Нивое који се јављају у оквиру ове поделе

чине:

- личне рачунарске мреже (енгл. *Personal Area Network, PAN*)
- локалне рачунарске мреже (енгл. *Local Area Network, LAN*)
- мреже на подручју града (енгл. *Metropolitan Area Network, MAN*)
- мреже на ширем подручју (енгл. *Wide Area Network, WAN*)
- глобална мрежа, Интернет

Коришћењем ових технологија формирају се сталне или привремене рачунарске мреже које пружају телекомуникациону функционалност на другом слоју *OSI* модела. Интернет, као глобална мрежа, подразумева повезивање рачунарских мрежа наведених величина на мрежном слоју *OSI* модела.

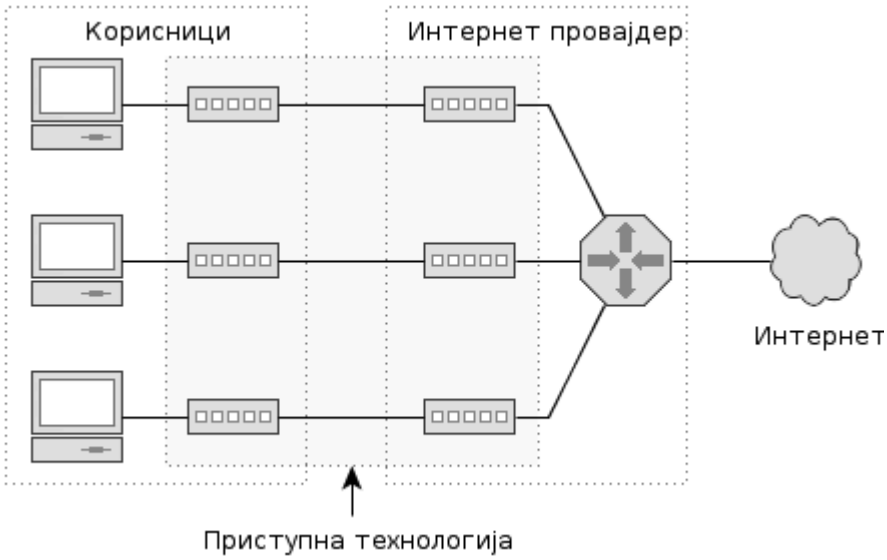
Основу рада Интернет мреже, као и њене приступне тачке, чине Интернет провајдери. Интернет провајдери су организације које успостављају и одржавају примарне телекомуникационе канале Интернет мреже, а уједно, у зависности од нивоа свог функционисања, омогућавају и повезивање крајњих корисника и њихових мрежа на Интернет мрежу.




*Слика 2. Интернет провајдери и приступ корисника Интернету*

Рачунарске мреже, првенствено оне на већим географским просторима, често се успостављају и путем коришћења већ постојећих телекомуникационих

инфраструктура, односно инфраструктура које нису развијане са идејом да се користе за рачунарске телекомуникације. Ове инфраструктуре се данас најчешће користе за повезивање крајњих корисника и мрежа са Интернет провајдерима. На пример, за остваривање везе са Интернет провајдерима данас се често користи инфраструктура јавне телефонске мреже (кроз модемско повезивање, *ISDN* и *ADSL* сервисе) или кабловске телевизије.



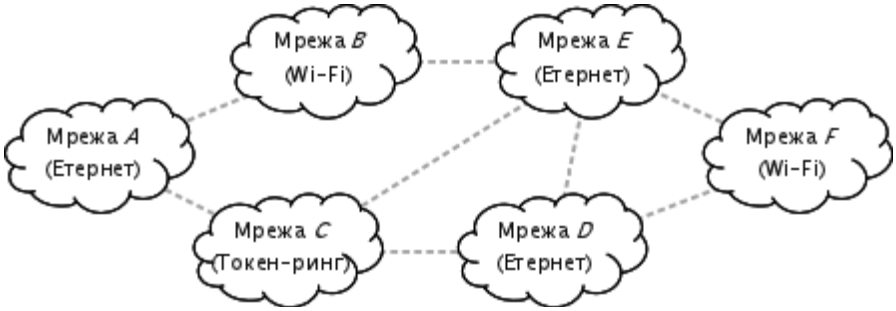

*Слика 3. Шема приступа Интернету путем Интернет провајдера*

Још једно важно питање приликом повезивања на Интернет мрежу путем Интернет провајдера је и питање границе између Интернет провајдера и клијента, односно тачке у којој престаје надлежност провајдера а почиње надлежност клијента. Та тачка се назива тачком разграничења (енгл. *demarcation point*) и у Европи је углавном чини телекомуникациони уређај на страни клијента. На пример, последња тачка за коју је одговоран провајдер приступа Интернету путем *ADSL* технологије, углавном је сам *ADSL* адаптер који се физички налази у просторијама корисника али се налази у власништву провајдера. Питање тачке разграничења је поред функционалних значајно и због безбедносних разлога.

## 1.1.4. Међумрежно повезивање

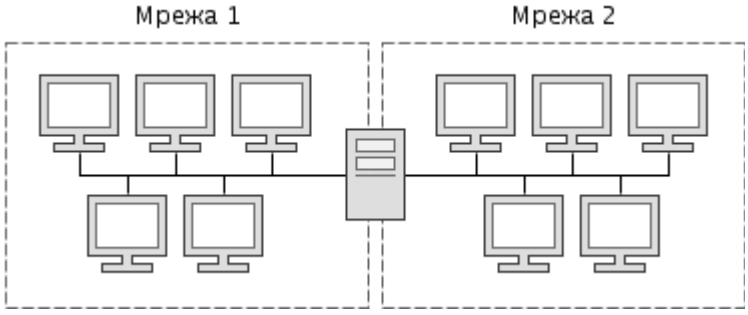
Технологије на физичком слоју и слоју везе података *OSI* комуникационог модела, односно на слоју приступа мрежи *TCP/IP* модела, омогућавају

успостављање везе и пренос података између чланова једне рачунарске мреже. Ове технологије поседују сопствене начине адресовања чланова у циљу достављања послатих података предвиђеном примаоцу. Међутим, систем адресовања на том нивоу није у стању да омогући комуникацију између чланова различитих рачунарских мрежа. Задатак мрежног слоја *OSI* комуникационог модела, односно међумрежног слоја *TCP/IP* модела, јесте да кроз одговарајући систем адресовања омогући проналажење оптималних путања између учесника у комуникацији који се налазе у различитим и међусобно директно или посредно повезаним рачунарским мрежама.




*Слика 1. Међусобно повезане различите рачунарске мреже*

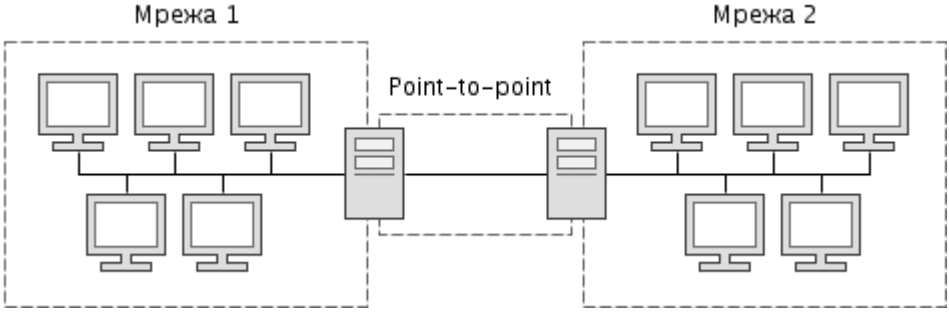
Међумрежно повезивање се заснива на постојању **пролаза** (енгл. *gateway*) између две или више мрежа, односно на **вишеструком удомљавању** (енгл. *multihoming*) чланова мреже способних да усмеравају мрежни саобраћај. Вишеструко удомљени чланови поседују онолико мрежних интерфејса са колико су мрежа повезани. Ови интерфејси могу бити међусобно различити, у зависности од примењене технологије у мрежи са којом су повезани.




*Слика 2. Вишеструко удомљавање чланова мреже*

Две географски суседне рачунарске мреже је лако повезати кроз вишеструко удомљавање њиховог заједничког члана. Међутим, у ситуацијама када се две

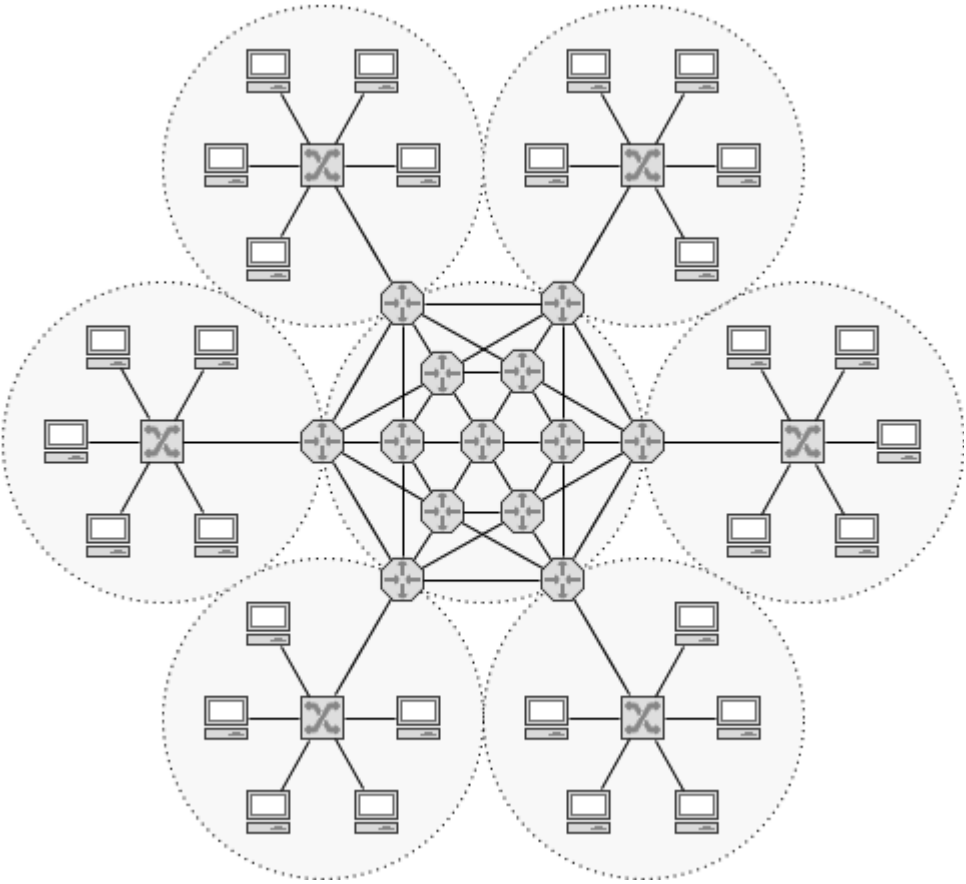
мреже које треба повезати налазе на великој географској удаљености, технологије које омогућавају рад локалних рачунарских мрежа обично нису у стању да то омогуће. Најчешће решење тада представља уметање нове међу-мреже засноване на технологијама које омогућавају повезивање на већим удаљеностима (изнајмљене линије, фрејм-релеј и сл.) а обично подразумевају тачка-тачка (енгл. *point-to-point*) тип комуникације.




*Слика 3. Повезивање мрежа уметањем међу-мреже*

Треба имати у виду да се самим вишеструким удомљавањем чланова у различитим мрежама не добија могућност међусобне комуникације њихових чланова, већ да је за то потребно укључити наменску функционалност. Основна функционалност мрежних пролаза за омогућавање комуникације између чланова различитих мрежа јесте **рутирање**, односно усмеравање саобраћаја. Из тог разлога се уређаји који имају улогу мрежног пролаза који усмерава саобраћај називају рутерима.

У неким ситуацијама је једноставно усмеравање саобраћаја недовољно па се уместо њега користи **превођење мрежних адреса** (енгл. *Network Address Translation, NAT*) заглављима пакета који путују кроз мрежни пролаз. Превођење мрежних адреса је сложенији процес код кога се у заглављима пакета Интернет протокола изворишна или одредишна адреса замењује другом адресом. На пример, пакети који долазе са Интернета не могу бити адресовани на приватне адресе, односно на адресе резервисане за коришћење у приватним мрежама (опсези адреса 10.0.0.0/8, 172.16.0.0/12 и 192.168.0.0/16). Са друге стране, треба имати у виду да је већина приватних корисника на Интернет повезана баш коришћењем приватних мрежа иза рутера добијеног од провајдера. У том случају се саобраћај који долази са Интернета адресује на јавну адресу рутера. Након добијања таквих пакета рутер у својој табели тражи члана локалне мреже коме је тај пакет намењен. Уколико га пронађе, одредишну адресу замењује адресом члана у приватној мрежи и наставља слање.



*Слика 4. Интернет мрежа повезује приватне рачунарске мреже*

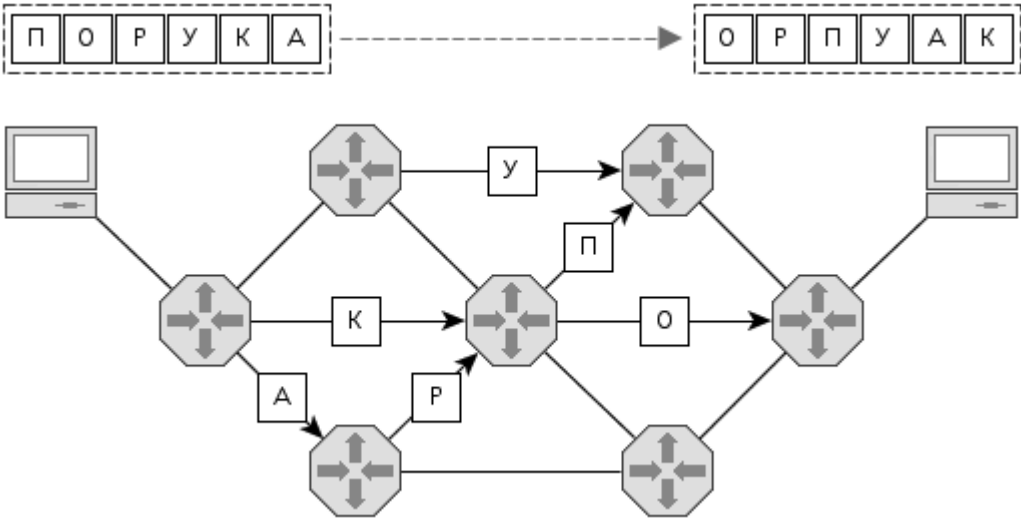
Коначно, целокупна Интернет мрежа се може посматрати као универзална, глобална међу-мрежа (енгл. *inter-network, inter-net*), односно мрежа која повезује приватне рачунарске мреже широм света.

## 1.1.5. Транспортовање података

Нижи комуникациони слојеви имају за задатак да омогуће комуникацију између чланова једне рачунарске мреже, као и између чланова више рачунарских мрежа које су међусобно повезане. Технологије и протоколи на овим слојевима омогућавају усмеравање и пренос послатих података до одредишта. Њихов задатак, међутим, није и брига о интегритету података који се преносе, као ни гарантована испорука у неизмењеном облику. Ове функционалности спадају у

домен задатака протокола на транспортном слоју.

Оно што се на нивоу међумрежног повезивања или самог приступа мрежи може сматрати успешном комуникацијом не мора задовољавати комуникационе потребе корисничких сервиса. Са аспекта корисничког мрежног сервиса могу се јавити различити проблеми у комуникацији: оштећење послатих података, немогућност испоруке делова садржаја, испорука садржаја у измењеном редоследу његових сегмената, затрпавање одредишта превеликом количином података и слично. Решавање наведених проблема, односно брига о успешном преносу података корисничких сервиса, представља основни задатак протокола транспортног слоја а уједно и разлог његовог постојања.




*Слика 1. Испорука пакета у измењеном редоследу*

У пракси постоје два најчешће коришћена протокола на транспортном слоју, у зависности од типа осетљивости података који се преносе. Први тип података чине **подаци осетљиви на измене током преноса**. У ту групу спадају подаци финансијских трансакција, преноса фајлова, електронске поште и других сервиса. За потребе преноса овог типа података најчешће се користи **протокол за контролу преноса** (енгл. *Transmission Control Protocol, TCP*). Други тип података чине **подаци осетљиви на кашњење током преноса**. То су углавном подаци везани за аудио/видео конференције путем рачунарских мрежа, видео надзор и слично. За потребе преноса овог типа података најчешће се користи **протокол корисничких датаграма** (енгл. *User Datagram Protocol*).

## 1.1.6. Кориснички сервиси

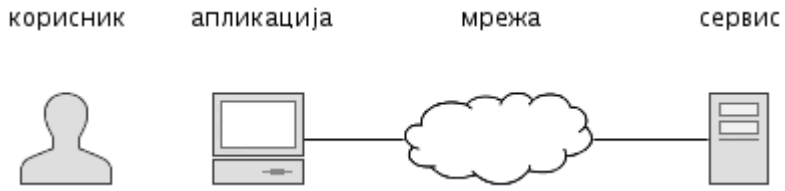
Целокупан комуникациони модел Интернет мреже је развијен са циљем да омогући функционисање корисничких сервиса. Задатак слоја за приступ мрежи је да омогући коришћење различитих телекомуникационих технологија на инфраструктурном нивоу, задатак међумрежног слоја да кроз одговарајући систем адресовања омогући проналажење удаљеног одредишта, задатак транспортног слоја да брине о испоруци података, а све у циљу несметаног коришћења мреже од стране корисника.

Са аспекта корисничког сервиса, односно компоненте на слоју апликације, она (компонента) се кроз захтев транспортном слоју (оперативном систему) за отварање сокета, директно обраћа компоненти на слоју апликације друге стране. Нижи слојеви омогућавају пренос података између две стране а протокол апликације брине о остваривању сервиса за корисника.




*Слика 1. Апликације представљају интерфејс између корисника и мреже*

Кориснички сервиси се по *OSI* и *TCP/IP* комуникационим моделима реализују на највишем слоју, слоју апликације. Овај слој је најближи кориснику и у виду апликације чини интерфејс корисника ка рачунарској мрежи. Апликације представљају облик реализације софтвера намењен за непосредно коришћење од стране корисника. У складу са тим апликације поседују одговарајући (графички, алфа-нумерички или неки други) кориснички интерфејс.




*Слика 2. Клијентске апликације често размењују податке са сервисима*

Сервиси, насупрот апликацијама, представљају реализацију код које се софтверски процеси извршавају „у позадини“ рачунарског система, односно не

поседују кориснички интерфејс. Корисник се о овим процесима може информисати само посредно (на пример, коришћењем наредбе *top* на Линукс оперативном систему) а њихова улога јесте подршка осталим софтверским процесима на локалном или удаљеним рачунарима. На пример, Веб сервис подразумева постојање корисничке апликације (Веб браузер) и серверског софтвера (Веб сервер).

## 1.2. Адресовање на Интернету

Број људи на свету који имају приступ Интернет мрежи је током 2011. године премашио две милијарде[2]. Број уређаја који приступају Интернет мрежи је током 2010. године премашио 5 милијарди а очекује се да тај број пређе 22 милијарде до 2020. године[3]. За потребе комуникације овог броја чланова потребно је имати систем адресовања који са једне стране омогућава једнозначно одређивање крајњих учесника у комуникацији, и у стању је да то чини за што мање времена и са што мање ресурса, а који је, са друге стране, довољно једноставан и аутоматизован да може бити коришћен од стране људи.

Архитектура Интернет мреже и њене технологије подразумевају адресовање чланова на више нивоа. Носеће телекомуникационе технологије дефинишу сопствене системе за адресовање који се користе да се одреди физички члан локалне мреже који треба да обради емитоване телекомуникационе сигнале. Интернет мрежа дефинише сопствене протоколе који уређајима прикљученим на више мрежа - међумрежним посредницима - омогућавају да одреде коме треба проследити примљене податке, односно ко је следећи чвор у комуникацији између крајњих чланова.

Наведени системи адресовања нису превише погодни за директно коришћење од стране људи, те су на највишем нивоу развијени симболички системи адресовања који поједностављују коришћење рачунарских мрежа. Сви ови системи за адресовање функционишу у тесној спрези са коначним, раније поменутим циљем да омогуће једнозначно одређивање чланова Интернет мреже као учесника у комуникацији.

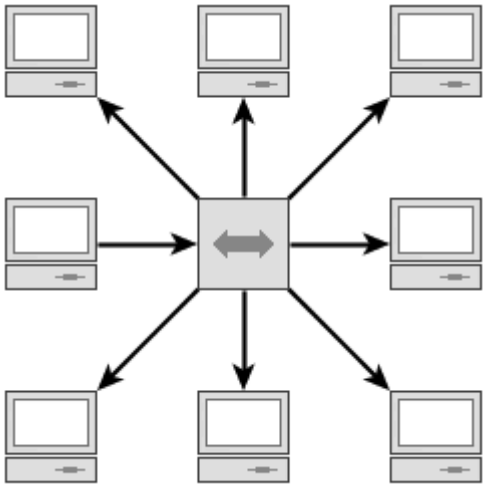
### 1.2.1. Адресовање на нивоу локалне мреже

Адресовање чланова на нивоу локалне мреже представља систем за одређивање који уређај у мрежи треба да прими и обради послате сигнале или

2 Internet World Stats http://www.internetworldstats.com

3 IMS Research, http://www.imsresearch.com

податке. Из тог разлога су системи адресовања који се користе у локалним мрежама уско везани за технологије које у њима користе.

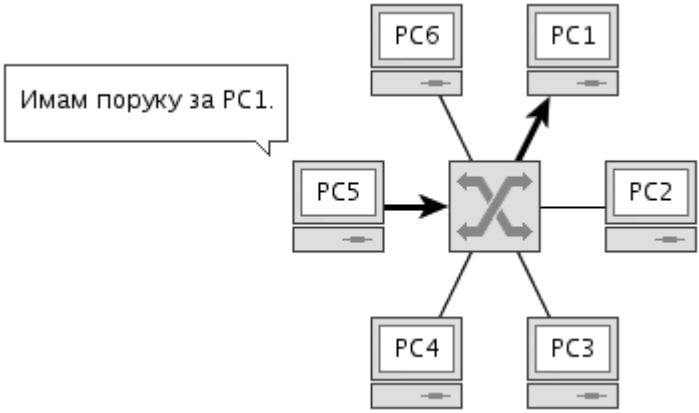


*Слика 1. Разводник у Етернет мрежи прослеђује послате битове свим члановима*

Адресовање чланова на нивоу локалне мреже је најједноставније (али не и најефикасније) решено у раним верзијама Етернет технологије. Комуникација у мрежама заснованим на тој технологији подразумевала је пренос сигнала до свих чланова (било коришћењем линијске магистрале, било коришћењем разводника), са препоруком да чланови који уоче да порука није адресована на њих поруку одмах одбаце, односно да је даље не интерпретирају.

Иако врло једноставан за реализацију, описани систем за адресовање је у мрежама у којима је примењиван доводио до два проблема - поверљивост у таквим мрежама није постојала, као ни могућност паралелне комуникације различитих парова. Из тог разлога су разводници касније замењени комутаторима - уређајима који имају могућност да послате сигнале усмеравају само до примаоца на кога је порука адресована.

Данас је чест случај, посебно на Интернет мрежи, да поруке до коначног примаоца путују у кроз десетак различитих мрежа, заснованих на различитим технологијама. Свака од тих мрежа користи сопствени систем адресовања, односно систем адресовања технологије на којој је заснована. Адресе чланова у локалној мрежи најчешће морају бити јединствене а домен важења адресе је локална мрежа. То значи да се исте адресе могу понављати у различитим мрежама.




*Слика 1. Комутатор у Етернет мрежи усмерава оквире до примаоца*

Адресе које се користе у локалним мрежама подразумевано никада не излазе ван њих. То је последица одбацивања оквира који се користио за комуникацију у приватној мрежи и инкапсулирања мрежног пакета у нови оквир, са новим адресама.

## 1.2.2. Међумрежно адресовање

Додатно у односу на системе адресовања локалних рачунарских мрежа, који имају задатак да одреде уређај у локалној мрежи који треба да прими и обради послате податке, системи за међумрежно адресовање морају да одреде и мрежу у којој се прималац налази. На Интернет мрежи систем за међумрежно адресовање мора да омогући јединствено одређивање и комуникацију са неколико милијарди чланова.




*Слика 1. Протоколи ARP и RARP се користе за превођење IP адреса у Етернет и обрнуто*

Примарни систем који се користи за међумрежно адресовање на Интернет

мрежи је Интернет протокол (енгл. *Internet Protocol, IP*). Тренутно су у употреби две верзије овог протокола, верзија 4 и верзија 6, које користе адресе дужине 32 и 128 битова. Рутери, као уређаји чији је задатак међумрежно повезивање, користе ове адресе за одређивање где се налази мрежа којој припада прималац, односно ко је следећи чвор коме треба послати податке да би се они доставили примаоцу.

Оно што треба имати у виду је да се системи за међумрежно адресовање користе само на крајњим члановима и посредујућим рутерима, а да се за пренос података и даље користе системи локалног адресовања за сваку од мрежа кроз коју путују подаци. На пример, уколико подаци путују кроз Етернет мрежу, коришћењем *ARP* протокола ће се IP адреса превести у физичку адресу која ће се користити за даљу комуникацију.

## 1.2.3. Систем доменских имена

Систем доменских имена (енгл. *Domain Name System, DNS*) је систем који чува информације везане за имена домена у виду дистрибуиране базе података, а реализован је као клијент-сервер сервис. Најважнија функционалност *DNS*-а је превођење доменских имена у адресе Интернет протокола и обрнуто.

Већина осталих мрежних сервиса (Веб, електронска пошта, пренос фајлова итд.) користи или има могућност да користи *DNS* сервис. На пример, једна од функционалности *DNS*-а је и обезбеђивање информације о томе који сервери су задужени за размену електронске поште за одређени домен. Без ове функционалности *DNS*-а сервис за размену електронске поште не би могао да функционише.

### 1.2.3.1. Историјат проблема и решења

Систем доменских имена развијен је као одговор на пораст величине рачунарских мрежа, пораст броја рачунарских мрежа (и појаве Интернета) и на потребу за једноставнијим адресовањем рачунара на мрежи. Под једноставнијим адресовањем се мисли на прилагођавање мрежног адресовања особини људи да лакше памте симболичка имена од бројева (на пример, лакше је запамтити *www.dir.singidunum.ac.rs* од 212.62.45.222). Проблем је у почетку био решен путем *hosts* фајлова на сваком од рачунара на мрежи. Међутим, порастом броја рачунара у рачунарским мрежама недостаци оваквог решења су постајали све озбиљнији проблем:

Узмимо за пример рачунарску мрежу од *N* чланова. *N* рачунара чува

информацију о *N* чланова те мреже у локалним *hosts* фајловима:

```
192.168.1.1 рачунар1.локална-мрежа
192.168.1.2 рачунар2.локална-мрежа
192.168.1.N рачунарN.локална-мрежа
```

Проблем 1: додавањем новог рачунара у мрежу потребно је на *N* рачунара додати нови запис у *hosts* фајл и на новом рачунару унети комплетан *hosts* фајл.

Проблем 2: изменом постојећег рачунара у мрежи потребно је на *N* рачунара изменити постојећи запис у *hosts* фајлу.

Проблем 3: уклањањем постојећег рачунара из мреже потребно је на *N*-1 рачунара додати или уклонити запис из *hosts* фајла.

Из наведеног се јасно види да код малих мрежа *hosts* фајлови могу бити једноставније решење од *DNS*-а јер нема потребе за постављањем *DNS* сервера. Међутим, код великих мрежа администрација се знатно отежава јер се при свакој измени мреже она односи на све рачунаре у мрежи. Први корак ка решавању наведених проблема био је дистрибуирани *hosts* фајл (један *hosts* фајл у мрежи коме могу да приступају сви чланови мреже) а проблем је у потпуности решен 1983. године када је Пол Мокапетрис изумео систем доменских имена.

```
#
# hosts This file describes a number of hostnametoaddress
# mappings for the TCP/IP subsystem. It is mostly
# used at boot time, when no name servers are running.
# On small systems, this file can be used instead of a
# "named" name server. Just add the names, addresses
# and any aliases to this file...
#
# By the way, Arnt Gulbrandsen <agulbra@nvg.unit.no> says that 127.0.0.1
# should NEVER be named with the name of the machine.
# It causes problems
# for some (stupid) programs, irc and reputedly talk. :^)
#
# For loopbacking.
```

```
127.0.0.1     localhost
192.168.1.1   tool.local tool
# End of hosts.
```

*Листинг 5.5.2.1-1. Пример hosts фајла на Линукс оперативном систему*

*Hosts* фајлови се могу користити у комбинацији са *DNS*-ом. У том случају они имају приоритет над *DNS*-ом тј. при разрешавању неког имена прво се проверава садржај hosts фајла а тек уколико он не садржи информацију о траженом имену упит се шаље *DNS* серверу. Овакав редослед у разрешавању имена има своје добре стране. На пример, могуће је "заобићи" *DNS* тј. могуће је заменити адресу неког рачунара при локалном разрешавању имена - уносом записа:

```
0.0.0.0       ad.doubleclick.net
```

у *hosts* фајл локалног рачунара он неће бити у могућности да приступи стварној адреси *ad.doubleclick.net*. Последица овога је да при сурфовању Интернетом ниједан садржај са поменуте адресе неће бити доступан. Међутим, како са поменуте адресе најчешће долазе само рекламе, оне неће бити досупне тако да ће се то одразити већом брзином учитавања осталих садржаја у којима се оне приказују. Са друге стране, заобилажење *DNS*-а могуће је као последица инфицирања система малициозним софтвером. На пример:

1. Нападач креира Веб страницу на сопственој адреси Интернет протокола - X.X.X.X - која је различита од адресе на којој се налази домен *www.google.com* - Y.Y.Y.Y.
2. Веб страница на адреси X.X.X.X је таква да визуелно у потпуности одговара оригиналној страници на *www.google.com* али су логика и база података претраживача који стоји иза те странице потпуно другачије од оних на стварној адреси претраживача *www.google.com*.
3. Лажни *www.google.com* на адреси Y.Y.Y.Y намењен је за промоцију клијената који нападачу за узврат дају новчану надокнаду.
4. Нападач затим креира малициозни софтвер који се шири путем Интернета и у hosts фајл заражених рачунара уноси запис: X.X.X.X *www.google.com*.

На овај начин, сваки од заражених рачунара при захтеву за страницом *www.google.com* приступа лажној адреси X.X.X.X уместо Y.Y.Y.Y а корисници

добијају погрешне информације у корист нападача.

### 1.2.3.2. Теорија рада система доменских имена

Простор доменских имена је стабло за чији сваки чвор постоји запис у *DNS*-у надлежном за ту зону. У надлежном *DNS* серверу (енгл. *authoritative DNS nameserver*) могуће је за одређену зону декларисати подзоне путем декларисања одговарајућих *DNS* под-сервера.

За разумевање система доменских имена и начина његовог функционисања потребно је разумети саму структуру имена домена (енгл. *domain name*). Назив домена се састоји од два или више делова раздвојених тачкама. Узмимо за пример домен *dir.singidunum.ac.rs*:

Прва ознака са десне стране представља домен највишег нивоа (енгл. *Top Level Domain, TLD*), у овом случају *rs.* Свака наредна ознака гледано са десне стране - *ac*, *singidunum* и *dir* - представља поддомен. Максималан број поддомена је 127 а сваки од чланова може имати максималну дужину од 63 карактера, с тим да целокупна дужина назива (укључујући све поддомене и тачке којим су раздвојени) не сме прећи 255 карактера.

Домен може имати једно или више дефинисаних имена домаћина (енгл. hostname) којима су придружене адресе Интернет протокола. У наведеном случају, домен је *dir.singidunum.ac.rs* а име домаћина би могло да буде *www.dir.singidunum.ac.rs* са одговајућом адресом Интернет протокола 212.62.45.222.

Сервис система доменских имена чине хијерархијски повезани сервери. За сваки од домена мора да буде декларисан један или више надлежних *DNS* сервера који су задужени за чување и давање информација о њему. Један *DNS* сервер може бити задужен и за већи број потпуно независних домена. У корену стабла постоје специјални *DNS* сервери који се зову корени сервери (енгл. *root servers*) и они су задужени за домене највишег нивоа - домене на самом корену стабла. Без поменутих корених сервера рад Интернета не би био могућ јер они чине основу сваког доменског именовања на њему. Тренутно постоји 13 корених сервера и њихова имена су *[A-M]. root-servers.net*.

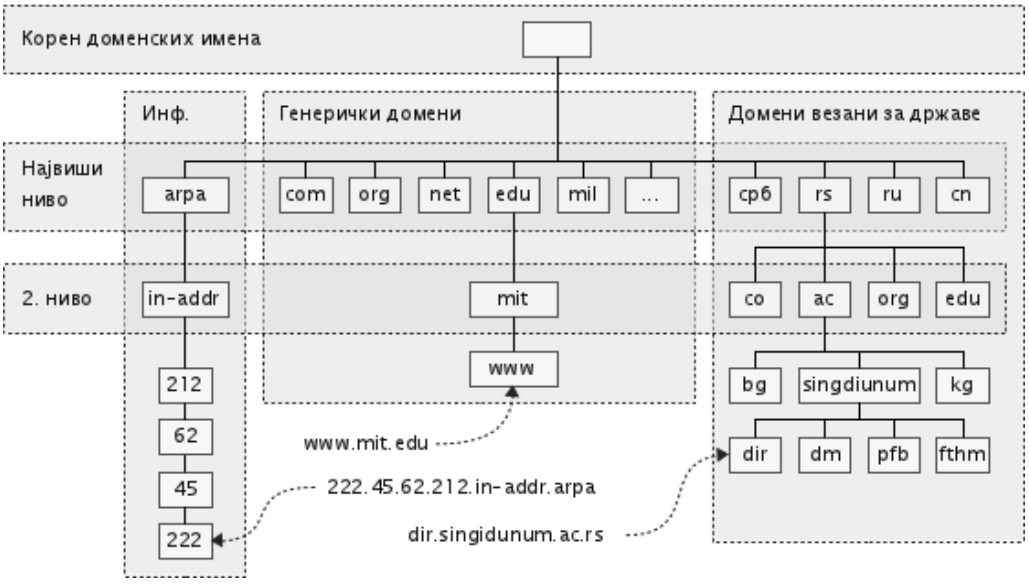
Домен највишег нивоа је прва ознака с десна у сваком имену домена - у домену *dir.singidunum.ac.rs* домен највишег нивоа је *rs.* Постоје три категорије домена највишег нивоа:

1. домени највишег нивоа везани за државе - домени дужине два слова везани за земљу или одређени географски простор: *rs* - Република

Србија, *ru* - Руска Федерација, *cn* - Народна Република Кина и тако даље;

2. генерички домени највишег нивоа - домени који се користе за одређену класу организација: *com* - комерцијални системи, *org* - непрофитне организације, *edu* - образовне установе и тако даље;
3. инфраструктурни домени највишег нивоа - једини у овој групи је *arpa* домен.

За нашу државу, као и за све државе чије писмо садржи и друге знакове сем енглеског алфабета, значајна је од скора доступна могућност коришћења и међународних знакова у називу домена. У складу са њом, дефинисано је и више домена највишег домена за наведени тип држава. Када су у питању ћирилични домени највишег нивоа, Република Србија је, првенствено захваљујући професионалном и ефикасном раду РНИДС-а, обезбедила срб домен највишег нивоа, одмах након Руске федерације. Почетак јавне употребе овог домена десио се крајем 2011. године.




*Слика 1. Хијерархијска организација система доменских имена*

Клијентска компонента *DNS* система назива се разрешивач (енгл. *resolver*). Ова компонента се обраћа *DNS* серверу да би од њега добила адресу Интернет протокола за задато име домена. Разрешивач је системска компонента која се користи посредно, односно путем програма којима је ова услуга потребна. Разрешивачи доменских имена користе системске мрежне параметре који

најчешће садрже логичку адресу једног или два *DNS* сервера.

Пример коришћења *DNS* услуге:

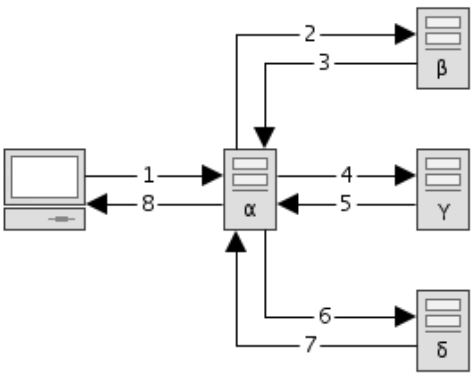
1. Апликација (на пример, Веб браузер) добија униформни локатор ресурса *http://www.dir.singidunum.ac.rs/index.php* од стране корисника и рашчлањује га на протокол (*http*), име домаћина (*www.dir.singidunum.ac.rs*) и локалну адресу ресурса (*/index.php*).
2. Апликација се обраћа разрешивачу доменских имена у циљу добијања адреса Интерент протокола за тражено име домаћина.
3. Разрешивач доменских имена обраћа се *DNS* серверу из мрежне конфигурације рачунара питањем: "Да ли знаш која је адреса Интернет протокола доменског имена *www.dir.singidunum.ac.rs*?"
4. Уколико *DNS* коме се разрешивач обратио није надлежан за домен у коме се тражени домаћин налази (*dir.singidunum.ac.rs*) он се обраћа једном од корених *DNS* сервера питањем: "Који је DNS сервер надлежан за rs домен?"
5. Корени сервер враћа одговор: "147.91.8.6".
6. DNS сервер се обраћа серверу 147.91.8.6 питањем: "Који је *DNS* сервер надлежан за *ac.rs* домен?".
7. *DNS* сервер на адреси 147.91.8.6 враћа одговор: "147.91.8.21".
8. *DNS* сервер се обраћа серверу на адреси 147.91.8.21 питањем: "Који је *DNS* сервер надлежан за домен *singidunum.ac.rs*?"
9. Сервер на адреси 147.91.8.21 враћа одговор: "212.62.48.42".
10. *DNS* сервер се обраћа серверу на адреси 212.62.48.42 питањем: "Који је *DNS* сервер надлежан за домен *dir.singidunum.ac.rs*?".
11. Сервер на адреси 212.62.48.42 враћа одговор: "212.62.45.222".
12. *DNS* сервер се обраћа серверу на адреси 212.62.45.222 питањем: "Која је адреса домаћина *www.dir.singidunum.ac.rs*?".
13. Сервер на адреси 212.62.48.222 враћа одговор: "212.62.45.222"
14. *DNS* сервер враћа одговор клијенту чији му се разрешивач обратио: "адреса Интернет протокола за домаћина *www.dir.singidunum.ac.rs* је 212.62.45.222".

На овај начин апликација на клијентском рачунару добија адресу Интернет

протокола Веб сервера и путем те адресе прослеђује захтев за Веб страницом */index.php*. Овај пример објашњава рекурзију у раду *DNS*-а. Треба имати у виду да један *DNS* сервер може чувати информације о више различитих домена као и да више *DNS* сервера могу пружати информацију о једном домену.

### 1.2.3.3. Кеширање код система доменских имена

У примеру рекурзивног *DNS* разрешавања, датог у делу „Теорија рада система доменских имена", приказан је скуп корака који је теоретски неопходно проћи да би клијент добио информацију од сервера. У пракси би, међутим, овакав начин рада код сваког *DNS* упита за сваки Интернет домен створио огромно оптерећење свих *DNS* сервера који учествују у разрешавању одређеног домена. Ово се пре свега односи на корене *DNS* сервере и *DNS* сервере којима клијент директно приступа. Да би се избегло поменуто оптерећење уведено је кеширање резултата.



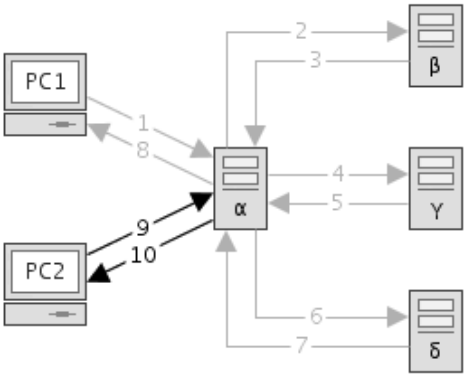

| | | |
|---|---|---|
| 1 | A | singidunum.ac.rs? |
| 2 | A | singidunum.ac.rs? |
| 3 | NS | ac.rs -> 147.91.173.4 (γ) |
| 4 | A | singidunum.ac.rs? |
| 5 | NS | singidunum.ac.rs -> 212.62.48.42 (δ) |
| 6 | A | singidunum.ac.rs? |
| 7 | A | singidunum.ac.rs -> 212.62.45.222 |
| 8 | A | singidunum.ac.rs -> 212.62.45.222 |

*Слика 1. Рекурзивно разрешавање адресе*

Кеширање код *DNS*-а има за циљ да омогући сваком од *DNS* сервера смањење броја упита које он поставља осталим *DNS* серверима при разрешавању упита везаног за домене из зоне за коју он није надлежан. Уколико је кеширање укључено на *DNS* серверу, он у својој интерној бази чува резултате свих успешно обављених разрешавања, тако да, уколико се исти упит понови, сервер не мора да тражи поново све информације од осталих *DNS* сервера већ користи потојећу информацију из базе.

Овакав начин рада штеди процесорске ресурсе и комуникационе канале *DNS* сервера али отвара и ново питање - уколико се информација о домену на за њега надлежном *DNS* серверу измени, како ће се то одразити на клијенте који врше упит за тај домен посредством других сервера који у својој бази имају

забележену претходну информацију?




| | | |
|---|---|---|
| 1 | A | singidunum.ac.rs? |
| 2 | A | singidunum.ac.rs? |
| 3 | NS | ac.rs -> 147.91.173.4 (γ) |
| 4 | A | singidunum.ac.rs? |
| 5 | NS | singidunum.ac.rs -> 212.62.48.42 (δ) |
| 6 | A | singidunum.ac.rs? |
| 7 | A | singidunum.ac.rs -> 212.62.45.222 |
| 8 | A | singidunum.ac.rs -> 212.62.45.222 |
| 9 | A | singidunum.ac.rs? |
| 10 | A | singidunum.ac.rs -> 212.62.45.222 |

*Слика 2. Претходно добијени резултати се чувају за наредне упите*

Одговор на претходно питање лежи у ограничењу периода важења (енгл. *Time To Live, TTL*) информација које је надлежни сервер дао. Овај параметар одређује након ког времена ће посреднички *DNS* сервери обновити информацију у својој бази везану за тај домен. Период важења се изражава у секундама и најчешће је постављен на 86.400 секунди, односно један дан. То у пракси значи да је максимално време (након измене домена на надлежном *DNS* серверу) током кога ће клијенти добијати застарелу информацију од посредничких *DNS* сервера један дан. Остали системски параметри сваке информације о имену домена су:

- *Serial*: серијски број зоне који се увећава при свакој измени података, а служи осталим серверима за утврђивање да ли се информација изменила на главном серверу.
- *Refresh*: број секунди након кога ће *slave* и *secondary* сервери освежити своје податке за зону.
- *Retry*: број секунди након кога ће *slave* и *secondary* сервери поново покушати освежавање података са *master* сервера уколико претходни покушај не успе.
- *Expire*: број секунди након кога ће *slave* и и *secondary* сервери одустати од покушаја да освеже своју базу са *master* сервера уколико претходни покушаји не успеју.

### 1.2.3.4. Типови записа

У основне типове записа у бази DNS сервера спадају:

- A - *Address* - запис који садржи IPv4 адресу домаћина.
- CNAME - *Canonical Name* - симболичка веза (линк) ка другом домаћину.
- MX - *Mail Exchange* - адреса сервера електронске поште за домен.
- NS - *Name Server* - адреса надлежног *DNS* сервера за домен.
- SOA - *Start Of Authority* - дефиниција почетка надлежности за одређени домен.

У наставку је дат пример зонског *DNS* фајла за домен *racunarskemreze.com*:

```
$ORIGIN racunarskemreze.com.
$TTL 300
@ SOA ns1.loopia.se. registry.loopia.se. (
        1372896000
        3H  ; Refresh after three hours
        1H  ; Retry after one hour
        1W  ; Expire after one week
        1D )  ; Minimum one day TTL

@       IN  NS  ns1.loopia.se.
@       IN  NS  ns2.loopia.se.
        IN  MX 10  88.198.75.150
        IN  MX 20  192.168.1.1
@       IN  A  88.198.75.150
*       IN  A  88.198.75.150
www     IN  A  88.198.75.150
```

### 1.2.3.5. Процес регистрације домена

Процес регистрације домена је, у основи, процес уношења записа о домену у базе *DNS* сервера надлежних за наддомен чији ће домен који региструјемо бити поддомен. На пример, за регистровање домена *dir.singidunum.ac.rs* потребно је у базу *DNS* сервера надлежних за домен *singidunum.ac.rs* унети *IP* адресе *DNS* сервера који ће бити надлежни за домен *dir.singidunum.ac.rs*.

Када је у питању регистровање комерцијалних домена, као поддомена Интернет домена највишег нивоа, процес регистрације се обавља преко за то овлашћених регистара. Регистри Интернет домена су комерцијалне или некомерцијалне организације које су овлашћене да клијентима изнајме одређени Интернет домен на временски период од једне или више година.

Приликом подношења захтева за изнајмљивање одређеног домена клијент је дужан да обезбеди два или више *DNS* сервера са јавним *IP* адресама који ће бити надлежни за тај домен. Један *DNS* сервер се може користити као надлежни за више различитих домена. Та услуга (тзв. *DNS hosting*) се може и закупити. Додатно, многи регистри ову услугу нуде бесплатно.

## 1.2.4. Униформни локатори ресурса

Униформни локатор ресурса (енгл. *Uniform Resource Locator, URL*) дефинише начин кодовања информација за лоцирање и приступ ресурсима на Интернету. У питању је Интернет стандард који се због погодности које нуди, осим за ресурсе на Интернету, често користи и у локалним мрежама, као и у софтверским системима који не подразумевају дистрибуирану обраду података. Типови ресурса који се овим начином адресовања лоцирају нису ограничени, а могу бити електронски документи, њихове графичке и мултимедијалне компоненте, софтверске компоненте и стања апликација, и т.д.

За потребе приступа различитим ресурсима дефинисано је више шема за описивање локације. Сама шема представља први део локатора и најчешће се односи на апликативни протокол којим се приступа ресурсу. Остали део локатора чини адреса специфична за ту шему и од ње је одвојена са две тачке. Спецификација шема које се користе на Интернету подразумева постојање две косе црте након сепаратора између шеме и за њу специфичне адресе. Поред овог, локатор ресурса може поседовати и више сепаратора који одређују његове различите делове.




*Слика 1. Структура униформног локатора ресурса HTTP протокола*

Када је у питању Веб, захтевани делови униформног локатора ресурса на њему

су шема (*HTTP* или *HTTPS* протокол) и адреса домаћина (симболичка или логичка). Овако једноставни локатори се углавном користе за иницијални приступ Веб сајтовима. Порт је такође неопходан део униформног локатора ресурса али се он у савременим Веб браузерима не приказује уколико се користе подразумеване вредности за протокол (порт 80 за *HTTP* протокол и порт 443 за *HTTPS* протокол).

Осим захтеваних делова у пракси се често користи и путања, упит и идентификатор фрагмента. Под путањом се подразумева хијерархијска путања до ресурса на самом домаћину а она може бити путања на фајл-систему или се може динамички обрађивати од стране Веб сервера (пример: *mod_rewrite* модул *Apache* Веб сервера). Упит, као део локатора, најчешће се користи код Веб апликација за прослеђивање параметара серверу путем *GET* метода *HTTP* протокола. Подразумевана улога идентификатора фрагмента је скакање на одређени део документа након његовог учитавања, с тим да одређене Веб технологије на страни клијента могу искористити овај параметар и за друге функционалности. Иначе, сам идентификатор фрагмента се не шаље серверу већ искључиво служи за локалну употребу унутар Веб браузера.

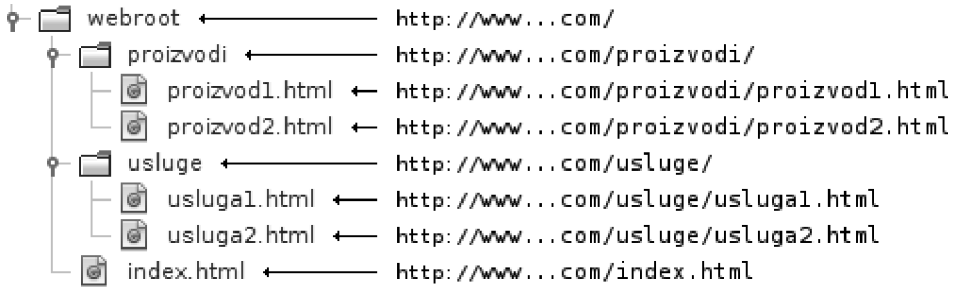



*Слика 2. Дијалог за аутентификацију код HTTP протокола*

Неки ресурси на Интернету и у осталим рачунарским мрежама јавно су доступни, док се за приступ некима захтева потврђивање идентитета корисника коме је приступ одобрен. Различити протоколи нуде различите системе за утврђивање идентитета корисника а сам *HTTP* протокол нуди једноставан систем за ауторизацију коришћењем корисничког имена и лозинке (слика 2.). Ови приступни параметри се такође могу укључити у униформни локатор заштићеног ресурса, од домаћина одвојени знаком „@", а међусобно одвојени

двема тачкама (корисник:лозинка@домаћин). При коришћењу система за аутентификацију *HTTP* протокола треба имати у виду да он самостално не нуди озбиљан ниво заштите, а да укључивање корисничког имена и лозинке у униформни локатор ресурса такође може довести до безбедносних проблема.

У пракси, документи на Вебу често садрже везе ка својим издвојеним садржајима и другим документима у облику релативних путања које прате ову хијерархијску организацију. При учитавању таквих веза Веб браузер израчунава њихове комплетне униформне локаторе на основу њихових релативних путања и униформног локатора документа у оквиру кога су оне дефинисане.



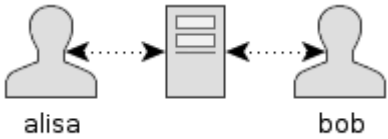

*Слика 3. Однос путања фајл-система и URL-а*

Униформни локатор ресурса дефинише локацију ресурса али не садржи информацију о томе да ли је ресурс тренутно доступан, нити гарантује да му се (у сваком или у било ком тренутку, као и са сваке или са било које локације) може приступити. Локација ресурса представља једну од његових могућих идентификација, док је униформни локатор ресурса један од типова униформног идентификатора ресурса (енгл. *Uniform Resource Identifier, URI*). Други популарни тип униформног идентификатора ресурса јесте униформни назив ресурса (енгл. *Uniform Resource Name, URN*).

## 1.2.5. Адресовање електронске поште

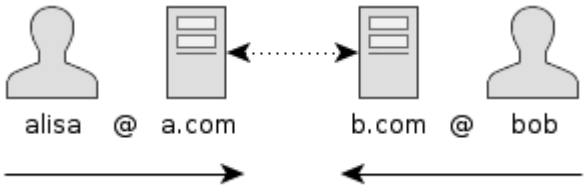
Корени сервиса електронске поште везују се за размену порука између корисника мејнфрејм рачунара који су радили под *CTSS* (*Compatible Time-Sharing System*) оперативним системом (1961. година), развијеним на *MIT* универзитету. Ова размена порука првобитно је обављана путем фајл-система, складиштењем поруке у фајл назван по имену примаоца - на пример, „*to bob*" за корисника са именом Боб. Касније је у ту сврху развијен програм *mail* који је од 1971. године доступан и на *UNIX* оперативном систему. При свему томе треба имати у виду да

су се сви корисници овакве размене порука, дакле и пошиљаоци и примаоци, налазили на истом рачунарском систему.




*Слика 1. Корени електронске поште у размени порука на мејнфрејм системима*

Од 1968. године активно је рађено на омогућавању слања порука између корисника рачунарских система повезаних на *ARPA* мрежу, а то је већ почетком 1972. године омогућено. Примаоца поруке је било потребно одредити у облику „корисник -at систем" а 1973. године је симбол „@" стандардизован[4] као сепаратор у адреси електронске поште.




*Слика 2. Принцип адресовања електронске поште*

Стандард за адресовање порука електронске поште данас[5] дефинише највише 64 *ASCII* карактера за идентификовање корисника (тзв. локални део, односно део пре знака @), с тим да је 2012. године стандард допуњен у циљу омогућавања коришћења међународних карактера[6].

Занимљив приступ у идентификовању примаоца има и Гугл провајдер сервиса електронске поште - тачка у адреси примаоца се не узима у обзир, тако да је власнику адресе:

```
racunarskemrezenet@gmail.com
```

могуће послати поруку и на адресу:

```
r.a.c.u.n.a.r.s.k.e.m.r.e.z.e.n.e.t@gmail.com
```

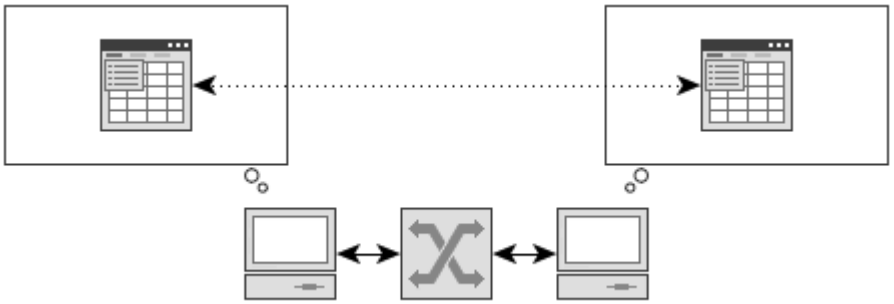
На овај начин семоже класификовати долазна пошта а одређене комбинације се

4 RFC 469 - Network Mail Meeting Summary
5 RFC 5322 - Internet Message Format
6 RFC 6531 - SMTP Extension for Internationalized Email

могу користити на местима код којих се адреса лако може наћи на некој спем листи.

## 1.3. Мрежно програмирање

Различити механизми за међупроцесну комуникацију (дељена меморија, фајлови и сл.) омогућавају да одвојени процеси на истом рачунарском систему комуницирају, односно размењују податке. Међутим, када се процеси налазе на удаљеним рачунарским системима, омогућавање комуникације између њих је знатно сложеније и захтева коришћење мрежних протокола и физичке инфраструктуре. Са тог аспекта се целокупна рачунарско-телекомуникациона инфраструктура може гледати као подршка комуникацији између процеса који се налазе на различитим рачунарима.



*Слика 1. Рачунарска мрежа омогућава комуникацију између удаљених процеса*

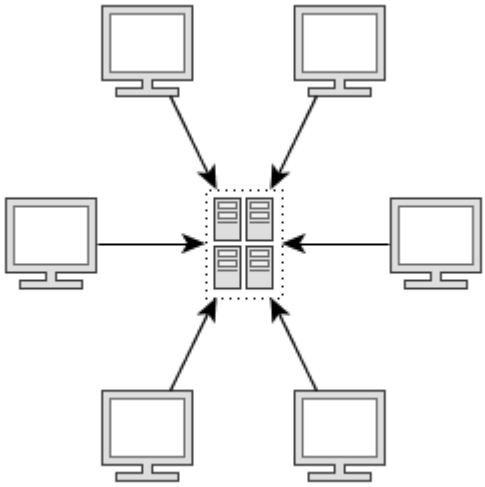
Савремени оперативни системи најчешће одрађују сав посао везан за коришћење протокола на транспортном и нижим комуникационим слојевима, па се мрежно програмирање своди на избор одговарајуће софтверске архитектуре и развој протокола на слоју апликације уз коришћења услуга повезивања које нуди оперативни систем - тзв. програмирање сокета (енгл. *socket programming*). Додатно, Веб програмирање, као један од данас најпопуларнијих облика реализације мрежног софтвера, омогућава коришћење већ постојећих протокола на слоју апликације (*HTTP*, *HTTPS*, *SOAP*...) што знатно поједностављује и убрзава развој мрежног софтвера.

### 1.3.1. Софтверске архитектуре

Софтверске архитектуре дефинишу на који начин се организују функције софтвера за дистрибуирану обраду података. За потребе одговарања на

различите потребе корисника рачунарских мрежа развијено је више софтверских архитектура које пружају одговарајуће функционалности.

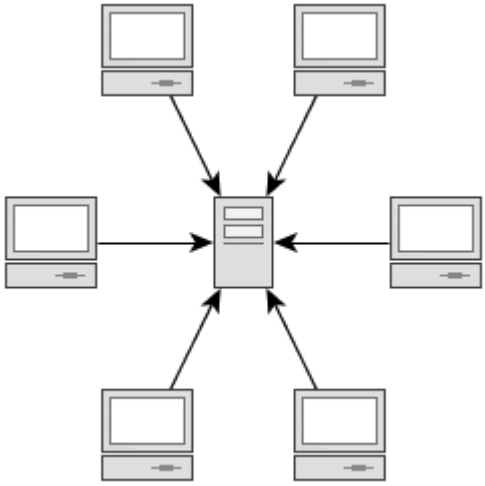
Прва мрежна сотфверско-хардверска архитектура јесте тзв. *host-based* архитектура коју су чинили мејн-фрејм рачунари и на њих прикључени терминали. Код ове архитектуре је целокупну обраду података извршавао мејн-фрејм рачунар а терминали су чинили интерфејс између мејн-фрејм рачунара и крајњих корисника. Данас одређени елементи ове архитектуре постају поново актуелни кроз Веб апликације и оперативне системе, као и рачунарство у облаку (енгл. *cloud computing*).



*Слика 1. Топологија мејн-фрејм архитектуре*

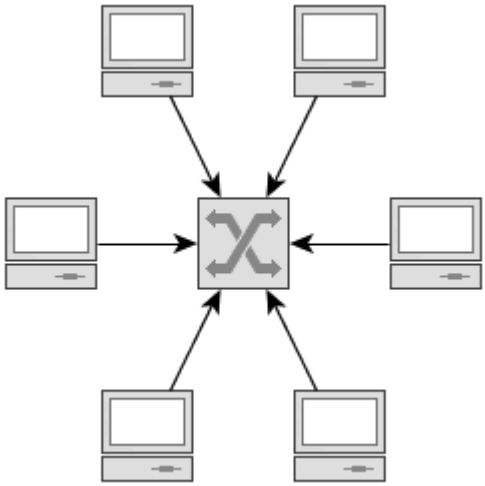
Вероватно најпопуларнија мрежна софтверска архитектура данас јесте клијент-сервер архитектура. Ова архитектура представља корак од мејн-фрејм архитектуре ка обради података на клијентима, али са одређеном централизованом обрадом на серверу.

Описаним приступом је омогућено растерећивање сервера кроз измештање одређених функција на клијенте, уз задржавање могућности централизовања одређених функција за потребе безбедности и стабилности система. Реализација ове архитектуре подразумева постојање упарених софтверских компонената - клијентске и серверске. Клијент-сервер архитектура може бити реализована у два или више слојева.



*Слика 2. Топологија клијент-сервер архитектуре*

Архитектура равноправних чланова (енгл. *peer-to-peer*) је софтверска архитектура са највишим нивоом аутономије у дистрибурираној обради података. Код ове архитектуре нема хијерархијске поделе чланова, већ сви чланови обављају локалну обраду података, уз могућност обраћања осталим члановима за функције и ресурсе који нису локално доступни. Основни недостатак архитектуре равноправних чланова односи се на отежану администрацију и контролу чланова.

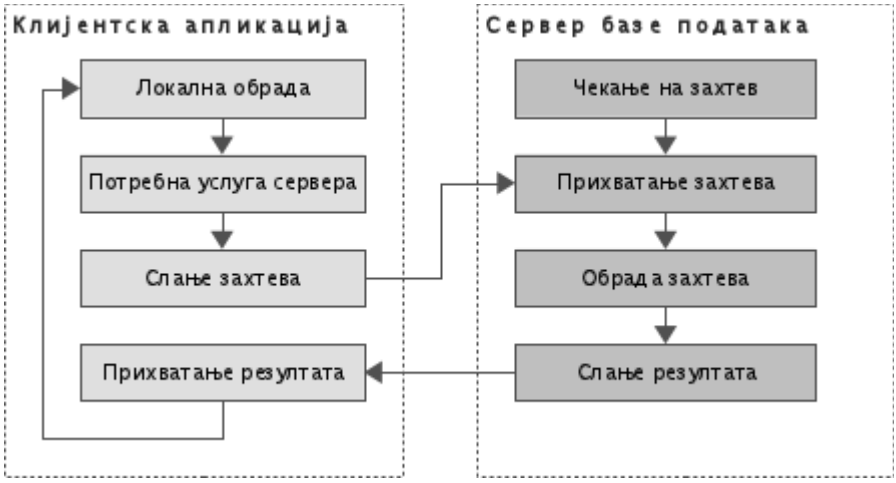


*Слика 3. Топологија архитектуре равноправних чланова*

Сервисно оријентисана архитектура један је од савремених приступа у дистрибуираном рачунарству. Основна идеја код ове архитектуре јесте омогућавање лакшег развоја и одржавања информационих система кроз мрежно коришћење софтверских функција доступних на осталим рачунарима. Данас постоји велики број затворених и отворених технологија којима се реализују системи засновани на овој архитектури.

### 1.3.1.1. Клијент-сервер архитектура

Клијент-сервер архитектура представља један од данас најчешће коришћених приступа код дистрибуиране обраде података. Корени ове архитектуре налазе се код мејн-фрејм рачунара и на њих прикључених терминала. Сличност са мејн-фрејм архитектуром јесте постојање једног члана способног за извршавање задатака који су ван могућности осталих чланова мреже. Постоје, међутим, битне разлике између ове две архитектуре. Код мејн-фрејм архитектуре терминали немају никакву могућност обраде података, док код клијент-сервер архитектуре клијенти од сервера добијају податке које, затим, користе у локалном процесу обраде. Затим, мејн-фрејм рачунари представљају аутономне чланове мреже који за процес обраде података користе локалне ресурсе. Насупрот томе, сервер се у виду клијента може обратити другим серверима у мрежи за одређени ресурс или дистрибуирану обраду.

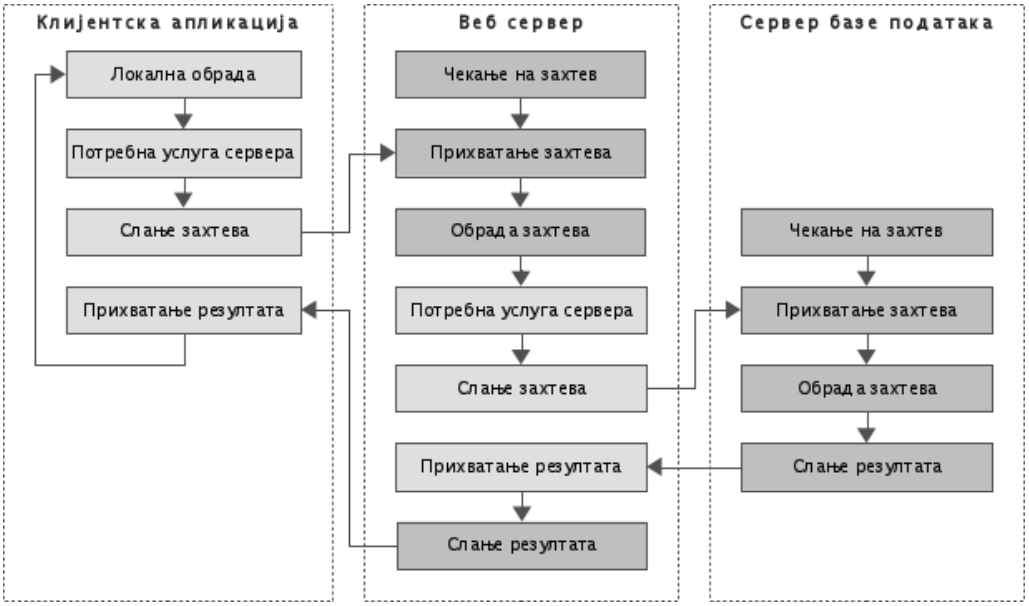



*Слика 1. Двослојна клијент-сервер архитектура*

Основна клијент-сервер архитектура подразумева постојање два слоја - слоја клијента и слоја сервера (слика 1). Код двослојне клијент-сервер архитектуре софтверско решење се разлаже на једну искључиво клијентску, и једну

искључиво серверску компоненту. Пример основне, двослојне клијент-сервер архитектуре јесте мрежа у којој корисници путем апликација на својим рачунарима приступају централној бази података.

У пракси су честе и клијент-сервер архитектуре са три слоја јер се њима лакше решавају питања заштите података и контроле приступа. На пример, Веб апликације најчешће користе трослојну клијент-сервер архитектуру на такав начин да основни клијентски део чини Веб браузер, средњи слој чини апликативни Веб сервер, а сервер базе података чини трећи, серверски слој (слика 2). У таквој организацији средњи слој је реализован као серверска компонента, гледано са аспекта Веб браузера, односно као клијентска компонента, гледано са аспекта сервера базе података.




*Слика 2. Трослојна клијент-сервер архитектура*

На примеру трослојне архитектуре јасно се може стећи увид да се појмови „клијент" и „сервер" не односе на хардверске карактеристике рачунара, већ на типове процеса, или чак њихове процедуре, који се на тим рачунарима извршавају. Уколико неки процес, или један његов део, захтева услугу од другог процеса (било на локалном или на удаљеном рачунару) његово понашање спада у клијентски део клијент-сервер архитектуре. И обрнуто, уколико неки процес чека на захтеве других процеса, његово понашање спада у серверски део клијент-сервер архитектуре. Опште правило за одређивање улоге код клијент-сервер архитектуре дефинише да је клијентска страна она која од друге

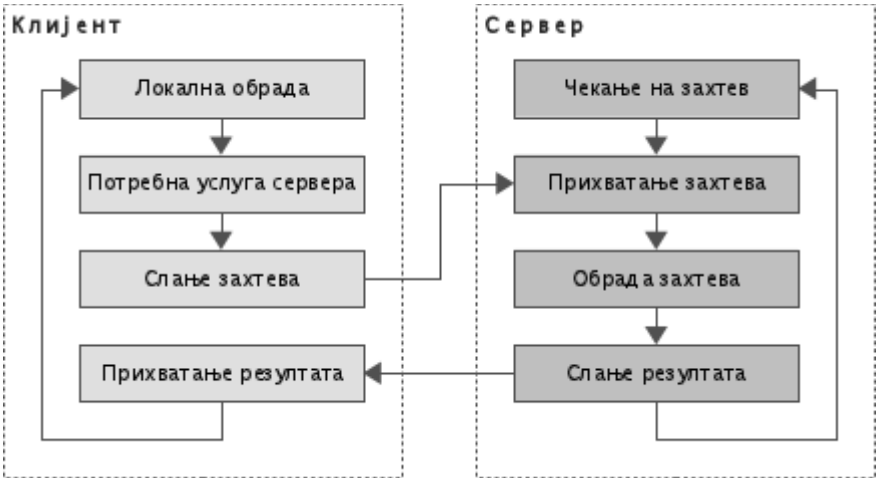
стране захтева услугу, а да је серверска страна она која чека на захтев од друге стране и испоручује одговор на њега.

#### 1.3.1.1.1. Итеративна и конкурентна обрада захтева

Постоји велики број различитих стратегија за оптимално искоришћавање процесорских, меморијских и комуникационих капацитета рачунарских система и мрежа. Сервери се, у зависности од начина на који распоређују обраду вишеструких захтева, деле на:

1. сервере са итеративном обрадом захтева и
2. сервере са конкурентном обрадом захтева.

Сервери са итеративном обрадом захтева ослушкују на додељеном порту чекајући на захтев клијента. Након прихватања захтева клијента захтеви осталих клијената се одбацују или чекају у улазном баферу све док се прихваћени захтев не обради и резултати његове обраде пошаљу натраг клијенту.




*Слика 1. Итеративна обрада захтева*

Мана итеративне обраде захтева је у томе што такав приступ може знатно утицати на перформансе у смислу броја обрађених захтева по јединици времена. Уколико обрада једног захтева у току свог извршења заузме све ресурсе сервера, перформансе се могу сматрати оптималним. Међутим, уколико обрада захтева одузме додатно време услед чекања на ресурс (који није потребан за обраду осталих захтева, одбачених или који чекају у улазном баферу), итеративни приступ показује лошије перформансе од конкурентног. Главна предност итеративног приступа јесте елиминисање проблема

конкурентног приступа интерним ресурсима сервера.

Конкурентна обрада захтева је приступ који нуди боље перформансе од итеративног приступа у ситуацијама у којима сервер обрађује велики број захтева од стране више клијената. Побољшање пефроманси произилази из могућности обраде више захтева паралелно. Паралелна обрада се постиже покретањем новог процеса (или нити процеса, у зависности од самог софтвера и од оперативног система) за обраду сваког клијентског захтева.




*Слика 2. Конкурентна обрада захтева*

За овакав приступ је потребан комплекснији серверски софтвер који се састоји од диспечерског дела (дела који је задужен за прихватање захтева и покретање процеса њихове обраде) и дела који је задужен за обраду захтева. Конкурентна обрада захтева може у одређеним ситуацијама показати слабије перформансе од итеративне обраде услед трошења процесорског времена на покретање нових процеса за обраду захтева. Такође, софтвер који омогућава конкурентну обраду је комплекснији јер интерно решава конкурентни приступ системским ресурснима. Софтвер за конкурентну обраду најчешће унапред покреће одређен број процеса за обраду захтева а по потреби тај број повећава до конфигурационе вредности или ограничења системским ресурсима.

### 1.3.1.2. Архитектура равноправних чланова

Архитектура равноправних чланова (енгл. *peer-to-peer, P2P*) представља вид диструбуираног рачунарства у коме сваки чвор (енгл. node) има двоструку улогу.

Сваки чвор мреже равноправних чланова комуникацију са осталим члановима P2P мреже обавља путем софтвера који се може понашати и као клијент (захтевајући податке или услуге од осталих чворова) и као сервер (одговарајући на захтеве осталих чворова). На овај начин архитектура равноправних чланова омогућава већу аутономију чланова мреже. Архитектура равноправних чланова се углавном примењује код потреба у којима постоји већа толеранција грешке код дистрибуиране одбраде. Главне примене су размена фајлова, директна комуникација, дистрибуирана обрада велике количине података, хеш табеле, софтвер за забаву...

Главни недостатак архитектуре равноправних чланова односи се на адресовање чланова мреже. Док је код клијент-сервер мрежа потребно само да клијенти имају информацију о томе који сервери су доступни на мрежи (и која је њихова адреса) код архитектуре равноправних чланова потребно је да сваки члан има информације о доступности осталих чланова. Из тог разлога постоји више различитих варијација унутар архитектуре равноправних чланова:

1. децентрализована архитектура равноправних чланова
2. централизована архитектура равноправних чланова
3. хибридна архитектура равноправних чланова

Децентрализована архитектура равноправних чланова представља архитектуру најближу основном моделу. Она је сачињена искључиво од peer чворова који међусобно комуницирају директно. Код децентрализоване архитектуре равноправних чланова не постоји централни регистар чланова већ се откривање осталих чланова врши преко интерног протокола (најчешће у виду *broadcast* захтева).

Централизована архитектура равноправних чланова представља мешавину архитектуре равноправних чланова и клијент-сервер архитектуре. Као и код децентрализоване архитектуре равноправних чланова мрежу чине чворови који међусобно размењују податке директно, са том разликом што постоји централни сервер чији је задатак евидентирање чланова мреже.

Хибридна архитектура представља варијанту централизоване архитектуре равноправних чланова која се користи у случајевима када се мрежа састоји од великог броја чворова, а улога сервера подразумева и додатне операције сем евидентирања чланова. Код хибридне архитектуре равноправних чланова улогу сервера преузима већи број тзв. „супер чворова". Ове чворове најближи чворови користе као сервере док адресне информације везане за остале чворове супер чворови међусобно размењују.

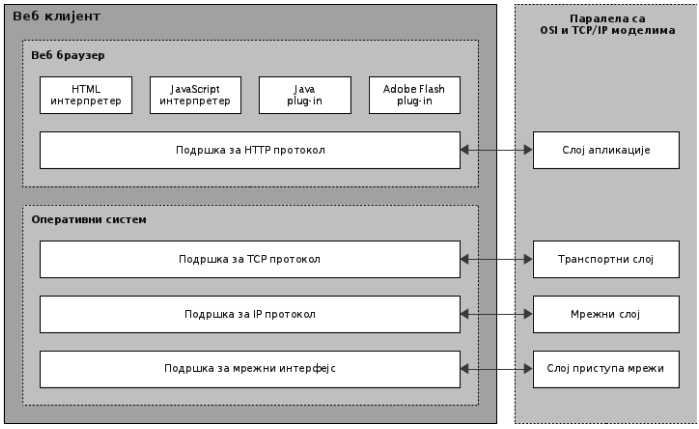
# 2. Веб и Веб технологије

Веб (енгл. *World Wide Web, Web*) је данас најпопуларнији сервис Интернет мреже. Развијен је почетком 90-их година прошлога века, паралелно са развојем *HTML* језика и *HyperText Transfer Protocol*-а, развијеним од стране Тим Бернерс-Лија, физичара из Европске организације за нуклеарно истраживање. Назив је добио по првом Веб браузеру који је поседовао графички кориснички интерфејс.

Веб је инцијално развијен да омогући приступ хипертекстуалним документима који су се налазили на различитим серверима. Временом су се појавили програмски језици на страни сервера који су омогућили креирање динамичких докумената као одговора на специфичне захтеве клијента, затим технологије за развој активних компонената на страни клијента (*JavaScript, Macromedia/Adobe Flash, Microsoft Silverlight* и друге), протоколи за сервисно оријентисану архитектуру засновани на Вебу и слично. Тиме је Веб све више постајао платформа за развој мрежних апликација тако да данас прети да у потпуности замени традиционалне десктоп апликације.

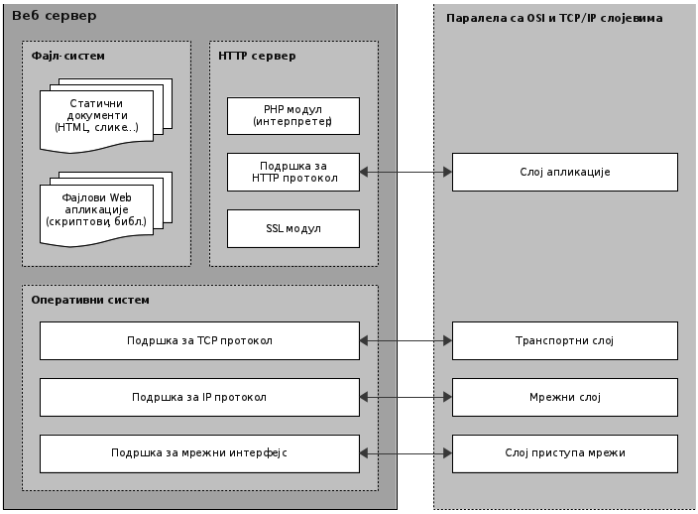
## 2.1. Клијент-сервер Веб модел

Архитектура Веба је заснована на класичном клијент-сервер моделу. Клијентску компоненту подразумевано представља Веб браузер у виду десктоп апликације, мада данас ресурсима и сервисима на Вебу приступају и друге софтверске компоненте (Веб апликације, апликације на мобилним телефонима и слично).



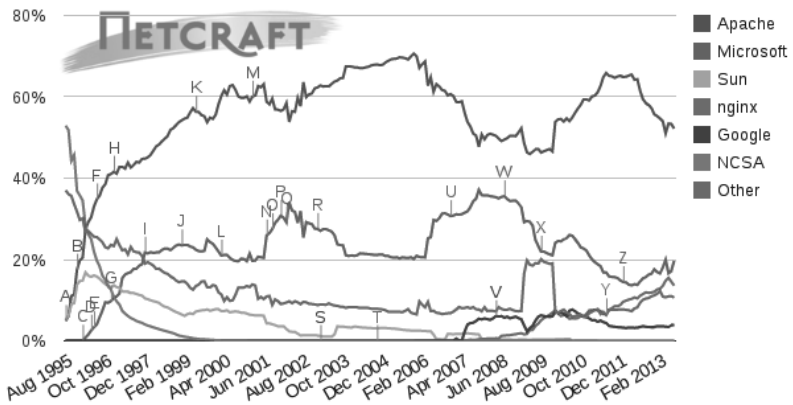

*Слика 1. Архитектура Веб клијента и паралела са комуникационим слојевима*

На серверској страни Веба налази се тзв. Веб или *HTTP* сервер чија је иницијална улога била да са фајл-система учита тражени статични документ. Данас Веб сервери имају далеко веће могућности с обзиром да представљају основну платформу за извршавање Веб апликација.




*Слика 2. Архитектура Веб сервера и паралела са комуникационим слојевима*

Најпопуларнији сервер данас, а готово и током целог постојања Веба је тзв. Апач (енгл. *Apache HTTP server*) који је 1995. године развијен као наставак обустављеног пројекта у америчком националном центру за супер-рачунарство.

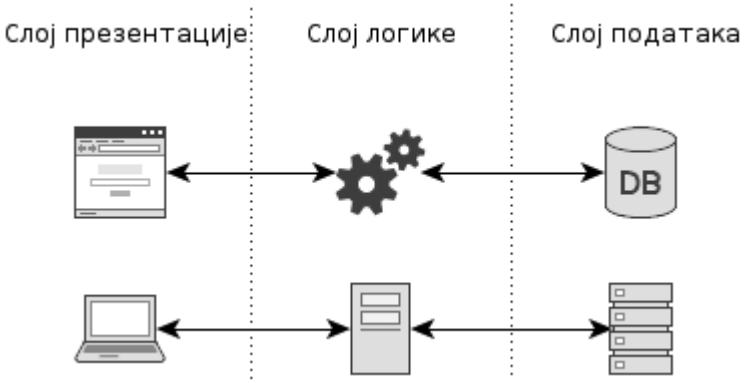



*Слика 3. Учешће најпопуларнијих Веб сервера на Вебу (извор: неткрафт)*

Поред Апача, данас се често користи и *nginx* код сајтова који имају велики број посета, као и *Microsoft IIS* код сајтова заснованих на технологијама те компаније.

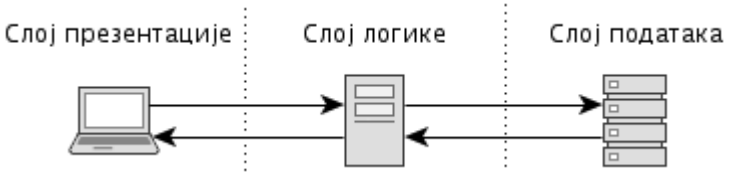
## 2.2. Трослојна архитектура

Веб апликације се углавном могу посматрати подељене на три слоја: слој презентације, слој логике и слој података. Слој презентације обухвата резултате обраде захтева које сервер шаље клијенту и који се интерпретира у Веб браузеру клијента. Овај резултат може бити ресурс било ког типа (слика, скрипт и сл.), с тим да је то углавном *HTML* документ или формулар који садржи контроле за слање нових захтева слоју логике. Поједностављено речено, слој презентације обухвата све компоненте које чине интерфејс ка кориснику.




*Слика 1. Трослојна архитектура Веб апликација*

Основни задатак слоја логике је обрада корисничких захтева и слање добијених резултата. У типичне кораке обраде захтева спадају нормализација и интерпретација захтева, преузимање потребних података из слоја података, кодовање података у *HTML* или други тражени облик и слање тако добијеног резултата. За дефинисање логике за обраду захтева користе се програмски језици чији се програми извршавају на страни сервера, односно на серверу слоја логике.




*Слика 2. Циклус комуникације између слојева трослојне архитектуре*

На слоју података се чувају подаци који се користе при креирању одговора на захтеве клијента. На пример, на Веб сајту за преглед најновијих вести, сам

садржај вести се мора чувати на некој локацији која је доступна слоју логике. За складиштење и рад са подацима углавном се користе системи за управљање базама података (СУБП) или системи фајлова.




*Слика 3. Статични Веб сајтови немају слој логике и података*

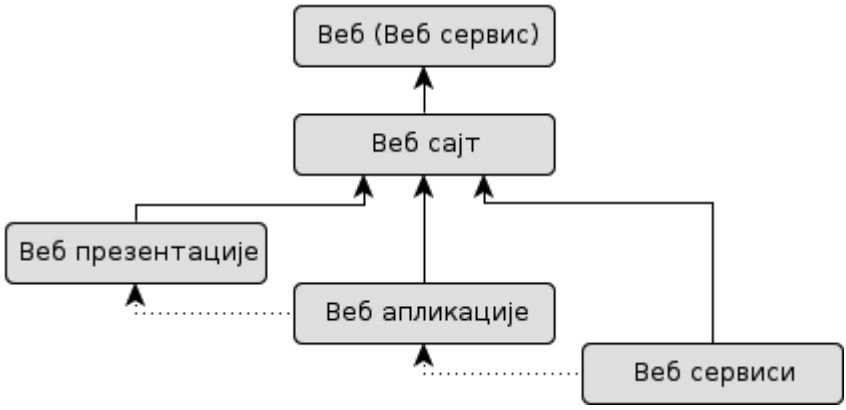
Код статичних Веб сајтова слојеви логике и података нису присутни јер се садржај чува у свом коначном облику, као статичан скуп *HTML* страница, *JavaScript* скриптова, *CSS* стилова, слика... Из тог разлога се статични Веб сајтови често називају и Веб презентацијама. Данас, услед веће доступности *CMS* система, као и све нижег нивоа потребног знања за рад са њима, корисници све чешће користе ове системе (који су, у суштини, Веб апликације) за развој Веб презентација. У складу са тим, данас се термин „Веб презентација" првенствено користи да означи низак ниво интерактивности Веб сајта, пре него за дефинисање структуре и технологија који се користе за његов развој и испоруку.

## 2.3. Сајтови, презентације, апликације и сервиси

У пракси се често јавља терминолошка нејасноћа када су у питању Веб презентације, апликације и сервиси. Најчешће се не прави одговарајућа разлика између Веба и Веб сервиса, као ни између Веб сајтова, Веб апликација и Веб презентација.

Сам Веб представља један од сервиса Интернета. У том смислу, израз **Веб сервис** означава један тип услуге за кориснике Интернет мреже, мада се у пракси реч **сервис** све више изоставља и поприма исто значење као и када се користи у множини. Када се користи у множини, овај израз - Веб сервиси (енгл. *Web services*) - не односи се на сам Веб већ на специфичан тип ресурса доступног путем Веба. Веб сервиси нису намењени непосредној употреби од стране људи

већ се користе за комуникацију између Веб апликација или њихових компонената.




*Слика 1. Хијерархијски однос израза везаних за Веб*

**Веб сајтови** (енгл. *Web site, website*) представљају основну јединицу за организовање садржаја на Вебу. Најчешће се под термином Веб сајт мисли на одређени Интернет домен (нпр. *www.racunarskemreze.com*) мада се Веб сајт пре односи на скуп сродних и међусобно повезаних садржаја који чине једну садржајну целину. У складу са тим, могуће је да се садржај једног Веб сајта налази на више Интернет домена, као и да се на једном Интернет домену налази више Веб сајтова.

У раним фазама Веба садржаје на њему су углавном чинили статични садржаји који су презентовани корисницима. Интеракција са оваквим садржајима је била на минималном нивоу а на основу њих се усталио израз **Веб презентација** (енгл. *Web presentation*). Временом, почели су да се појављују скрипт језици који су омогућавали динамичко креирање Веб садржаја те се за њих све чешће користио израз **Веб апликације**.

Почетком XXI века започео је и убрзан развој система за управљање саржајем на Вебу (енгл. *Content Management System, CMS*). Коришћењем ових система све већи број корисника Интернета дошао је у могућност да брзо и једноставно креира и одржава своје Веб сајтове, чак и они који нису имали никакво техничко познавање *HTML*-а и осталих Веб технологија. С обзиром да је интерактивност оваквих сајтова остала на минималном нивоу, као и да је њихова сврха и даље била презентовање одређеног садржаја, израз Веб презентације је наставио да се користи и за њих, без обира на то што је сам садржај добијен коришћењем Веб апликације. Данас, дакле, израз Веб презентација описује Веб садржаје са

ниским нивоом интерактивности. Са друге стране, израз Веб апликација се користи за интерактивне Веб садржаје, односно сајтове чија улога није једносмерно информисање корисника већ пружање одређене услуге.

## 2.4. Развојна и продукциона окружења

Два основна окружења Веб апликација су развојно и продукционо. **Развојно окружење**, како му и само име каже, служи за развој Веб апликација од стране једне особе или развојног тима (програмера, дизајнера, администатора...). Развојно окружење симулира планирано продукционо окружење по присутним компонентама (Веб сервер, сервер базе података, модули за коришћене програмске језике...) али углавном има знатно мањи капацитет. Још једна карактеристика развојног окружења је и то да је оно подешено да одмах прикаже све грешке и недостатке Веб апликације која се развија.




*Слика 1. Однос развојног и продукционог окружења*

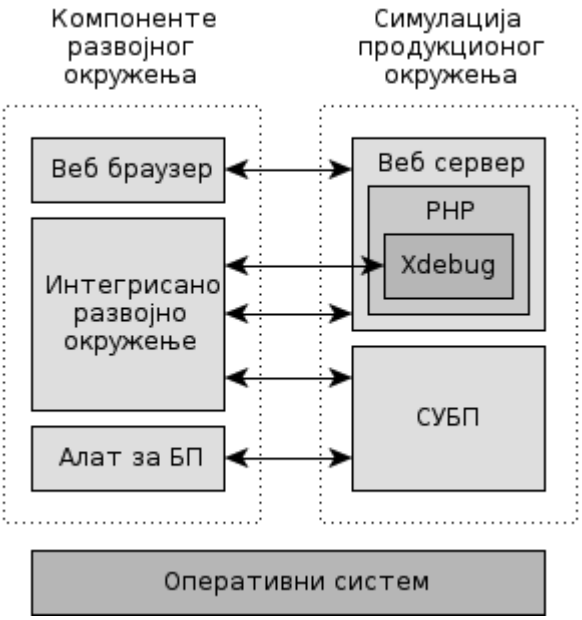
**Продукционо окружење** служи за стварно извршавање развијене Веб апликације и њено коришћење од стране крајњих корисника. Обично има далеко веће ресурсе на располагању од развојног окружења, што значи да је у могућности да подржи већи број клијената који паралелно шаљу захтеве. Насупрот развојном окружењу, код продукционог окружења се информације о грешкама из безбедносних разлога не приказују, мада се складиште у дневнике

догађаја за потребе анализе од стране развојног тима.

Код великих и сложених пројеката најчешће није могуће, или макар није безбедно, развијене измене Веб апликације из развојног окружења директно применити у продукционо. У таквим ситуацијама се између развојног и продукционог окружења користи и међуокружење (енгл. *staging environment*) у коме се проверава како ће се развијене измене одразити на активно продукционо окружење. Другим речима, задатак овог окружења је да се у њему тестирају измене и отклоне све неисправности.

## 2.4.1. Развојно окружење

Развојно окружење за Веб је софтверско окружење које се састоји од компонената које омогућавају развој и тестирање Веб апликације или презентације на којој се ради и компонената које симулирају продукционо окружење у коме је могуће извршавање Веб апликација. С обзиром да је у развојном окружењу циљ да се све грешке које постоје у апликацији прикажу одмах, компоненте које симулирају продукционо окружење су тако и подешене, а често садрже и додатке који у томе помажу (какав је, на пример, *Xdebug* додатак за *PHP* интерпретер).




*Слика 1. Архитектура развојног окружења за Веб*

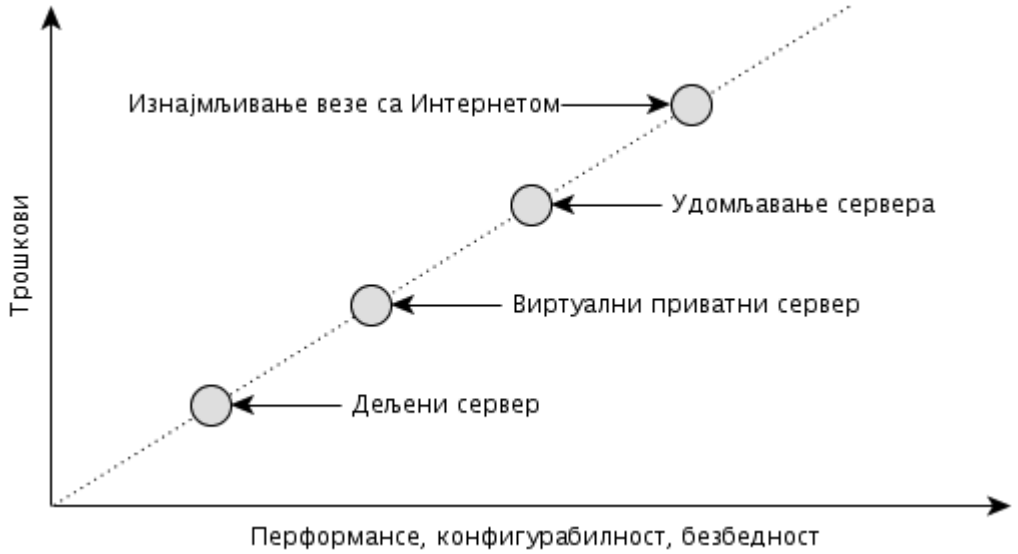
У компоненте које служе за развој спадају интегрисано развојно окружење, алат за рад са базама података (који се често налази у интегрисаном развојном окружењу) и Веб браузер (који служи за тестирање урађеног). **Интегрисано развојно окружење** (енгл. *Integrated Development Environment, IDE*) је софтверски алат који у себи садржи више компонената, као што су уређивач текста, односно програмских кодова, алат за отклањање грешака, алат за моделовање база података и рад са подацима у њима, и слично. Савремена интегрисана развојна окружења су углавном модуларна па се њихов скуп подржаних функционалности може проширивати.

Развојно окружење се може налазити на једном рачунару (физичком или виртуалном) када су у питању мањи пројекти на којима ради само једна особа. Са друге стране, код већих пројеката се развојно окружење налази на више рачунара, што рачунара које користе програмери и дизајнери, што сервера који симулирају продукционо окружење. У тим случајевима се подаци на различитим рачунарима синхронизују коришћењем *FTP* протокола или напреднијим алатима за контролу верзија (енгл. *version control*) као што су *Git* и *Subversion*.

## 2.4.2. Продукционо окружење

Продукционо окружење за Веб је софтверско окружење које омогућава извршавање Веб апликација у циљу одговарања на захтеве клијената. У односу на развојно окружење за Веб, продукционо окружење је једноставније јер не садржи развојне алате а и сами сервери у њему садрже само компоненте неопходне за рад, тако да често немају инсталиран ни графички кориснички интерфејс. Са друге стране, продукциона окружења често имају инсталиране додатне компоненте које омогућавају повећање перформанси (нпр. алати за кеширање) и нивоа безбедности.

Постоји више решења за обезбеђивање продукционог окружења за сопствену Веб апликацију, односно за њено објављивање на Интернету. Најјефтиније, али уједно и најскромније продукционо окружење чини дељени Веб сервер. Затим следи изнајмљивање виртуалног приватног сервера и постављање сопственог сервера код Интернет провајдера. Коначно, најмоћнија, али уједно и најскупља опција је развој сопственог дата-центра и измнајмљивање брзе везе са Интернет мрежом.



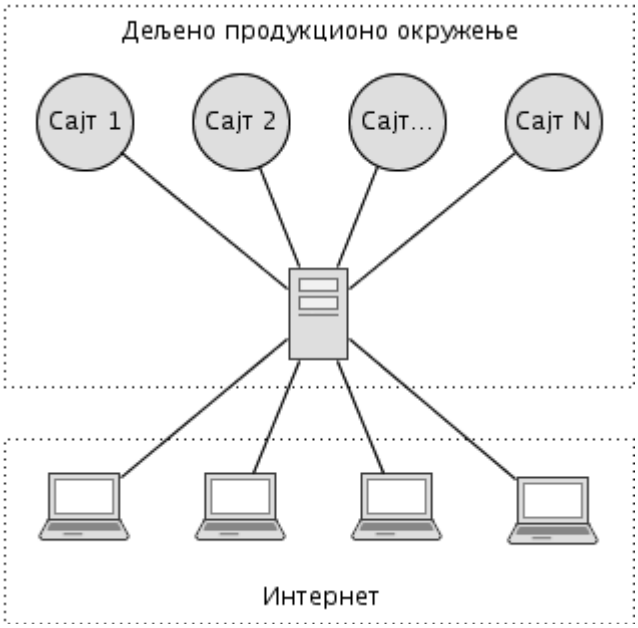

*Слика 1. Однос цене и карактеристика различитих типова продукционих окружења*

Компоненте продукционог окружења су тако подешене да о евентуалним грешкама и проблемима у раду Веб апликације клијентима дају што мање информација. Ово се ради из безбедносних разлога, јер је циљ да потенцијални

нападач о Веб апликацији зна што мање и да му се на тај начин смањи простор за напад. Са друге стране, поменуте информације о поменутим грешкама и проблемима се складиште у дневнике догађаја и касније користе за отклањање грешака из Веб апликације и анализу покушаних напада.

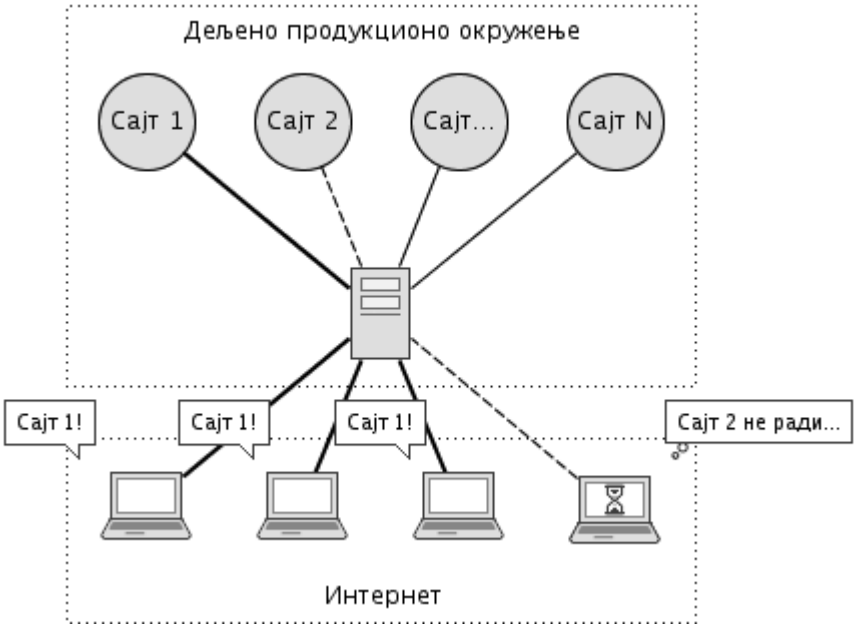
### 2.4.2.1. Дељени сервер

Једна од најјефтинијих, а самим тим и најпопуларнијих опција за објављивање Веб сајтова на Интернету је коришћење дељеног продукционог окружења у виду дељења сервера (енгл. *shared hosting*). Осим због ниске цене, ова опција је популарна и из тог разлога што корисник не мора да брине о администрацији и одржавању сервера, у шта спадају хардвер, софтвер и мрежни линк, већ је то задатак компаније која нуди услугу.



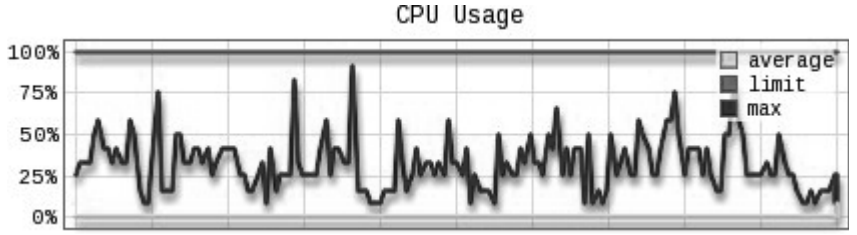

*Слика 1. Код дељеног сервера више сајтова се испоручује са истог сервера*

Једна од главних мана коришћења дељеног сервера јесте ограничена количина ресурса (централни процесор, меморија, комуникациони канал) која је на располагању сваком од сајтова, с обзиром да се ресурси деле на више сајтова. Из тог разлога дељени сервер није добар избор за популарне Веб сајтове код којих су битне висока доступност и високе перформансе.




*Слика 2. Код дељеног сервера може се јавити недоступност услед преоптерећења*

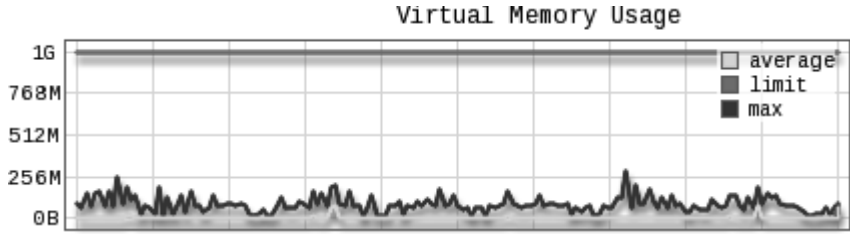
У случају када се у једном тренутку појави велики број посетилаца за неки од сајтова, у зависности од подешавања сервера, може се догодити да се захтеви једног дела клијената одбаце или да се сви захтеви обраде али да због тога трпе остали сајтови. У пракси се углавном за сваког од корисника дефинишу ограничења везана за употребу централног процесора, радне меморије, броја конекција и заузеће комуникационог канала.




*Слика 3. Дијаграм искоришћења додељеног времена централног процесора*

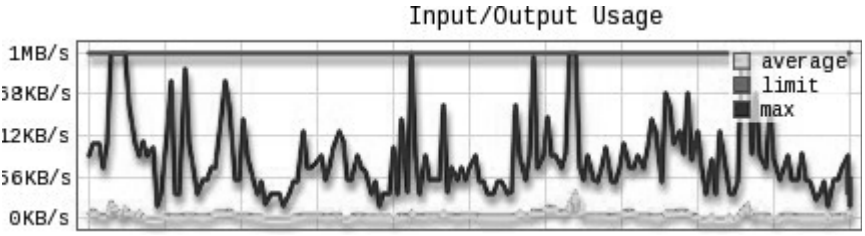
На слици 3. је приказан пример дијаграма искоришћења додељеног времена централног процесора. На дијаграму се може видети да ово време углавном задовољава потребе сајтова чији се скриптови извршавају у оквиру једног налога. Добра страна ограничења у погледу времена централног процесора је

да се захтеви углавном неће одбацивати, већ ће бити потребно више времена за њихово извршавање. Са друге стране, спор одзив сајтова има негативан утицај на њихову ефикасност, позиционирање на претраживачима...




*Слика 4. Дијаграм искоришћења додељене радне меморије*

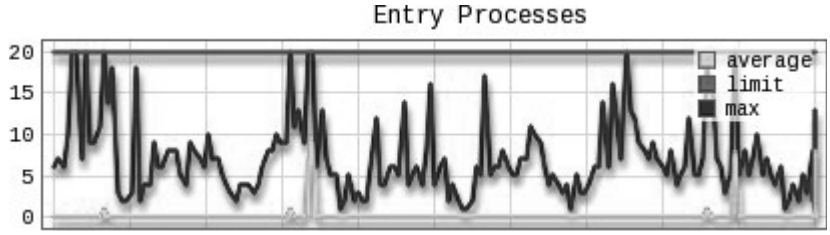
Достизање ограничења у заузећу додељене радне меморије, за разлику од достизања ограничења у погледу времена централног процесора, доводи до насилног прекида обраде захтева. У том случају се, на пример, могу јавити оштећења база података. Заузеће радне меморије зависи од тога колико су скриптови Веб апликације оптимизовани, као и од тога колики је број паралелних захтева.




*Слика 5. Дијаграм коришћења капацитета комуникационог канала*

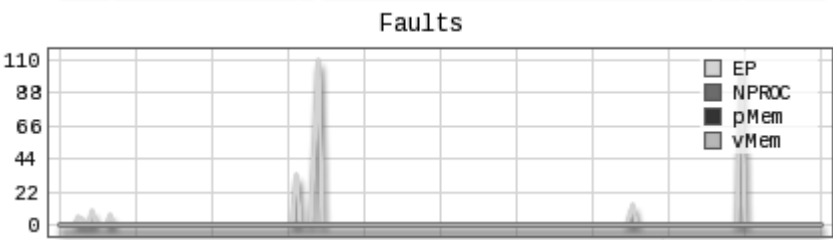
С обзиром на то да је ограничен и капацитет комуникационих канала којима су сервери дељеног продукционог окружења повезани на Интернет, то ограничење се преноси и на кориснике. На слици 5. се може видети да је администратор ограничио брзину преноса података на 1MB/s, као и да је то ограничње неколико пута достигнуто у посматраном периоду.

Поред ограничења у тренутној брзини преноса података, у пракси је често присутно и ограничење у количини пренетих података, најчешће на месечном нивоу (на пример, од неколико десетина GB па до неколико TB). Стратегије пружаоца услуга у случају достизања овог ограничења могу бити различите али се углавном своде на смањење брзине преноса података на минималну или на одбијање захтева ка сајтовима који се налазе на том налогу.




*Слика 6. Дијаграм броја паралелних захтева*

Као што је раније речено, утрошак процесорског времена, заузеће радне меморије и заузеће комункационог канала зависе од два фактора: степена оптимизације скриптова који се извршавају на серверу и броја паралелних захтева. С обзиром на то да је први фактор у надлежности корисника, администратори често ограничавају и број паралелних захтева ка ресурсима који припадају једном налогу. На слици 6. је дат пример броја паралелних захтева (процеса) а са истог дијаграма се може уочити и да је ограничење од 20 паралелних захтева достигнуто неколико пута у посматраном периоду. Након достизања ограничења сервер одбија даље захтеве све док се не заврши извршавање неког од постојећих процеса.




*Слика 7. Дијаграм коришћења централног процесора*

О свим наведеним ограничењима треба водити рачуна с обзиром да она доводе до успореног одговарања на захтеве посетилаца или чак до њиховог одбијања. На слици 7. је приказан дијаграм проблема у извршавању захтева са кога се види да су се у одређеним периодима јављале грешке као последица ограничења у броју обрада паралелних захтева. Занимљиво је и да је у овом конкретном случају узрок великог броја паралелних процеса било њихово успорено извршавање услед отказивања хардверских компонената сервера.

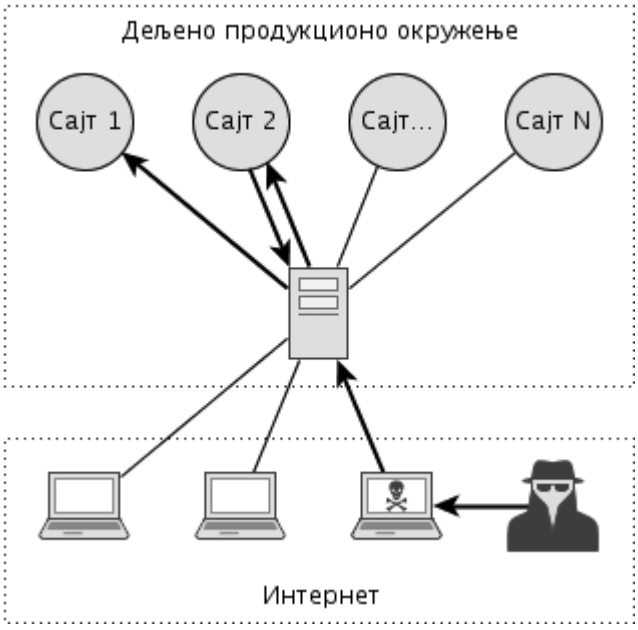
Није редак случај ни да компанија која пружа услугу дељеног сервера одређени сајт једноставно искључи уколико примети да се за обраду захтева ка њему троши далеко више ресурса од очекиваног просека. Логика иза тога је да је боље изгубити једног клијента уместо више других чијих је чување сајтова

исплативо, а који би били угрожени недостатком ресурса за нормалан рад.

Ограничење које у неким случајевима може бити значајно је и смањена могућност за специфично подешавање сервера за потребе неке Веб апликације која се на њему извршава. Администратори често не желе да врше сложеније измене из страха да се не угрози стабилност или безбедност сервера, односно осталих сајтова који се на њему држе. Није редак случај да сами администратори не поседују потребно знање из области у којој се захтева измена, те да из тог разлога не желе да начине чак ни мање измене у подешавањима.

#### 2.4.2.1.1. Безбедносни аспекат

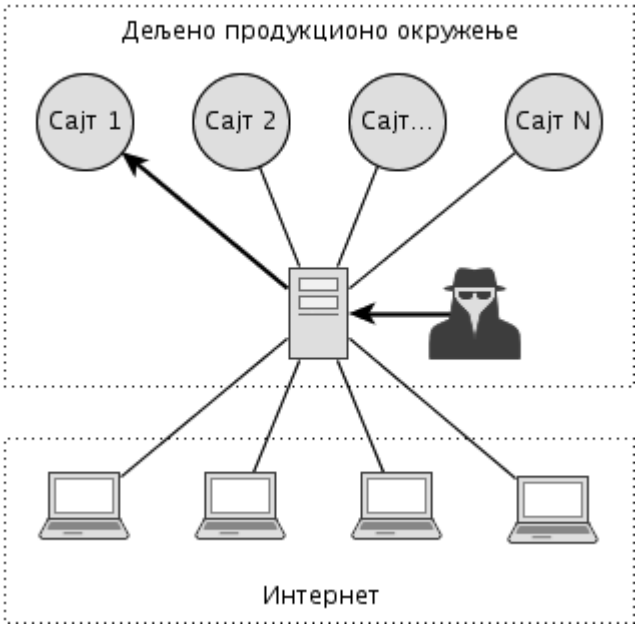
Још један недостатак дељеног продукционог окружења је и низак ниво безбедности апликације и података. С обзиром да се на истом серверу извршавају програми великог броја корисника, потенцијални нападачи до могућности за извршавање својих програма на серверу могу доћи кроз улогу регуларног корисника.




*Слика 3. Код дељеног окружења пропусти у једном сајту могу угрозити остале*

На пример, уколико сервер није правилно подешен, скриптови једног корисника могу из конфигурационих фајлова другог корисника ишчитати параметре за

приступ серверу базе података, а затим и преузети целокупну базу података у којој се налазе поверљиви подаци (лозинке уредника, приватне поруке, адресе за електронску пошту и друго). Често се дешава и да неки од корисника у дељеном окружењу користе застареле верзије бесплатних Веб апликација које садрже пропусте чијим се искоришћавањем може компромитовати цео сервер.

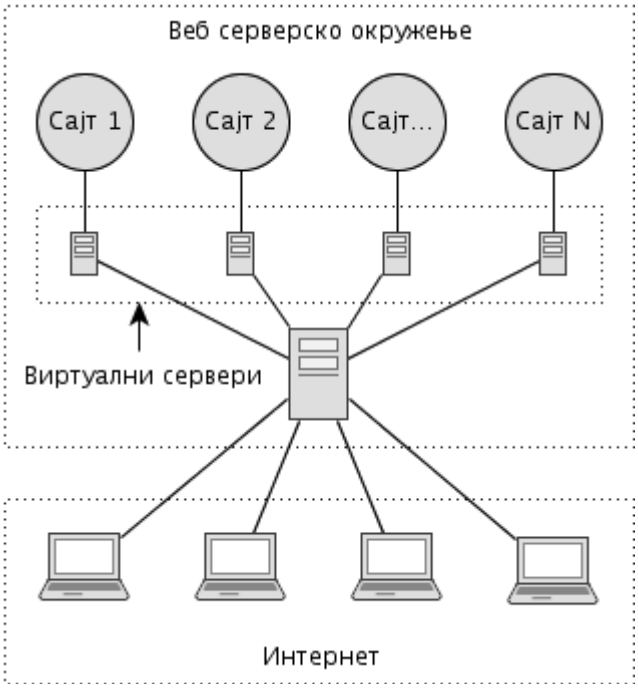



*Слика 4. Администратор дељеног продукционог окружења има приступ свим подацима и фајловима апликације*

Следеће о чему треба водити рачуна код дељеног продукционог окружења, када су безбедност и поверљивост у питању, јесте ниво поверења у самог администратора система. Треба имати у виду да администратор има приступ свим подацима везаним за Веб сајтове који се налазе на серверима које он одржава. У те податке спадају фајлови апликација, базе података, статистике коришћења и слично. Са друге стране, у уговорима које корисник дељеног продукционог окружења потписује са пружаоцем услуга ово питање се обично не поставља, нити корисник има икакву информацију ко је администратор сервера на коме се његов сервер налази.

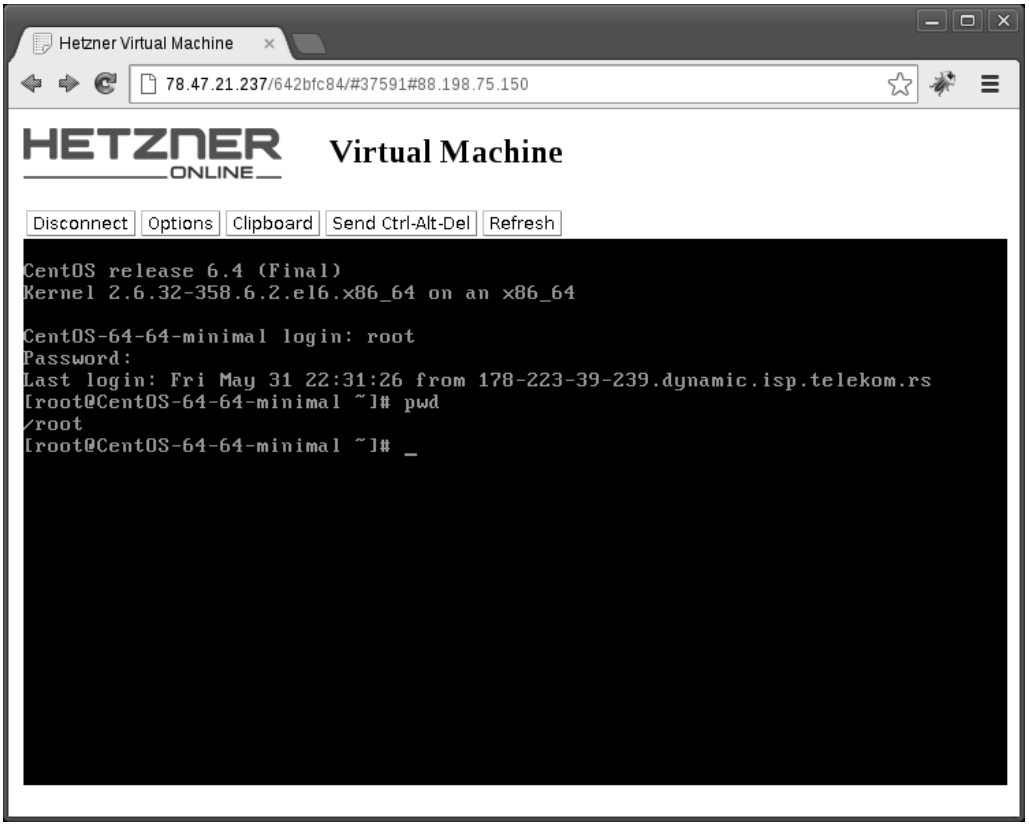
### 2.4.2.2. Виртуални приватни сервери

Виртуални приватни сервери (енгл. *Virtual Private Server, VPS*) представљају напреднију варијанту дељеног продукционог окружења, односно јефтинију варијанту удомљавања сервера. У овом случају корисници од хостинг компанија закупљују једну или више виртуалних машина на којима ће се чувати и извршавати њихови Веб сајтови.




*Слика 1. На једном виртуалном приватном серверу се налазе сајтови једног корисника*

У предности код коришћења виртуалних приватних сервера, у односу на дељено продукционо окружење, спадају виши ниво заштите, гарантовани ниво ресурса резервисан за корисника и могућност подешавања свих параметара сервера у складу са сопственим потребама, укључујући и инсталацију додатног софтвера. Са друге стране, цена ове услуге је и преко десет пута виша у односу на дељено продукционо окружење. Додатно, корисник подразумевано преузима на себе задатак администрирања сервера - освежавање верзија софтвера, прављење резервних копија и друго.



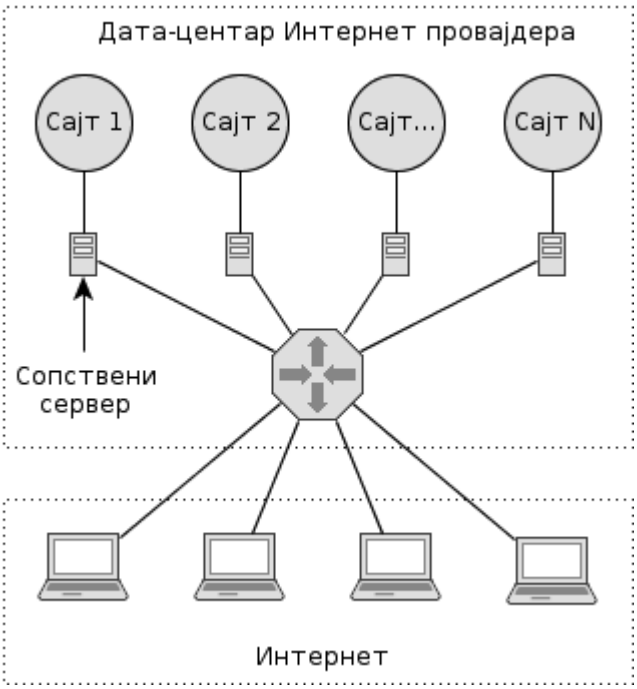

*Слика 2. Приступ серверу коришћењем SSH протокола и Јава аплета*

И у случају коришћења виртуалног приватног сервера значајно је питање поверења у администратора физичког сервера пошто он и даље може да приступи свим подацима који се налазе на виртуалном серверу, подацима које он размењује са клијентима, као и статистикама коришћења.

Хостинг компаније најчешће имају предефинисане виртуалне машине са инсталираним одређеним оперативним системом (углавном нека од популарних дистрибуција Линукса) и основним софтвером потребним зарад Веб сервера. С обзиром да је у том случају процес прављења нове инстанце веома једноставан, време потребно за активирање ове услуге мери се у минутима. Уколико се на виртуалну машину инсталира додатни софтвер за који се плаћа лиценца (на пример *CPanel* или *MS Windows* као оперативни систем), њену цену је потребно додати на основну цену услуге.

### 2.4.2.3. Удомљавање сервера

Удомљавање сервера (енгл. *server housing*) је приступ код кога корисник сам обезбеђује хардвер сервера, на њега инсталира жељени софтвер и поставља Веб сајт, а затим такав сервер односи до Интернет провајдера. Провајдер сервер односи у своју серверску салу и тамо га прикључује на мрежни кабл и кабл за напајање електричном енергијом. У серверској сали су обезбеђени потребни услови за несметан рад сервера (температура, влажност ваздуха, непрекидно напајање и слично).

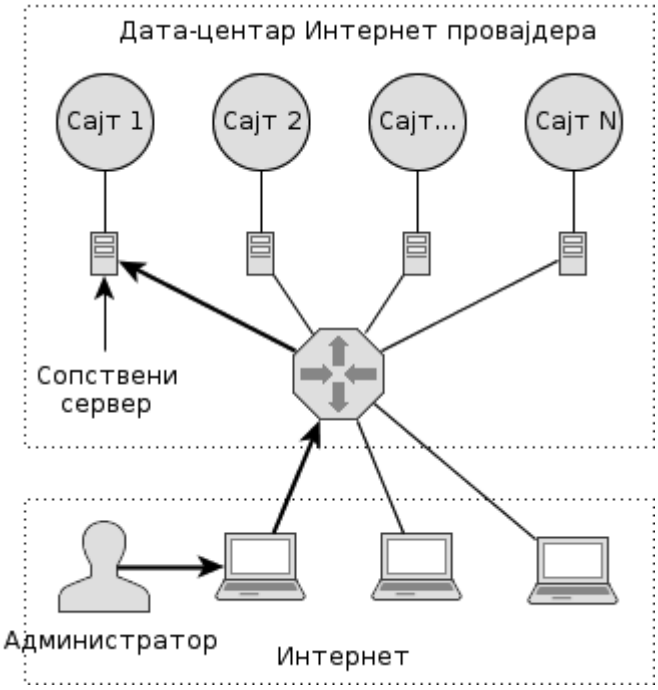



*Слика 1. Сопствени сервер се поставља у дата-центар Интернет провајдера*

Администрирање сервера је код његовог удомљавања углавном задатак самог корисника услуге. То значи да је корисник задужен да пре одношења сервера код Интернет провајдера на њега инсталира одговарајући оперативни систем и подеси мрежне интерфејсе у складу са параметрима које је добио од провајдера.

Одржавање сервера - накнадна подешавања сервиса, освежавање верзија софтвера, прављење резервних копија и слично - обавља се путем Интернета. У случају да постоји потреба за физичком интервенцијом на серверу (блокада

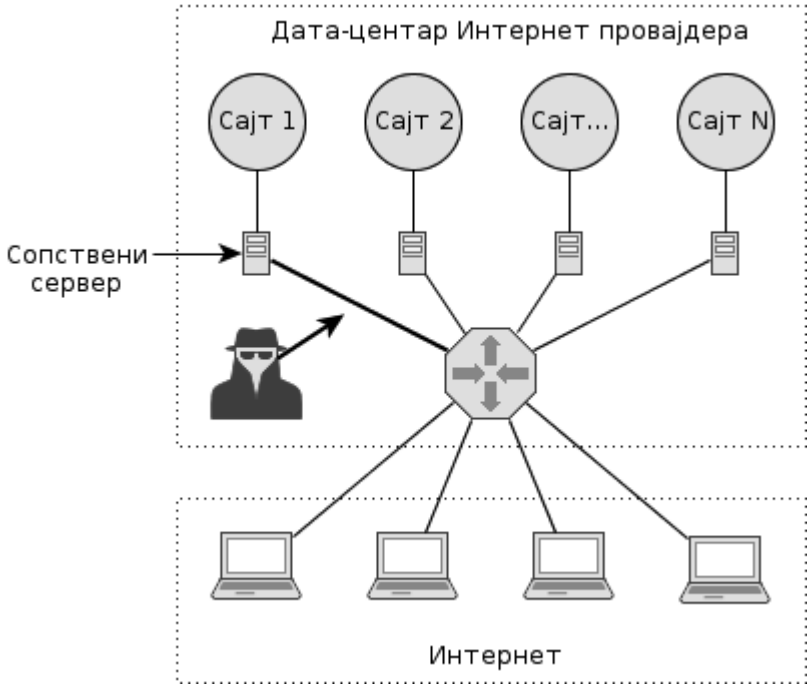
оперативног система, надоградња...) сервер се преузима од провајдера и враћа након интервенције.




*Слика 2. Одржавање сервера се обавља путем Интернета*

Када је безбедносни аспекат у питању удомљавање физичког сервера нуди виши ниво безбедности од изнајмљивања виртуалног приватног сервера и коришћења дељеног окружења. Међутим, основни ризик који овде остаје (када је у питању неповерење у администраторе Интернет провајдера) је надгледање комуникационог канала. Дакле, администратори Интернет провајдера могу да читају све поруке које нису шифроване, као и да добију статистичке податке за посматрани период (број захтева, пренета количина података и слично).

Додатни ризик који постоји је и искључивање сервера од стране администатора Интернет провајдера (насилно или регуларно) и копирање садржаја са извађених хард-дискова. Овакав догађај, након враћања дискова и поновног укључивања сервера, може проћи неопажено или се заменити са краткотрајним прекидом рада комуникационог канала, као и правдати разлозима као што су проблеми на електричној мрежи и слично.



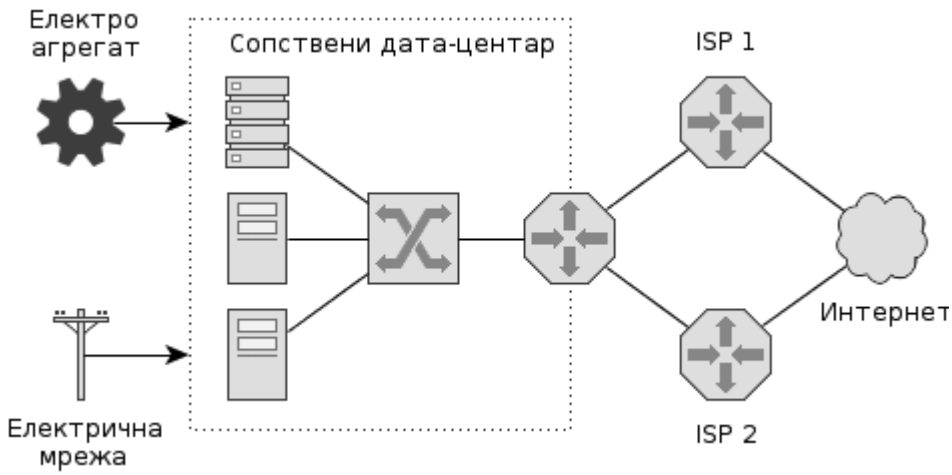

*Слика 3. Администратор Интернет провајдера има приступ ком. каналу*

Да би се заштитили подаци од оваквих напада потребно је користити шифроване фајл-системе на серверима који су физички доступни потенцијалним нападачима.

### 2.4.2.4. Изнајмљивање брзе везе са Интернетом

Најфлексибилније и најбезбедније решење за продукционо Веб окружење, са потенцијално највишим перформансама, јесте повезивање сопственог дата-центра са Интернет мрежом и испорука Веб апликација са сопствених Веб сервера. Овакав тип решења је посебно згодан за кориснике који имају потребу да њихове јавно доступне Веб апликације комуницирају са интерним системима и базама података (на пример, портал за електронско банкарство).

При развоју оваквог решења треба обезбедити брзу и поуздану везу са Интернет мрежом. Везе треба да буду засноване на технологијама које нуде мало кашњење и брз пренос података у оба смера. Препоручљиво је коришћење вишеструких веза ка Интернету (по могућству преко различитих Интернет провајдера) и система који омогућавају несметан рад уколико нека од веза постане недоступна.




*Слика 1. Сопствени дата-центар са Интернет везом*

Иако развој сопственог решења нуди највиши степен безбедности и флексибилности, само корисници са највећим потребама и могућностима одлучују се за њега. Разлог томе су високе цене Интернет веза које се овде користе, али и додатни трошкови који се односе на куповину и одржавање рачунарских система, обезбеђивање сталног и поузданог напајања електричном енергијом, обезбеђивање одговарајуће температуре и влажности ваздуха у дата-центру и слично.

### 2.4.2.5. Мреже за испоруку садржаја

Код Веб сајтова који имају велики број посетилаца широм света превелика удаљеност између посетилаца и сервера на којима се налазе садржаји Веб сајта може бити узрок ниских перформанси. Додатно, на превише велики број захтева често није могуће одговорити са једне локације, коришћењем једног комуникационог канала ка Интернету. Да би се на овакве изазове одговорило у тим случајевима користе се мреже за испоруку садржаја.

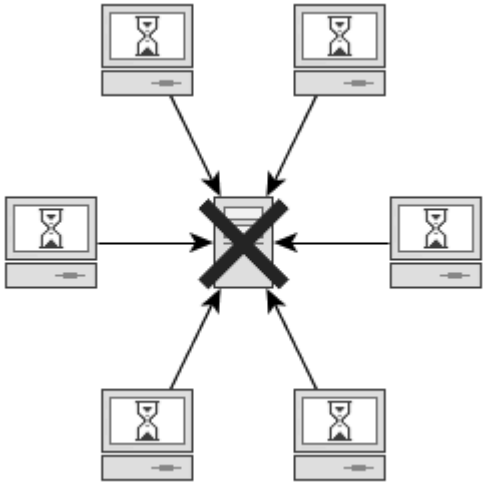
**Мреже за испоруку садржаја** (енгл. *Content Delivery Network, CDN*) су системи који имају сопствене сервере у чвориштима Интернет провајдера на различитим географским локацијама. Клијенти који закупе коришћење мреже за испоруку података статичне садржаје сајта (слике, *JavaScript* и *CSS* кодове и слично) снимају на њене сервере. Након тога, будући захтеви клијената за тим садржајима се преусмеравају на најближи сервер мреже за испоруку који их садржи.

## 2.4.3. Доступност

Доступност неког ресурса или услуге најчешће се израчунава као однос времена у коме је исте могуће користити и укупног времена које се посматра. На пример, Веб сервер који се годишње искључује шест пута на по сат времена, због одржавања хардвера и софтвера или нестанка струје, има доступност 8.754 од 8.760 сати, односно 99,931507 процената на годишњем нивоу.

У пракси се за означавање доступности неког система често користи и метрика „деветки", односно број деветки у проценту доступности (на пример, у претходном случају је то „три деветке"). Данас се често захтева доступност система на нивоу „пет деветки" (око 5 минута недоступности годишње) или чак „шест деветки" (око 30 секунди недоступности годишње).

Недоступност се може јавити услед отказивања система, али и услед његовог преоптерећивања - добијања већег броја захтева него што је у стању да обради. Такође, у дељеним продукционим окружењима сам сервер често има довољно ресурса да одговори на велики број захтева али је капацитет ограничен софтверски, у складу са количином ресурса које је корисник закупио.



*Слика 1. Недоступност сервера онемогућава коришћење сервиса и података на њему*

Оно што треба имати у виду је и то да иако је систем укључен и оперативни систем се извршава (енгл. *uptime*) не подразумева да су сервиси и подаци на њему доступни. На пример, за одржавање већине софтверских сервиса није потребно ресетовати рачунар, али је саме сервисе потребно привремено

искључити. Такође, код већине система је након физичког укључивања рачунара потребно знатно време док се не покрену сви његови делови (оперативни систем, мрежни интерфејси, сервиси...) и достигне потпуна оперативност.

У извештају IBM-а[7] се процењује да је 1996. године недоступност рачунарских система коштала америчку привреду преко 4,5 милијарде америчких долара, а тај износ је за 1999. годину процењен на око 6,6 милијарди. Висине савремених губитака је тешко проценити али се може проценити да су оне потенцијално далеко веће од наведених с обзиром на то да све већи број пословних система уопште није у стању да функционише без информационе подршке и везе са Интернетом.

Једноставан пример губитака услед недоступности Веб сајта јесу плаћене рекламе на Гугл претраживачу. Ове рекламе се плаћају по клику на њих (енгл. *Cost Per Click, CPC*), без обзира на то да ли је Веб страница ка којој води линк у реклами доступна или не. То значи да би једнодневна недоступност Веб сајта чији је дневни рекламни буџет €100 створила губитак раван трогодишњој цени услуге дељеног продукционог окружења. При том не треба заборавити ни негативне ефекте као што су стварање лоше слике код потрошача, лошије позиционирање на претраживачима и слично.

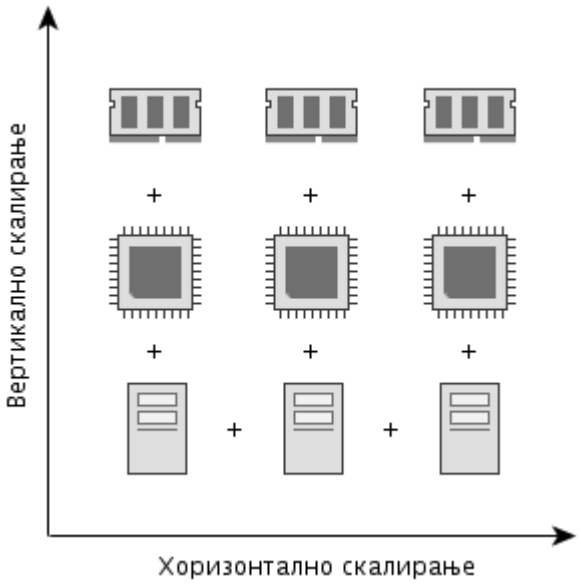
## 2.4.4. Управљање оптерећењем и скалабилност

Једна од веома значајних карактеристика технологија које се бирају за развој одређеног Веб решења је скалабилност. **Скалабилност** представља могућност да се постојеће решење прошири у циљу одговарања на новонастале потребе, односно да није потребно да се због њих развија ново решење.

Често се димензије продукционог окружења више пута мењају током животног века апликације. На пример, одређени Веб сајт може почети као једноставна апликација у дељеном продукционом окружењу али због пораста популарности убрзо захтевати прелазак на издвојени виртуални приватни сервер, а потом и на неколико физичких наменских сервера.

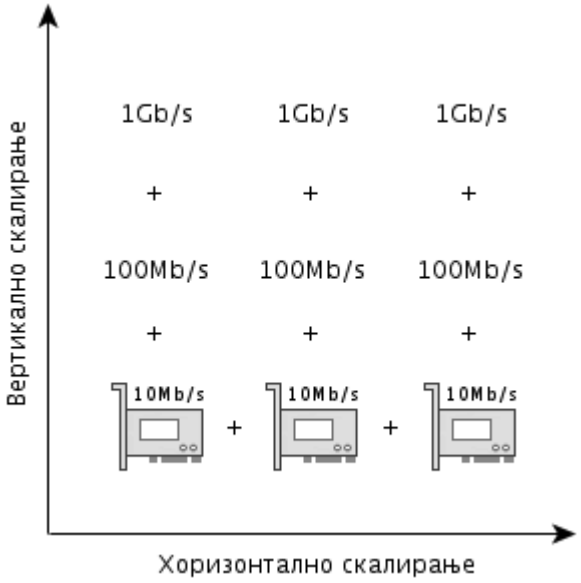
Потреба за унапређивањем продукционог окружења Веб апликације углавном се јавља због пораста оптерећења - последице повећаног броја захтева клијената. Међутим, она се може јавити и због пораста вредности података у њему, односно потребе да се они боље заштите. На пример, Веб сајт на коме се уведе плаћање путем платних картица углавном је препоручљиво изместити из дељеног продукционог окружења на сопствени сервер.

7 IBM Global Services, Improving Systems Availability, http://www.cs.cmu.edu/~priya/hawht.pdf




*Слика 1. Хоризонтално и вертикално скалирање серверске инфраструктуре*

Скалирање може бити хоризонтално и вертикално. **Хоризонтално скалирање** сервера подразумева појачавање постојећих сервера, примарно кроз додавање нових и јачих процесора и радне меморије.




*Слика 2. Хоризонтално и вертикално скалирање комуникационих канала*

**Вертикално скалирање** подразумева додавање нових сервера и распоређивање функција на њих. При том треба имати у виду да коришћене софтверске технологије морају имати могућност да искористе више процесора/сервера, мада је то код Веб апликација углавном случај.

Оно што треба имати у виду је да се скалабилност не односи само на серверску инфраструктуру, већ и на комуникационе канале, коришћене технологије, софтверску архитектуру и друго. Скалирање комуникационих канала такође може бити хоризонтално и вертикално, где се код хоризонталног скалирања повећава пропусна моћ постојећих канала а код вертикалног се додају нови.

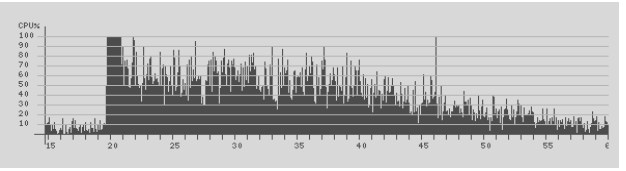
### 2.4.4.1. Пример вертикалног скалирања

У наставку је дат пример вертикалног скалирања Веб платформе, описан у раду под називом „Развој система за проверу знања студената у оквирима постојећег ИС универзитета“[8].

Прва коришћења система реализована су на хардверској платформи *Intel P4* процесор 2.4GHz, 1GB RAM, 1Gb/s мрежним линком. Уско грло се појавило приликом приступања провери од стране већег броја студената истовремено (~50 студената), када се брзина одзива драматично смањила (~120 секунди). Систем је дизајниран да за сваког студента у провери врши рандомизацију редоследа питања и понуђених одговора. Максимално оптерећење процесора у тренутку приступа провери могло се изразити на следећи начин

$$O_{\max} = Ns * Np * (\frac{No}{Np} + 1)$$

при чему *Ns* представља број студента, *Np* је број питања у провери и *No* представља укупан број одговора у провери.



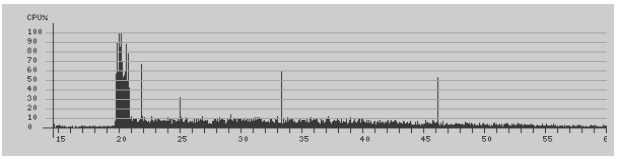

*Слика 1. Оптерећење апликативног сервера са Intel P4 2,4GHz процесором*

На горњој слици представљено је оптерећење серверског процесора током релизације провере над 63 студента који су одговарали на 100 питања и бирали

8 Шимић Г., Јевремовић А., „Развој система за проверу знања студената у оквирима постојећег ИС универзитета“, 15. телекомуникациони форум Телфор 2007., Београд, 2007., CD издање

између 400 одговора. Уочава се да се вршно оптерећење појавило на почетку периода проверавања, тј. у тренутку преузимања питања и одговора.

У следећој верзији хардверске платформе, наведене компоненте су замењене бољим: процесор са 8 језгара типа *Intel Xeon* 3,2GHz и 8GB радне меморије (ECC).




*Слика 2. Оптерећење процесора апликативног сервера са 8 језгара типа Intel Xeon 3,2GHz*

Овом променом, уз додатно оптимизовање кода апликације (екстракција и енкапсулација учестало коришћених метода у статичке методе посебне класе, редукција позива брокерских класа према БП коришћењем унапред припремљених и парсираних *SQL* наредби), добијени су задовољавајући резултати, сликовито представљени на претходној слици. Достигнута је планирана брзина одзива система (на клијентској страни, у интранет окружењу) – боља од 5 секунди, са преко 100 студената, преко 500 питања и преко 2.000 одговора. Евентуална уска грла у будућој експлоатацији могу се јавити услед ограничене пропусне моћи комуникационих канала, што се може решити додавањем нових мрежних адаптера и оптимизацијом конфигурације диспечерске компоненте система.

# 3. Језик за означавање хипер-текста

Језик за означавање хипер-текста (енгл. *Hyper-Text Markup Language, HTML*) је основни језик за креирање докумената на Вебу, као и за креирање корисничког интерфејса за Веб апликације. Спада у групу језика за означавање а често се, погрешно, сматра програмским језиком. Развијен је почетком деведесетих година прошлога века од стране Тим Бернерс-Лија, физичара из Европске организације за нуклеарно истраживање, са првенственим циљем да омогући једноставан приступ референцираним садржајима.

## 3.1. Структура и садржај хипертекстуалних докумената

*HTML* документи, у суштини, нису ништа више од обичних текстуалних фајлова који могу бити креирани и мењани у било ком програму за уређивање текста. Међутим, да би се одређени текстуални документ сматрао *HTML*-ом потребно је да поштује одређену верзију/варијанту *HTML* стандарда. Тренутно актуелна верзија *HTML* стандарда је 5, с тим да је највећи број докумената на Вебу развијен коришћењем претходних верзија.

Два основна елемента које *HTML* документ мора да садржи су декларација његовог типа и сам *html* елемент. Декларација типа документа (енгл. *Document Type Declaration, DTD*) служи да Веб браузеру клијента пружи информацију о ком типу/верзији HTML-а је реч, односно на који начин треба интерпретирати садржај. Декларација типа токумента је код 5. верзије *HTML*-а веома једноставна

```
<!DOCTYPE html>
```

док је у претходним верзијама било више варијанти због постојања више стандарда.

Сам *html* елемент унутар *HTML* документа чини његов садржај а састоји се из два основна дела, заглавља и тела, за које се користе ознаке *head* и *body*:

```
<html>
   <head>
      ...
   </head>
```

```
   <body>
      ...
   </body>
</html>
```

У заглављу документа чувају се информације о самом документу - наслов, кључне речи, опис, везе ка стиловима приказа и *JavaScript* скриптовима, начин кодовања и слично - док се у телу налази садржај који се приказује у Веб браузеру.

Садржај се у *HTML* документима налази у виду елемената који се дефинишу одговарајућим ознакама. Ознаке (енгл. *tag*) су предефинисане одговарајућим *HTML* стандардом. Неки елементи представљају крајње чворове стабла, на пример:

```
<hr />
```

док неки елементи у себи могу садржати друге елементе или текстуални садржај:

```
<ul>
   <li>Први елемент листе</li>
   <li>Други елемент листе</li>
   <li>...</li>
</ul>
```

Елементима се могу додељивати различити, предефинисани атрибути који се дефинишу унутар њихових ознака:

```
<img src="slika.png" alt="опис слике" />
```

За неке атрибуте се не мора експлицитно задати њихова вредност, већ само њихово присуство указује на неку особину:

```
<select name="grupa">
   <option value="1">Прва</option>
   <option value="2" selected>Друга</option>
   <option value="3">Трећа</option>
```

```
</select>
```

Дати пример се односи на падајућу листу код које ће друга опција подразумевано бити изабрана (селектована) након учитавања документа.

Унутар тела документа могу се користити текстуалне ознаке - коментари - које се у Веб браузеру неће приказивати:

```
<p>Фијуче ветар у шибљу, леди пасаже и куће иза њих и гунђа у оџацима.</p>
<!-- На ово место додати други панграм -->
<p>Ниџо, чежњиво гледаш фотељу, а Ђура и Мика хоће позицију себи.</p>
```

Коментари се најчешће користе као подсетници за саме ауторе докумената или их генеришу програми на страни сервера за потребе лакшег проналажења евентуалних грешака.

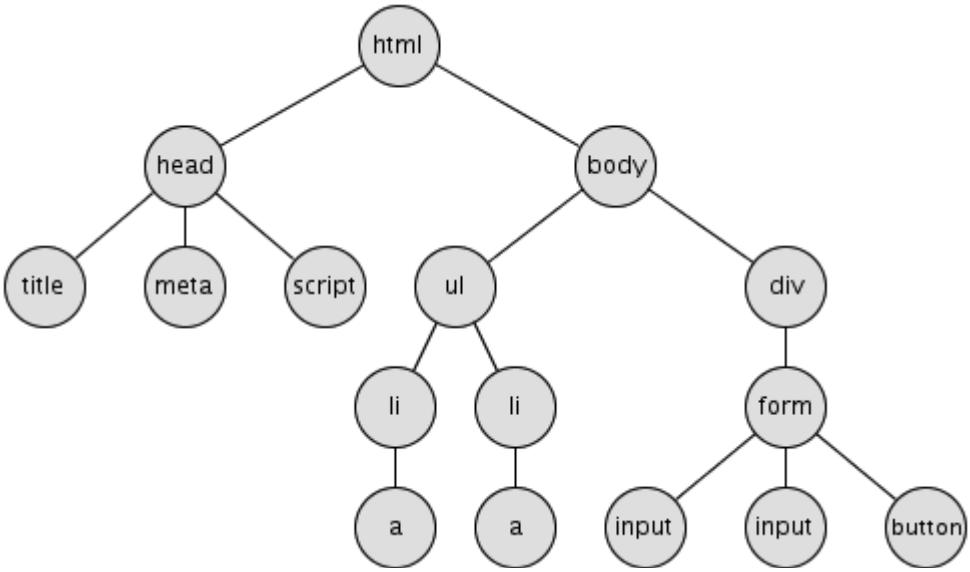
## 3.1.1. Хијерархијска структура HTML докумената

Елементи *HTML* докумената су организовани хијерархијски, у виду стабла. На пример, следећи код

```
<!DOCTYPE html>
<html>
   <head>
      <title>Пријава корисника</title>
      <meta charset="UTF-8">
      <script type="text/javascript" src="skript.js"></script>
   </head>
   <body>
      <ul id="menu">
         <li>
            <a href="index.html">Почетна</a>
         </li>
         <li>
            <a href="lozinka.html">Опоравак лозинке</a>
         </li>
      </ul>
```

```
      <div id="content">
         <form action="prijava.php">
            <input type="text" name="username" />
            <input type="password" name="password" />
            <button type="submit">Пријава</button>
         </form>
      </div>
   </body>
</html>
```

може се графички представити у виду стабла:




*Слика 1. Стабло елемената HTML документа*

Овакав начин организације садржаја *HTML* докумената омогућава њихово једноставније претраживање, као и приступање одређеним елементима од стране активних компонената на страни клијента.

## 3.2. Повезивање докумената путем хипер-веза

Постоје два основна типа повезивања *HTML* докумената са другим документима: повезивање са саставним ресурсима самог документа (слике, *CSS*

дефиниције, *JavaScript* документи и слично) и повезивање са другим документима путем хипер-веза. За први начин повезивања првенствено се користе инструкције у заглављу документа, док се хипер-везе користе у његовом телу.

**Хипер-везе** (енгл. *hyper-link*) омогућавају да се неком елементу у телу документа (део текста, слика...) придружи веза ка неком другом документу или делу (фрагменту) актуелног документа. Оне се у *HTML* документима креирају коришћењем ознаке *a,* а одредиште се дефинише у оквиру атрибута *href*:

```
Сајт о <a href="http://www.racunarskemreze.com">рачунарским мрежама</a>.
```

Резултат интерпретирања наведеног кода дат је на наредној слици:

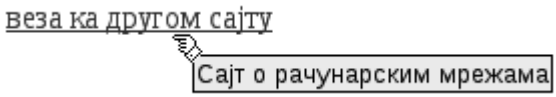
Сајт о рачунарским мрежама.

*Слика 1. Пример подразумеваног приказивања хипер-веза*

Још један специфичан атрибут за хипер-везе је и *target* којим се дефинише у ком оквиру или прозору ће се учитати документ ка коме води веза. На пример, уколико се атрибуту *target* додели вредност *_blank* документ ка коме води хипер-веза учитаће се у новом прозору. Још један општи атрибут који није лоше користити код хипер-веза је и атрибут *title*:

```
<a href="http://www.racunarskemreze.com" title="Сајт о рачунарским мрежама">
   веза ка другом сајту
</a>
```

којим се кориснику може дати додатно објашњење о одредишту ка коме веза води:




*Слика 2. Додатне информације о одредишту хипер-везе*

Осим ка другим документима, хипер-везе могу да повезују део документа у коме се налазе са одређеним делом (фрагментом) тог истог другог документа. За ову намену се у оквиру одредишта хипер-везе наводи идентификатор фрагмента самостално (уколико је локални документ у питању) или у комбинацији са адресом удаљеног документа:

```
<a href="clanak2.php#zakljucak">закључак другог чланка</a>
```

Да би Веб браузер знао на који део документа треба отићи, у њему мора да се налази **идентификатор фрагмента** (енгл. *fragment identifier*) који може бити хипер-веза са атрибутом *name*:

```
<a name="zakljucak">Закључак</a>
```

или било који HTML елемент са одговарајућом вредношћу за атрибут *id*:

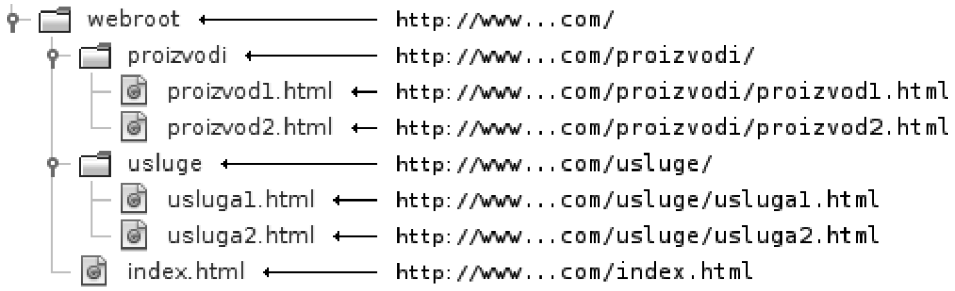
```
<h1 id="zakljucak">Закључак</a>
```

с тим да други приступ има предност у случају да су оба присутна у документу.

Када су у питању хипер-везе које воде ка другим документима, оне могу бити дефинисане коришћењем апсолутних и релативних адреса одредишта. Апсолутно адресовање подразумева унос целокупне адресе од корена Веб сајта а могу да укључују и сам назив домена (када је обавезно и навођење протокола):

```
<a href="/proizvodi/proizvod1.html">Производ 1</a>
<a href="http://www.example.com/proizvodi/proizvod1.html">Производ 1</a>
```

Укључивање домена у апсолутно адресовање докумената на локалном сајту није препоручиво јер се у случају промене домена адресе таквих хипер-веза морају ажурирати.




*Слика 3. Однос путања фајл-система и URL-а*

Релативно адресовање, са друге стране, подразумева коришћење адресе одредишног документа у односу на тренутни документ. На пример, у примеру датом на слици 3 релативно адресована хипер-веза ка страници *usluga2.html* у

оквиру странице *index.html* изгледала би:

```
<a href="usluge/usluga2.html">Услуга 2</a>
```

док би релативно адресована хипер-веза ка страници *proizvod1.html* у оквиру странице *usluga2.html* изгледала:

```
<a href="../proizvodi/proizvod1.html">Производ 1</a>
```

Релативно адресовање код хипер-веза ограничено је само на документе који се налазе на истом Веб сајту. Такође, у ситуацијама у којима је то могуће, боље је користити релативне путање уместо апсолутних јер су отпорније на промене локација на којима се документи налазе. На пример, уколико се садржај сајта *www.racunarskemreze.com* премести у директоријум:

```
www.racunarskemreze.com/stari_sajt
```

адресе свих апсолутно адресованих хипер-везе ће морати да се ажурирају, док код релативно-адресованих хипер-веза то није случај.

## 3.3. Уређивање текста

Основна намена HTML језика била је креирање докумената са текстуалним садржајем које је могуће аутоматски повезивати. У складу са тим *HTML* нуди широке могућности за форматизовање текстуалног садржаја (заглавља, параграфи, листе, табеле, специјални симболи...) и уметање графичких елемената.

За правилно приказивање текста у *HTML* документима Веб браузер мора да има информацију о томе на који начин је текст кодован. У раним фазама *HTML*-а било је омогућено само коришћење *ASCII* кодовања. Касније је уведена могућност коришћења кодних страница а данас је препоручено коришћење *unicode* стандарда. Постоји више начина да се Веб браузер информише о начину на који је текст документа кодован:

- у оквиру заглавља *HTTP* протокола,
- коришћењем *charset* атрибута код хипер-веза које воде ка документу,
- коришћењем *byte-order mark (BOM)* на почетку документа,
- и многи други

али је у 5. верзији *HTML*-а то најједноставније да се учини додавањем у заглавље документа ознаке *meta* са вредношћу *UTF-8* за атрибут *charset*:

```
<html>
   <head>
      ...
      <meta charset="UTF-8">
      ...
   </head>
   ...
</html>
```

Код претходних верзија *HTML*-а ова мета ознака за одређивање *UTF-8* сета карактера имала је сложеније атрибуте:

```
<meta http-equiv="Content-type" content="text/html;charset=UTF-8">
```

## 3.3.1. Параграфи и основни текст

Основна јединица за организацију текста у *HTML* документима је параграф (пасус). Ознака за додавање елемента *paragraph* је *p*. Ознаку параграфа је потребно затворити а њоме обухваћен садржај представља садржај параграфа. Унутар параграфа може се наћи једна или више реченица, слике, линкови и друго.

```
<p>Прва реченица, први параграф <a href="http://www.google.com">линк</a>.</p>
<p>Прва реченица, други параграф. <img src="uros-predic.jpg" /></p>
```

Прва реченица, први параграф линк.



Прва реченица, други параграф.

Табеле и листе (нумерисане или ненумерисане) не треба да се налазе унутар параграфа већ мимо њих:

```
<p>Прва реченица, први параграф. Друга реченица, први параграф.</p>
<table>...</table>
<p>Прва реченица, други параграф. Друга реченица, други параграф.</p>
<ul>...</ul>
<p>Прва реченица, трећи параграф. Друга реченица, трећи параграф.</p>
```

Прва реченица, први параграф. Друга реченица, први параграф.

| | |
|---|---|
| 1 | 2 |
| 3 | 4 |

Прва реченица, други параграф. Друга реченица, други параграф.

- Ставка 1
- Ставка 2
- Ставка 3

Прва реченица, трећи параграф. Друга реченица, трећи параграф.

При интерпретирању *HTML* кода табулатори се поистовећују са размацима а вишеструки размаци између речи се ингоришу, односно своде на највише један. У случају да је потребно у неком делу текста унети вишеструке размаке, за то се користи специјални симбол * * који се може поновити неограничен број пута.

```
<p>Поновљени знаци размака се                игноришу.</p>
<p>За вишеструке     размаке користи се посебна ознака.</p>
```

Поновљени знаци размака се игноришу.

За вишеструке размаке користи се посебна ознака.

Нови ред унутар изворног текста *HTML* докумената се приликом интерпретације игнорише, односно поистовећује са размација, а садржај више редова се приказује у истом реду:

```
<p>
   Реченица у првом реду.
   Реченица у другом реду.
</p>
```

Реченица у првом реду. Реченица у другом реду.

Први начин да се започне нови ред јесте прекид текућег параграфа и почетак новог. Други начин јесте да се унутар параграфа, на месту на коме се жели прећи у нови ред, дода ознака *<br />* која означава тзв. „меки" нови ред:

```
<p>Реченица у првом реду.<br /> Реченица у другом реду.</p>
```

Реченица у првом реду.
Реченица у другом реду.

У случају да је у тексту потребно нагласити одређене делове за то се користе ознаке *em* и *strong*. Ознака em представља блаже наглашавање и текст означен њоме се подразумевано приказује курзивом. Ознака strong представља посебно наглашавање и текст означен њоме се приказује подебљаним словима.

```
<p>Дођите у <em>четвртак</em>, у <strong>16:00</strong> часова.</p>
```

Дођите у *четвртак*, у **16:00** часова.

Приказ експонентног текста (суперскрипта, натписа) или индексног текста (субскрипта, потписа) у хипертекстуалним документима може се постићи на два начина: коришћењем специјалних јуникод знакова или коришћењем одговарајућих ознака. Ознака за експонентни текст је *sup* док је ознака за индексни текст *sub*:

```
x<sup>2</sup> = y<sub>1</sub> + y<sub>2</sub>
```

$$x^2 = y_1 + y_2$$

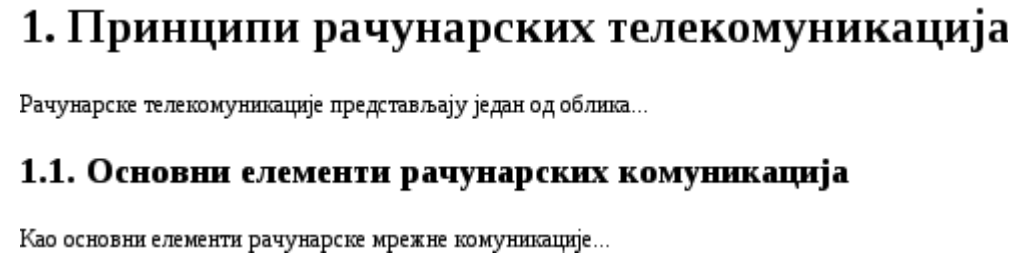
Коришћење ознака уместо јуникод карактера је препоручени начин због једноставнијег рада и претраге докумената.

### 3.3.2. Заглавља

*HTML* језик поседује предефинисане ознаке за шест нивоа заглавља, од *h1* до *h6*. У наставку је дат пример *HTML* кода који користи заглавља првог и другог нивоа

```
<h1>Принципи рачунарских телекомуникација</h1>
<p>Рачунарске телекомуникације представљају један од облика...</p>
<h2>Основни елементи рачунарских комуникација</h2>
<p>Као основни елементи рачунарске мрежне комуникације...</p>
```

а интерпретација наведеног кода дата је на следећој слици:




*Слика 1. Подразумеван приказ заглавља првог и другог нивоа*

Заглавља у *HTML* документима се подразумевано приказују подебљаним фонтом и већом величином слова, пропорционалном са значајем заглавља. Подразумевани изглед текста заглавља може се мењати коришћењем *CSS*-а, а са друге стране се исти изглед може постићи и код текста који се не налази у оквиру ознака за заглавље. Ипак, треба имати у виду да правилно коришћење ознака за заглавље документима даје и семантичку вредност, као и да се садржају у овим ознакама даје већи значај од стране Веб претраживача.

### 3.3.3. Додавање слика у документе

Постоје два основна начина за додавање графичких садржаја у хипертекстуалне документе: као додавање везе ка спољном ресурсу и као додавање графичких садржаја директно у хипертекстуални документ. Оба приступа имају своје предности и мане, с тим да је директно укључивање графичких садржаја у хипертекстуалне документе новији приступ који није подржан у старијим верзијама Веб браузера.

Додавање графичких садржаја кроз уметање везе ка спољном ресурсу је далеко

популарнији приступ који прати филозофију Веба као мреже увезаних ресурса. Правила која овде важе идентична су правилима везаним за повезивање докумената путем хипер-веза. Ознака за уметање графичког садржаја у хипертекстуалне документе је *img*, њен основни атрибут је *src* којим се дефинише путања на којој се налази графички садржај, обавезним се сматра атрибут alt (опис слике за браузере који немају могућност приказа графичких садржаја), а може имати и друге атрибуте као што су *width*, *height* и други:

```
<img src="uros-predic.jpg" alt="Урош Предић" width="150" /><br />
Косовка девојка
```



Косовка девојка

Додавање графичких садржаја директно у хипертекстуални документ представља бољи избор када је потребно целокупан садржај документа објединити у један фајл, односно искључити зависност од доступности спољних ресурса. Пошто се графички садржаји најчешће налазе у бинарном облику (*JPEG*, *PNG*, *GIF*...) њих је пре укључивања у хипертекстуалне документе потребно кодовати у текстуални облик, углавном коришћењем *Base64* шеме за кодовање:

```
<img src="...rkJggg==" />
```

Мане директног укључивања графичког садржаја у документ су повећање величине документа, немогућност коришћења кешираних резултата при накнадним позивима и немогућност коришћења истог графичког садржаја у више различитих докумената.

## 3.3.4. Унос специјалних симбола

Специјални симболи се у *HTML* документе могу уносити директно (уколико су ови кодовани као *UTF-8* или *UTF-16*) или као специјални ентитети, док је за резервисане симболе унос у виду специјалних ентитета једини начин. Специјални ентитети се у документима означавају својим симболичким или

нумеричким индексом коме претходи префикс & и суфикс *;*. Резервисани симболи, као што су знакови < и >, имају посебно значење и да би се приказали у документу морају се уносити као специјални ентитети. На пример, да би се у документу исписао следећи текст:

```
Ако је 3 > 2, онда да је и 2 < 3
```

потребно је ознаке за веће и мање унети као специјалне ентитете да би се исправно интерпретирали:

```
Ако је 3 &gt; 2, <strong>онда да је и</strong> 2 &lt; 3
```

И сам симбол *&* због своје улоге представља резервисани симбол па се за приказивање мора уносити као *&* или *&#38*;. Такође, и ћирилична слова се могу уносити као специјални ентитети, путем својих индекса у *unicode* табели. На пример, *HTML* код:

```
<p>Tekst &#x040B;&#x0438;&#x0440;&#x0438;&#x043B;&#x0438;&#x0446;&#x0430;</p>
```

ће након интерпретирања дати резултат:

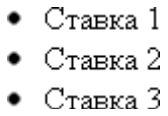
```
Tekst Ћирилица
```

Овакав начин кодовања слова, међутим, не препоручује се јер се непотребно повећава величита документа и време за његово интерпретирање, као и отежава сама претрага и јасноћа изворног кода.

## 3.3.5. Нумерисане и ненумерисане листе

У *HTML* документима могу се користити нумерисане и нумерисане листе. Ове листе треба да се налазе ван параграфа. За додавање ненумерисане листе (енгл. *Unordered List*) користи се ознака *ul*. Ставке у оквиру листе (енгл. *List Item*) се додају коришћењем ознаке *li*. Обе ознаке је потребно затворити.

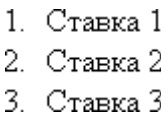
```
<ul>
   <li>Ставка 1</li>
   <li>Ставка 2</li>
   <li>Ставка 3</li>
</ul>
```




*Слика 1. Пример једноставне ненумерисане листе*

Са нумерисаним листама (енгл. *Ordered List*) се у *HTML* документима ради на исти начин као и са ненумерисаним листама, а ознака за њихово додавање је *ol*. Интерпретер сам води рачуна о нумерацији ставки, односно о њиховом правилном означавању бројевима:

```
<ol>
   <li>Ставка 1</li>
   <li>Ставка 2</li>
   <li>Ставка 3</li>
</ol>
```




*Слика 2. Пример једноставне нумерисане листе*

Листе, и нумерисане и ненумерисане, могу се угњеждавати у било којој комбинацији и било ком броју нивоа кроз додавање нове листе унутар ставке постојеће листе:

```
<ol>
   <li>
      Ставка 1
      <ol>
         <li>Ставка 1.1</li>
         <li>Ставка 1.2</li>
      </ol>
   </li>
   <li>Ставка 2</li>
</ol>
```

1. Ставка 1
    1. Ставка 1.1
    2. Ставка 1.2
2. Ставка 2

*Слика 3. Пример угњеждавања листи*

За дефинисање различитих параметара изгледа листи (ознака ненумерисаних листи, тип нумерације код нумерисаних листи, позиција ознака...) може се искористити технологија каскадних стилова приказа, *CSS*.

## 3.3.6. Табеле

Табеле представљају један од најчешће коришћених, али и често погрешно коришћених типова садржаја у *HTML* документима. Основна намена табела јесте представљање дводимензионалних скупова података, односно позиционирање података у одређеном дводимензионалном координатном систему. Језик за означавање хипер-текста има подршку за укључивање табеларних података у документе.

За рад са табелама у језику за означавање хипер-текста постоји више ознака али у основне ознаке спадају ознака за дефинисање табеле *table*, ознака за додавање новог реда у табели *tr* и ознака за дефинисање ћелије у реду. Све наведене ознаке је потребно затворити, односно њима уоквирити ознаке нижег реда.

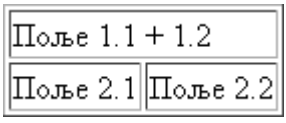
```
<table>
   <tr>
      <td>Поље 1.1</td>
      <td>Поље 1.2</td>
   </tr>
   <tr>
      <td>Поље 2.1</td>
      <td>Поље 2.2</td>
   </tr>
</table>
```

| Поље 1.1 | Поље 1.2 |
|---|---|
| Поље 2.1 | Поље 2.2 |

*Слика 1. Основна табела од два реда и две колоне*

Подразумева се постојање истог броја ћелија у сваком реду табеле. У случају да је потребно простирање једне ћелије проширити на више наредних хоризонталних поља за ћелије, за то се користи атрибут  чија целобројна вредност означава број места која ће та ћелија заузети.

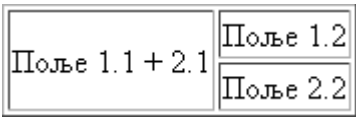
```
<table border="1">
   <tr>
      <td colspan="2">Поље 1.1 + 1.2</td>
   </tr>
   <tr>
      <td>Поље 2.1</td>
      <td>Поље 2.2</td>
   </tr>
</table>
```




*Слика 2. Хоризонтално спајање поља у табели*

У случају да је потребно проширити простирање ћелије на више места вертикално, за то се користи атрибут *rowspan*, такође са целобројном вредношћу:

```
<table border="1">
   <tr>
      <td rowspan="2">Поље 1.1 + 2.1</td>
      <td>Поље 1.2</td>
   </tr>
   <tr>
      <td>Поље 2.2</td>
   </tr>
</table>
```




*Слика 3. Вертикално спајање поља у табели*

У најзначајније атрибуте ознаке *table* спадају атрибут *border* који одређује дебљину оквира ћелија, атрибут *align* који одређује лево, централно или десно позиционирање табеле и атрибут *width* који одређује релативну или апсолутну ширину табеле. Постоји велики број атрибута којим се може дефинисати начин приказа табела али је због потреба одржавања садржаја у ту сврху препоручено коришћење *CSS* технологије.

Због могућности употребе табела за позиционирање садржаја у документу оне се често користе и у ту сврху. Међутим, оваква употреба табела није наменска и доводи до два проблема:

1. садржај се неправилно интерпретира од стране програма које користе слабовиде особе за приступ Вебу и
2. претраживачи садржаја на Вебу (нпр. Гугл) ниже рангирају странице са ненаменском употребом табела.

Из наведених разлога табеле у *HTML* документима не треба користити мимо њихове намене - за позиционирање садржаја у документу - већ у ту сврху треба користити наменску технологију: каскадне стилове.

Постоји велики број атрибута за одређивање приказа табела и садржаја у њима - на пример дебљине оквира, удаљености између поља, удаљености између оквира и садржаја, и т.д. Међутим, због једноставнијег управљања приказом већег броја докумената у ту сврху такође треба користити каскадне стилове.

## 3.4. Формулари

Еволуцијом Веба од система за повезивање статичних докумената до платформе за развој дистрибуираног софтвера јавила се потреба за већом интеракцијом између садржаја и корисника. Основни начин у Веб апликацијама за преузимање података од корисника јесу формулари. Формулари, како им и сам назив каже, имају задатак да ограниче слободу уноса вредности, односно да вредности које корисник уноси **уобличе**. *HTML* језик поседује подршку за уграђивање формулара у документе.

Основне јединице за преузимање података од корисника унутар формулара називају се контролама. Постоји више типова контрола, прилагођених узимању података у различитим облицима - контроле за унос текста у једном или више редова, контроле за избор једне или више од предефинисаних вредности, контроле за слање и ресетовање уноса, и т.д.

Све контроле унутар формулара имају своју тренутну вредност. При учитавању формулара вредности свих контрола су постављене на предефинисане, а затим се у току интеракције са корисником њихова вредност мења. Корисници вредност контрола могу мењати директно (нпр. уносом новог садржаја у поље за унос текста) или путем скриптова.

Свака контрола унутар формулара подразумевано поседује свој назив. Овај назив је јединствен унутар формулара у коме се контрола налази али не мора бити јединствен за цео документ с обзиром да се у једном документу може наћи више одвојених формулара. Изузетак од овог правила чине радио-дугмад код којих је за једну групу дугмади назив исти али се вредности разликују. Назив контрола се користи након слања уноса у формулар, односно у процесу његове обраде, за приступање достављеним вредностима.

Све контроле унутар формулара се подразумевано сматрају омогућеним за интеракцију са корисником. Међутим, у случају потребе за тим, ниво интеракције са контролом се може ограничити на два начина: онемогућавањем измене тренутне вредности и потпуним онемогућавањем контроле. Оба начина остављају контролу приказаном и онемогућавају измену њене вредности, с тим да се потпуним онемогућавањем контроле њој онемогућава добијање фокуса, њена вредност се не шаље на обраду и није могуће преузимање њене вредности од корисника.

Употреба формулара подразумева њихову обраду на страни сервера, коришћењем програмских језика као што су *PHP*, *JSP*, *ASP* и други. Обрада послатих података најчешће укључује приступ подацима, проверу исправности уноса, унос у базу података и сл. Алтернативно, унос се може обрадити и на страни клијента, најчешће коришћењем *JavaScript* програмског језика.

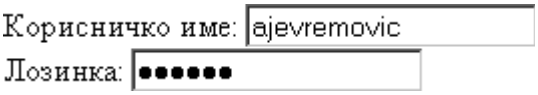
## 3.4.1. Поља за унос текста од стране корисника

У формуларима *HTML* докумената могу се користити три основне врсте поља за унос текста од стране корисника:

1. унос текста у једном реду
2. унос текста у више редова
3. унос сакривеног текста (лозинки)

За унос текста у једном реду, као и за унос скривеног текста, користи се иста ознака, *input,* а њен тип се одређује задавањем вредности *text* или *password* за атрибут *type*.

```
Корисничко име: <input type="text" name="korisnik" /><br />
Лозинка: <input type="password" name="lozinka" />
```

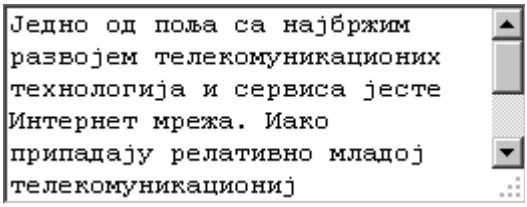



*Слика 1. Коришћење поља за унос текста у једном реду*

У случају да је потребно поставити почетну вредност у ова два типа поља, то се може учинити уносом жељеног садржаја као вредност атрибута *value*. За ограничење у погледа броја унетих знакова у поље користи се атрибут *maxlength* а забрана измене вредности поља постиже се укључивањем атрибута *readonly*.

Поље за унос текста у више редова додаје се коришћењем ознаке *textarea*. Једини неопходни атрибут ове ознаке јесте њен назив - *name*, с тим да се за потпуно исправан код захтева и одређивање броја колона и редова поља, односно постављање вредности атрибута *cols* и *rows*:

```
<textarea name="sazetak" cols="30" rows="5"></textarea>
```




*Слика 2. Коришћење поља за унос текста у више редова*

Ознаку *textarea* је потребно затворити а садржај означен њоме се приказује као почетна вредност при учитавању формулара.

## 3.4.2. Контроле за избор предефинисаних вредности

Осим слободног уноса садржаја од стране корисника у формуларе, у пракси често постоји потреба да корисник изабере једну или више од понуђених, предефинисаних вредности. Основне контроле за избор предефинисаних вредности у *HTML* формуларима су радио-дугмад, поља за штиклирање и падајуће листе.

Уколико је у формулару, на пример, потребно да корисник означи пол коме

припада, то се може постићи и коришћењем поља за слободан унос текста. Такав приступ, међутим, има два главна недостатка: кориснику се одузима више времена а одговори нису стандардизовани те је потребно више времена за њихову обраду.

Радио-дугмад су основни тип контроле у формуларима када је потребно да корисник изабере само једну од више понуђених вредности. За додавање радио дугмади користи се ознака *input* код које је вредност атрибута *type* постављена на *radio*. Потребно је да све ознаке свих понуђених вредности из исте групе деле исту вредност атрибута *name*. Код за постизање горе наведене функционалности коришћењем радио-дугмади је:

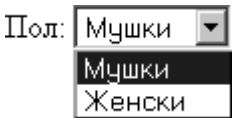
```
Пол: Мушки <input type="radio" name="pol" value="m" />
Женски <input type="radio" name="pol" value="z" />
```




*Слика 1. Коришћење радио-дугмади у формуларима*

Алтернатива коришћењу радио-дугмади у формуларима за избор једне од више понуђених вредности јесу такозване падајуће листе. Ознака за додавање падајућих листи је *select* којој се уједно додељује и атрибут *name*. Унутар листе додају се понуђене вредности путем ознаке *option*. Свакој ознаци се додељује одговарајућа вредност а њоме означен текст ће се исписати као вредност коју корисник види:

```
Пол:
<select name="pol">
   <option value="m">Мушки</option>
   <option value="z">Женски</option>
</select>
```




*Слика 2. Коришћење падајућих листи у формуларима*

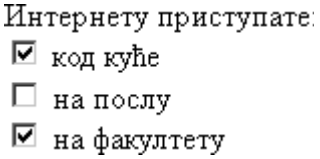
Предност коришћења падајућих листи наспрам радио-дугмади јесте потреба за мањим простором за њихово приказивање. Са друге стране, предност коришћења радио-дугмади огледа се у могућности корисника да одједном види

све опције, као и да путем само једног клика мишем означи жељену опцију (код падајућих листи је потребан додатни клик за приказивање опција).

Осим могућности да изабере једну од понуђених вредности, кориснику је често потребно понудити могућност да изабере и више њих. На пример, на питање „Где све користе Интернет?" корисници могу изабрати више понуђених вредности - код куће, на послу, на факултету, и т.д.

Основна контрола у *HTML* формуларима за избор вишеструких понуђених вредности јесу поља за означавање (енгл. *checkbox*). Она се у формуларе додају путем ознаке *input* код које је тип постављен на *checkbox*:

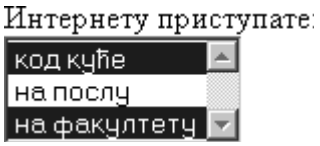
```
Интернету приступате:<br />
<input type="checkbox" name="ikuca" /> код куће <br />
<input type="checkbox" name="iposao" /> на послу <br />
<input type="checkbox" name="ifakultet" /> на факултету
```




*Слика 3. Коришћење поља за означавање у формуларима*

Алтернативу коришћењу поља за означавање представља коришћење листи са понуђеним избором које се додају путем ознаке *select* код које је укључен атрибут *multiple*. Број иницијално приказаних ставки се одређује преко атрибута *size*:

```
Интернету приступате:<br />
<select name="internet" multiple size="3">
   <option value="kuca">код куће</option>
   <option value="posao">на послу</option>
   <option value="fakultet">на факултету</option>
</select>
```




*Слика 4. Коришћење листи у формуларима*

Предност коришћења листи наспрам поља за означавање јесте могућност приказивања већег броја опција на мањем простору (клизачи садржаја се активирају уколико постоји више опција него што је приказано). Међутим, битан недостатак оваквог приступа је потреба за држањем притиснутог тастера *CTRL* код избора више опција, тако да се он не препоручује код израде формулара које не користе лица са добрим познавањем рада на рачунару.

## 3.4.3. Дугмад

Дугмад представљају основну контролу у документима и формуларима за покретање одређене акције, првенствено за слање унетог садржаја. Основни догађај за иницирање додељене акције дугмету јесте клик левим тастером миша на њега. Ознака *button* захтева затварање а уметнути садржај представља натпис на дугмету.

Ознака за уметање дугмади јесте *button* а њихов тип се одређује путем атрибута *type*. Атрибут *type* је обавезан пошто ће у случају његовог изостављања различити Веб браузери дугмету самостално доделити различите улоге. *HTML* језик дефинише три типа дугмади, у зависности од њихове намене:

1. дугмад за слање садржаја унетог у формулар
2. дугмад за ресетовање садржаја формулара на почетно стање
3. дугмад без подразумеване акције (користе се *JavaScript* језиком)

Дугмад за слање унетог садржаја у формулар упућују Веб браузер корисника да кориснички унос запакује у одговарајући тип захтева (најчешће *get* или *post*) и на тај начин проследи серверу на обраду.

```
<button type="submit">Слање унетих података</button>
```



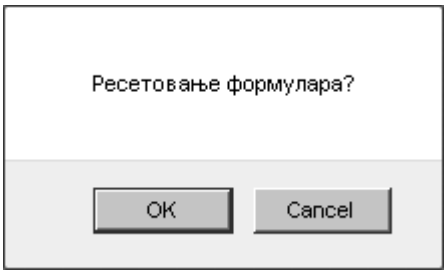

*Слика 1. Изглед дугмета за слање садржаја формулара*

Дугмад за ресетовање садржаја формулара се користе за враћање формулара на почетно стање, на пример у ситуацијама када корисник схвати да је формулар попунио погрешним вредностим и жели да започне поново, без освежавања целе странице. Ипак, с обзиром на то да је мало вероватно да ће корисник погрешно попунити цео формулар, као и на то да активирање дугмета за ресетовање поништава кориснички унос без додатне потврде и могућности опозива, овакву дугмад треба веома опрезно користити у дизајнирању

формулара.

У пракси се често може срести постављање дугмета за ресетовање на дно формулара, одмах поред дугмета за његово слање. Таква организација често доводи до фрустрације корисника који након дужег попуњавања формулара уместо слања грешком губи сав садржај, без могућности његовог повратка. Из тог разлога је пожељно пре ресетовања уноса затражити додатну потврду од корисника, на пример коришћењем *JavaScript* функције *confirm*:

```
<button type="reset" onclick="if (!confirm('Ресетовање формулара?')) return
false;">Ресетовање</button>
```




*Слика 2. Захтев за потврду поништавања корисничког уноса*

Последњи тип дугмади у HTML формуларима јесу дугмад без подразумеване акције, односно дугмад код којих атрибут *type* има вредност *button*. Оваква дугмад немају предодређену намену већ се повезују са функцијама *JavaScript* језика, најчешће за једноструки клик левог тастера миша путем атрибута *onclick*. На пример, уколико корисницима желимо да омогућимо проверу да ли је у формулар унета валидна адреса за електронску пошту, пре слања целокупног формулара, то можемо урадити додавањем одговарајућег дугмета:

```
<button type="button" onclick="EmailCheck();">Провера адресе</button>
```

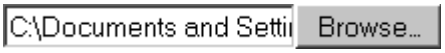
Једноструким кликом левог тастера миша на наведено дугме биће позвана функција *EmailCheck* реализована у *JavaScript* програмском језику.

## 3.4.4. Слање фајлова путем формулара

У пракси је често потребно да корисници приликом слања података путем Веб формулара пошаљу и фајлове који се налазе на њиховим рачунарима. Контрола за ову намену се у формуларе укључује коришћењем ознаке *input* чији је атрибут *type* постављен на вредност *file*. Ширина поља за испис путање фајла

може се контролисати путем атрибута *size* чију вредност чини жељени број слова.

```
<form enctype="multipart/form-data" method="post">
   <input type="file" name="fajl" size="25" />
   ...
</form>
```




*Слика 1. Изглед контроле за слање фајлова путем формулара*

Важно је уочити присуство атрибута *enctype* и његове вредности *multipart/form-data* у ознаци самог формулара. Ова вредност атрибута код саме ознаке формулара је неопходна за успешно слање фајлова серверу. Фајлове које је корисник послао потребно је прихватити на одговарајући начин на страни сервера коришћењем програмских језика као што су *PHP*, *JSP* и други.

## 3.4.5. Означавање и груписање контрола

За правилно коришћење формулара од стране корисника, у општем случају, контроле унутар њих је потребно означити, односно описати вредност коју је потребно да корисник унесе. У ту сврху се најчешће користе текстуални описи поред самих контрола, а понекад и унутар њих.

Најједноставнији начин за постављање текста за опис контроле у формулару јесте унос чистог текста поред контроле, без икакве ознаке. Међутим, наменска ознака за опис контроле у формуларима јесте *label*. Ова ознака има захтевани атрибут *for* чија вредност треба да буде иста као и вредност атрибута *id* контроле на коју се опис односи. Предност коришћења ове ознаке (наспрам обичног текста) је аутоматско постављање фокуса на контролу на коју се опис односи у случају клика мишем на њега.

```
<label for="korisnickoime">Корисничко име:</label>
<input type="text" name="korisnik" id="korisnickoime" />
```




*Слика 1. Пример означавања контроле путем ознаке label*

Добру праксу при дизајнирању *HTML* формулара чини и **груписање** сродних

поља и контрола. На пример, при дизајнирању формулара за пријављивање за упис на основне студије корисницима може бити значајно лакше сналажење уколико се визуелно јасно раздвоје и групишу поља која се односе на адресу становања и поља која се односе на претходно образовање.

```
<fieldset>
   <legend>Основна школа</legend>
   Назив: <input type="text" name="os_naziv" />
   <br />
   Место: <input type="text" name="os_mesto" />
   <br />
   Разред: <input type="text" name="os_razred" />
</fieldset>
```




*Слика 2. Пример груписања ознака у формуларима*

Груписање контрола коришћењем *fieldset* ознаке уједно омогућава и уштеду простора за приказ јер у случају њеног коришћења није потребно у опису сваке контроле појединачно наводити којој групи она припада.

## 3.4.6. Слање садржаја формулара

Након интеракције корисника са формуларом, осим у случајевима када се формулар комплетно обрађује функцијама *JavaScript* језика, подразумева се слање корисничког уноса серверу. Слање се иницира активирањем дугмета за слање од стране корисника или позивом одговарајуће функције *Javascript* језика. Кораци у слању корисничког уноса у формулар серверу су следећи:

1. Утврђује се које контроле унутар формулара треба укључити вредности у податке који се шаљу серверу. У ове контроле подразумевано спадају све контроле осим контрола које су намерно искључене (атрибут *disabled*) и неактивирана дугмад за означавање и радио-дугмад.
2. На основу утврђене листе контрола које се укључују у слање формира се

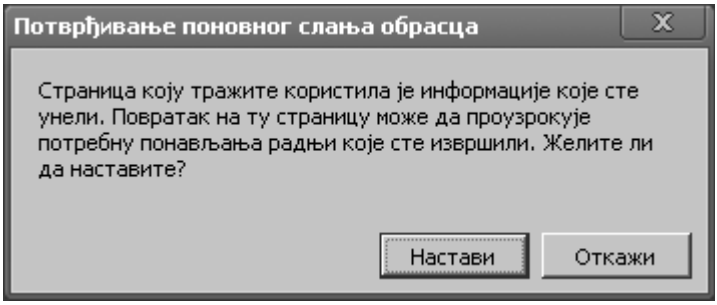
листа парова назив-вредност.

3. Формирана листа назива-вредности се кодује у складу са подразумеваним или задатим типом кодовања (атрибут *enctype* ознаке формулара).
4. На одговарајући начин кодована листа назива контрола и њихових вредности се прослеђује серверу на адресу задату у атрибуту *action* ознаке формулара, коришћењем одговарајућег метода *HTTP* протокола.

Протокол за пренос хипер-текста поседује два основна метода за прослеђивање садржаја формулара - *GET* и *POST*. У већини браузера метод *GET* је подразумеван уколико се другачије не одреди атрибутом *method* ознаке формулара. Код овог метода сви парови из листе вредности се додају *URL*-у дефинисаним атрибутом *action* формулара, од њега одвојени знаком питања а међусобно одвојени знаком амперсанд. На пример, *URL* на коме се обрађује садржај формулара може изгледати овако:

```
http://www.racunarskemreze.com/obrada.php?ime=Петар&prezime=Петровић
```

Метод *GET* треба користити у случајевима када није опасно освежавање странице на којој је формулар обрађен, када није потребно сакрити кориснички унос или када је потребно кориснику оставити могућност да копирањем *URL*-а опет приступи идентичној страници.

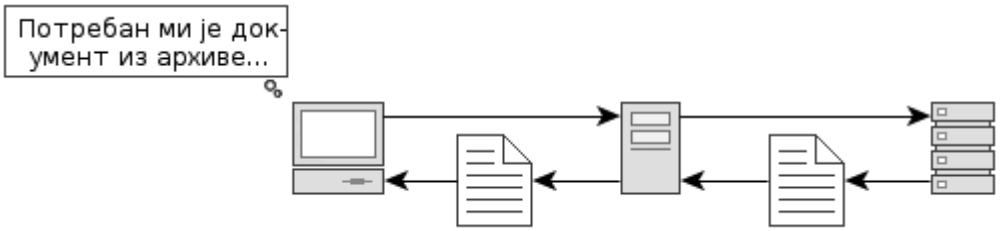



*Слика 1. Дијалог за потврду освежавања странице за обраду формулара*

Насупрот методу *GET* стоји метод *POST* који не смешта садржај формулара у *URL* на коме се врши обрада већ га уграђује у одговарајуће поље протокола за пренос хипер-текста. На тај начин се кориснички уноси не чувају у забележеним адресама на корисничком агенту или прокси серверу. Додатно, у случају покушаја корисника да освежи страницу на којој су обрађени подаци из формулара, кориснички агент ће захтевати додатну потврду ове акције (слика 1).

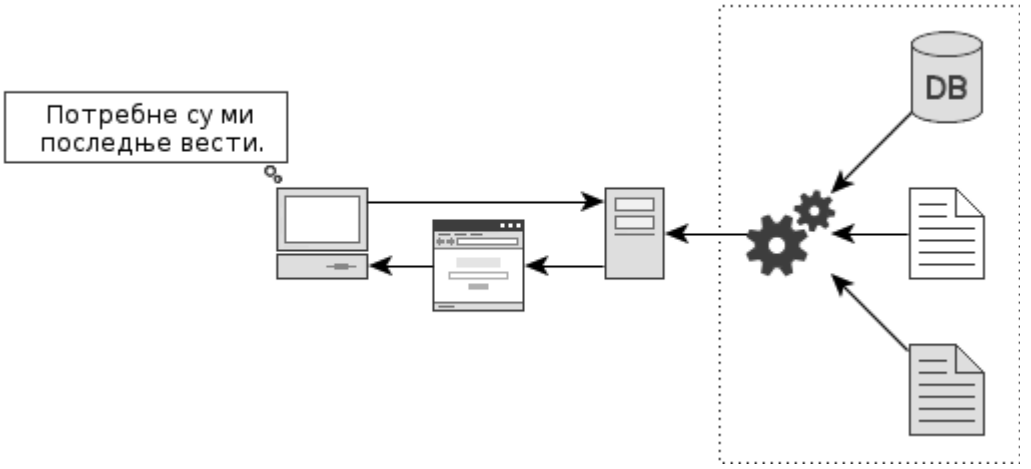
# 4. Програмирање на страни сервера

У првим данима Веба улога сервера била је да са спољне меморије учита тражени статични садржај (*HTML* документ, слику и друго) и да га у оквиру *HTTP* протокола пошаље назад клијенту. Такав принцип рада нуди високе перформансе и безбедност, али су његове могућности веома ограничене и своде се на готово једносмерну комуникацију, односно „емитовање садржаја“.




*Слика 1. Принцип рада Веб сервера са статичним документима*

Динамичко креирање садржаја, са друге стране, приступ је код кога клијент серверу шаље стандардан захтев за одређеним садржајем, а тај захтев се обрађује одговарајућом програмском процедуром на серверу. Резултат те обраде је тражени садржај, креиран у складу са тренутном ситуацијом и специфичностима дефинисаним у захтеву. Овакав приступ знатно умањује перформансе и ствара могућност за безбедносне проблеме, али нуди готово неограничене могућности за развој мрежних апликација.




*Слика 2. Принцип рада Веб сервера са динамички креираним садржајима*

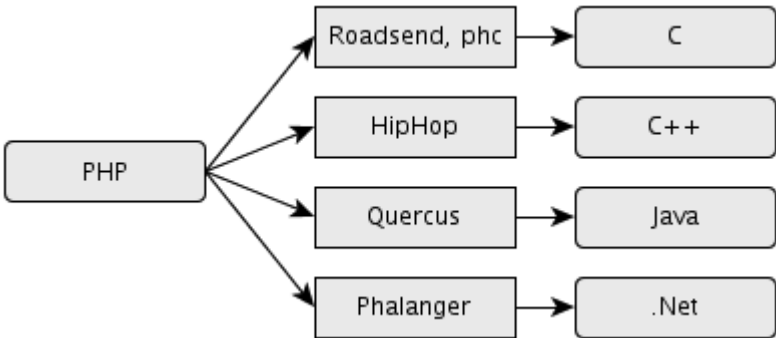
Зачеци динамичког креирања садржаја, односно програмирања на страни Веб сервера, налазе се код коришћења *JavaScript* кодова на страни сервера (тзв. *server side scripting*). Касније је омогућено коришћење *C*, *Perl* и *shell* скриптова кроз *CGI* интерфејс.

Најпопуларнија решења за програмирање на страни сервера данас су *PHP - PHP Hypertext Preprocessor*, *ASP - Active Server Pages* и *JSP - JavaServer Pages*. *ASP* је комерцијална технологија компаније Мајкрософт и ограничена је на употребу само на оперативним системима те компаније. *JSP* технологија је значајна јер омогућава употребу већ постојећих класа развијених у *Java* програмском језику а слободно је доступна на различитим оперативним системима. Међутим, далеко најпопуларније решење за развој Веб апликација данас је *PHP* који је доступан на различитим оперативним системима и углавном је део стандардне понуде Веб хостинг провајдера.

## 4.1. Програмски језик PHP

Сам *PHP* програмски језик се појавио 1995. године а иницијално је развијен у *Perl* програмском језику и коришћен је кроз *CGI* интерфејс. Његов развој је покренуо Расмус Лердорф, тада 27-годишњи програмер данског порекла, а касније је настављен од стране Ендија Гутманса и Зива Шураског, израелских програмера, у оквиру компаније Зенд Технолоџис.

Током свог животног века је више пута преписиван од нуле а данас се развија коришћењем *C* програмског језика. У четвртој верзији је добио основну подршку за објектно-оријентисано програмирање а од пете верзије је та подршка слична као код *Java* програмског језика.




*Слика 1. Постоји више решења за превођење PHP кода на друге језике*

До четврте верзије *PHP* је био интерпретирани програмски језик. Међутим,

2000. године је уведен Зенд Енџин који је извршавање *PHP* кодова поделио у неколико фаза, од којих је једна превођење изворног кода у опкод који је сличан бајткоду *Java* програмског језика. Овиме је уведена могућност кеширања опкода чиме је отворена могућност за знатно подизање перформанси при извршавању.

С времена на време се појављују критике *PHP* програмског језика. У нападима на њега често се користе аргументи који су по озбиљности и информисаности критичара блиски коментару који су дали водитељи извесне бразилске ТВ станице да „уколико се Веб адреса завршава са *.php* на њој се обично налази вирус"[9]. Критичари често нападају динамичност типова променљивих код *PHP*-а што, међутим, проблем може представљати само осредњим програмерима, док је за почетнике и експерте овакав третман променљивих незаменљив. Укратко, за потврду значаја и квалитета овог програмског језика довољно говори податак да он покреће преко 80% динамичких Веб сајтова на Интернету[10] као и да се он користи за неке од највећих платформи као што су Фејсбук[11] и Википедија[12].

### 4.1.1. Извршно окружење

Примарни начин извршавања програма написаних у *PHP* програмском језику је у оквиру обраде захтева ка Веб серверу. Овакав начин извршавања подразумева постојање *PHP* интерпретера/извршиоца унутар Веб сервера, коме се прослеђују сви учитани фајлови са спољне меморије који имају екстензију *.php*. Из ових фајлова се узимају делови који се налазе између ознака:

```
<?php
```

и

```
?>
```

и код унутар њих се интерпретира и извршава. Резултати извршавања се укључују на места на којима се налазио извршени блок изворног кода или се смештају у заглавље *HTTP* пакета уколико је тако захтевано.

Осим у оквиру обраде захтева ка Веб серверу *PHP* програми се могу извршавати

9 http://blog.doh.ms/2013/01/04/we-are-all-php-evangelists/
10 http://w3techs.com/technologies/details/pl-php/all/all
11 http://arstechnica.com/business/2012/04/exclusive-a-behind-the-scenes-look-at-facebook-release-engineering/
12 http://en.wikipedia.org/wiki/Wikipedia:FAQ/Technical

и самостално - директним повезивањем путем алфа-нумеричке љуске оперативног система - или у оквиру графичких десктоп апликација. Међутим, ови приступи су далеко ређе коришћени.

### 4.1.2. Конфигурациони фајлови

Два основна фајла којим се укључује и подешава коришћење *PHP*-а на Веб серверу јесу конфигурациони фајл самог Веб сервера (где се укључује коришћење самог *PHP*-а) и конфигурацини фајл *PHP*-а којим се дефинише његово понашање. У наставку је дат садржај конфигурационог фајла *mod_php.conf* као дела конфигурације Апач Веб сервера којим се укључује подршка за PHP:

```
# mod_php - PHP Hypertext Preprocessor module

# Load the PHP module:
LoadModule php5_module lib/httpd/modules/libphp5.so

# Tell Apache to feed all *.php files through PHP.
<FilesMatch \.php$>
    SetHandler application/x-httpd-php
</FilesMatch>
```

У оквиру фајла *php.ini* дефинишу се параметри који одређују како ће се *PHP* понашати - колико ће меморије и времена бити доступно за извршавање, која проширења су укључена и слично. Ови параметри се дефинишу у формату:

```
параметар = вредност
```

У наставку је дат пример типичних конфигурационих параметара *PHP*-а:

```
max_execution_time = 180
memory_limit = 128M
display_errors = Off
file_uploads = On
extension=mysqli.so
extension=zip.so
```

```
date.timezone = "Europe/Belgrade"
...
```

Треба имати у виду да је исправно подешен *PHP* услов за несметано и исправно извршавање Веб апликација на серверу, као и да пропусти у конфигурацији могу отворити безбедносне проблеме. Такође, постоје значајне разлике у конфигурацији између развојних и продукционих окружења, као и између различитих продукционих окружења.

### 4.1.3. Управљање грешкама

Три основна типа грешака код извршавања *PHP* скриптова су опомена, упозорење и фатална грешка. Опомене (енгл. *notice*) су мање грешке које могу довести до проблема у извршавању али је могуће и да је такав приступ намерно коришћен од стране програмера (на пример, покушај да се приступи непостојећем атрибуту променљиве која није објекат).

```
[Wed Jul 17 22:19:03.626987 2013] [:error] [pid 2494:tid 2961177408] [client
127.0.0.1:38383] PHP Notice:  Trying to get property of non-object in
/home/ajevremovic/public_html/Lab/knjiga-it/db/izmena.php on line 38,
referer: http://localhost/Lab/knjiga-it/db/select.php
```

Упозорења (енгл. *warning*) су озбиљније грешке, за које је тешко претпоставити да спадају у планирано понашање апликације, али је и након њих могуће наставити извршавање скрипта. У наставку је дат пример упозорења везаног за покушај додавања заглавља у *HTTP* пакет у тренутку када је већ започето слање његовог садржаја:

```
[Fri Dec 07 23:40:23.249784 2012] [:error] [pid 3213:tid 2844785472] [client
127.0.0.1:54438] PHP Warning:  Cannot modify header information - headers
already sent by (output started at /ws/s1.php:468) in /ws.s1.php on line 467
```

Фаталне грешке (енгл. *fatal error*) су најозбиљнији тип грешака после чијег јављања се даље извршавање прекида јер се сматра да није могуће исправно наставити и добити очекиване резултате. У наставку је дат пример поруке о фаталној грешци, насталој као последица позива функције која није дефинисана:

```
[Sun Dec 09 22:30:53.911059 2012] [:error] [pid 2470:tid 2995780416] [client
127.0.0.1:58113] PHP Fatal error:  Call to undefined function
lzf_decompress() in /home/ajevremovic/public_html/kompresija.php on line 25
```

Информације о грешкама се могу приказати директно или преусмерити у фајл са дневником догађаја. У развојним окружењима је директно приказивање бољи избор јер програмеру одмах даје информацију о грешкама које се јављају, док приказивање грешака корисницима у продукционом окружењу може довести до безбедносних проблема. Да ли ће се грешке приказивати или складиштити у фајл одређује се следећим директивама у конфигурационом фајлу:

```
display_errors = On / Off
log_errors = On / Off
```

Уколико је укључено чување информација о грешкама у фајлу са дневником догађаја, подразумевано ће се користити фајл *error_log* самог Веб сервера.

Један начин да се информација о насталој грешци при извршавању функције или исказа не прикаже нити забележи у фајлу јесте коришћење оператора „@“:

```
$x = 1;
echo @$x->Ime;
```

Међутим, овакав начин одбацивања порука о грешкама треба веома опрезно користити јер он може довести до драстично отежаног проналажења и отклањања грешака у сложеним апликацијама.

## 4.2. Основе програмирања

Синтакса програмског језика *PHP* је слична језицима као што су *C*, *C++*, *Java* и *Perl.* У одмах приметне специфичности *PHP*-а спадају флексибилни типови променљивих, односно могућност промене типа током извршавања и елиминисање потребе да се он дефинише унапред, означавање променљивих знаком за долар, могућност и процедуралног и објектно оријентисаног програмирања, и недостатак иницијалне функције/класе.

### 4.2.1. Променљиве и константе

У основне типове променљивих у *PHP* програмском језику спадају:

- логичка вредност (енгл. *boolean*),
- целобројна (енгл. *integer*) и децимална (енгл. *float*) вредност и
- низ симбола (енгл. *string*)

У сложене типове променљивих у PHP програмском језику спадају:

- низови елемената (енгл. *array*) и
- објекти као инстанце класа (енгл. *object*)

Посебан тип променљивих у *PHP* програмском језику су тзв. ресурси (енгл. *resource*) у којима се чувају успостављене везе ка бази података, фајловима и слично. Додатно, променљива не мора имати вредност и тип уколико је у „нултом“ стању (енгл. *null*).

Тип и вредност променљиве се током извршавања могу више пута мењати:

```
$x = 1;
$x = 2.2;
$x = "текст";
$x = array(1,2,3,4);
$x = new stdClass();
$x = null;
```

При додељивању вредности променљивој може се експлицитно задати ког типа она треба да буде:

```
$x = 142.857;
$y = (int) $x;
```

При том, резултат извршавања наведеног скрипта ће променљивој *y* доделити вредност 142, односно почетни део задате вредности који се могао интерпретирати као цео број, док је за заокруживање потребно користити наменску функцију.

Услед флексибилности рада са променљивима у *PHP*-у се могу упоређивати променљиве различитих типова. При том се за тзв. „меко“ поређење, које узима у обзир само вредност, користи оператор „==“, док се за тзв. „тврдо“ поређење, које у обзир узима и тип променљивих, користи оператор „===“:

```
$x = (int) 123;
$y = (float) 123;
if ($x == $y) echo 'Иста вредност'; // Да
if ($x === $y) echo 'Иста вредност и тип'; // Не
```

Да би се приступило одређеној променљивој њен назив се може користити као вредност друге променљиве (тзв. променљиве променљиве):

```
$ime = "Петар";
$pp = "ime";
echo $$pp; // Исписује „Петар“
```

При давању назива променљивама треба бити опрезан код коришћења знака „_“ као почетка имена јер њиме почињу и називи системских, предефинисаних променљивих као што су *$_GET, $_POST, $_REQUEST, $_SERVER, $_SESSION* и сличне.

Насупрот променљивама, тип и вредност једном дефинисане константе остају исти до краја извршавања програма, а за дефинисање константи користи се функција *define*:

```
define("PI", 3.14);
```

У називима променљивих и константи прави се разлика између великих и малих слова.

### 4.2.1.1. Видљивост променљивих

Под видљивошћу променљивих подразумева се њихова област важења, односно област програма у којој се променљива може користити. Област важења променљиве директно зависи од места на којем је променљива иницијално декларисана. На пример, променљива која је дефинисана ван било које функције сматра се **глобалном променљивом**:

```
<?php
$x = 3;
```

и њој се може приступити из глобалног простора, односно ван функција. Са друге стране, променљиве декларисане у оквиру функција се називају **локалним променљивама** и могу се користити само унутар те функције:

```
<?php
$x = 3;
function IspisBroja()
```

```
{
   $x = 4;
   echo $x;
}
IspisBroja(); // Исписаће локалну вредност 4
echo $x; // Исписаће глобалну вредност 3
```

Уколико у некој функцији желимо да користимо глобалну променљиву, у ту сврху ћемо искористити реч *global*:

```
<?php
$x = 3;
function IspisBroja()
{
   global $x;
   echo $x;
}
IspisBroja(); // Исписаће глобалну вредност 3
```

Системске, предефинисане променљиве као што су *$_COOKIE, $_ENV, $_FILES, _GET, $_POST, $_REQUEST, $_SERVER* и *$_SESSION* се сматрају **суперглобалним** и могу се користити у било којој области програма.

## 4.2.2. Употреба наводника

У *PHP*-у се текстуални подаци - низови знакова (енгл. *string*) - дефинишу унутар једноструких или двоструких знакова навода. Употреба једноструких и двоструких знакова навода је готово идентична, али одређене разлике ипак постоје. Главна разлика се налази у употреби променљивих унутар знакова навода. На пример, код:

```
$procenat = 80;
echo 'PHP се користи на преко $procenat процената сајтова';
```

исписаће:

```
PHP се користи на преко $procenat процената сајтова
```

док ће код:

```
$procenat = 80;
echo "PHP се користи на преко $procenat процената сајтова";
```

исписати:

```
PHP се користи на преко 80 процената сајтова
```

Дакле, текст унутар једноструких наводника се не интерпретира, док се у тексту унутар двоструких наводника проналазе ознаке променљивих и на њихова места се постављају одговарајуће вредности. То даље значи да ће се код са једноструким наводницима брже интерпретирати од еквивалентног кода који користи двоструке наводнике.

### 4.2.3. Низови и матрице

*PHP* подржава нумерички и асоцијативно индексоване једнодимензионалне и вишедимензионалне низове, односно матрице. То значи да се као индекс може користити редни број елемента или симболички низ карактера, као и да број индекса није ограничен. Тип вредности елемената низа није ограничен и може се разликовати за различите чланове. За декларисање новог низа користи се функција *array*.

```
$niz = array();
$niz[0] = 1;
$niz[1] = 0.5;
$niz[2] = array();
$niz[2]['студент'] = new stdClass();
$niz[2]['студент']->Ime = 'Петар';
```

*Листинг 1. Формирање низа са различитим типовима индекса и чланова*

На примеру датом у листингу 1. приказано је креирање низа чије елементе чине нумерички индексовани цео број, децималан број и низ са симболички индексованим чланом типа објекат. Садржај ове променљиве, добијен коришћењем функције *print_r* дат је у листингу 2:

```
Array
```

```
(
    [0] => 1
    [1] => 0.5
    [2] => Array
        (
            [студент] => stdClass Object
                (
                    [Ime] => Петар
                )
        )
)
```

*Листинг 2. Садржај променљиве формиране кодом из претходног листинга*

Од верзије 5.4 *PHP* програмског језика омогућена је једноставнија синтакса за брзо дефинисање низова, која не подразумева коришћење функције *array* већ само задавање вредности у оквиру угластих заграда. На пример, коришћењем ове синтаксе дефинисање аутоматски нумерички индексованог низа би изгледало:

```
$boje = ['црвена', 'плава', 'бела'];
```

Код коришћења поменуте скраћене синтаксе формирање низова са симболичким индексима је приказано на следећем листингу:

```
$godista = ['Петар'=>1971, 'Митар'=>1982, 'Лазар'=>1993];
```

*PHP* подржава формирање *N*-димензионалних матрица, односно матрица код којих је број димензија ограничен само архитектуром и меморијом рачунара на коме се програм извршава:

```
$matrica = array();
for ($i=0; $i<3; $i++)
   for ($j=0; $j<3; $j++)
      for ($k=0; $k<3; $k++)
         for ($l=0; $l<3; $l++)
            $matrica[$i][$j][$k][$l] = "$i-$j-$k-$l";
```

У вишедимензионалним матрицама, као и у једнодимензионалним, тип индекса и вредности, као и број елемената у редовима, може се разликовати. При том на низове и матрице у *PHP* програмском језику не треба гледати као на сложене и апстрактне геометријске структуре података, већ пре као на једноставан систем за њихово организовање и адресовање.

## 4.2.4. Структуре гранања

Структуре гранања омогућавају да се у зависности од тренутне вредности или стања одређених променљивих и константи изврше различити делови програма. *PHP* програмски језик подржава две основне структуре гранања: *if/else* и *switch/case*.

### 4.2.4.1. If/else структура гранања

Структура гранања *if/else* се чешће користи јер има компактнију синтаксу и није ограничена на рад са дискретним вредностима.

Структура гранања *if/else* подразумева дефинисања основног услова и њему придруженог скупа наредби који ће се извршити уколико је он испуњен. Додатно, у оквиру исказа *else* се дефинише скуп корака који ће се извршити уколико постављени услов није задовољен.

```
if ($stanje > $iznos)
   $transakcija_odobrena = true;
else
   $transakcija_odobrena = false;
```

Поред ова два основна случаја могу се дефинисати и додатни случајеви коришћењем *else if* исказа:

```
if ($broj == 1)
   echo "Број је 1";
elseif ($broj == 2)
   echo "Број је 2";
else
   echo "Број није ни 1 ни 2";
```

У случају  да се блок наредби који ће се извршити услед испуњавања одређеног

услова састоји од више од једне наредбе, у том случају се он уоквирује витичастим заградама:

```
if ($stanje > $iznos)
{
   $transakcija_odobrena = true;
   $stanje = $stanje - $iznos;
}
else
{
   $transakcija_odobrena = false;
}
```

Коришћење витичастих заграда се препоручује чак и када се у оквиру блока дефинише само једна наредба, јер се јасно уочава које наредбе ће се у том случају извршити, а пре свега због избегавања грешака при будућим проширивањима.

### 4.2.4.2. Switch/case структура гранања

Структура гранања *switch/case* је знатно једноставнија од *if/else* структуре, али и нуди скромније могућности.

```
switch ($boja)
{
   case 'бела':
      echo 'Боја је бела.';
      break;
   case 'црна':
      echo 'Боја је црна.';
      break;
   default:
      echo 'Боја није ни бела, ни црна.';
      break;
}
```

При раду са овом структуром гранања треба водити рачуна о коришћењу

наредбе *break*. Ова наредба одређује крај блока наредби које ће се извршити у одређеном случају. На пример, уколико се гранање:

```
switch ($broj)
{
   case 3:
      echo "Број је већи од 2.";
   case 2:
      echo "Број је већи од 1.";
   case 1:
      echo "Број је већи од 0.";
}
```

позове за број 3 као вредност променљиве *broj* добиће се следећи резултат:

```
Број је већи од 2.
Број је већи од 1.
Број је већи од 0.
```

## 4.2.5. Цикличне структуре

Цикличне структуре се користе за понављање одређеног блока наредби већи број пута који може бити унапред познат или не. За понављање блока наредби унапред познати број пута користи се циклична структура *for*:

```
for ($i = 1; $i<=100; $i++)
{
   echo "$i. извршавање.\n";
}
```

која ће у приказаном случају дати резултат:

```
1. извршавање.
2. извршавање.
...
100. извршавање
```

За случајеве у којима није унапред познат број понављања блока наредби погодније је коришћење наредбе *while*:

```
$broj = 1;
while ($broj <= 128)
{
   echo "$broj \n";
   $broj = $broj + $broj;
}
```

## 4.2.6. Кориснички дефинисане функције

Осим у оквиру цикличних структура извршавање одређеног блока функција се може више пута позивати у оквиру програма и путем његовог измештања у кориснички дефинисану функцију. Ове функције служе за бољу логичку организацију програма кроз издвајање и означавање блокова наредби специфичне намене. За дефинисање корисничких функција користе и се службена реч *function*:

```
function Sabiranje($a, $b)
{
   $zbir = $a + $b;
   return $zbir;
}
```

Коришћење кориснички дефинисаних функција је исто као и коришћење већ уграђених функција:

```
$x = Sabiranje(2, 3);
```

Аргументи код кориснички дефинисаних функција нису обавезни, као ни враћање вредности од стране функције:

```
function UgasiRacunar()
{
   system("shutdown -h now");
}
```

Код именовања функције важе иста правила као и код именовања променљивих, а жељено име не сме бити већ додељено некој другој функцији.

### 4.2.7. Објектно оријентисано програмирање

Основу објектно оријентисаног програмирања чини декларисање класа и њихово инстанцирање у објекте. За декларисање класе у *PHP* програмском језику користи се службена реч *class*. Класа може имати нула или више атрибута и метода, који могу бити јавни, заштићени и приватни (енгл. *public, protected, private*). Јавним атрибутима и методима је могуће приступити и ван објекта док код заштићених и приватних то није случај. Разлика између заштићених и приватних метода је то што се приватни методи не наслеђују. У наставку је дат пример декларације једноставне класе која би се користила за повезивање и рад са *MySQL* базом података:

```
class MySQL
{
   protected $Host;
   protected $User;
   protected $Pass;
   protected $DB;
   protected $Link;

   public function __construct()
   {
      $this->Host = 'localhost';
      $this->User = 'korisnik';
      $this->Pass = 'lozinka';
      $this->DB   = 'baza';
   }

   public function Povezivanje()
   {
      $this->Link = mysqli_connect($this->Host, $this->User, $this->Pass,
         $this->DB);
   }
```

```
   public function IzvrsavanjeUpita($Upit)
   {
      return mysqli_query($this->Link, $Upit);
   }
}
```

Пример коришћења наведене класе дат је у следећем листингу кода:

```
$Baza = new MySQL():
$Baza->Povezivanje();
$Rezultat = $Baza->Upit("select * from Student");
```

## 4.2.8. Разлагање апликације на више фајлова

Развој Веб апликације у виду једног великог фајла у коме се налази целокупан код није превише практично решење. У пракси се често апликација реализује путем већег броја фајлова (у којима су, на пример, смештене кориснички дефинисане функције и класе) који се приликом извршавања позивају, односно чији се садржај укључује у иницијално покренути фајл. За укључивање садржаја других фајлова користе се наредбе *include, include_once, require* и *require_once.*

```
<?php
include 'Lib/DB/MySQL.lib.php';
// Позивање функције дефинисане у учитаној библиотеци
...
```

Разлика између наредби типа *include* и *require* је у томе што ће, уколико не постоји фајл чије се учитавање захтева, наредбе типа *include* послати упозорење, док ће наредбе тип *require* обуставити даље извршавање уз поруку о фаталној грешци. Са друге стране, суфикс *_once* означава да жељени фајл не треба учитавати поново уколико је он већ раније учитан у неком другом делу кода.

## 4.2.9. Коментари

Коментари представљају део програмског кода који се неће извршити већ служи као информација за програмера или програм који са тим кодом ради. Постоје

два основна типа коментара у *PHP*-у: коментари у једном реду и коментари у више редова. За дефинисање коментара у једном реду користе се две косе црте на његовом почетку:

```
$x = 1;
$y = 2;
$z = $x / $y; // TODO: спречити дељење нулом
...
```

Коментари који се простиру на више редова се уоквирују ознакама */** и ***/***:

```
/*
   У наредном делу кода је реализована компресија података.
   Овај код ради исправно уколико је улаз скуп ASCII симбола.
*/
$redovi = explode("\n", $podaci);
...
```

Добар програмски код треба да садржи довољно коментара да програмеру који код чита омогући правилно разумевање његовог рада, било да је он аутор тог кода или не. Са друге стране, превелики број коментара, као и коментари са ниском информативном вредношћу, не служе својој основној сврси већ, напротив, смањују прегледност.

## 4.2.10. Правила кодовања

Иако не постоје чврста правила на који начин треба писати *PHP* код, Група за интероперабилност радних оквира *PHP*-а[13] је развила неколико стандарда чије поштовање омогућава виши ниво интероперабилности између различитих кодова. У наставку су дате неке од препорука:

- Програмски код треба да се налази у *UTF-8* кодној страници.
- Учитавање библиотека и других *PHP* фајлова не треба радити директно већ путем *autoload* функције.
- Конфигурационе параметре извршног окружења не треба мењати унутар програма већ, евентуално, на самом почетку извршавања.

13 http://www.php-fig.org/

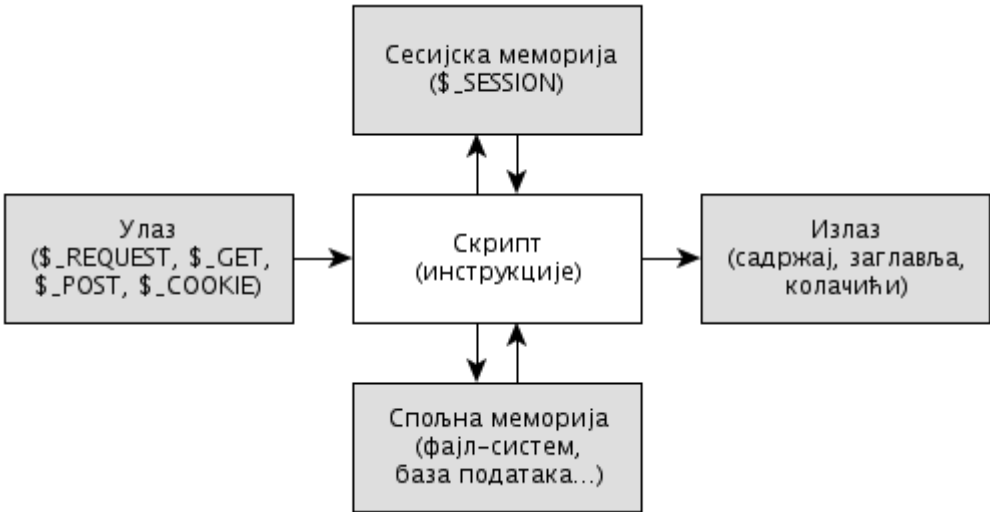
- За називе променљивих и класа треба користити *PascalCase*. На пример, класу треба назвати *PristupBaziPodataka* а не *pristupBaziPodataka, pristup_bazi_podatak* и томе слично.
- За називе функција и метода треба користити тзв. *camelCase*. На пример, функцију треба назвати *funkcijaZaSabiranje*.
- За називе константи треба користити сва велика слова уз раздвајање речи доњом цртом. На пример, *VERZIJA_PROGRAMA*.
- Службене речи PHP језика, као што су *true, false, null* и сличне, треба писати малим словима.
- За сваки ниво увлачења кода треба користити по четири спојена знака размака. За увлачење не треба користити табулатор.
- Не постоји ограничење у броју знакова по реду кода али је пожељно да он буде мањи од 80.
- Витичасте заграде треба да се налазе у издвојеним редовима.
- У фајловима који садрже само *PHP* код не треба користити *?>* ознаку за затварање.
- Код навођења више аргумената функције или метода након запета треба да се налази један размак.

## 4.3. Комуникација са окружењем

Основну комуникацију *PHP* скриптова са окружењем представља преузимање параметара захтева - на пример, података достављених путем формулара - и слање резултата обраде у виду одговора. За приступ овим подацима користе се системске променљиве *$_GET* (за податке достављене *GET* методом *HTTP* протокола), *$_POST* (за податке достављене *POST* методом *HTTP* протокола) и *$_REQUEST* (за оба претходно наведена метода). Додатно, променљива *$_COOKIE* се користи за приступ вредностима колачића који су претходно послати кориснику.

Са друге стране, резултат извршавања скриптова (обраде захтева) се подразумевано шаље назад кориснику и то у виду садржаја (*HTML*, бинарни садржај и др.), заглавља (подаци за преусмеравање корисника, опис садржаја и др.) и колачића. Колачићи се такође шаљу и оквиру заглавља *HTTP* пакета али се могу сматрати посебним типом одговора, а и за њихово слање се користи

засебна функција, *setcookie*. За слање обичних параметара заглавља користи се функција *header* док се за слање самог садржаја може користити већи број функција, као што су *echo*, *print* и сл.




*Слика X. Могућности комуникације скрипта са окружењем*

Осим улаза и излаза, *PHP* скриптови често током извршавања комуницирају и са подацима који се налазе у спољној меморији, путем фајл-система, система за управљање базама података и сл. Ова комуникација може подразумевати само читање података или и њихову измену.

Један од веома важних меморијских простора са којим *PHP* скриптови комуницирају током извршавања је сесијска меморија. Сесијска меморија представља посебан део радне меморије који је резервисан за појединачног корисника током целе његове радне сесије. Ова меморија је, насупрот меморији која се резервише приликом пријема захтева корисника, перзистента (трајна), односно њен садржај се не уклања након завршетка извршавања скрипта већ се чува током целе радне сесије корисника. Приступ сесијском меморијском простору се скриптовима омогућава путем фунције *session_start* (мада се самом конфигурацијом сервера ово често аутоматски одрађује) а за сам приступ меморији користи се системска променљива *$_SESSION* која је типа низ.

### 4.3.1. Обрада података из формулара

Вредности из *HTML* формулара се серверској страни могу достављати путем *HTTP* метода *GET* и *POST*. Приступ послатим подацима из формулара у PHP програмском језику врши се путем суперглобалних променљивих *$_GET* и *$_POST*. У наставку је дат пример *HTML* формулара који користи метод *POST* за слање података серверу:

```
<form action="obrada.php" method="post">
   <input type="text" name="ime" />
   <button type="submit">Слање</button>
</form>
```

Приступ послатим подацима унутар *PHP*-а приказан је на следећем примеру:

```
echo $_POST['ime'];
```

Приступ подацима добијеним путем *GET* метода врши се коришћењем суперглобалне променљиве *$_GET*. На пример, приступом адреси:

```
http://localhost/obrada.php?ime=Петар&prezime=Петровић
```

подацима достављеним *GET* методом би се приступило на следећи начин:

```
echo $_GET['ime'].' '.$_GET['prezime'];
```

Приступ подацима из *HTTP* захтева може се обједињено вршити и путем суперглобалне променљиве $*_REQUEST*:

```
<form action="obrada.php?ime=Петар" method="post">
   <input type="text" name="prezime" />
   <button type="submit">Слање</button>
</form>
```

а оба начина ће у следећем случају дати исти резултат:

```
echo $_GET['ime'].' '.$_POST['prezime'];
echo $_REQUEST['ime'].' '.$_REQUEST['prezime'];
```

Међутим, при коришћењу променљиве *$_REQUEST* треба имати у виду да уколико постоји исти индекс у оба метода, у променљиву *$_REQUEST* ће ући само вредност добијена методом *POST*, односно вредност добијена методом *POST* ће бити уписана преко вредности добијене методом *GET*. Ово је подразумевани понашање које се може мењати директивом:

```
variables_order = "GPCS"
```

у конфигурационом фајлу *PHP*-а, с тим да то није препоручено због очувања компатибилности Веб апликације са подразумевано подешеним продукционим окружењима.

## 4.3.2. Рад са фајловима

PHP нуди више функција за подршку различитим приступима рада са фајловима. Две основне функције које омогућавају уписивање садржаја у фајлове и његово ишчитавање су *file_put_contents* и *file_get_contents*:

```
// Уписивање садржаја у фајл
$a = 'Ово ја садржај који се уписује у фајл.';
file_put_contents('fajl.txt', $a);

// Учитавање садржаја фајла
$b = file_get_contents('fajl.txt');
echo $b;
```

Приказани начин рада са фајловима је применљив када се ради са малом количином података који су унапред познати. Међутим, код великих података, као и код података који константно пристижу (рецимо читање података са неког сензора) потребно је применити другачији приступ, односно направити везу са фајлом а затим учитавати или уписивати мање блокове садржаја. У наставку је дат пример ишчитавања садржаја фајла бајт по бајт са директним исписивањем, што је приступ са најмањим утрошком меморије:

```
<?php
$fajl = fopen('fajl.txt','r');
while (!feof($fajl))
{
```

```
   echo fgetc($fajl);
}
fclose($fajl);
?>
```

Осим за рад са садржајем фајлова, *PHP* поседује и функције за руковање њима у оквиру фајл-система. У наставку је дат пример ишчитавања садржаја директоријума:

```
<?php
$dir = opendir("podaci");
while ($stavka = readdir($dir))
{
   echo "$stavka <br />";
}
closedir($dir);
?>
```

У наредном примеру се приказује копирање, премештање и уклањање фајлова:

```
// Премештање фајла путем копирања и уклањања
copy('dir1/stari.fajl', 'dir2/novi.fajl');
unlink('dir1/stari.fajl');
// Премештање фајла коришћењем наменске функције
rename('dir1/stari.fajl', 'dir2/novi.fajl');
```

При том треба приметити да се код уклањања фајла, у ствари, само раскида веза према њему унутар фајл система док његов садржај и даље постоји на коришћеном меморијском уређају. Међутим, то и јесте подразумевани начин уклањања фајлова уопште, јер је преписивање његовог садржаја битовима чија је вредност нула временски скупа операција која не подиже значајно ниво безбедности.

Препорука аутора је да се фајлови и фајл-систем за чување података користе само у ситуацијама када је то неизбежно. Односно, кад год је то могуће треба изабрати системе за управљање базама података јер нуде веће перформансе, решавају проблем конкурентног приступа и слично.

### 4.3.2.1. Руковање фајловима из формулара

Коришћењем *HTML* формулара могуће је постављати фајлове на Веб сервер. Да би се то постигло потребно је поставити тип кодовања садржаја на *multipart/form-data* а сам формулар послати методом *post*:

```
<form enctype="multipart/form-data" action="upload.php" method="post">
   Фајл: <input type="file" name="fajl1" />
   <button type="submit">Слање</button>
</form>
```

Након слања фајла серверу путем наведеног формулара, посталатом фајлу се може приступити путем системске променљиве *$_FILES* која је у суштини низ у коме је послати фајл индексован називом који му је дат у формулару:

```
    [fajl1] => Array
        (
            [name] => HTML.pdf
            [type] => application/pdf
            [tmp_name] => /tmp/phpVtgcqy
            [error] => 0
            [size] => 143420
        )
```

Из низа особина придружених послатом фајлу може се видети његов изворни назив, тип, привремена локација у фајл-систему сервера, величина, као и број који одређује грешку која се јавила приликом слања. Уколико је број грешке 0, то значи да је фајл достављен серверу без проблема, док више вредности указују на различите типове грешака (прекорачење дозвољене величине фајла, различите грешке при складиштењу на привремену локацију и слично).

Послати фајл се на серверу може даље обрађивати на више начина. У два најчешће коришћена спадају директно ишчитавање његовог садржаја са привремене локације или његово премештање на неку другу локацију (постоји наменска функција *move_uploaded_file*). Треба имати у виду да се фајл након достављања серверу **привремено** складишти на **привремену** локацију са које ће временом аутоматски бити уклоњен, на пример приликом рестартовања сервера.

## 4.3.3. Рад се базама података

Једна од вероватно најважнијих могућности *PHP*-а је рад са системима за управљање базама података, односно управљање подацима путем њиховог коришћења. Најпопуларнији систем за управљање базама података који се користи у комбинацији са *PHP* језиком за развој Веб апликација је *MySQL*.

Базе података се најчешће моделују у спољним алатима а кроз Веб апликације се врши манипулација подацима у њима. Четири основне операције за управљање подацима у бази података су унос, преглед, измена и уклањање.

### 4.3.3.1. Повезивање са сервером базе података

За повезивање *PHP* скрипта са *MySQL* сервером базе података користи се функција *mysqli_connect* са задата четири параметра - адреса или име сервера, корисничко име, лозинка и назив базе података:

```
// Повезивање са сервером базе података
$bp = mysqli_connect("localhost","korisnik","loz1nka","baza");
if (!$bp)
   die('Јавила се грешка при повезивању са сервером базе података.');
```

Да би процес повезивања био успешан потребно је да постоји могућност мрежне комуникације са сервером, да је отворен налог који се користи при повезивању, као и да тај налог има право да приступи жељеној бази података. Уколико нешто од овога није испуњено, систем ће пријавити грешку:

**Warning**: mysqli_connect(): (HY000/1045): Access denied for user 'korisnik'@'localhost' (using password: YES) in **/home/ajevremovic/public_html/temp.php** on line **3**
Јавила се грешка при повезивању са сервером базе података.

*Слика 1. Порука о грешци при повезивању на сервер базе података*

Када је веза са сервером базе података успешно успостављена, она ће остати активна све до краја извршавања захтева уколико се експлицитно не захтева њен раскид наредбом *mysqli_close*:

```
// Раскид везе са сервером базе података
mysqli_close($bp);
```

У току извршавања једног захтева могуће је успоставити више паралелних веза

ка различитим базама података а база над којом ће се извршити одређени упит одређује се путем променљиве која означава везу (тзв. идентификатор везе). Тип променљивих које означавају везе са базама података је **ресурс**.

### 4.3.3.2. Додавање података у базу

За додавање нових записа у базу података користи се општа *PHP* наредба за извршавање упита *mysqli_query* у комбинацији са *SQL* наредбом *insert*:

```
// Унос новог записа у табелу
$upit = "insert into Studenti (Indeks, Ime, Prezime) values ('1/2013',
'Петар', 'Петровић')";
$rezultat = mysqli_query($bp, $upit);
if (!$rezultat)
   die(mysqli_error($bp));
```

Наведени пример неће дати никакав испис уколико се упит изврши без грешака, а уколико до њих дође прекинуће се даље извршавање уз исписивање поруке о грешци. На пример, уколико се зада погрешан назив табеле у коју се додаје запис, систем ће исписати грешку:

```
Table 'baza.Studenti' doesn't exist
```

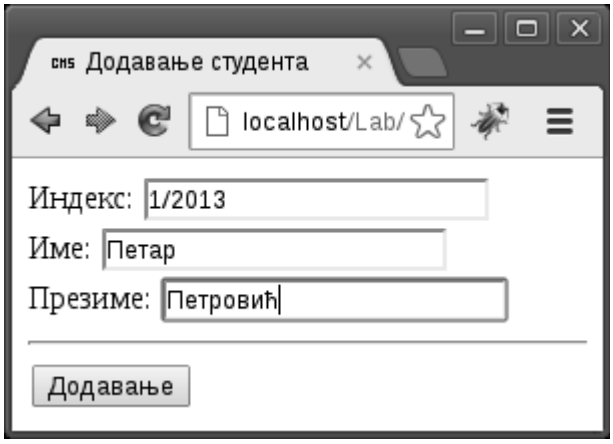
Код упита за унос нових записа у табелу није неопходно наводити имена колона већ је довољно само навести вредности у одговарајућем редоследу. Међутим, таква пракса није пожељна с обзиром на то да ће скриптови писани на такав начин престати да раде оног тренутка када се буде повећао или смањио број колона у табели. Још горе, у случају промене редоследа колона у бази, скриптови ће, уколико су замењене колоне имале исти тип података, наставити са радом али ће давати погрешне резултате.

Најчешћи сценарио додавања података у базу код Веб апликација је на основу уноса података од стране корисника путем *HTML* формулара. У наставку је дат пример кода којим би се добио формулар за додавање студената:

```
<!DOCTYPE html>
<html>
   <head>
      <title>Додавање студента</title>
```

```
      <meta charset="UTF-8">
   </head>
   <body>
      <form action="dodavanje-db.php" method="post">
         Индекс: <input type="text" name="Indeks" />
         <br />
         Име: <input type="text" name="Ime" />
         <br />
         Презиме: <input type="text" name="Prezime" />
         <hr />
         <button type="submit">Додавање</button>
      </form>
   </body>
</html>
```

Резултат интерпретирања наведеног кода приказан је на слици 1:




*Слика 1. Формулар за додавање новог студента*

У наведеном формулару је дефинисано да се унети подаци прослеђују скрипту *dodavanje-db.php* за даљу обраду, односно њихов унос у базу података. У наставку је дат претходни пример *PHP* кода за додавање нових записа проширен делом за преузимање достављених података и њихову нормализацију. Додатно, код је проширен и делом који аутоматски враћа корисника поново на формулар за унос уколико није дошло до грешке приликом извшравања упита у бази података.

```
// Повезивање са сервером базе података
$bp = mysqli_connect("localhost","korisnik","lozinka","baza");
if (!$bp)
   die('Јавила се грешка при повезивању са сервером базе података.');

// Нормализација улазних података
$Indeks  = mysqli_real_escape_string($bp, @$_POST['Indeks']);
$Ime     = mysqli_real_escape_string($bp, @$_POST['Ime']);
$Prezime = mysqli_real_escape_string($bp, @$_POST['Prezime']);

// Унос новог записа у табелу
$upit = "insert into Student (Indeks, Ime, Prezime) values ('$Indeks',
'$Ime', '$Prezime')";
$rezultat = mysqli_query($bp, $upit);
if (!$rezultat)
                        die(mysqli_error($bp));

// Раскид везе са сервером базе података
mysqli_close($bp);

// Преусмеравање на формулар за унос
die(header("Location: dodavanje.html"));
```

Једна од веома важних ствари код коришћења улазних података добијених од стране корисника у упитима у бази података јесте њихова нормализација. **Нормализација улазних података** првенствено представља њихово довођење у опсег исправних и планираних вредности, односно вредности које имају смисла за даље коришћење. На пример, при уносу имена се не очекује да она садрже цифре, као ни да бројеви телефона садрже слова. Нормализација улазних података је посебно важна јер представља основну одбрану од *SQL injection* напада.

### 4.3.3.3. Учитавање података из базе

Учитавање података из базе и њихов приказ је релативно једноставна операција која се обавља у једном кораку. У наставку је дат пример преузимања података из табеле у бази података и њихово приказивање у табели *HTML* документа:

```
<!DOCTYPE html>
<html>
   <head>
      <title>Преглед студента</title>
      <meta charset="UTF-8">
   </head>
   <body>
      <table border="1">
         <thead>
            <tr>
               <th>ID</th>
               <th>Индекс</th>
               <th>Име</th>
               <th>Презиме</th>
            </tr>
         </thead>
<?php

// Повезивање са сервером базе података
$bp = mysqli_connect("localhost","korisnik","lozinka","baza");
if (!$bp)
   die('Јавила се грешка при повезивању са сервером базе података.');

// Учитавање података из базе
$upit = "select * from Student";
$rezultat = mysqli_query($bp, $upit);
if (!$rezultat)
   die(mysqli_error($bp));

// Приказ података
while ($red = mysqli_fetch_object($rezultat))
{
   echo "<tr>\n";
   echo " <td>{$red->ID}</td>\n";
```

```
   echo " <td>{$red->Indeks}</td>\n";
   echo " <td>{$red->Ime}</td>\n";
   echo " <td>{$red->Prezime}</td>\n";
   echo "</tr>\n";
}

?>
      </table>
   </body>
</html>
```

Резултат извршавања наведеног кода приказан је на следећој слици:

| ID | Индекс | Име | Презиме |
|---|---|---|---|
| 1 | 1/2013 | Петар | Петровић |
| 2 | 2/2013 | Ђорђе | Ђорђевић |
| 3 | 3/2013 | Лазар | Лазаревић |

*Слика 1. Резултат учитавања и приказивања података из базе*

Табела која чини резултат извршавања приказује податке који се налазе у бази, али не нуди никакву могућност за даљи рад са њима. За потребе измене и уклањања одређених слогова њих је прво потребно одредити. Табела у којој се приказују подаци је згодно место за то, тако да ћемо скрипт за приказ проширити додавањем две хипер-везе које воде на странице за измену и уклањање и при том достављају идентификатор записа који желимо да мењамо/уклањамо:

```
   ...
   echo " <td>{$red->Prezime}</td>\n";
   echo " <td><a href='izmena.php?ID={$red->ID}'>Измена</a></td>\n";
   echo " <td><a href='uklanjanje.php?ID={$red->ID}'>Уклањање</a></td>\n";
   echo "</tr>\n";
   ...
```

Резултат извршавања наведеног кода приказан је на следећој слици:

| ID | Индекс | Име | Презиме | | |
|---|---|---|---|---|---|
| 1 | 1/2013 | Петар | Петровић | Измена | Уклањање |
| 2 | 2/2013 | Ђорђе | Ђорђевић | Измена | Уклањање |
| 3 | 3/2013 | Лазар | Лазаревић | Измена | Уклањање |

*Слика 2. Приказ података са линковима за измену и уклањање*

Такође, да би се могло приступити страници за додавање новог студента након табеле је додата одговарајућа хипер-веза:

```
...
   </table>
   <a href="dodavanje.html">Додавање новог студента</a>
</body>
...
```

тако да је сада са основне странице апликације могуће приступити свим акцијама:

| ID | Индекс | Име | Презиме | | |
|---|---|---|---|---|---|
| 1 | 1/2013 | Петар | Петровић | Измена | Уклањање |
| 2 | 2/2013 | Ђорђе | Ђорђевић | Измена | Уклањање |
| 3 | 3/2013 | Лазар | Лазаревић | Измена | Уклањање |

Додавање новог студента

*Слика 3. Табела са линковима за измену, уклањање и додавање студената*

Описани пример је основа за приказивање података из базе путем *HTML* табела, а у пракси се може знатно мењати и зависности од конкретне ситуације и потреба.

### 4.3.3.4. Измена података у бази

Измена података у бази је најсложенија операција основног руковања подацима. За успешну измену неког слога у бази потребно је проћи кроз следеће кораке:

1. избор слога који треба изменити

2. учитавање слога и приказивање у формулару за измену
3. примена нових вредности за задати слог који се мења

У наредном примеру кренућемо од формулара за измену података, подразумевајући да је идентификатор слога у бази који треба мењати већ достављен коришћењем претходно описаних линкова у табели за приказ података:

```
<!DOCTYPE html>
<html>
   <head>
      <title>Измена студента</title>
      <meta charset="UTF-8">
   </head>
   <body>
<?php

// Повезивање са сервером базе података
$bp = mysqli_connect("localhost","korisnik","lozinka","baza");
if (!$bp)
   die('Јавила се грешка при повезивању са сервером базе података.');

// Нормализација улазних података
$ID = (int) @$_REQUEST['ID'];

// Учитавање података записа из базе
$upit = "select * from Student where ID=$ID";
$rezultat = mysqli_query($bp, $upit);

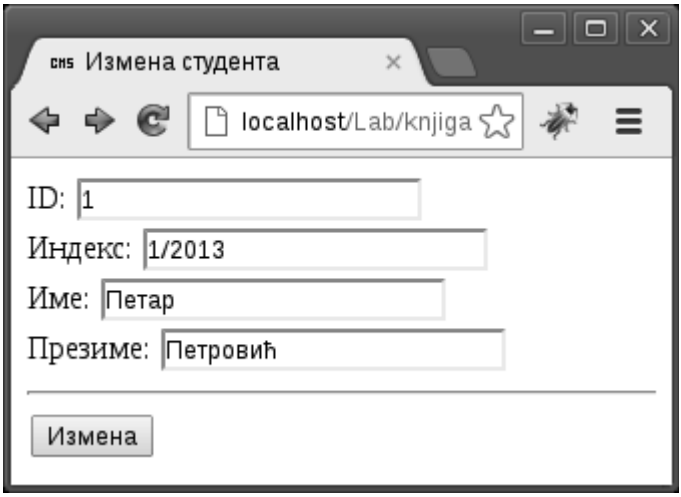
// Провера да ли је било грешака и да ли постоји тражени запис
if (!$rezultat)
   die(mysqli_error($bp));
if (mysqli_num_rows($rezultat) != 1)
   die('Не постоји студент са задатим ID параметром.');

// Пребацивање резултата у објекат 'Student'
```

```
$Student = mysqli_fetch_object($rezultat);


?>
      <form action="izmena-db.php" method="post">
         ID: <input type="text" name="ID" value="<?php echo $Student->ID; ?>"
readonly />
         <br />
         Индекс: <input type="text" name="Indeks" value="<?php echo $Student-
>Indeks; ?>" />
         <br />
         Име: <input type="text" name="Ime" value="<?php echo $Student-
>Ime; ?>" />
         <br />
         Презиме: <input type="text" name="Prezime" value="<?php echo
$Student->Prezime; ?>" />
         <hr />
         <button type="submit">Измена</button>
      </form>
   </body>
</html>
```

Резулатат извршавања наведеног кода приказан је на следећој слици:




*Слика 1. Формулар за измену попуњен иницијалним подацима*

Оно што треба запазити је да је пољу у коме се чува *ID* записа који треба мењати

постављен и атрибут *readonly*. Разлог томе је спречавање да корисници нехотице измене погрешан запис, мислећи да могу да мењају и вредност овог поља, односно колоне у бази. Вредност *ID* је неопходно проследити у наредни корак да би се знало који запис треба изменити. Алтернативно, а у пракси боље решење је коришћење контроле у којој се чува *ID* вредност типа *hidden* уместо тренутне контроле за унос текста. У наставку је дат код којим се унете измене примењују у бази података а корисник преусмерава на почетну страницу:

```
// Повезивање са сервером базе података
$bp = mysqli_connect("localhost","korisnik","lozinka","baza");
if (!$bp)
   die('Јавила се грешка при повезивању са сервером базе података.');

// Нормализација улазних података
$ID = (int) @$_REQUEST['ID'];
$Indeks  = mysqli_real_escape_string($bp, @$_POST['Indeks']);
$Ime     = mysqli_real_escape_string($bp, @$_POST['Ime']);
$Prezime = mysqli_real_escape_string($bp, @$_POST['Prezime']);

// Измена података у бази
$upit = "update Student set Indeks='$Indeks', Ime='$Ime', Prezime='$Prezime'
where ID=$ID";
$rezultat = mysqli_query($bp, $upit);
if (!$rezultat)
   die(mysqli_error($bp));

// Преусмеравање на почетну страницу
die(header("Location: index.php"));
```

Наведени пример представља основни модел за измену података у бази. Међутим, у њему није обрађен случај у коме подаци у бази садрже наводнике, односно није спречена грешка која ће се јавити приликом постављања тих вредности у формулар. Ипак, треба имати у виду да овај пример служи само за илустровање концепта, а да је препорука аутора што бржи прелазак на начине програмирања код којих се приказ и логика раздвајају јер је код њих спречавање таквих грешака далеко једноставније.

### 4.3.3.5. Уклањање података из базе

Уклањање података из базе је најсложенија операција основног руковања подацима и одвија се у једном кораку. Скрипту за уклањање података је потребно доставити услов који слогови треба да испуне да би били уклоњени, што је у конкретном случају *ID* студента:

```
<?php

// Повезивање са сервером базе података
$bp = mysqli_connect("localhost","korisnik","lozinka","baza");
if (!$bp)
   die('Јавила се грешка при повезивању са сервером базе података.');

// Нормализација улазних података
$ID = (int) @$_REQUEST['ID'];

// Уклањање захтеваног слога
$upit = "delete from Student where ID=$ID";
$rezultat = mysqli_query($bp, $upit);
if (!$rezultat)
   die(mysqli_error($bp));

// Преусмеравање на почетну страницу
die(header("Location: index.php"));
```

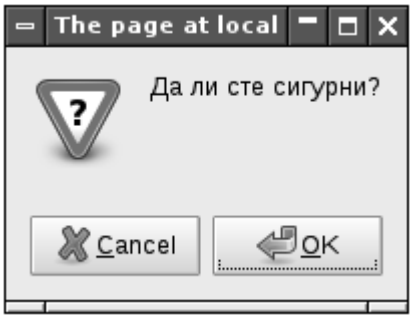
Приказани код уклања тражени запис из базе и преусмерава корисника на почетну страницу. У довољно брзој комуникацији између клијента и сервера, корисник ће толико брзо бити враћен на страницу за преглед података да ће цео процес изгледати као нестанак једног реда из табеле. Да би се корисник заштитио од случајног уклањања записа из базе, у хипер-везу која активира уклањање пожељно је, коришћењем *JavaScript*-а, додати потврду, односно хипер-вези:

```
<a href='uklanjanje.php?ID={$red->ID}'>Уклањање</a>
```

додати *JavaScript* функцију која додатно обрађује клик мишем, путем атрибута *onclick*:

```
onclick='if (!confirm(\"Да ли сте сигурни?\")) return false;'
```

која ће приликом активирања хипер-везе захтевати додатну потврду:




*Слика 1. Подразумевани дијалог за потврду активирања хипер везе*

У наведеном примеру је приказан основни начин за уклањање података из базе. Са друге стране, имајући у виду колико је једноставно уклонити податке из базе, а колико их је тешко или немогуће повратити у неким случајевима, посебну пажњу треба обратити на спречавање случајног уклањања података од стране регуларних корисника, као и на отклањање евентуалних пропуста које би у циљу уклањања података могао да искористи злонамерни корисник.

## 4.3.4. Рад са колачићима

Колачићи (енгл. *cookies*) су подсистем *HTTP* протокола који омогућава да Веб апликација на клијенту сачува одређену вредност коју може користити при наредним захтевима тог клијента. На пример, колачићи се могу користити за бројање посета одређеног клијента, иако се он није аутентификовао на сајту:

```
Добродошли, ово је ваша 3. посета!
```

За слање колачића клијенту у PHP језику се користи функција *setcookie*. Ова функција има више аргумената а најважнија су прва три којима се одређује назив, вредност и рок трајања. На пример, уколико желимо да поставимо колачић којим се кориснику одобрава приступ неком ресурсу у наредних сат времена, то ћемо учинити коришћењем кода:

```
setcookie('pristup', 'odobren', time() + 3600);
```

Након тога, овој вредности ћемо при наредним захтевима у току сат времена

моћи да приступимо на следећи начин:

```
if (@_COOKIE['pristup'] == 'odobren')
{
   ...
}
else
   die('Приступ захтеваном ресурсу није одобрен');
```

Уколико се као време до кога важи колачић не зада као аргумент функције, или се зада вредност нула, колачић ће важити до ресетовања браузера. Са друге стране, уколико се као рок трајања зада вредност мања од тренутног времена, колачић ће бити уклоњен из Веб браузера:

```
setcookie('pristup', 'odobren', time() - 3600);
```

Једна од најважнијих употреба колачића у *PHP* језику је за чување идентификатора сесије корисника. Колачић који је за то задужен носи назив *PHPSESSID* а његов подразумевани рок трајања је до затварања Веб браузера (с тим да сесија може и раније истећи на серверу).

## 4.3.5. Управљање сесијом и сесијском меморијом

Веб сервер сваки захтев обрађује засебно, додељујући му потребну меморију за учитавање садржаја, скриптова и чување садржаја променљивих. Ова меморија се резервише након пријема захтева а ослобађа након завршетка његове обраде и слања резултата клијенту. Другим речима, њен садржај се уклања и није доступан при следећем захтеву.

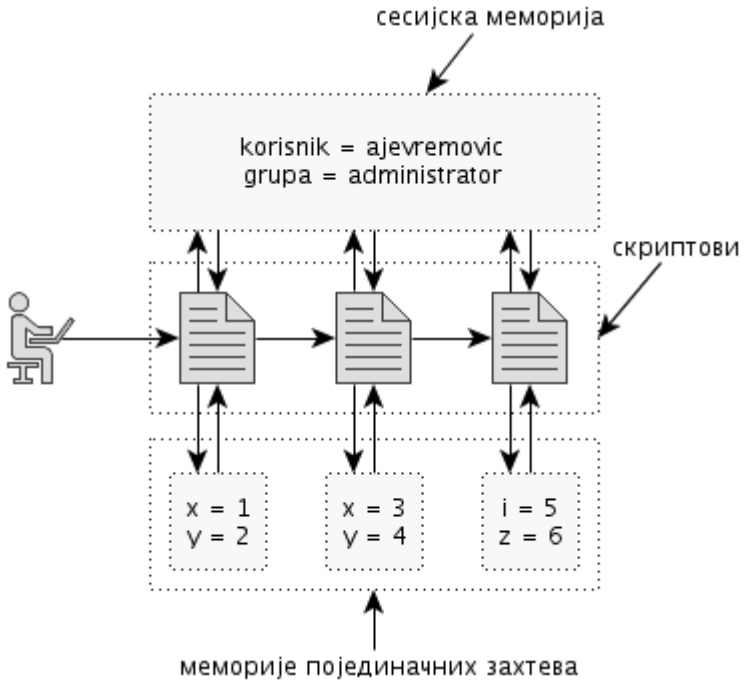
Једно решење за чување података између два захтева јесте њихово слање клијенту, као резултат претходног захтева, и њихово преузимање од клијента у оквиру следећег захтева. Ово се може постићи на два начина: коришћењем колачића или уградњом података у *URL*-ове који ће се користити за наредне захтеве (на пример линкове, формуларе и слично). Међутим, оба поменута начина имају озбиљне недостатке по питању једноставности реализације, безбедности и робусности.

Са друге стране, Веб сервер временски блиске захтеве појединачних клијената групише у сесије. Свака сесија има свој јединствени идентификатор (енгл. *session id*) који се (псеудо)случајно одређује приликом првог захтева и важи за

све наредне захтеве у оквиру те сесије. Идентификатор сесије се кориснику додељује и од њега преузима у виду колачића, који код PHP програмског језика носи назив *PHPSESSID*:

```
PHPSESSID=849d75ff70d8c751c8d44392b3d9a246
```

Груписање временски блиских захтева клијента у сесију значајно је пре свега због могућности чувања података између два захтева на серверу, у оквиру сесијске меморије. Ово елиминише потребу да се подаци на претходно описане начине сваки пут шаљу клијенту и преузимају од њега. Сесијска меморија је низ коме се приступа путем системске променљиве *$_SESSION*. Број и тип елемената овог низа није ограничен.




*Слика 1. Сесијска меморија је трајна и заједничка за све захтеве*

За омогућавање приступа сесијској меморији користи се функција *session_start()* (која мора бити позвана у оквиру сваког захтева чији скриптови имају потребу за приступ сесијској меморији), док се за уништавање њеног комплетног садржаја користи функција *session_destroy()*. Садржај сесијске меморије се такође аутоматски уклања и после дуже неактивности корисника (тзв. истицање сесије,

енгл. *session timeout*) која подразумевано значи више од 1.440 секунди између два захтева, мада се ово време може подешавати.

## 4.4. Контрола приступа апликацији

Један од веома важних захтева код већине реалних Веб апликација је и контрола приступа неким или свим секцијама, односно функцијама. На пример, велики број Веб сајтова има јавни део, у коме се неке функције могу слободно користити од стране анонимних, непријављених корисника, и део за администраторе коме је за приступ потребно унети одговарајуће корисничко име и лозинку. Додатно, сложеније Веб апликације укључују и категоризацију корисника код које се корисницима одобрава коришћење одређених функционалности само уколико кориснички налог путем кога су пријављени припада одговарајућој категорији корисника.

Два у пракси најчешће коришћена приступа за контролу приступа Веб апликацијама су:

1. контрола приступа на нивоу директоријума
2. контрола приступа на нивоу апликације

Контролу приступа на нивоу директоријума је једноставније реализовати јер се користе постојеће функције самог Веб сервера, док је контролу приступа на нивоу апликације потребно самостално реализовати.

### 4.4.1. Контрола приступа на нивоу директоријума

Контрола приступа или заштита апликације на нивоу директоријума представља једноставан начин за ограничавање приступа ресурсима у одређеном директоријуму на серверу само корисницима којима су додељена корисничка имена и лозинке. Ова функционалност је уграђена у *Apache* Веб сервер, а остварује се коришћењем *.htaccess* и *.htpasswd* фајлова и програма *htpasswd*.

Улога *.htaccess* фајлова је измена подразумеваног подешавања *Apache* сервера за директоријум у коме се сам фајл налази. За ограничавање приступа ресурсима у локалном директоријуму само на пријављене кориснике потребно је дефинисати неколико параметара, као што је локација фајла у коме се кориснички налози чувају, порука приликом пријављивања, тип захтева на који се контрола примењује и слично (листинг 1).

```
AuthUserFile /home/ajevremovic/.htpasswd
AuthGroupFile /dev/null
AuthName "Захтевани ресурс је заштићен - морате да се пријавите за приступ."
AuthType Basic

<Limit GET POST>
require valid-user
</Limit>
```

*Листинг 1. Укључивање контроле приступа путем .htaccess фајла*

Након уноса конфигурационих опција у *.htaccess* фајл потребно је креирати *.htpasswd* фајл са налозима корисника (њиховим корисничким именима и лозинкама). Оно о чему треба водити рачуна је да се тај фајл постави ван директоријума којима корисници имају приступ преко Веб сервера јер га у противном могу преузети и из њега преузети туђе приступне параметре. Сам *.htaccess* фајл је текстуалног типа и може се ручно мењати али је за рад са њим предвиђен програм *htpasswd* који долази уз сам серверски софтвер.

```
# htpasswd -c -b /home/ajevremovic/.htpasswd ajevremovic lozinka123
Adding password for user ajevremovic
```

*Листинг 2. Иницијално креирање .htpasswd фајла и додавање корисника*

Додавање налога за новог корисника приказано је на листингу 2. Пошто је тренутно у питању рад са *.htpasswd* фајлом који тренутно не постоји, односно треба да се креира, користи се параметар *-c*. Параметар *-b* се користи за унос лозинке одмах након корисничког имена, а не у оквиру додатног корака.

```
# htpasswd -b /home/ajevremovic/.htpasswd ajevremovic loz456
Updating password for user ajevremovic
```

*Листинг 3. Измена лозинке постојећег корисника*

Измена лозинке за постојећи налог је идентична његовом додавању. Другим речима, након задавања корисничког имена и лозинке кориснички налог ће бити креиран уколико не постоји, односно лозинка ће бити промењена у случају да се задато корисничко име већ налази у бази.

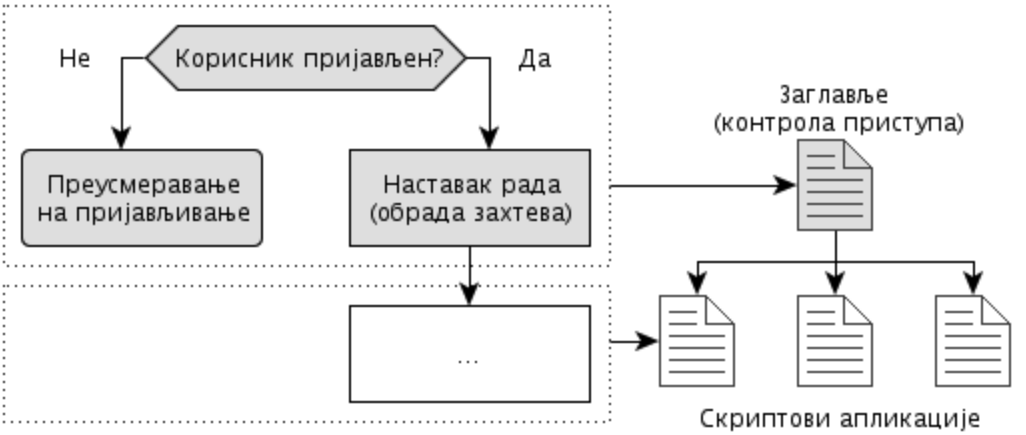
```
# htpasswd -D /home/ajevremovic/.htpasswd ajevremovic
Deleting password for user ajevremovic
```

*Листинг 4. Уклањање корисничког налога*

За уклањање постојећег корисничког налога користи се параметар *-D* уз навођење корисничког имена на које се уклањање односи.

## 4.4.2. Контрола приступа на нивоу апликације

Знатно сложенији, али и знатно флексибилнији начин за контролу приступа Веб апликацији (у односу на контролу приступа на нивоу директоријума) јесте интеграција контроле приступа у саму Веб апликацију. Контрола приступа на нивоу апликације реализује се на тај начин што се у све скриптове апликације којима се контролише приступ уграђује део кода који прво проверава да ли је корисник успешно пријављен или не. У случају да је резултат ове провере позитиван, скрипт се извршава. У противном, скрипт се не извршава а корисник се преусмерава на страницу за пријављивање. За чување информације о томе да ли је корисник пријављен користи се сесијска меморија а самој апликацији се додаје и посебна страница која служи за пријављивање корисника.




*Слика 1. Регуларни скриптови апликације се штите додавањем провере у заглавље*

У оквиру листинга 1. дат је пример кода којим се проверава да ли је корисник пријављен, односно да ли има право да изврши захтевани скрипт. Провера се врши на основу тога да ли је дефинисана вредност за индекс *Korisnik* у оквиру сесијске меморије. Уколико то није случај извршавање скрипта се прекида а

корисник се преусмерава на страницу на којој је формулар за пријављивање. Приказани код би требало да се налази на почетку сваког скрипта који се штити од покретања од стране корисника који нису пријављени.

```
<?php
session_start();

// Код којим се врши провера да ли је корисник пријављен
if (!isset($_SESSION['Korisnik'])
   die(header('Location: prijavljivanje.php');

// Изворне инструкције везане за функционалност скрипта
// ...
?>
```

*Листинг 1. Пример кода којим се проверава да ли је корисник пријављен*

Сам процес пријављивања корисника састоји се од два корака:

1. уноса корисничког имена и лозинке у формулар
2. провера да ли постоји кориснички налог са задатим параметрима

Сам формулар за пријављивање је стандардни *HTML* формулар са пољима за унос корисничког имена и лозинке. У оквиру листинга 2. дат је пример *HTML* кода таквог формулара, са додатком *PHP* кода који у садржај додаје поруку да пријављивање није успешно, односно да не постоји корисник са задатим параметрима, уколико је то захтевано параметром *greska* у самом захтеву.

```
<html>
   <head>
      <title>Пријављивање</title>
      <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
   </head>
   <body>
      <form action="provera.php" method="post">
         Корисник: <input type="text" name="korisnik" /> <br />
         Лозинка: <input type="password" name="lozinka" />  <br />
         <button type="submit">Пријављивање</button>
```

```
        <?php
        if (@$_REQUEST['greska'])
        {
           echo "<hr />";
           echo "Грешка - не постоји корисник са задатим параметрима.";
        }
        ?>
      </form>
   </body>
</html>
```

*Листинг 2. HTML формулар странице за пријављивање корисника*

Изглед *HTML* формулара за пријављивање, датог у листингу 2. приказан је на слици 2.




*Слика 2. Изглед формулара за пријављивање корисника*

Формулар за пријављивање унете параметре шаље на обраду скрипту *provera.php*. Задатак овог скрипта је да провери да ли су задати параметри исправни, односно да ли постоји корисник са задатим корисничким именом и лозинком. У нашем примеру (листинг 3.) провера да ли су задати параметри исправни врши се на основу покушаја да се из *MySQL* базе података, из табеле *Korisnik*, у којој се корисничко име налази у колони *ID* а лозинка у колони *Lozinka*, учита запис у коме се они налазе.

```
<?php
session_start();

// Повезивање са сервером базе података
// ...

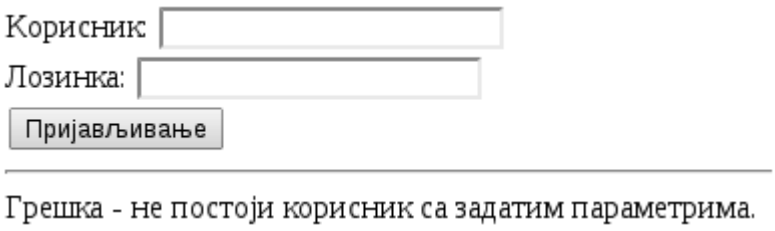
// Нормализација улазних параметара
$k = mysqli_real_escape_string(@$_REQUEST['korisnik'];
```

```
$l = mysqli_real_escape_string(@$_REQUEST['lozinka'];

// Провера да ли постоји запис у табели 'Korisnik' са задатим параметрима
$upit = "select * from Korisnik where ID='$k' and Lozinka='$l'"
$r = mysqli_query("select * from Korisnik where uname='$u' and pass='$p'");
if (mysqli_num_rows($r) != 1) // Не постоји задати корисник
{
   // Корисник се поново враћа на формулар за пријављивање
   die(header('Location: prijavljivanje.php?greska=1'));
}
else // Пријављивање је успешно
{
   // Објекат се снима у меморију сесије
   $_SESSION['Korisnik'] = mysqli_fetch_object($r);
   // Корисник се преусмерава на апликацију
   die(header('Location: index.php'));
}
?>
```

*Листинг 3. Пример кода за проверу и пријављивање корисника*

У случају да број учитаних редова из табеле *Korisnik* није један, то значи да корисник није унео исправну комбинацију корисничког имена и лозинке. Тада се врши његово преусмеравање на страницу за пријављивање, са додатком параметра који иницира приказ поруке да пријављивање није успело. У противном, учитани запис се конвертује у *PHP* објекат који се снима у сесијски низ, а сам корисник се преусмерава на почетну страну апликације.




*Слика 3. Изглед формулара у случају неуспешног пријављивања*

За одјављивање корисника, односно прекид сесије је унутар приказаног решења потребно из сесијске меморије уклонити објекат у коме се налазе

параметри пријављеног корисника:

```
// Одјављивање корисника
unset($_SESSION['Korisnik']);
```

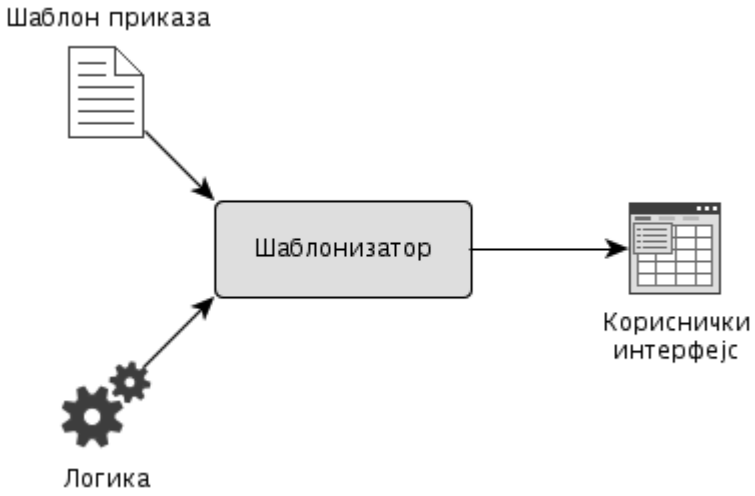
Након овога ће код којим се при покушају приступа проверава да ли је корисник пријављен:

```
if (!isset($_SESSION['Korisnik'])
   die(header('Location: prijavljivanje.php');
```

добити негативан резултат и преусмериће корисника на страницу за пријављивање.

## 4.5. Раздвајање логике и приказа

Максима „подели па владај“ посебно је применљива код развоја и одржавања већих решења за Веб. Један тип веома корисних алата за бољу организацију Веб апликације и њено једноставније проширивање чине тзв. шаблонизатори (енгл. *templating engine*).




*Слика 1. Шаблонизатори омогућавају раздвајање логике и приказа*

Шаблонизатори су софтверске компоненте које омогућавају да се дефиниција логике апликације чува одвојено од дефиниције приказа (корисничког

интерфејса), односно да не дође до њиховог мешања. На тај начин је поједностављено мењање различитих делова апликације и увођење нових функционалности, а омогућен је и паралелан рад програмера и радника на корисничком интерфејсу.

Један од најпопуларнијих шаблонизатора за PHP програмски језик је *Smarty*, јавно доступан на адреси *www.smarty.net*. Овај шаблонизатор је и сам развијен у *PHP* програмском језику у виду скупа класа које омогућавају парсовање шаблона приказа и њихово упаривање са резултатима извршавања дела апликације на слоју логике.

У наставку је дат пример коришћења *Smarty* шаблонизатора за раздвајање логике и дефиниције приказа апликације. Посматра се случај учитавања записа о студенатима из базе података. Целокупно решење се састоји од три фајла (не укључујући фајлове *Smarty* шаблонизатора):

1. фајл *Logika/Studenti.class.php* који представља логику апликације за захтевану акцију,
2. фајл *Prikaz/Studenti.Ucitavanje.tpl* који представља шаблон приказа апликације за захтевану акцију и
3. фајл *index.php* који прихвата захтев, позива одговарајућу акцију на слоју логике и резултате њеног извршавања упарује са одговарајућим шаблоном приказа.

```
<?php

class Logika__Studenti
{
   protected $DB;

   public function __construct()
   {
      $this->DB = new mysqli('localhost', 'korisnik', 'lozinka', 'baza');
   }

   public function Ucitavanje()
   {
      $Result = $this->DB->query('select * from Student');
```

```
      $Studenti = array();
      while ($Student = $Result->fetch_object())
         $Studenti[] = $Student;
      return $Studenti;
   }
}

?>
```

*Листинг 1. Код класе Studenti на слоју логике (фајл Logika/Studenti.class.php)*

```
<html>
   <head>
      <title>Преглед студената</title>
      <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
   </head>
   <body>
      <table border="1">
         <tr style="font-weight: bold; background-color: #DDDDDD;">
            <td>Индекс</td>
            <td>Име</td>
            <td>Презиме</td>
         </tr>
         {foreach $Studenti as $Student}
            <tr style="background-color: {cycle values='#FFFFFF,#EEEEEE'}">
               <td>{$Student->Indeks}</td>
               <td>{$Student->Ime}</td>
               <td>{$Student->Prezime}</td>
            </tr>
         {/foreach}
      </table>
   </body>
</html>
```

*Листинг 2. Код шаблона приказа (фајл Prikaz/Studenti.Ucitavanje.tpl)*

```
<?php

// Учитавање класе која садржи логику апликације
require_once('Logika/Studenti.class.php');

// Извршавање захтеване акције на слоју логике
$CStudenti = new Logika__Studenti();
$Studenti = $CStudenti->Ucitavanje();

// Учитавање, иницијализација и подешавање шаблонизатора
require('Smarty.class.php');
$smarty = new Smarty();
$smarty->setTemplateDir('Prikaz');

// Повезивање резултата слоја логике са одговарајућим шаблоном
$smarty->assign('Studenti', $Studenti);
$smarty->display('Studenti.Ucitavanje.tpl');

?>
```

*Листинг 3. Код контролера (фајл index.php)*

```
<html>
   <head>
      <title>Преглед студената</title>
      <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
   </head>
   <body>
      <table border="1">
         <tr style="font-weight: bold; background-color: #DDDDDD;">
            <td>Индекс</td>
            <td>Име</td>
            <td>Презиме</td>
         </tr>
```

```
         <tr style="background-color: #FFFFFF">
            <td>1/2013</td>
            <td>Петар</td>
            <td>Петровић</td>
         </tr>
         <tr style="background-color: #EEEEEE">
            <td>2/2013</td>
            <td>Ђорђе</td>
            <td>Ђорђевић</td>
         </tr>
         <tr style="background-color: #FFFFFF">
            <td>3/2013</td>
            <td>Лазар</td>
            <td>Лазаревић</td>
         </tr>
      </table>
   </body>
</html>
```

*Листинг 4. Резултат извршавања скрипта index.php*

| Индекс | Име | Презиме |
|---|---|---|
| 1/2013 | Петар | Петровић |
| 2/2013 | Ђорђе | Ђорђевић |
| 3/2013 | Лазар | Лазаревић |

*Слика 2. Рендеровани резултат извршавања скрипта index.php*

Шаблонизатор *Smarty*, као и шаблонизатори за *PHP* програмски језик уопште, понекад се критикују са аспекта да је сам *PHP* програмски језик већ довољан за раздвајање логике од приказа. Међутим, употреба шаблонизатора може знатно да поједностави фајлове са дефиницијом приказа а уједно чини и својеврсну универзалну конвенцију на том пољу. Из тог разлога велики број популарних пројеката заснованих на PHP програмском језику користи шаблонизаторе.

## 4.6. Рад са графичким садржајем

*PHP* програмски језик омогућава манипулацију графичким садржајем у различитим форматима (*GIF, PNG, JPEG* и многим другим) коришћењем више стандардних библиотека и проширења: *Cairo*, *Exif*, *GD, Gmagick* и *ImageMagick*. У наставку је дат пример креирања *PNG* слике са основним елементима коришћењем *GD* библиотеке:

```
<?php

// Прављење слике величине 300x300 пиксела
$image = imagecreate(300, 300);

// Одређивање RGB боја које ће се користити на слици
$crna   = imagecolorallocate($image, 0, 0, 0);
$crvena = imagecolorallocate($image, 255, 0, 0);
$zelena = imagecolorallocate($image, 0, 255, 0);
$plava  = imagecolorallocate($image, 0, 0, 255);
$zuta   = imagecolorallocate($image, 255, 255, 0);
$siva   = imagecolorallocate($image, 245, 245, 245);

// Бојење свих пиксела сивом бојом
imagefill($image, 0, 0, $siva);

// Исцртавање троугла путем три линије
imageline($image, 45, 240, 150, 60, $crvena);
imageline($image, 150, 60, 255, 240, $zelena);
imageline($image, 255, 240, 45, 240, $plava);

// Исцртавање правоугаоника и круга
imagefilledrectangle($image, 100, 150, 200, 239, $zuta);
imagefilledellipse($image, 150, 195, 80, 80, $plava);

// Испис текста на слици
imagettftext($image, 20, 0, 50, 120, $crna, 'DejaVuSans.ttf', 'Испис текста');
```

```
// Слање заглавља и садржаја слике (PNG формат)
header('Content-type: image/png');
imagepng($image);
```

Резултат извршавања кода дат је на следећој слици:




*Слика 1. PNG слика генерисана PHP програмским језиком*

Рад са графичким садржајем у *PHP* језику није ограничен само на креирање нових слика већ је могућа и измена постојећег садржаја. У наредном примеру ће бити приказан поступак стеганографског уметања садржаја у постојећу слику применом технике која користи бит најмањег значаја. Због једноставнијег и разумљивијег кода боје слика су преведене у палету сивих боја (осмобитна графика, односно 8 битова по пикселу). Вероватноћа измене сваког пиксела за једну нијансу сиве боје (1/256) је 50 процената, односно испод 0,2% укупно.

```
<?php

// Учитавање носеће слике и конвертовање у сиве нијансе
$imgn = imagecreatefrompng('nosilac.png');
```

```
imagefilter($imgn, IMG_FILTER_GRAYSCALE);
$imgn_sirina = imagesx($imgn);
$imgn_visina = imagesy($imgn);
$imgn_piksela = $imgn_sirina * $imgn_visina;

// Учитавање слике која се скрива и конвертовање у сиве нијансе
$imgs = imagecreatefrompng('stegano.png');
imagefilter($imgs, IMG_FILTER_GRAYSCALE);
$imgs_sirina = imagesx($imgs);
$imgs_visina = imagesy($imgs);
$imgs_piksela = $imgs_sirina * $imgs_visina;

//  Алоцирање свих 256 нијанси сиве боје
$siva = array();
for ($i=0; $i<256; $i++)
   $siva[$i] = imagecolorallocate ($imgn, $i, $i, $i);

// Провера да ли је могуће сместити скривену слику у носећу
if ($imgs_piksela * 8 > $imgn_piksela)
   die('Величина скривене слике је превелика.');

// Уметање скривене слике у носећу
$imgn_indeks = 0;
for ($i = 0; $i < $imgs_piksela; $i++)
{
   // Учитавање битова пиксела који се скрива
   $imgs_x = $i % $imgs_sirina;
   $imgs_y = floor($i / $imgs_sirina);
   $imgs_piksel = imagecolorat($imgs, $imgs_x, $imgs_y) & 255;
   $imgs_pikselb = str_pad(decbin($imgs_piksel), 8, 0, STR_PAD_LEFT);

   // Уметање битова скривеног пиксела у пикселе носеће слике
   for ($j = 0; $j < 8; $j++)
   {
```

```
      $imgn_x = $imgn_indeks % $imgn_sirina;
      $imgn_y = @floor($imgn_indeks / $imgn_sirina);
      $imgn_piksel = imagecolorat($imgn, $imgn_x, $imgn_y) & 255;
      $imgn_piksel += (int) $imgs_pikselb{$j} - $imgn_piksel % 2;
      imagesetpixel($imgn, $imgn_x, $imgn_y, $siva[$imgn_piksel]);
      $imgn_indeks++;
   }
}

// Приказивање резултата
header('Content-type: image/png');
imagepng($imgn);
```

Резултат извршавања кода, као и слике које су коришћене за добијање тог резултата, дати су на следећој слици:




*Слика 2. Резултат једноставне стеганографске функције*

На овом примеру је илустровано како се у PHP програмском језику може мењати постојећа слика. Додатно, приказани резултат јасно илуструје и могућности стеганографије с обзиром да се величина нове слике, односно слике са уметнутим скривеним објектом, не мења нити у бајтовима, ни у пикселима, а да је измене у приказу практично немогуће открити голим оком.

## 4.7. Рад са електронском поштом

Рад са електронском поштом подратумева слање електронске поште, коришћењем *SMTP* протокола, и њено преузимање са сервера, коришћењем *POP3* и *IMAP* протокола. Рад са *POP3* и *IMAP* протоколима у *PHP* језику знатно је сложеније, а само читање пристигле електронске поште се у пракси далеко ређе

користи од њеног слања. Слање електронске поште је саставни део већине савремених Веб апликације (потврда при регистрацији, обавештавање и слично) а у наставку су приказани основни начини за њега.

Два основна начина за слање електронске поште директно из *PHP* скриптова су коришћењем системске функције *mail* и коришћењем класа у којима је имплементиран *SMTP* протокол. Коришћење системске функције *mail* је једноставније али нуди скромне могућности када су специфична подешавања у питању:

```
$zaglavlje = 'From: ajevremovic@singidunum.ac.rs';
$za = 'mveinovic@singidunum.ac.rs';
$naslov = 'Порука из PHP-a';
$poruka =  "Поздрав, \n";
$poruka .= "ова порука је послата из PHP-a.\n";
$poruka .= "Александар";
if (!mail($za, $naslov, $poruka, $zaglavlje))
   die('Јавила се грешка приликом слања поруке');
else
   echo 'Порука је успешно послата.';
```

У наставку је дат пример слања поруке коришћењем *SMTP* протокола путем класе *PHPMailer*, слободно доступне са адресе *http://phpmailer.worxware.com*:

```
require 'PHPMailer/class.phpmailer.php';

$mail = new PHPMailer();
$mail->IsSMTP();

// Подешавање грешака које се приказују, 0-2
$mail->SMTPDebug  = 2;

// Подешавање параметара за приступ серверу
$mail->Host       = 'mail.singidunum.ac.rs';
$mail->Port       = 25;
$mail->SMTPAuth   = true;
```

```
$mail->Username   = 'ajevremovic@singidunum.ac.rs';
$mail->Password   = 'l0z1nka';

// Подешавање параметара и садржаја поруке
$mail->SetFrom('ajevremovic@singidunum.ac.rs', 'Александар Јевремовић');
$mail->AddAddress('mveinovic@singidunum.ac.rs', 'Младен Веиновић');
$mail->Subject = 'Порука из PHP-а';
$poruka =  "Поздрав, \n";
$poruka .= "ова порука је послата из PHP-а.\n";
$poruka .= "Александар";
$mail->AltBody = $poruka;
$mail->MsgHTML(nl2br($Mailer->AltBody));
$mail->AddAttachment('fajlovi/it.pdf');

// Покушај слања поруке
if(!$mail->Send())
   die("Грешка: " . $mail->ErrorInfo);
else
   echo "Порука је успешно послата.";
```

У оба случаја треба узети у обзир да се успешност код слања у оба случаја односи на успешно повезивање са *SMTP* сервером и успешно достављање поруке њему. Међутим, након тога се може десити више ствари (непостојећи корисник на одредишном серверу, прекорачење дозвољене величине поруке и слично) које ће онемогућити успешну испоруку поруке коначном примаоцу.

## 4.8. Серијализација података и Веб сервиси

Вредности променљивих основних типова (логичког, целобројног, децималног или текстуалног) могу се директно исписивати, складиштити или путем мреже размењивати са другим апликацијама (заснованим на PHP-у или некој другој технологији). Међутим, сложенији типови вредности - низови, матрице и објекти - не могу се директно репрезентовати јер имају сложенију структуру.

Да би се поменуте сложене вредности складиштиле и касније поново користиле (на пример, приликом снимања тренутног стања програма у циљу каснијег

настављања извршења) или да би се преко мреже пренеле у друго извршно окружење (сервисно оријентисана архитектура) потребно их је превести у једноставнију, линеарну репрезентацију. Овај процес се назива **серијализација** (енгл. *serialization*).

За потребе размене сложених променљивих између два просторно или временски удаљена PHP окружења могу се користити уграђене функције за ту намену. Уколико се ради о комуникацији између PHP-а и окружења заснованог на другој технологији (*Java*, *JavaScript* и слично) углавном се користе *XML*, *JSON* и *SOAP* системи за серијализацију.

## 4.8.1. Серијализација

За серијализацију и десеријализацију сложених типова променљивих у PHP-у користе се наменске функције *serialize* и *deserialize*:

```
$Student = new stdClass();
$Student->Indeks  = '1/2013';
$Student->Ime     = 'Петар';
$Student->Prezime = 'Петровић';
echo serialize($Student);
```

Резултат серијализације датог објекта који ће се исписати је:

```
O:8:"stdClass":3:
{s:6:"Indeks";s:6:"1/2013";s:3:"Ime";s:10:"Петар";s:7:"Prezime";s:16:"Петрови
ћ";}
```

Оно што треба запазити је да се прикиком серијализације снимају само атрибути објеката и веза ка класи чији је објекат инстанца, али не и имплементација саме класе и њених метода. Из тог разлога је и у окружењу у коме се врши десеријализација и коришћење десеријализованог објекта потребно имати декларисану класу чији је објекат инстанца. На пример, уколико у једном окружењу декларишемо класу са њеним методама, инстанцирамо је у виду једног објекта и серијализујемо:

```
class VeomaRetkaKlasa
{
   public function NekaFunkcija() {}
```

```
}
$Objekat = new VeomaRetkaKlasa();
echo serialize($Objekat);
```

а затим покушамо да резултат десеријализујемо у окружењу у коме класа *VeomaRetkaKlasa* није декларисана и позовемо њен метод:

```
$Objekat = deserialize('O:15:"VeomaRetkaKlasa":0:{}');
$Objekat->NekaFunkcija();
```

десеријализација ће бити делимично успешна (односно, објекат ће бити инстанца генеричке класе *__PHP_Incomplete_Class*) а позивање њених метода неће бити могуће.

Треба имати у виду да променљиве које су типа ресурс није могуће серијализовати. На пример, уколико објекат у себи садржи активну везу ка локалном серверу базе података, након његове серијализације и десеријализације у другом извшрном окружењу коришћење те везе неће бити могуће. Међутим, *PHP* дефинише специјалне методе *__sleep* и *__wakeup* који се аутоматски позивају пре серијализације, односно након десеријализације, а коришћењем којих је могуће дефинисати процедуре за исправно гашење и рестаурирање сложених атрибута:

```
class ObjekatSaVezomKaBazi
{
   protected $Host;
   protected $User;
   protected $Pass;
   protected $DB;
   protected $Link;

   public function __construct()
   {
      $this->Host = 'localhost';
      $this->User = 'korisnik';
      $this->Pass = 'lozinka';
      $this->DB   = 'baza';
```

```
    }

    public function Povezivanje()
    {
        $this->Link = mysqli_connect($this->Host, $this->User, $this->Pass,
            $this->DB);
    }

    public function RaskidVeze()
    {
        mysqli_close($this->Link);
    }

    public function __sleep()
    {
        $this->RaskidVeze();
        return array('Host', 'User', 'Pass', 'DB');
    }

    public function __wakeup()
    {
        $this->Povezivanje();
    }
}
```

Дакле, приликом покушаја серијализације датог објекта у њему ће се аутоматски иницирати раскид везе са сервером базе података а функцији за серијализацију ће бити дата листа атрибута које треба серијализовати. Са друге стране, у окружењу у коме се врши десеријализација објекта, аутоматски ће бити позвана функција за поновно повезивање са сервером базе података.

### 4.8.2. XML

Проширив језик за означавање (енгл. *eXtensible Markup Language, XML*) представља стандард за размену структуираних података између различитих софтверских окружења. Често се користи самостално, али и као носећи формат

за поруке протокола вишег нивоа какав је, на пример, *SOAP*. Вероватно најједноставнији начин за рад са *XML*-ом у PHP-у је коришћењем *SimpleXML* проширења.

У наставку је дат пример *XML* фајла, *studenti.xml*:

```
<?xml version="1.0" encoding="UTF-8"?>
<studenti>
   <student indeks="1/2013">
      <ime>Петар</ime>
      <prezime>Петровић</prezime>
   </student>
   <student indeks="2/2013">
      <ime>Лазар</ime>
      <prezime>Лазаревић</prezime>
   </student>
</studenti>
```

чије је парсовање од стране *PHP*-а остварено следећом наредбом:

```
$xml = simplexml_load_file($x);
```

Након учитавања и парсовања садржаја *XML* фајла променљива $*xml* је постала објекат путем кога можемо приступити елементима као атрибутима или елементима низова:

```
echo $xml->student[0]['indeks'];   // Исписује: 1/2013
echo $xml->student[1]->ime;        // Исписује: Лазар
```

Креирање *XML* докумената коришћењем PHP-а такође је веома једноставно. У наставку је дат пример креирања претходно приказаног *XML* садржаја коришћењем *DOM* проширења PHP-а које је нешто сложеније за рад од *SimpleXML*-а али нуди веће могућности.

```
$xml = new DOMDocument('1.0', 'UTF-8');
$xml->formatOutput = true;

$studenti = $xml->createElement('studenti');
```

```
$xml->appendChild($studenti);

$student = $xml->createElement('student');
$student->setAttribute('indeks', '1/2013');
$ime = $xml->createElement('ime');
$ime->appendChild($xml->createTextNode('Петар'));
$student->appendChild($ime);
$prezime = $xml->createElement('prezime');
$prezime->appendChild($xml->createTextNode('Петровић'));
$student->appendChild($prezime);
$studenti->appendChild($student);

$student = $xml->createElement('student');
$student->setAttribute('indeks', '2/2013');
$ime = $xml->createElement('ime');
$ime->appendChild($xml->createTextNode('Лазар'));
$student->appendChild($ime);
$prezime = $xml->createElement('prezime');
$prezime->appendChild($xml->createTextNode('Лазаревић'));
$student->appendChild($prezime);
$studenti->appendChild($student);

header('Content-type: text/xml');
echo $xml->saveXML();
```

### 4.8.3. JSON

Још један данас веома популаран начин за размену серијализованих података је коришћењем објектног записа *JavaScript* језика (енгл. *JavaScript Object Notation, JSON*). Поред *JavaScript*-а подршку за овакав тип серијализације поседује велики број популарних језика: *C, C++, C#, Java, Lisp, MatLab, Perl, Python, R* и многи други. Подржаност од стране самог *JavaScript*-а чини *JSON* веома популарних код развоја *AJAX* Веб апликација.

Један од основних разлога за популарност *JSON* дугује свом једноставном и меморијски ефикасном начину записивања вредности. На пример, следећи

објекат:

```
$Objekat = new stdClass();
$Objekat->CeoBroj = 123;
$Objekat->DecBroj = 123.456;
$Objekat->Logicka = true;
$Objekat->Tekst   = 'abcd1234';
$Objekat->Niz     = array(1,2,3);
echo json_encode($Objekat);
```

би након кодовања у JSON формат, коришћењем наведене функције *json_encode*, изгледао овако:

```
{
   "CeoBroj":123,
   "DecBroj":123.456,
   "Logicka":true,
   "Tekst":"abcd1234",
   "Niz":[1,2,3]
}
```

За декодовање *JSON* кодованих објеката користи се функција *json_decode*.

### 4.8.4. SOAP

Најозбиљнији протокол данас за коришћење у Веб апликацијама које користе сервисно оријентисану архитектуру, односно Веб сервисе, је *Simple Object Access Protocol, SOAP*. Овај омогућава размену података у оквиру *XML* порука а као носећи протокол најчешће користи *HTTP*, мада није ограничен на њега.

Протокол *SOAP* је сложенији од алтернатива као што је *JSON* али нуди већу дефинисаност порука и стандардизован позив функција у удаљеном окружењу. У наставку је дат пример *SOAP* поруке којом се у удаљеном окружењу позива функција за учитавање параметара студента чији је индекс *1/2013*:

```
<?xml version="1.0"?>
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
   <soap:Header>
```

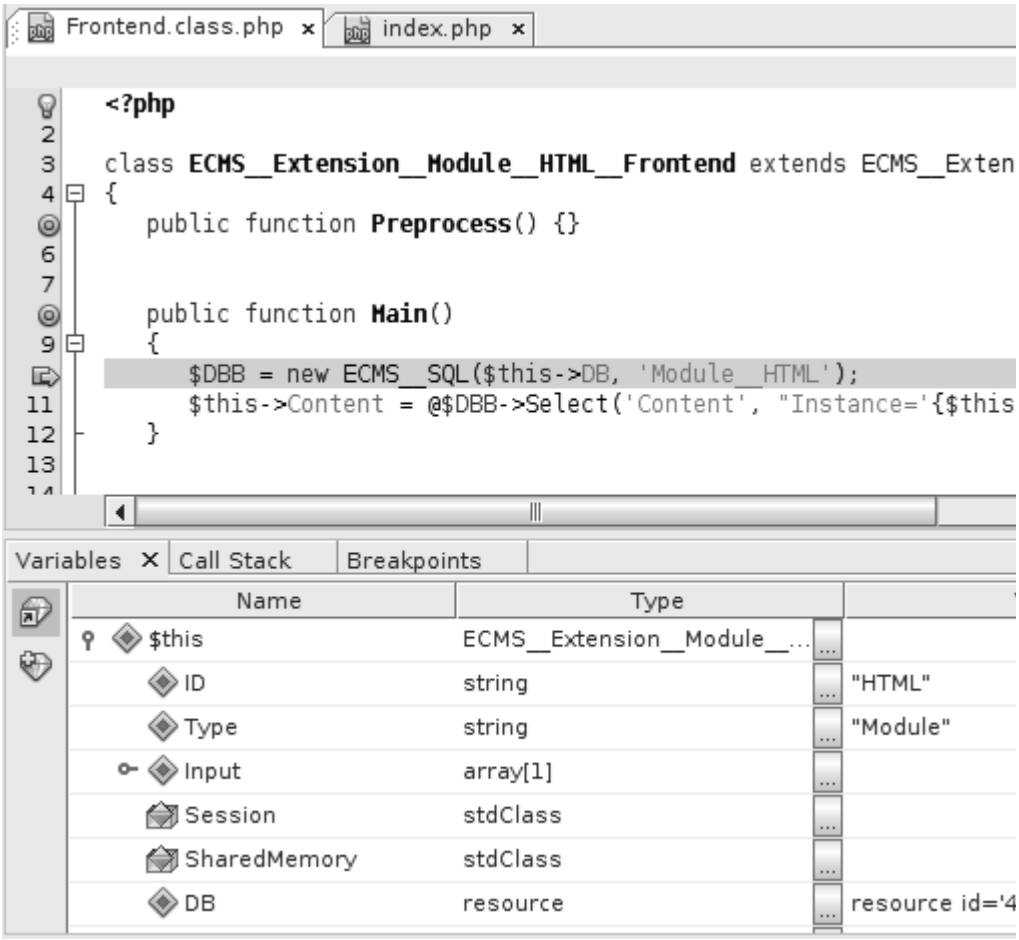
```
   </soap:Header>
   <soap:Body>
      <m:UcitavanjeStudenta>
         <m:BrIndeksa>1/2013</m:BrIndeksa>
      </m:UcitavanjeStudenta>
   </soap:Body>
</soap:Envelope>
```

Предност *SOAP* протокола се огледа и у сродним стандардима, какви су *Web Services Description Language* (*WSDL*) и *Universal Description Discovery and Integration* (*UDDI*) који нуде одређени ниво аутоматизације при коришћењу Веб сервиса. Такође, *SOAP* није уско везан за *PHP* тако да га је могуће користити за повезивање Веб апликација заснованих на *PHP*-у са Веб апликацијама заснованих на осталим технологијама.

## 4.9. Отклањање грешака и повећање перформанси

Отклањање грешака из малих и једноставних Веб апликација релативно је једноставан задатак за који нису потребни додатни алати и методи. Оно обично подразумева прегледање кода, уз евентуално додавање инструкција за исписивање неког параметра тренутног стања унутар секције у којој се претпоставља да се налази извор грешке.

Велике и сложене Веб апликације, са друге стране, као и апликације код којих није могуће отклањање грешака и уских грла ван реалног окружења у којем се оне интензивно користе, захтевају посебне алате и методе за ту активност.




*Слика 1. Праћење стања програма при контролисаном извршавању*

Један од најбољих алата за проналажење грешака и профилисање Веб апликација заснованих на *PHP* језику је проширење под називом *Xdebug*. Значајну карактеристику *Xdebug* проширења представља и могућност његовог повезивања са *NetBeans* развојним окружењем. То се постиже коришћењем *DBGP*-а, општег протокола за комуникацију између програмских језика и система који имају потребу за праћењем извршавања. Коришћењем ове могућности директно из интегрисаног развојног окружења може се контролисати извршавање скриптова и пратити стање променљивих. Осим тога, *Xdebug* омогућава и сакупљање података у за потребе профилисања Веб апликација.

## 4.9.1. Профилисање

Профилисање софтвера је динамичке анализе којом се утврђују његове одређене карактеристике: различити аспекти употребе меморије, утрошак времена од стране различитих функција, број позивања одређене функције и слично. Резултати ове анализе су драгоцени код процеса оптимизације одређеног кода јер се на основу њих може закључити на којим деловима кода ће ефекти оптимизације бити најзначајнији, односно које делове кода нема смисла оптимизовати јер ће резултати имати занемарљив допринос.

Процес профилисања у основи подразумева бележење различитих параметара током извршавања одређеног кода. Резултати овог процеса су текстуални фајлови који представљају дневнике извршавања:

```
==== NEW PROFILING FILE ============================================
version: 1
creator: xdebug 2.2.1
cmd: /home/ajevremovic/public_html/index.php
part: 1
positions: line
events: Time
fl=php:internal
fn=php::session_start
3 195
fl=php:internal
fn=php::spl_autoload_register
```

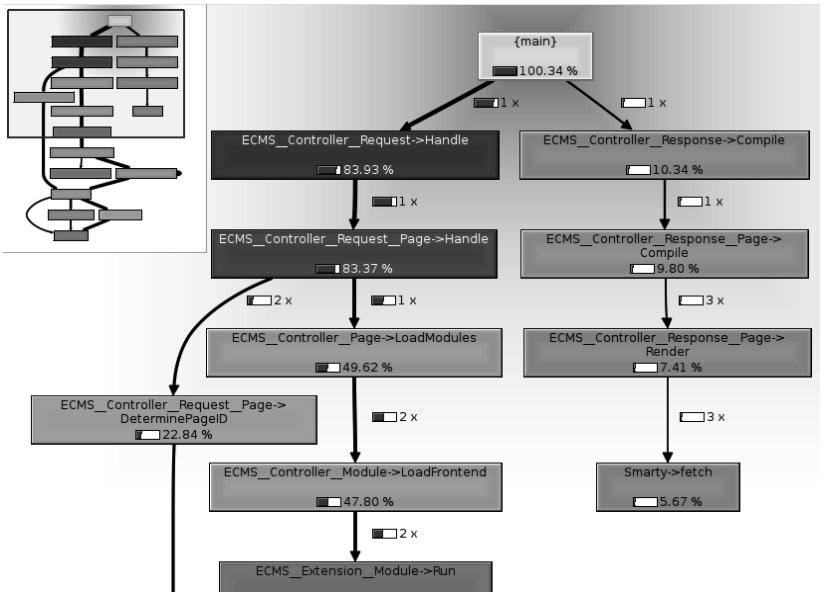
```
18 6181
fl=php:internal
fn=php::date_default_timezone_set
21 20
fl=php:internal
fn=php::mysql_connect
2 6044
fl=php:internal
fn=php::mysql_select_db
9 144
 fl=php:internal
fn=php::mysql_query
14 149
...
```

Из датог листинга се може видети редослед позивања функција, као и време потребно за њихово извршавање. Међутим, резултати у овом облику су тешки за анализу и извлачење закључака па се они углавном визуализују коришћењем одвојених алата. Најпопуларнији алати ове намене су *WinCacheGrind* (за *Windows* оперативни систем) и *KCacheGrind* (за Линукс оперативни систем).

| Incl. | Self | Distance | Calling | Callee |
|---|---|---|---|---|
| 83.65 | 0.08 | 1 | 1 | ECMS__Controller__Request->Handle (Request.class.php) |
| 83.09 | 0.44 | 2 | 1 | ECMS__Controller__Request__Page->Handle (Page.class.php) |
| 72.92 | 8.00 | 3-8 (8) | 37 | ECMS__SQL->Select (SQL.class.php) |
| 57.45 | 57.45 | 2-10 (10) | 133 | php::mysql_query (php:internal) |
| 49.45 | 0.20 | 3 | 1 | ECMS__Controller__Page->LoadModules (Page.class.php) |
| 47.64 | 1.76 | 4 | 2 | ECMS__Controller__Module->LoadFrontend (Module.class.php) |
| 42.63 | 0.11 | 5 | 2 | ECMS__Extension__Module->Run (Module.class.php) |
| 42.49 | 0.15 | 6 | 1 | ECMS__Extension__Module__Menu__Section2Tree__Frontend->Main (Frontend.class.p... |
| 39.75 | 6.36 | 4-9 (9) | 37 | ECMS__SQL->Describe (SQL.class.php) |
| 34.22 | 0.66 | 7 | 1 | ECMS__Controller__Page->LoadTree (Page.class.php) |
| 22.76 | 0.56 | 3 | 2 | ECMS__Controller__Request__Page->DeterminePageID (Page.class.php) |
| 16.18 | 0.64 | 4-9 (9) | 37 | ECMS__SQL->SetCharset (SQL.class.php) |
| 10.30 | 0.07 | 1 | 1 | ECMS__Controller__Response->Compile (Response.class.php) |
| 9.77 | 0.50 | 2 | 1 | ECMS__Controller__Response__Page->Compile (Page.class.php) |
| 7.38 | 0.54 | 3 | 3 | ECMS__Controller__Response__Page->Render (Page.class.php) |
| 7.23 | 0.17 | 3-7 (7) | 2 | ECMS__Controller__Page->DetermineRootID (Page.class.php) |
| 5.65 | 0.13 | 4 | 3 | Smarty->fetch (Smarty.class.php) |

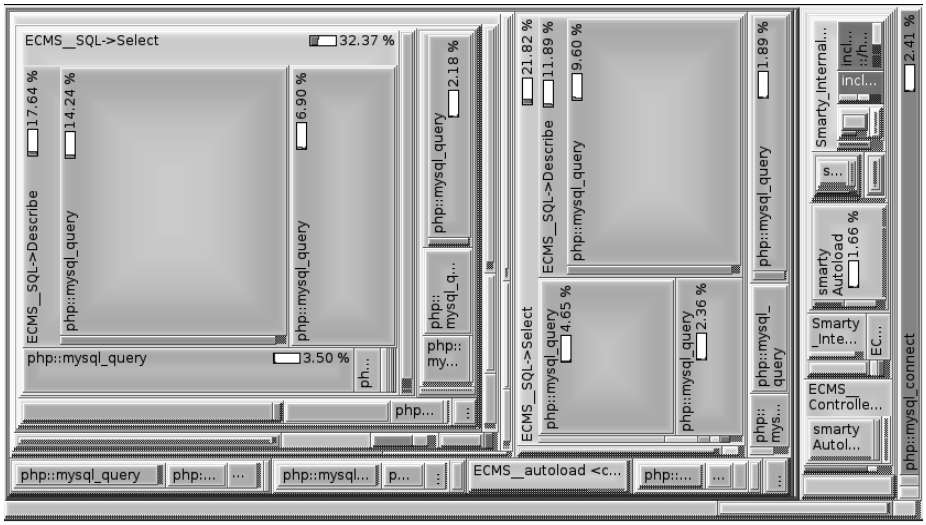
*Слика 1. Број позива и параметри извршавања функција*

Један тип визуализације добијених података приказан је на слици 1. У оквиру њега је дат број позива позивања одређене функције, као и остали параметри који су при том забележени. На слици 2. је приказано стабло позивања функција и самог извршавања програма.




*Слика 2. Стабло извршавања различитих функција*

Следећи тип визуализације, посебно значајан за потребе оптимизације решења, приказан је на слици 3. На њему је графички приказана расподела утрошеног времена на различите функције.




*Слика 3. Расподела утрошеног времена за извршавање различитих функција*

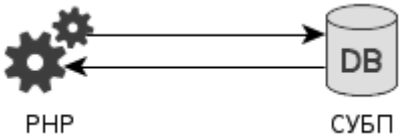
У конкрентом примеру може се уочити да је већина времена утрошена за

извршавање упита у бази података. Са друге стране, не могу се уочити кориснички дефинисане функције чијом би се оптимизацијом добили значајнији резултати. Дакле, за подизање перформанси Веб апликације приказане у примеру треба се усмерити на подизање перформанси сервера базе података и брзине комуникације између њега и Веб сервера. Додатно, треба проверити и да ли је могуће смањити број упита у бази података, као и да ли се они могу оптимизовати.

Једна од значајних карактеристика профилисања *PHP*-а је могућност да се оно врши у реалном, продукционом окружењу, односно током обраде реалних захтева. Међутим, треба имати у виду да профилисање значајно успорава извршавање кода па га треба користити опрезно и само у ситуацијама када су његови резултати потребни.

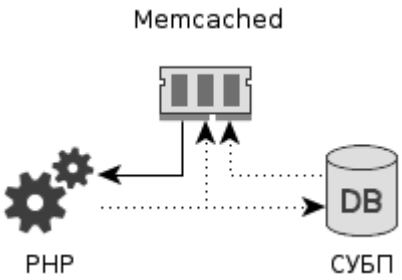
## 4.9.2. Кеширање података из базе података

Повезивање са сервером базе података, извршавање упита и обрада резултата на страни *PHP*-а временски су захтевни процеси који значајно могу да умање перформансе Веб апликација које се интензивно користе. Такође, може се приметити и да се код већине захтева изврши идентичан скуп упита који врати идентичне резултате. На пример, на Веб порталу са вестима који има 1.000 посета на сат времена, а код кога се у просеку појављује нова вест свакога сата, само код посете првог посетиоца након додавања вести ће упит из базе вратити оригиналан скуп вести, док ће се за упите код наредних 999 посета вратити већ виђени резултат.



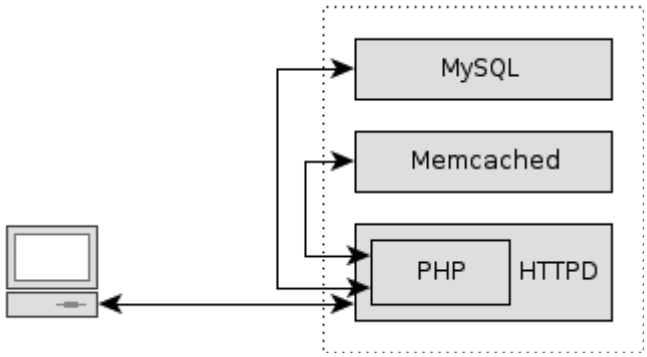

*Слика 1. Приступ подацима преко СУБП, без кеширања*

Овакво понављање отвара простор за подизање перформанси интензивно коришћених Веб апликација које на описани начин приступају бази података. На пример, уколико би се једном учитани скуп података снимио у брзу и лако приступачну меморију (кеш), наредни захтеви за тим подацима би могли одатле да се опслужују, без потребе за повезивањем на сервер базе података. Отворено питање остаје испорука застарелих података након измена у бази, али је оно лако решиво уколико се приликом измена такође ажурирају и подаци у кеш меморији.




*Слика 2. Приступ подацима преко СУБП са кеширањем помоћу Memcached-а*

Једно од најпопуларнијих решења за описани начин кеширања често коришћених података је *Memcached*. У питању је дистрибуирани систем за кеширање, развијен као серверска компонента клијент-сервер архитектуре која се подразумевано одазива на порту 11.211.




*Слика 3. Комуникација између процеса PHP-а, Memcached-а и MySQL-а*

Подаци се пре слања *Memcached*-у серијализују, подразумевано основном функцијом за серијализацију у *PHP*-у. Уколико постоји потреба да се исти подаци користе и од стране програма развијених у другим програмским језицима, потребно је користити систем за серијализацију који и ти језици подржавају (на пример *JSON*).

У наставку је дат пример кода којим се *Memcached* користи у комбинацији са *MySQL* системом за управљање базама података:

```
// Connection creation
$Memcache = new Memcache();

// Објекат који ће бити уписан/ишчитан из базе
```

```
$S = stdClass();
$S->Indeks  = '1/2013';
$S->Ime     = 'Петар';
$S->Prezime = 'Петровић';

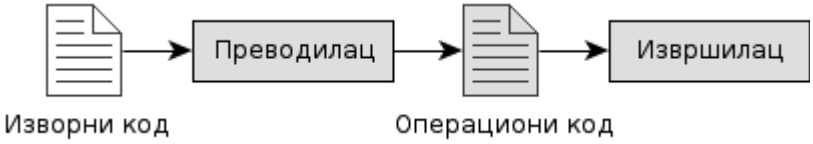
// Уписивање података у базу и мемкеш
$U = "insert into Student values ('{$S->Indeks}','{$S->Ime}','{$S->Prezime}";
$Rezultat = mysqli_query($DB, $U);
if ($Rezultat)
   $Memcache->set($S->Indeks, $S);

// Учитавање података из мемкеша или базе
$S = $Memcache->get('1/2013');
if (!$S)
{
   $U = "select * from Student where Indeks='1/2013'";
   $Rezultat = mysqli_query($DB, $U);
   $S = mysqli_fetch_object($Rezultat);
   $Memcache->set($S->Indeks, $S); // Снимање резултата у мемкеш
}
```

У приказаном примеру се подаци приликом уписа у базу уписују и у *Memcached*. Приликом учитавања података прво се проверава да ли они постоје у *Memcached*-у. Уколико постоје, нема потребе за приступом бази података. У противном, подаци се учитавају из базе али се и складиште у *Memcached* због наредних упита.

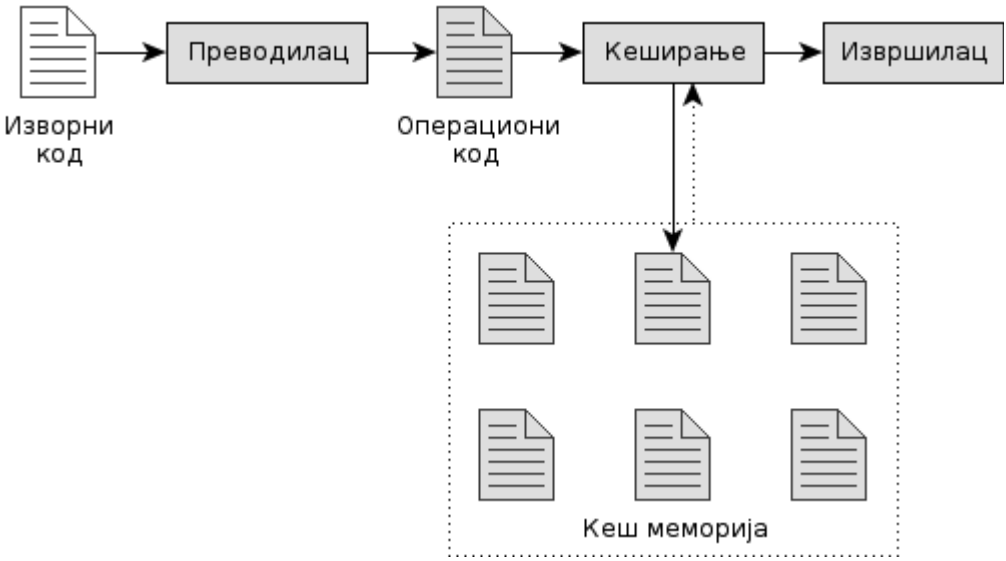
## 4.9.3. Кеширање операционог кода

Један од често критикованих недостатака *PHP*-а је потреба за превођењем изворног кода при сваком његовом извршавању. Такав приступ негативно утиче на перформансе јер се време за превођење изворног кода троши при свакој обради захтева клијената.



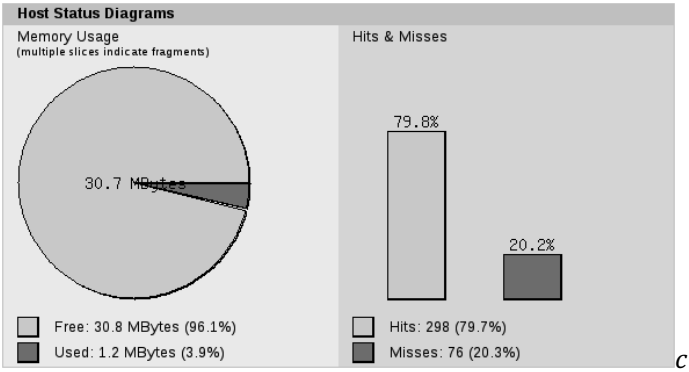

*Слика 1. Подразумеван процес извршавања PHP кода*

Решење за овај проблем се налази у кеширању операционог кода, за шта је развијено више различитих решења, од којих је вероватно најпопуларније проширење *PHP* језика под називом алтернативни *PHP* кеш (енгл. *Alternative PHP Cache, APC*). У питању је систем за подизање перформанси Веб апликација заснованих на *PHP*-у путем чувања једном добијеног операционог кода и његовог директног коришћења за будуће захтеве.




*Слика 2. Принцип рада кеширања операционог кода*

При подразумеваним подешавањима *APC* проширењу се додељује 32MB радне меморије у којој се чува операциони код као резултат превођења различитих *PHP* скриптова.




*Слика 3. Дијаграми заузећа меморије и ефикасности APC-а*

Уз сваки операциони код се чува и време последње измене скрипта на основу кога је он добијен. Разлог томе је ажурност операционих кодова, односно могућност да се препозна када је изворни код измењен и потребно је његово поновно превођење.

Побољшања перформанси због коришћења алтернативног *PHP* кеша знатно варирају, у зависности од сложености кода, односно односа времена потребног за превођење изворног кода и времена потребног да се тај код изврши. У неким примерима мерења показују побољшање перформанси и до 300%, као што је то случај код *Drupal CMS*-а[14].

## 4.9.4. Заштита изворног кода

Једна од честих замерки *PHP* језику је подразумевана испорука решења крајњим корисницима у виду изворног кода и његова употреба у продукционом окружењу. Такав приступ, осим што онемогућава заштиту интелектуалне својине аутора, отвара и одређене безбедносне проблеме, посебно у случају коришћења дељеног продукционог окружења, као и других типова продукционог окружења код којих постоји неповерење у администратора система.

За решавање наведеног проблема у пракси је развијено више приступа и решења. Најједноставнији, али уједно и најмање ефикасан приступ јесте прављење за људе неразумљиве варијанте изворног кода (енгл. *obfuscation*), путем непрегледног форматирања, замене назива променљивих, функција и слично, уз задршавање идентичне функције. Иако, на први поглед резултат ове операције може одвратити друге програмере да мењају и покушавају да разумеју на такав начин измењен изворни код, овај приступ не представља озбиљну препреку за озбиљнијег и посвећенијег нападача.

Други приступ за решавање проблема доступности изворног кода нуде тзв. енкодери. Софтверски алати овог типа раде на нивоу изворног или операционог кода, кога кодују у јавно непознат формат, односно, прецизније речено шифрују. У представнике овог типа решења спадају *Zend Guard, ionCube Encoder* и *SourceGuardian*. Међутим, треба имати у виду да ова решења не нуде висок ниво заштите с обзиром да се дешифровање заштићеног кода не одвија у безбедном окружењу. Аутори овог уџбеника су детаљније разматрање овог проблема изложили у раду под називом „*Using Cryptology Models for Protecting*

14 http://2bits.com/articles/benchmarking-drupal-with-php-op-code-caches-apc-eaccelerator-and-xcache-compared.html

*PHP Source Code*“ који је представљен на 11. међународној конференцији из области нумеричке анализе и примењене математике[15]. Још један важан недостатак енкодера је потреба да се купују њихове нове верзије при свакој новој верзији *PHP*-а у коју су унете значајније измене, као и да се изворни код поново шифрује и замењује у продукционом окружењу.

15 Aleksandar Jevremović, Nenad Ristić, Mladen Veinović, "Using Cryptology Models for Protecting PHP Source Code“ 3rd Symposium on Computer Languages, Implementations and Tools - 11th International Conference of Numerical Analysis and Applied Mathematics, ICNAAM2013, Rhodes, Greece, 2013.

# 5. Литература


1. http://www.internetworldstats.com
2. Internet World Stats http://www.internetworldstats.com
3. IMS Research, http://www.imsresearch.com
4. RFC 469 - Network Mail Meeting Summary
5. RFC 5322 - Internet Message Format
6. RFC 6531 - SMTP Extension for Internationalized Email
7. IBM Global Services, Improving Systems Availability, http://www.cs.cmu.edu/~priya/hawht.pdf
8. Шимић Г., Јевремовић А., „Развој система за проверу знања студената у оквирима постојећег ИС универзитета“, 15. телекомуникациони форум Телфор 2007., Београд, 2007., CD издање
9. http://blog.doh.ms/2013/01/04/we-are-all-php-evangelists/
10. http://w3techs.com/technologies/details/pl-php/all/all
11. http://arstechnica.com/business/2012/04/exclusive-a-behind-the-scenes-look-at-facebook-release-engineering/
12. http://en.wikipedia.org/wiki/Wikipedia:FAQ/Technical
13. http://www.php-fig.org/
14. http://2bits.com/articles/benchmarking-drupal-with-php-op-code-caches-apc-eaccelerator-and-xcache-compared.html
15. Aleksandar Jevremović, Nenad Ristić, Mladen Veinović, "Using Cryptology Models for Protecting PHP Source Code“ 3rd Symposium on Computer Languages, Implementations and Tools - 11th International Conference of Numerical Analysis and Applied Mathematics, ICNAAM2013, Rhodes, Greece, 2013.
16. Том ван Влек, Историја електонске поште, http://www.multicians.org/thvv/mail-history.html

CIP - Каталогизација у публикацији - Народна библиотека Србије, Београд

004.738.5

ЈЕВРЕМОВИЋ, Александар, 1983-
Интернет технологије / Александар Јевремовић, Младен Веиновић. - 4. изд. - Београд : Универзитет Сингидунум, 2020 (Београд : Caligraph). - V, 163 стр. : илустр. ; 24 cm

На врху насл. стр.: Факултет за информатику и рачунарство. - Тираж 800. - Библиографија: стр. 163.

ISBN 978-86-7912-648-1

1. Веиновић, Младен, 1962- [аутор]

а) Интернет

COBISS.SR-ID 282672140

© 2020.
Sva prava zadržana. Nijedan deo ove publikacije ne može biti reprodukovan u bilo kom vidu i putem bilo kog medija, u delovima ili celini bez prethodne pismene saglasnosti izdavača.

www.singidunum.ac.rs

9 788679 126481



Младен Веиновић
Александар Јевремовић

# ИНТЕРНЕТ ТЕХНОЛОГИЈЕ

Интернет програмирање и развој Веб апликација данас представљају неке од најперспективнијих специјализација у области информатике. Циљ овог уџбеника је да студенте упозна са архитектуром и протоколима Интернета и Веба, као и технологијама на страни клијента и сервера, односно да их оспособи за развој Веб презентација и сервиса по актуелним стандардима.

Приликом писања ове књиге трудили смо се да наша знања и искуства организујемо и представимо тако да буду разумљива потпуним почетницима у овој области, али и да им пруже јаку основу за будући професионални рад и даље усавршавање. Изабрани приступ је резултат нашег вишегодишњег искуства у раду са студентима на новом пољу, а посебно нам је задовољство што је велики број њих, често и пре дипломирања, пронашао запослење баш у овој области и остварује значајне професионалне резултате.

Универзитет
Сингидунум