Sustav upravljanja sadržajem realiziran pomoću Zend razvojnog okvira

Đekić, Nikola

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU
ELEKTROTEHNIČKI FAKULTET

Sveučilišni studij

SUSTAV UPRAVLJANJA SADRŽAJEM REALIZIRAN POMOĆU ZEND RAZVOJNOG OKVIRA

Diplomski rad

Nikola Đekić

Osijek, 2016.

SADRŽAJ

1. UVOD

Internet je javno dostupna globalna paketna podatkovna mreža koja zajedno povezuje računala i računalne mreže korištenjem internetskog protokola (engl. *Internet Protocol – IP*). [1] Takav način djelovanja svrstava Internet, kao jedinstvenu mrežu, u informacijski sustav. Navedeni informacijski sustav, danas je jedan od glavnih izvora kako informacija, tako i zabave, gdje vodeću ulogu imaju dinamičke internetske stranice, popularnije zvane internetske aplikacije (engl. *WEB applications*) (u daljnjem tekstu internetske aplikacije). [2]

Internetska aplikacija dinamički generira niz HTML (engl. *Hypertext Markup Language*) dokumenata koji se mogu vidjeti pomoću internetskog preglednika[1]. Svaka pojedina stranica prikazuje se korisniku kao statični dokument (stranica), ali se stranice interaktivno mogu mijenjati u internetskoj aplikacij. [3]

Većina današnjeg Interneta temeljena je na klijentsko-poslužiteljskoj strukturi. Struktura se temelji na dihotomiji između uređaja koji predstavljaju klijente i uređaja koji predstavljaju poslužitelje. Klijentski uređaj je npr. računalo kojim se pregledavaju internetske stranice, dok je poslužitelj računalo na kojem su sve te informacije sačuvane. Klijent šalje zahtjev prema poslužitelju, poslužitelj prima isti te vraća odgovarajući sadržaj.

Za potrebe diplomskog rada, odnosno izradu sustava upravljana sadržajem[2] (u daljnjem tekstu CMS) korištene su klijentske tehnologije HTML, CSS (engl. *Cascading Style Sheets*), Java Script, sustav za kreiranje predložaka internetskih stranica SMARTY (engl. *SMARTY template engine*), dok su za poslužiteljske tehnologije korišteni skrpitni jezik PHP (engl. *PHP: Hypertext Preprocessor*), strukturni upitni jezik SQL (engl. *Structured Query Language*) – obrađeno u drugom i trećem poglavlju, PHP razvojni okvir ZEND (engl. *ZEND Framewok*) te Doctrine 2 objektno-relacijsko mapiranje (engl. *Object Relational Mapper - ORM*) – obrađeno u četvrtom poglavlju. Peto poglavlje objašnjava problematiku stvaranja arhitketure CMS-a i njegovu izradu.

[1] Internetski preglednik (engl. *WEB browser*) je program koji korisniku omogućuje pregled internetskih stranica i multimedijalnih sadržaja vezanih uz njih.

[2] Sustava upravljana sadržajem (eng. *Content management system - CMS*): U najširem smislu odnosi se na svako rješenje koje omogućuje klasifikaciju, organizaciju, povezivanje i svaki drugi oblik uređivanja sadržaja.[4]

1.1. Zadatak završnog rada

Objasniti način korištenja i rada ZEND razvojnog okvira. Izraditi CMS aplikaciju uporabom ZEND razvojnog okvira. Omogućiti korisnicima pristup bazi podataka. Nakon izrade CMS sustava testirati rad izradom neke jednostavne web stranice.

2. KLIJENTSKE TEHNOLOGIJE

Klijentske tehnologije omogućavaju stvaranje vidljivog dijela internetske aplikacije te interakciju korisnika s internetskim servisom. Iste su zadužene preuzeti i prikazati podatke koji dolaze iz pozadine u internetskom pregledniku.

Potpoglavlja koje slijede opisuju sve sastavnice vidljivog dijela internetske aplikacije koje su korištene za potrebe diplomskog rada.

2.1. HTML

HTML je jezik za označavanje ili prezentacijski jezik koji se sastoji od kolekcije oznaka (engl. *tags*) pomoću kojih je omogućeno definiranje prikaza sadržaja na internetskoj stranici. Oznake su elementi internetske aplikacije koje se upotrebljavaju za određivanje načina prikaza iste. Najčešće se upotrebljavaju u paru koji okružuje element na koji utječu. Ono što čini temelj svakog HTML elementa njegov je identifikator koji internetskom pregledniku govori o kojem se elementu zapravo radi kako bi ga isti znao ispravno prikazati.

```
<!DOCTYPE html>
<html>
<head>
    <title>Heading</title>
</head>
<body>

<h1>Hello World</h1>

</body>
</html>
```

Sl.2.1. *Struktura HTML dokumenta*

Prema slici 2.1. `<html>` element govori internetskom pregledniku da se ovdje radi o HTML dokumentu. Kraj HTML dokumenta označava se s `</html>`. Sadržaj koji se nalazi između elemenata `<head>` i `</head>` predstavlja zaglavlje koje se ne prikazuje u internetskom pregledniku, svrha mu je detaljan opis stranice. `<title>` je element kojim je definiran naslov internetske stranice, nema veze s fizičkim imenom pod kojim je dokument fizički pohranjen u računalo. `<body>` predstavlja element gdje se nalazi sadržaj vidljiv korisniku internetske aplikacije, u ovom slučaju „Hello world“. [5]

2.2. CSS

CSS tehnologija služi za formatiranje prikaza sadržaja te se sastoji od selektora, svojstva te pripadajuće vrijednosti svojstva.

Selektorom odabiremo HTML element na koji se stil primjenjuje. Selektori mogu biti HTML oznake, CSS klase (engl. *class*), označni identifikatori te kombinacija navedenih selektora. Svojstvo određuje svojstvo koje mijenjamo određenom selektoru. Vrijednost određuje vrijednost koju dodjeljujemo nekom svojstvu (slika 2.2.). [6]

```
#buttonAddEndDate { margin-left: 2em;}
```

Sl.2.2. *CSS sintaksa*

2.3. JavaScript

JavaScript je klijentska tehnologija poput HTML-a i CSS-a, ali jedina koja omogućava dinamički prikaz sadržaja na internetskim stranicama. JavaScript je pravi skriptni jezik kojim se omogućava puna funkcionalnost unutar internetskog preglednika te uz CSS i HTML prezentacijske jezike omogućava gotovo neograničene mogućnosti kreiranja internetskih korisničkih sučelja.

Za razliku od HTML-a i CSS-a, smatra se da je JavaScript puno teže savladati, zbog toga su programeri razvili biblioteke (engl. *libraries*) koje olakšavaju implementaciju na internetsku stranicu tako da se koriste gotove skripte. Jedna od njih je *jQuery* skripta koja je danas najrasprostranjenija po pitanju dinamičkih skripti (slika 2.3.). [7]

```
<script>
    $(document).ready(function(){
        $("#buttonAddEndDate").click(function(){
            $("#endDateField").show(100);
            $("#buttonAddEndDate").attr('disabled','disabled');
        });
    });
</script>
```

Sl.2.3. *JQuery sintaksa*

2.4. SMARTY

SMARTY je sustav za kreiranje predloška internetskih aplikacija napisanih u PHP-u. Kao takav, služi za odvajanje implementacije dotične internetske aplikacije od njene prezentacije, što olakšava održavanje njenog izgleda i njene funkcionalnosti, kao i paralelan rad programera i dizajnera. Činjenica da omogućuje proširivanje skupa naredbi, kao i korištenje PHP-a kada je to neizbježno, čini SMARTY vrlo fleksibilnim alatom za održavanje predloška internetskih aplikacija. SMARTY kôd se u osnovi sastoji od običnog HTML kôda, u koji se umeću SMARTY naredbe. Nakon obrade PHP prevoditelja, rezultirajući kôd je običan HTML spreman za prikaz u pregledniku.

SMARTY instrukcije označavaju se vitičastim zagradama. Instrukcije mogu biti naredbe za prikaz sadržaja, naredbe za kontrolu toka, odnosno petlje i grananja, te funkcije, koje mogu biti ugrađene u SMARTY-u i definirane od strane programera. [8]

```
{foreach from=$this->educations item=education}
<tr>
   <td>{$this->escape($education->getQualification()->getName()|truncate:50:"...":true)}</td>
</tr>
{/foreach}
```

Sl.2.4. *SMARTY sintaksa*

Na slici 2.4. prikazana je iteracija niza objekata „educations“ gdje je prikazano svojstvo „ime“ svakog objekata unutar iteracije. Ukoliko je rezultat dulji od pedeset znakova, nakon pedesetog znaka stavljaju se tri točkice kako bi se sačuvala prilagodljivost dizajna za male ekrane[3].

[3] Prilagodljivost (eng. *Responsive design*) je tehnika kojom se izrađuju internetske aplikacije koje reagiraju na različite dimenzije uređaja ili internetskog preglednika kojima im pristupamo. Za pojedine širine ekrana postoji poseban dio stilova koji prilagođava izgled stranice. [9]

3. POSLUŽITELJSKE TEHNOLOGIJE

Poslužiteljske tehnologije izvode se na poslužitelju gdje je internetska aplikacija smještena (engl. server).

Potpoglavlja koja slijede opisuju sve sastavnice poslužiteljskih tehnologija koje su korištene za potrebe diplomskog rada.

3.1. PHP

PHP je jezik za pisanje skripti koje rade na poslužitelju, namjenski stvoren za korištenje na Internetu. Unutar HTML dokumenta može se implementirati PHP kôd koji se izvršava kada posjetitelj Internet lokacije zatraži stranicu. Poslužitelj tumači PHP kôd implementiran u internetsku aplikaciju te generira HTML kôd ili drugu vrstu izlaznih podataka koje posjetitelj ima mogućnost vidjeti. PHP za prijenos podataka koristi HTTP (engl. *Hypertext Transfer Protocol)* koji funkcionira na modelu zahtjeva i odgovora. [10]

```php
public function educationlistAction(){
    $this->view->actionName = "educationlist";
    $educations = $this->_em->getRepository('App_Model_Education')->getEducations($this->_user);
    if($educations) $this->view->educations = $educations;
}
```

Sl.3.1. *Primjer PHP kôda*

Slika 3.1. predstavlja primjer PHP kôda, odnosno akcije unutar upravitelja (engl. *controller*) koja na pogled (engl. *view*) šalje niz objekata modela „App_Model_Education“.

3.2. SQL

SQL ili strukturirani upitni jezik je jezik koji omogućuje pristup bazi podataka (engl. *database*), dohvaćanje podataka iz baze, dodavanje, brisanje te izmjenu postojećih podataka u bazi. Pored toga, SQL daje mogućnost definiranja elemenata baze podataka i administraciju korisnika i prava pristupa podacima. [10]

Naredbe kojima upravljamo relacijskim bazama podataka putem SQL-a nazivaju se upiti (engl. *Query)* koje možemo postavljati koristeći operatore poput =, > i <. Slika 3.2. predstavlja naredbu kojom je stvorena tablica „user“ s pripadajućim svojstvima „id“, „password“, „created“ i „email“.

```
CREATE TABLE user
(
        id INT AUTO_INCREMENT NOT NULL,
        password VARCHAR(255) DEFAULT NULL,
        created DATETIME NOT NULL,
        email VARCHAR(255) DEFAULT NULL,
        PRIMARY KEY(id)
) DEFAULT CHARACTER SET utf8 COLLATE utf8_unicode_ci ENGINE = InnoDB
```

Sl.3.2. *Primjer SQL naredbe*

4. ARHITEKTURA CMS-A

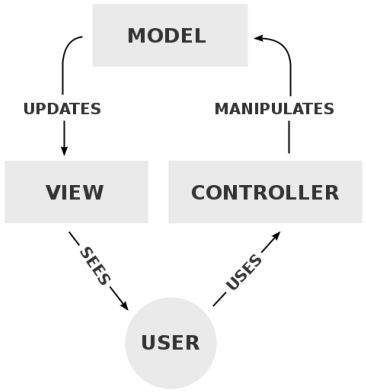
Arhitektura CMS-a zasnovana je na MVC ZEND razvojnom okviru te Doctrine 2 objektno-relacijskom mapiranju.

4.1. MVC

MVC (engl. Model-View-Controller) kao obrazac softverske arhitekture koristi se u programskom inženjerstvu za odvajanje pojedinih dijelova aplikacije u komponente ovisno o njihovoj namjeni. MVC dijeli aplikaciju na tri glavne komponente:

- model (engl. *model*)
- pogled (engl. *view*)
- upravitelj (engl. *controller*)

MVC programski obrazac specificira gdje će biti smješten određeni tip aplikacijske logike. Tako će logika korisničkog sučelja[4] biti u komponenti pogleda, poslovna logika u model komponenti, dok će logika unosa biti smještena u upravitelj komponenti. Ovakvo razdvajanje omogućuje manju kompleksnost pri razdvajanju aplikacija tako što omogućava fokusiranje samo na jedan njen dio (slika 4.1.).



Sl.4.1. *Shema MVC obrazaca*[5]

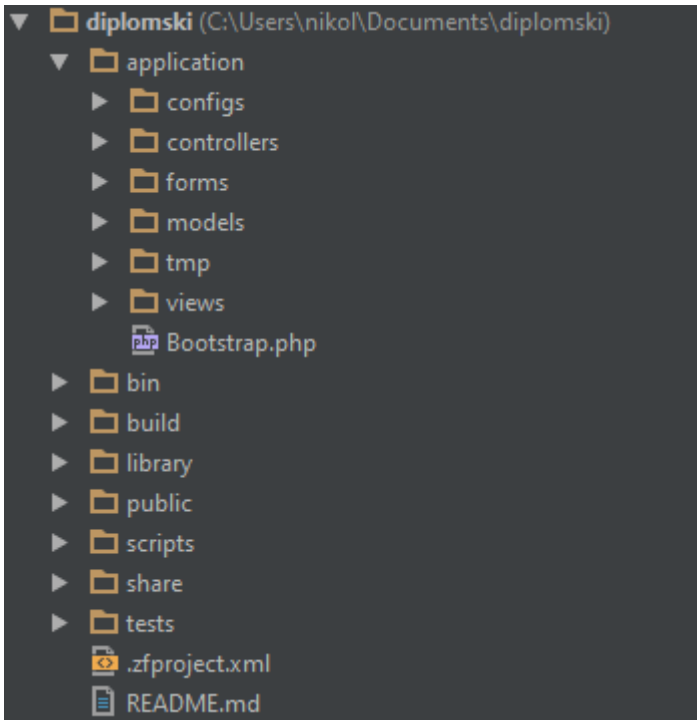
[4] Korisničko sučelje je premosnica između računalnog sustava ili programa i korisnika. Pomoću korisničkog sučelja upravljamo računalom, koristeći se pritom ulaznim uređajima poput miša, tipkovnice i touchscreena (zaslona osjetljivog na dodir). Izlazni uređaj na kojem se vizualno manifestiraju brojne naredbe, kao i naše akcije u današnjem korisničkom sučelju je monitor.[11]

[5] Izvor: https://commons.wikimedia.org/wiki/File:MVC-Process.svg (2016-06-28)

4.2. ZEND razvojni okvir

Pojam razvojni okvir (engl. framework) često se koristi u programskom inženjerstvu, posebno kada se govori o projektiranju i implementaciji objektno orijentiranih aplikacija. Najjednostavnije rečeno, okvir predstavlja kostur aplikacije, koji sadrži kompletan kod osnovnih funkcija cijelog sustava, a koji se može prilagoditi za potrebe određene aplikacije.

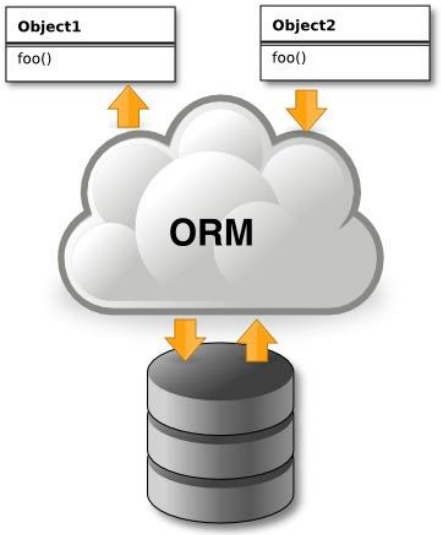
ZEND razvojni okvir je okvir otvorenog kôda za izradu internetskih aplikacija i servisa zasnovan na programskom jeziku PHP. ZEND je u potpunosti implementiran korištenjem objektno orijentirane paradigme. Cilj ZEND-a je pružiti programerima robusnu okolinu za razvoj vlastitih internetskih usluga. On nudi MVC implementaciju visoke učinkovitosti (poglavlje 4.1.), apstraktni sloj nad bazom podataka koji olakšava upravljanje nad podacima i niz komponenata koje implementiraju prikaz HTML stranica te validaciju i filtriranje korisničkog unosa (slika 4.2.). [12]



Sl.4.2. *MVC arhitektura CMS-a*

4.3. Doctrine 2 objektno-relacijsko mapiranje

Objektno-relacijsko mapiranje (ORM) u računalnom svijetu je tehnika programiranja za pretvaranje podataka između nekompatibilnih tipova podataka u objektno-orijentiranim programskim jezicima. To zapravo strvara „virtualnu objektnu bazu podataka“ koja se može koristiti unutar programskog jezika (slika 4.3.).



Sl.4.3. *Shema ORM-a*[6]

Doctrine 2 je ORM koji je na vrhu snažnog podatkovnog apstrakcijskog sloja (DBAL). Jedna od ključnih značajki je mogućnost pisanja upita u objektno orijentiranom SQL dijalektu zvanom Doctrine Query Language (DQL), inspiriran Hibernates HQL-om. To daje razvojnim programerima dobru alternativu za SQL koji održava fleksibilnost bez potrebe nepotrebnog dupliciranja kôda. [13]

[6] Izvor: http://www.info-novitas.hr/tehnologija/orm/ (2016-06-29)

5. PROGRAMSKO RJEŠENJE

Potpoglavlja koja slijede opisuju problematiku, izradu te konačno programsko rješenje CMS-a.

5.1. Problematika CMS-a

CMS je sustav koji omogućuje upravljanje sadržajem. U najširem smislu odnosi se na svako rješenje koje omogućuje klasifikaciju, organizaciju, povezivanje i svaki drugi oblik uređivanja sadržaja. Nove generacije CMS-a, danas uglavnom služe dinamičkom kreiranju internetskih aplikacija nove generacije. Problematika svakog CMS-a je takva da je najbitnije doći do odgovarajuće i lako ostvarujuće poslovne logike te implementacije istu u algoritamskom obliku.

Za potrebe diplomskog rada, biti će izrađen CMS pod nazivom *Curiculum vitae* koji omogućava korisniku jednostavnu izradu detaljnog životopisa po Europass[7] standardu, a na kojem će biti pokazane neke od mogućnosti ZEND razvojnog okvira te objektno-relacijskog mapiranja (objašnjeno u četvrtom poglavlju).

Glavne funkcionalnosti CMS-a podijeljene su u dvije kategorije: korisnik i administrator, a iste će biti objašnjene u narednim potpoglavljima.

- Korisnik (korisnički upravitelj)
    - Registracija novog korisnika
    - Prijava postojećeg korisnika
    - Unos, prikaz, promjena i brisanje podataka
        - Osobne informacije
        - Obrazovanje
        - Radno iskustvo
        - Znanje stranih jezika
        - Vještine
- Administrator (administratorski upravitelj)
    - Registracija novog korisnika
    - Uređivanje postojećih korisnika
    - Dodavanje novih obavijesti

[7] Europass: pet dokumenata koji europskim državljanima pomažu da svoje vještine i kvalifikacije jasno i razumljivo predstave u Europi.

5.2. Model

Model se sastoji od podataka, poslovnih pravila, logike i funkcija ugrađenih u programsku logiku. Čini samu bit aplikacije, a sastoji se od skupa klasa koje modeliraju i podržavaju rješavanje problema kojim se aplikacija bavi i taj dio je obično stabilan i trajan koliko i sam problem. Postoji nekoliko načina na koje je moguće stvoriti kostur modela. Programer može prvo definirati klase te odnose atributa unutar svake klase i tako stvarati bazu podataka ili može prvo stvoriti bazu podataka te dopusti samom procesu da automatski stvara klase. Model u MVC arhitekturi obično se sastoji od modela područja (engl. *Domain Model*) i modela aplikacije (engl. *Application Model*). Model područja sadrži glavne objekte koji opisuju problem (npr. korisnici, edukacije, vještine...), dok se model aplikacije sastoji od tehničkih objekata potrebnih za izradu aplikacije (npr. popis registriranih korisnika...). [14]

5.2.1. Implementacija Doctrine 2 u ZEND razvojni okvir

Kako Doctrine 2 nije sastavni dio ZEND razvojnog okvira, potrebno ga je implementirati u isti i to na sljedeći način:

- Doctrine biblioteku potrebno je preuzeti sa službene internetske stranice[8] iste te ju uključiti u ZEND biblioteku (`diplomski/library/Doctrine/`)
- Unutar Bootstrap[9] datoteke (`diplomski/application/Bootstrap.php`) potrebno je kreirati javnu metodu `_initDoctrine()` kojom se postavljaju globalne Doctrine postavke za čitav projekt
- Unutar ZEND konfiguracijske datoteke (`diplomski/application/application.ini`) potrebno je navesti podatke kojima uspostavljamo komunikaciju s lokalnom bazom podataka (slika 5.1.)

```
;Doctrine settings
doctrine.conn.host = '127.0.0.1'
doctrine.conn.user = 'nikola'
doctrine.conn.pass = 'nikola'
doctrine.conn.driv = 'pdo_mysql'
doctrine.conn.dbname = 'app'
doctrine.path.models = APPLICATION_PATH "/models"
```

Sl.5.1. *Uspostavljanje komunikacije s bazom podataka*

[8] http://www.doctrine-project.org/projects/orm.html

[9] Bootstrap: pruža zajedničku funkcionalnost za većinu pokretačkih potreba, uključujući i provjeru ovisnosti algoritama, kao i sposobnost za učitavanje resursa na zahtjev

5.2.2. Modeliranje

Pomoću Doctrine 2 anotacija, moguće je brzo i jednostavno modelirati bazu podataka i generirati SQL naredbe kojima se stvaraju tablice unutar baze podataka.

```php
<?php
use Doctrine\ORM\Mapping as ORM;

/**
 * App_Model_Education
 *
 * @ORM\Table(name="skill")
 * @ORM\Entity(repositoryClass="App_Model_SkillRepository")
 */
class App_Model_Skill
{
    /**
    * @ORM\Column(type="string",nullable=false)
    */
    private $name;

}
```

Sl.5.2. *Doctrine 2 anotacije*

Slika 5.2. predstavlja jednostavni primjer modeliranja objektno-relacijskim mapiranjem, a tekst koji slijedi objašnjava svaku od komponenata:

- `@ORM\Table(name="skill")` definira naziv tablice
- `@ORM\Entity(repositoryClass="App_Model_SkillRepository")` definira naziv repozitorija u kojem se navode SQL upiti
- `@ORM\Column(type="string",nullable=false)` definira tip podataka atributa i je li atribut obavezan prilikom unosa

Kada su definirane sve potrebne anotacije, potrebno je generirati model i metode za postavljanje i dohvaćanje vrijednosti (engl. *getter*, *setter*) atributa (slika 5.3.) pomoću konzole linije kôda:

```
php "..\bin\doctrine" orm:generate-entities --generate-annotations=true --regenerate-entities=true ../application/models/generated
```

```php
public function getName()
{
    return $this->name;
}

public function setName($name)
{
    $this->name = $name;

    return $this;
}
```

Sl.5.3. *Metode za postavljanje i dohvaćanje vrijednosti atributa*

Sljedeći korak je generiranje SQL naredbi (slika 5.4.), kojima se u ovom slučaju stvara tablica „skill“ s atributom „name“ pomoću konzolne linije kôda:

```
php "..\bin\doctrine-dbal" migration:diff --configuration "..\bin\migrations.xml"
```

```php
namespace DoctrineMigrations;

use Doctrine\DBAL\Migrations\AbstractMigration,
    Doctrine\DBAL\Schema\Schema;

class Version20160626181622 extends AbstractMigration
{
    public function up(Schema $schema)
    {
        // this up() migration is auto-generated, please modify it to your needs
        $this->abortIf($this->connection->getDatabasePlatform()->getName() != "mysql", "Migration can only be executed safely on 'mysql'.");

        $this->addSql("CREATE TABLE skill (name VARCHAR(255) NOT NULL) DEFAULT CHARACTER SET utf8 COLLATE utf8_unicode_ci ENGINE = InnoDB");
    }

    public function down(Schema $schema)
    {
        // this down() migration is auto-generated, please modify it to your needs
        $this->abortIf($this->connection->getDatabasePlatform()->getName() != "mysql", "Migration can only be executed safely on 'mysql'.");

        $this->addSql("DROP TABLE skill");
    }
}
```

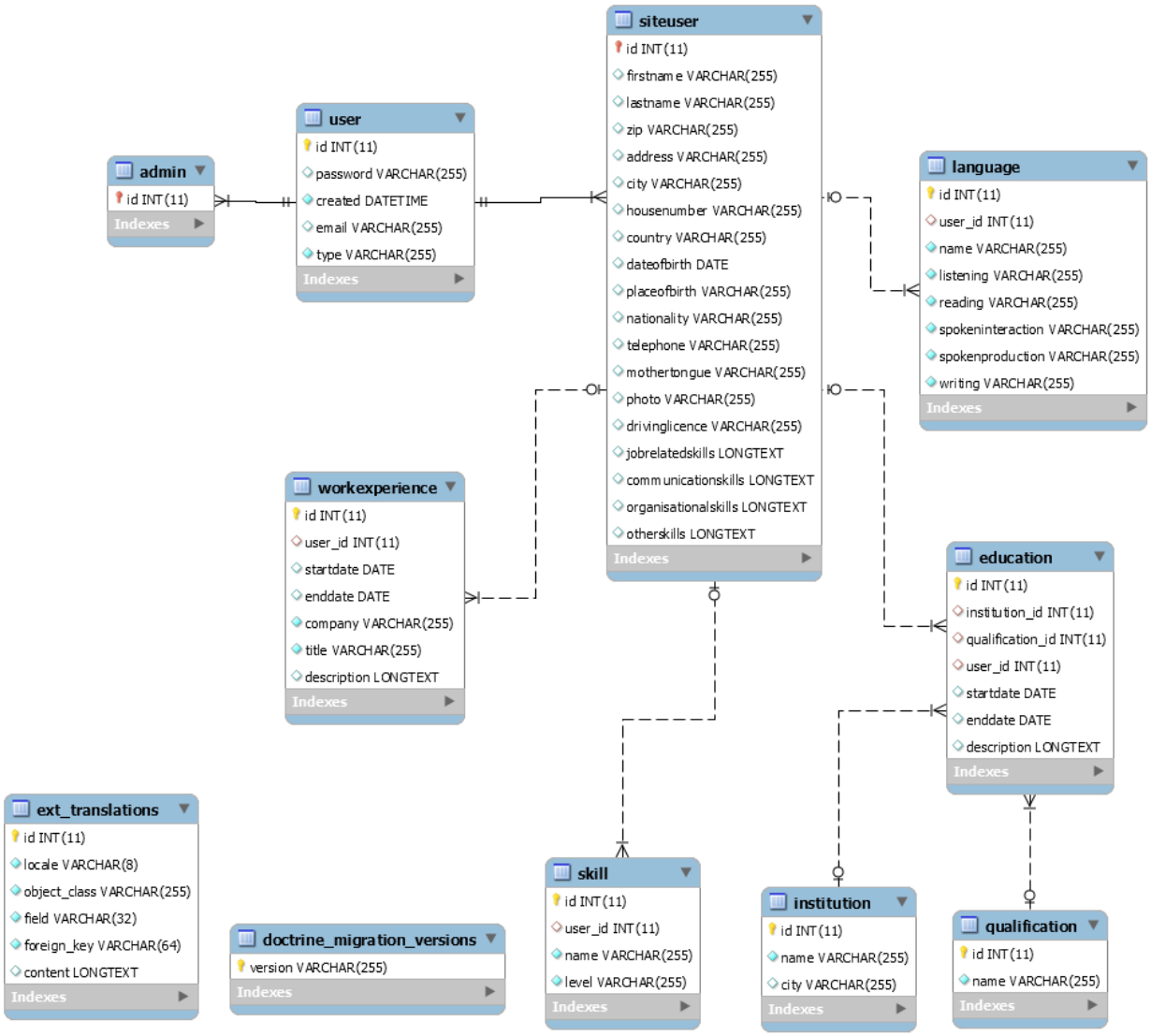
Sl.5.4.*Genereirani SQL upiti*

Posljednji korak je izvršavanje SQL naredbi konzolnom linijom kôda:

```
php "..\bin\doctrine-dbal" migration:migrate --configuration "..\bin\migrations.xml"
```

Tako su generirani su svi modeli CMS-a s pripadajućim atributima i vezama između istih. Slika 5.5. prikazuje ER dijagram[10] generiranog modela na kojem se zasniva poslovna logika CMS-a.

[10] ER dijagram (engl. *Entity-Relationship Diagram*): detaljan logički prikaz entiteta, veza i podataka radi organizacije poslovanja



Sl.5.5. *ER dijagram*

5.3. Upravitelj

Upravitelj u osnovi prima zahtjeve korisnika. Upravitelj je objekt koji upravlja pogledima. Pojednostavljeno se može reći da upravitelj kontrolira ulaz (engl. *input*), dok pogledi (engl. *views*) kontroliraju izlaz (engl. *output*). Da bi mogli upravljati ulazom, upravitelji moraju poznavati na kojoj se platformi, odnosno operacijskom sustavu i bazi podataka radi u internetskoj aplikaciji. Pogledi nasuprot tome ne moraju to poznavati. [14]

CMS *Curiculum vitae* sadrži četiri upravitelja koji će biti objašnjeni u narednom tekstu:

- Upravitelj greškama

- Korisnički upravitelj
- Standardni upravitelj
- Administratorski upravitelj

5.4. Upravitelj greškama

Upravitelj greškama (engl. *ErrorController*) sadrži akciju `errorAction()` kojoj je glavna zadaća dohvaćanje grešaka (slika 5.6.).

```
public function errorAction()
{
    $errors = $this->_getParam('error_handler');

    if (!$errors || !$errors instanceof ArrayObject) {
        $this->view->message = 'You have reached the error page';
        return;
    }

    switch ($errors->type) {
        case Zend_Controller_Plugin_ErrorHandler::EXCEPTION_NO_ROUTE:
        case Zend_Controller_Plugin_ErrorHandler::EXCEPTION_NO_CONTROLLER:
        case Zend_Controller_Plugin_ErrorHandler::EXCEPTION_NO_ACTION:
            // 404 error -- controller or action not found
            $this->getResponse()->setHttpResponseCode(404);
            $priority = Zend_Log::NOTICE;
            $this->view->message = 'Page not found';
            break;
        default:
            // application error
            $this->getResponse()->setHttpResponseCode(500);
            $priority = Zend_Log::CRIT;
            $this->view->message = 'Application error';
            break;
    }

    // Log exception, if logger available
    if ($log = $this->getLog()) {
        $log->log($this->view->message, $priority, $errors->exception);
        $log->log('Request Parameters', $priority, $errors->request->getParams());
    }

    // conditionally display exceptions
    if ($this->getInvokeArg('displayExceptions') == true) {
        $this->view->exception = $errors->exception;
    }

    $this->view->request   = $errors->request;
}
```

Sl.5.6. *Akcija* `errorAction()`

Na slici 5.6. može se vidjeti algoritam dohvaćanja grešaka i prikaz istih na pogled. Varijabla `errors` dohvaća niz objekata definiranih parametrom `'error_handler'`. Ukoliko je vrijednost `null` ili varijabla ne sadrži niz objekata, na pogled se prosljeđuje poruka „You have reached the

error page“. U suprotnom dohvaća se tip pogreške `$errors->type` i u ovisnosti od tipa, prosljeđuje se poruka na pogled.

5.5. Korisnički upravitelj

Korisnički upravitelj (engl. *UserController*), u narednom tekstu *UserController*, kao upravitelj s najviše akcija čini najveći dio algoritma CMS-a. Nadalje biti će objašnjene sve akcije i njima pripadajuće forme.

5.5.1. Registracija

Akcija `registerAction()` (slika 5.7.) omogućava registraciju novog korisnika na sustav pomoću obrasca (engl. *form*) `App_Form_Register()` koja za parametre prima niz `$preData` u kojem se nalaze željena elektronička pošta (engl. *email*) i zaporka (engl. *password*) korisnika. Ukoliko je niz `$preData` poslan POST zahtjevom[11], prolazi unaprijed definiranu validaciju (slika 5.8.) te se poziva metoda `save()` koja sprema validirane podatke u model `App_Model_SiteUser` čime se stvorio novi korisnik.

```
public function registerAction() {
    $this->view->actionName = "register";
    $this->checkIfUserIsLoggedIn();
    $preData = null;
    $registered = false;
    if($this->getRequest()->isPost()) $preData = $this->getRequest()->getPost();
    $form = new App_Form_Register(null,$preData);
    $this->view->form = $form;
    if($this->getRequest()->isPost() ) {
        $formData = $this->_request->getPost();
        if ($form->isValid($formData)) {
            $object = $form->save(null);

            if ($object) {
                $this->_em->persist($object);
                $this->_em->flush();
            }

            $registered = true;
        }
    }
    $this->view->registered = $registered;

}
```

Sl.5.7. *Akcija* `registerAction()`

```
$email = $this->addElement('text', 'email', array(
    'filters'     => array('StringTrim'),
    'placeholder' => 'Email',
```

[11] POST zahtjev: zahtijeva da se s poslužitelja prihvaća podatak poslan unutar HTTP (engl. *Hypertext Transfer Protocol*) zahtjeva.

```
    'validators' => array(
        new Zend_Validate_EmailAddress(),
        new App_Form_Validate_EmailExists(),
        array('StringLength', false, array(0, 60)),
    ),
    'required'   => 'required',
    'class' => 'form-control',
    'label'    => $view->translate('Email')
));
$this->getElement('email')->setAttrib('required', '');
```

Sl.5.8. *Dodavanje novog elementa forme i validacija istog*

5.5.2. Prijava

Akcija `loginAction()` instancira objekt klase `App_Form_Login` te isti prosljeđuje na pogled gdje se prikazuju HTML polja za unos kao što su elektronička pošta i zaporka. Ako korisnik unese vrijednosti u oba polja te potvrdi unos, vrijednosti se prosljeđuju na akciju `processloginAction()` gdje je smještena logika prijave (slika 5.9.).

```
public function processloginAction()
{
    $this->checkIfUserIsLoggedIn();
    $request = $this->getRequest();

    if (!$request->isPost()) {
        return $this->_helper->redirector('login');
    }

    $form = new App_Form_Login();
    if (!$form->isValid($request->getPost())) {
        $this->view->form = $form;
        return $this->render('login');
    }

    $adapter = new App_Model_AuthAdapter($form->getValue('email'),$form->getValue('password'));
    $auth    = Zend_Auth::getInstance();
    $result  = $auth->authenticate($adapter);

    if (!$result->isValid()) {
        $form->setDescription('Invalid email or password');
        $this->view->form = $form;
        return $this->render('login');
    }

    $identity = $auth->getIdentity();
    $user = $this->_em->getRepository('App_Model_User')->find($identity['id']);
    $registry = Zend_Registry::getInstance();
    $registry->user = $user;
    $this->_helper->redirector('dashboard','user');

}
```

Sl.5.9. *Logika prijave*

Na slici 5.9. prikazan je način korištenja osnovnih alata ZEND razvojnog okvira, a to su `ZendAuth` i `Zend_Registry`.

- `ZendAuth` dohvaća objekt korisnika, odnosno njegovu šifru prilikom uspješne prijave te ju čuva dokle god nije došlo do pozivanja metode `clearIdentity()` koja briše objekt iz sjednice (engl. *session*) i odjavljuje korisnika
- `Zend_Registry` omogućava spremanje objekta korisnika iz `ZendAuth` u `$registry->user` čime se omogućava provjera je li korisnik prijavljen na sustav unutar svakog upravitelja, odnosno pripadajućih akcija

5.5.3. Unos, promjena i brisanje

UserController sadrži četiri modula na kojima je omogućen unos novih, te promjena i brisanje postojećih vrijednosti te jedan modul gdje je omogućena isključivo promjena postojećih podataka.

Unos, promjena i brisanje kreirani su jedinstvenim algoritmom prikazanim na slici 5.10.

```
public function educationAction() {

    $this->view->actionName = "education";
    $saved = false;
    $educationId = $this->getRequest()->getParam('id');
    $educationModel = (!$educationId) ? null : $this->_em->getRepository('App_Model_Education')-
>findOneBy(array('id' => $educationId));

    $form = new App_Form_Education($educationModel);
    if ($this->getRequest()->isPost()) {
        $formData = $this->getRequest()->getPost();
        if ($form->isValid($formData)) {
            $object = $form->save($educationModel);
            if($object) {
                $this->_em->persist($object);
                $this->_em->flush();
                $saved = true;
            }
        } else {
            $this->view->form = $form;
        }
    }

    $deleteValue = $this->getRequest()->getParam('delete');
    $isDeleted = false;
    if($educationModel && !empty($deleteValue)){
        $this->_em->remove($educationModel);
        $this->_em->flush();
        $isDeleted = true;
    }

    $this->view->form = $form;
    $this->view->saved = $saved;
    $this->view->isDeleted = $isDeleted;


}
```

Sl.5.10. *Unos, promjena i brisanje*

Slika 5.10. prikazuje akciju `educationAction()` koja ima ulogu dodavanja novih edukacija, odnosno promjenu ili brisanje postojećih. Ako se detaljnije prouči algoritam, vidljivo je kako se opcija dodavanja i uređivanja razlikuje samo u jednom parametru. Taj parametar je objekt edukacije koji može postojati ili može biti `null`. Ako postoji, riječ je o promjeni, u suprotnom riječ je o dodavanju nove edukacije.

Kako bi se provjerilo postoji li objekt, potrebno je provjeriti postoji li GET[12] parametar „id“ ZEND metodom `$this->getRequest()->getParam('id')`. Ako postoji, poziva se Doctrine metoda `$this->_em->getRepository('App_Model_Education')->findOneBy(array('id' => $educationId))` koja pretražuje postoji li unutar modela `App_Model_Education` objekt s parametrom „id“ identičnim kao GET parametar „id“. Ako postoji, objekt se instancira i spremi u varijablu `$educationModel` te se ista prosljeđuje u formu `App_Form_Education`, u suprotnom prosljeđuje se `null`. Nakon validacije, poziva se `save()` metoda koja sprema novi objekt ili sprema promjene postojećeg objekta (slika 5.11.).

```
public function save($object) {

    $auth = Zend_Auth::getInstance();
    $identity = $auth->getIdentity();

    if(!$object)
        $object = new App_Model_Education();

    $object->setStartdate(new DateTime($this->getValue('startdate')));
    if ($this->getValue('enddate') !== '')
        $object->setEnddate(new DateTime($this->getValue('enddate')));
    else
        $object->setEnddate(null);
    $object->setInstitution($this->_em->getRepository('App_Model_Institution')-
>findOneBy(array('id' => $this->getValue('institution'))));
    $object->setQualification($this->_em->getRepository('App_Model_Qualification')-
>findOneBy(array('id' => $this->getValue('qualification'))));
    $object->setUser($this->_em->getRepository('App_Model_User')->findOneBy(array('id' =>
$identity['id'])));
    if($this->getValue('description') !== '')
        $object->setDescription($this->getValue('description'));
    else
        $object->setDescription(null);

    return $object;
}
```

Sl.5.11. `save()` *metoda*

Brisanje je omogućeno GET parametrom „delete“, odnosno provjerom postoji li isti ZEND metodom `$this->getRequest()->getParam('delete')`. Ako postoji GET parametar „delete“

[12] GET zahtjev: asocijativni niz varijabli prosljeđen na pogled preko URL (engl. *Uniform Resource Locator*) parametra

te ako postoji objekt modela `App_Model_Education` poziva se Doctrine metoda `remove()` koja prima navedeni objekt i briše ga iz baze podataka.

5.5.4. Pregled

Pregled postojećih zapisa kreiran je jedinstvenim algoritmom prikazanim na slici 5.12.

```
public function educationlistAction(){
    $this->view->actionName = "educationlist";
    $educations = $this->_em->getRepository('App_Model_Education')->getEducations($this->_user);
    if($educations) $this->view->educations = $educations;
}
```

Sl.5.12. *Pregled*

Na slici 5.12. prikazan je način dohvaćanja niza objekata iz modela `App_Model_Education` koji pripadaju prijavljenom korisniku tako da se u metodu `getEducations()` proslijedi objekt korisnika iz `Zend_Registry` (poglavlje 5.2.2.) Navedena metoda je dio repozitorija modela `App_Model_Education` u kojem se navode svi SQL upiti za isti. (slika 5.13.).

```
use Doctrine\ORM\EntityRepository;

class App_Model_EducationRepository extends EntityRepository
{
    public function getEducations($user)
    {
        $qb = $this->_em->createQueryBuilder();
        $qb->select('e')
            ->from('App_Model_Education', 'e')
            ->where('e.user = :userId')
            ->setParameter(':userId', $user->getId())
            ->orderBy('e.startdate', 'DESC')
        ;

        return $qb->getQuery()->getResult();

    }
}
```

Sl.5.13. `getEducations()` *metoda*

Ako metoda vrati niz objekata, isti se prosljeđuje na pogled.

5.5.1. Kontrolna ploča

Kontrolna ploča, odnosno `dashboardAction()` akcija omogućava korisniku pregled unesenih podataka na jednom mjestu. Sve podatke moguće je dobiti iz objekta korisnika, što je omogućeno ispravnim modeliranjem (slika 5.5.).

5.6. Pogled

Pogled predstavlja bilo kakav prikaz podataka kao što je obrazac, tablica ili dijagram. Bavi se načinom prikaza informacija korisniku ili aplikaciji, odnosno odgovara na pitanje kako će se nešto prikazati korisniku ili drugoj aplikaciji, a sadrži sučelja koja su veza s modelom (s prvim dijelom arhitekture sustava). S obzirom na to da u aplikaciji obično postoji više sučelja, ovaj dio često se naziva u množini: pogledi (engl. *views*). [14]

Unutar ZEND razvojnog okvira broj pogleda mora biti identičan broju akcija u pojedinom upravitelju.

5.6.1. Registracija

Na slici 5.7. prikazan je način kako iz `registerAction()` akcije proslijediti objekt forme na istoimeni pogled. Objekt sadrži sva polja navedena u formi `App_Form_Register()` te se istima na pogledu pristupa preko operatora `->` (slika 5.14.).

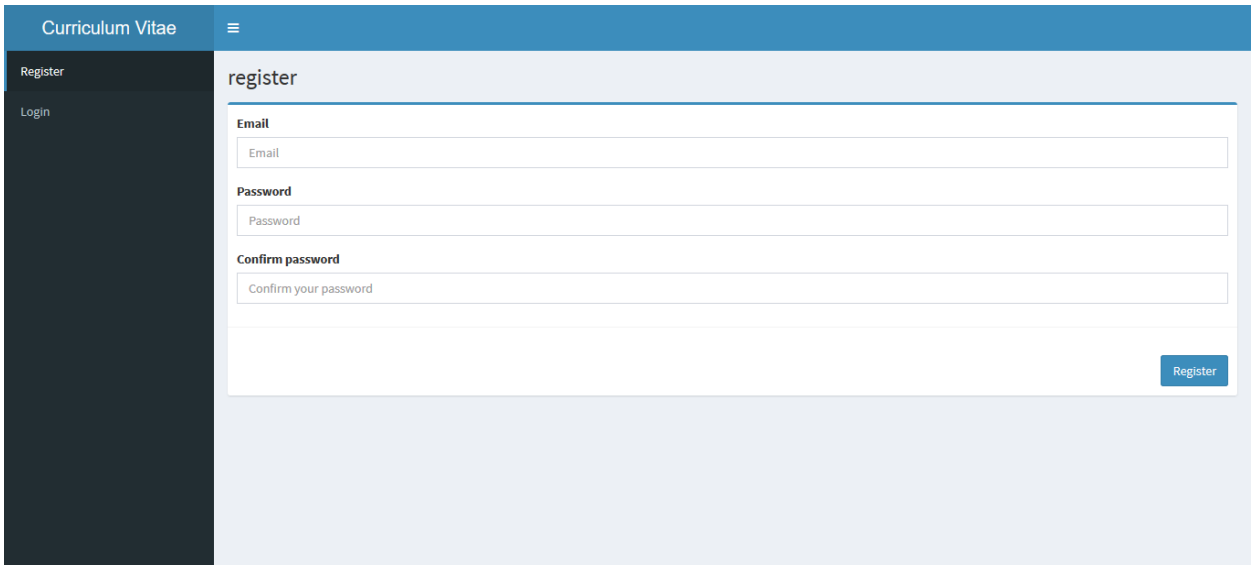
```
{if $registered eq true}
    <div class="alert alert-success text-center">
        <h4>Success!</h4>
    </div>
    <meta http-equiv="refresh" content="3; URL={$this->url(['controller' => 'user', 'action' =>
'login'], 'default', true)}">
{else}
<div class="box box-primary">
    <!-- form start -->
    <form role="form" method="{$form->getMethod()}" action="{$form->getAction()}">
        <div class="box-body">
            <div class="form-group">
                <div class="{if $form->email->hasErrors()}has-error{/if}">
                    {$form->email}
                </div>
            </div>
            <div class="form-group">
                <div class="{if $form->password->hasErrors()}has-error{/if}">
                    {$form->password}
                </div>
            </div>
            <div class="form-group">
                <div class="{if $form->passwordconfirm->hasErrors()}has-error{/if}">
                    {$form->passwordconfirm}
                </div>
            </div>
        </div>
        <div class="box-footer">
            {$form->register}
        </div>
    </form>
</div>
{/if}
```

Sl.5.14. *Pogled registracije*

Rezultat pogleda, odnosno korisničko sučelje vidljivo je na slici 5.15. Pogled sadrži polja za unos:

- Elektronička pošta
- Zaporka
- Ponovite zaporku

te gumb za potvrdu unosa.



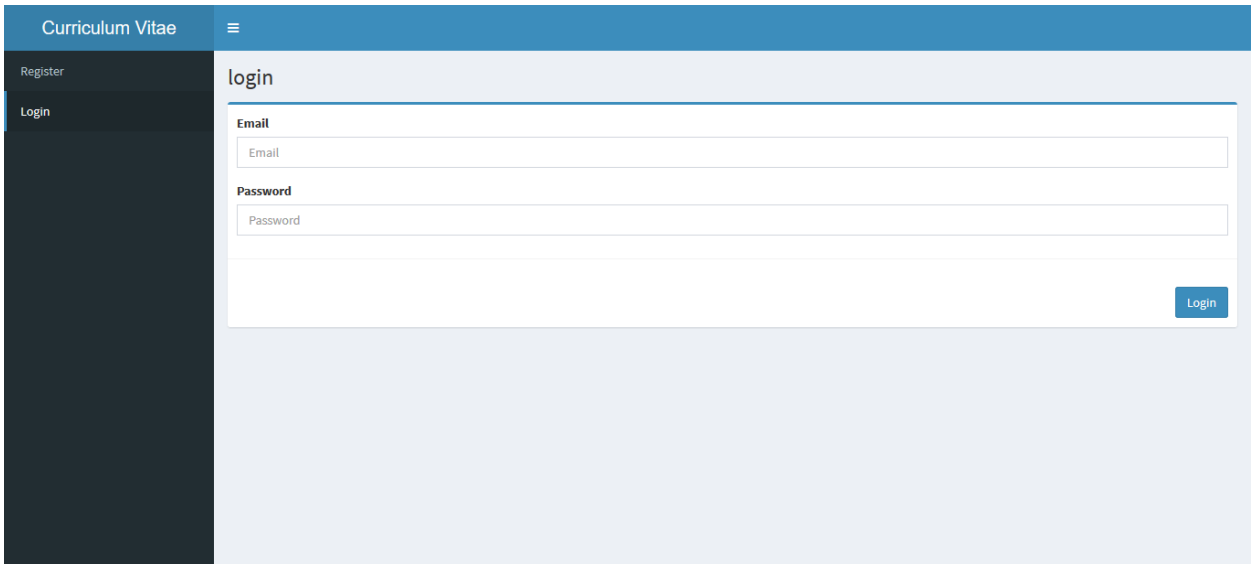
Sl.5.14. *Izgled pogleda registracije*

5.6.2. Prijava

Na isti način kao i kod pogleda registracije, iz akcije `loginAction()` na istoimeni pogled prosljeđen je objekt forme koji sadrži polja za unos:

- Elektronička pošta
- Zaporka
- Ponovite zaporku

te gumb za potvrdu unosa (slika 5.15.).

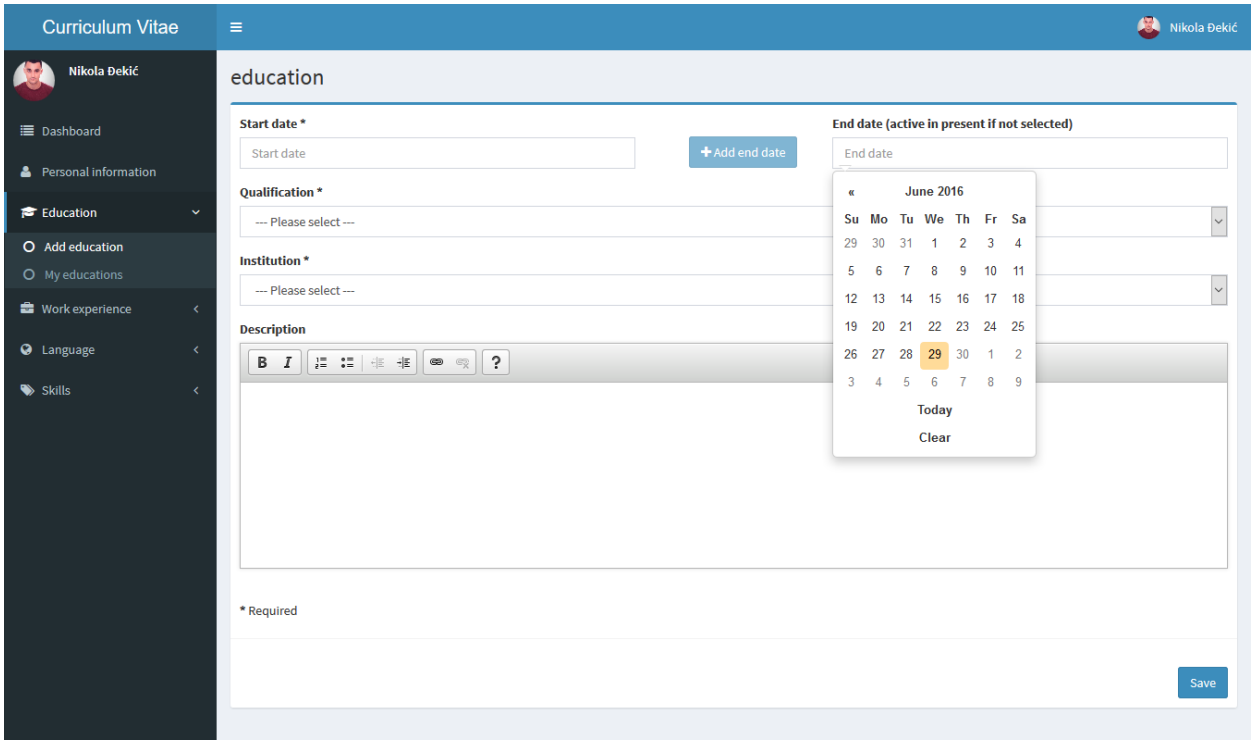


Sl.5.15. *Izgled pogleda prijave*

5.6.3. Unos, promjena i brisanje

Na slici 5.16. prikazan je način kako iz `educationAction()` akcije proslijediti objekt forme na istoimeni pogled. Pogled sadrži polja:

- Datum početka edukacije
- Datum kraja edukacije (opcionalno)
- Odabir stručne spreme
- Odabir ustanove gdje se stekla edukacija
- Dodatni komentar (opcionalno)



Sl.5.16. *Izgled pogleda unosa edukacije*

Opcionalno polje „datum kraja edukacije“ skriveno je prilikom otvaranja pogleda, te je isto moguće aktivirati pritiskom na gumb „Add end date“. Ako polje nije dodano, ili ako je dodano a nije ispunjeno, smatra se da korisnik i dalje pohađa edukaciju. Ovakav algoritam omogućen je klijentskom tehnologijom JavaScript, dok se ZEND razvojnim okvirom omogućava unos `null` u bazu podataka za navedeni slučaj (slika 5.17.).

```
<script>
    $(document).ready(function(){
        $('#startdate').datepicker({
            format: 'dd.mm.yyyy',
            todayBtn: "linked",
            clearBtn: true,
            autoclose: true,
            todayHighlight: true,
            endDate: '+0d'

        });
        $('#enddate').datepicker({
            format: 'dd.mm.yyyy',
            todayBtn: "linked",
            clearBtn: true,
            autoclose: true,
            todayHighlight: true,
            endDate: '+0d'
        });

        $("#startdate").change(function(){
            var $startPicker = $(this);
            var $endPicker = $("#enddate");
            var startDate = $startPicker.datepicker("getDate");
            var endDate = $endPicker.datepicker("getDate")
```

```
            if(endDate < startDate){
                endDate = new Date(startDate.getTime());
                $endPicker.datepicker("setDate", endDate);
            }else{
                endDate = new Date(startDate.getTime());
            }
            $endPicker.datepicker("setStartDate", endDate);

        });

        $("#buttonAddEndDate").click(function(){
            $("#endDateField").show(100);
            $("#buttonAddEndDate").attr('disabled','disabled');
        });
        CKEDITOR.replace( 'description' );

    });
</script>

if ($this->getValue('enddate') !== '')
    $object->setEnddate(new DateTime($this->getValue('enddate')));
else
    $object->setEnddate(null);
```

Sl.5.17. *Algoritam za dodavanje datuma*

Na slici 5.17. pored algoritma za dodavanja datuma kraja edukacije, prikazan je algoritam za izbor datuma. Naime, korisniku nije dopušteno staviti datum početka edukacije veći od trenutnog datuma, dok se datum kraja edukacije mora nalaziti u intervalu između početka edukacije i današnjeg datuma (ako je edukacija završila).

5.6.4. Pregled

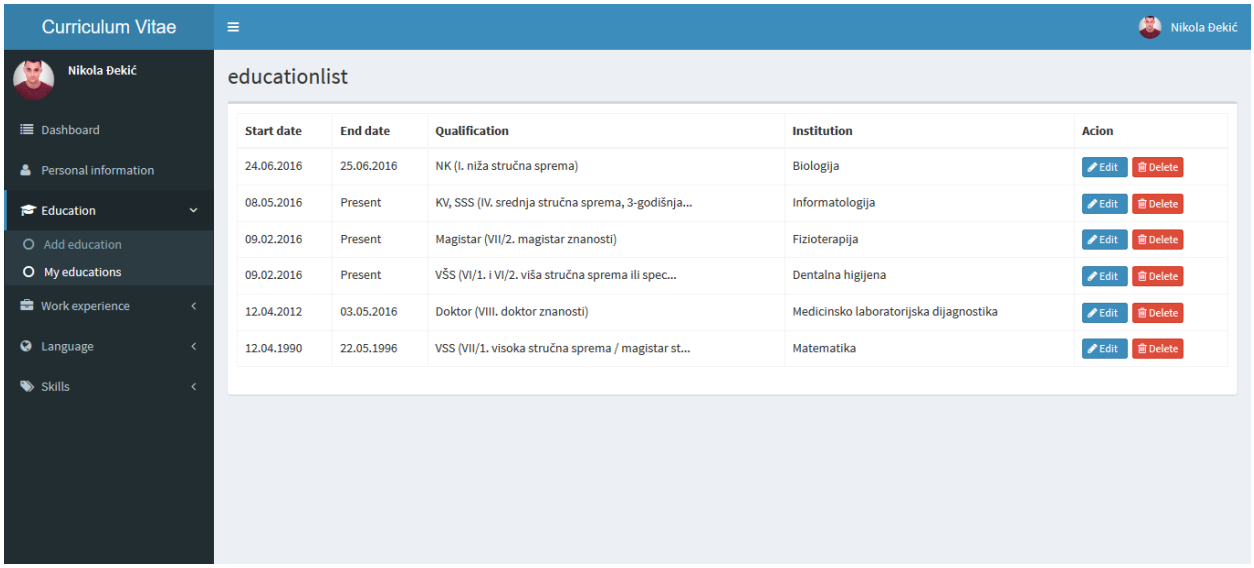
Na slici 5.12. prikazan je algoritam koji na pogled `educationlist` prosljeđuje niz objekata edukacija. Na pogledu, isti je potrebno iterirati SMARTY `foreach` petljom kako bi se prikazao pojedini objekt s pripadajućim atributima i njegovim svojstvima (slika 5.18.).

```
{if !$this->educations}
<br/>
<div class="alert alert-warning text-center">
    <h4>No data.</h4>
</div>
{else}
<div class="box">
    <!-- /.box-header -->
    <div class="box-body">
            <div class="table-responsive">
                <table class="table table-bordered table-hover">
                    <thead>
                    <tr>
                        <th>Start date</th>
                        <th>End date</th>
                        <th>Qualification</th>
                        <th>Institution</th>
                        <th>Acion</th>
```

```
                    </tr>
                    </thead>
                    <tbody>
                    {foreach from=$this->educations item=education}
                        <tr>
                            <td>{$this->escape($education->getStartdate()->format('d.m.Y'))}</td>
                            <td>{if $education->getEnddate()}{$this->escape($education-
>getEnddate()->format('d.m.Y'))}{else}Present{/if}</td>
                            <td>{$this->escape($education->getQualification()-
>getName()|truncate:50:"...":true)}</td>
                            <td>{$this->escape($education->getInstitution()-
>getName()|truncate:50:"...":true)}</td>
                            <td>
                                <a href="{$this->url(['controller' => 'user', 'action' =>
'education', 'id' => $education->getId()],'', false)}" class="btn btn-primary btn-xs"><i
class="fa fa-pencil" aria-hidden="true"></i> Edit  </a> 
                                <a href="{$this->url(['controller' => 'user', 'action' =>
'education', 'id' => $education->getId(), 'delete' => 'delete'],'', false)}" class="btn btn-
danger btn-xs"><i class="fa fa-trash-o" aria-hidden="true"></i> Delete</a>
                            </td>
                        </tr>
                    {/foreach}
                    </tbody>
                </table>
            </div>
    </div>
    <!-- /.box-body -->
</div>
<!-- /.box -->
{/if}
```

Sl.5.18. *SMARTY* `foreach` *petlja*

Rezultat iteracije (slika 5.18.) prikazan je na slici 5.19. Korisnik osim što ima pregled svih edukacija, također na intuitivan način može pristupiti promjeni, odnosno brisanju pojedine edukacije.



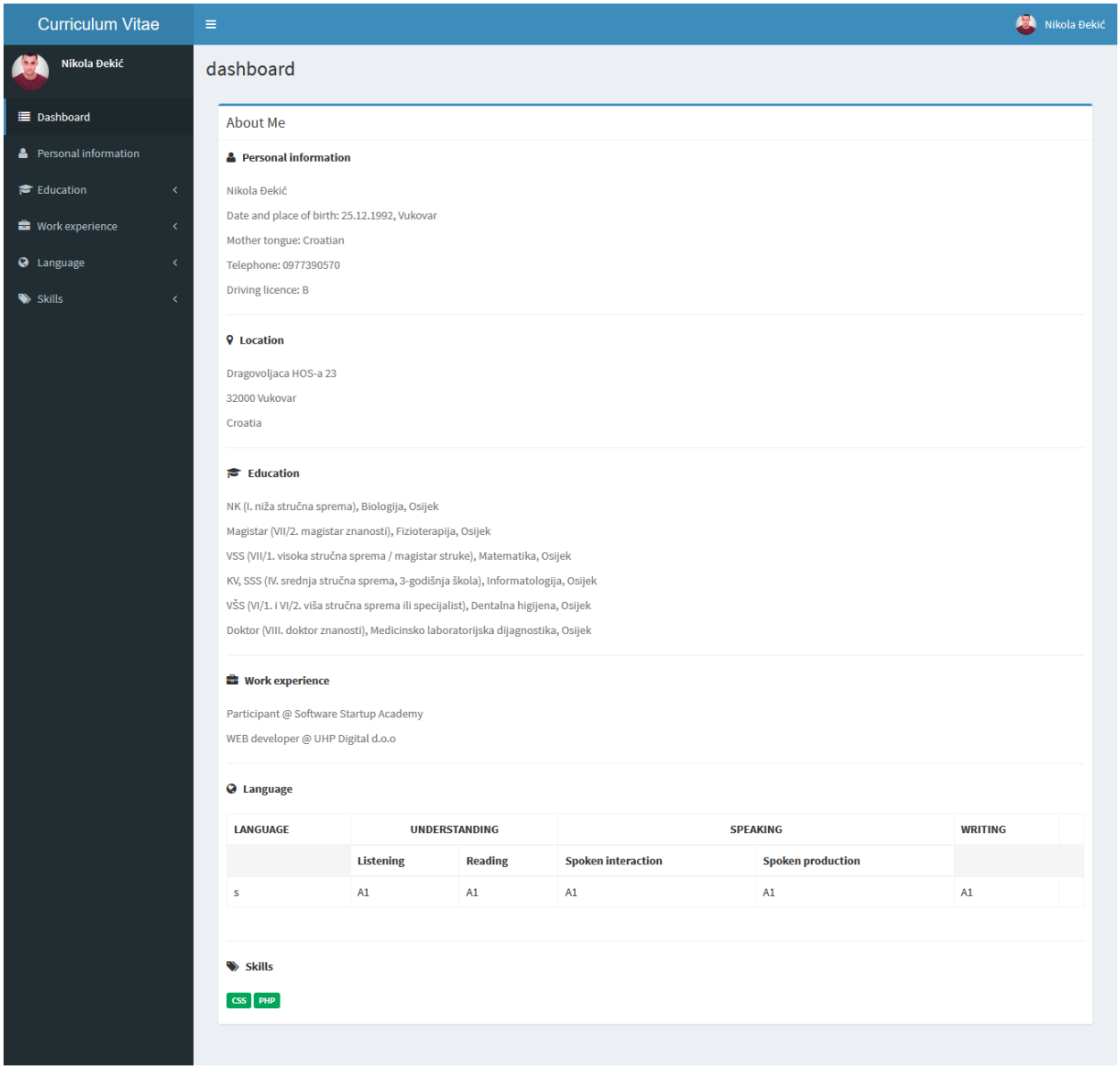
Sl.5.18. *Pregled edukacija*

5.6.5. Kontrolna ploča

Ako je korisnik prijavljen na sustav, odnosno ako postoji objekt korisnika, na pogledu kontrolna ploča (`dashboard`) moguće je vidjeti sve unesene podatke (slika 5.20.) iz objekta `$user` koji se na pogledu instancira kao što je prikazano na slici 5.19.

```
{if Zend_Registry::isRegistered('user')}{assign var=user
value=Zend_Registry::get('user')}{else}{assign var=user value=null}{/if}
```

Sl.5.19. *Istanciranje objekta korisnika na pogledu*



Sl.5.20. *Pogled kontrolne ploče*

Analogno pogledu kontrolne ploče, odnosno akcijama za unos, promjenu, brisanje i pregled kreirani su ostali pogledi.

5.7. Sigurnost

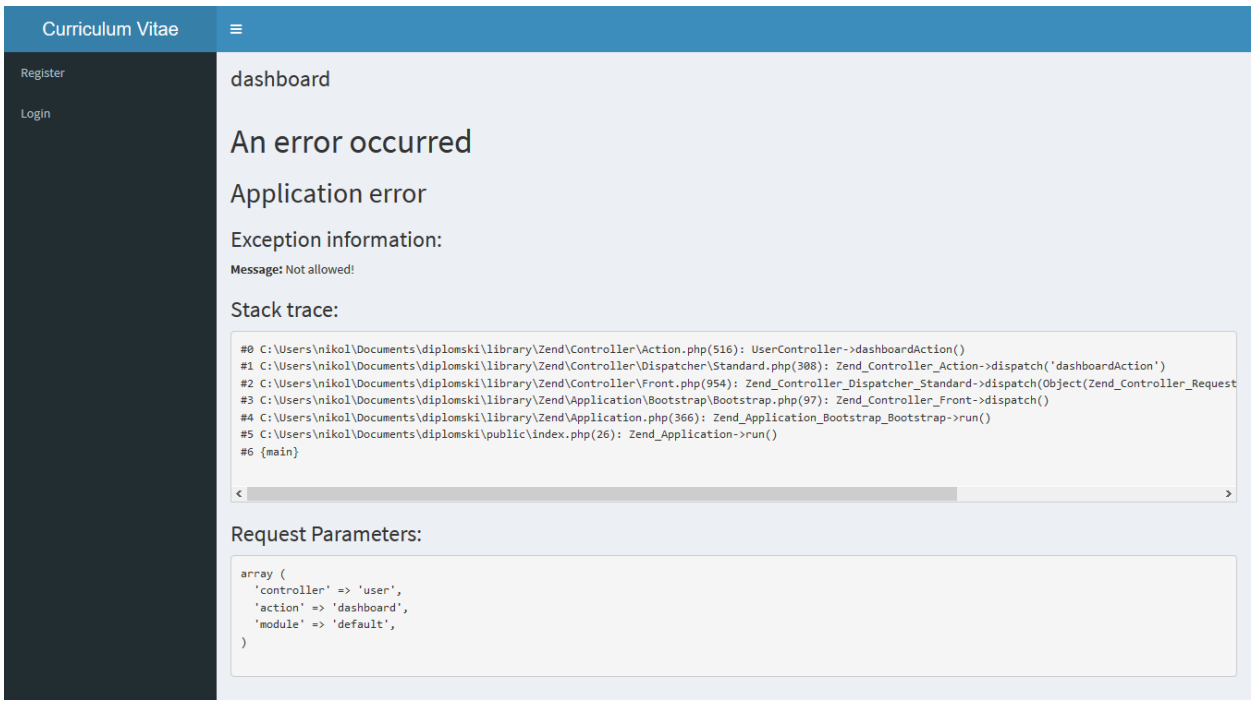
Sigurnost internetske aplikacije, pored same funkcionalnosti iste, jedna je od najvažnijih stavki. Naime, sve akcije, odnosno svi pogledi nisu dostupni svakome, tako da ih je potrebno ograničiti. To se može definirati pomoću ZEND dodatka `Zend_Acl` ili na vrhu svake akcije unutar pojedinog upravitelja. Slika 5.21. prikazuje način na koji se dohvati objekt korisnika prijavljenog na sustav i provjera istog.

```php
public function init()
{
    $registry = Zend_Registry::getInstance();
    $this->_em = $registry->entitymanager;
    $this->_user = isset($registry->user) ? $registry->user : null;
}

public function dashboardAction() {
    $this->view->actionName = "dashboard";
    if (!$this->_user) throw new Exception('Not allowed!');
}
```

Sl.5.21. *Provjera korisnika prijavljenog na sustav*

Metoda `init()` stvara objekt korisnika `$this->_user` prilikom aktivacije bilo koje akcije unutar upravitelja. Objekt sadrži sve podatke o korisniku tako da ih uzima iz `Zend_Registry` (slika 5.9.). Kako bi se ograničio pristup akcijama samo prijavljenim korisnicima, na vrhu svake akcije dodaje se provjera postoji li objekt `$this->_user`. Ako ne postoji, korisnik nije prijavljen te se aktivira upravitelj greškama (poglavlje 5.4.), odnosno prikazuje se pogled kao na slici 5.22.



Sl.5.22. *Pogled upravitelja greškama*

6. ZAKLJUČAK

Zadatak ovog diplomskog rada bila je izrada CMS-a pomoću ZEND razvojnog okvira. Za izgled CMS-a korišteni su HTML, CSS, JavaScript i SMARTY dok su za funkcionalnost korišteni PHP razvojni okvir ZEND i Doctrine 2 objektno-relacijsko mapiranje. Izradom istog, korisniku je omogućena registracija te prijava na sustav unutar koga korisnik ima mogućnost unosa, promjene, brisanja i pregleda podataka potrebnih za životopis po Europass standardu.

Prednost korištenja ZEND razvojnog okvira ogleda se u MVC arhitekturi koja poboljšava dizajn kôda, čime kôd postaje čitljiviji i lakši za održavanje, štiti obrasce od sitnih pogrešaka (kao što je sprječavanje SQL injekcije) te vodi računa oko detalja niske razine. Kada je riječ o nedostacima ZEND razvojnog okvora, gotovo da ih ne postoji, međutim potrebno je izdvojiti dosta vremena i truda kako bi se omogućilo ispravno korištenje svih resursa koje ZEND nudi s predpostakom da je razina znanja objektno orijentiranog programiranja na visokom nivou.

Daljnja proširenja i razvoj CMS-a su mogući. Prvenstveno, CMS bi trebao biti prevedena na više svjetskih jezika, a poželjno bi bilo napraviti ispis životopisa u *.pdf* formatu.

LITERATURA

[1] Željko Panian, "Informatički enciklopedijski rječnik", 2005.

[2] http://www.informatika.buzdo.com/s910-internet-definicija.htm (2016-06-27)

[3] http://www.mathos.unios.hr/wp/wp2009-10/P14_Web_aplikacije.pdf (2016-06-27)

[4] John H. Gibbons (1981) "Computer-Based National Information Systems: Technology and Public Policy Issues", str. 17.

[5] M, Hrnjak, HTML priručnik, Algebra d.o.o, Zagreb, 2009.

[6] M, Hrnjak, CSS priručnik, Algebra d.o.o, Zagreb, 2009.

[7] D, Goodman, JavaScript bible, IDG Books Worldwide, Inc., 1998.

[8] http://www.smarty.net/docs/en/ (2016-06-27)

[9] http://www.elements.hr/sto-je-responzivna-web-stranica.html (2016-06-27)

[10] L, Welling; L, Thomson, PHP i MySQL – Razvoj aplikacija za Web, Pearson Education, Inc., 2003.

[11] http://www.vidipedija.com/index.php?title=Korisni%C4%8Dko_su%C4%8Delje (2016-06-27)

[12] Dr. Siniša Vlajid: Projektovanje programa (Skripta), FON, Beograd, 2004.

[13] http://www.info-novitas.hr/tehnologija/orm/ (2016-06-29)

[14] http://nastava.fpmoz.ba/nastava/diplomski/fpmoz_diplomski_informatika_raic.pdf (2016-06-29)

SAŽETAK

Ovaj diplomski rad opisuje izradu sustava upravljana sadržajem pomoću ZEND razvojnog okvira i Doctrine 2 objektno-relacijskog mapiranja. Sve sastavnice izrade sustava upravljana sadržajem te alati korišteni za stvaranje relacijske baze podataka, poput HTML-a, CSS-a, JavaScript-a, PHP-a, SQL-a, ZEND-a te ORM-a detaljno su objašnjeni u ovom radu. Također, detaljno je opisano na koji se način korisnik može ispravno služiti sustavom. Kraj diplomskog rada čini zaključak u kojem su navedene moguće prilagodbe i nadogradnje te prednosti i mane korištenih tehnologoja.

Ključne riječi: ZEND razvojni okvir, sustava upravljana sadržajem, Doctrine 2 objektno-relacijsko mapiranje

ABSTRACT

Content management system built on ZEND framework

This thesis illustrates the design of a content management system using ZEND framework and Doctrine 2 Object-relational mapper. All of the components used in building the content management system, and the tools for setting up a relational database, such as HTML, CSS, JavaScript, PHP, SQL, ZEND and ORM are explained in detail in this thesis. Furthermore, ways in which a user may properly use the system, are described in detail. Finally, some of the possible adaptations and upgrades to the system and advantages/disadvantages of using technologies are specified in the Conclusion.

Keywords: ZEND framework, content management system, Doctrine 2 Object-relational mapping

ŽIVOTOPIS

Nikola Đekić rođen je 25.12.1992. godine u Vukovaru. Osnovnu školu „II Osnovna škola Vukovar“ završio je 2007. godine u Vukovaru. Pohađao Opću gimnaziju u Vukovaru, koju je završio 2011. godine. Iste godine upisao je preddiplomski studij računarstva na Elektrotehničkom fakultetu u Osijeku na kojem je 2014. godine stekao zvanje prvostupnika inženjera računarstva. 2014. godine upisuje diplomski studij računarstva na Elektrotehničkom fakultetu u Osijeku, smjer procesno računarstvo.