Web sustav za upravljanje sadržajem za PocDoc aplikaciju

Valentin, Branimir

Undergraduate thesis / Završni rad

2016

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **Josip Juraj Strossmayer University of Osijek, Faculty of Electrical Engineering, Computer Science and Information Technology Osijek / Sveučilište Josipa Jurja Strossmayera u Osijeku, Fakultet elektrotehnike, računarstva i informacijskih tehnologija Osijek**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:200:472157

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-04-20**

Repository / Repozitorij:

Faculty of Electrical Engineering, Computer Science and Information Technology Osijek

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA, OSIJEK

Stručni studij Informatike

WEB SUSTAV ZA UPRAVLJANJE SADRŽAJEM ZA POCDOC APLIKACIJU

Završni rad

Student: Branimir Valentin
Mentor: doc.dr.sc. Krešimir Nenadić

Osijek, 2016.

Sadržaj

1. UVOD 1
1.1.Zadatak rada 1
2. CMS – SUSTAV ZA UPRAVLJANJE SADRŽAJEM 2
3. TEHNOLOGIJE KORIŠTENE U IZRADI APLIKACIJE 4
3.1. HTML 4
3.2. CSS (Bootstrap) 4
3.3. JavaScript (jQuery) 4
3.4. PHP 5
3.5. MySQL 5
3.6. Laravel 6
4. STRUKTURA I DIZAJN APLIKACIJE 7
4.1. Instalacija projekta 7
4.2. Konfiguriranje razvojne okoline projekta 8
4.3. Kreiranje migracija i modela 9
4.4. Kreiranje ruta i kontrolera 12
4.5. Čitanje podataka putem API-a 14
4.6. Dizajn aplikacije ..... 17~~16~~
5. ZAKLJUČAK ..... 22~~21~~
LITERATURA ..... 23~~22~~
SAŽETAK ..... 24~~23~~
ABSTRACT ..... 25~~24~~
ŽIVOTOPIS ..... 26~~25~~

1. UVOD

Tema ovog završnog rada je izgraditi jednostavan Web sustav za upravljanje sadržajem (*Content Managment System*, u daljnjem tekstu CMS) za PocDoc aplikaciju. Cilj ovog rada je pružiti dovoljno informacija te steći praktična znanja koja će pomoći u daljnjem razvijanju CMS-a samostalno ili kao člana tima. U nastavku će kratko biti opisan CMS za PocDoc aplikaciju, dijelovi, funkcioniranje, kakve informacije pruža, koji su programski jezici korišteni te koji je njegov cilj.

Za izradu CMS-a za PocDoc aplikaciju korišten je skup skripti Laravel temeljen na PHP-u, opisni jezik HTML, stilski jezik CSS3, JavaScript te MySQL.

Glavni cilj CMS-a za PocDoc aplikaciju je omogućiti liječniku lakši unos simptoma, bolesti, popis simptoma za svaku bolest, te vratiti rezultat aplikaciji preko aplikacijskog programskog sustava (*Application Programming Interface* – u daljnjem tekstu API) u JSON formatu.[1]

O cijeloj izradi ovog procesa može se pročitati u poglavljima koja slijede.

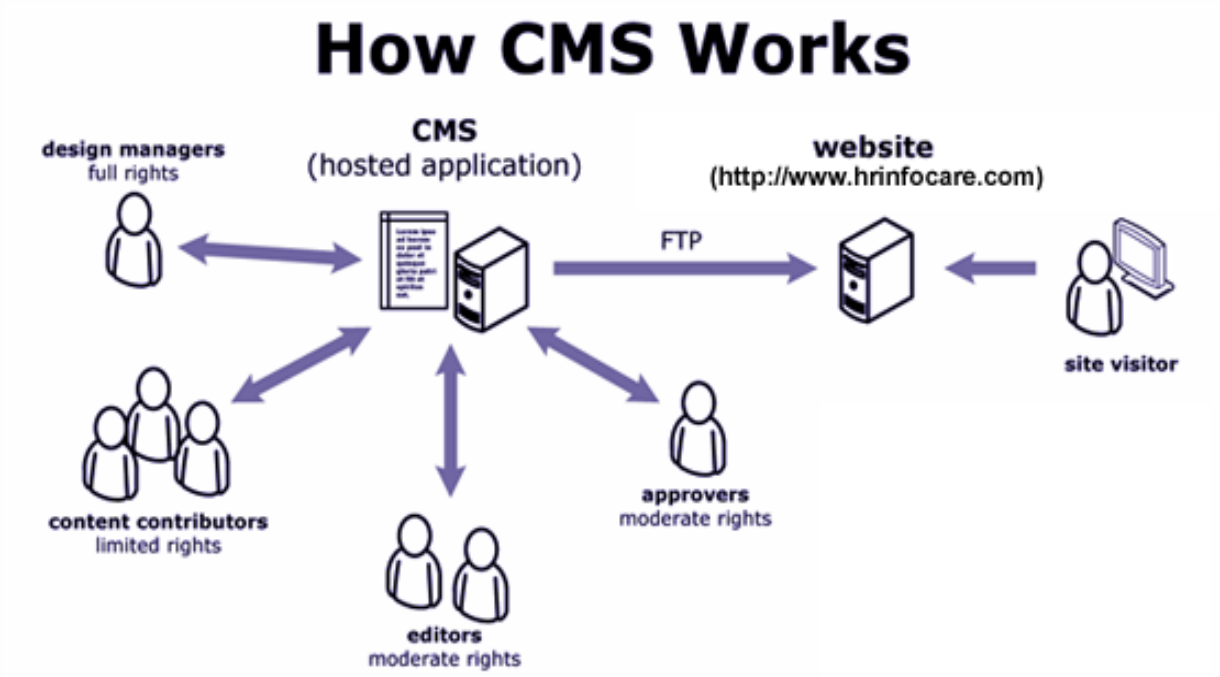
1.1. Zadatak rada

Objasniti način rada CMS-a te izgraditi jednostavan sustav za upravljanje sadržajem koji će biti primjenjen za izradu aplikacije PocDoc. U sustav ugraditi podršku za baze podataka. Pomoću izrađenog sustava napraviti jednostavnu web-stranicu uz objašnjenje načina izrade stranice.

[1] (*JavaScript Object Notation* – standard izveden iz JavaScript jezika koji je dizajniran za ljudima razumljivu razmjenu podataka)

2. CMS – SUSTAV ZA UPRAVLJANJE SADRŽAJEM

CMS (engl. *Content management system*) je sustav koji omogućuje upravljanje sadržajem web-stranica, kako programerima tako i samom korisniku. Koristi se kao rješenje koje omogućuje klasifikaciju, organizaciju, povezivanje i svaki drugi oblik uređivanja sadržaja tj. dodavanja sadržaja, slika ili mijenjanja istog. Iako se pojam može koristiti za manualne procese upravljanja sadržajem, danas se u prvom redu primjenjuje za različita programska rješenja koja omogućuju napredno upravljanje velikim brojem informacija. CMS sustavi se koriste pri sinkronizaciji podataka iz više izvora, za izvršavanje kolaborativnih projekata, za organizaciju rada u korporacijskim okruženjima i slično.



Sl. 2.1 *Rad CMS-a*

Temeljna primjena CMS-a danas je u dinamičkom kreiranju web-stranica nove generacije koristeći CMS rješenja, a svima njima može se upravljati s jednog mjesta.

Neke od osobina koje dobar CMS treba sadržavati su:

- web sučelje za upravljanje podacima
- online uređivanje sadržaja s vjernim pregledom (*WYSIWYG*)
- ugrađeno pretraživanje po ključnim riječima i kategorijama
- sustav različitih prava pristupa zasnovan na ulogama
- upravljanja s više povezanih organizacija ili portala s jednog mjesta
- upravljanje korisnicima
- stalan i kontinuiran razvoj novih inačica u skladu s razvojem web-standarda
- mogućnost korištenja za različite namjene, prilagođavanje krajnjem korisniku
- sigurnost

Također, primjena CMS-a može biti besplatna i komercijalna. Besplatna je namijenjena velikim i malim sustavima odnosno predviđena za manje korisnika te za povećanje razvoja samog okruženja, dok je komercijalna namijenjena velikim sustavima gdje je olakšan unos sadržaja za više korisnika te je veća mogućnost prilagodbe.

CMS-ovi koji se koriste u komercijalne svrhe su ColdFusion, Kentico, Sitefinity itd. Poznati CMS-ovi namijenjeni za besplatne svrhe su Wordpress, Drupal, Joomla, Symfony, Laravel. U ovom projektu korišten je Laravel koji je ujedno i PHP okvir.

3. TEHNOLOGIJE KORIŠTENE U IZRADI APLIKACIJE

3.1. HTML

HTML (eng. Hyper Text Markup Language) jezikom oblikuje se sadržaj stranice. Njegova zadaća jest uputiti internet preglednik za prikaz poveznice dokumenta. HTML dokument opisuje web-stranicu, sadrži oznake elemenata i tekst. Kod HTML-a moguće je unijeti bilo kojim programom za uređivanje teksta koji omogućava unos teksta bez filtriranja. Elementi dolaze najčešće u paru tj. sastoje se od početnog i završnog elementa. Također, potrebno je reći da dokument treba u imenu treba sadržavati nastavak .html kako bi internet preglednik prepoznao HTML-ov dokument.

3.2. CSS (Bootstrap)

CSS (eng. Cascading Style Sheets) je stilski jezik koji daje svojstva HTML-ovim elementima. Umetanje CSS svojstava moguće je na tri načina: vanjskim povezivanjem, unutarnjim povezivanjem ili pisanje svojstava unutar HTML elementa. U najširem smislu odnosi se na uređivanje izgleda i rasporeda web-stranice. Dokument CSS (vanjskim povezivanjem) treba sadržavati nastavak .css.

Bootstrap je okvir otvorenog koda koji služi za brže i lakše dizajniranje te stvaranje web-stranice. Jedna od važnijih stavki okvira je to što omogućava lagano korištenje predprocesa i responzivnost stranice. U njemu se nalaze gotove globalne postavke za HTML, CSS i JavaScript.

3.3. JavaScript (jQuery)

JavaScript je skriptni jezik koji omogućava kreiranje dinamičkih web-stranica. Takve stranice omogućavaju međudjelovanje s korisnikom, upravljanje internet preglednikom te dinamičko kreiranje web-stranice. Izvodi se na strani klijenta. Skripta je povezana internim i vanjskim putem te pisanjem događaja u HTML element. Vanjskim putem skripta je povezana HTML-ovim elementom *script*, a u zasebnom dokumentu s nastavkom .js se piše kod. Praksa je da se kod piše unutar *body* dijela HTML dokumenta zbog bržeg učitavanja stranice. Također, potrebno je reći da JavaScript sadrži biblioteke poput jQuery koji sam koristio u samom projektu.

jQuery je biblioteka koja omogućava da web-stranici budu dodane funkcionalnosti koje inače sa HTML-om i CSS-om ne bi mogli postići. Pomoću jQuery-a možemo pristupiti raznim dijelovima web-stranice kao i samom sadržaju, dodavati razne animacije te povući informacije bez potrebe za ponovnim učitavanjem stranice. Za pravilan rad jQuery-a potrebno je pozvati biblioteku koja je povezana s vanjskom datotekom ili preko interneta, kao što je navedeno na slici 3.

3.4. PHP

PHP (eng. Hypertext Preprocessor) je programski jezik orijentiran po C i Perl sintaksi. PHP je slobodni softver strane poslužitelja. Namijenjen je prvenstveno za programiranje dinamičkih web-stranica tj. radi se o skriptnom programskom jezik pomoću kojeg je moguće kreirati web-stranicu prije nego što je ona poslana klijentu i popunjena sadržajem. PHP je danas jedan od najnaprednijih i najkorištenijih skriptnih tehnologija od strane poslužitelja. Važno je i da je PHP bogat funkcijama za manipuliranje mnogo različitih tipova sadržaja tipa grafike, učitavanja modula i rada sa XML-om. Ono što PHP stavlja u prednost nad ostalim tehnologijama je neovisnost o operacijskom sustavu i podrška za upravljanje širokim opsegom baze podataka. Podržava sve popularnije baze podataka kao što su MySQL, SQLite, Oracle itd. Potrebno je reći da datoteka treba sadržavati nastavak u imenu .php.

3.5. MySQL

MySQL je standardni jezik za pristupanje bazi podataka. Podaci u MySQL-u spremljeni su u tablicama, tj. u skupinama prikupljenih podataka koje se sastoje od stupaca i redova. On nam omogućava dodavanje, unošenje i mijenjanje podataka koji se nalaze u toj bazi podataka. Kako bi pristupili podatcima potreban je server kojem u ovom slučaju pristupamo na sličan način kao i Internet serveru, s tom razlikom da obično pristupamo s unošenjem korisničkog imena i lozinke.

3.6. Laravel

Laravel je skup skripti (*framework*) koji omogućava izradu dinamičkih internet aplikacija. Laravel je okvir otvorenog koda, veoma moćan te omogućava alate koji su potrebni za veliku i složenu aplikaciju. Baziran je na PHP-u, tj. njegovoj sintaksi.

Velike značajke Laravel-a su to što se sastoji od:

- ORM-a (*Object Relational Mapping*) koji nam omogućava pretvorbu objekta u skupinu jednostavnih naredbi za SQL[2] tj. omogućava nam da izgradimo sloj apstrakcije između baze podataka i kontrolera
- Nasljeđivanja predložaka s kojim kreiramo glavni predložak od kojeg drugi podpredlošci mogu naslijediti svojstva (HTML, CSS, JS)
- Migracija koja nam omogućava mijenjanje sheme baze podataka u samom kodu.
- Composer-a koji nam služi za instaliranje dodataka i upravljanje povezanim dijelovima u Laravelu.

[2] SQL(eng. Structured Query Language) – služi za izradu, ažuriranje i brisanje podataka u bazi podataka

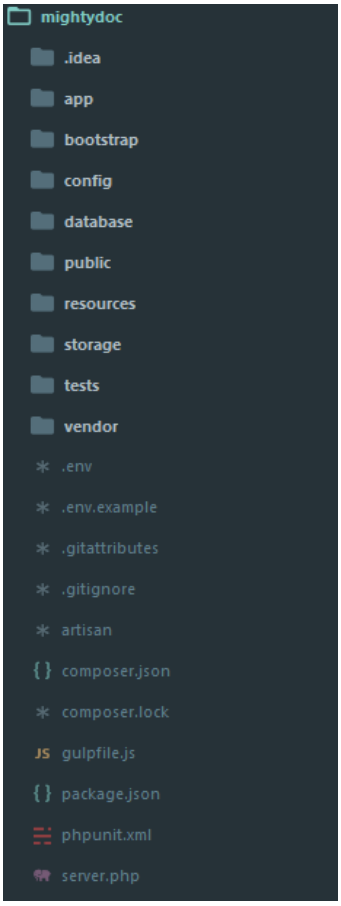
4. STRUKTURA I DIZAJN APLIKACIJE

4.1. Instalacija projekta

Za uspješno kreiranje projekta potrebno je skinuti Composer koji omogućava upravljanje PHP skriptama. Nakon uspješne instalacije Composer-a potrebno je otvoriti komandnu liniju (cmd.exe) koja dolazi s operacijskim sustavom Windows ili instalirati Git Bash koji je korišten u projektu. U Git Bash-u je potrebno prvo pristupiti direktoriju gdje se želi instalirati te potom upisati sljedeće:

„composer create-project laravel/laravel --prefer-dist MigtyDoc“

Ova linija koda će skinuti zadnju inačicu Laravela, otpakirati ju te prebaciti u projekt direktorija nazvan MigtyDoc. U programu za uređivanje teksta dobije se ovakva struktura projekta.



***Sl 4.1**. Struktura aplikacije*

Uvid u izgled projekta tj. pokretanja aplikacije u Internet pregledniku radi se pomoću naredbe u komandnoj liniji:

„php artisan serve“

Php artisan je alat sučelja komande linije koji dolazi s Laravelom te omogućava brzo generiranje velike količine koda. Pregled svih alata koji artisan koristi dobije se naredbom u komandnoj liniji:

„php artisan“.

4.2. Konfiguriranje razvojne okoline projekta

Razvojna okolina projekta konfigurira se u .env datoteci. U .env datoteci uspostavlja se veza sa lokalnim okruženjem, povezuje s *mightydoc* bazom podataka te postavlja korisničko ime i lozinku za pristup samoj bazi.

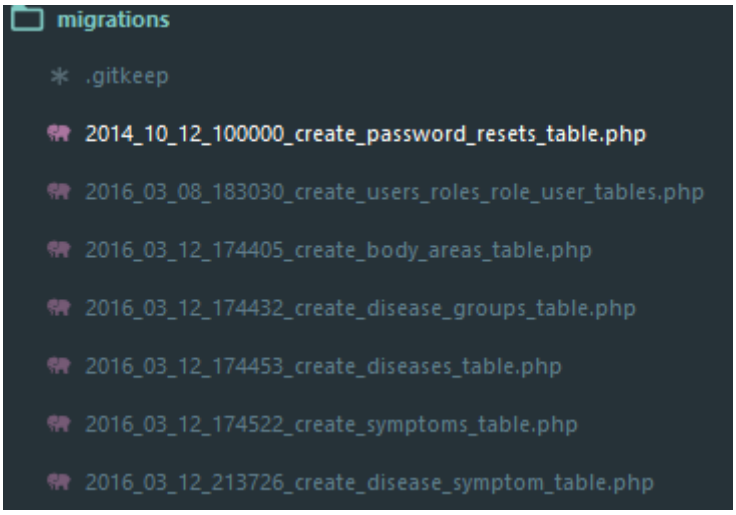
.env datoteka je povezana s database.php datotekom gdje je postavljena veza sa MySQL bazom podataka.

```
'mysql' => [
        'driver'    => 'mysql',
        'host'      => env('DB_HOST', 'localhost'),
        'database'  => env('DB_DATABASE', 'forge'),
        'username'  => env('DB_USERNAME', 'forge'),
        'password'  => env('DB_PASSWORD', ''),
        'charset'   => 'utf8',
        'collation' => 'utf8_unicode_ci',
        'prefix'    => '',
        'strict'    => false,
        'engine'    => null,
    ]
```

Slika 4.2. Povezivanje projekta s bazom podataka

4.3. Kreiranje migracija i modela

Nakon instaliranja projekta i postavljanja razvojne okoline projekta postavljen je model baze podataka pomoću migracija. Migracije su te koje omogućavaju željenu strukturu podataka tj. kasnije se u životnom ciklusu projekta mogu mijenjati postavke podataka u bazi. Laravel već dolazi sa integrirane dvije migracije koje služe za postavljanje korisnika i lozinki (password_resets_table). Osim te dvije integrirane migracije napravljene su migracije prema slici 4.3. pomoću kojih se upravlja podacima u bazi.



Sl. 4.3. Migracije

Migracije su napravljene pomoću naredbe u Git Bash-u:

„php artisan make:migrations create_symptom table –create=symptom“

Pomoću „—create=sypmtom“ dobije se dokument koji već ima ugrađene javne funkcije „up“ i „down“. Metoda „up“ služi za kreiranje sheme baze podataka tj. za dodavanje novih tablica, stupaca ili indeksa u bazi, uspostavljanja relacija s ostalim tablicama, dok „down“ metoda jednostavno invertira operaciju koja se izvodi od strane „up“ metode.

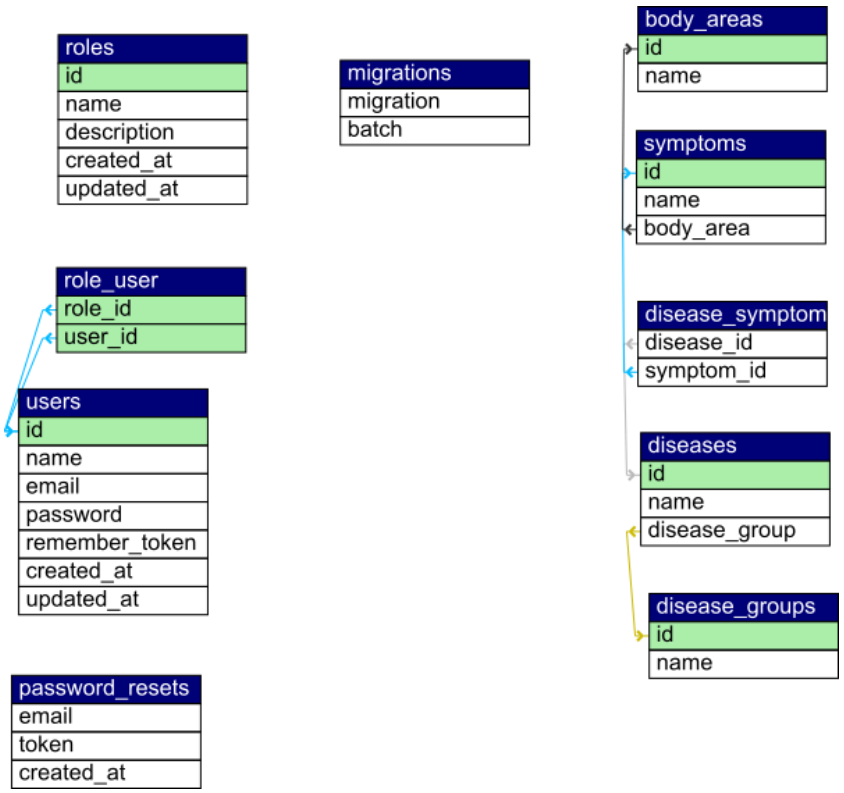
```
public function up()
    {
        Schema::create('symptoms', function (Blueprint $table) {
            $table->increments('id');
            $table->string('name');
            $table->integer('body_area')->unsigned();
            $table->foreign('body_area')->references('id')->on('body_areas');
        });
    }
```

Sl. 4.4. Metoda „up“ u datoteci „2016_03_12_174522_create_symptoms_table.php

Nakon ispunjavanja vrijednosti u „up“ i „down“ metodama, u komandnu liniju unosi se naredba koja omogućava normalan rad migracija:

„php artisan migrate“

koja isto tako omogućava vizualizaciju relacije i poveznice *mightydoc* baze podataka prema slici 4.5.



Sl. 4.5. Izgled *mightydoc* baze u MySQL-u

-

Nakon namještanja baze podataka i kreiranja svih migracija, izrađuju se modeli koji se nalaze u app direktoriju. Modeli rade sa tablicama iz baze podataka s kojom smo povezani. Kreiraju se modeli za uloge korisnika, simptome, bolesti, grupe bolesti i dijelove tijela pomoću naredbe u komandnoj liniji:

„php artisan make:model Symptom“

Jedna od stvari koju model već ima ugrađen je ta da pokušava odlučiti koja tablica u bazi podataka bi se mogla koristiti na temelju naziva same klase. U ovom slučaju unose se dodatni atributi koji pomažu samom usmjeravanju u točnu bazu podataka. Unutar simptom modela definiraju se odnosi s bolestima i dijelovima tijela pomoću javne funkcije s nazivima istoimenih modela s kojima trebaju biti povezani.

```
class Symptom extends Model
{
    public $timestamps = false;
    protected $fillable = ['name', 'body_area'];

    public function diseases()
    {
        return $this->belongsToMany(Disease::class, 'disease_symptom');
    }
    public function bodyArea()
    {
        return $this->belongsTo(BodyArea::class, 'body_area', 'id');
    }
}
```

Sl. 4.6. Model simptoma

Unutar javne funkcije „Disease“ koristi se sintaksa „belongsToMany“ što označavu relaciju na koji način je povezana s bazom tj. N:M relaciju, a unutar javne funkcije „bodyArea“ koristi se „belongsTo“ metoda koja označava 1:1 relaciju.

4.4. Kreiranje ruta i kontrolera

Nakon izrade modela i postavljanja veza (migracija), uspostavljene su rute tj. rečeno je aplikaciji kako se treba ponašati kada primi HTTP zahtjev. Rute se nalaze u direktoriju app/config pod nazivom routes.php. U datoteci se kreiraju sve rute koje su potrebne da bi aplikacija radila. Za izradu je potrebna skripta „Route::“ koja je integrirana u Laravel-u te nakon toga je potrebno reći koja metoda se želi koristiti. U rutu se stavljaju GET, POST ili PUT metode kako bi se dobili potrebni podaci. Napravi se grupa ruta u kojoj se postavljaju podrute za prijavu korisnika, simptome, bolesti, grupe bolesti i dijelove tijela prema slici 4.7.

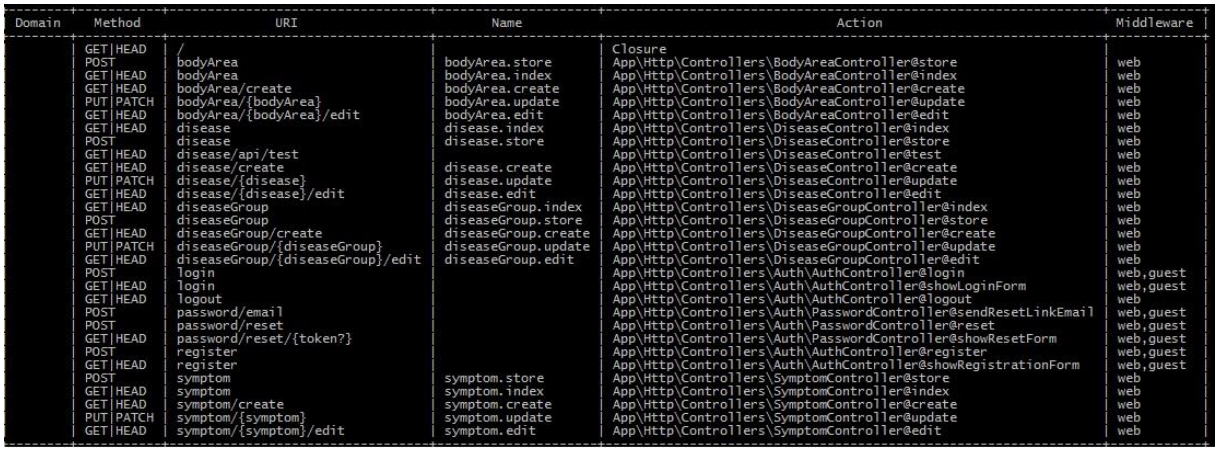
```
Route::group(['middleware' => 'web'], function () {
    Route::auth();

    /*napravi više ruta koje se potrebne za edit, update, create, odnosi se na route resoursces*/

    Route::resource('symptom', 'SymptomController', ['except' => ['show', 'destroy']]);
    Route::resource('diseaseGroup', 'DiseaseGroupController', ['except' => ['show', 'destroy']]);
    Route::resource('disease', 'DiseaseController', ['except' => ['show', 'destroy']]);
    Route::resource('bodyArea', 'BodyAreaController', ['except' => ['show', 'destroy']]);


});
```

Sl. 4.7. Uspostavljene rute u datoteci routes.php

Rute sa „resource“ metodom rade više ruta koje su potrebne za kreiranje, dodavanje i izmjenjivanje vrijednosti u aplikaciji.
Uvid u točne rute koje su napravljene se mogu vidjeti preko naredbe u Git Bash-u:

„php artisan route: list“

prema slici 4.8.



Domain	Method	URI	Name	Action	Middleware
	GET|HEAD	/		Closure	
	POST	bodyArea	bodyArea.store	App\Http\Controllers\BodyAreaController@store	web
	GET|HEAD	bodyArea	bodyArea.index	App\Http\Controllers\BodyAreaController@index	web
	GET|HEAD	bodyArea/create	bodyArea.create	App\Http\Controllers\BodyAreaController@create	web
	PUT|PATCH	bodyArea/{bodyArea}	bodyArea.update	App\Http\Controllers\BodyAreaController@update	web
	GET|HEAD	bodyArea/{bodyArea}/edit	bodyArea.edit	App\Http\Controllers\BodyAreaController@edit	web
	GET|HEAD	disease	disease.index	App\Http\Controllers\DiseaseController@index	web
	POST	disease	disease.store	App\Http\Controllers\DiseaseController@store	web
	GET|HEAD	disease/api/test		App\Http\Controllers\DiseaseController@test	web
	GET|HEAD	disease/create	disease.create	App\Http\Controllers\DiseaseController@create	web
	PUT|PATCH	disease/{disease}	disease.update	App\Http\Controllers\DiseaseController@update	web
	GET|HEAD	disease/{disease}/edit	disease.edit	App\Http\Controllers\DiseaseController@edit	web
	GET|HEAD	diseaseGroup	diseaseGroup.index	App\Http\Controllers\DiseaseGroupController@index	web
	POST	diseaseGroup	diseaseGroup.store	App\Http\Controllers\DiseaseGroupController@store	web
	GET|HEAD	diseaseGroup/create	diseaseGroup.create	App\Http\Controllers\DiseaseGroupController@create	web
	PUT|PATCH	diseaseGroup/{diseaseGroup}	diseaseGroup.update	App\Http\Controllers\DiseaseGroupController@update	web
	GET|HEAD	diseaseGroup/{diseaseGroup}/edit	diseaseGroup.edit	App\Http\Controllers\DiseaseGroupController@edit	web
	POST	login		App\Http\Controllers\Auth\AuthController@login	web,guest
	GET|HEAD	login		App\Http\Controllers\Auth\AuthController@showLoginForm	web,guest
	GET|HEAD	logout		App\Http\Controllers\Auth\AuthController@logout	web
	POST	password/email		App\Http\Controllers\Auth\PasswordController@sendResetLinkEmail	web,guest
	POST	password/reset		App\Http\Controllers\Auth\PasswordController@reset	web,guest
	GET|HEAD	password/reset/{token?}		App\Http\Controllers\Auth\PasswordController@showResetForm	web,guest
	POST	register		App\Http\Controllers\Auth\AuthController@register	web,guest
	GET|HEAD	register		App\Http\Controllers\Auth\AuthController@showRegistrationForm	web,guest
	POST	symptom	symptom.store	App\Http\Controllers\SymptomController@store	web
	GET|HEAD	symptom	symptom.index	App\Http\Controllers\SymptomController@index	web
	GET|HEAD	symptom/create	symptom.create	App\Http\Controllers\SymptomController@create	web
	PUT|PATCH	symptom/{symptom}	symptom.update	App\Http\Controllers\SymptomController@update	web
	GET|HEAD	symptom/{symptom}/edit	symptom.edit	App\Http\Controllers\SymptomController@edit	web

Sl. 4.8. Pregled svih ruta u Git Bash-u

Kontroleri dolaze sa već popunjenim metodama poput *index*, *create*, *store*, *show*, *edit*, *update* i *destroy*. Takve stvari omogućavaju brži i praktičniji rad. Umjesto definiranja ruta na route-logici u jednoj routes.php datoteci, organiziraju se takva ponašanja pomoću klasa kontrolera. Kontroleri čine grupno povezanu logiku unutar klase.

Index metoda predstavlja *GET* zahtjev prema izvoru bez ID-a izvora. *Store* metoda predstavlja *POST* metodu prema izvoru također bez pristupa ID-a izvora. *Update* je *PUT* metoda prema izvoru. Laravel izvodi svu tu logiku umjesto korisnika.

```
class SymptomController extends Controller
{
   /**
    * Display a listing of the resource.** @return \Illuminate\Http\Response*/
   public function index()
   {
      $symptoms = Symptom::all();
      return view('symptom.index', compact('symptoms'));
   }
   /*** Show the form for creating a new resource.** @return
\Illuminate\Http\Response*/
   public function create()
   {
       $bodyAreas = BodyArea::all();
```

```
        return view('symptom.create', compact('bodyAreas'));
    }
    /*** Store a newly created resource in storage.** @param
\App\Http\Requests\SymptomRequest  $request
     * @return \Illuminate\Http\Response
     */
    public function store(SymptomRequest $request)
    {
        Symptom::create($request->all());
        return redirect(route('symptom.index'));
    }
    public function getAllSymptoms()
    {
    $symptoms = App\Symptom::all();
    return $symptoms;
    }
}
```

Slika 4.9. Funkcije u kontroleru simptom

4.5. Čitanje podataka putem API-a

Nakon unosa grupe bolesti i simptoma u tablicu bolesti, rezultat u istoimenoj tablici prikazuje se u JSON formatu pomoću API-a na URL-u: localhost:8000/api/test. Kako bi se napravio API prvo se kreiraju rute gdje da se prikaže url routes.php dokumentu kao na sl.

```
Route::group(array('prefix' => 'api'), function() {
        Route::get('test', 'ApiController@index');
        });
```

Sl. 4.9. Prikaz API ruta u routes.php

Sljedeći korak je kreiranje API konotrolera u kojem se nalazi funkcija za vraćanje JSON-a. Api kontroler kreiran je pomoću naredbe:

php artisan make:controller ApiController

u komandoj liniji (Git Bash).

Funkcija za vraćanje JSON-a izgleda ovako:

```
class ApiController extends Controller
{
    public function index()
   {
      $diseases = Disease::with('diseaseGroup')->with('symptoms')->get();
      return json_encode($diseases);
   }
}
```

Sl. 4.10 funkcija za prikaz JSON formata

Na sljedećoj slici prikazan je pogled na JSON format.


```
{
    id: 11,
    name: "Upala srednjeg uha",
  - disease_group: {
        id: 4,
        name: "Bolesti uha i mastoidnih procesa"
    },
  - symptoms: [
      - {
            id: 26,
            name: "Bolnost",
            body_area: 6,
          - pivot: {
                disease_id: 11,
                symptom_id: 26
            }
        },
      - {
            id: 27,
            name: "Iscjedak",
            body_area: 6,
          - pivot: {
                disease_id: 11,
                symptom_id: 27
            }
        },
      - {
            id: 28,
            name: "Povišena temperatura",
            body_area: 6,
          - pivot: {
                disease_id: 11,
                symptom_id: 28
            }
        }
    ]
},
```


Sl 4.11. JSON format

4.6. Dizajn aplikacije

Dizajn i izgled aplikacije napravljen je pomoću Bootstrap okvira. Kako bi okviri radili u aplikaciji potrebno je uključiti datoteke (Bootstrap, Javascript) u gulpfile.js datoteci koja se nalazi u izvornom direktoriju. Za pravilan rad gulpfile.js dokumenta potrebno je instalirati Node.js [3]i Npm. U datoteci su uključene funkcije koje preusmjeravaju CSS i JavaScript u *public* direktorij.

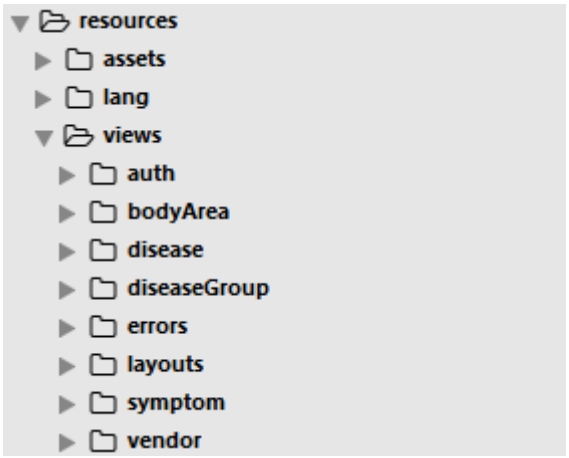
```
elixir(function(mix) {
    mix.styles([
        'bootstrap.min.css',
        '../font-awesome/css/font-awesome.min.css',
        'dataTables.bootstrap.min.css',
        'select2.min.css',
        'custom.css',
        'style1.css'
    ], 'public/css/back.css');

    mix.sass('app.scss')
        .version('css/app.css');

    mix.scripts([
        'jquery-2.2.0.min.js',
        'bootstrap.min.js',
        'jquery.dataTables.min.js',
        'dataTables.bootstrap.min.js',
        'select2.full.min.js'
    ]);

    mix.version(['js/all.js', 'css/back.css']);
```

[3] Node.js – alat potreban za pravilo izvođenje gulpfile.js

```
});
```

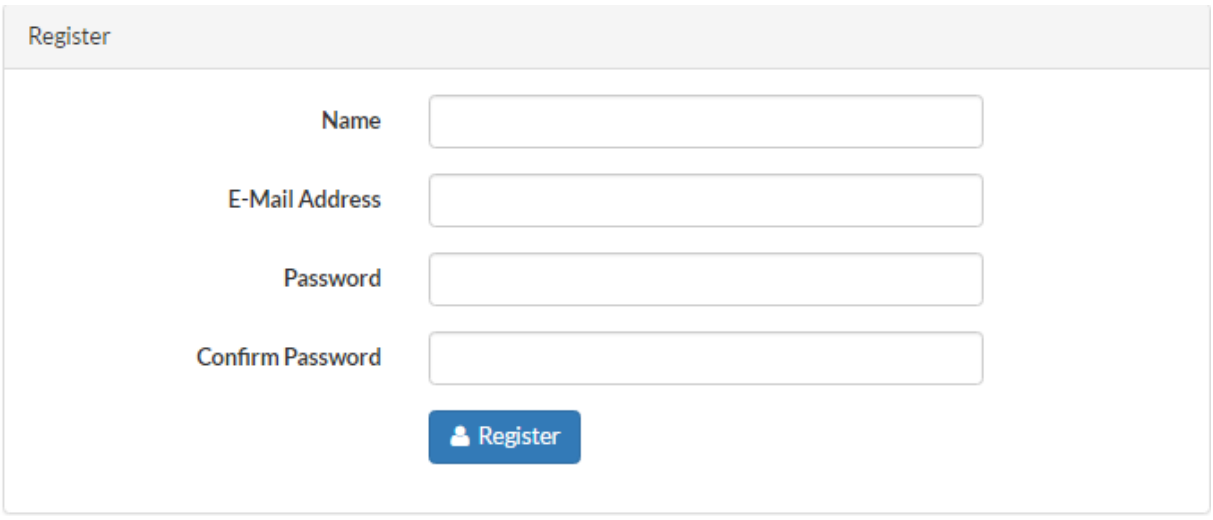
Sl. 4.12. funkcije u gulpfile.js datoteci

Pogledi u aplikaciji nalaze se u direktoriju resources/views te su podijeljeni na direktorije za bolesti *(diseaes*), grupe bolesti (disease groups), simptome (symptoms) i dijelove tijela (body area) te direktorij za autorizaciju korisnika.

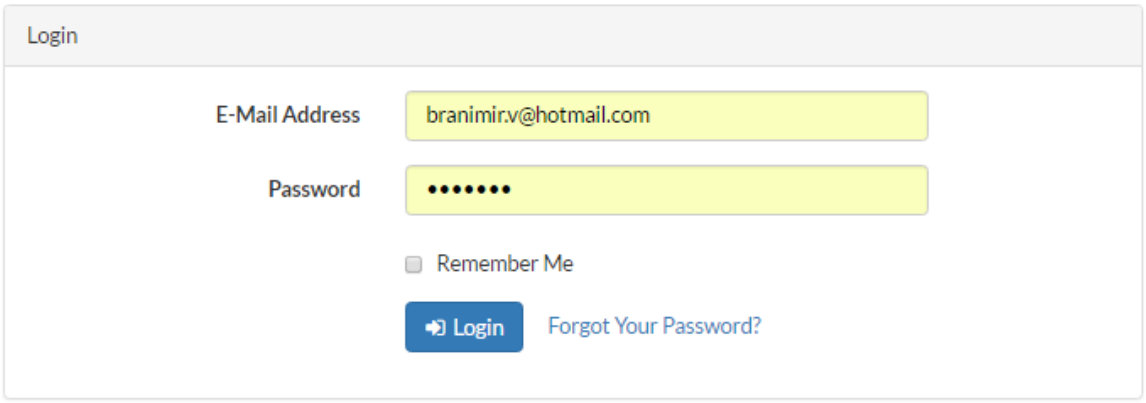


Sl. 4.13. Pregled pogleda u direktoriju *views*

Direktorij za autorizaciju stranice dolazi u standardnom paketu Laravel-a. Kako bi autoriziranje korisnika radilo pravilno, potrebno je pokrenuti migraciju. Autorizacija stranice ujedno je i početni pogled aplikacije. Autorizacija korisnika se nalazi na URL-u: localhost:8000/login za prijavu korisnika te localhost:8000/register za kreiranje korisnika.



Sl. 4.14. Registracija korisnika u aplikaciji



Sl. 4.15. Prijava korisnika u aplikaciji

Nakon uspješnog autoriziranja korisnika stranica navodi na sljedeći pogled tj. tablicu bolesti (rezultat unosa grupe bolesti, simptoma i dijelova tijela). Pogled tablice bolesti nalazi se na sljedećem URL-u: localhost:8000/disease.

Diseases

Show 10 entries Search:

ID	Disease name	Disease group	Symptoms	Edit
9	Konjuktivitis	Bolesti oka i adneksa	14 15 16 17 18 19	
10	Pareza n. Facijalisa (bellova kljenut)	Bolesti živčanog sustava	20 21 22 23 24 25	
11	Upala srednjeg uha	Bolesti uha i mastoidnih procesa	26 27 28	
12	Srčani udar	Bolesti krvožilnog sustava	29 31 32 33 34 35	
13	Infektivna mononukleoza	Bolesti oka i adneksa	36 37 38 39 40	
14	Plućna embolija	Bolesti krvožilnog sustava	29 41 42 43 44	
15	Zloćudni tumor jednjaka	Bolesti probavnog sustava	45 46 47 48 49 50	
16	Sindrom subakromijalnog sraza	Bolesti lokomotornog sustava	51 52 53	
17	Cistitis	Bolesti genitalno-urinarnog sustava	54 55 56 57	
18	Gonartroza	Bolesti lokomotornog sustava	58 59 60 61	

Showing 1 to 10 of 10 entries Previous 1 Next

Tab. 1. pogled na tablicu bolesti

U svakom pogledu nalazi se navigacija koja je stilizirana pomoću Bootstrap-a. Navigacija se sastoji od sljedećih veza: simptomi, bolesti, grupe bolesti i dijelovi tijela. Svaka od navedenih veza sadrži listu u kojoj se nalaze veze za izlistanje i kreiranje nastalih vrijednosti. Veza za izlistanje vrijednosti sadrže tablicu koja se nalazi u „index.blade.php“ datotekama. Tablica je responzivna i vrlo laka za pregledavanje, pretraživanje i sortiranje vrijednosti.

```
@extends('layouts.back')
@section('content')
   <div class="row">
      <div class="col-sm-12">
         <h2>Body Areas</h2>
         <div class="table-responsive">
            <table id="example" class="table table-striped table-bordered table-hover">
               <thead>
               <tr>
                  <th>ID</th>
                  <th>Body Area Name</th>
               </tr>
               </thead>
               <tbody>
               @foreach($bodyAreas as $bodyArea)
                  <tr>
                     <td>{{ $bodyArea->id }}</td>
                     <td>{{ $bodyArea->name }}</td>
                  </tr>
               @endforeach
               </tbody>
            </table>
         </div>
      </div>
      </div>
   <script>
      $('#example').DataTable();
   </script>
@stop
```

Sl. 4.15. Primjer koda za prikaz tablice dijelova tijela

Veza za kreiranje vrijednosti sadrži polja za unos vrijednosti i njihovo povezivanje gdje je potrebno, koja se nalazi u „create.blade.php“ datoteci.

```
@extends('layouts.back')


@section('content')
   <h2>New Body Area</h2>
   <form method="POST" action="{{ route('bodyArea.store') }}">
      {!! csrf_field() !!}

      <div class="form-group{{ $errors->has('name') ? ' has-error' : '' }}">
         <label for="exampleInputEmail1">Body Area Name</label>
         <input type="text" class="form-control" name="name" placeholder="Body area name">
         @if ($errors->has('name'))
            <span class="help-block">
               <strong>{{ $errors->first('name') }}</strong>
            </span>
         @endif
      </div>

      <button type="submit" class="btn btn-default">Submit</button>
   </form>
@stop
```

Sl. 4.16. Primjer koda za kreiranje dijelova tijela

5. ZAKLJUČAK

Laravel je definitivno vrlo moćan i bogat skup skripti koji mnogo obećava te pruža pristup širokom rasponu korisnih biblioteka i alata koji se mogu koristiti za izradu bogatih aplikacija. Jedna od njegovih glavnih prednosti je što prati rad MVC strukture tj. *model*(baza podataka i logika)-*view*(HTML,CSS)-*controller*(korisnički zahtjevi).

U sklopu rada je napravljena Web aplikacija pomoću koje se liječniku omogućava lakši unos simptoma, bolesti, popis simptoma za svaku bolest, te radi diferencijalna dijagnozu stanja ljudskog organizma na temelju primljenih simptoma iz PocDoc aplikacije i vraća rezultat aplikaciji preko aplikacijskog programskog sučelja (API) u JSON formatu.

LITERATURA

[1] HTML5 , stranica http://www.w3schools.com/html/default.asp (pristup: 20.5.2015.)

[2] CSS, stranica https://developer.mozilla.org/en-US/docs/Web/CSS (pristup: 20.6.2016.)

[3] PHP, stranica http://www.w3schools.com/php/default.asp (pristup: 3.6.2016.)

[4] SQL, stranica http://www.w3schools.com/sql/default.asp (pristup: 4.6.2016.)

[5] JavaScript(jQuery), stranica http://www.w3schools.com/js/default.asp (pristup:15.6.2016.)

[6] Laravel, stranica https://laravel.com/docs/5.2 (pristup 20.6.2016.)

[7] Izvor slike 2.1., Rad CMS-a, stranica http://www.opendesigns.org/od/wp-content/uploads/2012/07/how-cms-works.jpg

SAŽETAK

Ovaj rad predstavlja projekt izrade web aplikacije koja omogućava liječniku lakši unos simptoma, bolesti, popis simptoma za svaku bolest. Cilj izrade aplikacije je steći praktična znanja iz informacijskih sustava. Projekt je izrađen uz pomoć programskog alata Sublime Text Editor. Za izradu aplikacije korišten je Laravel skup skripti baziran na PHP-u, HTML, CSS, JavaScript i MySQL koji omogućavaju diferencijalnu dijagnozu stanja ljudskog organizma na temelju primljenih simptoma iz PocDoc aplikacije te vratiti rezultat aplikaciji preko aplikacijskog programskog sustava u JSON formatu.

Ključne riječi: Laravel, web, diferencijalna dijagnoza, baza simptoma

ABSTRACT

WEB CONTENT MANAGEMENT SYSTEM FOR POCDOC APPLICATION

This paper presents a project of developing a Web application that allows the doctor easier entry of symptoms, diseases, list of symptoms for each disease. The aim of the application is to acquire practical knowledge of information systems. The project was created with the help of software tools Sublime Text Editor. In the process of developing application is used Laravel framework based on PHP, HTML, CSS, JavaScript and MySQL, which enables differential diagnosis of human disease states based on the likely symptoms of PocDoc application and return the result of the application via an application programming system (API) in JSON format.

ŽIVOTOPIS

Branimir Valentin rođen je 12.06.1992. godine u Đakovu, Republika Hrvatska. Osnovnu i srednju školu pohađao je u Đakovu , osnovnu od 1999. – 2007. godine , te srednju ekonomsku školu u razdoblju od 2007. - 2011. godine i stekao zvanje Ekonomista. Godine 2012. upisao se na stručni studij Elektrotehničkog fakulteta u Osijeku (smjer: Informatika).

_____

Branimir Valentin