Факултет инжењерских наука Универзитета у Крагујевцу

Марија С. Ракић

Веб апликација за издавање лекарских рецепата

мастер рад

Крагујевац, 2014.

Факултет инжењерских наука Универзитета у Крагујевцу

Назив студијског програма: Информатика у инжењерству
Ниво студија: Мастер академске студије
Модул: M_7
Предмет: Пројектовање информационих система и база података
Број индекса: 402/2011

Марија С. Ракић

Веб апликација за издавање лекарских рецепата

мастер рад

Комисија за преглед и одбрану:

1. **Др Милан Ерић - ментор**
2. _____ Датум одбране:_____
3. _____ Оцена:_____

У оквиру мастер рада са темом "*Веб апликација за издавање лекарских рецепата***" кандидат треба да:**

За реалан систем (издавање лекарских рецепата) изврши развој веб апликације помоћу које се штампају подаци на рецепту, а који се налазе у бази података. Сам интерфејс треба да се састоји од делова за унос података у базу, приказ одређених података из базе и опције за штампу података које чине један лекарски рецепт. Потребно је описати како се креира једна релациона база података, како се из PHP програмског језика позива та база, како се помоћу MySQL-ових упита манипулише базом података, и на крају описати само софрверско решење. Рад треба да обухвати уводна теоријска разматрања везана за развој веб софтверских система и база података, опис реалног система, коришћене технологије, дијаграме тока података, логички и физички модел база података као и опис развијеног софтвера. На крају дати закључна разматрања.

Препоручена литература:

1. Полишчук Ј.: **Пројектовање информационих система**, Електротехнички факултет, Подгорица, 2007.
2. Rainer K., Turban E.: **Introduction to Information Systems**, John Wiley & Sons, Inc., 2009
3. Вељовић А.: **Пројектовање информационих система**, Компјутер библиотека, 2003.
4. Williams H., Lane D.: **Web Database Applications with PHP & MySQL**, O'Reilly, 2004.
5. Лазаревић Б., Марјановић З., Аничић Н., Бабарогић С.: **Базе података**, ФОН Београд, Београд 2003.

Крагујевац, јун 2014.

ментор:
Др Милан Д. Ерић

_____

Изјава о самосталној изради рада

Изјављујем да сам овај мастер рад урадла самостално, служећи се стеченим знањем и искуством као и наведеном литературом.

402/2011 Марија Ракић

_____

(потпис)

Резиме рада

Сврха овог рада је садржана у теми, а то је израда веб апликације за чување свих података једне здравствене установе и штампање рецепата. Идеја за овај рад је настала у здравственој установи, где хронични болесници сваког месеца морају да дођу код лекара по рецепте за лекове које годинама користе, чиме се у лекарским ординацијама ствара гужва. Применом савремених технологија могуће је картон пацијента водити рачунарски, тј.чувати у бази података. Ако терапија пацијенту није промењена, могуће је штампање рецепата директно из базе, чиме се убрзава процес и смањују гужве у ординацијама. Како постоји више ординација једне здравствене установе, искоришћен је интернет као сервер, да би се информације чувале у јединственој бази података.

У овом раду су коришћене следеће технологије: програмски језици PHP и JavaScript, презентацијски језик HTML, MySQL систем за управљање базама података и програм који симулира рад интернета на рачунару ван мреже WAMP.

Шифарници лекова, дијагноза, држава, ослобађања од партиципације су добијене у Апотекарској установи Крагујевац.

Постигнуто је да се сви подаци једне здравствене установе чувају у јединственој бази података, и да се рецепти штампају из базе.

Кључне речи

PHP, JavaScript, MySQL, HTML, базе података, пацијент, лекар, рецепт

The summary of work

Purpose of this work is to develop veb application to store all data of medical clinic and print perscriptions for the pacient directly from database. Idea for this work originated in the medical clinic where chronic patients have to go to see a doctor every month for their prescriptions for drugs that that they use for years. This made crowd in clinics. By using a modern technology, it is possible to keep the patient's medical data in the database. If the therapy of the patient is not changed, it is possible to print the perscriptions directly from the database, thus speeding up the process and reduce congestion in clinics. Since there are a lot of clinics, Internet is used as a server to keep informations in universal database.

Following technologies are used in this work: programming languages PHP and JavaScript, presentational language HTML, MySQL relational database management system and WAMP.

Code lists for medications, diagnoses, states, exemptions from co-payments are received in the Pharmaceutical Institute Kragujevac.

It is achieved that all the data from medical clinic is stored in a single database, and the recipes can be printed directly from the database.

Key words

PHP, JavaScript, MySQL, HTML, database, pacient, doctor, perscription

Садржај

1. Увод

Појава и развој рачунара и развој информационих технологија сигурно су у највећој мери потпомогле, не само технолошки развој, већ и развој у свим областима човековог деловања. Може слободно да се каже да је појава интернета довела до револуције у многим сферама људског рада. Информације, њихова доступност и брзина размене представљају основу успешног пословања савремених предузећа, а интернет у том контексту представља савршено средство за проналажење и дистрибуцију информација. Сложене дистрибуиране веб апликације постале су неопходне за опстанак и развој савремених организација и предузећа.

Идеја за овај рад је настала у здравственој установи, где хронични болесници сваког месеца морају да дођу код лекара по рецепте за лекове које годинама користе, чиме се у лекарским ординацијама ствара гужва.

Циљ овог рада је да се применом савремених технологија картон пацијента води рачунарски, тј. да се чува у бази података. Ако терапија пацијенту није промењена, потребно је штампати рецепт директно из базе, чиме се убрзава процес и смањују гужве у ординацијама. Како постоји више ординација једне здравствене установе, искоришћен је интернет као сервер, да би се информације чувале у јединственој бази података.

На почетку рада описана је архитектура софтверских система, као и фазе, методе и алати које се користе у развоју информационих система.

Затим је описан реалан систем, дате су процедуре за издавање лекарских рецепата и информациони захтеви које треба задовољити.

Представљене су и значајне карактеристике коришћених технологија, и то: програмских језика PHP и JavaScript, презентацијског језика HTML, као и MySQL система за управљање базама података, као и програма који симулира рад интернета на рачунару ван мреже WAMP.

У петом поглављу описане су базе података. Коришћене су релационе базе података, које су постале својеврсни стандард у домену перзистенције (чувања) података.

У последњем поглављу дат је веб сајт који је развијен коришћењем горе поменутих технологија. Веб сајт се налази на адреси http://stamparecepata.xtreemhost.com/. Такође, приказан је одштампан рецепт, као коначан резултат рада.

2. Архитектура и развој софтверских система (модел пројектовања)

2.1. Једнослојна архитектура (Single Tier)

Аутоматизација пословања почела је увођењем огромних централних (маинфраме) рачунара који су опслуживали велики број корисника. Сви ресурси система (магнетне траке, дискови, штампачи) били су директно повезани на њих. Овим рачунарима приступало се помоћу такозваних "глупих" терминала. У оваквим системима сав посао процесирања (извршавање програма, обрада података, комуникација са терминалима) обављао је централни рачунар. Велики недостатак ове архитектуре, са аспекта хардвера, огледа се у комуникацији између терминала и сервера: Сервер прати притиске тастера терминала, интерпретира их и враћа адекватан одговор. На овај начин се, поред података који се обрађују, рачунарском мрежом шаљу и делови корисничког интерфејса апликације која се привидно извршава на терминалу.

Апликације за ове рачунаре писане су наменски за решавање конкретних проблема. Њихове компоненте (кориснички интерфејс, пословна логика и структура података) се преплићу и представљају једну нераздвојиву целину. Због тога се оваква архитектура софтвера назива једнослојном.

Преплитање компоненти софтвера проузрокује два проблема карактеристична за ову архитектуру:

- немогућност поновног коришћења делова кода претходно написаних програма
- немогућност измене једне компоненте апликације независно од осталих.

Једнослојну архитектуру карактерише једноставност развоја и испоруке софтверских производа са једне стране, и изузетно висока цена имплементације хардверских решења са друге.

Увођењем персоналних рачунара у пословну примену настала је потреба за новим хардверским и софтверским моделом који би омогућио да се подаци ефикасно деле између њих. За те потребе развијена је клијент-сервер архитектура.

Данас кад се говори о архитектури великих пословних апликација морамо разликовати физичку и логичку архитектуру апликације. Физички се апликација најчешће испоручује двослојно, базни сервер – дебели клијент, или трослојно, базни сервер – апликацијски сервер – танки клијент. Број физичких слојева зависи о величини система и функцијама које систем мора обављати. Приликом физичке изградње система треба размишљати о комуникацији између делова система, и о дистрибуцији нових верзија апликација, извештаја и модула што директно зависи од физичке имплементације решења [13].

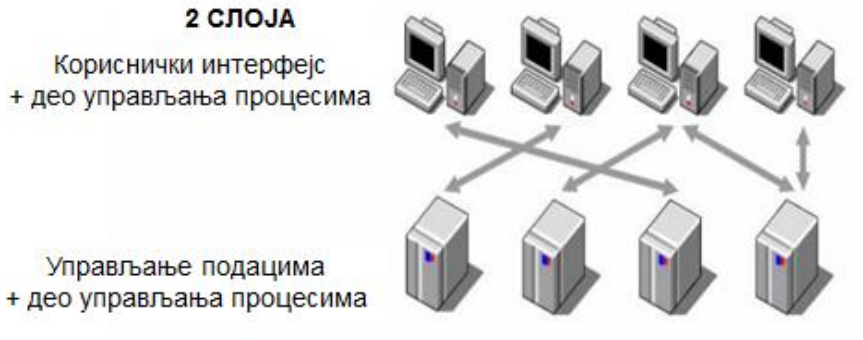
2.2. Клијент сервер системи

Клијент-сервер модел обраде података је модел дистрибуиране обраде података, где се функције једног корисничког програма расподељују на најмање два процеса који међусобно комуницирају, и то клијентски и серверски. Клијентски процес шаље поруку серверском процесу, тражећи одређену услугу, а серверски процес извршава одређени задатаки шаље поруку клијентском процесу о резултату (успеху или неуспеху) извршеног задатка.

Клијентски и серверски процеси су специјализовани за извршавање одређених задатака и увек комуницирају они делови процеса који су надлежни за захтевани задатак. Оба процеса се могу извршавати на једном рачунару, али пун опсег могућности се реализује дистрибуирањем ових процеса на велики број физички дислоцираних рачунара. Тада је рачунар који извршава клијентски део процеса клијент-рачунар, или краће клијент, а по аналогији, исто важи и за сервер.

2.3. Двослојна архитектура клијент-сервер система (Dual Tier)

У пракси се реализују различите архитектуре клијент-сервер система. Најједноставнији систем чине један клијент и један сервер, а сложенији системи укључују више клијената и више сервера истовремено. Ако се расподела задатака обавља на два нивоа, онда се ради о двослојној архитектури, код које независно од броја клијената и сервера постоје само два слоја: слој базе података и клијентски (кориснички) слој.



Слика 1. Двослојна архитектура клијент-сервер система

Основна одлика двослојне клијент-сервер архитектуре је та што базу података чини независном од апликација, јер интегритет базе података обезбеђује њену независност од апликација које над њом раде. Једини механизам потребан за комуникацију апликације са базом података је SQL упит који се прослеђује директно или кроз неку рачунарску мрежу. Ипак значајан део апликације се везује за клијент, што проузрокује основне недостатке овакве архитектуре. Одржавање система везаних за клијент је скупо, због честих

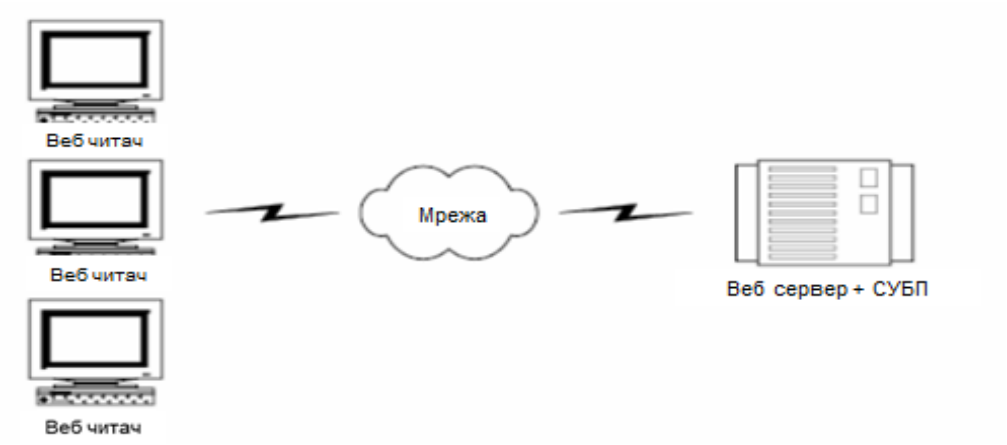
измена у технологији корисничких интерфејса. Софтвер развијен у оваквој архитектури је недовољно модуларан и тешко преносив. Системи развијени у двослојној архитектури незадовољавајуће подржавају велики број корисника, велики обим трансакција и непредвидиво радно оптерећење за нове интернет/интранет апликације. Ова ограничења су условила развој трослојног модела, који се данас широко користи у пројектовању дистрибуираних информационих система.

Предности оваквог модела обраде података су:

- централизовано управљање ресурсима система
- једноставније обезбеђивање сигурности података.

Основни проблем је недостатак скалабилности.

- Под скалабилношћу се подразумева особина система да омогући ефикасан рад великом броју корисника, и да даље повећавање броја корисника не изазива драстичан пад перформанси система.



Слика 2. Скица двослојног система заснованог на WWW технологијама [13]

2.4. **Трослојна архитектура (Three Tier)**

Апликације су подељене на три логички независна слоја који међусобно комуницирају посредством интерфејса. Први слој, најближи кориснику, назива се слој презентације и садржи кориснички интерфејс апликације, средњи слој садржи пословну логику система, док трећи слој представља слој података.

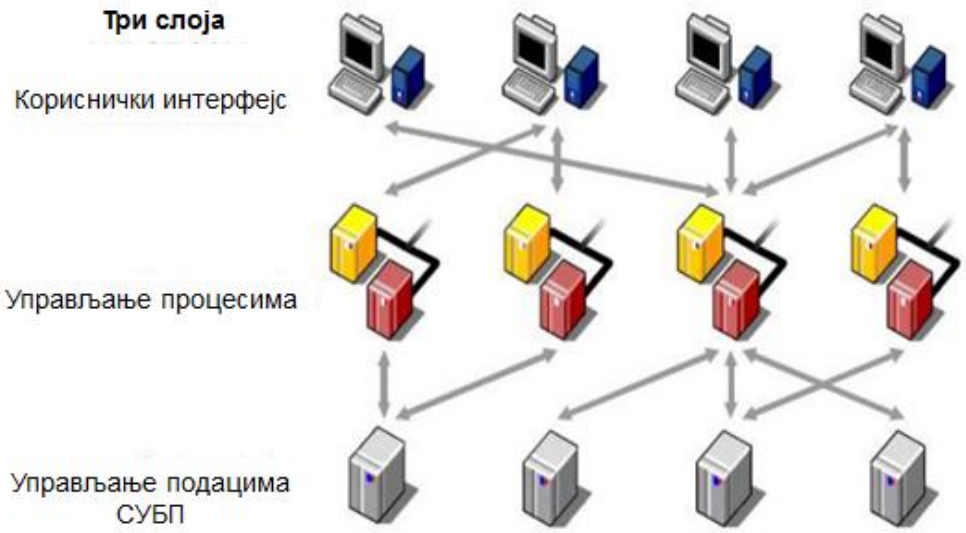
Трослојне апликације за извршавање користе сервер апликације као посредника између клијентских рачунара и сервера базе података. Корисник посредством слоја презентације упућује захтев пословној логици за подацима.

Слој пословне логике тражене податке прибавља од слоја података и шаље их слоју презентације који их по пријему форматира и приказује кориснику.

Слој података може садржати различите изворе информација потребне за извршавање апликације као што су системи за руковање релационим базама података (MS SQL сервер, Oracle, MySQL), XML документи, веб сервиси и сл.

За разлику од клијент-сервер архитектуре, слој пословне логике трослојних апликација потпуно је независан од других слојева. Састоји се из:

- скупа објеката који описују оперативну структуру података апликације,
- скупа објеката који садрже операције којима се реализују пословне функције система, и
- скупа објеката који садрже операције за комуникацију са слојем података.



Слика 3. Трослојна архитектура клијент-сервер система

Предности трослојне архитектуре:

- Трослојни концепт је довео до поделе програмског кода на сегменте који имплементирају тачно одређене функције система. Тако организован систем је једноставнији за одржавање, јер је могуће независно развијати кориснички интерфејс и логику апликације.
- Важна карактеристика трослојних система је скалабилност:
    a) повећавање броја клијената је једноставно
    b) повећавање пропусне моћи и брзине одзива сервера средњег слоја је могуће кроз додавање нових серверских машина уз коришћење постојећих.
- Систем са више сервера карактерише и повећана поузданост и флексибилност
- Логика апликације се може мењати и у току рада система.

- Могуће је ефикасно вршити балансирање оптерећења серверског подсистема.
- Трослојне архитектуре система подразумевају ослањање на стандарде у одговарајућим областима, засноване на Интернет технологијама. Ослањање на стандарде омогућава интеграцију система хетерогених у погледу коришћене хардверске и софтверске опреме.

Даљим проширивањем концепта трослојних система долази се до појма вишеслојних система (Multi Tier), где се врши даља подела на компоненте у оквиру средњег слоја са циљем још већег повећања скалабилности, односно перформанси [13].

2.5. Фазе, методе и алати које се користе у развоју информационих система

Стратешко планирање развоја информационих система почиње снимањем стања (иницијална стратегија). Ово доприноси одређивању пословних циљева, идентификовању проблема и идеја, одређивању начина њиховог решавања и дефинисању захтева који се постављају пред систем. Садржи студију изводљивости, односно прегледну анализу проблемског подручја и одређивање граница пројеката. Планирање пројекта се састоји од израде плана рада, одређивања кадрова, управљање и надзор пројекта. Пословни циљ је израда главног пројекта.

Анализа система и захтева корисника обухвата прецизирање граница пројекта и пословни захтеви. Врши се детаљна анализа постојећег система, проблема и пословних захтева, детаљна анализа понашања реалног система и корисничких захтева. Спецификација захтева је детаљни опис захтева, обликован тако да га разумеју пројектанти и крајњи корисници.

У **спецификацији апликација** се израђују детаљне функционалне спецификације програма који, за сваки идентификовани програм, дефинише шта он треба да ради, односно које излазе треба да генерише, на бази датих улаза и стања система.

Израда концептуалног модела која представља целокупан информациони садржај базе података. Концептуални модел је модел реалног система у коме је систем описан као скуп објеката, њихових атрибута и њихових међусобних веза. У неким приступима се концептуални модел изграђује директно, на бази пословног модела система, а објекти концептуалног модела служе за спецификацију апликација.

Логичко пројектовање базе података у датом СУБП се своди на трансформацију концептуалног модела у опис базе података конкретног Система за управљање базом података као што је на пример SQL сервер.

Логички модел базе података дефинише структуре погодне за развој апликација. Најчешће се ова трансформација врши коришћењем CASE алата у коме се концептуални модел развија. Пројектовање логичког модела база података се врши техникама моделовања података (нпр.модел објекти везе).

Физичко пројектовање базе података је пројектовање интерне, физичке структуре базе података, на основу логичког модела и спецификација свих апликација и нефункционалних спецификација. Физичка структура базе података треба да омогући остваривање задовољавајућих перформанси система. Кључна разлика између логичког и физичког пројектовања је у томе што је физичко зависно од имплементационог окружења, а логичко није.

Пројектовање апликација се обавља на основу спецификације апликација и логичког модела базе података у конкретном СУБП-у. Пројектовање апликација, односно дизајн (моделовање), представља поглед пројектанта на будући информациони систем. Служи за доношење одлуке о томе како градити систем. Садржи дизајн потребних решења. Детаљни дизајн представља разраду решења, односно израду технолошког модела информационог система (поглед извођача). Пројектовање првенствено обухвата трансформацију модела добијених у анализи система у одговарајуће имплементационе системе.

Имплементација и подешавање базе података врши се имплементирањем у конкретном СУБП-у, а затим се прате перформансе система, измене у апликацијама и логичкој структури базе и врши одговарајуће подешавање физичке структуре. Имплементација апликација подразумева програмирање, односно кодирање. Дакле, имплементација подразумева изградњу базе података, кодирање (програмирање) процеса (функција) новог информационог система, изградња софтверских компоненти, размештање компоненти у дистрибуираном имплементационом окружењу, а врши се након избора имплементационе платформе.

Тестирање је провера уграђених компоненти. Интеграција и провера система је удруживање делова и провера целине, да би се доказало да систем ради (да је исправно направљен), као и да ради оно што је захтевано (да је направљен прави систем који испуњава захтеве корисника).

Увођење са одржавањем и модификација система представља преношење радних активности и конверзија података са старог на нови систем, инсталацију опреме, апликација и база података новог система, иницијални унос података, успостављање система заштите и одржавања као и обуку коришћења новог информационог система. Одржавање се састоји од измена система ради побољшања његових радних карактеристика (перформанси), побољшања или прилагођавања начина употребе. Одржавање подразумева и подршку добављача опреме, помоћ техничког особља корисницима

информационог система у току његове употребе, као и израду плана одржавања. Одржавање апликација подразумева промене у апликацијама, а понекад и у логичком моделу базе података до којих долази због откривених грешака, промена у имплементационом окружењу или промене захтева корисника. Одржавање је трајна активност која започиње одмах након увођења система у примену. Без обзира како је добро систем дизајниран, конструисан и тестиран, грешке ће се неизбежно појавити. Исправљање грешака у пракси се назива одржавањем система. Одржавање само по себи не подразумева уградњу побољшања или нових могућности, али се она уобичајено спроводе. Одржавање система може бити превентивно- оно подразумева заштиту од могућих проблема. Потребно је вршити редовну израду сигурносних копија (backup). Под коративним одржавањем се подразумева поправка након што се проблем појавио. Врши се враћање података из сигурносне копије (restore) или уклањање узрока грешке (исправљање апликације). Адаптивно одржавање је прилагођавање функционалности (начина послуживања), прилагођавања структуре (промене структуре података) или побољшање перформанси (оптимизација програма). Перфективно одржавање је надградња система да би се решили нови проблеми или уградња нових могућности. Током примене и одржавања обавља се анализа додатних захтева, планирање и припрема активности које следе, те тако започиње нови циклус развоја.

Нови развојни циклус се спроводи након преиспитивања читавог система и констатације да су потребне веће измене услед промена у пословању или промена пословних циљева. Нови развојни циклус, најчешће, представља нови пројекат [12].

3. Реалан систем

3.1. Процедуре за издавање лекарских рецепата

Лекове могу прописивати само доктори медицине и доктори стоматологије (лекари). Могу се прописивати само они лекови који се стављају у промет који су законом дозвољени. У апотекама и другим здравственим установама које имају организовану службу руковања лековима лекове може издавати само лице које је савезним прописима за то овлашћено.

Лекови се прописују и издају на лекарски рецепт. Одређени лекови могу се издавати без рецепта. Једним рецептом може се прописати само један лек и само за једно лице.

Рецепт мора да садржи:

1) назив лека (и шифру лека)

2) фармацеутски облик лека

3) количину лека

4) начин употребе лека

5) шифру дијагнозе

6) потпис лекара са идентификационим бројем

7) датум издавања рецепта

8) име, презиме и адресу корисника лека

9) матични број.

Слика 4. Попуњен лекарски рецепт

На рецепту који се издаје за лице млађе од 15 година морају бити назначене године живота, а за лице млађе од једне године живота назначени дан, месец и година рођења.

Ако се рецепт издаје у здравственој установи, на заглављу рецепта мора бити одштампан или штамбиљом утиснут назив здравствене установе, а ако се рецепт издаје ван здравствене установе, морају бити одштампани или штамбиљом утиснути име, презиме и адреса лекара (место, улица и број), као и број телефона. На рецепту који се издаје у здравственој установи мора бити утиснуто име и презиме лекара који издаје рецепт.

Ако се при изради лека мора мерити његова маса, лекар на рецепту мора означити масу у грамима, и то арапским бројевима односно римским бројевима ако се ради о капима или другим јединицама које се не могу изразити грамима.

Ако се прописује готов лек који се у промету налази у разним облицима, величинама, односно јачинама, лекар мора на рецепту означити облик, величину, односно јачину лека.

Ако лекар намерава да пропише дозу већу од максималне дозе која је одређена националном фармакопејом или другим прописима, дужан је да такву дозу назначи и речима и да поред те ознаке стави знак узвика (!), а уз њега свој потпис.

Апотеке, односно здравствене установе по правилу, издају лекове на основу рецепта и то најкасније седам дана од дана прописивања.

3.2. Кратак опис постојећег система

У амбулантама за здравствену заштиту, преко изабраног лекара пружа се примарна медицинска заштита која обухвата превентивне и куративне облике здравствене заштите. У овим здравственим јединицама се ради на заштити и унапређењу здравља, спречавању и раном откривању болести, лечењу, превентивној здравственој заштити, групацији становништва са повећаним ризиком обољења у складу са посебним програмима превентивне здравствене заштите, здравственом васпитању и саветовању за очување и унапређење здравља.

Организатори непосредног процеса рада су начелници одељења и одељенске сестре. Служба за здравствену заштиту у свом саставу има много здравствених јединица. Свака здравствена јединица (амбуланта) има по један компјутер где се уносе подаци о заказивању прегледа. Амбуланте нису умрежене интранетом, али имају приступ интернету. Сви прегледи пацијента воде се кроз картон, у коме се чувају сви подаци физички, и нема електронског вођења картона. На књижици пацијента је уписан број картона и изабрани лекар, па када се пацијент јави на заказани преглед, сестра на основу података са књижице нађе картон и однесе га лекару. Лекар прозива пацијента и прегледа га или поприча са њим о тегобама, и преписује одговарајућу терапију, коју уписује у картон пацијента и попуњава рецепте. Ако је пацијент хронични болесник и има терапију која се не мења годинама, лекар само попуни рецепте и изда их пацијенту. Како се на преглед код изабранног лекара не јављају само

пацијенти који имају заказано, већ и они којима је лоше, па им је хитно потребан лекар, стварају се гужве у чекаоницама.

Да би се унапредио систем рада и убрзао процес издавања лекова, потребан је информациони систем којим би се обухватили сви подаци који се воде на физичким картонима у здравственим установама.

3.3. Информациони захтеви који треба да буду задовољени

Применом савремених технологија могуће је картон пацијента водити рачунарски, тј.чувати у бази података. Ако терапија пацијенту није промењена, могуће је штампање рецепата директно из базе, чиме се убрзава процес и смањују гужве у ординацијама. Како постоји више ординација једне здравствене установе, треба искористити интернет као сервер, да би се информације чувале у јединственој бази података.

Информациони захтеви који треба да буду реализовани овим радом су:

- Унос података у базу, и то: унос новог пацијента у базу, новог лекара/техничара, лекова, дијагноза и свих осталих ентитета базе.
- Измена (унос) нове дијагноза/терапије постојећег пацијента.
- Штампа рецепата- опција којом се директно иде на страну са које се бира пацијент којем се штампају рецепти директно из базе.

4. Коришћене технологије

4.1. HTML

HTML (енгл. **H**yper **T**ext **M**arkup **L**anguage) је описни језик специјално намењен опису веб страница, а у буквалном преводу језик за означавање хипертекста. Хипер текст је текст који поред речи садржи и слику, видео и аудио записе. Помоћу њега се једноставно могу одвојити елементи као што су наслови, параграфи, цитати и слично. Поред тога, у HTML стандард су уграђени елементи који детаљније описују сам документ као што су кратак опис документа, кључне речи, подаци о аутору и слично. Ови подаци су општепознати као мета подаци и јасно су одвојени од садржаја документа.

HTML је подскуп једног ширег језика, SGML (Standard General Markup Language) и користи се за дефинисање изгледа веб докумената (страница) као и за успостављање веза (линкова) међу документима [1].

4.1.1. Дефиниција типа документа

Сви HTML документи би требало да почињу са дефиницијом типа документа DTD, Document Type Definition који прегледачу дефинише по ком стандарду је документ писан.

Пример: <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dtd">

Овај кôд говори прегледачу да је документ писан по строгом HTML 4.01 стандарду. Овај конкретан стандард искључује коришћење презентационих елемената да би се што боље одвојила презентација од садржаја.

4.1.2. Синтакса

Основ HTML-а представљају тагови (енгл. tags) и атрибути. Тагови могу бити прости, облика (<X>), када служе за маркирање и сложени, када се користе као ограђивачи (<X> и </X>) за делове текста између њих. Сваки таг представља посебну команду, којом се одређени део документа одваја од остатка примењујући правила дефинисана самим тагом. Атрибути се налазе унутар сложених ознака и садрже додатне информације о начину приказивања и понашању означеног дела документа. Атрибути се смештају између знакова навода. Интернет прегледач игнорише непознате атрибуте и елементе и не прави разлику између великих и малих слова (енгл. case insensitive). Он текст аутоматски прелама према ширини блока или прозора, правећи нове редове ако је потребно.

Пример:

```
<p>Текст параграфа.</p>

<p align="right">Текст параграфа</p>
```

У првом примеру се одабрани део означава као параграф. У другом примеру, користи се атрибут за поравнавање параграфа на десно [2].

4.1.3. Структура документа

HTML документи се састоје из два основна дела: оног који описује документ и начин на који треба да буде представљен и дела који представља садржај документа. Информације које описују документ се смештају између ознака за заглавље документа <head>, <head>. У оквиру заглавља поставља се наслов документа између ознака <title>, <title>, док се сам садржај смешта између <body>, <body>. Ова три елемента заједно са ознакама <html>, <html> чине минимилану структуру HTML документа.

Пример:

```
<!DOCTYPE HTML>
<html>
<head>
<meta charset="UTF-8">
   <title>Наслов документа</title>
</head>
<body>
    <h1>Пример документа</h1>
    <p>Ово је пример једног простог HTML документа.</p>
</body>
</html>
```

UTF-8 је карактер сет за српски језик, односно слова која се појављују у спрском језику.

4.1.4. Коментари

Коментари у оквиру HTML кôда, који се не приказују помоћу интерент прегледача пишу се између ознака <!-- и -->.

Пример:
<!-- Ово је коментар -->

4.1.5. Елементи

-
 — ознака за нови ред (енгл. break line), ради се о простој ознаци, која има само почетак, нема ознаке за крај.
- <hn> ... </hn> — ознака за наслов (енгл. heading). n може узети вредности од 1 до 6. Користи атрибуте глобалног типа. Што је наслов већег нивоа (што је мањи број), то ће он бити истакнутије приказан.

Кôд:	Приказ:
<h1**>Наслов величине 1<**/h1**>**	**Наслов величине 1**
<h2**>Наслов величине 2<**/h2**>**	**Наслов величине 2**
<h3**>Наслов величине 3<**/h3**>**	**Наслов величине 3**
<h4**>Наслов величине 4<**/h4**>**	**Наслов величине 4**
<h5**>Наслов величине 5<**/h5**>**	**Наслов величине 5**
<h6**>Наслов величине 6<**/h6**>**	Наслов величине 6

- ... или само ознака за нови пасус (енгл. paragraf)
- ... неуређена листа (енгл. unordered list), са елементима листе (енгл. list item) ...

```
<ul>
  <li>Елемент 1</li>
  <li>Елемент 2</li>
</ul>
```

- ... уређена листа (енгл. ordered list) са редним бројевима испред елемената листе <li>..., где је могуће поставити атрибут start=”почетни редни број“.

```
<ol start="5">
  <li>Елемент 5</li>
  <li>Елемент 6</li>
</ol>
```

- табела (енгл. table) се пише помоћу ознака <table></table>, између којих могу стајати: ознаке за редове (енгл. rows) <tr></tr>, колоне, односно податке у табели (енгл. table data) <td></td> и заглавља за колоне (енгл. table header) <th></th>

```
<table border="2">
  <tr>
      <td>ред 1, колона 1</td>
       <td>ред 1, колона 2</td>
  </tr>
  <tr>
       <td>ред 2, колона 1</td>
       <td>ред 2, колона 2</td>
  </tr>
</table>
```

- одељак служи као контејнер за груписање других елемената. Када се користи са CSS служи за постављање заједничких атрибута стилова великим блоковима елемената.
- контејнер за текст. Када се користи са CSS служи за постављање заједничких атрибута стилова за текстове.
- хипервеза или веза (енгл. hyperlink) за реч, групу речи или слику. Притиском на дату везу врши се скок на други документ или други део истог документа. Реализује се помоћу ознаке <a>. Најважнији атрибут за везу је href у коме се одређује документ на коју треба скочити. Такође, могуће је поставити и атрибут target којим се одређује да ли ће се документ отворити у истом прозору (“_self “) или у новом прозору (“_blank “). Ако се не постави ни једна вредност атрибута, отвара се у истом прозору.

```
<a href="http://php.org/" target="_blank">веза на PHP</a>
```

- слика <img src="URL адреса слике" alt="алтернативни текст за слику" height="висина у пикселима" width="ширина у пикселима" ismap="ismap"> енгл. image приказује се на страници преко URL адресе, нема ознаку за крај. Опција ismap се наводи када је слика на страни

сервера, када је потребно одредити област слике на којој је извршен клик, при чему се координате шаљу на сервер. Ако је слика-мапа на страни клијента, користи се опција usermap [2].

```
<img src="http://php.org/ pic.png" alt="Слика" />
```

4.1.6. Формати текста

- **подебљана слова** (енгл. bold)
- *искошена - курзив слова* (енгл. italc)
- <u>подвучен текст</u> (енгл. underline)
- ~~прецртана слова~~ (енгл. struckthrough)
- <big>велика слова</big> (енгл. big)
- <small>мала слова</small> (енгл. small)
- ^{горњи индекс}(енгл. superscripted)
- _{доњи индекс} (енгл. subscripted)
- <code>текст компјутерског кôда</code> (енгл. computer code)
- <pre>...</pre> дословни пренос текста, онако како је унет (енгл. preformatting)

4.1.7. HTML форме

HTML форма је део документа који у себи може да садржи елементе као што су разна поља за унос потатака, разне врсте дугмића за покретање акције и сл.

Форма се креира тагом <form>, а појединачни елементи форме таговима <input>.Таг <form> и тагови <input> могу имати низ атрибута који их поближе дефинишу.

Пример:

```
<form>
Име:
<input type="text" name="ime">
<br>
Презиме:
<input type="text" name="prezime">
```

```
</form>
```

У претраживачу ће то изгледати овако:

Име : 

Презиме: 

Таг <input> се користи и за дефинисање дугмића (енгл.buttons) који имају различите облике и употребу.

Акционо дугме се користи за покретање неке акције, рецимо брисање свих поља у форми, стартовање неког јавскрипт програма и сл. Акционо дугме се креира на начин приказан у следећем примеру.

```
<form>
<input type="button" name="dugme" value="Start" onClick="Akcija">
</form>
```

Овако би то изгледало у претраживачу:

Радио дугме се користи када желите да правите избор између малог броја опција као у следећем примеру.

```
<form>
<input type="radio" name="pol" value="muški"> Мушки
<br>
<input type="radio" name="pol" value="ženski"> Женски
</form>
```

Овако то изгледа у претраживачу:

- Мушки
- Женски

Само једна од две опције може бити одабрана када се кликне на дугме.

Checkbox дугме се користи када желите да одаберете једну или више опција из неког мањег скупа као што то показује следећи пример:

```
<form>
Шта све имам?
Имам бицикл:
<input type="checkbox" name="vozilo" value="Bicikl">
```

```
<br>
Имам ауто:
<input type="checkbox" name="vozilo" value="Auto">
<br>
Имам авион:
<input type="checkbox" name="vozilo" value="Avion">
</form>
```

А ево како би то изгледало у претраживачу:

Шта све имам?

Имам бицикл: 

Имам ауто : 

Имам авион: 

Код checkbox думића може да се бира више опција (за разлику од радио дугмића где се бира само једну опција).

Форме се најчешће користе за слање података ка серверу. За слање податка се користи дугме типа submit, а у тагу <form> се додаје атрибут action којим се дефинише који ће програм на серверској страни преузети послате податке.

Атрибутом "method" дефинише се метод слања података, који може бити "GET" или "POST" [1].

Пример:

```
<form name="input" action=http://program.com/login.php  method="get">
        Корисник:
        <input type="text" name="korisnik">
        <input type="submit" value="Submit">
</form>
```

Овако то изгледа у претраживачу:

Корисник: 

У примеру је показано како се податак који се унесе у празно поље преноси до фајла login.php , који у себи садржи кôд којим се над преузетим податком спроводи одговарајућа акција [2].

4.2. PHP

PHP је специјализовани скриптни језик првенствено намењен за израду динамичног веб садржаја и изводи се на страни сервера.

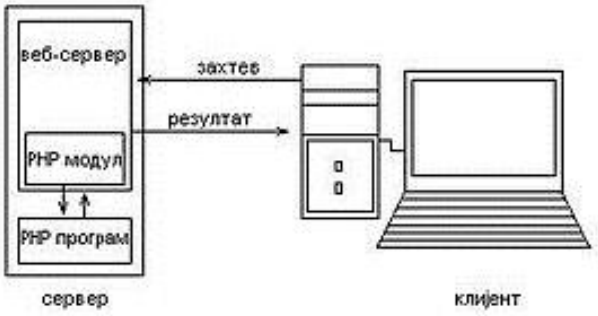
PHP је стекао популарност због своје једноставности и синтаксе наслеђене из програмског језика C. Током времена језик се проширивао и стицао могућности за објектно оријентисано програмирање, нарочито од верзије 5.0. Наликује језику C++ у смислу што дозвољава и чисто процедурално програмирање, али истовремено омогућава и коришћење класа и других концепата објектно оријентисаног програмирања (наслеђивање, апстрактне класе и методе, интерфејсе итд.) [3].

4.2.1. Употреба језика PHP

Иако се PHP може користити за програмирање конзолних апликација и графичких интерфејса (библиотека PHP-GTK), његова основна и главна употреба је у програмирању динамичних страница на интернету.

Данас PHP користи неколико стотина хиљада програмера и неколико милиона сајтова укључујући неке од најпосећенијих сајтова на свету.

4.2.2. Начин извршења



Слика 5. Извршавање PHP скрипте на веб-серверу

Програм који се напише у PHP-у не захтева превођење (компајлирање), него се интерпретира при сваком извршавању. PHP интерпретатор може радити по PHP CGI принципу, односно тако што ће интерпретатор постојати као екстерна апликација, која се позива да изврши дату скрипту сваки пут кад буде захтевана од неког корисника, а може бити инсталиран и као модул веб-сервиса. Друга варијанта је данас у највећој употреби јер пружа знатно већу брзину извршавања - интерпретатор је на тај начин увек учитан у меморију те се не мора позивати спољашњи програм.

Уобичајен сценарио по ком се извршавају PHP скрипте је следећи:

- клијент (корисник Интернета који користи неки интернет прегледач) захтева PHP страницу са сервера
- сервер прослеђује захтев сервису за веб (програм веб-сервер на серверу)
- веб-сервер препознаје да се тражи PHP датотека
- не шаље његов садржај клијенту, него га извршава као програм помоћу PHP модула
- излазни текст програма (стандардни излаз) се шаље клијенту као резултат захтева
- клијент препознаје врсту резултата (HTML кôд, слика, PDF садржај, архива итд.)
- резултат се приказује клијенту на одговарајући начин [5].

4.2.3. Структура програма

За разлику од већине програмских језика који поседују почетну функцију (main у C-у, први блок BEGIN и Паскалу, класа camain методом у Јави итд.), PHP датотека, налик на већину скриптних језика, једноставно садржи скуп наредби, које се извршавају једна за другом, од прве до последње, где се последња уједно сматра и крајем PHP програма.

PHP датотеке могу да садрже следеће врсте елемената:

- HTML ознаке
- Ознаке за PHP
- PHP наредбе
- Белине (енгл. white spaces)
- Коментаре

Постоје четири стила за исписивање PHP ознака:

- XML стил уз помоћ већ поменутих ознака <?php и ?> и такав стил се препоручује, а обавезан је када је PHP кôд усађен у HTML кôд.
- скраћени стил између ознака <? и ?>. Може да се користи у колико се у конфигурацискоj датотеци php.ini активирана опција short_open_tag, али се његова употреба не препоручује зато јер у неким окружењима није подржан. Међутим, од верзије 5.4, скраћени стил у облику <?= ... ?> је омогућен без обзира на подешавања унутар конфигурацијске датотеке, а служи као замена за <?php ... ?>.

- скрипт стил између ознака <script language="php"> и </script>, је дужа верзија XML стила и обично се препоручује када се истовремено користе и други скриптни језици, посебно ако HTML едитор прави проблеме.
- ASP стил између ознака <% и %> представља још једну скраћену верзију. Најчешће се користи у ASP и ASP.NET окружењу, али је неопходно претходно активирати опцију short_open_tag у конфигурацискоj датотеци php.ini. Такође се не препоручује [5].

Под белинама или белим просторима (енгл. white spaces) се подразумевају знакови за раздвајање, као што су ознаке за повратак на почетак реда, размаци и табулатори. Интерпретатори за HTML кôд и PHP изворно окружење занемарују ове просторе.

PHP наредбе „угнеждавају“ се у заглавље (енгл. header) и/или тело (енгл. body) веб стране. Уколико нека страна садржи PHP скрипту, требало би да носи наставак .php, како би веб сервер био обавештен о њиховом постојању. У PHP датотеци, блок који је окружен језичким структурама, које започињу HTML ознаком (енгл. tag) <?php, а завршавају се ознаком ?>, се сматра PHP кôдом и извршава се помоћу PHP интерпретатора (тумача), док се остатак кôда, ван ових ознака, сматра хипертекстом, који једноставно треба да се испише на стандардни излаз, без интерпретирања.

Пример најједноставнијег PHP програма који исписује текст „Здраво свете!“:

```
<!-- део написан у HTML-у се преписује директно на стандардни излаз -->
<html>
<body>
   <p>
      <!-- наредни део припада PHP блоку, те се извршава -->
      <?php
          echo "Здраво свете!";
      ?>
<!-- део до краја кôда се такође преписује директно на стандардни излаз -->
   </p>
</body>
</html>
```

Тумач за PHP кôд потпуно игнорише празан простор између наредби, тако да његова употреба зависи од личног избора програмера и само доприноси визуелној прегледности исписаног кôда. Крај сваке наредбе обележава се тачка зарезом.

PHP кôд може бити организован у функције и класе, и може се организовати у више датотека. Као почетна датотека, тј. датотека чије наредбе се извршавају прве, се узима она датотека која се да интерпретатору на извршавање. Уколико датотека није наведена, сервер ће аутоматски покушати да покрене датотеку index.php [10].

4.2.4. Основна синтакса

Коментари

Коментари се могу писати иза две косе црте // или симбола # ако су једноредни (налазе се у истом реду), или између симбола/* и */ за једноредне и вишередне коментаре.

```
<?php

echo ”Проба за једноредни коментар”;  // Ово је једноредни коментар

/* Ово је коментар
            исписан у два реда */

echo ”Још један једноредни коментар”;    # Ово је једноредни коментар

?>
```

4.2.4.1. Променљиве

Променљиве (енгл. variables) у PHP језику се представљају помоћу знака $, иза кога следи име променљиве. Име мора да почне словом или доњом цртом _ иза којег може да следи неограничен број слова, бројева или доњих црта, при чему се словом сматра слово латинице [A-Z], [a-z] или било који знак са ASCII кôдом већим од 127. Тумач за PHP разликује велика од малих слова, па тако променљива $A није исто што и променљива $a.

Променљиве се не декларишу, а њихов тип се одређује према вредностима које им се у изразима додељују током програма, те постоји могућност да се тип променљиве промени.

PHP има следеће типове променљивих:

- Основни типови променљивих:
  - **boolean** - логичке вредности: тачно - true и нетачно - false
  - **integer** - цели бројеви
  - **float** - реални бројеви са покретном запетом
  - **string** - низ знакова (симбола), односно текст
- Сложени типови променљивих:
  - **array** - низ
  - **object** - објекат
- Посебни типови променљивих:
  - **resource** - ресурс, садржи упућивач на спољашњи ресурс, нпр. датотеку или веза са базом података
  - **NULL** - променљива нема вредност, може садржати само вредност null

Није неопходна додела почетних вредности, мада се препоручује, ради прегледности, безбедности и бољег разумевања кôда, с обзиром да се њиховом применом у изразима који садрже мешовите типове података, могу добити различити резултати. Тако нпр:

```
<?php
$a = 1.2;
$b = 2;
$b = $a + $b;
?>
```

Променљива $b у почетку је цео број, а затим се претвара у реалан број са једном децималом (3.2). Осим додељивања вредности имплицитно, промена типа променљиве се може извршити експлицитном конверзијом, тако што се испред имена упише тип променљиве, између заграда.

Пример:

```
<?php
$a = 1.2;
$b = 2;
$b = (int)$a + $b;
?>
```

У овом случају променљива $b добија вредност 3. При томе се код променљиве $a неће променити ни тип, ни вредност [3].

4.2.4.2. Корисне функције

За исписивање променљивих на стандардни излаз користе се две основне функције:

- **print** ($arg)
- **echo** ($arg1 [, $arg2 ...])

Обе су језички конструктори, па се при њиховом позивању не морају користити заграде. Разлика између једне и друге функције је у томе што функцији print може да се проследи само један аргумент, док echo може да штампа више њих одједном, тако да што при позиву променљиве стоје међусобно раздвојене запетом. Променљиве свих типова биће претворене у тип string приликом исписивања. Друга разлика између ове две функције је у томе што наредба echo нема повратну вредност, а print има [4].

За испитивање да ли је нека променљива празна, односно да ли има додељену вредност, користи се функција **empty**. За испитивање да ли нека променљива има почетну вредност, односно да њена вредност није null, користи се функција **isset**. Ако се функција проследи више од једне променљиве, враћа вредност тачно само у случају да ни једна од променљивих празна. Испитивање променљивих извршава се с лева на десно и завршава се

у случају да наиђе на прву променљиву која је празна, јер у том случају враћа вредност нетачно.

Елиминисање променљиве врши се помоћу функције **unset**. Понашање ове функције у великој мери зависи од врсте променљиве и места одакле је позвана. Нпр. у колико се позове унутар блока неке корисничке функције, биће елиминисана вредност само локалне променљиве. За промене вредности глобалних променљивих препоручује се употреба низа $GLOBALS [3].

4.2.4.3. Испитивање типа променљивих

За испитивање типа променљивих могу се користити функције из табеле 1.

Табела 1. Функције за испитивање типа променљивих

Назив функције	**Функција испитује да ли је**:
is_array	променљива низ
is_double, is_float, is_real	променљива у формату са покретном запетом
is_long, is_int, is_integer	промељнива цео број
is_string	променљива знаковни низ
is_object	променљива објекат
is_resource	променљива ресурс
is_null	променљива има вредност null
is_scalar	променљива скаларног типа
is_numeric	променљива број
is_callable	вредност променљиве име постојеће функције

4.2.4.4. Константе

За разлику од променљивих, константе се пишу без симбола $. Правило за имена константи је исто као и за променљиве. Мада није неопходно, константе се обично пишу великим словима, а њихове вредности се додељују на следећи начин:

```
<?php
define(„PI“, 3.14);
echo PI;
// Ispisuje 3.14
?>
```

Неке константе су дефинисане у самом програмском језику PHP. Тако није потребно дефинисати вредност константе Пи, као што је то дато у примеру, већ се може користити већ готова дефиниција M_PI, а она износи 3.14159265358979323846.

Једном дефинисана константа, не може да мења вредност. Константама је могуће доделити искључиво скаларне вредности, већ поменутих основних типова података (boolean, integer, float и string).

Овај начин дефинисања може да се користи на глобалном нивоу, али не и у оквиру блокова структура, појединачних функција, метода или класа, па се у тим случајевима користи кључна реч const.

```
<?php

define(”MIN_VREDNOST”, "0.0");  // Правилно дефинисање константе ван класе
define(”MAX_VREDNOST”, "1.0");

class ClasaKonstanta
{
   const MIN_VREDNOST = 0.0;              /* Правилан начин за дефинисање
константе унутар класе*/
   const MAX_VREDNOST = 1.0

   public static function getMinVrednost()
   {
      return self::MIN_VREDNOST;
   }
   public static function getMaxVrednost()
   {
      return self::MAX_VREDNOST;
   }
}
?>
```

4.2.4.5. Изрази и оператори

Начин употребе израза и оператора није знатно другачији од других програмских језика. Најсличнији је програмском језику C. Највећа разлика је у чињеници да за писање програма у програмском језику PHP није потребно декларисати типове података [4].

Оператори су симболи који омогућавају извршавање операција над вредностима и променљивама. Могу имати један, два или више аргумената.

Најважнији оператори су:

- аритметички оператори: сабирање (+), одузимање (−), множење (*), дељење (/), модуо (%).
- оператор за придруживање знаковних променљивих је тачка
- оператор доделе (=)
- комбиновани оператори доделе

Табела 2. Комбиновани оператори доделе

оператор	употреба	значење
++	$a++	$a = $a + 1
--	$a--	$a = $a - 1
+=	$a += $b	$a = $a + $b
-=	$a -= $b	$a = $a - $b
*=	$a *= $b	$a = $a * $b
/=	$a /= $b	$a = $a / $b
%=	$a %= b	$a = $a % $b
.=	$a .= b	$a = $a . $b

- оператори поређења за који као резултат поређења израза враћају вредности тачно (true) или нетачно (false)

Табела 3. Оператори поређења

оператор	назив	употреба
==	једнако	$a == $b
===	идентично (тип и вредност)	$a === $b
!=	није једнако (различито)	$a != $b
<>	није једнако (различито)	$a <> $b
!==	није идентично	$a !== $b
<	мање од	$a < $b
>	веће од	$a > $b
<=	мање или једнако	$a <= $b
>=	веће или једнако	$a >= $b

- логички оператори поређења, такође враћају вредности тачно (true) или нетачно (false). Осим оператора из табеле може да се користе и логички оператор XOR за искључиву дисјункцију са вредношћу тачно (true) ако су један оператора или оба нетачни [10].

Табела 4. Логички оператори

оператор	назив	употреба	резултат
! (NOT)	негација	!$a	израз тачан ако је $a нетачно
&& (AND)	конјункција	$a && $b	тачан ако су $a и $b тачни
|| (OR)	дисјункција	$a || $b	тачан ако је $a или $b тачно

4.2.4.6. Наредбе и управљачке структуре

Наредбе (енгл. statements) или искази одређују неку акцију. Свака PHP скрипта је састављена од низа наредби. По правилу наредбе се извршавају оним редом којим су наведени у програму, али овај редослед може да се промени неком од наредби за пренос тока програма, при чему пренос може ићи на било који други део програма или ван њега. Основни облик наредбе је израз иза кога следи знак тачка зарез, као завршни знак наредбе.

Сложена наредба или блок наредба је механизам који групише скуп наредби у једну семантичку целину. Блокови могу да буду уклопљени и сваки може да садржи своје сопствене декларације, иницијализације и извршне наредбе.

Наредбе којима се одређује ток извршавања под одређеним условима, могу да се групишу у једну засебну структуру. Оваква структура се поставља између витичастих заграда: { }. Управљачке структуре, саме за себе, такође представљају једну наредбу у облику блока. Користе се различити облици управљачких (контролних) структура (енгл. control structure), које се међусобно разликују по начину испитивања услова [10].

Условно гранање (if, switch)

Једна од основних управљачких структура је **условно гранање**, које у зависности од потребе, може имати различите форме, а једна од њих је следећа:

```
<?php
if (условни_израз_1)
        {   блок_наредби_1; }

elseif (условни_израз_2)
        {   блок_наредби_2; }
/*... */
```

```
elseif (условни_израз_н)
        {   блок_наредби_н; }
else
        {   последњи_блок_наредби; }
?>
```

Када интерпретатор наиђе на овакву структуру, он прво проверава да ли је први први услов (условни_израз_1) испуњен. Ако је први израз тачан, извршиће се први блок наредби. Тај блок може бити група од једне или више наредби, од којих неке такође могу бити сложене управљачке структуре. У колико израз није тачан, испитиваће се истинитост другог условног израза. Затим, у колико је други условни израз тачан, извршиће се други блок наредби. У супротном, наставља се испитивање следећих услова и тако редом, све до последње клаузуле „else“. Овај задњи обухвата све оне случајеве који нису испунили ни један од претходних услова.

У случају да се блок_наредби састоји од само једне наредбе, није неопходна употреба витичастих заграда. Постоје случајеви када није потребно настављати даље гранање, већ се оно завршава са проверавањем првог израза. У том случају неће се писати ни једна од клаузула elseif, а ни else. Најпростији облик условног гранања је следећи:

```
<?php
if ($godina_rodjenja < 1996)
   echo "ради се о пунолетној особи";
?>
```

У неким ситуацијама је неопходно једну исту променљиву или израз упоредити са различитим вредностима, те у зависности од резултата поређења, извршити различите акције. У тим случајевима је, уместо структуре **if**, једноставније употребити структуру **switch**, која има следећи облик:

```
<?php
switch (израз)
{
   case "вредност_1":
        блок_наредби_1;
   case "вредност_2":
        блок_наредби_2;
   /*... */
   case "вредност_н"
        блок_наредби_н;
   default:
        последњи_блок_наредби;
}
?>
```

Ова структура се извршава тако што се вредност израза, знаковног (string) или целобројног (integer) типа, редом пореди са различитим случајевима (енгл. case) могућих очекиваних вредности, све док се не наиђе на случај када су те две вредности једнаке. Одатле, па на даље, извршавају се сви блокови наредби, по реду, све до последњег, или се прекида извршавање, у колико постоји наредба за излазак из структуре (енгл. break). Последњи блок наредби је подразумевани случај (енгл. default), који није обавезно навести, односи се на онакве вредности израза, које нису биле обухваћене ни једним од претходних случајева. Блокови наредби могу да садрже групу од једне или више наредби, од којих неке, такође могу бити сложене [10].

Приликом исписивања блокова наредби, посебно треба обратити пажњу на места на којима ће се стајати наредба break, однос прекид извршавања. Јер, у колико не постоји наредба прекида, једном када је резултат поређења да тачну вреност, извршавају се сви следећи блокови наредби, до краја.

```
<?php
switch ($slovo)
{
   case "а":
   case "А":
         $rezultat = "словао а";      /* резултат ће бити "слово а",  када је $slovo == "а" и када је $slovo == "А" */
         break;
   case "б":
   case "В":
         $rezultat = "слово б";
         break;                  /* резултат ће бити "слово б",  када је $slovo == "б" и када је $slovo == "Б" */
  }
?>
```

Петље

Петље (енгл. loops) су управљачке структуре за циклично (итеративно) понављање групе наредби, под одређеним условом. Извршавање петљи могуће је прекинути у било ком делу петље употребом наредбе break, док се тренутни прелазак (скок) на следећу итерацију врши наредбом continue. У зависности од начина испитивања услова за извршавање петље и места где је он постављен разликујемо неколико врста петљи.

Петља „докле год“ (енгл. **while**) је једна од најједноставнијих. Услов за понављање налази се на врху наредбе. Она се представља на следећи начин:

```
<?php
while (условни_израз)
{
   блок_наредби;
}
?>
```

Блок наредби се понавља све док је испуњен услов у изразу. Ако је његова вредност нетачна од самог почетка, блок наредби се никад неће извршити.

Специјалан случај петље „докле год“ (енгл. **do-while**) са условом који се испитује на крају сваке итерације, има следећи облик:

```
<?php
do
{
   блок_наредби;
}
while (условни_израз)
?>
```

Блок наредби ће се извршити макар једном, чак и када услов није испуњен ни у првој итерацији.

Петља **for** је најсложенији тип петљи. Може да се користи када је број итерација унапред познат. Има следећи облик:

```
<?php
for (израз1; израз2; израз3)
{
   блок_наредби;
}
?>
```

Први израз се користи да би се поставила почетна вредност бројача. У другом изразу се поставља услов за понављање, а у трећем се мења вредност бројача. Често се користе код низова и/или када је потребно попунити неку табелу.

Петља за „сваки елемент“ (енгл. **foreach**) је варијација петље for, специјално направљена за рад са низовима и другим објектима. Она има два облика:

```
<?php
foreach (низ as вредност)
{
   блок_наредби;
}
?>
```

Пролази се кроз цео низ и у свакој итерацији се вредност елемента смешта у променљиву. Ова вредност касније се користи у блоку наредби. Други облик наредбе је следећи:

```
<?php
foreach (низ as индекс => вредност)
{
   блок_наредби;
}
?>
```

Овде се осим вредности елемента користи и променљива за смештање вредности индекса (кључа) сваког од елемената низа.

4.2.4.7. Низови

Један од сложенијих типова променљивих су **низови** (енгл. arrays). Низ је група променљивих референцирана преко истог имена. За разлику од других програмских језика, овде елементи низа могу бити подаци различитог типа. У употреби су две врсте низова:

- **нумерички**, код којих се користе позитивни цели бројеви за индексирање, почевши од индекса 0

```
<?php
$niz  = array(10, 20, 30, 40, 50, 60);      /* променљива $niz је низ од 6 елемената са индексима од 0 до 5 */
$zbir = $a[0] + $a[1] + $a[2] + $a[3] + $a[4] + $a[5]; // збир ће имати вредност 210
?>
```

За низове је веома погодно користити петље, да би се прошли сви елементи:

```
<?php
$a = array(1, "проба", "успешна");  /* низ од 3 елемента, од којих је сваки другачијег типа */
$suma = "";
for ($i = 0; $i < sizeof($a); $i++)
{
   $suma .= " " . $a[$i];          // придружује чланове низа у једну променљиву
}
echo $suma;                        // на крају сума има вредност "1 проба успешна"

?>
```

- **асоцијативни**, који као индексе (кључеве) могу да користе податке знаковног типа (string), осим целобројних (integer)

```
<?php
$hrana = array("доручак" => "сендвич", "ручак" => "мусака", "вечера" => "јабука");
// дефинише низ $hrana од 3 елемента
// од којих сваки има свој кључ (знаковни индекс) и вредност
foreach ($hrana as $kljuc => $vrednost)
{
   echo "$kljuc = $vrednost<br />"; /*   на излазу приказује "доручак = сендвич
                                        ручак = мусака
                                        вечера = јабука"      */
}
?>
```

Табела 5. Оператори за рад са низовима

оператор	назив	употреба	значење
+	унија	$а + $б	враћа низ који се састоји од свих елемента $а и $б
==	једнако	$а == $б	израз је тачан ако су низови $а и $б имају исте парове индекс/вредност
===	идентично	$а === $б	израз је тачан ако су низови $а и $б имају исте парове индекс/вредност, по истом редоследу и типу података
!=	различито	$а != $б	израз је тачан ако је $а различит од $б
<>	различито	$а <> $б	израз је тачан ако је $а различит од $б
!==	није идентично	$а !== $б	израз је тачан ако $а није идентичан $б

Приступ елементима низа је веома једноставан коришћењем угластих заграда []. PHP располаже великим бројем функција за рад са низовима.

Табела 6. Функције за рад са низовима

назив функције	опис функције
unset()	елиминисање елемената из низа
array_key_exists()	провера за постојање неког индекса (кључа) у низу
array_search()	претрага елемената на основу индекса (кључа)
in_array()	провера за постојање неке вредности у низу
array_keys()	враћа низ кључева наведеног низа
array_values()	враћа низ вредности наведеног низа
array_splice()	брисање елемената низа или њихова замена
count()	враћа број елемената наведеног низа (синоним: sizeof())
sort()	уређује низ по њиховим вредностима у растући редослед
rsort()	уређује низ по у опадајући редослед

asort()	уређује низове по њиховом вредностима у растући абецедни редослед, при чему се врши њихова реиндексација
ksort()	уређује индексе низа по вредностима индекса у растући абецедни редослед
krsort()	уређује индексе низа по вредностима индекса у опадајући редослед
shuffle()	меша елементе низа (насумично уређивање)
array_sum()	сабира вредности свих елемената наведеног низа
array_merge()	спаја два или више низова у један. Важно је водити рачуна о индексима, јер се приликом спајања елиминишу дупликати (важи само за знаковне индексе, док се бројчани аутоматски уређују)
explode()	формира низ из наведене знаковне променљиве (текста). Осим имена низа и променљиве, захтева и знак за раздвајање
implode()	формира текст из наведеног низа. Осим имена низа и променљиве, захтева и знак за раздвајање
array_diff()	упоређује два или више низова, и креира нови на основу елемената по којима се они разликују
array_unique()	уклања дупликате из наведеног низа

4.2.4.8. Објекти

Основа објектно – орјентисане програмске функционалности додата је у PHP 3. Рад са објектима је комплетно изнова написан за PHP 5, проширујући сет особина и унапређујући перформансе. У претходним верзијама PHP-а, објектима се руковало као са примитивним типовима. Лоша особина ове методе је та што је цео објект бивао копиран када би променљива била додељена или послата као параметар методу. Нови приступ, објекте позива као поступак, а не као вредност. PHP 5 уноси скривене и заштићене чланове променљивих и методе, заједно са апстрактним класама и коначним класама као и апстрактне методе и коначне методе. Такође уводи стандардно декларисање конструктора и деструктора, слично другим објектно – оријентисаним језицима као што је C++, и стандардни модел обраде изузетака. Осим тога PHP 5 додаје интерфејсе и допушта имплементацију мулти интерфејса. Постоје специјални интерфејси који омогућавају објектима интеракцију са прометним системом. Објекти имплементирани у ArrayAccess могу се користити помоћу синтаксе низа (array), а објекти имплементирани у Iterator или IteratorAggregate помоћу језичке конструкције foreach.

Статичка метода и особине класе вариабле у Зенд машини 2 не раде како би се очекивало. У њему не постоји виртуелна табела особина, па се

статичке променљиве везују за име уместо за референце за време компајлирања.

Овај пример показује како да се дефинише класа foo, која се наслеђује из class bar-а. Функција mystaticfunc је јавна статичка функција и може се позвати помоћу foo::mystaticfunc();.

```
class foo extends bar
{
 function __construct()
 {
 $doo = "wah dee dee";
 }

 public static function mystaticfunc()
 {
 $dee = "dee dee dum";
 }
}
```

Ако програмер креира копију објекта користећи резервисану реч clone, Зенд машина ће проверити да ли је __clone() метод дефинисан. Ако није, позваће предефинисани (default) __clone() који ће ископирати све особине објекта. Ако је __clone() метод дефинисан, биће одговоран за постављање неопходних особина за креирани објекат. Зенд машина обезбеђује функцију која импортује све особине првобитног (source) објекта, тако да програмер може да почне са копијом вредности особина првобитног објекта и само да промени особине које је потребно изменити [10].

4.2.4.9. Функције

Функције омогућавају извршавање одређеног блока наредби, које се понављају. PHP има доста већ уграђених, али постоји могућност дефинисиња сопствених функција. Функција може да се позове из неког другог дела скрипте, прослеђујући јој аргументе, а она затим, када се изврши, може да врати неку вредност. Разлози због којег се користе функције могу бити:

- избегавање поновног писања већ написаног дела кôда
- лакше разумевање кôда

Функција се декларише помоћу кључне речи **function**, на следећи начин:

```
function име_функције(листа_параметара)
{
   блок наредби;
}
```

Приликом избора имена функције, треба поштовати иста правила, која важе за имена променљивих, с том разликом да се код функција мала и велика слова не разликују и не пише се знак $. Препоручује се да име функције асоцира на акције, које она треба да раеализује. Мада је у другим програмским

језицима дозвољено, овде се за имена функција не могу искористи имена већ постојећих функција, јер то доводи до грешке у програму, осим ако се истоимена функција не налази у некој спољашњој датотеци. Ипак, чак ни тада се не препоручује, јер може отежати тумачење написаног кôда.

У параметрима се наводи низ променљивих, међусобно раздвојених запетом. Функција може имати један или више параметара. Постоје случајеви када неће бити потребан ни један параметар. Међутим чак и у тада је обавезно писање обичних заграда приликом декларисања функције.

```
function Pozdrav()
{
  return "ДОБРО ДОШЛИ";
}
```

Да би се функција извршила, није довољно да само буде декларисана. Да би се искористила она мора да буде позвана. Није неопходно, али се препоручује, да пре позивања, функција буде већ декларисана. Приликом позивања функције, наводи се листа аргумената.

```
име_функције (листа_аргумената);
```

Редослед аргумената, који се наводе у позиву, треба да одговара редоследу параметара у декларацији функције. Као аргументи, могу се навести променљиве, одређене вредности или изрази. Бројне вредности се могу проследити са или без наводника. Није неопходно да број елемената из листа аргумената буде једнак броју елемената из листе параметара. Ако је број аргумената мањи и немају унапред додељену вредност недостајућим променљивама се додељује вредност null (мада долази до грешке, она се неће зауставити извршавање функције). Ако функција враћа неку вредност, она се у позиву може доделити некој променљивој или употребити у неком изразу.

Када се извршавање функције заврши, ток програма се наставља, тачно од места одакле је позвана. Извршавање може да се прекине пошто се прође кроз целу функцију или се ток извршавања може прекинути у било ком делу блока, употребом једне или више наредби за повратак (енгл. return). Претерана употреба наредбе за повратак у оквиру исте функције, може отежати разумевање тока функције. У пракси, најбоље је да постоји један улаз и један излаз из блока наредби. Дозвољена је рекурзија, односно, функција може да позове саму себе. У следећем примеру рачуна се факторијал прослеђеног природног броја:

```
function faktorijal($broj)
{
   if ($broj < 0)
   {
      return "недозвољена вредност";
   }
   elseif ($broj == 0)
   {
      return 1;
   }
```

```
   else
   {
      $pomocna = $broj * faktorijal ($broj - 1);    /* вредност факторијала се добија рекурзијом*/
      return $ pomocna;
   }
}
```

Навођењем променљивих у листи аргумената, променљиве се могу проследити:

- **по вредности** (енгл. by value), када се унутар функције њихове вредности не мењају
- **по адреси**, односно референци (енгл. by reference) помоћу симбола &, када је неопходно да се вредност променљиве промени унутар функције. Позив по адреси могућ је једино приликом декларисања функције, никако при њеном позивању, у оквиру аргумената

```
function dodaj_jedan(&$broj)        /* декларација функције са параметром по адреси (&) */
{
  $broj = $broj + 1;                //увећава вредност за један
}

/* касније се позива функција*/
$broj = 1;                          // променљивој се додељује почетна вредност
dodaj_jedan($broj);      // позива се функција са аргументом (променљива $broj)
echo $broj;                         // добија вредност 2
```

Параметри могу да имају унапред додељене вредности (енгл. default values). При позивању функције, њих није обавезно навести и само у том случају, унапред додељене вредности имају ефекта. У супротном се користе вредности прослеђене из листе аргумената. У декларацији функције, параметри са унапред одређеним вредностима, наводе се на крају листе.

```
function dodaj_sabirak(&$broj, $sabirak = 1)  // параметар по адреси (&) и параметар са унапред додељеном вредношћу
{
  $broj = $broj + $sabirak;
}
/* касније се позива функција*/
$broj = 1;                          // променљивој се додељује почетна вредност
dodaj_sabirak($broj);               // позива се функција са аргументом (променљива $broj)
echo $broj;                         // $број добија вредност 2
$zbir = 1;
$moj_sabirak = 10;
dodaj_sabirak($moja_suma, $moj_sabirak);     /* позива функцију са оба аргумента*/
echo $zbir;                         // збир је 11
```

Све промељниве које се користе у оквиру функција, осим оних које се налазе у листи параметара прослеђене по адреси, добијају почетне вредности приликом сваког позивања функције. Те промељниве су локалног карактера, њихове вредности се додељују и расположиве су искључиво у оквиру исте функције. Важи и обрнуто, променљиве дефинисане ван функције, нису расположиве унутар функције, чак ни онда када се ван и у оквиру функције користе истоимене променљиве [11].

Да би променљиве коришћене локално, у оквиру неке функције имале глобални карактер, односно да би њене вредности биле на располагању и ван функције, променљиве је неопходно посебно назанчити на један од следећих начина:

- коришћењем кључне речи **global**:

```
global $име_променљиве [, ...];
```

```
<?php
function moja_funkcija()
{
   global $glob;
   echo $glob;                  //променљива $glob има вредност 1
}
?>
```

- коришћењем **асоцијативног низа** $GLOBALS, који је распложив увек и свуда, у било ком окружењу, због чега се назива супер глобалним. Елементе овог низа чине вредности, док индексе низа чине имена расположивих променљивих.

```
<?php
function moja_funkcija()
{
   global $globalna;
   $GLOBALS["c"] = $globalna + 1;   /* дефинисана је локална
променљива $c */
}
?>
```

Локалне променљиве у оквиру функције губе своје вредности, између различитих позива и поново се додељују приликом сваког извршавања. И ако вредности локалних променљивих не могу бити коришћене ван функције, постоји могућност да се вредности променљивих из претходног позива сачувају, тако што се променљиве унутар функције декларишу као статичке, помоћу кључне речи **static** и при томе им се може доделити нека почетна вредност.

```
static $име_променљиве [= вредност];
```

Почетна вредност ће бити додељена само при првом позиву функције. Пример: funkcija_uvecanja ће при сваком позиву увећати вредност променљиве $stat за 6:

```
<?php
function funkcija_uvecanja()
{
  static $stat = 8;
  $stat = $stat + 6;    /* 1. позив $stat = 14, 2. позив $stat = 20, 3. позив $stat = 26 */
}
?>
```

4.2.4.10. Управљање датотекама

На располагању је група функција за рад са датотечним системима. Прва функција која омогућава приступ датотеци је функција за отварање датотеке, при чему се ствара објекат типа ресурс, за опис датотеке (енгл. file descriptor), где се смешта информација о датотеци. Отварање се постиже уз помоћ функције **fopen**:

```
$фд = fopen( $име_датотеке", $начин_приступања [,
$користити_путању_за_спољашње_датотеке = false [, $ресурс]] );
```

која има четири могућа параметра, од којих су два обавезна и најчешће се користе:

- име датотеке се претпоставља да се ради о некој доступној веб адреси, која се тражи уз помоћ управљача протокола, односно омотача (енгл. wrapper).
- начин приступања датотеци може да има једну од следећих вредности:

Табела 7. Комбиновани оператори доделе

начин	опис
"r"	отвара за читање, показивач се поставља на почетак датотеке и одатле почиње да пише, без брисања претходног садржаја, ако датотека већ постоји, даје обавештење
"r+"	отвара за читање и писање, остало исто као и претходној опцији
"w"	отвара само за писање, показивач се поставља на почетак датотеке и одатле почиње да пише, бришући сав претходни садржај. Ако датотека не постоји, покушаће да направи нову
"w+"	отвара за читање и писање, остало исто као и у претходној опцији
"a"	отвара само за писање, чува претходни садржај датотеке, поставља показивач на крај датотеке и одатле почиње да пише. Ако датотека не постоји, покушаће да направи нову
"a+"	за читање и писање, остало исто као и опција a
"x"	прави нову датотеку и отвара је само за писање, постављајући показивач на почетак датотеке. Ако датотека већ постоји, даје грешку. Ако датотека не постоји, покушаће да направи нову

"x+"	прави нову датотеку и отвара је за читање и писање. Остало исто као претходна опција
"c"	отвара датотеку само за писање. Ако датотека не постоји, прави нову датотеку, ако постоји прави нову датотеку. Показивач се поставља на почетак датотеке.
"c+"	отвара датотеку за читање и писање. Остало исто као и опција c

- индикатор за коришћење путање спољашњих датотека наведен у опцији include_path у оквиру конфигурацијске датотеке php.ini

Резултат функције може се сместити у променљиву, која садржи показивач на ресурс (означену као $фд у примеру), ако је отварање било успешно или се добија вредност нетачно (false), ако је дошло до грешке приликом отварања.

Једном када се датотека успешно отвори, могуће је читање и писање података из ње.

Писање се се врши уз помоћ функције **fwrite**:

```
fwrite ( $фд , $садржај [, $дужина_у_бајтовима] );
```

којој се прослеђују: показивач на ресурс, садржај који се уписује, као обавезни аргументи, променљива типа string. Могуће је одредити дужину информације за упис, која се изражава бројем бајтова. Резултат ове функције може бити број уписаних бајтова, ако се уписивање успешно извршило или вредност нетачно ако је дошло до грешке приликом уписивања [11].

Читање садржаја, могуће је једино ако је датотека претходно успешно отворена. Може се извршити:

- линију по линију, помоћу функције **fgets**:

```
fgets ( resource $фд [, $број_бајтова ] )
```

где је променљива која садржи показивач на ресурс (означена као $фд у примеру) једини обавезни параметар. Уз њега се може навести променљива у коју ће бити смештен број учитаних бајтова. Учитавање је могуће све док показивач не дође до краја датотеке. Функција враћа учитану линију у колико се успешно извршила, односно вредност нетачно, ако је дошло до грешке. Учитана линија обично се смешта у низ или у знаковну промнљиву (string).

- целог садржаја који се смешта у низ помоћу функције **fgets**

- целог садржаја који се смешта у знаковну променљиву **file_get_contents**

Пошто је број отворених ресурса ограничен, када се заврше све потребне акције над датотеком, препоручује се њено затварање, помоћу функције **fclose**:

```
fclose ( $фд );
```

4.2.4.11. Спољашње датотеке

PHP у свом конструктору (језичкој основи) садржи неколико функција које омогућавају употребу спољашњих датотека (енгл.external files) у оквиру скрипте. Различити су разлози због којих се користе спољашње датотеке за смештање: заједничких променљивих, корисничких функција, класа, HTML кôда за различите странице / скрипте, података за приступ базама података. Смештањем у спољашње датотеке постиже се лакша прегледност главног кôда, уштеда времена, када је потребно да се промени неки податак из датотеке и омогућава се лакша локализација могућих грешака у кôду. Користе се четири функције:

```
include("путања/име_спољашње_датотеке");
require("путања/име_спољашње_датотеке");
include_once("путања/име_спољашње_датотеке");
require_once("путања/име_спољашње_датотеке");
```

Свакој од њих се прослеђује име датотеке и путања до ње. Навођење путање није обавезно у колико је она наведена у опцији include-path конфигурацијске датотеке php.ini или ако се датотека налази у истом директоријуму као и скрипта која позива спољашњу датотеку. Није обавезна ни употреба заграда, с обзиром да се ради о основним функцијама, најчешће се користе због лакше прегледности. Називи датотека, које се наводе као аргументи ових функција могу да буду произвољни, али се препоручује да њихове екстензије увек буду .php или .inc.php. У супротном, ако корисник покуша сам да отвори неку од оваквих датотека, интерпретатор неће неће препознати као PHP кôд и приказаће њихов садржај, као да се ради о обичним тексту.

Све четири функције веома слично раде: укључују читав садржај наведене датотеке у текућу скрипту. Разлика између функције include и require је у понашању у случају да датотека не постоји или није видљива на наведеној путањи. У том случају функција include само даје упозорење, док је функција require „строжа“ и даје грешку, која зауставља извршавање скрипте. Суфикс „_once“ се односи на функције и класе из спољашње датотеке. Њиме се појашњава да се деклрација објеката врши само једном, чиме се спречава вишеструко декларисање, које PHP не дозвољава, пошто иначе, иста спољашња датотека може да се наведе више пута, на различитим местима у програму.

Исто заглавље и подножје, па чак и мени, могу да се искористе за различите странице, чиме се постиже уштеди времена. У колико се користи ћирилица, спољашње датотеке најбоље је да буду сачуване у УТФ-8 формату [11].

4.2.4.12. Кључне речи

Кључне речи (енгл. keywords) су резервисане речи. Имају специјално значење и на располагању су у било ком делу програма (скрипте). Оне чине

основу PHP језика. Употреба њихових имена за дефинисање сопствених променљивих може довести до конфузије. Ипак, могу слободно да се користе као имена корисничких константи, функција, метода, класа и објеката [6].

Табела 8. Кључне речи

__halt_compiler()	abstract	and	array()	as
break	callable (od PHP 5.4)	case	catch	class
clone	const	continue	declare	default
die()	do	echo	else	elseif
empty()	enddeclare	endfor	endforeach	endif
endswitch	endwhile	eval()	exit()	extends
final	finally (od PHP 5.5)	for	foreach	function
global	goto (od PHP 5.3)	if	implements	include
include_once	instanceof	insteadof (od PHP 5.4)	interface	isset()
list()	namespace (od PHP 5.3)	new	or	print
private	protected	public	require	require_once
return	static	switch	throw	trait (od PHP 5.4)
try	unset()	use	var	while
Xor	yield (as of PHP 5.5)			

4.2.4.13. Рад са формама

Форма, формулар или образац (енгл. form) је један од значајних елемената динамичког аспекта већине интеренет апликација, у којима приказ веб-странице зависи од корисничких акција. Међутим, форма је заправо део HTML структуре, за груписање различитих елемената, чија је функција прикупљање података од корисника. У зависности од акције корисника, добијена информација се може проверити код корисника, затим се може послати као захтев серверу, на поновну проверу и обраду података, а у зависности од потребе на крају се може послати и повратна информација [10].

Пример HTML странице са уграђеним формуларом:

```
<!DOCTYPE html>
<html lang="mul" dir="ltr">
   <head>
      <meta charset="utf-8" />
      <title>Пример странице са формуларом</title>
   </head>
```

```
  <body>
    <form action="http://www.barmymaja.xtreemhost.com/index.php">
      <input id="searchInput" name="search" type="search" size="20"
             autofocus="autofocus" accesskey="F" />
      <select id="searchLanguage" name="language">
        <option value="sr" lang="sr" selected="selected"> Srpski</option>
        <option value="es" lang="es">Español</option>
        <option value="ru" lang="ru">Русский</option>
        <option value="en" lang="en">English</option>
      </select>
      <input class="formBtn" type="submit" value="  ⇒  " name="go"
style="margin-top: 20px;" />
      <input type="hidden" value="Go" name="go" />
    </form>
  </body>
</html>
```

Форме су заправо невидљиви контејнери постављени између ознака за почетак <form> и крај </form>. Битни атрибути ове тагови су:

- action - која садржи веб адресу на серверу на којој ће се извршити обрада информације добијене из формулара
- method - којом се дефинише метода за преношење података из формулара. Постоје две методе: GET и POST

Метода GET се користи ако се не наведе ни једна метода. Њоме се информација унета у формулар надовезује на веб адресу из атрибута action и има следећи облик:

http://веб_адреса?име_поља1=вредност1&име_поља2=вредност2 ...

при чему треба водити рачуна да постоје ограничења за укупну дужину веб адресе. Осим тога, ова метода није сигурна и не може се користити за слање лозинки и осталих личних података корисника.

Метода POST преноси информацију на нешто сигурнији начин, нема ограничења у величини информације, а шаље се као део документа у облику улазног тока података, формирајући парове:

име_поља1=вредност1&име_поља2=вредност2

4.2.5. Техничке могућности

Развојни тим PHP-а се састоји од неколико десетина програмера, и још неколико десетина радника који раде на другим пројектима везаним за PHP, као што је PEAR и документација PHP-а. Поред овога, PHP-у су добровољно доприносили многи програмери широм света. Брз развој је проузроковао да PHP поседује велики број библиотека и функција, али и проблем неконзистентности у именовању уграђених функција.

Следи детаљан списак могућности које PHP нуди кроз своје библиотеке и додатке:

- Комуникација са базама података; подржане базе:
  - MySQL
  - mSQL
  - PostgreSQL
  - SQLite
  - Sybase
  - ODBC
  - Oracle
  - Microsoft SQL
  - DB++
  - dBase
- Рад са XML документима; независне библиотеке:
  - XML DOM
  - XMLReader и XMLWriter
  - SimpleXML
- Рад са текстовима у страним језицима
  - Библиотека Gettext
  - Функције за рад са нискама вишебајтних карактера
- Рад са великим бројевима; библиотека GMP (Енг. GNU Multiple Precision)
- Рад са датумима и календаром (подржава грегоријански и јулијански календар)
- Креирање PDF докумената; независне библиотеке:
  - Haru PDF
  - PDFLib
- Компресија; подржани алгоритми:
  - Rar
  - ZIP
  - Bzip2
  - Zlib
- Криптовање (функција mcrypt); подржани алгоритми:
  - DES
  - TripleDES
  - Blowfish
  - 3-WAY
  - SAFER-SK64

  - SAFER-SK128
  - TWOFISH
  - TEA
  - RC2
  - GOST
  - RC6
  - IDEA
- Рад са сликама
  - Уређивање слика (библиотека GD)
  - Читање мета-информација о сликама (Библиотека Exif)
- Рад са датотечним системом (директоријумима и датотекама)
- Кеширање стандардног излаза
- Конзолна графика (библиотека NCurses)
- Рад са штампачем
- Мрежна комуникација преко сокета
- Интеграција PHP-Јава
- Директан приступ сервису електронске поште
- Рад са стандардним протоколима:
  - FTP
  - HTTP
- Извршавање екстерних извршних датотека
- Рад са COM и .NET објектима за Виндоус

4.2.6. Компатибилност

PHP је подржан у већини популарних оперативних система, укључујући Unix, Linux, Microsoft Windows и Mac OS.

4.2.7. Доступност и лиценца

PHP се може бесплатно преузети широм Интернета и на званичном сајту PHP-а, а лиценциран је PHP лиценцом.

4.3. JavaScript

JavaScript је скриптни језик који се користи на webу за:

- проверу унесених података у обрасцу - пре него што их корисник пошаље на сервер. Овим се штеди процесорска снага клијента, а корисницима се убрзава чекање на одговор.
- детектовање врсте претраживача (browser) коју користи корисник – како поједини прегледачи различито приказују исте HTML ознаке, на овај начин је могуће кориснику приказати дизајн који је специфично израђен за његов претраживач.
- читање и чување података у cookies – тиме је могуће чувати поставке корисника, те приликом идуће посете корисника, поставити исте поставке
- омогућава једноставно програмирање HTML дизајнерима – због својег једноставног језика особама који нису програмери омогућава уметање једноставних функција које додају динамику у изгледу
- динамичко додавање и мењање садржаја – могуће је додати или променити текст или део HTML кôда на HTML страницу
- реаговање на догађаје (енг. events) – извршавање дела JavaScript кôда, након појављивања неког догађања, као што је завршено учитавање странице, позиционирање миша над неким HTML елементом и слично.

Сама синтакса овог језика је врло слична језику C, а његов кôд се налази унутар HTML кôда странице. Због сличности имена са програмским језиком Java, већина мисли да су то једнаки језици, али су JavaScript и Java различити и по концепту и по дизајну.

JavaScript се извршава на клијентском рачунару, и врло је једноставан програмски језик, док је Java врло моћан и комплексан језик налик на C/C++/C# који се извршава на самом клијенту [7].

Сам кôд JavaScripta се уписује унутар HTML странице. Ознака за почетак JavaScripta је HTML ознака <script language="JavaScript" type="text/javascript">, а za kraj </script>.

Пример најједноставнијег програма писаног у језику JavaScript је исписивање неког текста. За исписивање текста најједноставније је користити наредбу document.write().

```
<html>
	<body>
		<script language="JavaScript" type="text/javascript">
			document.write("Hello World!")
		</script>
	</body>
</html>
```

Ако се погледа HTML кôд у претраживачу, може се приметити да укључује изворни кôд написан у језику JavaScript. То је карактеристика овог језика, за разлику од PHP-а код којег се у претраживачу може видети само резултат извршавања, али не и сам кôд. Ипак JavaScript је врло раширен и готово све веб странице га користе.

JavaScript је могуће уметнути у HTML неограничени број пута, а тренутак извршавања зависи од места где је JavaScript кôд постављен унутар странице:

- Ако је JavaScript постављен унутар BODY дела – JavaScript се извршава приликом учитавања странице
- Ако је JavaScript постављен унутар HEAD дела – JavaScript се извршава када се позове функција унутар њега. Функција се може позвати из БОДУ дела или када се активира неки догађај
- Ако је JavaScript кôд у додатној датотеци – JavaScript се извршава једнако као да се сам кôд налази у HEAD делу. JavaScript се може позивати из додатне датотеке, само је унутар HTML кôда то потребно дефинисати, како би претраживач знао коју датотеку још треба преузети. JavaScript дефинисан на овај начин чини HTML кôд прегледним, а може се и једноставно додати на више HTML страница [7].

Ако се JavaScript кôд налази у одвојеној датотеци, тада у датотеци није потребно писати кôд унутар ознака <script>, али је потребно доделити наставак .js.

```
<html>
	<head>
		<script src="funkcije.js"></script>
	</head>
	<body>
	</body>
</html>
```

Овим примером показано је како се из HTML-а позива JavaScript датотека „funkcije.js“.

JavaScript синтакса има сличности са PHP синтаксом. Тако се коментари наводе исто као код PHP-а. Употреба израза if и switch, као и петљи while, do while и for иста је као код PHP језика. Употреба оператора је такође иста.

Употреба функција једнака је оној која се користи у програмском језику C или скриптном језику PHP. Променљиве које се декларишу унутар функција, могу се користити једино из функција, док оне које су декларисане изван функција су глобалне променљиве и могу се користити унутар свих функција.

4.3.1. Вредности и променљиве

JavaScript разликује следеће типове вредости:

- Бројеве, као 42 или 3.14159
- Логичке (boolean) вредности, true или false
- Стрингове, као што је "Поздрав свима!"
- null, специјални вреност која значи ништа.

JavaScript не прави посебну разлику између целих и децималних (реалних) бројева..

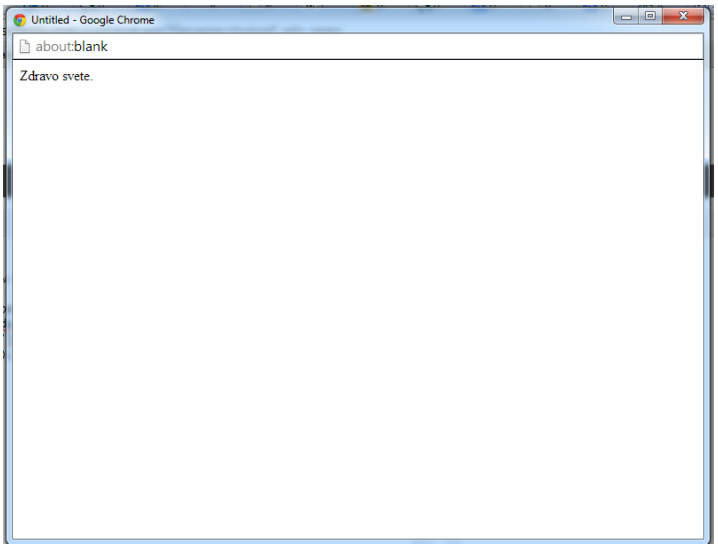
Тип променљиве у језику JavaScript није потребно дефинисати. JavaScript разликује велика и мала слова, па ове променљиве сматра различитима: Brojac, brojac, BROJAC, BROjac,… Променљиве морају започети словом или ознаком подцрте '_' [7].

4.3.2. JavaScript објекти

JavaScript је објектно орјентисан језик. Када се помоћу претраживача приступа неком HTML документу на вебу, онда се креира низ JavaScript објеката са својствима и могућностима које се базирају на HTML-у и другим деловима садржаја учитаног документа. Ови објекти су повезани хијерархијски у скалду са структуром HTML странице.

Од свих објеката најчешће су у употреби објекти window и document. Објекат window има метод који омогућава да се креира нови прозор (window) помоћу open() и close() метода.

```
<html>
	<head>
		<title>Poruka u novom prozoru</title>
		<script>
			newwindow=window.open();
			newdocument=newwindow.document;
			newdocument.write('Zdravo svete.');
			newdocument.close();
		</script>
	</head>
</html>
```



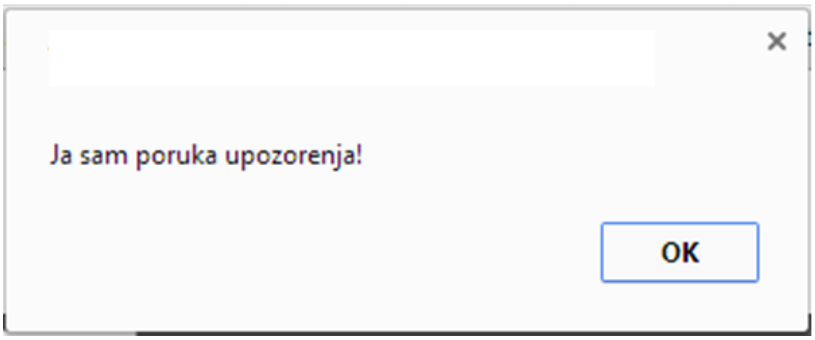
Слика 6. Резултат скрипта са називом „Порука у новом прозору“

Могу да се креирају и мали прозори (message boxes) коришћењем alert(), confirm(), и prompt() метода. Сваки од ових малих прозора приказује текст који стоји између заграда.

Пример прозора alert():

```
<!DOCTYPE html>
<html>
	<body>
		<button onclick="myFunction()">Probaj</button>

		<script>
			function myFunction() {
			  alert("Ja sam poruka upozorenja!");
			}
		</script>

	</body>
</html>
```

У претраживачу је приказано дугме Probaj . Када се кликне на ово дугме појављује се мали прозор:



Слика 7. Порука упозорења [8]

Објекат document представља HTML страницу. Објекат document садржи низове у којима су меморисане све компоненте од којих се састоји садржај веб странице као што су слике, линкови, форме итд. Може се приступати свим тим елементима и са њима повезаним методама преко ових низова.

Објектима из ове предефинисане хијерархије се може приступати, и могу се модификовати. Да би се приступило неком од елемената, мора се специфицирати његов назив као и називи свих његових предака уназад све до објекта document. Тачка, '.', се користи као сепаратор у тој хијерархији имена. Генерално говорећи, сваки елемент или објекат има оно име које је дато у NAME атрибуту одговарајућег HTML тага. На пример, следећа конструкција се односи на вредност (value) текстуалног податка (text1) који је део форме чије је име mojaforma, а која је део текућег документа.

```
document.mojaforma.text1.value
```

Истом том елементу се може приступити и преко горе поменутих низова. У горњем примеру, ако је форма mojaforma била права форма у документу и ако је text1 био треће поље у форми, онда би исти елемент могао да буде реферисан и на следећи начин:

```
document.forms[0].elements[2].value
```

Функције (методе) објеката могу бити позиване на сличан начин. На пример, следећа наредба ресетује (брише) другу форму у документу [7].

```
document.forms[2].reset();
```

Својства, методи и процедуре за обраду догађаја објеката JavaScripta дата су у следећој табели:

Табела 9. Хијерархијски објекти

Објекат	Својства	Методе	Процедуре за обраду догађаја
Window	defaultStatus frames opener parent scroll self status top window	alert blur close confirm focus open prompt clearTimeout setTimeout	onLoad onUnload onBlur onFocus
Frame	defaultStatus frames opener parent scroll self status top window	alert blur close confirm focus open prompt clearTimeout setTimeout	нема
Location	hash host hostname href pathname por protocol search	reload replace	нема
History	length forward go	back	нема
Navigator	appCodeName appName appVersion mimeTypes plugins userAgent	javaEnabled	нема
document	alinkColor anchors applets area bgColor cookie fgColor forms images lastModified linkColor links location referrer title	clear close open write writeln	нема

image	border complete height hspace lowsrc name src vspace width	none	нема
form	action elements encoding FileUpload method name target	submit reset	onSubmit onReset
text	defaultValue name type value	focus blur select	onBlur onCharge onFocus onSelect

4.4. WAMP

Да би се горе описаним технологијама програмирало у localhost-у потребна је апликација која ће симулирати рад интернет сервера. WAMP је софтверски пакет који представља скраћеницу од Windows, Apache, MySQL, а P се може односити на PHP, Python или Perl .

WAMP омогућава да се на личном рачунару симулира рад интернет сервера, и да се провери функционалност које је имплементирана у сајт или интернет презентацију.

У скраћеници WAMP - W означава Windows оперативни систем, док се A односи на Apache веб сервер који омогућава да се на рачунару налази истовремено и корисник (browser) и локални сервер. MySQL је систем за управљање релационим базама података, док је PHP већ описани скриптни језик за израду динамичнг веб садржаја, који генерише нову веб страну сваки пут када корисник приступи серверу. Такође, у пакету могу бити укључени и додатни програми, као што је phpMyAdmin, креиран у PHP-у, који између осталог служи за креирање, модификовање и брисање база података, табела, колона и редова, користећи SQL програмски језик помоћу графичког интерфејса (за разлику од стандардне командне линије).

Инсталација еквивалентна WAMP-у на Windows ОС-у, на Linux-у се зове LAMP.

4.4.1. Лиценца

Осим windows оперативног система, компоненте WAMP/LAMP-а доступне су као freeware (слободан софтвер). То значи да за постављање динамички генерисане веб странице нису потребна било каква велика улагања.

Поред freeware лиценце за WAMP пакет, постоје и donationware као и комерцијални пакети WAMP-а.

5. Базе података

База података је колекција података организованих за брзо претраживање и приступ, која заједно са системом за администрацију, организовање и меморисање тих података, чини систем базе података. Из угла корисника, подаци су на неки логички начин повезани. Они представљају неке аспекте реалног света.

Корисници приступају бази података првенствено преко упитника. Коришћењем кључних речи и сврставањем команди корисници могу брзо да пронађу, преуреде, групишу и одаберу област у многим записима које треба вратити или помоћу којих треба саставити извештаје о нарочитој скупини података у складу с правилима дотичног система вођења базе података.

Постоје различите врсте база података, зависно од тога на који начин су подаци интерно организовани. Тако се разликују хијерархијске, мрежне (CODASYL), релационалне, објектно-оријентисане, објектно-релационе, прилагођене за WEB, XML и мултимедијске базе података.

Подаци су представљени на униформни начин (нпр. у релационим базама података подаци су организовани у табелама), што олакшава приступ и коришћење од стране екстерних програма. Тако једну базу података може користити низ различитих програма, писаних у различитим програмским језицима [9].

5.1. Релационе базе података

Релациона база података је посебан тип базе података код којег се организација података заснива на релационом моделу. Подаци се у оваквим базама организују у скуп релација између којих се дефинишу одређене везе. Релација се дефинише као скуп н-торки са истим атрибутима, дефинисаних над истим доменима из којих могу да узимају вредности. У релационим базама података, свака релација мора да има дефинисан примарни кључ, који представља атрибут помоћу којег се јединствено идентификује свака н-торка.

Релација опционо може да поседује и спољни кључ, преко којег остварује везу са другим релацијама.

Управљање оваквим базама података се реализује преко система за управљање релационим базама података. Међу најпопуларнијим таквим системима данас су: Microsoft SQL Server, Oracle Database, MySQL и други. Већина тих система користи упитни језик SQL за манипулацију подацима.

Свака релација може без икаквих проблема да се представи у табеларном облику, али и поред тога, релација и табела нису исто. То је из разлога што је код табела битан редослед редова и колона, док код релација није битан редослед атрибута и н-торки. На пример, табела 1 има колоне са редоследом: име, презиме, јмбг. Табела 2 има колоне са редоследом: јмбг, презиме, име. Те две табеле се не сматрају истим, без обзира што имају исте називе колона. И поред тога што релација и табела нису синоними, данас се углавном свака релација назива табелом, тј. њени елементи се поистовећују са елементима који чине једну табелу. Из тог разлога су у следећој табели дати појмови из релационог и табеларног модела који се односе на исто.

Табела 10. Појмови релационог и табеларног модела

Табела	Релација
Ред табеле	Н-торка
Колона, назив колоне, вредност колоне	Атрибут, назив атрибута, вредност атрибута
Табела	Базна релација
Скуп назива колона	Релациона шема
Поглед, резултат упита	Изведена релација

Основни појмови:

- Н-торка се дефинише као уређени скуп од „н“ елемената.
- Атрибут представља име којим се у релационом моделу, идентификује сваки од елемената једне н-торке. На пример: атрибути су "студент", "статус" и "просек" а елементи н-торке су "Марија", "Буџет" и "9.2''.
- Релација се дефинише као скуп н-торки који имају исте атрибуте којима су идентификовани њихови елементи, и који узимају вредности из истих домена. Пример релације:

(Студент: "Марија", **Статус**: "Буџет" , **Просек**: 9.2**)**
(Студент: "Јован" , **Статус**: "Буџет" , **Просек**: 9.1**)**
(Студент: "Бојан" , **Статус**: "Самоф" , **Просек**: 7.8**)**

У релационим базама података постоје две врсте релација: базне и изведене. **Базне релације** су релације које се већ налазе у бази података, тј. складиштене су на хард диску или неком другом медијуму за чување података. **Изведене релације** (или погледи) су релације које се добијају читањем и комбиновањем података из једне или више базних релација постављањем услова упитним језиком, о којем ће бити речи касније.

У свакој релацији мора да постоји један или више атрибута заједно, чије вредности јединствено идентификују сваку н-торку у тој релацији. Тај атрибут, или група атрибута се назива примарним кључем релације. У случају када један атрибут јединствено идентификује сваку н-торку у релацији имамо прост примарни кључ, а ако је идентификују два или више атрибута, онда је у питању сложени примарни кључ. Примарни кључ је дакле скуп од К елемената неке релације, који морају да задовољавају следећа два услова: особину јединствености и особину нередудантности. Особина јединствености подразумева да не могу да постоје било које две н-торке са истом вредношћу атрибута „К“. Особина нередудантности подразумева да ако се изостави било који атрибут из К, губи се особина јединствености. У доњем примеру, атрибут "Шифра" представља прост примарни кључ релације.

Спољни кључ представља атрибут (или групу атрибута) неке релације Р1, који у њој није примарни кључ, али јесте у некој другој релацији Р2. С тим у вези, релација Р1 се повезује са релацијом Р2 преко спољног кључа. Да би веза између те две релације била исправна, морају се задовољити правила **референцијалног интегритета**. Укратко, под референцијалним интегритетом се подразумева да све вредности које узима атрибут који представља спољни кључ, морају да постоје и у релацији у којој је тај атрибут примарни кључ.

Формализми за манипулисање подацима, који су саставни део релационог модела су : релациона алгебра и релациони рачун. Упитни језици који су у саставу конкретног система за управљање релационим базама података се заснивају на једном од ова два формализма, или на њиховој комбинацији [9].

5.2. Системи за управљање базама података

Систем за управљање базом података (СУБП) обезбеђује скуп услуга које заједно пружају широку подршку апликацијама које захтевају складиштење великих количина података које ће делити више корисника. Међу најважнијим се налазе:

1. Управљање секундарним складиштењем. Могућност за складиштење података на магнетним дисковима или другим медијима за дугорочно складиштење, тако да подаци могу да опстану након извршења програма на којима су креирани. Како би се подржало ефикасно складиштење и приступ великим количинама података потребни су софистицирани системи управљања секундарним складиштем. Системи за управљање обично укључују услуге за брзи приступ одређеним подацима помоћу индекса, чиме се умањује потребан број различитих приступа секундарном складишту, груписање сродних података и обезбеђивање, где је то могуће, ефикасног коришћење бафера који у меморији рачунара на којем се извршава систем базе података, задржава податке којима је скоро приступано.

2. Контрола упоредног приступа. Велике базе података су драгоцене за организације које их поседују, а често је потребно да им приступи више корисника истовремено. Неконтролисане промене у дељеном простору за складиштење, могу довести до оштећења интегритета базе, или губљења података као резултат интерференције програма на непредвидив начин. Да би се то спречило, системи за управљање базом података поседују одређене заштитне механизме како би се осигурало да сваки корисник приступа бази података изоловано од других, и такође са довољним гаранцијама о понашању система када више корисника покуша учитати или уписати, исти податак у исто време.

3. Рекуперација. Многе апликације за базе података су активне 24 сата дневно, неретко им приступа много различитих програма који се често извршавају на различитим рачунарима, упоредно користећи базу података. Као резултат тога, неизбежно је да се деси да програми који приступају бази података праве грешке, као и да се деси да се рачунар на коме се извршава систем за управљање базом података блокира. Према томе, систем за управљање базом података мора да буде флексибилан, односно толерантан на грешке у софтверу и отказивање хардвера, умањујући шансе да база података буде оштећена, или да се деси да се информације трајно изгубе. Како би се избегли ови проблем и постојала могућност за рекуперацију базе података, често се ради привремена репликација података на више места, тако да има довољно информација да се омогући аутоматско проналажење и исправљање различитих врста грешака када се оне десе.

4. Безбедност. Већина великих система база података садрже податке који не треба да буду видљиви неким корисницима, као и да им не буде омогућено да

их мењају. Из тог разлога, неопходно је да постоји један општи механизам који треба да обезбеди да ниједан корисник не приступи информацијама којима по правилу не треба да приступи, и још важније, да постоје механизми контроле над корисницима који имају једну, или више дозвола за измене садржаја базе података (нпр. радници једног предузећа не би требало да буду у стању да приступе и врше измене ставки који се тичу њиховог личног дохотка, и томе слично). Према томе, системи за управљање базом података по правилу обезбеђују механизме који омогућавају администратору базе података да контролише ко и где може да приступи као и шта може да уради у бази података [9].

5.3. Упитни језици

Упитни језици су намењени за комуникацију са релационом базом података, тј. за креирање релационих шема и ажурирање и читање података из релација. Могу се најопштије класификовати на: језике добијене надградњом процедуралних програмских језика, језике засноване на релационом рачуну домена или н-торки, језике засноване на релационој алгебри и језике који се заснивају на комбинацији релационе алгебре и релационог рачуна. Најзаступљенији упитни језик данас је SQL(Structured Query Language), развијен 80-их година прошлог века од стране истраживачке IBM лабораторије у Сан Хозеу у Калифорнији. Подржавају га готово сви системи за управљање релационим базама података. И поред његове велике распрострањености, велики значај се придаје и упитном језику QUEL(QUEry Language), који је развијен такође у Калифорнији, на универзитету у Берклију.

5.4. Structured Query Language (SQL)

SQL представља последњу фазу развоја упитних језика од стране истраживачке IBM лабораторије. Његови претходници развијени од стране те IBM лабораторије су били упитни језици SQUARE и SEQUEL. SQL стандард је објављен 1989. године и одмах је био широко прихваћен на тржишту. Неки од битних својстава су му: језик за дефинисање података(Data Definition Language), језик за манипулацију подацима(Data Manipulation Language), спољашње и унутрашње спајање, каскадно ажурирање и брисање, скуповне операције (унија, пресек и разлика) спајање итд. Приказаћемо неке најосновније појмове везане за овај упитни језик.

5.4.1. Елементи језика

Овде ћемо видети неколико елемената SQL језика помоћу којих се креирају јединични искази. SQL језик је подељен у неколико језичких елемената који укључују:

1. Исказе који могу имати трајно дејство на структуру и податке, или који могу вршити контролу трансакција, ток програма, конекција, сесија, или дијагностику.
2. Упите који приказују податке у зависности од задатих критеријума.
3. Израза који дају вредности у облику скаларних вредности или табела које се састоје од редова и колона у којима се налазе подаци.
4. Тврдње које спецификују стања која се могу испитивати логичким вредностима (Boolean truth values) користе се за ограничења у Исказима и Упитима, као и за промену програмског тока.
5. Услови који су у већини случајева компонента Исказа и Упита.

SQL игнорише прореде (размак) у исказима и упитима, што олакшава читљивост кôда.

SQL искази садрже (";") који служе као терминатори исказа (завршетак реда).

5.4.2. Упити

Упити су врло честа операција у MySQL базама података. Синтакса изгледа у облику изјаве: SELECT ime_atributa FROM tabela WHERE ...

ime_atributa је име/имена жељених атрибута.

Ако уместо имена атрибута ставимо симбол ("*") добијамо као излаз сва поља из базе.

SELECT је један од најкомплекснијих израза (кључних речи) у SQL-у којим се селектују жељени атрибути.

Помоћу кључне речи FROM бира се табела у којој се налазе жељени атрибути.

WHERE је исказ поређења помоћу кога се ограничава излаз, одн. Број редова добијених на излазу.

GROUP BY пише се након WHERE исказа, служи да групише или комбинује редове са вредностима који су у одређеној релацији у елементе мањег сета редова или да избаци редове са истим вредностима. Користи се са исказом HAVING и функцијом агрегације (aggregate).

ORDER BY је исказ помоћу кога бирамо (идентификујемо) колону пеко које ће редови бити сортирани. Сортирање може бити у растућем (ascending) или опадајућем (descending) поретку.

Пример:

```
SELECT *
FROM knjige
WHERE cena > 100.00
ORDER BY naslov;
```

5.4.3. Управљање (манипулисање) подацима

Стандардни сет елемената који се користи за манипулисање подацима је DML (**D**ata **M**anipulation **L**anguage). DML је део језика који служи за унос, промену и брисање података.

INSERT се користи за додавање редова у табелу:

```
INSERT INTO ime_tabele (polje1, polje2, polje3) VALUES ('test', 'N', NULL);
```

UPDATE се користи за промену сета вредности у постојећим редовима табеле:

```
UPDATE ime_tabele SET polje1 = 'nova_vrednost' WHERE polje2 = 'N';
```

DELETE уклања један или више постојећих редова из табеле:

```
DELETE FROM ime_tabele WHERE polje2 = 'N';
```

MERGE се користи за комбиновање података из више табела.

5.4.4. Дефинисање података

За дефинисање података користи се сет елемената DDL (**D**ata **D**efinition **L**anguage). DDL дозвољава кориснику да дефинише нове табеле и асоцијативне елементе. Основни елементи DDL-а су CREATE, ALTER, RENAME, TRUNCATE и DROP искази:

CREATE креира табелу (објекат) унутар базе података.
DROP брише постојећи објекат унутар базе.
TRUNCATE брише све податке из табеле.
ALTER омогућује кориснику да на разне начине модификује постојећи објекат (нпр. додавање колоне у постојећој табели).

Пример:

```
CREATE TABLE ime_tabele (
  ime_polja1   INT,
  ime_polja2   VARCHAR (50),
  ime_polja3   DATE        NOT NULL,
  PRIMARY KEY (ime_polja)
);
```

5.4.5. Контрола над подацима

Трећи сет је DCL (**D**ata **C**ontrol **L**anguage). DCL управља нивоима ауторизације и даје контролу кориснику какав тип приступа може имати треће лице на бази података (нпр.само читање или и могућност манипулације). Две кључне речи DCL-а су:

GRANT даје право једном или више корисника на одређену операцију или више операција над објектом у табели.

REVOKE онемогућује или ограничава могућност корисника да изврши операцију или више операција унутар базе.

Пример:

GRANT SELECT, UPDATE ON ime_tabele TO jedan_korisnik, drugi_korisnik;

Са аспекта безбедности и очувања целине базе података није препоручљив већи приступ базама.

5.5. MySQL

MySQL је вишенитни, вишекорисnички SQL систем за управљање базама података. Систем ради као сервер, обезбеђујући вишекорисnички интерфејс за приступ бази података.

Библиотеке за приступ бази података MySQL постоје за већину програмских језика, чији облик зависи од датог програмског језика. Додатно, постоји ODBC интерфејс по називу MyODBC који дозвољава приступ бази података за оне програмске језике који подржавају ODBC интерфејс, као што су ASP и ColdFusion. MySQL сервер и званично подржане библиотеке су углавном писани у програмским језицима C и C++.

Процењује се да постоји око десет милиона инсталација MySQL-а.
MySQL је популаран у развоју веб апликација, нарочито у комбинацији "ЛАМП" (Линукс-Апач-MySQL-PHP/Perl/Piton). Његова популарност се веже за популарност PHP-а, који се обично комбинује са MySQL-ом.

Фирма MySQL AB, смештена у Шведској, је јединствени власник и спонзор пројекта, и држи највећу већину права над кôдом. MySQL AB пружа MySQL као софтвер отвореног кôда / слободан софтвер под ГНУ-овом Општом јавном лиценцом.

Фирма такође развија и одржава систем, а зарађује преко наплаћивања техничке подршке, сервиса, као и продаје лиценци за комерцијалну употребу програма.

За администрацију базе података MySQL, администратори користе или интерфејс у облику командне линије, или графички интерфејс "MySQL администратор" и друге.

Поред алата које производи фирма MySQL AB, постоји и неколико комерцијалних и некомерцијалних алата приступачних на тржишту. Алат PhpMyAdmin је слободан софтвер чији је интерфејс у облику веб странице а који је написан у програмском језику PHP.

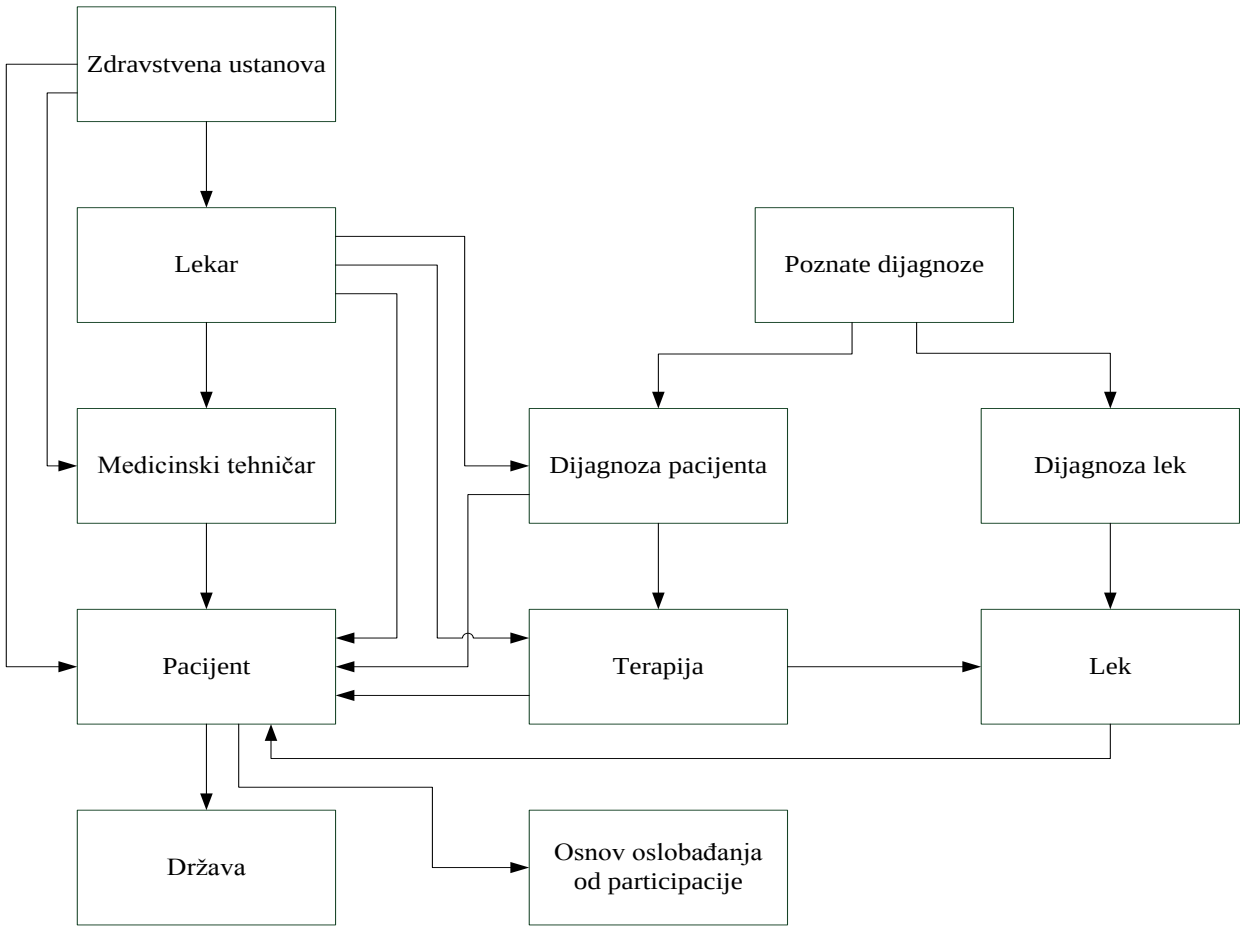
MySQL ради на већини оперативних система.

6. Реализација веб апликације за штампање рецепата

Да би се реализовала једна веб апликација, треба прво пројектовати базу података. База података је основ сваке апликације. У фази пројектовања базе података треба најпре препознати објекте реалног света, који ће бити табеле (ентитети), њихове атрибуте, утврдити зависност ентитета, дефинисати примарне кључеве табела. Релација, као основни концепт релационог модела је заправо математичка релација, и има једноставан приказ у облику дводимензионалне табеле са подацима. Релациони модел даје основу за језик високог нивоа (MySQL) за приступ подацима.

6.1. Дијаграм зависности ентитета

Дијаграм зависности ентитета описује делове базе података који комуницирају међусобно. Дијаграм зависности ентитета приказан је на следећој слици, на којој се види који ентитети чине базу и како су повезани.

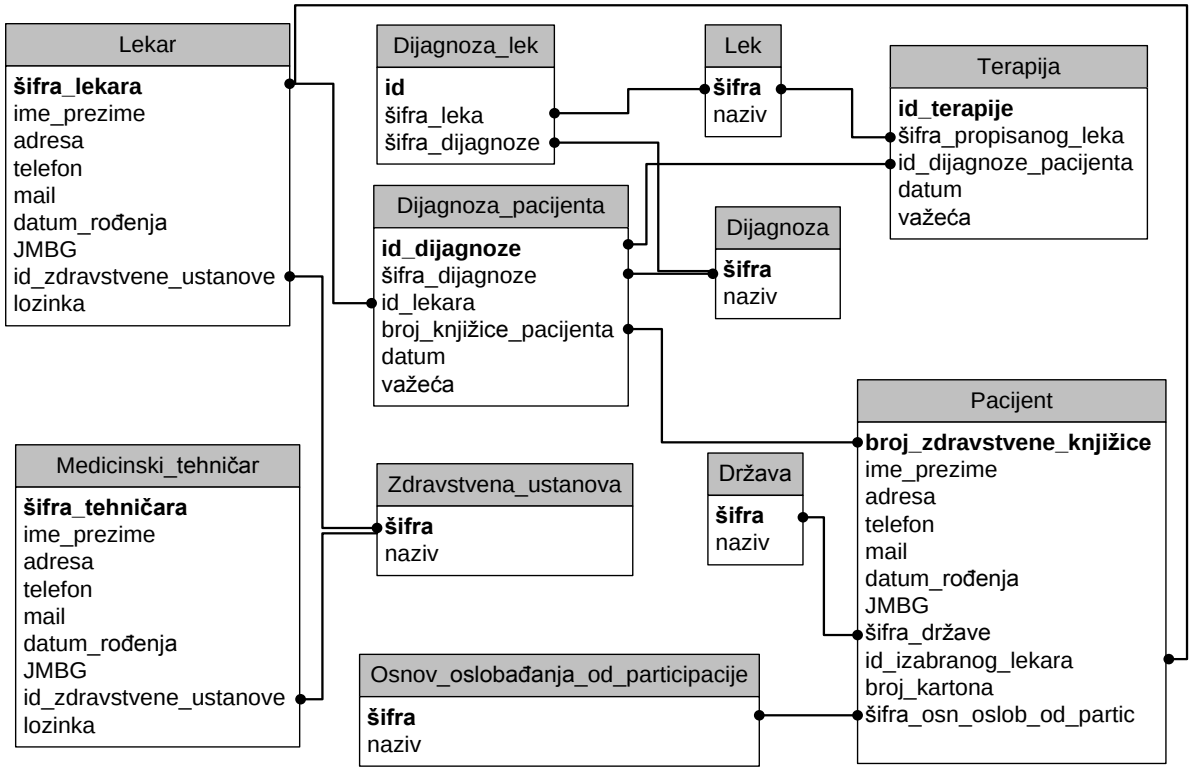


Слика 8. Дијаграм зависности ентитета

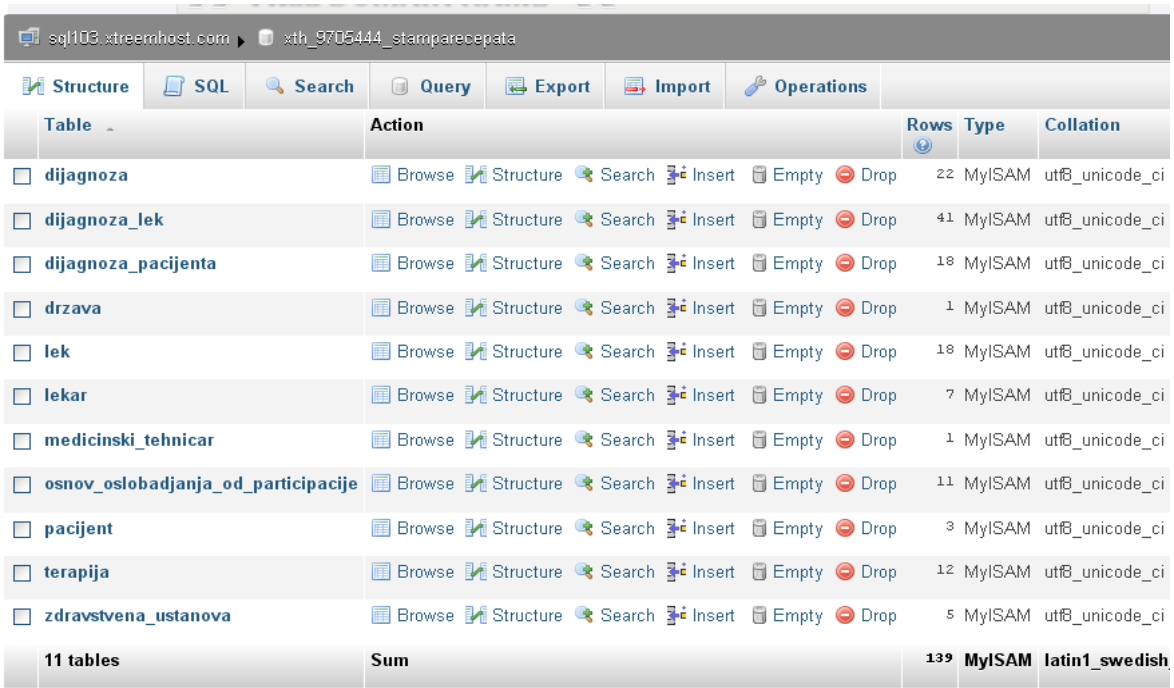
6.2. База података

База података се састоји од 11 табела. Све табеле су у релацији 1 према бесконачно.

Оно што је битно, јесте да не постоји редудантност података што је заправо циљ правилно пројектованог информационог система.



Слика 9. Релациони дијаграм базе података

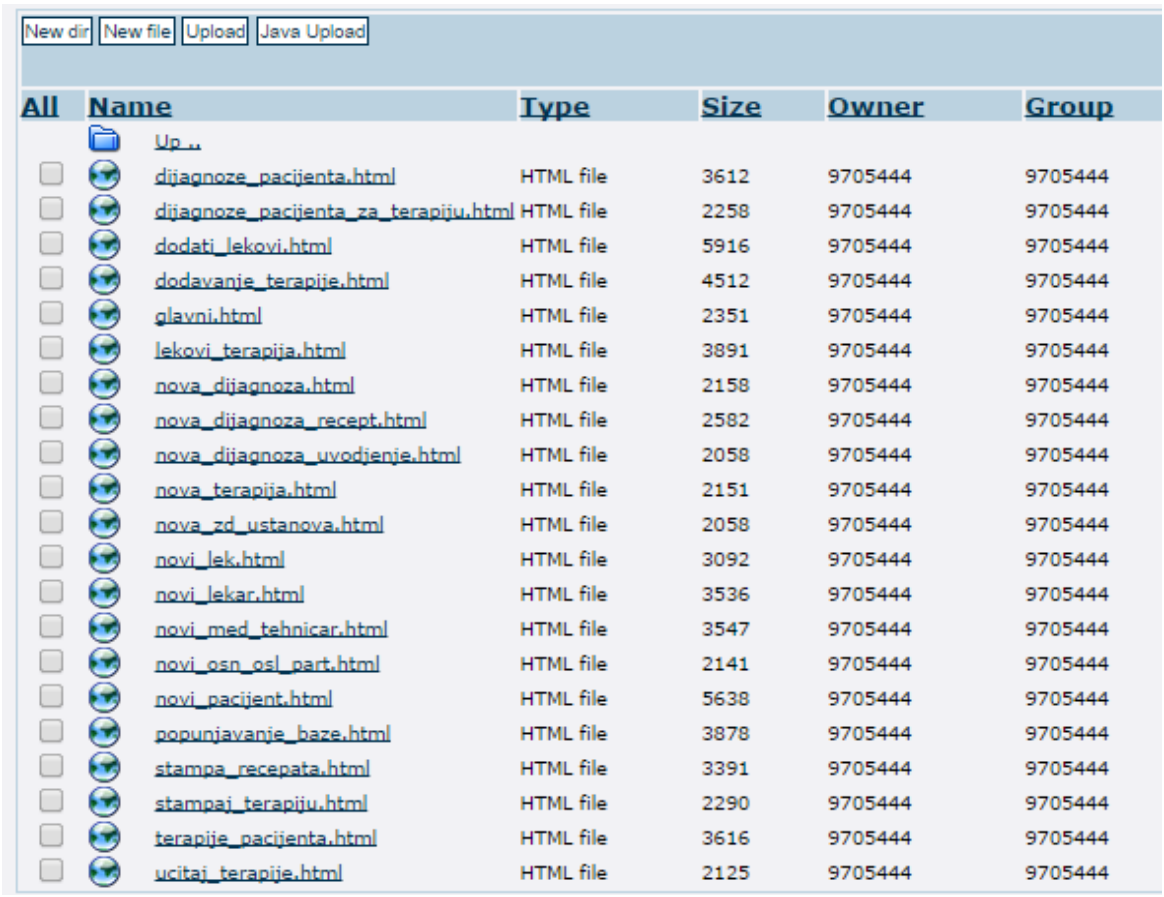


Слика 10. Изглед базе података у phpMyAdmin-у

6.3. Апликација

При писању ове апликације вођено је рачуна да се HTML стране одвоје од PHP-а. Сав PHP кôд се налази у фајловима са екстанзијом .php, док се HTML стране налазе у фолдеру „template“ сем почетне стране која је у главном директоријуму, а слике су у фолдеру „images“.

Слика 11. Почетни (root) директоријум на xtreemhost-у



Слика 12. Директоријум „template“

Страна glavni.html укључује слику и заглавље, док се централни део мења, укључивањем осталих страна. PHP кôд који укључује главну страницу као и ону која треба да се прикаже у делу који се мења, дат је на следећој слици.

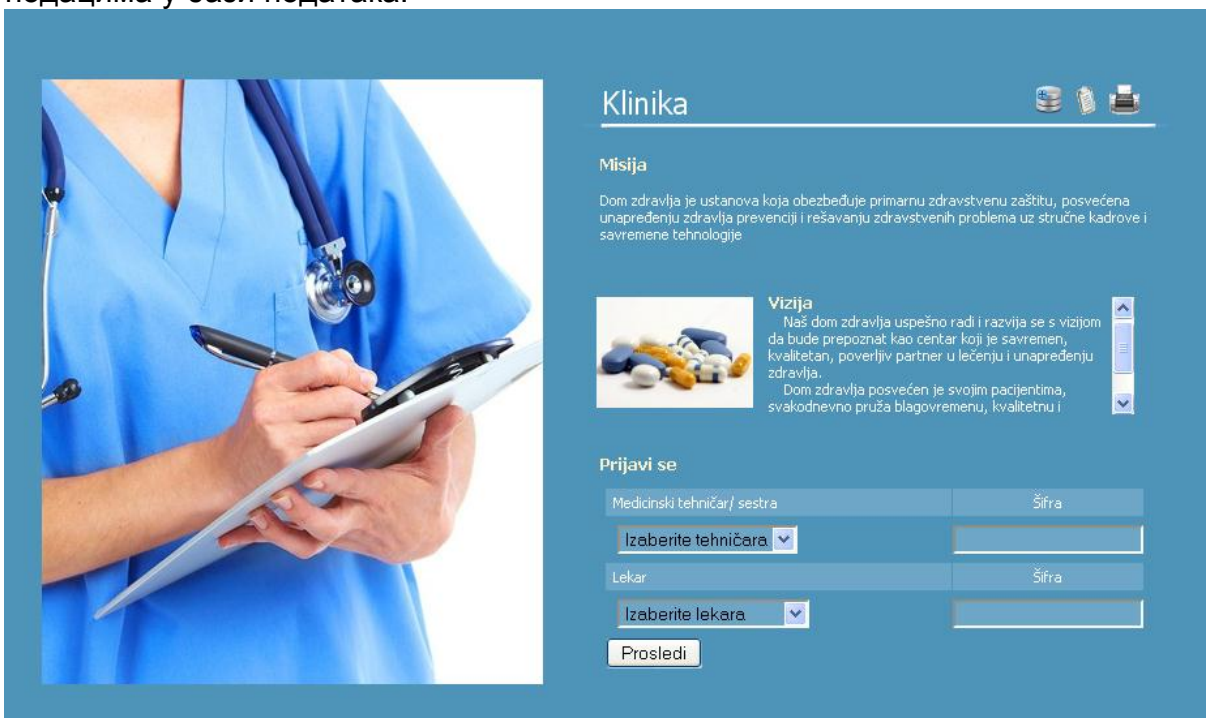
```
        $template->set_filenames (array ('body' => TEMPLATE_URL . "glavni.html"));
        $template->pparse ('body');

//*****************************  * F U N K C I J E  ***************************************

        function PrikaziStranu($htmlzaprikaz)
        {
                global $template;

                $template->set_filenames (array ('centralni_html' => TEMPLATE_URL . $htmlzaprikaz));
                $template->assign_var_from_handle ('CENTRALNI_HTML', 'centralni_html');
        }

        function Prikazi_centralni_html()
        {
                global $connection, $template, $Strane, $Strana;

                        foreach($Strane as $key => $val)
                        {
                                if($Strana == $key)
                                    PrikaziStranu($val);
                            }
        }
```

Слика 13. PHP кôд за учитавање страна из фолдера „template“

```
<table width="900" height="600" border="0" cellpadding="0" cellspacing="0">
    <tr>
        <td colspan="4" bgcolor="#4D94B8" width="766" height="84">
        <div style="margin-top: 32px; margin-bottom: 15px; margin-left: 20px;">
        <a href="?strana=popunjavanje_baze">
            <img src="images/right_arrow.png" width="15" height="15" border="0" hspace="0" vspace="0">
            <font class="white">Popunjavanje baze</font></a>
        <a href="?strana=nova_dijagnoza_recept">
            <img src="images/right_arrow.png" width="15" height="15" border="0" hspace="0" vspace="0">
            <font class="white">Nova dijagnoza/recept</font></a>
        <a href="?strana=stampa_recepata">
            <img src="images/right_arrow.png" width="15" height="15" border="0" hspace="0" vspace="0">
            <font class="white">&#352;tampa recepata</font></a></div>
        </td>
    </tr>
    <tr>
        <td valign="top" bgcolor="#4D94B8" width="30" height="428" >
        </td>
        <td background="images/DOKTOR22.jpg" style="background-repeat: no-repeat;"
                width="400" height="428" valign="top">
            </td>

        <td valign="top" bgcolor="#4D94B8" width="30" height="428" >
        </td>
        <td width="400" height="428" valign="top">
            {CENTRALNI_HTML}
        </td>
        <td bgcolor="#4D94B8" width="8" height="428">
        </td>
    </tr>
    <tr>
        <td colspan="4" bgcolor="#4D94B8" width="800" height="88" class="copy">
        <div style="margin-right: 7px; margin-bottom: 7px; margin-left: 336px; margin-top: 68px;"></div>
        </td>
    </tr>
</table>
```

Слика 14. Централна табела странице glavni.html

На почетној страни (Слика 15.) се лекари или техничари логују својим именом и шифром, како би дошли до стране на којој се може манипулисати подацима у бази података.



Слика 15. Почетна страница апликације

Постоје 3 опције:

1. Попуњавање базе - на овом линку могу се наћи стране за унос новог пацијента у базу, новог лекара, лекова, дијагноза и свих осталих ентитета базе.
2. Нова дијагноза/рецепт- где се може унети нова дијагноза или терапија (лек).
3. Штампа рецепата- опција којом се директно иде на страну са које се бира пацијент којем треба да се штампају рецепти, а онда се одабиром дијагнозе долази до странице са које се штампају рецепти.

Слика 16. Страница ulogovani.php

Slika 17. Страница за попуњавање базе

Слика 18. Додавање новог пацијента

Форма за унос новог пацијента дата је на следећој слици.

```
<form  action="ulogovani.php?dodaj_pacijenta=dodaj" name="myForm" method="post" onsubmit="return(validate());">
 <table>
     <tr height="20">
         <td colspan="3"></td>
     </tr>
     <tr>
         <td width="30"></td>
         <td width="300">Broj zdravstvene knji&#382;ice:</td>
         <td width="500" align="left"><input type="text" name="br_kjizice" id="br_kjizice"></td>
     </tr>
     <tr>
         <td width="30"></td>
         <td width="300">Ime i prezime:</td>
         <td width="500" align="left"><input type="text" name="ime_prezime" id="ime_prezime"></td>
     </tr>
     <tr>
         <td width="30"></td>
         <td width="300">Adresa:</td>
         <td width="500" align="left"><input type="text" name="adresa" id="adresa"></td>
     </tr>
     <tr>
         <td width="30"></td>
         <td width="300">Telefon:</td>
         <td width="500" align="left"><input type="text" name="tel" id="tel"></td>
     </tr>
     <tr>
         <td width="30"></td>
         <td width="300">e-mail:</td>
         <td width="500" align="left"><input type="email"  name="mail" id="mail"></td>
     </tr>
<tr>
    <td width="30"></td>
    <td width="300">Datum rodjenja:</td>
    <td><input type="date" name="datum" id="datum" value=""></td>
</tr>
<tr>
    <td colspan="2"></td>
    <td>
        <span style="font-size: 9px; color: #FFF8C6;"> Datum rodjenja dati u formi GGGG-MM-DD </span>
    </td>
</tr>
<tr>
    <td width="30"></td>
    <td width="300">JMBG:</td>
    <td width="500" align="left"><input type="text" name="jmbg" id="jmbg"></td>
</tr>
<tr>
    <td width="30"></td>
    <td width="300">Broj kartona:</td>
    <td width="500" align="left"><input type="text" name="br_kartona" id="br_kartona"></td>
</tr>
<tr>
    <td width="30"></td>
    <td width="300">Izabrani lekar:</td>
    <td width="500" align="left">
    <select id="sifra_lekara"  name="sifra_lekara">
        <option value="000" selected="selected">[izaberite]</option>
        <!-- BEGIN Prikazi_lekare -->
        <option value="{Prikazi_lekare.ID}">{Prikazi_lekare.NAZIV}</option>
        <!-- END Prikazi_lekare -->
    </select>
    </td>
```

```
</tr>
<tr>
    <td width="30"></td>
    <td width="300">Osnov oslob. od participacije:</td>
    <td width="500" align="left">
    <select id="sif_osl_particip"  name="sif_osl_particip">
        <option value="0" selected="selected">[izaberite]</option>
        <!-- BEGIN Prikazi_oslobadjanje_od_participacije -->
        <option value="{Prikazi_oslobadjanje_od_participacije.SIFRA}">
        {Prikazi_oslobadjanje_od_participacije.NAZIV}</option>
        <!-- END Prikazi_oslobadjanje_od_participacije -->
    </select>
    </td>
</tr>
<tr>
    <td width="30"></td>
    <td width="300">Dr&#382;ava:</td>
    <td width="500" align="left">
    <select id="sif_drzave"  name="sif_drzave">
        <option value="000" selected="selected">[izaberite]</option>
        <option value="00">Srbija</option>
        <option value="01">Austrija</option>
        <option value="02">Belgija</option>
        <option value="03">Bugarska</option>
        <option value="04">V.Britanija</option>
        <option value="05">Italija</option>
        <option value="06">Luksemburg</option>
        <option value="07">Madjarska</option>
        <option value="08">Nema&#269;ka</option>
        <option value="09">Rumunija</option>
        <option value="10">Francuska</option>
            <option value="11">Holandija</option>
            <option value="12">&#268;e&#353;ka</option>
            <option value="13">Hrvatska</option>
            <option value="14">Makedonija</option>
            <option value="15">Bosna i Hercegovina</option>
            <option value="17">Slovenija</option>
            <option value="18">Crna Gora</option>
            <option value="19">Slova&#353;ka</option>
            <option value="20">Poljska</option>
            <option value="21">Rusija</option>
            <option value="22">Irak</option>
            <option value="23">Ju&#382;na Koreja</option>
        </select>
        </td>
    </tr>
    <tr>
        <td width="30"></td>
        <td colspan="2"><input type="submit" value="Dodaj pacijenta"><br></td>
    </tr>
</table>
</form>
```

Слика 19. Форма за унос на страни novi_pacijent.html

Слика 20. Успешно додат пацијент и опција за додавање новог пацијента

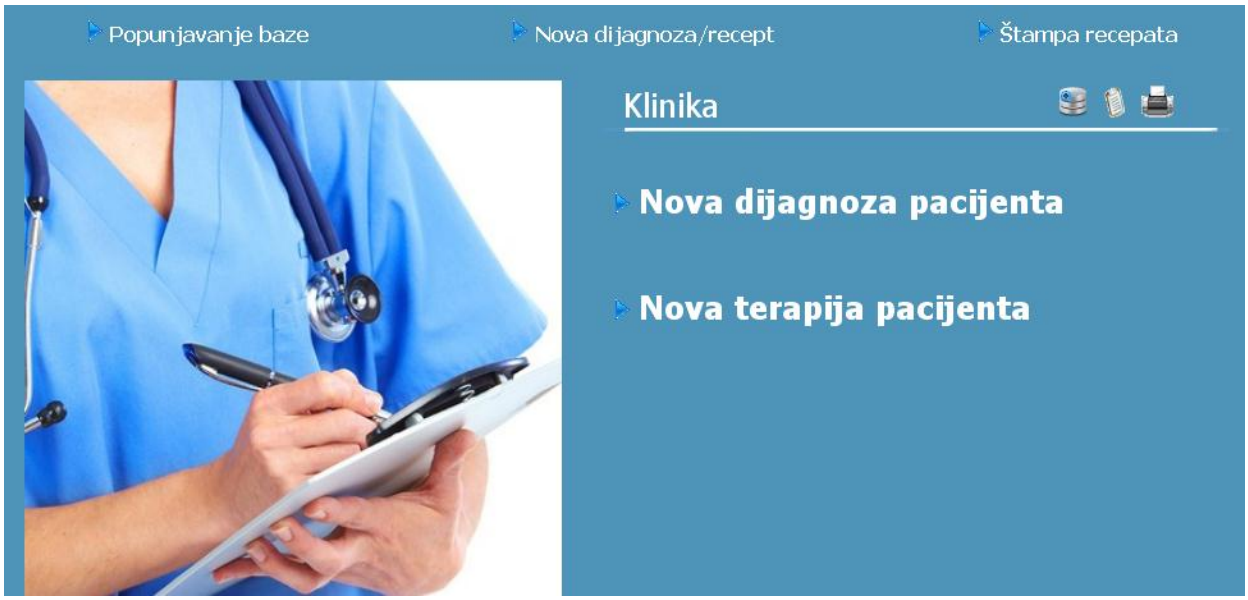
JavaScript кôд за проверу података који се уносе у базу дат је на следећој слици.

```
<script type="text/javascript">
<!--
function validate()
{


   if( document.myForm.br_knjizice.value == "" ||
     isNaN( document.myForm.br_knjizice.value )  )
         {
               alert( "Molim Vas unesite broj knji&#382;ice." );
    document.myForm.br_knjizice.focus() ;
    return false;
  }

   if( document.myForm.ime_prezime.value == "" )
   {
               alert( "Molim Vas unesite ime i prezime!" );
               document.myForm.ime_prezime.focus() ;
             return false;
  }

  if( document.myForm.adresa.value == "" )
   {
               alert( "Molim Vas unesite adresu!" );
               document.myForm.adresa.focus() ;
             return false;
  }
```

```
   var emailID = document.myForm.mail.value;
         atpos = emailID.indexOf("@");
         dotpos = emailID.lastIndexOf(".");
         if (atpos < 1 || ( dotpos - atpos < 2 ))
                  {
                        alert("Molim Vas unesite pravilan email.")
                         document.myForm.mail.focus() ;
                           return false;
                  }


   if( document.myForm.jmbg.value == "" ||
      isNaN( document.myForm.jmbg.value ) ||
            document.myForm.jmbg.value.length != 13 )
          {
              alert( "Molim Vas unesite ta&#269;an JMBG." );
    document.myForm.jmbg.focus() ;
    return false;
  }
  if( document.myForm.br_kartona.value == "" ||
      isNaN( document.myForm.br_kartona.value )  )
          {
              alert( "Molim Vas unesite broj kartona." );
    document.myForm.br_kartona.focus() ;
    return false;
  }
   if( document.myForm.sifra_lekara.value == "-1" )
    {
    alert( "Molim Vas izaberite &#353;ifru lekara!" );
    return false;
   }
   if( document.myForm.sif_osl_particip.value == "-1" )
    {
    alert( "Molim Vas izaberite &#353;ifru oslobadjanja particpacije!" );
    return false;
   }
   if( document.myForm.sif_drzave.value == "-1" )
    {
    alert( "Molim Vas izaberite &#353;ifru dr&#382;ave!" );
    return false;
   }
   return( true );
}
//-->
</script>
```

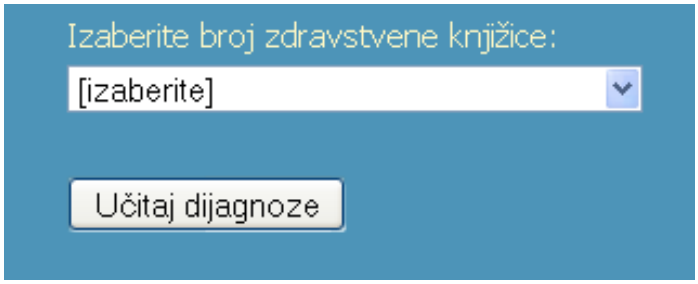
Слика 21. JavaScript кôд за проверу унетих података

Слично се додају сви ентитети базе. Кликом на линк Нова дијагноза/рецепт , отвара се страница за додавање нове дијагнозе или терапије.



Слика 22. Нова дијагноза/рецепт

Одабиром опције Нова дијагноза пацијента, отвара се страна где се бира пацијент из опадајућег менија, или се куца број здравствене књижице и тако изабере пацијент.

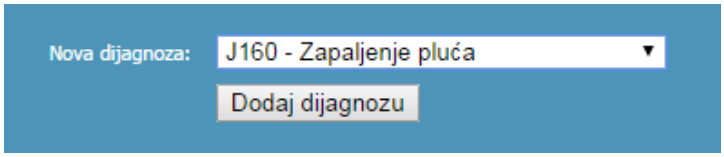


Слика 23. Нова дијагноза пацијента- избор пацијента

Слика 24. Нова дијагноза пацијента- изглед опадајућег менија

Слика 25. Страница са претходним дијагнозама и опцијом за додавање нове

У овом случају изабрани пацијент има пуно успостављених дијагноза, па се може одабрати нека од њих, или да се дода нова дијагноза.



Слика 26. Одабир нове дијагнозе

Слика 27. Додата дијагноза

За нову терапију пацијента, на линку Нова дијагноза/ рецепт, у подменију Нова терапија пацијента, такође треба изабрати број књижице, који је примарни кључ табеле пацијент и која једнозначно идентификује пацијента. Када се изабере пацијент по броју књижице долази се до странице за додавање терапије.

Слика 28. Додавање терапије

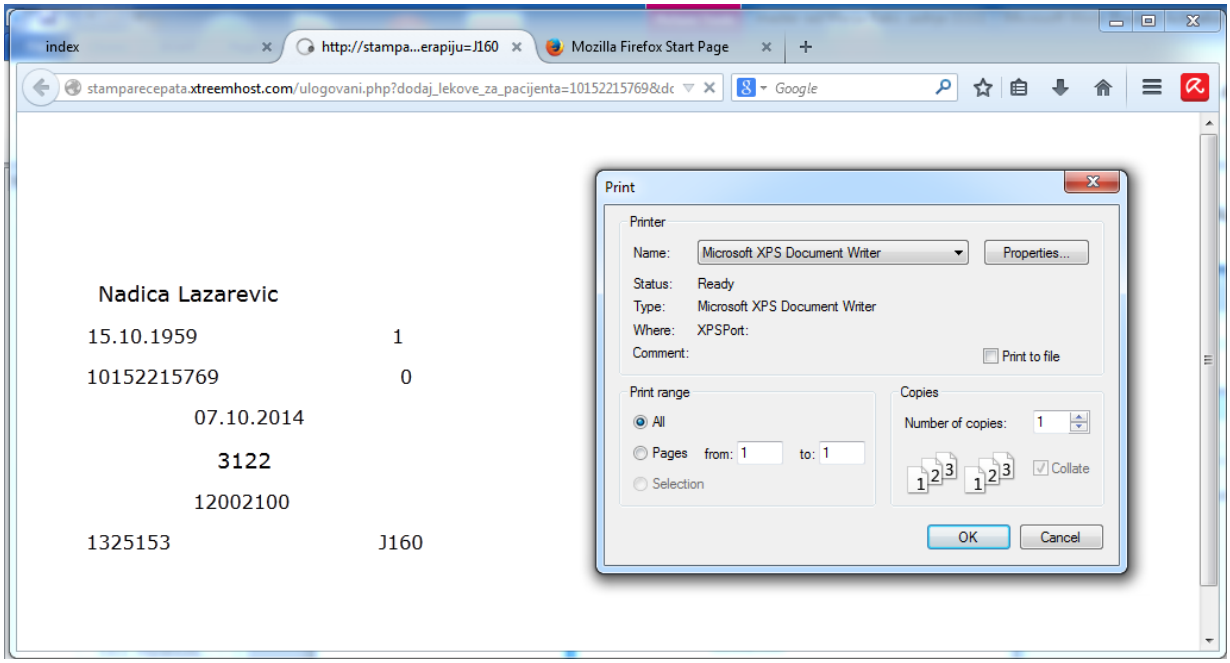
Одабиром дијагнозе за коју се додаје терапија долази се до странице на којој се додају лекови које пацијент треба да пије.

Слика 29. Додавање терапије

Слика 30. Страна са додатим терапијама

На овој страници су додате терапије за пацијента Марију Ракић са дијагнозом запаљење плућа, и опцијом да се штампа рецепт. До ове странице долази се и директно преко линка штампа рецепата и одабиром пацијента и дијагнозе.

Кликом на линк штампај, отвара се нови прозор са подацима за штампу и опцијом за потврду за слање података на штампач.



Слика 31. Штампа рецепата

Слика 32. Одштампан рецепт

7. Закључак

Тема овог мастер рада је била креирање информационог система, коме је крајњи циљ штампа рецепата директно из базе података. Резултат је постигнут и пројектовани информациони систем испуњава све захтеве наведене у спецификацији проблема. Систем је тако изграђен да омогући релативно лаку и једноставну надоградњу у циљу извршавања сложенијих операција и испуњавања могућих нових захтева.

PHP је најнапреднија и најкоришћенија скриптна технологија која се изводи на серверу. Врло је једноставан за коришћење и учење, јер користи функције других језика, па му је синтакса јасна. Коришћењем PHP-а могуће је направити апликације различитог подручја примене. Постоје свакако и недостаци овог скрипног језика. Они су углавном везани уз коришћење појединих функција. Како је PHP скриптни језик отвореног кôда многи програмери раде на његовом константном побољшавању и развоју. Приликом откривања сваког проблема, он се врло лако пријављује на службеној страници PHP-а (www.php.net/), буде исправљен, те на тај начин и сами корисници раде на његовом усавршавању.

Даље унапређење оваквог система може се тражити у заштити података, јер се подаци налазе на вебу и врло су подложни крађи.

8. Литература

[1] ПОПОВИЋ, Милан, Основе HTML-а, Београдска пословна школа, Интернет адреса: http://www.osnove-programiranja.com/prirucnici/osnove_html.htm, приступљено 03-03-2014.

[2] СТОЈЧЕВ, Миле, Увод у HTML, Електронски факултет Ниш, Интернет адреса: http://es.elfak.ni.ac.rs/rmif/Materijal/lab%20vezba%206.pdf , приступљено 05-03-2014.

[3] ВИЛИЈАМС, Хју, ЛЕЈН, Дејвид: Веб апликације и базе података: PHP и MySQL, Микро књига, 2003.

[4] ПАУНОВИЋ, Влатка, ТОМИЋ, Синиша, PHP приручник уз семинар, Хрватска удруга за отворене суставе и интернет, Загреб 2006, Интернет адреса: http://www.open.hr/wp-content/uploads/2012/04/PHP_prirucnik.pdf, приступљено 01-08-2014.

[5] Веб програмирање – Основе PHP језика, ПМФ, Крагујевац, Интернет адреса: https://imi.pmf.kg.ac.rs/component/docman/doc_view/445-wp-vezbe-2.html, приступљено 01-08-2014.

[6] PHP приручник – Листа кључних речи, Интернет адреса: http://php.net/manual/en/reserved.keywords.php, приступљено 01-09-2014.

[7] ПОПОВИЋ, Милан, Основе JavaScript-а, БПШ, Интернет адреса: http://www.osnove-programiranja.com/prirucnici/osnove_javascript.html, приступљено 03-08-2014.

[8] JavaScript, Интернет адреса: http://www.w3schools.com/js/js_popup.asp, приступљено 03-08-2014.

[9] ПАВЛОВИЋ ЛАЖЕТИЋ, Гордана, Увод у релационе базе података, Београд, 2006.

[10] ШВЕНДИМАН, Блејк, PHP4 Водич за програмере, Микро књига, Београд, 2005.

[11] ВИЛИНГ, Лук, ТОМСОН, Лаура, PHP и MySQL: развој апликација за веб, Микро књига, Београд, 2010.

[12] Фазе, методе и алати који се користе у развоју информационих система, Интернет адреса: http://www.sedna.rs/KakoRadimo , приступљено 03-10-2014.

[13] КОЛАЧ, Крешимир, Увод у развој вишеслојних пословних апликација у microsoft .NET околини, Међимурско велеучилиште у Чаковцу, 2012.