ВИСОКА ШКОЛА ЕЛЕКТРОТЕХНИКЕ И РАЧУНАРСТВА
СТРУКОВНИХ СТУДИЈА

Илић Небојша

ВЕБ апликација за евиденцију пројектних активности

- завршни рад -

Београд, јануар 2020.

Кандидат: **Илић Небојша**

Број индекса: **РТ-11/15**

Студијски програм: **Рачунарска техника**

Тема: **ВЕБ апликација за евиденцију пројектних активности**

Основни задаци:

1. Опис клијентске и серверске стране апликације за евиденцију пројектних активности

2. Имплементација ВЕБ апликације за евиденцију пројектних активности

3. Опис корисничког интерфејса

Ментор:

Београд, јануар 2020.године.

_____

др Перица Штрбац, проф. ВИШЕР

РЕЗИМЕ:

У раду је представљен развој веб апликације за евиденцију евиденцију пројектних активности. Freamework за PHP који је коришћен је специјално направљен за извођење овог задатка ради убрзавања и олакшавања процеса израде апликације и потенцијалног проширења. MySql база података преко PDO конекције због потенцијалног баратања са различитим базама и подацима. За израду интерфејса апликације претежно су коришћеми Bootstrap, jQuery, Ajax и JavaScript. Поред Bootstrap-а коришћен је класичан CSS.

Кључне речи: *Freamework, PHP, PDO, MySql, Bootstrap, jQuery, Ajax, JavaScript.*

ABSTRACT:

The paper presents the development of a web application for the record of project activities. Framework that was used for develop was specially made for developing this application in order to increasing speed of developing and simplify process and potential updates. MySql data base with PDO connection was used for the project in order to allow usage of multiply data bases and different data. For interface of the application was used Bootstrap, Ajax and JavaScript. Besides Bootstrap for design was used costume CSS.

Key words: *Freamework, PHP, PDO, MySql, Bootstrap, jQuery, Ajax, JavaScript*

САДРЖАЈ

1. УВОД

Апликација рађена као завршни рад има за циљ лакшу евиденцију активности унутар тима, одељења или било какве организације у којој је потребна нека врста организације и евиденције. Апликација пружа табелу са насловима активности и коментарима на исте као и секцију за креирање и брисање корисника по потреби.

Апликација је реализована помоћу следећих технологија:

- PHP
- JavaScript / Ajax / jQuery
- CSS
- HTML
- SQL
- Bootstrap

Организација кода је MVC (Model View Controller), о овој организацији ће бити више у подглављу технологије.

2. ТЕХНОЛОГИЈЕ

Технологије за креирање ове апликације су PHP, JavaScript / Ajax / jQuery, CSS, HTML, SQL, Bootstrap ове технологије као и организација кода MVC биће описани у овом подглављу.

2.1. PHP

PHP Hypertext Preprocessor је програмски језик оригинално дизајниран за веб дивелопмент. Оригинално је креиран од стране Расмус Лердорфа 1994 године. Због своје брзине, флексибилности и прагматичности, PHP покреће већину светских сајтова. Тренутно се користи 7.3 верзија која је избачена у децембру 2018. године. Ради израде ове апликације израђен је специјални framework а неки од постојећих за комерцијалну употребу су: Laravel, CodeIgniter, Symfony, CakePHP, Yii, Zend Framework, Phalcon, FuelPHP, PHPixie, Slim… Тренутно су најпопуларнији Laravel и Symphony. То што су најпопуларнији их не чини најбољим као и у свему као и овде сваки има своје предности и мане. На пример за израду light-weight апликације једна од најбоља комбинација је израда преко CodeIgniter-а и Api позиви преко Slim-а. Добар пример коришћења ове комбинације је веб шоп.

2.2. JAVASCRIPT / AJAX / JQUERY

JavaScript је једна од најбитнијих технологија World Wide Web-а. Омогућава интерактивне веб странице и самим тим је есенцијални део веб апликације. Већина сајтова као и већина веб претраживача садрже JavaScript engine како би га извршавали.

Ajax је метода која омогућава моменталне поромене делова странице без поновног учитавања стране “refresh”.

Док је jQuery библиотеке уграђена у JavaScript са већ уграђеним функцијама која олакшава програмеру куцање кода и скраћује процес изведбе.

Поред Ajax-а и jQuery-ја framework-ови за JavaScript су React, Vue, Angular, Ember и Backbone.js. Такође и овде је сулчај да ни један није савршен али ових пет су најкоришћенији.

2.3. HTML & CSS

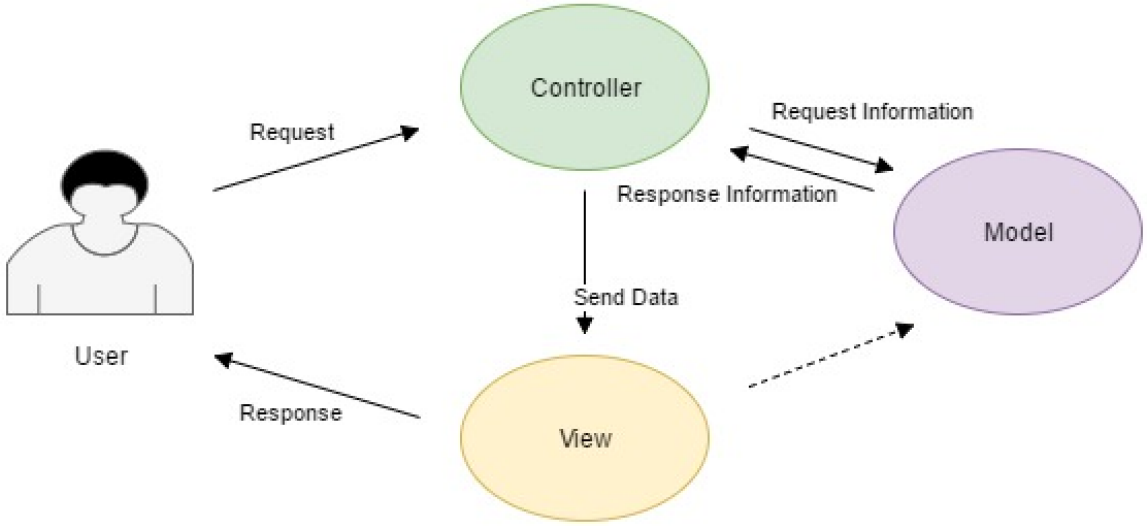
HTML (Hyper Text Markup Language) је описни језик специјално намењен за опис веб страница. Помоћу њега се једноставно могу одвојити елементи као што су наслови, параграфи, цитати и слично. Поред тога, у HTML стандард су уграђени елементи који детаљније описују сам документ као што су кратак опис документа, кључне речи, подаци о аутору и слично. Ови подаци су опште познати као мета подаци и јасно су одвојени од садржаја документа.

Док је CSS (Cascading Style Sheets) језик који је намењен да опише HTML или XML. Описује како елементи треба да се понашају на екрану, на папиру, у говору, или другим медијима. Поред CSS-а недавно је настао SCSS (SASS) ово је верзија CSS-а која подржава функције и неке функционалности које су се до тад извршавали само преко JavaScript-а. SASS је прва верзија која се после развила у SCSS који су у суштини исти осим неких делова у синтакси које су промењени. Коришћењем оваквог начина омогућава на пример дефинисање променљиве које се могу позивати на више места (нпр. боја позадине) и у случају да је потребно променити нешто уместо промене на више места довољно је променити једну линију кода на једном месту и та промена ће се учитати на свим местима где се та променљива позива. Ово је само један од примера могућности SCSS-а. На разне начине поред овог олакшава израду апликације и евентуалну промену у будућности.

2.4. MVC

MVC (Model View Controller) је архитектурални патерн који приказује апликацију у три главне логичке компоненте: model, view и controller. Свака од ових компонената је направљена да прикаже израду и поглед на апликацију. MVC је један од најчешће коришћених индустријских стандарда веб производње у виду framework-а за креирање пројеката који се могу са лакоћом повећавати и проширивати.

На (Слика 2.4.1) приказано је како то функционише. View је оно што корисник види кад приступи сајту. На било коју акцију (захтев корисника) View шаље захтев моделу који комуницира са контролером и обрађује податке које шаље назад у View и корисник добија резултат акције коју је извршио нпр захтев друге странице, login, logout, неке податке из базе података…



Слика 2.4 – Модел понашања MVC-а.

2.4.1 MVC КОМПОНЕНТЕ

Следеће су компоненте MVC-а

2.4.2 Model

Модел је компонента која одговара свом логиком која се односи на податке са које корисник користи. Ово може да представља или податке који пролазе између компоненти view-а и контролера или било којих других података који се односе на логику бизниса. На пример, податке о кориснику ће корисник објекат узети из базе података, манипулисаће са њима (одрадити жељену акцију или их спаковати у жељени формат), ажурирати их у бази података или их користити да направи нове податке.

2.4.3 View

View компонента се користи за сву UI (User Interface) логику у апликацији. На пример, корисник view ће укључити све UI компоненте као што су text box, dropdown list, или било које компоненте које су потребне које ће корисник користити или видети на тој апликацији.

2.4.4 Controller

Controllers ради као интерфејс између модел и view компоненти како би процесуирао сву пословну логику и долазеће захтеве, манипулисање са подацима користећи модел компоненте и интеракција са view како би сколопио све компоненте у финални приказ. На пример, корисник контролер ће баратати са свим интеракцијама и уносима које ће корисник правити преко корисник view-а и ажурирати базу података користећи корисник модел. Исто контролер ће се користити како би се користили и видели подаци о кориснику.

2.5 ASP.NET MVC

ASP.NET подржава три највећа модела за производњу: веб странице, веб форуме, и MVC (Model View Controller). ASP.NET MVC framework је лаган (тј. мало меморије захтева), веома стабилана презентација framework-а која интегрише већ постојеће ASP.NET додатке (features), као што су на пример мастер странице, аутентикације, сесије... Унутар .NET, овај framework је дефинисан у System.Web.Mvc assembly. Последња верзија MVC framework-а је 5.0. Користи се Visual Studio како би се креирао ASP.NET MVC апликације које могу да се додају као Visual Studio template, за будућу поновну употребу.

2.5.1 ASP.NET MVC додаци (Features)

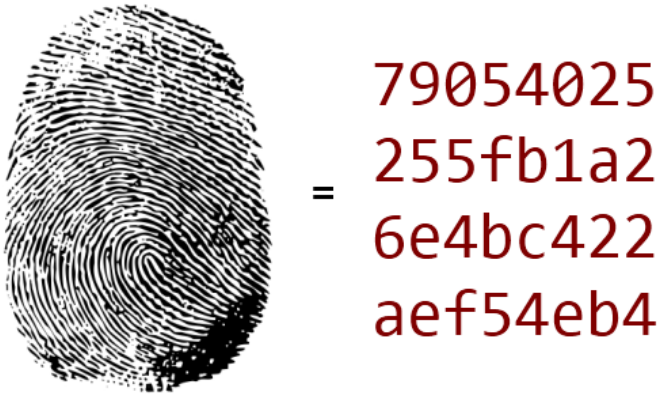
ASP.NET MVC пружа следеће додатке (features) −

- Идеална за израду комплексних али такође не толико захтевних апликација.
- Пружа framework који се може са лакоћом проширити као и додавати или избацивати као и преуређивати одређене. Као на пример, не желите да користите већ постојећи in-built Razor или ASPX View Engine, у том случају можете користити неки потуно независан view engine или чак могуће је преуредити већ постојећи како би добили оно што желите.
- Употребљава дизајн који је component-based (базиран на компонентама), самим тим дизајнира апликацију и дели је у три компоненте (модел, контролер и view). Ово омогућава људима који раде на апликацији да управљају комплексношћу на великим пројектима и рад на индивидуалним (одвојеним) компонентама.
- MVC структура унапређује тестни део производње апликације, с обзиром на то да све компоненте могу да буду дизајниране interface-based (базиране на интерфејсу) и тестиране коришћењем mock објектима. Стога ASP.NET MVC Framework је идеалан за пројекте које изводи велики тим са доста програмера.
- Подржава све огромне постојеће ASP.NET функционалности, као што су ауторизација и аутентикација, мастер странице, повезивање података (Data Binding), корисничке контроле, Memberships (постојећи налози), ASP.NET рутирање, и слично.
- Не користи концепт View State (који је присутан у ASP.NET-у). Ово помаже у производњи апликације, која је lightweight и даје потпуну контролу програмерима који изводе производњу апликације.

Стога, можете гледати на MVC Framework као на један од највећих и најкоришћенијих призвода са ASP.NET листе, која пружа велики сет додавања функционалности на структури која је базирана на компонентама у виду производње и тестирања.

2.6 HASH

Хеш је као отисак прста за податке.



Слика 2.6 – Пример 128-битног хеша.

Дати хеш једнозначно представља фајл, или вило коју колекцију података, бар у терорији. Ово је 128-битни MD5 хеш коју видите на слици (Слика 2.6). тако да највише може да представља 2^{128} јединствених комбинација, или 340 трилион трилиона, трилиона. У стварности простор који можемо користити је знатно мањи. Као што можете приметити почеће те да видите знатне колизије (сударања) када попуните половину квадратног корена простора, али квадратни корен од нестварно великог броја је и даље веома велики.

Разлика вредна помене између а checksum-а и хеша. Узмите за пример пуно име, Небојша Б. Илић. Име је пречица ка јединственошћу која је веома брза и лака, и лако се прави зато што сигурност није поента имена. Када прићете особи на улици, кафићу, ресторану, радном месту... не тражите му отисак да би сте проверили особу која вам се представила именом. Имена су прикладна разјашњења, не апсолутни доказ индентитета. У свету може бити људи са истим именима, не би било ни тешко да промените своје име да се подудара са нечијим, али да промените ваш отисак прста да се подудара са било чијим другим је готово немогуће сем у филмовима.

2.6.1 Сигурни хешеви су дизајнирани да буду спори

Брзина checksum калкулације је битна, с обзиром на то да checksum ради са подацима који се тренутно шаљу. Ако checksum-у потребно много времена то може утицати на брзину пребацивања података. Ако checksum изазиве значајно CPU преоптерећење, то такође значи да ће се трансфер података успорити или преоптеретити ваш рачунар. На пример замислите врсту checksum-ова који су коришћени на видео стандардима попут DisplayPort-а, који могу достићи висине од 17.28 Gbit/sec.

Али хешови нису дизајнирани за брзину, баш супротно, хешеви када се користе за сигурност требају да буду спори. Што је већа брзина прорачуна хеша, то је већа могућност напада чистом силом (brute force) како би се извршили напади. Нажалост споро у 1990 и 2000 години нису исти. С обзиром на напредак технологије генералне брзине су се такође повећале. Стога оно што је било споро 1990 је недовољно споро за 2000. годину а поготово за данашње стандарде 20 година након тога. Иако су дизајнери хеш алгоритама можда предвидели унапређење у процесорској снази путем Муровог закона, нису могли да предпоставе толико велико повећање у GPU снази. Колико велика разлика? Погледајте поређења CPU који омогућавају hascat са GPU који помоћу oclHashcat израчунаван MD5 хешом:

Radeo 8
n 7970 213.6 M c/s

6-core 5

AMD CPU 2.9 M c/s

GPU на једној модерној графичкој картици производи преко 150 пута већи број хешовања по секунди у поређењу са процесором. Ако Муров закон очекује дуплирање процесорске снаге сваких 18 месеци то је као да провиримо 10 година у будућност.

2.6.2 Хешови и шифре

Када говоримо о шифрама, пошто су хешовање и шифре директно повезане, осим ако складиштимо шифре неправилно. Увек ће се чувати корисникова шифра као salted hash (о овоме ће бити речи у даљем тексту) никад као обичан текст. Ово значи да и ако база података садржи све те хешове и компромитована је или су подаци исцурели кориснки је и даље заштићен, нико не може да провали њихову шифру на основу хешова сачуваних у бази података. Постоји dictionary attack метода која може бити невероватно ефикасна, али не можемо заштитити кориснике који као своју шифру имају на пример „password1“. Право решење за овакве кориснике није да им буде наметнуто да стављају дугачке и компликоване шифре, већ тај посао обавља хеш и salt део.

Ово је има једну несрећну последицу за хешовање шифара. Мали број њих је дизајниран са толико великим и доступним GPU ресорсима на уму. Ово су неки од резултата рачунара, који има две **ATI Radeon 7970** графичке картице које стварају скоро 16000 M c/s са MD5. У истраживању је коришћен oclHashcat-lite са пуним опсегом америчке тастатуре укључујући велика слова, мала слова, бројеве, и све симболе:

all	6	character	47 seconds
password MD5s			
all	7	character	1 hour, 14 minutes
password MD5s			
all	8	character	~465 days
password MD5s			
all	9	character	*fugged aboudit*
password MD5s			

Процес се скалира скоро савршено када додате графичку картицу, тако да можете да смањите време за дупло, тако што ставите четири графичке картице у једну машину. Ентузијасти су ово радили још од 2008. године. Могли би сте да смањите време опет за дупло тако што додате још једну радну станицу са још четире графичке картице које деле обим напада. Са овом конфигурацијом смо дошли до полу разумних 117 дана да створимо свих 8 MD5 карактера. Са обзиром да већина шифара нема ни један специјални карактер ово је можда превелики опсег који гађамо. Шта би било када би урадили исту ствар користећи само велика слова, мала слова и бројеве.

all password	6 MD5s	character	3 seconds
all password	7 MD5s	character	4 minutes
all password	8 MD5s	character	4 hours
all password	9 MD5s	character	10 days
all password	10 MD5s	character	~62 5 days
all password	11 MD5s	character	*fug gedaboudit*

Ако сте заинтересовани за најгори могући сценарио, шифра од 12 карактера која садржи само мала слова биће доступна за од прилике 75 дана на оваквом рачунару.

2.6.3 Када складиштите шифре користите salt како би сте били сигурни од свих случајева

У случају напада можете да покушате да сакријете salt, можете га ставити негде другде као на пример у другу базу података, конфигурациони фајл, или у неки наиме сигуран хадрвер са додатим слојевима заштите. У случају да нападач дође у посед ваше

базе података са хешовима шифара, али у том случају не мора да има знање где је и да ли постоји salt.

Ово ће пружити илузију сигурности која је заправо ефикаснија од праве заштите, зато што је потребан salt и избор хеш алгоритама да се генерише и провери хеш. Мало је вероватно да је нападач у поседу једног али не и другог. Ако сте били компромитовани до те мере да нападач има вашу базу шифара, логично је предпоставити да имају или могу пронаћи ваш тајни, сакривени salt.

Прво правило сигурности је да увек предпоставите најгоре. Да ли је добро да имате насумични salt за сваког корисника понаособ, тако да хипотетички два корисника могу да имају исту шивру која ће бити складиштена у бази на различите начине. Данас сам salt више не може да спаси од особе која је спремна да потроши велику количину новца за графичку картицу.

3. Опис клијентске и серверске стране апликације за евиденцију пројектних активности

Клијентска страна се састоји од следећих страница, home (почетна страна), help, login, dashboard, users, create user.

Home страница је почетна (прва која се приказује уласком на сајт) она садржи податке о креатору овог рада као и назив пројекта.

Help страница предвиђена за помоћ око навођења кроз user interface на сајту која у овом случају садржи комплетну документацију приказану преко pdf-a.

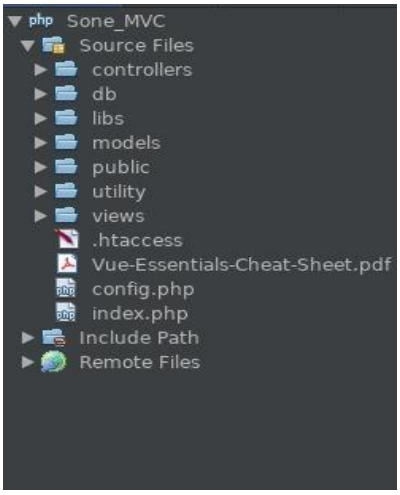
Login страница садржи форму за логовање и тиме омогућава постојећим корисницима приступ на остале странице у зависности од улоге (права) налога.

Постојеће улоге су: owner, admin и default. Default има права за приступ dashboard страници на којој му је омогућено да дода нови коментар са произвољним насловом и да едитује своје коментаре. Admin има поред ових права да едитује било који коментар као и да га обрише, поред тога има приступ и осталим страницама а то су users и create user. Он је у могућности да направи новог корисника и има евиденцију у остале кориснике. Док owner има поред свих ових права могућност да брише кориснике, да им промени права из default у admin и обрнуто и могућност да едитује постојеће кориснике тј. да промени име или шифру по потреби. Owner не може да се направи преко сајта, због сигурности та улога је hard-code-ована у базу како би се отежао напад на страницу тако што би нападач променио свом налогу улогу у највишу могућу.

За све странице су коришћени исти header и footer стим што се header разликује у случајевима да ли је корисник улогован или не као и од права које његов налог има.

4. ИМПЛЕМЕНТАЦИЈА ВЕБ АПЛИКАЦИЈЕ ЗА ЕВИДЕНЦИЈУ ПРОЈЕКТНИХ АКТИВНОСТИ

Као што је већ поменуто апликација је урађена преко MVC архитектуре. Организацију фајлова у пројекту можете видети на (Слика 4.1). Као што може да се примети поред фолдера са ове три компоненте постоје и други фолдери, у којима су фајлови који употпуњују фникционалност и додатно смањују потребу за поновним куцањем кода и олакшавају имплементацију.



Слика 4.1 – Архитектура фајлова унутар пројекта.

Фолдер db садржи копију базе за поновно подизање у случају промене сревера или губитка података услед хакерског напада на сајт. Back-up се не ради аутоматски, за

то је задужен власник или друго лице коме је задата та улога као и приступ базама које су live.

Фолдер libs као што само име каже библиотеке садржи као и такав фолдер код било ког другог framework-а овоф типа. А то су фајлови који обухватају неку целину функционалности, у овом случају су то на пример модел и контролер. Чија улога је да за одговарајучу страницу пронађу одговарајуће компоненте и споје их у целину како би се добила одговарајућа страница. Овај метод омогућава да ови фајлови само траже руте о обрађују их уместо да се цео тај процес ради појединачно за сваку страницу. Самим тим добијамо смањење количине кода и лакшу рефакторизацију јер је у случају жељене промене потребно променити код у само овим фајловима уместо у фајловима за сваку страницу. Начин и код на који је ово изведено можете видети испод на примеру контролера и модела (Слика 4.2 и Слика 4.3).

```
class Controller
{
    function __construct()
    {
        $this->view = new View();
    }
    public function load_model($name, $model_path='models/')
    {
        $path = $model_path.$name.'_model.php';
        if (file_exists($path)) {
            require $model_path.$name.'_model.php';
            $model_name  = $name.'_Model';
            $this->model = new $model_name();
        }
    }
}
```

Слика 4.2 – Класа библиотеке за контролер.

```
class Model
{

    function __construct()
    {
        $this->db = new My_database();
    }
}
```

Слика 4.3 – Класа библиотеке за модел.

Поред ових фајлова можемо видети и фајл View који обавља исту улогу као и предходна 2 само што је његова улога да обради view тј. део странице који је најближи кориснику, пронађе руте и рендерује целину. Поред класичних фајлова који представљају садржај странице постоје универзални header и footer који су учитани у сваку страницу. Овај приступ такође смањује количину кода као и могућност лакше измене тај пример можете видети у header фајлу. Header садржи навигациони бар који у зависности од тога да ли је корисник улогован или не и у зависности од права налога приказује различите елементе у навигационој листи (Слика 4.4).

```
<nav class="navbar navbar-inverse">
        <div class="container-fluid">
          <div class="navbar-header">
            <a class="navbar-brand" href="<?php echo URL; ?>index">MVC</a>
          </div>
          <div id="header">
            <ul class="nav navbar-nav">
              <?php
              if (My_session::get('logged_in') == false):;
                ?>
                <li><a href="<?php echo URL; ?>index">Home</a></li>
                <li><a href="<?php echo URL; ?>help">Help</a></li>
              <?php endif; ?>
              <?php
              if (My_session::get('logged_in') == true):;
                ?>
                <li><a href="<?php echo URL; ?>dashboard">Dashboard</a></li>
                <?php
                if (My_session::get('role') == 'owner' || My_session::get('role')
                  == 'admin'):;
                  ?>
                  <li><a href="<?php echo URL; ?>user">Users</a></li>
                <?php endif; ?>
                <li><a href="<?php echo URL; ?>dashboard/logout"><span class="glyphicon glyphicon-log-
out"></span> Logout</a></li>
              <?php else: ?>
                <li><a href="<?php echo URL; ?>login"><span class="glyphicon glyphicon-log-
in"></span> Login</a></li>
<?php endif; ?>
            </ul>
          </div>
        </div>
      </nav>
```

Слика 4.4 – Header са варијацијама у односу на налог.

Ту су такође и фајлови: database, hash и session.

Database служи за комуникацију са базом, што подразумева конекцију са њом као и функције за интеракцију са њом. Конекција је извршена на следећи начин (Слика 4.5). Овај фајл наслеђује PDO о коме ће бити речи даље у тексту. Уз помоћ делова који

су дефиниссани у том фајлу овде је довољно да у конструктору проследимо податке о бази који су потребни за успостављање везе.

```
public function __construct() {
        $dns = 'mysql:host=localhost;dbname=MVC;charset=utf8';
        $uname = 'root';
        $pass = 'Nebojsa.Ilic1996';                    //ne znam da li je prazan ili root
        parent::__construct($dns, $uname, $pass);
    }
```

Слика 4.5 – Параметри за повезивање са базом.

У овом фајлу су такође дефинисане функције које стандардизују и олакшавају све акције које могу да се врше над базом (select, insert, update, delete)(Слика 4.6). Овим је постигнуто далеко лакше извођење ових акција јер је за било коју од наведених довољна само једна линија кода која дефинише тачно податке који су жељени на том месту. Поред ових основних функција постоје и додатне за селекцију сви података тог типа као и за селектовање само једног реда у табели (један коментар или један корисник) што олакшава имплементацију функција за edit и delete.

```
public function select($sql, $array = array(), $fetch_mode = PDO::FETCH_ASSOC) {
        $sth = $this->prepare($sql);
        foreach ($array as $key => $value) {
            $sth->bindValue("$key", $value);
        }
        $sth->execute();
        return $sth->fetchAll($fetch_mode);
    }

    public function insert($table, $data) {
        ksort($data);  //sort by key
        $field_names = implode('`,`', array_keys($data));
        $field_values = ':' . implode(', :', array_keys($data));
        $sth = $this->prepare("INSERT INTO $table (`$field_names`) VALUES ($field_values)");
        foreach ($data as $key => $value) {
            $sth->bindValue(":$key", $value);
        }
        $sth->execute();
    }

            public function update($table, $data, $where) {
                        ksort($data);  //sort by key
                        $field_details = null;
            foreach ($data as $key => $value) {
                        $field_details .= "`$key`=:$key,";
            }
            $field_details = rtrim($field_details, ',');
            $sth = $this->prepare("UPDATE $table SET $field_details WHERE $where");
            foreach ($data as $key => $value) {
                        $sth->bindValue(":$key", $value);
            }
            $sth->execute();
            }

    public function delete($table, $where, $limit = 1) {
        return $this->exec("DELETE FROM $table WHERE $where LIMIT $limit");
```

```
  }
```

Слика 4.6 – Функције за комуникацијом са базом.

Session је задужен за управљање сесијама односно од тренутка кад се корисник улогова до његовог logout-a. Поред овога подаци из кључева сесија се могу користити и у друге сврхе као на пример провера типа налога и самим тим провера права које тај налог садржи. Или порука за добродошлицу коју можете видети на dashboard stranici nakon uloga.

Hash (Слика 4.7) скужи за сигурност. Његова улога је контролосање лозинки. Лоша пракса је складиштити лозинке у базу у формату у ком су унете јер у том случају свако ко има приступ бази има потпун приступ сваком од налога. Због тога постоји hash чија је улога да коришћењем неког алгоритма измени садржај и на тај начин га убаци у базу, наравно поред овога мора да буде у стању да тај унос врати у првобитно стању када корисник покуша да се улогује и унесе лозинку у првобитном стању. Следећи проблем настаје због тога што су сви лагоритми за ове потребе познати и јавни тако да већина алгоритама може веома лако да се пробије па чак и преко разних веб апликација које обављају тај посао. Због тога након хешовања добра је пракса да се том генерисаном низу карактера дода део (salt) који је такође један такав низ и после тога уради још један hash пожељно другим алгоритмом. Salt део је дефинисан у другом фајлу о коме ће бити реч даље у тексту. Ево и један пример како изгледа лозинка када је корисник уноси и како изгледа у бази после овеих акција.

```
class Hash
{

    /**
     * @param string $algo The algorithm (md5, sha1, whirlpool, etc)
     * @param string $data The data to encode
     * @param string $salt (This should be the same throughout the system probably)
     * @return string The hash/salted data
     */
    public static function create($algo, $data, $salt)
    {
        $context = hash_init($algo, HASH_HMAC, $salt);
        hash_update($context, $data);

        return hash_final($context);
    }
}
```

Слика 4.7 –Hash класа.

И последњи али и не мање битан је bootstrap фајл. Далеко од тога да је није битан јер овај фајл седржи неке од најбитнијих ствари. Овде се врши коначно рендеровање и спајање у једну компоненту. Поред овога задужен је за рутирање као и за генерисање URL-а. Генерисање URL-а је битно за почетак да би с избациле празнине или непожељни карактери. Такође промена имена директоријума тако рећи скривена навигација како не би била присутна дирактна имена фајлова који се користе на тој

страници што знатно отежава нападачу да са лакоћом види начин на који сајт ради и то искористи за напад. Поред ових естетских и сигурносних делова обававља редирекцију као на пример ако корисник укуца име странице која не постоји вратиће га на почетну страницу(Слика 4.8). Или ако покуша да приступи страници или страницама за која нема права то јесте users и dashboard без налога или са налогом који нема право за users. Ова библиотека ће га послати на error страницу (Слика 4.9). Делове кода можете видети испод (Слика 4.10).

```
private function get_URL()
    {
        $this->url = isset($_GET['url']) ? $_GET['url'] : null;   //default index
        $this->url = str_replace('-', '', $this->url);
        $this->url = filter_var($this->url, FILTER_SANITIZE_URL);
        $this->url = rtrim($this->url, '/');   //izbacuje '/' sa kraja url-a
        $this->url = explode('/', $this->url);
    }

    public function set_controller_path($path)
    {
        $this->controller_path = trim($path, '/').'/';
    }

    public function set_model_path($path)
    {
        $this->model_path = trim($path, '/').'/';
    }

    public function set_error_file($path)
    {
        $this->error_file = trim($path, '/');
    }

    public function set_default_file($path)
    {
        $this->default_file = trim($path, '/');
    }

    private function load_default_controller()
    {

        require $this->controller_path.$this->default_file;
        $this->controller = new index();
        $this->controller->index();
    }

    private function load_existing_controller()
    {
        $file = $this->controller_path.$this->url[0].'.php';
        if (file_exists($file)) {
            require $file;
            $this->controller = new $this->url[0];
            $this->controller->load_model($this->url[0]);
        } else {
            $this->F_error();
        }
    }
```

Слика 4.8 – Сређивање и генерисање URL-а.

```
public function F_error()
    {
        //require 'controllers/error.php';
        require $this->controller_path.$this->error_file;

        $this->controller = new My_error();
        $this->controller->index();
        exit;
    }
```

Слика 4.9 – Функција за потребу за пребацивање на error страну .

```
/*** If a method is passed in the GET url paremter
     *  http://localhost/controller/method/(param)/(param)/(param)
     *  url[0] = Controller
     *  url[1] = Method
     *  url[2] = Param
     *  url[3] = Param
     *  url[4] = Param*/
    public function call_controller_method()
    {
        $length = count($this->url);
        if ($length > 1) {  // Make sure the method calling exists
            if (!method_exists($this->controller, $this->url[1])) {
                $this->F_error();
            }
        }
        // Determine what to load
        switch ($length) {
            case 5:
                //Controller->Method(Param1, Param2, Param3)
                $this->controller->{$this->url[1]}($this->url[2], $this->url[3],
                    $this->url[4]);
                break;

            case 4:
                //Controller->Method(Param1, Param2)
                $this->controller->{$this->url[1]}($this->url[2], $this->url[3]);
                break;

            case 3:
                //Controller->Method(Param1)
                $this->controller->{$this->url[1]}($this->url[2]);
                break;

            case 2:
                //Controller->Method()
                $this->controller->{$this->url[1]}();
                break;
            default:
                $this->controller->index();
```

```
            break;
        }
    }
```

Слика 4.10 – Генерисање URL-а и управљање учитавања.

U фолдеру public се налазе CSS и JavaScript фајлови. Сав дизајн сајта који није урађен преко bootstrap-а и слично је одрађен овде. Дизајн је одвојен на овај начин због лакше претраге и промене у случају промене дизајна. У фолдеру public постоји још једна подела по фолдерима како би било још прегледније а то су фолдери за дизајн и логику.

Што се тиче дизајна већина је одрађена преко bootstrap-а у овом фолдеру су минорне измене како би дизајн довели на одговарајући ниво.

Логика је одрађена преко Ajax-а и JQuery-ја за разлику од дизајна већина логике је ручно направљена за ове потребе. А то су на пример провере да ли су подаци добро унети приликом прављења налога или додавања коментара. Поред тога можете приметити анимацију приликом брисања података и pop-up window приликом измене постојећег коментара. Међутим приликом измене корисника уместо pop-up window-а постоји нова страница. Разлог за то је како би преко овог задатка (ове веб апликације) показао различите начине да се одради једна те иста ствар. Примере кода из овог фајла можете видети испод за брисање података (Слика 4.11), за pop-up, (Слика 4.12), за валидацију (Слика 4.13).

```
// Delete data
    $('.delete').click(function () {

        var el = this;
        var id = this.id;
        var splitid = id.split("_");
        // Delete id
        var deleteid = splitid[1];
        var hostString = window.location.protocol + "//" + window.location.host + "/";
        $.ajax({
            url: hostString + '/dashboard/deleteAjax',
            type: 'GET',
            data: {id: deleteid},
            success: function (response) {
                if (response == 1) {
                    // Remove row from HTML Table
                    $(el).closest('tr').css('background', 'tomato');
                    $(el).closest('tr').fadeOut(800, function () {
                        $(this).remove();
                    });
                } else {
                    alert('Invalid ID.');
                }
            }
        });
    });
```

Слика 4.11 – Ајах функција за брисање података из базе.

```
// Edit data, open modal pop-up
  $(".edit").click(function() {
    var id = $(this).attr("id");
    var hostString =
      window.location.protocol + "//" + window.location.host + "/";
    $.ajax({
      url: hostString + "/dashboard/selectUser",
      method: "POST",
      data: { id: id },
      dataType: "json",
      success: function(data) {
        $("#title").val(data.title);
        $("#text").val(data.text);
        $("#id_edit").val(data.id);
        window.$("#myModal").modal("show");
      }
    });
  });
```

Слика 4.12 – Ајах функција за edit-овање података из базе и pop-up.

```
//Validacija na front strani
  $("#submit_user").click(function(e) {
    // using serialize function of jQuery to get all values of form
    var serializedData = $("#user_create_form").serialize();
    var hostString =
      window.location.protocol + "//" + window.location.host + "/";
    // Variable to hold request
    var request;
    // Fire off the request to process_registration_form.php
    request = $.ajax({
      url: hostString + "user/createUser",
      type: "POST",
      data: serializedData
    });

    // Callback handler that will be called on success
    request.done(function(jqXHR, textStatus, response) {
      // you will get response from your php page (what you echo or print)
      // show successfully for submit message
      $("#result")
        .html(response)
        .css("background", "tomato");
    });

    // Callback handler that will be called on failure
    request.fail(function(jqXHR, textStatus, errorThrown) {
      // Log the error to the console
      // show error
      $("#result")
        .html("There is some error while submit")
        .css("background", "tomato");
      console.error("The following error occurred: " + textStatus, errorThrown);
```

```
    });

    e.preventDefault();
  });
```

Слика 4.13 – Ајах функција за валидацију и контролисање порука упозорења.

Фолдер utility садржи само један фајл за аутентификацију који је поменут код сесија чија је улога да прати да ли сесија постоји или не и у случају да је сесија истекла врати корисника на login страну јер би у супротом на покушај било које акције био пребачен на error стану (Слика 4.14). Овај фолдер као што и само име каже може да садржи бико који вид портабилног (корисног) кода који надгледа целу апликацију или део ње који такође може бити искоришћен како би допунио неку функцију која ће обављати задатак у целини.

```
public static function handleLogin()
    {
        session_start();
        $logged = (array_key_exists('logged_in', $_SESSION)) ? $_SESSION['logged_in'] : null;

        if (!$logged) {
            session_destroy();
            header('Location: ' . URL. 'login');
            exit;
        }
    }
```

Слика 4.14 –Utility функција за контролу сесије.

Поред ових фолдера ту су ифајлови .htaccess, config, index.

Index је главна страница на којој су укључене све компоненте које су потребне како би се било која страница формирала (Слика 4.15). А коментару овог дела кода каже да је боља пракса користити autoloader који би аутоматизовао овај процес и у случају додавања нових фајлова у било ком од већ дефинисаних фолдера аутоматски би био пронађен и додат у ову рутину без потребе да се на овом месту ручно додаје.

```
//bolja prakse je da se koristi autoloader ali posto ovaj projekat
//nije toliko opsiran fajlovi su inkludovani na ovaj nacin
require 'config.php';
require 'utility/Auth.php';
//Za prikazivanje erora
ini_set('display_errors', 1);
require 'libs/Bootstrap.php';
require 'libs/Controller.php';
require 'libs/View.php';
require 'libs/Model.php';
require 'libs/Database.php';
require 'libs/Session.php';
require 'libs/Hash.php';
```

```
$bootstrap = new Bootstrap();
$bootstrap ->init();
```

Слика 4.15 – Inedex у коме су учитане компоненте.

Config као што и само име каже садржи конфигурације (Слика 4.16) као што су руте до root-a (корена) апликације, поред њега видимо и руту до фолдера libs. За потребе ове апликације ово је било довољно ако наравно могуће је на овај начин дефинисати све путање што би олакшалп даље рутирање. Поред тога имамо конструктор у коме су прослеђени подаци за повезивање са базом података. И константе у виду кључева за хешовање лозинки као и генерални кључ који тренутно нема улогу али је дефинисан у случају да је потребно хешовати још неки податак. Ово је изведено на овај начин зато што кључеви морају остати исти јер у случају промене систем не би био у могућности да пронађе ни један податак који одговара унетом. Ови кључеви се користе у salt делу о коме је било речи у делу како функционише хешовање лозинки и зашто је то добра пракса.

```
//Paths
define('URL', 'http://sone-mvc.local/');    // menja se u slucaju promene servera
define('LIBS', 'libs/');

//Database
class Database extends PDO
{

    function __construct()
    {
        parent::__construct('mysql:host=localhost;dbnameSone_MVC', 'root',
            'Nebojsa.Ilic1996');
    }
}
//Constants
//For other
define('HASH_GENERAL_KEY', 'E7r9t8@Q#h%Hy+M');
//For password
define('HASH_PASSWORD_KEY', '2Beers2$50Cents');
```

Слика 4.16 – Config fajl.

И .htaccess је конфигурацион фајл за потребе веб сервиса који покрећу софтвер Apache Web Server-a. Могу се користити за измену конфигурације у софтверу Apache Web Server-a како би се укључиле или искључиле додатне функције и додаци које Apache Web Server може да понуди (Слика 4.17).

```
        php_flag display_errors on

        php_value error_reporting 9999

            Options -MultiViews

        RewriteEngine On

        RewriteCond %{REQUEST_FILENAME} !-d
```

```
RewriteCond %{REQUEST_FILENAME} !-f

RewriteCond %{REQUEST_FILENAME} !-l

RewriteRule ^(.+)$ index.php?url=$1 [QSA,L]
```

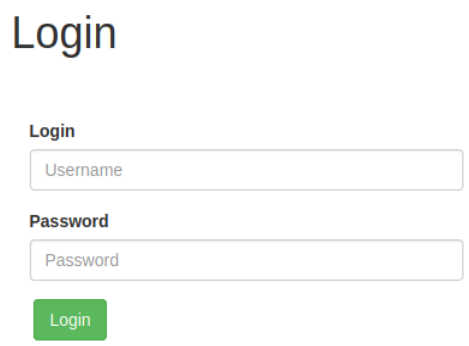
Слика 4.17 – .htacces пример из овог пројекта.

5. Опис корисничког интерфејса

При приступу веб апликацији корисник се среће са почетном страницом (home), на којој се налазе подаци о креатору и пројекту (Слика 5.1).

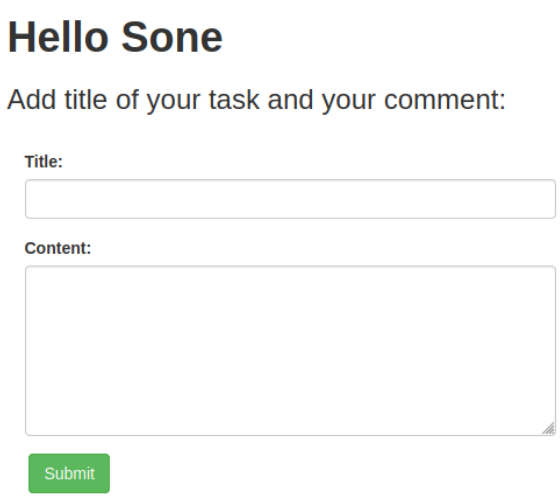
Слика 5.1 – Home страна.

Поред ове странице корисник има приступ help страници која садржи потпуну документацију у PDF формату. Док Login Страница садржи форму за логовање за постојеће кориснике. Корисник није у стању да самостално направи налог, како би дошао до налога потребно је да ту уради неко од админа или власник. Испод овог текста можете видети слику са изгледом login странице (Слика 5.2). Help страница ће бити финализирана тек са завршетком ове документације која ће бити убачена.

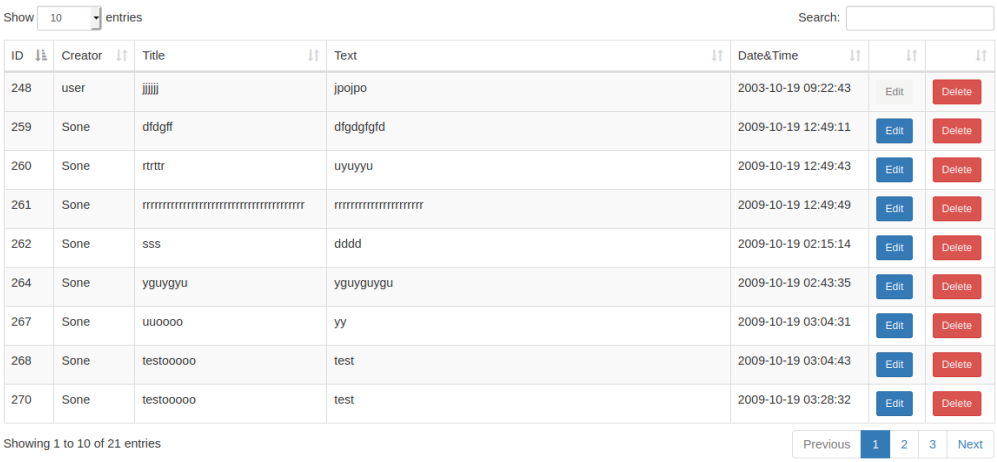


Слика 5.2 – Форма за login.

Након успешног логовања као што можете видети (Слика 5.3.1 и 5.3.2) приказана је страница dashboard на којој корисници имају евиденцију у табелу са коментарима и насловима активности које је та особа лично или било која друга особа која је укључена у тај пројекат или било какву организацију за коју је потребна евиденција додала. Такође има опцију да сортира табелу по било којим од датих параметара као и опцију search у којој може укуцати слово, слог или кључну реч по којој жели да претражи табелу. Наравно поред увида у већ постојеће податке може додати нове, уређивати своје старе уносе као и опцију брисања истих.

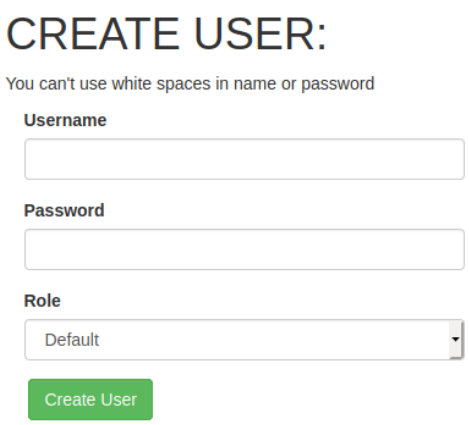


Слика 5.3.1 – Форма за додавање података.

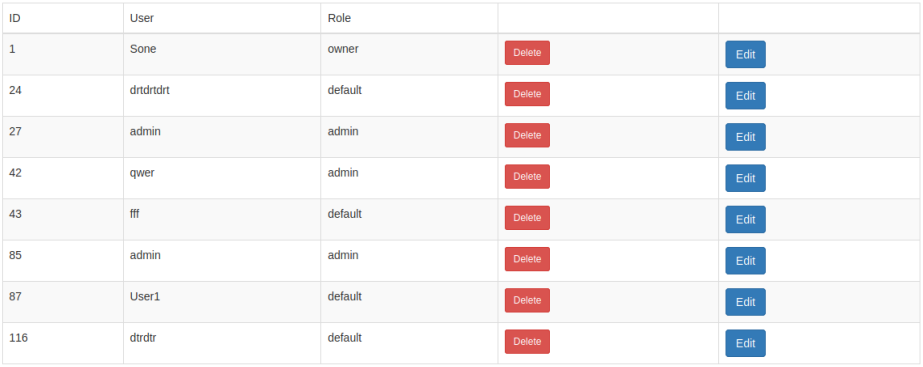


Слика 5.3.2 – табела са подацима.

У зависности од типа налога админ или власник имају приступ страници Users на којој имају опцију за креирање новог налога као и табелу са већ постојећим налозима. Постојећи налози могу бити преуређени по потреби или промењена шифра у колико је корисник захтевао промену или је заборавио стару. Поред ове опције постоји опција за брисање налога. (Слика 5.4.1 И Слика 5.4.2).



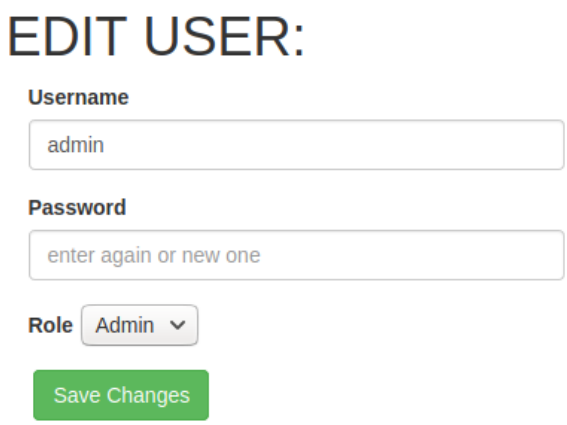
Слика 5.4.1 – Форма за креирање корисничких налога.



ID	User	Role		
1	Sone	owner	Delete	Edit
24	drtdrtdrt	default	Delete	Edit
27	admin	admin	Delete	Edit
42	qwer	admin	Delete	Edit
43	fff	default	Delete	Edit
85	admin	admin	Delete	Edit
87	User1	default	Delete	Edit
116	dtrdtr	default	Delete	Edit

Слика 5.4.2 – Табела корисничких налога.

За уређивање налога постоји још једна страна (Слика 5.5) по приступу страници поље за корисничко име је већ испуњено старим именом са могућношћу да се промени и поље за шифру као и drop-down мени за промену улоге по потреби.



Слика 5.5 – Страна за edit корисничких налога.

6. ЗАКЉУЧАК

Сваки framework као и овај значајно убрзавају процес израде апликације, смањују поновно куцање и олакшавају имплементацију истог кода на друге пројекте. Захваљујући библиотекама и класама комплетна логика и функционалност могу да се имплементирају на различитим местима уз само једну линију кода. Поред тога, у овом случају се користи и за конфигурацију, комуникацију и повезивање са базом података као и опцију за писање ручних упита (Api), који даље поједностављују писање кода.

Овом логиком и наведеним технологијама је у задатку израђена апликација која преко веома једноставног user interface-a омогућава корисницима једноставну организацију и евиденцију активности унутар неког тима. Без обзира на подручје рада. Поред тога корисници са већим правима имају велики избор уређивања, додавања, или брисања коментара као и самих корисника.

Свака од технологија је омогућила:

- PHP је омогућио комуникацију са сервером и базом података, сесије и провере прикликом било каквог уношења података.
- JavaScript је коришћен преко библиотека као што су Ajax и jQuery.
- Захваљујући Ajax-u након уноса података није потребно освежавање (refresh) странице како би се виделе промене које су постигнуте притиском на било које дугме.
- jQuery као и Ajax побољшава кориснички интерфејс самим тим чинећи га пријатнијим за корисника. На пример анимација и промена боје након брисања податкаиз табеле као и поруке о грешкама или успешним акцијама.
- MVC архитектура је поједносавила цео процес и омогућила лако унапређење апликације на било који начин.
- Hash-ом је обезбеђен интегритет корисничких налога у случају било каквог напада или компромитовања базе, захваљујући целокупном процесу који лозинка прође приликом креирања налога или логовања.

Апликација таква каква јесте испуњава све основе потребе било ког тима што се тиче евиденције активности и организације, с тим што захваљујући својој архитектури и извођењу приликом израде може лако да се унапреди на било који начин. Ово је урађено због различитих потреба тимова у зависности од активности и надлежности тимова, тако да основна верзија се може користити у овом облику или унапредити на начин који највише одговара потребама клијента.

7. ИНДЕКС ПОЈМОВА

8. ЛИТЕРАТУРА

[1] David Sklar, Adam Trachenberg, „*PHP Cookbook“*, O'Reilli, 2002

Преузето са: http://webalgarve.com/books/MySQL%20&%20PHP/PHP%20Cookbook,%203rd%20Edition.pdf

[2] Kyle Simpson, „*You Dont Know JS Yet“* (book series)

Преузето са: https://github.com/getify/You-Dont-Know-JS

[3] Tutorials Point – MVC Framework Introduction: https://www.tutorialspoint.com/mvc_framework/mvc_framework_introduction.htm

[4] Speed Hashing: https://blog.codinghorror.com/speed-hashing/

[5] Youtube, JREAM (channel), „Create Your Own MVC Series“: https://www.youtube.com/channel/UCFfuK45zBZxhq0m1bxYP-Zw

[6] Stack Overflow: https://stackoverflow.com/

[7] Php Documentation: https://www.php.net/docs.php

[8] JavaScript: https://www.w3schools.com/js/, https://www.tutorialspoint.com/javascript/index.htm

9. ИЗЈАВА О АКАДЕМСКОЈ ЧЕСТИТОСТИ

Студент (име, име једног родитеља и презиме): **Небојша Бранко Илић**

Број индекса: **РТ-11-15**

Под пуном моралном, материјалном, дисциплинском и кривичном одговорношћу изјављујем да је завршни рад, под насловом:

ВЕБ апликација за евиденцију пројектних активности

1. резултат сопственог истраживачког рада;
2. да овај рад, ни у целини, нити у деловима, нисам пријављиво/ла на другим високошколским установама;
3. да нисам повредио/ла ауторска права, нити злоупотребио/ла интелектуалну својину других лица;
4. да сам рад и мишљења других аутора које сам користио/ла у овом раду назначио/ла или цитирао/ла у складу са Упутством;
5. да су сви радови и мишљења других аутора наведени у списку литературе/референци који је саставни део овог рада, пописани у складу са Упутством;
6. да сам свестан/свесна да је плагијат коришћење туђих радова у било ком облику (као цитата, прафраза, слика, табела, дијаграма, дизајна, планова, фотографија, филма, музике, формула, вебсајтова, компјутерских програма и сл.) без навођења аутора или представљање туђих ауторских дела као мојих, кажњиво по закону (Закон о ауторском и сродним правима), као и других закона и одговарајућих аката Високе школе електротехнике и рачунарства струковних студија у Београду;
7. да је електронска верзија овог рада идентична штампаном примерку овог рада и да пристајем на његово објављивање под условима прописаним актима Високе школе електротехнике и рачунарства струковних студија у Београду;
8. да сам свестан/свесна последица уколико се докаже да је овај рад плагијат.

У Београду, 12. 1. 2020.године

Својеручни потпис студента