Izrada aplikacije za oglasnik ponude i potražnje proizvoda

Papratović, Nikola

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU

FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA

Stručni studij

IZRADA APLIKACIJE ZA OGLASNIK PONUDE I POTRAŽNJE PROIZVODA

Završni rad

Nikola Papratović

Osijek, 2016.

Sadržaj

1. UVOD

U ovom radu obrađena je izrada jednostavne aplikacije za ponudu i potražnju proizvoda i usluga. Aplikacija je sinergija 2 dijela – mobilne i web aplikacije.

Web dio aplikacije se sastoji od internet stranice koja služi kao kataloški prikaz ponude i potražnje oglasa. Stranica ima funkcionalnosti tražilice, kategorizacije i pregleda pojedinog oglasa. Cijela internet stranica se oslanja na administracijsko sučelje u kojem ovisno o tzv. korisničkim ulogama korisnicima omogućava predaju i upravljanje njihovih oglasa.

Mobilna aplikacija se oslanja na stranicu, jer dohvaća ponudu oglasa i omogućava prikaz unutar aplikacije. Mobilna aplikacija je po načinu izrade hibridna mobilna aplikacija i može raditi na Android platformi.

Prilikom izrade oglasnika korišteno je nekoliko tehnologija i programskih jezika. Osnovna struktura aplikacije je MVC okvir (engl. *Model-View-Controller framework*). Za web aplikaciju je korišten Laravel kao PHP okvir koji omogućava brz razvoj, a mobilna aplikacija je bazirana na Ionic okviru. Najviše zastupljeni programski jezici su PHP, HTML, CSS, i JavaScript.

Rad se sastoji od 2 glavna dijela. U prvom dijelu su opisane teoretske osnove koje je potrebno poznavati prije izrade aplikacije. U teoretskom dijelu su opisane glavne tehnologije i programski jezici za web i mobilni dio aplikacije.

U drugom dijelu, koji je praktični dio, je opisana izrada aplikacije. Posebno je odvojen dio koji se odnosi na poslužiteljski dio aplikacije od klijentskog dijela aplikacije. Osnovna ideja je pokazati kako se može brzo razviti oglasnik korištenjem Laravel MVC okvira. Mobilna aplikacija je razvijena na Ionic okviru. Od svih dostupnih okvira odabrani su Laravel i Ionic zbog svojih visokih performansi, velike zajednice korisnika i jednostavnosti korištenja.

Završni rad završava zaključkom te najavom mogućih poboljšanja oglasnika i dodatnih funkcionalnosti.

2. MVC (*engl. Model-View-Controller*)[1]

2.1. Povijest i razvoj

MVC je okvir softverske arhitekture koji služi za implementaciju korisničkog sučelja na računalima. U svojoj srži, MVC dijeli računalni program na tri dijela koja su međusobno povezana – podatak, prezentaciju tog podatka i logiku obrade podatka. Iako je u početku MVC korišten za grafičko sučelje kod stolnih računala, s vremenom daleko veću primjenu ima kod razvoja web aplikacija.

MVC je nastao ranih 1970ih i autorom se smatra Trygve Reenskaug. Prva implementacija je bila u programskom jeziku „Smalltalk-76“, dok je bio u „Xerox Paolo Alto“ centru za istraživanje, (*engl. Xerox Paolo Alto Research Center - PARC*). Tokom slijedećih desetljeća, MVC je evoluirao i imao je nekoliko različitih varijanti:

- HMVC – (*engl. Hierarchical model-view-controller*) – karakterizira ga puno složenija struktura, gdje se jedna veća aplikacija razdvaja na manje tzv. module i svaki ima unutrašnju strukturu složenu prema pravilima MVC-a. Najbolji primjer za HMVC su sustavi za upravljanjem sadržaja (*engl. CMS – Content Managment Systems*) na mrežnim stranicama. Oni implementiraju HMVC na način da pojedinu funkcionalnost kao što su mogućnost pisanja članaka ili komentara razdvoje u zaseban modul. Tako korisnik ne primijeti da mu se stranica koju mu preglednik prikazuje sastoji od manjih dijelova, i svaki taj dio je napravio poseban modul. U HMVC arhitekturi moduli su neovisni jedan od drugih, i zasebno funkcioniraju.
- MVA – (*engl. Model-view-adapter*) – karakterizira ga činjenica da su model (podaci) i pregled (izgled koji korisnik vidi), strogo odvojeni i jedini način da međusobno komuniciraju je preko središnjeg člana (kontrolera).
- MVP – (*engl. Model-view-presenter*) – to je okvir koji se koristi kod dizajna korisničkog sučelja. *Presenter* je u ovom slučaju poveznica između podatka i pregleda. *Presenter* ima mogućnost da ažurira pregled na osnovu promjene stanja podatka, kao i obrnuto – mijenja podataka na osnovu korisničke akcije. Primjerice, ako je korisnik odabrao neku vrijednost iz padajućeg izbornika, onda će se i vrijednost podatka promijeniti.

[1] MVC dolazi od kratice Model-View-Controller i odnosi se na arhitekturu okvira koji se sastoji od 3 glavna dijela: podaci, pregled podataka i upravljač (*engl. Controller*)

- MVVM – (*engl. Model-view-view-model*) – razdvaja programiranje grafičkog korisničkog sučelja od programiranja logike podataka. Nastao je kako bi pojednostavio programiranje korisničkog sučelja koje ovisi o akcijama korisnika.

Iako je MVC apstraktan prikaz arhitekture softvera, Trygve Reenskaug ga je 1979. definirao na slijedeći način:

1. Model

Model predstavlja podatke. Model može biti jedan objekt ili može biti struktura objekata. Trebala bi postojati veza 1-na-1 između modela sa jedne strane i svijeta kojeg vidi korisnik sa druge strane. Veza između modela i vanjskog svijeta je nekakav kontroler koji je odgovoran za komunikaciju između ta dva dijela sustava.

2. Pogled

Pogled je vizualna prezentacija modela. Sam po sebi, pogled naglasi određene atribute modela, i isto tako, neke atribute potisne. Pogled se na taj način ponaša kao filtar. Pogled je vezan za model (ili dio modela), i dobije podatke koji su mu potrebni za prezentaciju postavljajući pitanja. Pogled može čak i ažurirati model slanjem odgovarajuće poruke. Svako pitanje i poruka koji se prenose model mora razumjeti, i pogled mora znati atribute modela kojeg predstavlja. Atributi su vrijednosti koje opisuju podatke. Drugim riječima, ukoliko pogled dostavlja korisniku informacije o nekom proizvodu, atributi koji opisuju informacije o tom proizvodu su naziv proizvoda, kratki opis, oznake proizvoda i slično.

3. Kontroler

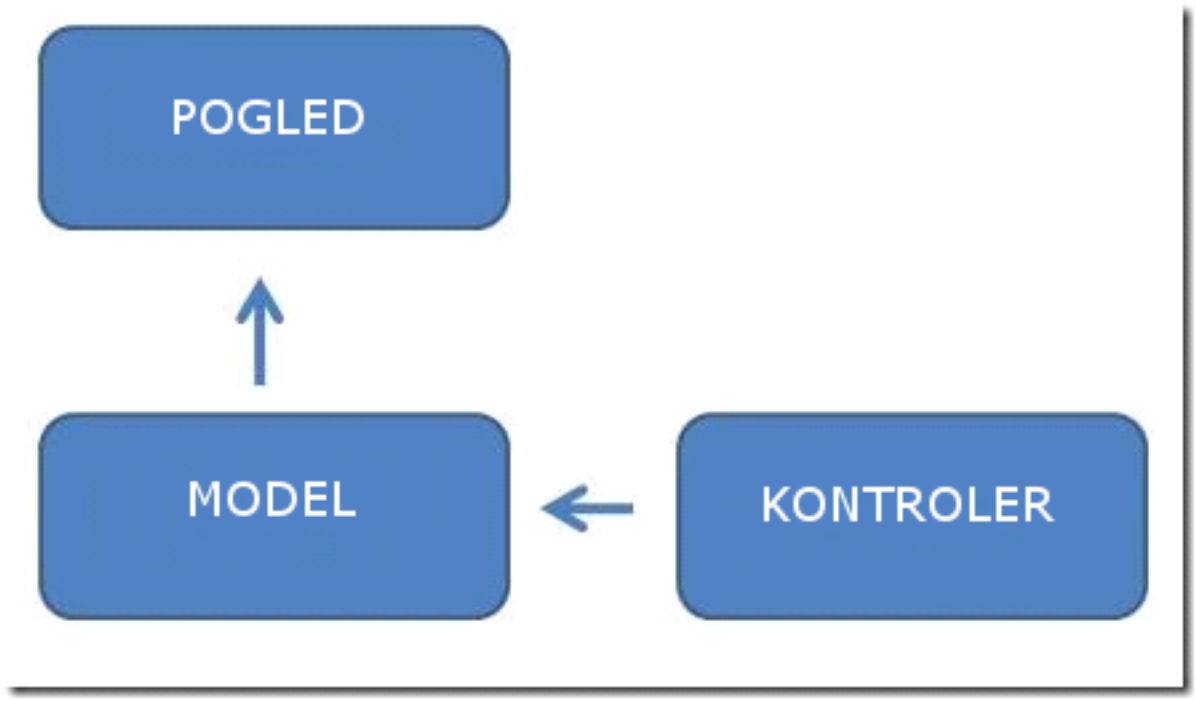
   Kontroler je poveznica između korisnika i podataka. Kontroler pruža korisniku odgovarajući prikaz podataka na njegovom zaslonu, ovisno o zatraženim informacijama. Te informacije je kontroler prikupio od korisnika i na osnovu njih mu prikazuje podatke koje je korisnik zatražio.

MVC kao okvir softverske arhitekture je nastao davno prije internet preglednika, i u početku je korišten kao obrazac za kreiranje grafičkog korisničkog sučelja. U početku MVC obrascu se nije pristupalo kao danas. MVC je isto imao 3 dijela – model, pogled i kontroler, ali u strukturi kontroler nije bio između pogleda i modela (slika 2.1.) Tada se radilo na slijedeći način:

MODEL: podatak koji aplikacija prikazuje. Primjerice, očitavanje temperature.

POGLED – jedna od nekoliko mogućnosti izgleda podatka iz modela. Jedan podatak može imati nekoliko pogleda vezanih uz sebe. Primjerice, očitanje temperature možemo prezentirati visinom stupca, ili oznakom temperature.

KONTROLER – skuplja akcije koje korisnik napravi i modificira model. Kontroler tako primjerice može skupljati akcije koje korisnik napravi mišem, i onda izmjenjuje model.



Slika. 2.1. Model 1 MVC obrazac

Prema Modelu 1 pogled se ažurira direktno na osnovu izmjena u modelu. Kada se model izmjeni, on pokrene akciju koja mijenja pogled. Isto tako, kontroler ne upravlja sa

om, nego modificira model, kako pogled promatra samo promjene u modelu, pogled se ažurira tek nakon tih promjena.

Krajem 1999. tvrtka Sun Microsystems je objavio tehnologiju koja je omogućila programerima da kreiraju dinamičke internet stranice bazirane na HTML, XML ili nekom drugom programskom jeziku. Ta se tehnologija zove Java Server Pages (JSP), i slična je programskim jezicima PHP i ASP, no koristi Java programski jezik.

U specifikaciji JSP, navedena su dva modela za isporuku internet stranica – Model 1, poznat od prije, i Model 2 (slika 2.2.), koji se tek pojavio. Model 2 je imao drugačiji pristup za preglednike koji su prikazivali internet stranice. U tom pristupu, zahtjev koji preglednik šalje nisu izvršavali same stranice, nego servisi koji se skupno zovu *Java Servlet*. *Servlet* generira rezultat i sprema rezultat u komponentu (slično kao među memorija). *Servlet* tada poziva Java Server Page datoteku, koja pristupa komponenti i prikazuje dinamični sadržaj u pregledniku.

Model 2 je počeo koristiti MVC okvir, i danas kada se govori o kontekstu MVC-a kod web aplikacija, misli se na Model 2 verzije MVC-a.

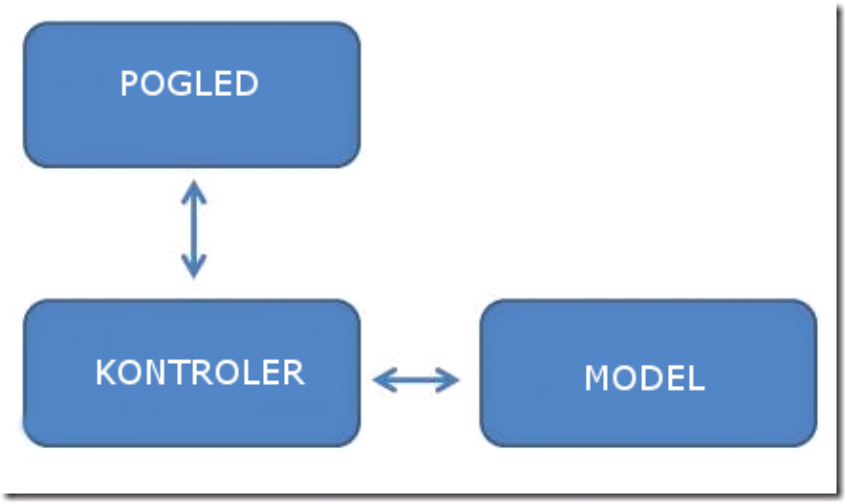
Model 2 je značajan napredak u odnosu na Modela 1. MVC struktura je oblikovana na slijedeći način:

MODEL: Upravlja podacima na način da ih kreira, čita, ažurira i briše iz jednog ili nekoliko izvora kao što su primjerice relacijske baze podataka (*engl. CRUD: create, read, update, destroy*).

POGLED: Korisničko sučelje koje prikazuje informacije o modelu korisniku.

KONTROLER: Mehanizam kontrole toka putem kojeg korisnik upravlja aplikacijom.

U ovoj novoj verziji MVC ne postoji više veza između prikaza podataka i modela. Sva komunikacija ide kroz kontroler.



Slika 2.2. Model 2 MVC obrazac

Preporuka je bila da ukoliko se radi kompleksna aplikacija za koju će biti potrebno dulje održavanje i dodavanje novih funkcionalnosti, da se koristi Model 2.

Danas je MVC kod programskih jezika jedan od standardnih okvira koji se koriste. Pa tako i PHP koristi MVC obrazac. Postoji jako puno okvira koji su implementirali MVC arhitekturu. Neki od njih su: „Laravel“, „CodeIgniter“, „Yii“, „CakePHP“. Okvir *(engl. framework)* je skup klasa koje omogućavaju programeru da puno brže razvije aplikaciju, i da se bazira na logiku aplikacije, a ne toliko na razvoj pomoćnih funkcionalnosti.

2.2. Slojevi MVC arhitekture

Glavna ideja iza MVC arhitekture je jednostavna: mora se omogućiti da je slijedeća podjela jasno odvojena u aplikaciji:

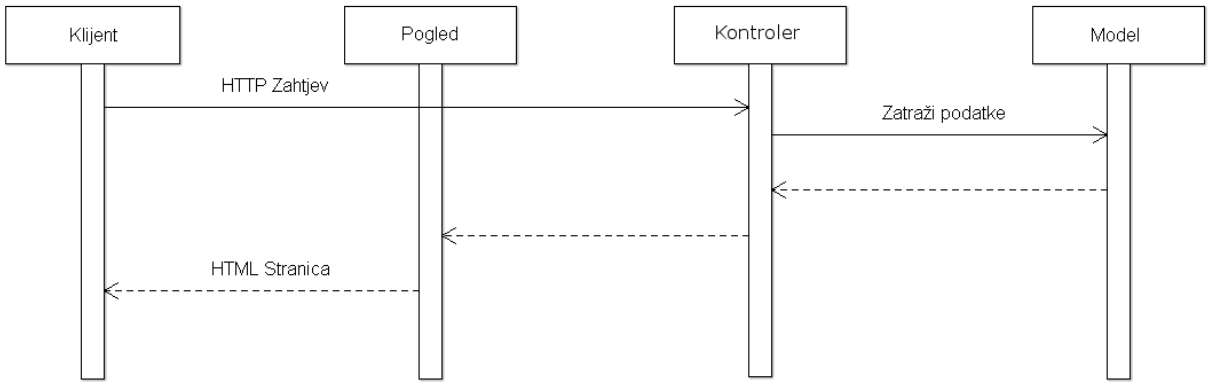
MODEL: upravlja sa podacima

POGLED: prezentira podatke korisniku

KONTROLER: prima zahtjeve korisnika i poziva metode na osnovu njih

Aplikacija je podijeljena u 3 glavne komponente, i svaka od te tri komponente je zadužena za specifične zadatke. MVC arhitektura omogućava da se jednom napisani kod uz minimalne ili čak nikakve izmjene može koristiti u više različitih projekata. Isto tako, MVC omogućava da se sustav podjeli u više međusobno neovisnih cjelina.

Upravo ta podjela je i glavna svrha MVC arhitekture – da se razdvoji poslovna logika od korisničkog sučelja. Svaki dio aplikacije koja koristi MVC arhitekturu ima svoju zadaću koju mora izvršiti. Glavne komponente MVC strukture su: model, pogled i kontroler (slika 2.3.)



Slika 2.3. Komponente MVC strukture

Na slici 2.3 može se vidjeti pojednostavljeni prikaz i redoslijed akcija koje se odvijaju na nekoj web aplikaciji. Iako je ovo samo šturi prikaz i ima puno više koraka, ovdje možemo u osnovnim crtama opisati što se događa:

1. Korisnik zatraži informaciju – primjerice naslov knjige sa određenim ISSN
2. Kontroler tada šalje zahtjev prema modelu za tim podatkom
3. Model radi dio sa poslovnom logikom i traži podatak iz baze
4. Model vraća podatak nazad u kontroler
5. Kontroler traži odgovarajuću datoteku koja će poslužiti kao pogled
6. Odabrana je datoteka i u njoj će se prikazati informacije
7. Korisnik je dobio informaciju na ekran

2.2.1. Model

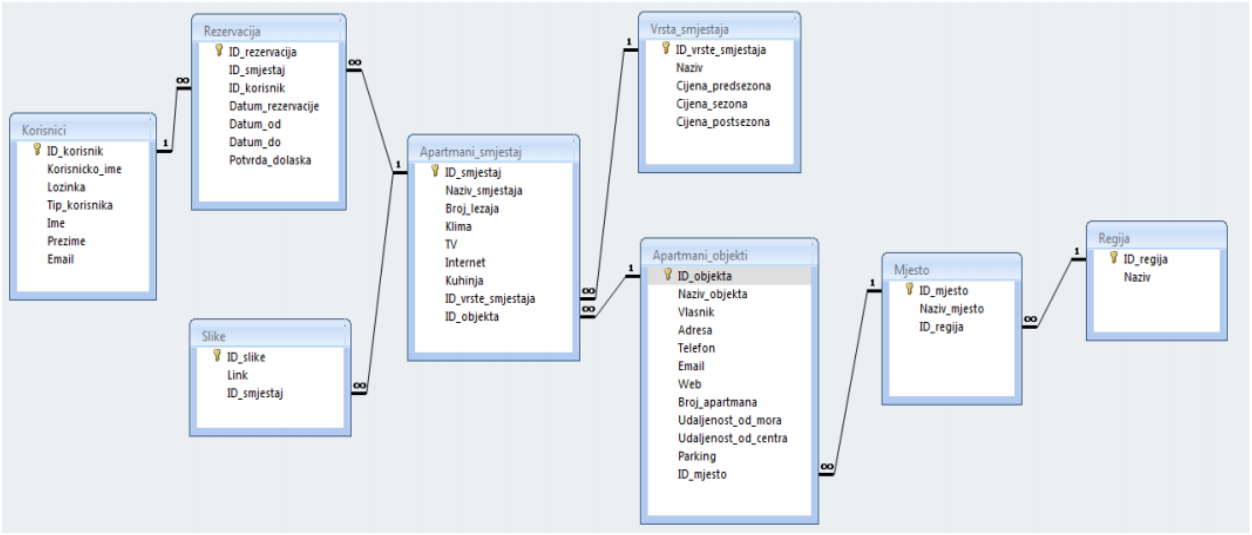
Kod MVC okvira, model se pojavljuje kao najvažniji sloj. U njemu se nalazi poslovna logika, i sadrži zapise koji se pohranjuju u bazi podataka. Model upravlja temeljnim ponašanjem i podacima unutar aplikacije. On može odgovoriti na upit za informacijom, može odgovoriti na upute za promjenu stanja informacije kada se zatraži ta promjena. Isto tako, model čak može i obavijestiti

neku metodu koja se ponaša kao promatrač (*engl. observer*) te obavijestiti kontroler koji će dalje poduzeti određene akcije.

Model može biti primjerice baza podataka ili bilo koja druga struktura podataka. Dakle, model su podaci i upravljanje podacima u okviru aplikacije. Model je također odgovoran za upravljanje sa podacima i informacijama – i to dvije vrste: unutar aplikacije (zapisi o izvršavanju određenih akcija) i izvan aplikacije (podaci koje klijent traži / želi sačuvati).

Model također pruža unutrašnji interfejs (*engl. interface*) – API (*engl. Application programming interface*) koji omogućava da ostali dijelovi aplikacije komuniciraju sa njime.

Integritet podataka je također odgovornost modela. Model odgovara na zahtjeve koji stižu iz kontroler objekta, priprema i obrađuje podatke. Bitna odlika kod modela za razliku od ostatka MVC arhitekture je to što model nema izravnu vezu sa vanjskim svijetom.



Slika 2.5. Relacijski model baze podataka

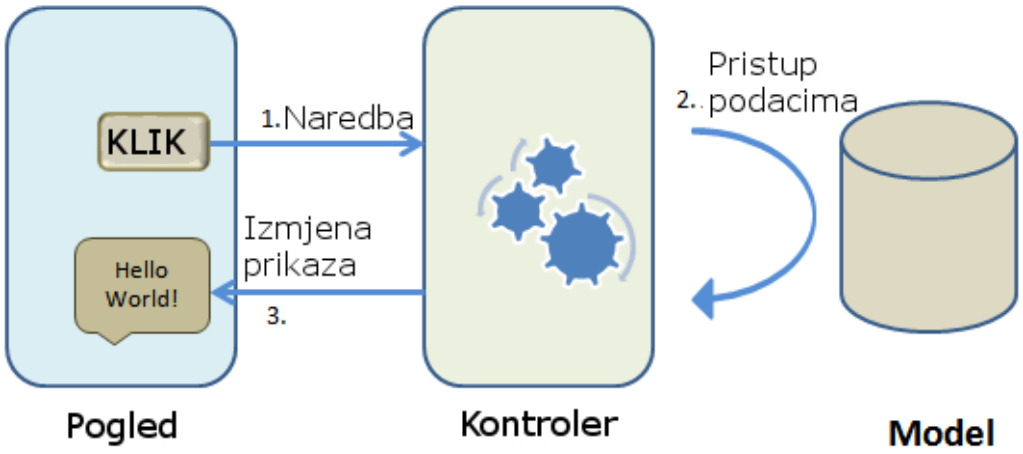
Kod web aplikacija koje su napravljene po MVC arhitekturi, modele se često sprema u bazu podataka – primjerice MySQL bazu. (slika 2.5.)

2.2.2. Pogled

Pogled pruža različite načine da se podaci koju su primljeni iz modela prikažu. Ovo mogu primjerice biti predlošci (*engl. templates*) u koje se podaci popunjavaju. U većini aplikacija najčešće postoji nekoliko pogleda i kontroler odlučuje koji će uzeti za prikaz podataka.

2.2.3. Kontroler

Kontroler skuplja korisničke zahtjeve (prima ih kao POST – ako nešto šaljemo ili GET – ako nešto zatražimo), kada korisnik klikne na neki element kako bi pokrenuo određenu akciju. Glavna mu je funkcionalnost da pozove i koordinira neophodne resurse ili objekte kako bi pokrenuo određenu akciju. Kontroler služi da pozove podatke iz određenog modela i prikaže ih na određenom pogledu (slika 2.6.)



Slika 2.6. Prikaz tijeka akcije nakon korisničkog zahtjeva

2.3. Prednosti i nedostatci MVC arhitekture

Glavna prednost za korištenje MVC arhitekture kod razvoja aplikacije u odnosu na razvoj bez korištenje MVC okvira je što se programski kod odvaja u smislene i odvojene cjeline. Iz pravilnog strukturiranja koda proizlazi i lagane izmjene, nadogradnje i budući razvoj svakog projekta.

U situaciji kada moramo mijenjati samo jednu funkcionalnost, bez da utičemo na ostatak aplikacije, MVC arhitektura se pokazala kao jako zahvalna.

Odvajanje u smislene i logičke cjeline i sama kompleksnost MVC arhitekture je i njen najveći nedostatak. Naime, kod manjih projekata implementacija takvog obrasca je dugotrajan proces, koji se ne pokazuje kao isplativ.

Isto tako, kako je sustav kompleksan, potrebna je naprednija tehnička podrška koja će raditi na održavanju tog sustava.

2.4 Laravel

Laravel je besplatni okvir otvorenog koda. Autor je Taylor Otwell, i namijenjen za izradu web aplikacija koje slijede MVC arhitekturu. Laravel je nastao kao pokušaj naprednije alternative u odnosu na tada popularni CodeIgniter okvir, koji nije pružao određene funkcionalnosti odmah po instalaciji (*engl. out-of-box*).

Glavne funkcionalnosti su bile autentifikacija (provjera da li je korisnik prijavljen u sustav), i autorizacija (ovlasti koje korisnik ima unutar aplikacije).

Prva verzija Laravela je bila u *beta* verziji, zvala se Laravel 1, i javno je bila dostupna od 9. lipnja 2011, a prva stabilna verzija je uslijedila mjesec dana kasnije. Laravel je već u svojoj prvoj inačici imao implementirane ključne funkcionalnosti: lokalizaciju, modele, poglede, sesije, rute, međutim, nije imao podršku za kontrolere, što ga tada još uvijek nije činilo pravim MVC okvirom.

Nakon verzije Laravel 1, uskoro su izašle još 3 veće nadogradnje (*engl. major release*). Te verzije su imale dodane funkcionalnosti koje su Laravel pretvorile u okvir sa velikim brojem mogućnosti primjene. U Laravel su implementirani „*Blade template engine"* – sustav zamjenskih varijabli koje koristimo da bi relativno nerazumljiv PHP kod zamijenili sa naredbama koje ljudi lakše razumiju.

Blade koristi tzv. *template tagove* – to je skup zamjenskih ključnih riječi iz PHP-a.

Primjerice:

Blade sintaksa	PHP sintaksa
`Hello, {{ $name }}.`	`Hello, <?php echo $name; ?>.`

U verziji 3, u Laravel je implementiran komandno linijski sustav Artisan – (*engl. CLI – command-line-interface*), koji je omogućio da kroz par naredbi u konzoli kreiramo i popunimo bazu sa tablicama i podacima.

Najveću promjenu Laravel je doživio u verziji 4. Ta verzija je imala kodno ime Illuminati, i za Laravel je značila da se napisao iz početka – drugim riječima veliki dio originalnog koda je prepisan i napisan iz početka, sa drugim standardima na umu. U toj verziji je dodana jedna značajna funkcionalnost koja je svojevrsna mini revolucija. Naime, implementiran je Composer – to je naziv za sustav „paketa“ koji se lagano mogu dodati u aplikaciju, i tako joj proširiti funkcionalnost.

Danas je aktualna verzija Laravel 5, za koji se od početka radilo da bude LTS (*engl. Long Term support*), odnosno drugim riječima, podrška će biti sigurnija i na dulji vremenski period. Zadnja pod verzija koja je stabilna i koristi se je verzija 5.1. i to je prva verzija koja ima pravu LTS podršku – za nju se očekuje da će imati ispravke grešaka (*engl. bug fixes*) i sigurnosne nadogradnje (*engl. security patches*) tokom slijedeće dvije odnosno tri godine. Za LTS verzije se planira da se nova verzija objavljuje svake 2 godine.

Za Laravel se može reći da je potpuno opremljen okvir sa setom funkcionalnosti koje omogućuju da se naprave jako komplekse aplikacije. Ako se poštuju načela MVC-a, i PHP standarda, aplikacija koja se radi korištenjem Laravel okvira može biti brzo napravljena. Također, održavanje te aplikacije u smislu dodavanje novih i proširivanje postojećih mogućnosti je lagano.

S obzirom da omogućuje modularnu strukturu odmah po instalaciji, jedna Laravel aplikacija se može razviti unutar jednog tima. I upravo ta suradnja između više programera na istoj aplikaciji je činila veliki napredak i učinila je MVC kao arhitekturu kod izrade aplikacija jako popularnim.

Popularnost Laravel duguje nekoliko faktora, izdvojio bih samo neke:

1. Velik broj mogućnosti odmah po instalaciji
2. Kvalitetna dokumentacija
3. Lagana upotreba već napisano koda (*engl. re-use*)
4. Composer – sustav malih paketa (modula)
5. Velika zajednica programera koja ga koristi

2.5. Arhitektura Laravel aplikacije

Za Laravel se kaže da je „*full-stack*“ okvir, zato što ima sve potrebne predispozicije da upravlja sa svim aspektima web aplikacije – od isporuke podataka, upravljanje bazom, sve do generiranje HTML dokumenata.

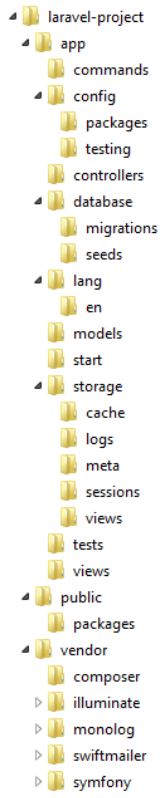
Laravel dolazi sa komandno-linijskim alatom Artisan koji omogućuje da se kroz par kratkih naredbi generira kostur aplikacije i izradi baza sa svim potrebnim tablicama. Artisan također i ima mogućnosti da izrađuje tzv. migracije i konfiguraciju aplikacije.

Primjer korištenja Artisana je slijedeći:

```
php artisan make:model User
```

Ta jednostavna naredna kreira model pod imenom „User“, i kreirao sve potrebne metode za CRUD (*engl. Create, Read, Update, Destroy*) postupke i pripadajuće komentare.

Za razliku od ostalih okvira, Laravel zapravo ima nekoliko ograničenja za strukturu (slika 2.7.) web aplikacije, međutim, iako su to načelno ograničenja (zapravo skup pravila po kojima se struktura aplikacije organizira), ona omogućuje da razvoj aplikacija bude olakšan.



Slika 2.7. Struktura tipične Laravel aplikacije

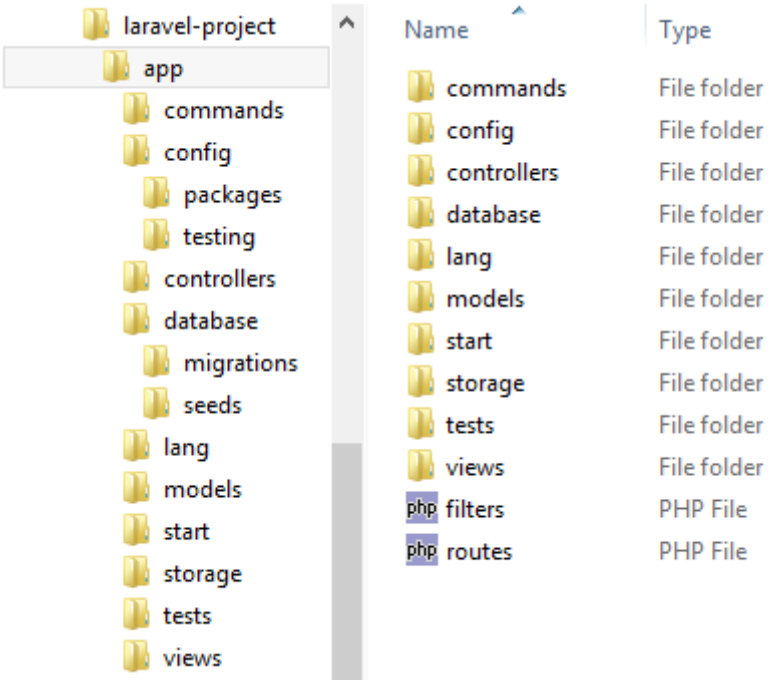
Osnovno pravilo za organiziranje strukture aplikacije je „pravila prije konfiguracije“. To ga čini drugačijim od primjerice Java, Python ili ostalih PHP okvira jer zahtjeva doslovno par linija koda u konfiguracijskim datotekama da bi radio. Isto tako, veliki broj Laravel aplikacija ima gotovo identičnu strukturu direktorija i datoteka – i u takvoj definiciji, svaka datoteka ima svoje unaprijed definirano mjesto. Često se u programerskim krugovima kaže da je struktura „Laravel-way“.

Standardna struktura se sastoji od relativno velikog broja poddirektorija, što se za prvi susret čini zbunjujuće, no struktura se može podijeliti u 3 glavne grupe:

1) `/app/` - sadrži kontrolere, modele i poglede za aplikaciju
2) `/public/` - sadrži *bootstrap* datoteku index.php preko koje ide svaki zahtjev, kao i javno dostupne resurse: CSS, JavaScript, slike itd.
3) `/vendor/` - sadrži jezgru Laravel aplikacije, i sve njene datoteke

Od ova 3, najviše se radi u direktoriju `/app/`

/app/ direktorij sadrži također nekoliko poddirektorija, međutim pravilno je organizirano sve što se nalazi u njemu (slika 2.8.). U tom direktoriju nalaze uglavnom konfiguracija aplikacije, MVC dio, te prijevodi i struktura i sadržaj baze podataka.

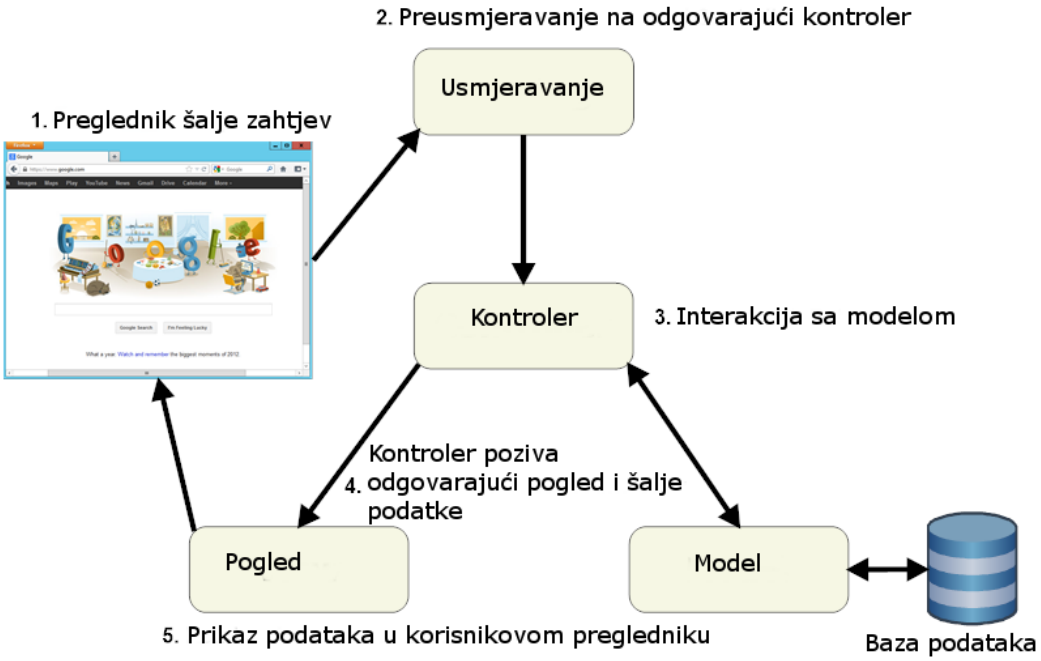


Slika 2.8. Struktura app direktorija kod Laravel aplikacije

Glavne komponente MVC-a kod Laravela su modeli, kontroleri i pogledi (*engl. Views*). Model je usko vezan za kontrolere, i kontroler kao poveznica ne bi mogao raditi bez modela, pa se čak i gubi smisao MVC arhitekture ako taj dio ne radi.

Kako bi se izbjegla toliko velika ovisnost kontrolera o modelu, a i kako bi se smanjilo opterećenje na aplikaciju, postoji praksa da se napravi dodatno sučelje (*engl. Interface*) – repozitorij (*engl. repositories*). Praksa je da se u takvoj situaciji modeli koriste isključivo za dohvat podataka iz baze, i za izradu API-ja na koje se naslanja neka druga aplikacije. Model se tada koristi za pristup „izvan“ aplikacije. Repozitoriji se koristi „unutar“ aplikacije – za spremanje, uređivanje i brisanje podataka u bazi. Tipična Laravel aplikacija se sastoji od spomenutih MVC komponenti koje su u interakciji (Slika 2.9.).

Kada se dogodi interakcija sa Laravel aplikacijom, preglednik šalje zahtjev (*engl. request*), koji prima poslužitelj i šalje Laravelovom sustavu za upravljanje ruta. Sustav tada šalje zahtjev prema odgovarajućem kontroleru i definiranoj metodi unutar njega na osnovu parametara iz rute. Kontroler preuzima zahtjev, i kod dinamičnih web aplikacija komunicira sa modelom koji je PHP objekt koji predstavlja element aplikacije (pr. korisnik, objavu bloga, oglas itd.) i zadužen je za komunikaciju sa bazom. Nakon što su dohvaćeni potrebni podaci, kontroler šalje sve u odgovarajući pregled i sve se prikazuje u korisnikovom pregledniku.



Slika 2.9. Redoslijed akcija u Laravel aplikaciji

Laravel promovira koncept da modeli, pogledi i kontroleri trebaju biti jako odvojeni na način da se kod za svaki dio posebno odvaja u posebnu datoteku i u posebni direktorij. To je specifično za Laravel.

Laravel je uveo jednu novinu – zove se Eloquent ORM. ORM (*engl. Object-Relational Mapping*) omogućava laganu interakciju sa bazom podataka. Eloquent pruža tzv. *ActiveRecord* implementaciju koja radi sa bazom – ona omogućava da svaki model koji se kreira odgovara jednoj tablici u bazi. Tako će `Classifieds` model pripadati `classifieds` tablici u bazi. To nam omogućava da lagano pozivamo podatke iz baze. Ako želimo dohvatiti sve oglase iz tablice `classifieds`, dovoljno je napraviti slijedeću naredbu:

```
$classifieds = Classifieds::all();
```

Eloquent omogućava i kompleksnije upite, primjerice, želimo dohvatiti samo zadnjih 10 aktivnih oglasa:

```
$classifieds = Classifieds::where('active', 1)
               ->orderBy('name', 'desc')
               ->take(10)
               ->get();
```

U strukturi direktorija kod Laravel aplikacije uočavamo i dva direktorija – „migrations“ i „seeds“. Oba sadrže datoteke koje upravljaju sa sadržajem u tablicama u bazi. Razlika je u tome što prve koristimo kako bi kreirali strukturu i veze između tablica a druge koristimo kako bi upisali sadržaj u bazu.

Primjer migracije:

```
Schema::create('classifieds', function(Blueprint $table)
{
    $table->increments('id');
    $table->string('name');
    $table->string('type');
    $table->timestamps();
});
```

Kada se kroz alat Artisan pokrene naredba kojom se pokreće migraciju, kreirati će se tablica „`classifieds`“ sa odgovarajućim poljima (`id, name, type te polja created_at i updated_at`).

Tablicu popunjavamo sa „seeds“ datotekama:

```
    public function run()
    {
        DB::table('users')->insert([
            'name' => str_random(10),
            'email' => str_random(10).'@gmail.com',
            'password' => bcrypt('secret'),
        ]);
    }
```

Na primjeru iznad će se u tablicu `users` upisati korisnički podaci za novi unos – ime, email i lozinka. Za pokretanje *Seed* klase, dovoljno je pokrenuti naredbu u Artisanu:

```
php artisan db:seed
```

3. HIBRIDNE MOBILNE APLIKACIJE

3.1. Razvoj hibridnih mobilnih aplikacija

Hibridne mobilne aplikacije su poput bilo koje druga aplikacije koje se mogu instalirati na mobitel. Objavljene su na tzv. trgovinama aplikacija (*engl. App Store*) i preuzimaju se i instaliraju na uređaj. Aplikacije mogu imati više namjena, mogu biti sa nekom specifičnom svrhom, primjerice, pronalazak lokacije na karti, ili mogu biti namijenjene za zabavu (igrice).

U podjeli aplikacija, postoji podjela u tri osnovna smjera – jedan su tzv. nativne aplikacije, drugi su HTML5 aplikacije, a treće su hibridne aplikacije. Razlikuje se prvenstveno u programskim jezicima u kojima su napisane, u platformi na kojima rade, te prema performansama.

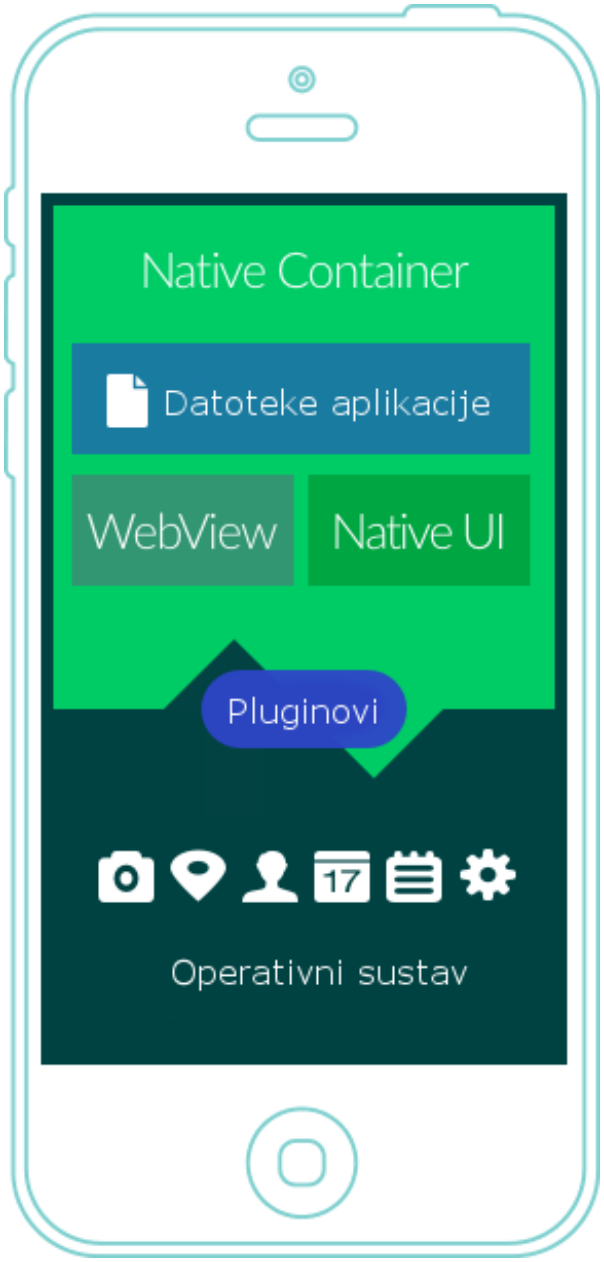
Nativne aplikacije su najbrže i najbolje izgledaju, ali su i najteže za napraviti. Hibridne aplikacije sa druge strane omogućuju da se isti programski kod prekompajlira i radi podjednako dobro na više platformi. Hibridne aplikacije su napravljene kombinacijom HTML5 i JavaScript tehnologija i zaokružene su sa tzv. *native container-om*, drugim riječima, oko hibridne aplikacije se nalazi svojevrsni omotač koji omogućava da aplikacija radi.

Hibridne aplikacije su na neki način simbioza web stranice i nativne aplikacije, jer imaju određene karakteristike i jedne i druge strane. Osnovna je karakteristika da su tehnologije HTML, CSS i JavaScript. Također, hibridne aplikacije se nalaze unutar nativne aplikacije koja se ponaša donekle kao internet preglednika. Ta se aplikacija zove *WebView* i možemo je shvatiti kao da je to obični internet preglednik, samo sa ograničenim funkcionalnostima, i da je stalno u punom prikazu (*engl. full-screen).*

Takav pristup hibridnim aplikacija omogućava da mogu imati dodatne mogućnosti koje internet preglednik nema – pristup kontaktima, kameri, žiroskopu i slično. Osim toga, hibridne aplikacije mogu i u sebi imati mogućnost da koriste nativne elemente korisničkog sučelja gdje je to potrebno.

Većina hibridnih mobilnih aplikacija koristi „Apache Cordova“ platformu. Ona pruža veliki broj JavaScript API-ja koji imaju mogućnost pristupa sposobnostima uređaja kroz dodatke, koji su napravljeni sa nativnim kodom. To znači da primjerice hibridna mobilna aplikacija može imati sposobnost korištenja kamere, i snimanja fotografija i videa, ali nije isprogramirana ta mogućnost u njoj, nego se naslanja na neki već napravljeni dodatak, napisan u nativnom kodu, i taj dodatak

ima mogućnost korištenja kamere, a komunicira sa aplikacijom putem nekakvog API-ja. (slika 3.1.)



Slika 3.1. Shematski prikaz hibridne aplikacije (izvor: ionicframework.com)

Apache Cordova omogućuje da su datoteke aplikacije kao što su HTML, CSS i JavaScript kod zapakirani kroz alata i da se prilagode specifičnoj platformi. Nakon što se izgradi (*engl. build*), može se pokrenuti kao bilo koja druga aplikacija na uređaju.

3.2. Ionic framework[2]

Ionic je HTML5 okvir za izradu hibridnih mobilnih aplikacije. On pruža na jednom mjestu set gotovih funkcionalnosti za brzu izradu mobilnih aplikacija koje su više-platformske (zato i naziv hibridne). Izradom aplikacija koristeći Ionic okvir, isti kod se (poslije kompajliranja) može koristi i za iOS, i za Android i za Windows Phone verziju aplikacije.

Upravo to ga svrstava u jedan jako koristan alat, jer više nije potrebno posebno razvijati aplikaciju za svaku mobilnu platformu. U svojoj srži, svaka aplikacija koja se izradi koristeći Ionic okvir je zapravo jedna mala internet stranica, koja se pokreće unutar nekakvog okvira i ima pristup sloju prema ostalim aplikacijama na uređaju.

Ionic okvir je sofver otvorenog koda (*engl. Open Source*), što znači da mu je kod javno dostupan, i programeri ga mogu modificirati kako žele.

Ionic se razvija od 2013. i zapravo je jako mlad, međutim zajednica ga je jako brzo prihvatila i samo u 2015. je kreirano preko 1,5 milijuna aplikacija koje su bazirane na Ionic okviru.

Ono što Ionic čini pravim okvirom je jako velika količina unaprijed dostupnih resursa. Tako da su dostupni razne mobilne komponente, tipografija, interaktivna ponašanja elemenata, i lako proširiv osnovni izgled. Kako je baziran na Angular JS, Ionic pruža i puno specifičnih komponenti i metoda za interakciju između elemenata. Tako da je Ionic u startu i opremljen sa setom gotovih funkcija koje omogućuju kontrolu toka programa. Postoje predefinirane funkcije koje omogućuje ispis podataka na određeni način – kao npr. *collection repeat* koja ispisuje kolekciju podataka unutar predloška.

Ionic se fokusira na izradu aplikacija korištenih modernim web standardima, i za moderne uređaje. Kod Androida, Ionic podržava verzije iznad 4.1., a kod iOS verziju 7 i više. Ionic u verziji 2 podržava i razvoj Windows Phone aplikacija u verziji Windows 10.

Iako je glavna mana u početku za Ionic bila relativna sporost u odnosu na aplikacije koje su napravljene nativno za neku platformu, međutim, u posljednje vrijeme se jako radilo na tome da je sporost zanemarivo mala.

[2] Jedan od najpopularnijih okvira za razvoj hibridnih mobilnih aplikacija, naziv je dobio po tvrtki koja ga je razvila

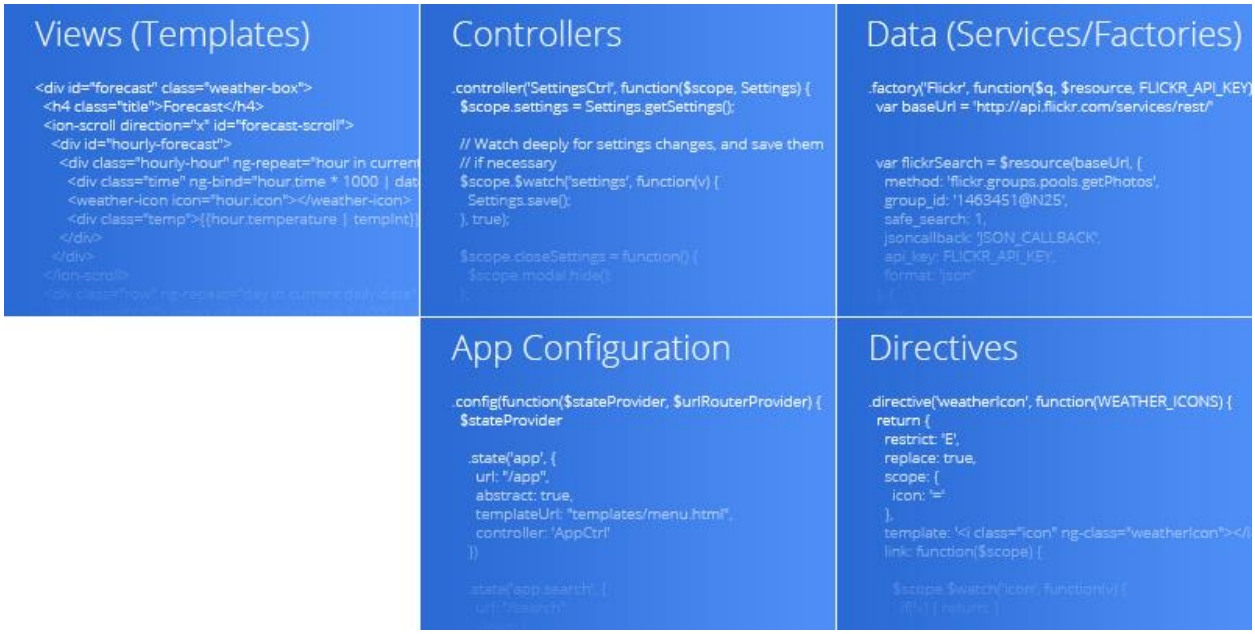
3.3. Struktura Ionic aplikacije

Hibridne aplikacije pružaju mogućnosti programerima koji se već bave razvojem web aplikacija da svoje znanje i vještine iskoriste i za izradu mobilnih aplikacija. Prilikom razvoja hibridnih aplikacije, oni svoje znanje samo proširuju.

Kao i sve druge moderne aplikacije napisane sa MVC načelima, tako i kod hibridnih aplikacija postoji neki predefiniran način kako je aplikacija strukturirana. Na primjeru Ionic aplikacije, podjela je na 5 slojeva (slika 3.3.):

1) Predlošci
2) Kontroleri
3) Podaci (*engl. Data*)
4) Konfiguracija aplikacije
5) Direktive

Predlošci, kontroleri i podatkovni sloj su već poznati iz MVC arhitekture, i oni predstavljaju paralelu (poveznicu) između primjene MVC-a u razvoju web aplikacija i MVC-a kod mobilnih aplikacija.



Slika 3.3. Osnovna struktura Ionic Aplikacije

3.3.1. Predlošci

Predlošci su datoteke sa ekstenzijom .html koji služe za prikaz podataka. Predlošci sadrže opisni kod (*engl. markup*) za određeni pogled ili postranice aplikacije. Predlošci su naslijeđe od Angular JavaScript okvira i paralela su onome što je tamo *template.* Upravo iz tog razloga se i direktorij u koji se spremaju zove „*templates*“.

Za razliku od standardnog HTML programskog jezika Ionic dodaje i neke svoje tzv. *template tagove* – njih možemo razumjeti kao da su zamjenske naredbe za određeni HTML kod. Drugim riječima, kroz set oznaka se zamjeni dio HTML koda koji kompajler razumije. Primjer jednog jednostavnog predloška bi bio:

```
<ion-view title="About">
  <ion-content>
  Ovo je rečenica koja će se ispisati u aplikaciji!
  </ion-content>
</ion-view>
```

Iz primjera vidimo da Ionic *template tagovi* koriste neke ideje i iz HTML-a. Na primjeru vidimo i da svi tagovi imaju svoj početak i kraj.

S obzirom da predložak služi da dinamički prikazujemo sadržaj, u kod se ubacuju i varijable, i to sa specifičnom sintaksom:

```
<ion-view title="About">
  <ion-content>
  Bok, ja sam aplikacija! Zovem se {{appname}}
  </ion-content>
</ion-view>
```

`{{appname}}` je varijabla koja će prikazati ono što joj je poslano iz kontrolera i u njemu je definirana sa dosegom:

```
$scope.name = „Oglasnik";
```

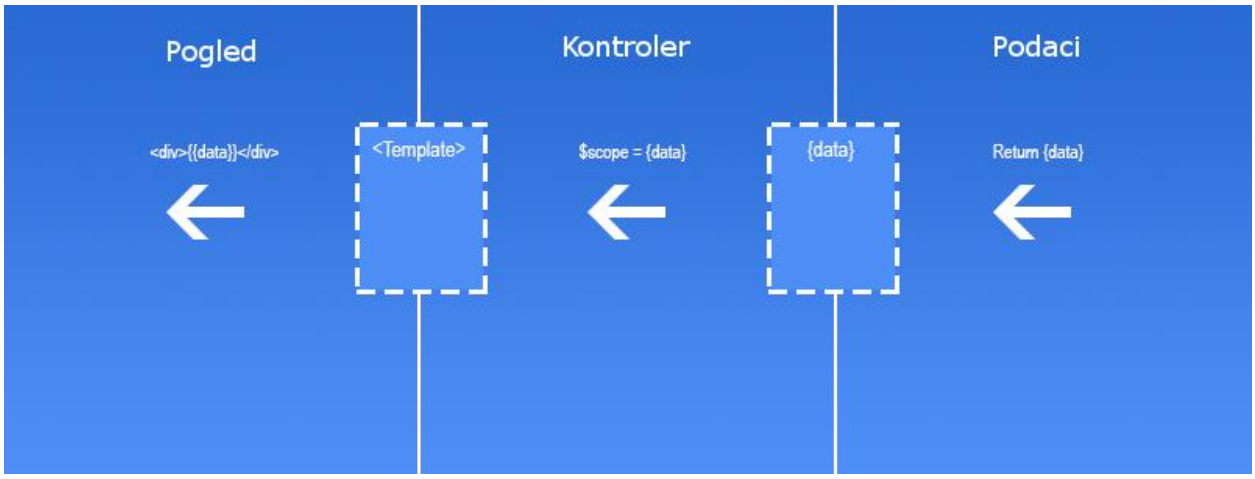
Tako bi ovaj predložak, kada se aplikacija bude izvršila prikazati rečenicu:

```
Bok, ja sam aplikacija! Zovem se Oglasnik.
```

Na predloške se mogu primijeniti i direktive te kontrola toka programa (pr. petlje).

3.3.2. Kontroleri

Kontroleri su „mozak“ aplikacije – to je mjesto gdje se izvršava obrada podataka i kontrola kako će se podaci prikazati. Kada se prikazuje neki dio aplikacije – recimo pod stranica sa tekstom o korisniku aplikacije, zapravo se poziva kontroler, koji dohvaća i obrađuje određeni set podataka i šalje te podatke na prikaz u odgovarajući predložak. Kontroler će koristiti predložak (*engl. template*) za prikaz podataka korisniku te će dohvaćati podatke iz podatkovnog sloja (primjerice – može napraviti poziv na API od neke web aplikacije) kako bi se ti podaci poslali u predložak. (slika 3.2.). Kontroler dodjeljuje podatke u `$scope` varijablu, koja se šalje u predložak. `$scope` je objekt koji sadrži podatke definirane od strane kontrolera, i koristi se da bi se prikazali u predlošku. To zapravo znači da kontroler koristi nekakav unificiran način da pošalje podatke u predložak. `$scope` možemo zamisliti kao pismo koje sadrži poruku.



Slika 3.3.2. Uloga kontrolera u Ionic aplikaciji

Scenarij u kojem aplikacija prikazuje podatke o korisniku bi ovako izgledao:

Aplikacija otvara slijedeći URL (*engl. Uniform Resource Locator*):

```
domain.com/#showUser/365
```

Tada se poziva kontroler showUser (`showUserCtrl`). Kontroler je konfiguriran da koristi predložak userpage.html koji izgleda ovako:

```
 <ion-view title="About">
  <ion-content>
    Dobrodošao, {{user.name}}.
  </ion-content>
</ion-view>
```

Kontroler se oslanja na `userService` klasu (iz API-ja vanjske web aplikacije) kojoj šalje `id` korisnika, kako bi dohvatio podatke o korisniku i spremio ih u objekt `$scope`:

```
.controller('MainCtrl', function($scope, $stateParams, userService,id) {
  id = 365;
  $scope.user = userService.getUser(id);
})
```

Funkcija `getUser` vraća objekt sa setom informacija, i jedna od njih je `name`:

```
var user = { name: "Nikola"; }
```

Kada se stranica sa prikazom profila bude prikazivala, kontroler će uzeti vrijednosti `user` varijable koja je dodijeljena u `$scope` objekt i poslati u predložak. U ovom slučaju, predložak poziva `name` vrijednost iz `$scope` varijable. To će rezultirati na slijedeći način:

```
<ion-view title="About">
  <ion-content>
  Dobrodošao, Nikola.
  </ion-content>
</ion-view>
```

I korisniku će se unutar aplikacije prikazati pozdravna poruka namijenjena njemu.

3.3.3. Podatkovni sloj

Podatkovni sloj Ionic aplikacije isporučuje podatke, uglavnom iz neke vanjske web aplikacije. Kontroler šalje zahtjev na podatkovni sloj, kako bi ih isporučio u predložak. Klase na koje se oslanja se zovu *Services* i *Factories*. Razlike između njih su male, rade slično i predstavljaju podatkovni sloj.

Factory je funkcija gdje se može manipulirati / dodavati logika prije nego se kreira objekt. Kada se koristi *Factory* kreira se objekt sa određenim svojstvima i povratno se vrati isti taj objekt. Kada se objekt vraća, njegova svojstva će biti dostupna u kontroleru.

Service je funkcija koja instancira novi objekt koristeći ključnu riječ `new`. Svojstva i funkcije se mogu dodati u *service* koristeći ključnu riječ `this`. Za razliku od *factory-a* vraća objekt sa metodama i svojstvima i koristi se kada se jedan objekt koristi više puta unutar aplikacije – primjerice korisnički podaci.

U najvećem broju slučaja, podatkovni sloj radi `http` zahtjev i dobije odgovor (*engl. call - promise*) koji kada se obradi sadrži podatke. Primjer jedne klase u podatkovnom sloju:

```
.factory('userService', function($http) {
  return {
    getUser: function() {
        return $http.get("showUser/{id}").then(function(response) {
          //Obrada podataka
          return response;
        });
    },
  }
})
```

Kod u kontroleru koji poziva podatkovni sloj bi izgledao ovako:

```
.controller('MainCtrl', function($scope, $stateParams, userService) {
  userService.getUser().then(function(response){
    $scope.user = response;
  });
})
```

3.3.4. Konfiguracija aplikacije

Konfiguracije aplikacije sadrži parametre i vrijednosti koje se pozivaju prve i određuju koji kontroler će se na osnovu rute (*engl. route*) ili stanja (*engl. state*) izvršiti. U konfiguraciji se zapisuju i koji će se predložak koristiti uz koji kontroler

Primjer jedne konfiguracije iz startne Ionic aplikacije:

```
.config(function($stateProvider, $urlRouterProvider) {
  $stateProvider
    // setup an abstract state for the tabs directive
    .state('tab', {
      url: "/tab",
      abstract: true,
      templateUrl: "templates/tabs.html"
    })
    .state('tab.dash', {
      url: '/dash',
      views: {
        'tab-dash': {
          templateUrl: 'templates/tab-dash.html',
          controller: 'DashCtrl'
        }
      }
    })
  // if none of the above states are matched, use this as the fallback
  $urlRouterProvider.otherwise('/tab/dash');
});
```

Iz te konfiguracije možemo vidjeti da će se za rutu `/tab` koristiti predložak koji se nalazi u direktoriju `templates/tabs.html` i da se neće koristiti nijedan kontroler. Predložak je samo statična HTML datoteka. Sa druge strane, za url `/dash` će se koristiti predložak u direktoriju `templates/tab-dash.html` i njega će ispuniti sa podacima kontroler koji se zove `DashCtrl`. Pojednostavljeno rečeno, `$stateProvider` određuje stranicu ili predložak aplikacije i povezuje ih sa odgovarajućim kontrolerom.

3.3.5 Direktive

Direktive su *markeri* na DOM elementu (*engl. Document Object Model*), kao što su atributi, ime elementa, komentari ili CSS klasa koji govore HTML kompajleru od Angular JS da pridruže određeni način izvršavanja na taj DOM element i/ili njegovim nasljednicima.

Direktive mogu primjerice biti samo atributi / klase koje okidaju određenu funkciju na element, ili ponekada, se ponašaju kao samostalni elementi. Konkretno, u Ionic-u, svi *tagovi* sa sintaksom `ion-*` (zvjezdica predstavlja sve nastavke ostalih *tagova*) su specifične direktive koje su ugrađene u Ionic framework.

Primjer direktive je `<ion-list>`

Direktive imaju svoju sintaksu:

```
IonicModule.directive('ionList', [ '$timeout',
function($timeout) {
  return {
    restrict: 'E',
    require: ['ionList', '^?$ionicScroll'],
    controller: '$ionicList',
    compile: function($element, $attr) {
  //... etc ...
  }
  }
 }
]);
```

Posebno treba uočiti dvije stvari:

1) Restrict: 'E',

Taj parametar govori direktivi gleda li atribut, element ili klasu. Mogli smo ga definirati na slijedeće načine:

'A' – traži samo imena atributa

'E' – traži samo imena elemenata

'C' – traži samo imena klasa

'AEC' – traži ili atribut ili klasu ili element.

Na primjeru `ion-list`, traži element sa tim imenom.

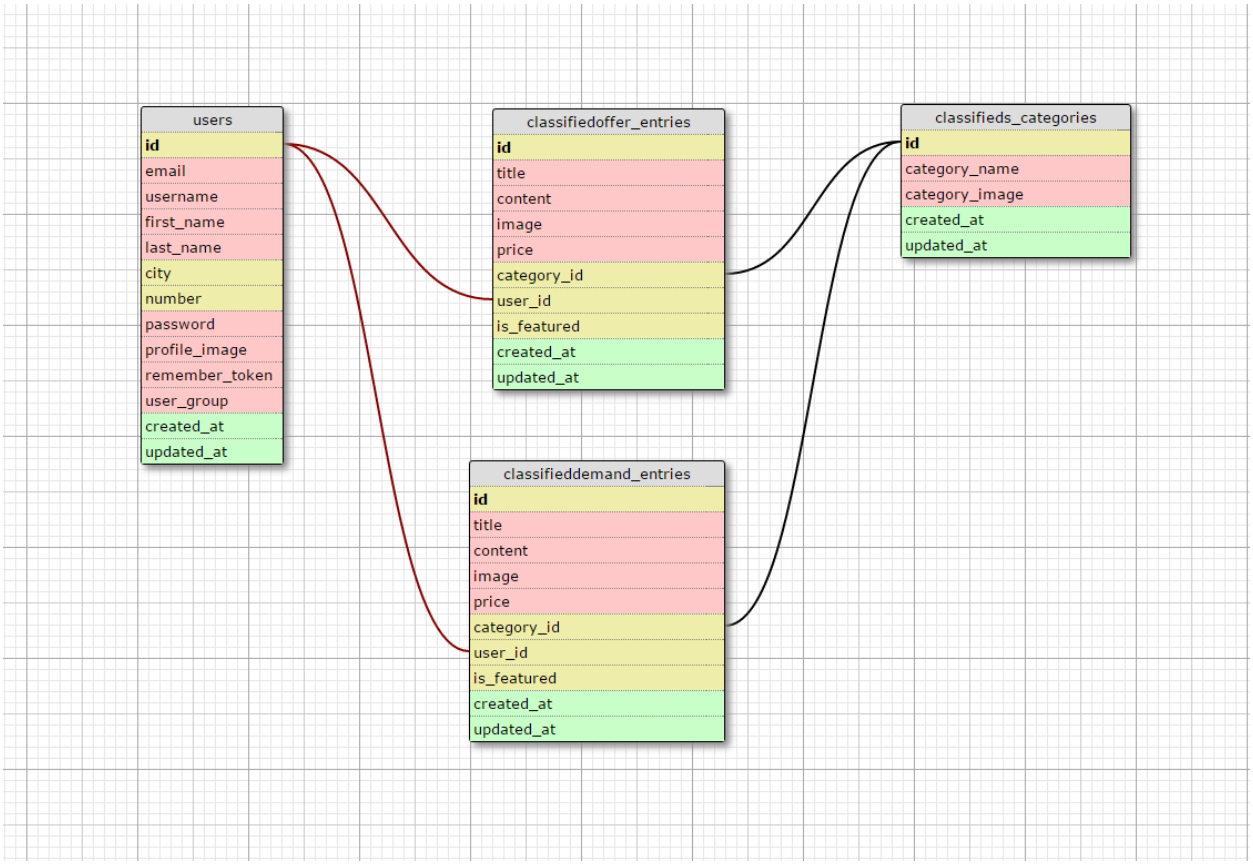
2) Ime direktive: `ionList` umjesto `ion-list`. Direktive ne zahtijevaju da se zovu kao i traženi element, pa tako za ime direktive možemo uzeti i `ion-tab` i `ionTab`, kako bi se odnosilo na isto.

Iz ovih osnovnih pravila izvodi se zaključak da nam Ionic dopušta da sami pišemo svoje direktive. Drugim riječima, Ionic je fleksibilan na način da praktički možemo definirati svoj vlastiti kod za prikaz elemenata u predlošku.

4. RAZVOJ SERVER-SIDE APLIKACIJE

4.1. Izrada baze podataka

Jedan od prvih koraka kod izrade server-side aplikacije je izrada baze podataka. Baza je bazirana na MySQL verziji 5.6.17. S obzirom da je ovo manja aplikacija, i baza koja će biti kreirana, se sastoji od ukupno 4 tablice i struktura je prikazana ER dijagramom (slika 4.1.). Neke velike potrebe za normalizacijom nije bilo. Prilikom konstruiranja tablica poštovale su se normalne forme (1., 2. i 3.). Svaka tablica ima jedan stupac pod imenom `'id'`, koje je primarni ključ, i kod popunjavanja mu je dodana vrijednost da je *autoincrement* (prilikom popunjavanja novog zapisa se vrijednost prethodnog poveća za 1). Tip podataka u tom polju je INT (kako kreće od 0, znači da nam je omogućeno 2147483647 unosu u tu tablicu). Polje `'id'` nam omogućuje da razlikujemo svaki zapis u tablici kao jedan red.



Slika 4.1. ER model baze podataka

Prilikom izrade tablica, u sve tablice su ugrađeni i stupci `'created_at'` i `'updated_at'`. To su stupci koje zahtjeva Eloquent (jedan od Laravelovih načina za komunikaciju sa bazom podataka).

Baza podataka se mogla podijeliti i u samo 3 tablice – gdje bi se oglase umjesto u 2 tablice (jedna za oglase ponude, druga za oglase potražnje) spremilo u samo jednu tablicu, međutim, zbog određenih prednosti, ipak su napravljene 2 tablice. Kako je aplikacija napravljena modularno i kako postoji mogućnost daljnjeg proširenja te kako bi se smanjilo opterećenje na samo jednu tablicu, odabrana je opcija da se podijeli u 2 tablice.

Struktura tablica je prikazana slijedećim SQL naredbama:

```
--
-- Table structure for table `classifieddemand_entries`
--
CREATE TABLE IF NOT EXISTS `classifieddemand_entries` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `title` varchar(255) NOT NULL,
  `content` text NOT NULL,
  `image` varchar(255) NOT NULL,
  `price` decimal(7,2) NOT NULL,
  `category_id` int(11) NOT NULL,
  `user_id` int(11) NOT NULL,
  `is_featured` varchar(1) NOT NULL,
  `created_at` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE
CURRENT_TIMESTAMP,
  `updated_at` timestamp NOT NULL DEFAULT '0000-00-00 00:00:00',
  PRIMARY KEY (`id`)
) ENGINE=InnoDB  DEFAULT CHARSET=utf8;


--
-- Table structure for table `classifiedoffer_entries`
--
CREATE TABLE IF NOT EXISTS `classifiedoffer_entries` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `title` varchar(255) NOT NULL,
  `content` text NOT NULL,
  `image` varchar(255) NOT NULL,
  `price` decimal(7,2) NOT NULL,
```

```
  `category_id` int(11) NOT NULL,
  `user_id` int(11) NOT NULL,
  `is_featured` varchar(1) NOT NULL,
  `created_at` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE
CURRENT_TIMESTAMP,
  `updated_at` timestamp NOT NULL DEFAULT '0000-00-00 00:00:00',
  PRIMARY KEY (`id`)
) ENGINE=InnoDB  DEFAULT CHARSET=utf8;


--
-- Table structure for table `classifieds_categories`
--
CREATE TABLE IF NOT EXISTS `classifieds_categories` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `category_name` varchar(255) NOT NULL,
  `category_image` varchar(255) NOT NULL,
  `created_at` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE
CURRENT_TIMESTAMP,
  `updated_at` timestamp NOT NULL DEFAULT '0000-00-00 00:00:00',
  PRIMARY KEY (`id`)
) ENGINE=InnoDB  DEFAULT CHARSET=utf8;


--
-- Table structure for table `users`
--
CREATE TABLE IF NOT EXISTS `users` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `email` varchar(255) NOT NULL,
  `username` varchar(30) NOT NULL,
  `first_name` varchar(30) NOT NULL,
  `last_name` varchar(30) NOT NULL,
  `city` varchar(30) NOT NULL,
  `number` varchar(30) NOT NULL,
  `password` varchar(100) NOT NULL,
  `profile_image` varchar(255) NOT NULL,
  `remember_token` varchar(255) NOT NULL,
  `user_group` varchar(20) NOT NULL,
```

```
  `created_at` datetime NOT NULL,
  `updated_at` datetime NOT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB  DEFAULT CHARSET=utf8;
-- ---
-- Foreign Keys
-- ---

ALTER TABLE `classifieddemand_entries` ADD FOREIGN KEY (category_id) REFERENCES `classifieds_categories` (`id`);
ALTER TABLE `classifieddemand_entries` ADD FOREIGN KEY (user_id) REFERENCES `users` (`id`);
ALTER TABLE `classifiedoffer_entries` ADD FOREIGN KEY (category_id) REFERENCES `classifieds_categories` (`id`);
ALTER TABLE `classifiedoffer_entries` ADD FOREIGN KEY (user_id) REFERENCES `users` (`id`);
```

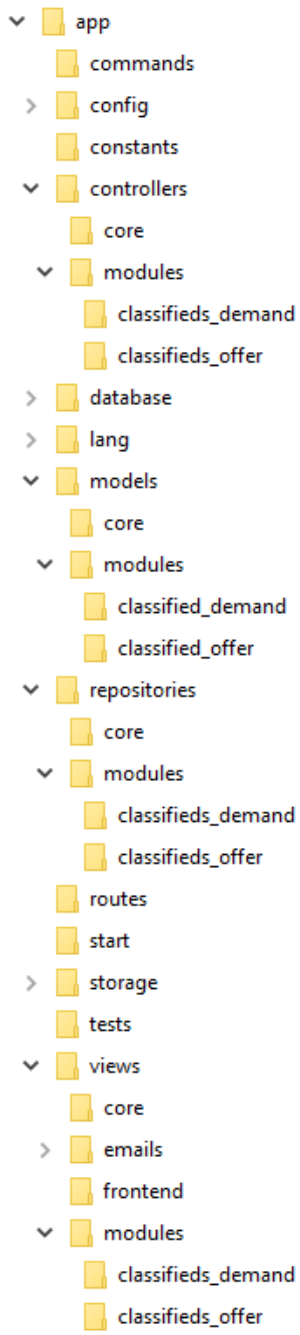
4.2. Struktura aplikacije

U poglavlju 2.5. je opisana struktura i arhitektura tipične Laravel aplikacije, međutim u ovoj aplikacije napravljena je malo drugačija struktura. Naime, kako se broj datoteka u aplikaciji povećava, praksa je da se one na neki način podijele drugačijom strukturom. Iako je ovo mala aplikacija, ona može poslužiti za razvoj puno veće, i zato su odmah u startu napravljene podjele.

Laravel sve kontrolere smješta u direktorij koji se zove „controllers“ – međutim, kada aplikaciju podijelimo primjerice na više od 10 modula (ili dijelova), i da svaki modul ima po nekoliko klasa, onda taj direktorij bude pun PHP klasama, i u slučaju da brzo moramo pronaći odgovarajuću klasu, to se može odužiti. Zato je ovdje implementirana mogućnost koju Laravel pruža – a to je da direktorije podijelimo dodatno u poddirektorije i unutar svakog poddirektorija smjestimo dio klasa (slika 4.2.). Ta podjela je semantička – što bi značilo da primjerice dio aplikacije koje je vezan za oglase ponude, bude u jednom direktoriju, a za oglase potražnje, u drugom direktoriju.

Oba modula se nalaze u direktoriju „modules“ a pored njega u arhitekturi se nalazi i direktorij „core“ koji sadrži jezgru aplikacije (prijava u sustava, autorizaciju, registracija i slično.)

Ono što se koristi kroz više modula se ostavilo u direktoriju kojem svi imaju pristup. Ista struktura je primijenjena kod kontrolera, modela, repozitorija i pogleda.



Slika 4.2. Struktura aplikacije

Ovdje je implementirana samo jedna od mogućnosti kako se mogla organizirati aplikacija. Druga mogućnost koja se mogla implementirati se zove HMVC (*engl. Hierarchical model–view–controller*). Pojednostavljeno rečeno, HMVC dijelu aplikaciju u direktorije gdje svaki modul ima svoje kontrolere, modele, rute, repozitorije. Svaki modul na taj način postaje svojevrsni paket koji se lagano može prebacivati iz aplikacije u aplikaciju.

4.3. Dizajn aplikacije

Zbog bržeg razvoja, prilikom dizajna aplikacije se koristi Bootstrap – to je HTML okvir koji omogućuje rapidni razvoj aplikacija. Dizajn možemo podijeliti na 2 glavna dijela:

1) *Frontend dio* – dizajn internet stranice
2) *Backend dio* – dizajn pozadinskog sučelja

Osnovne smjernica kod dizajna je da bude „uniformni“, odnosno da isti elementi koji se pojavljuju na nekoliko mjesta u aplikaciji svugdje isto izgledaju. Osim toga, kako bi se smanjio broj različitih predložaka, isti predložak se koristio na više mjesta i ukupno postoje 3 glavna dizajna za dizajn internet stranice:

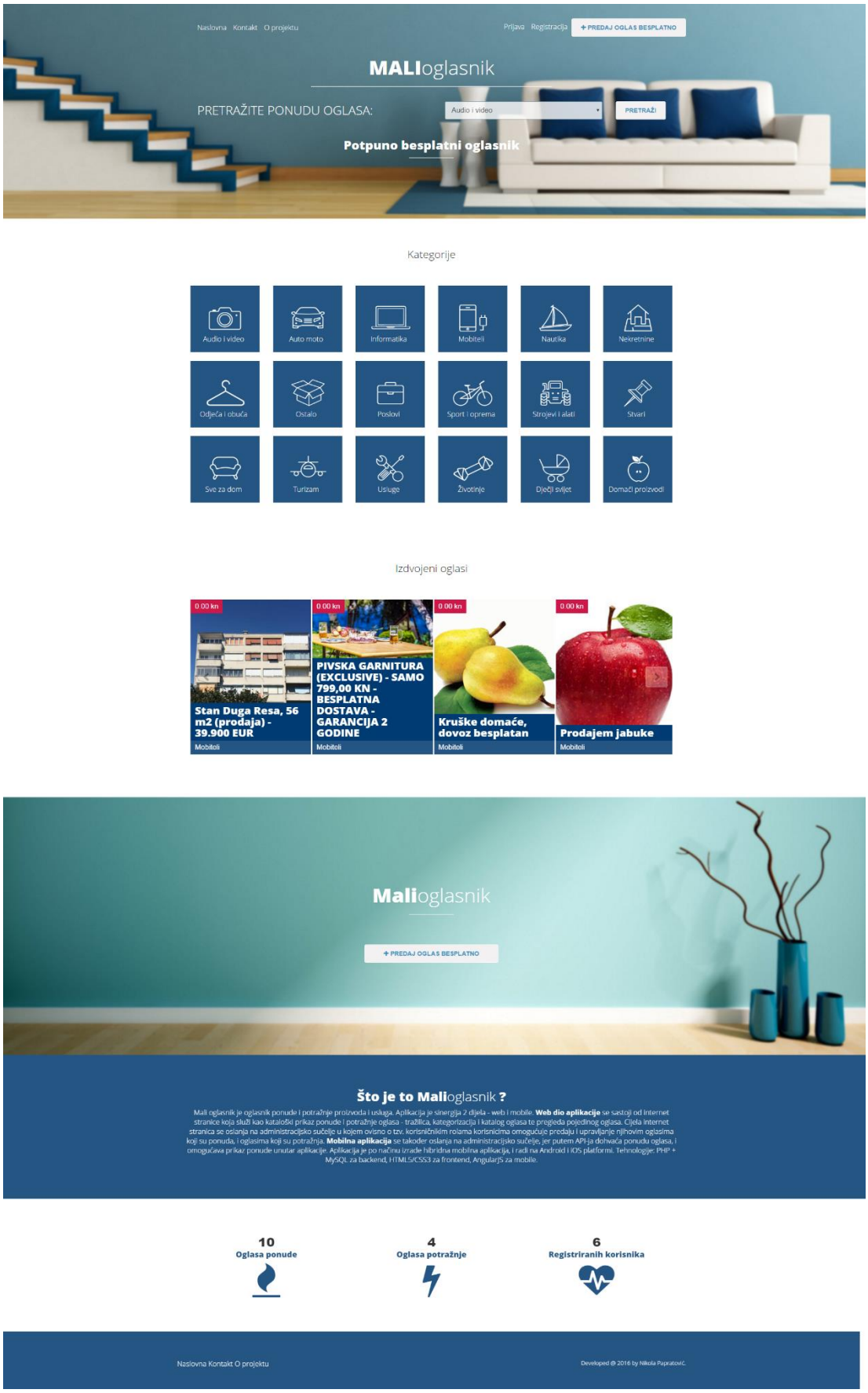
1) Dizajn naslovne stranice
2) Dizajn stranice jedne kategorije oglasa
3) Dizajn pod stranice jednog oglasa

Naslovna stranica (slika 4.3.) je zamišljena kao prezentacijska stranica sa nekoliko elemenata:

- Zaglavljem stranice sa tražilicom
- Popis kategorija
- Izdvojeni oglasi
- Poziv na predaju oglasa
- Opis stranice, i osnovne informacije
- Statistički podaci
- Podnožje stranice

Stranica kategorije ima malo drugačiju vizualnu strukturu u odnosu na naslovnu – i ispisuje oglase iz odabrane kategorije. Na stranici pojedinog oglasa pored svih informacija o tom oglasu, korisnik može i kontaktirati oglašivača.

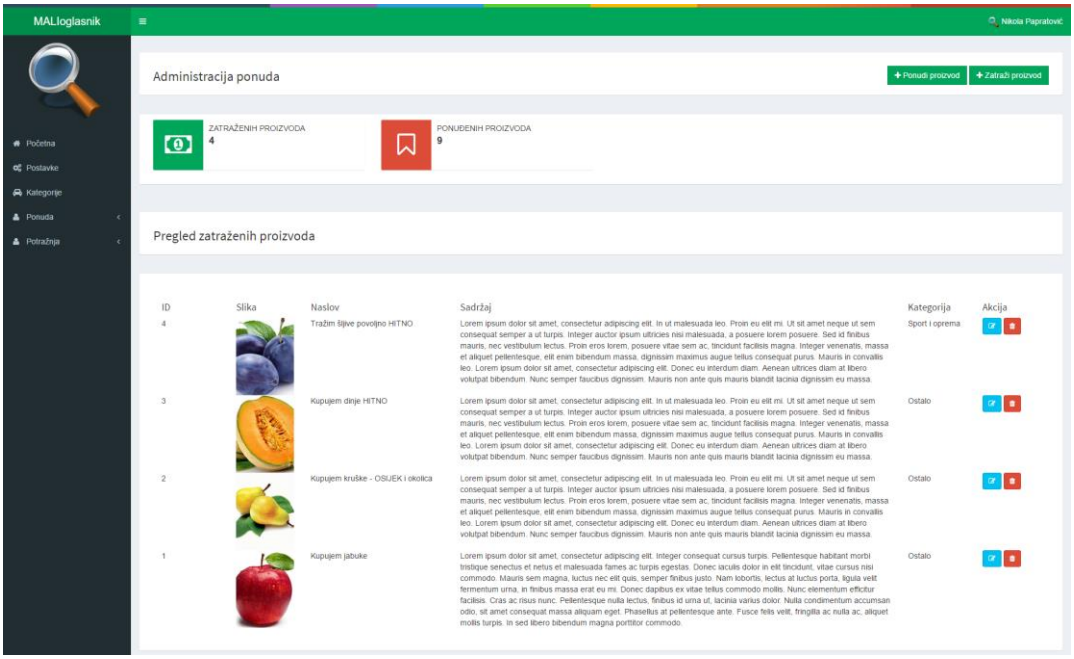
Za izdvojiti je još i korisničke pod stranice na kojima se novi korisnici mogu registrirati u sustav ili prijaviti.



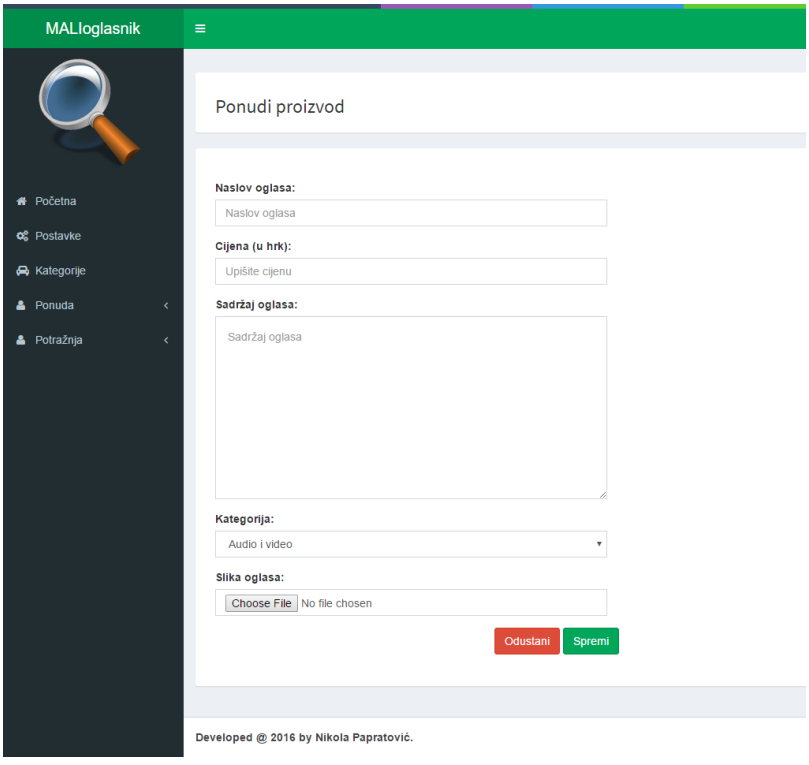
Slika 4.3. Dizajn naslovnice

Kod dizajna pozadinskog sučelja, također ga možemo podijeliti u 3 glavna pogleda:

1) Dizajn početne stranice administrativnog sučelja (slika 4.4.)
2) Dizajn ispisa (kategorije, ili liste oglasa iz ponude i potražnje)
3) Dizajn za dodavanje novog oglasa ili uređivanje postojećeg (slika 4.5.)



Slika 4.4. Početna stranica administracije



Slika 4.5. Dizajn stranice za dodavanje jednog oglasa

Također, bitno je navesti da postoje 2 korisničke uloge sa različitim pravima pristupa, pa tako postoje obični korisnici i administratori. Obični korisnici vide samo svoje oglase, a administratori vide sve oglase, i sve ih mogu uređivati i pregledavati, te mogu i administrirati kategorije.

4.4. CRUD funkcionalnosti[3]

Dio koda sa kojim kreiramo i uređujemo zapise u bazi bi u standardnom PHP-u bio jako veliki. Laravel nam je omogućio da to izgleda jako jednostavno. Možemo reći da postoje 3 glavna dijela koda – jedan je za ispis podataka, drugi je za prikaz HTML forme sa kojom unosimo / uređujemo podatke i treći dio je za spremanje zapisa uz bazu. Ta tri dijela možemo nazvati *list, create / edit, store / update.* Tome možemo još dodati i *destroy* metodu i imamo sve potrebno za potpunu funkcionalnost sustava.

4.4.1. Kreiranje novog zapisa u bazi (*engl. Create*)

Prvi korak nam je definiranje rute:

```
Route::get('add-entry', array('before' => 'auth', 'as' =>
'ClassifiedsOfferGetAddEntry', 'uses' =>
'ClassifiedsOfferController@getAddEntry'));
```

Iz toga možemo pročitati par parametara:

- Stalna veza glasi: `'add-entry'`
- Korisnik mora biti prijavljen u sustav
- Koristi se kontroler `'ClassifiedsOfferController'` i pokreće se metoda `'getAddEntry'`

U kontroleru funkcija koja prikazuje HTML formu za popunjavanje podataka izgleda ovako:

```
public function getAddEntry()
      {
$this->layout->content = View::make('modules.classifieds_offer.entry',
'postRoute' => 'ClassifiedsOfferPostAddEntry', 'title' => 'Ponudi proizvod'
));
```

[33] CRUD (*engl. create, read, update, destroy*) je kratica za kreiranje, čitanje, ažuriranje i brisanje zapisa

```
    }
```

Ova funkcija samo kaže pregledniku da učita pogled koji će prikazati formu za unos novog oglasa. Pogled kod Laravela koristi tzv. *Blade sintaksu* – to je poseban „jezik“ koji mijenja HTML sa odgovarajućim *tagovima.*

```
{{ Form::open(array('role' => 'form', 'class' => 'form-horizontal',
'autocomplete' => 'off', 'files' => true)) }}

{{ Form::text('title', isset($entry->title) ? $entry->title : null, ['class'
=> 'form-control', 'placeholder' => 'Naslov oglasa']) }}

{{ Form::textarea('content', isset($entry->content) ? $entry->content : null,
array('class' => 'form-control content', 'placeholder' => 'Sadržaj
oglasa','id' => 'content')) }}

{{ Form::button(array('type' => 'submit', 'class' => 'btn btn-success')) }}

{{ Form::close() }}
```

Ovih par redaka će generirati sav potreban HTML kod koji ima 2 polja – jedno je za unos naslova oglasa, a drugi je za unos sadržaja. I kada korisnik popuni polja, i odabere da spremi oglas, sve se putem POST metode šalje na rutu koja poziva odgovarajuću metodu za spremanje u kontroleru:

```
Route::post('new-entry', array('before' => 'auth', 'as' =>
'ClassifiedsOfferPostAddEntry', 'uses' =>
'ClassifiedsOfferController@postAddEntry'));
```

Iz te rute možemo vidjeti da će se pokrenuti metoda `'postAddEntry'` u `'ClassifiedsOfferController'` klasi. Metoda koja sprema podatke skraćeno izgleda:

```
public function postAddEntry()
    {
        Input::merge(array_map('trim', Input::all()));
        $addNewEntry = $this->repo->addEntry(
        Input::get('title'),
        Input::get('content')
    );
}
```

U toj metodi stoji da se poziv još jedna metoda – koja se nalazi u repozitoriju i zove se `'addEntry'`. Procesi spremanja novog, uređivanje postojećeg i brisanje zapisa se odvijaju u repozitoriju. Samo se metode za dohvat podataka o jednom ili svim zapisima odvijaju u modelima. Metoda za spremanje zapisa u repozitoriju izgleda ovako:

```
public function addEntry($title, $content)
    {
        $entry = new ClassifiedOfferEntry;
```

```
            $entry->title = $title;
            $entry->content = $content;
            $entry->save();
      }
```

Metoda ima dva dijela – u prvom dijelu se poziva model `ClassifiedsOfferEntry` – to je Eloquent klasa koja kreira novi zapis u `classifieds_offer` tablici. Drugi dio su samo vrijednosti varijabli koje se spremaju.

Nakon što se podaci spreme u odgovarajuću tablicu, u metodi za spremanje novog zapisa se pokrene ruta koja vraća preglednik na ispis svih oglasa i prikazuje se poruka da je zapis uspješno spremljen:

```
if ($addNewEntry['status'] == 0)
      {
      return Redirect::route('getClassifiedsList')->with('success_message',
Lang::get('core.msg_success_entry_added', array('name' =>
Input::get('name'))));
      }
```

4.4.2 Uređivanje zapisa u bazi (*engl. Update*)

Uređivanje (*engl. edit*) pa spremanje (*engl. update*) uređenog oglasa je jako slično dodavanju novog – razlika leži u tome, što se na početku poziva metoda za uređivanje oglasa, i prima parametar sa identifikatorom oglasa – primjerice njegovim `'id'`-em iz baze. Prilikom spremanje podataka za taj oglas se prvo dohvati postojeći oglas iz baze, i njegove vrijednosti se samo ažuriraju (dakle, ne pokreće se model koji stvara novi zapis).

4.4.3. Brisanje zapisa iz baze (*engl. Destroy*)

Brisanje zapisa iz baze je puno jednostavnije – pokrene se ruta za brisanje i dohvati se odgovarajući oglas:

```
Route::get('delete-entry/{id}', array('before' => 'auth', 'as' =>
'ClassifiedsOfferGetDeleteEntry', 'uses' =>
'ClassifiedsOfferController@getDeleteEntry'));
```

Metoda u kontroleru koja pokreće brisanje:

```
public function getDeleteEntry($id = null)
	{
		//Dohvaćanje oglasa
		$entry = ClassifiedOfferEntry::getSingleOfferEntry($id);
		//Brisanje oglasa
		$deleteEntry = $this->repo->deleteEntry($id);
}
```

Metoda iz kontrolera je pozvala metodu iz repozitorija koja briše zapis iz tablice:

```
public function deleteEntry($id)
	{
		$entry = ClassifiedOfferEntry::find($id);
		$entry->delete();
	}
```

Nakon uspješnog brisanja, preglednik vraća korisnika na pregled svih oglasa.

4.4.4. Čitanje zapisa iz baze (*engl. Read*)

Ispis svih oglasa se sastoji od par koraka i kreće sa pozivom rute za ispis stranice sa prikazom svih oglasa:

```
Route::get('/', array('as' => 'ClassifiedsOfferLanding', 'uses' =>
'ClassifiedsOfferController@getLanding'));
```

Metoda `getLanding` u klasi `ClassifiedsOfferController`:

```
public function getLanding()
	{
		$classifiedsoffers =
ClassifiedOfferEntry::getLastClassifiedsOffers();
		$this->layout->content =
View::make('modules.classifieds_offer.entryList', array('title' => 'Pregled
oglasa "nudim"', 'classifiedsoffers' => $classifiedsoffers ));
	}
```

Unutar metode getLanding, treba uočiti da se poziva odgovarajući model i metoda u njemu:

```
$classifiedsoffers = ClassifiedOfferEntry::getLastClassifiedsOffers();
```

Modeli su napravljeni da samo dohvaćaju zapise iz baze. Metoda za dohvat zapisa:

```
public static function getLastClassifiedsOffers)
      {
            $entries = DB::table('classifiedoffer_entries')
                        ->select(
                              'classifiedoffer_entries.title AS title',
                              'classifiedoffer_entries.content AS content'
                        )
                        ->orderBy('classifiedoffer_entries.created_at',
'DESC')
                        ->get();
      }
```

Metoda je pozvala pogled koji se zove `entryList`, i oglasi se mogu prikazati:

```
@foreach($classifiedsoffers as $classifiedsoffer)
      <div class="row">
            <div class="col-lg-6">
                  {{ $classifiedsoffer->title }}
            </div>
            <div class="col-lg-6">
                  {{ $classifiedsoffer->content }}
            </div>
</div>
@endforeach
```

4.5. Sigurnost i proširenja

Može se uočiti da je tijek radnji uvijek isti i slijedi putanju kao što je prikazano na slici 2.9. Preglednik pozove neku rutu, pozove se metoda iz kontrolera, kontroler tada poziva metodu iz modela ili metodu iz repozitorija, i na kraju se prikaže pogled korisniku. Primjeri koda koji su navedeni su samo reprezentativni primjeri.

MVC način razmišljanja što je prikazano za oglase, se na isti način radi i za kategorije. Prednji dio internet stranice je jednostavan na način da se tamo samo ispisuju oglasi, kategorija oglasa ili pojedinačni oglas.

Primjerice, na naslovnoj se nalazi ispis zadnjih oglasa – za ispis je odgovoran kontroler `FrontendController` koji komunicira sa modelom i metodom za dohvat oglasa, te ih šalje u pogled koji iz prikazuje.

U stvarnoj aplikaciji, dodana je još autorizacija i autentifikacija. Tako da uvijek kada se učitava metoda za uređivanje određenog oglasa, prvo se provjerava da li je korisnik prijavljen u sustav, i onda se provjerava ima li on prava pristupa toj metodi (ako recimo oglas nije njegov, ne može ga uređivati). Zatim, sve metode koje se izvršavaju u modelima i repozitorija sadrže i tzv. `try {} catch {}` blok sa kojim se prvo *pokuša* izvršiti kod, a ako se slučajno ne izvrši, vrati se iznimka (*engl. Exception*).

Što se tiče sigurnosti – integrirana je CSRF zaštita (*engl. Cross-site request forgery*) sa kojom štitimo podatke od korisnika. Dodana je i validacija (provjera unesenoga, da li su unesene sve potrebne vrijednosti) te salinizacija – ispravljanje potencijalno malicioznog koda.

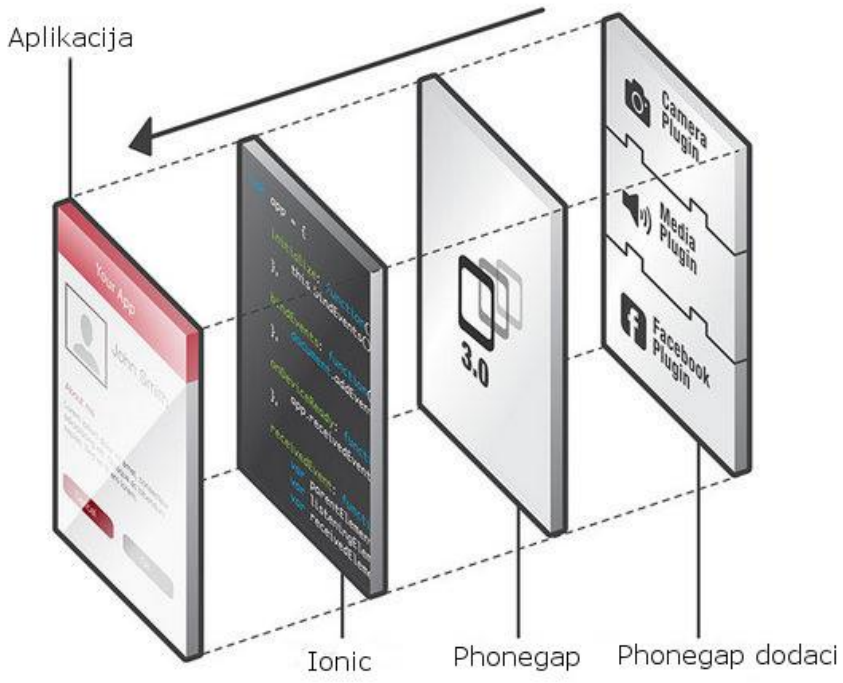
5. RAZVOJ CLIENT-SIDE APLIKACIJE

5.1. Dizajniranje izgleda mobilne aplikacije

Klijentska strana aplikacije je također napravljena prema MVC obrascu. U pozadini se nalazi Ionic okvir, i on omogućava brzi razvoj. Ionic je u svojoj strukturi jednostavniji od poslužiteljske strane i moguće je lakše uvidjeti MVC načela. Isto kao i kod poslužiteljske strane, i ovdje imamo određene *rute* koje pozivaju kontrolere. Kontroleri tada rade zahtjev za podacima (kod ove aplikacije pozivaju model iz web aplikacije) i sve to prikazuju u pogledu.

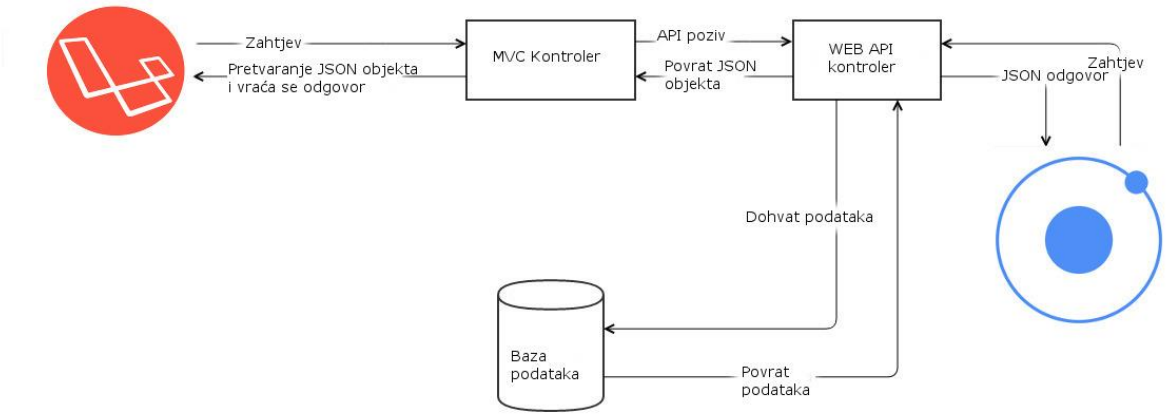
Kao što je opisano u poglavlju 3.1. Ionic hibridna mobilna aplikacija ima više slojeva, i one su obične internet stranica, ali smještena u nativni omotač. To znači da se može koristiti HTML, CSS i JavaScript sa svim efektima kako želimo. Jedina razlika je, da umjesto internet stranice koja je povezana sa drugim internet stranicama, pravimo aplikaciju koja ima unutrašnju strukturu poveznica.

Ionic aplikaciju možemo podijeliti u nekoliko slojeva (slika 5.1.) – Phonegap je određena platforma koja povezuje programski kod koji je napisan sa dodacima koji imaju nativne funkcionalnosti (kamera, žiroskop itd.). Povrh svega se nalazi dizajn sa kojim radimo izgled aplikacije.



Slika 5.1. Slojevi hibridne aplikacije (izvor ionicframework.com)

Mobilna aplikacija se jako oslanja na web aplikaciju, i bez nje ne bi radila. Naime, osnovni zadatak mobilne aplikacije je da dohvati podatka sa servera. (Slika 5.2.) Zadatak je jednostavan: pošalje se `$http` zahtjev – za ispisom oglasa iz određene kategorije ili za pojedinog oglasa. Web aplikacija zaprimi zahtjev, i povratno vrati podatke u formatu koji se može prikazati u mobilnoj aplikaciji.



Slika 5.2. Tok upita za podacima između Laravel web aplikacije i Ionic mobilne aplikacije

Prilikom izrade mobilne aplikacije, glavni zadatak je bio napraviti aplikaciju koja služi za prikaz ispisa. Korisničko sučelje nije komplicirano, i lagano je za upotrebu. Primjerice, početna stranica (Slika 5.3.) aplikacije ima samo 3 opcije:

1) Ponuda
2) Potražnja
3) Info

Svaka od te tri opcije vodi na neku poveznicu unutar same aplikacije. Odabirom „Ponuda“ ili „Potražnja“ otvara se popis sa kategorijama i tražilica kategorija na vrhu. Tražilica radi po principu da filtrira ispisane kategorije prema imenu.

Odabirom jedne od kategorije, prikazuje nam se ponuda oglasa iz te kategorije. Ispisuju se samo imena oglasa, i svaki oglas je poveznica za pod stranicu koja sadrži samo podatke o tom oglasu.

Slika 5.3. Početna stranica aplikacije

5.1.Dohvat i prikaz kategorija oglasa

Ova aplikacija se bazira na dohvatu podataka i prikazu unutar aplikacije. Samim time, njena veličina je jako malo, potrebno je samo napraviti minimalni set funkcionalnosti koje će raditi. Sav sadržaj unutar aplikacije je samo rezultat onoga što se u web aplikaciji napravilo. Tako recimo, čim se doda nova kategorija oglasa, odmah će i biti vidljiva unutar aplikacije, bez dodatnih ažuriranja.

Aplikacija unutar sebe ima određenu strukturu linkova, i da bi se prikazao neki pogled, prvo je potrebno definirati rutu (poveznicu, *engl. state*) koja će biti pokazivač koji kontroler će prikazati pogled na toj ruti.

`Index.html` datoteka koja se nalazi na početku aplikacije služi kao omotač za ostale poglede, jer sadrži putanju do svih bitnih resursa (JavaScript i CSS datoteke)

Kako bi prikazali listu svih kategorija, prvo dodajemo `state` u app.js datoteku, unutar konfiguracije:

```
    .state('menu.categories-demand', {
      url: "/categories-demand",
      views: {
        'menuContent' :{
          templateUrl: "templates/categories-demand.html"
        }
      }
    })
```

Iz te konfiguracije možemo vidjeti da kada se pokrene poveznica `'categories-demand'` da će se učitati predložak `'categories-demand.html'`:

```
<ion-view title="Kategorije">
    <ion-nav-buttons side="left">
        <a href="#/menu/home" class="button button-icon icon backButton"><i class="ion-ios7-arrow-thin-left"></i></a>
    </ion-nav-buttons>
    <ion-nav-buttons side="right">
        <a href="#/home" class="button button-icon icon homeButton"><i class="ion-ios7-keypad-outline"></i></a>
    </ion-nav-buttons>
    <ion-content class="categoriesPage">
        <div class="category-list padding" ng-controller="getRemoteDataCategories">
            <div class="category-item item-text-wrap" ng-repeat="category in categories | filter:data.searchQuery">
                <a class="item item-icon-right" href="#/menu/demand-by-category/{{category.id}}">
                            {{category.category_name}}
                        <i class="icon ion-ios7-arrow-right"></i>
                    </a>
            </div>
        </div>
    </ion-content>
</ion-view>
```

Pogled je obični HTML dokument koji se sastoji od HTML i Ionic *tagova.* Međutim, možemo uočiti da se poziva controller '`getRemoteDataCategories'`. Ispod poziva kontrolera se nalazi i Ionic direktiva `ng-repeat='category in categories'`. Ta direktiva je odgovorna za ispis kategorija iz polja (engl. array) koje se povratno vratilo iz kontrolera. Kontroler za dohvat kategorija glasi:

```
.controller('getRemoteDataCategories', function ($http, $rootScope) {
  $http.get('http://classifiedapp.com.hr/mobile/getAllCategories')
    .success(function(data) {
        $rootScope.categories = data;
      }).error(function (){
      alert('notok');
    });
})
```

Kontroler se spaja na udaljeni link preko `$http.get` metode. Link na koji se spaja je ruta od Laravel aplikacije. Ta ruta pokazuje na metodu u kontroleru, koji poziva metodu iz modela, dohvaća podatke iz baze, konvertira ih u JSON format, i vraća nazad u aplikaciju. Metoda u repozitoriju koja dohvaća sve kategorije:

```
public function getAllCategories () {
      $AllCategories = DB::table('classifieds_categories as cc')
            ->select('cc.id', 'cc.category_name', 'cc.created_at')
            ->orderBy('cc.id', 'desc')
            ->get();
            return json_encode($AllCategories);
}
```

Kada se polje sa svim kategorijama vrati nazad u kontroler, onda se prikazuje u HTML predlošku (slika 5.4.) sa Ionic direktivom `ng-repeat`.

Slika 5.4. Prikaz kategorija oglasa

5.2. Dohvat i prikaz jednog oglasa

Prikaz jednog oglasa je nešto drugačiji od prikaza kategorije. Ovdje više ne trebamo listu svih resursa, nego se u aplikaciji prikazuju podaci samo za jedan oglas. Prvo se postavlja *state* u konfiguracijskoj datoteci:

```
.state('menu.single-offer', {
      url: "/single-offer/{offerId}",
      views: {
        'menuContent' :{
          templateUrl: "templates/single-offer.html",
          controller: 'offerSingleCtrl'
        }
      }
    })
```

Ovdje je promjena u tome što se uz url bilježi i *offerId*, te se odmah poziv odgovarajući kontroler – `offerSingleCtrl`. Kontroler za dohvat jednog oglasa ponude:

```
.controller('offerSingleCtrl', function($scope, $stateParams, $rootScope,
$http) {
     $http({
```

```
      url: "http://classifiedapp.com.hr/mobile/getOfferByID",
      method: "POST",
          headers: {
              'Content-Type': 'application/x-www-form-urlencoded'
          },
          data: {
              id:$stateParams.offerId
          }
     })
    .success(function(data) {
     $scope.offerbyID = data;
    }).error(function (err1, err2,err3,err4){
    //  alert(err1+err2+err3+err4);
    });
})
```

U tom kontroleru možemo vidjeti da se putem POST metode na rutu šalje `offerId`. Taj `offerId` dolazi na rutu od Laravela, ona ga prosljeđuje u kontroler u odgovarajuću metodu. Ta metoda poziva drugu metodu iz repozitorija, i dohvaća se točno određeni zapis iz baze u JSON formatu:

```
 public function getOfferByID ($id) {
      $offer = DB::table('classifiedoffer_entries')
            ->join('classifieds_categories', 'classifieds_categories.id', '=',
'classifiedoffer_entries.category_id')
            ->select(
                  'classifiedoffer_entries.id AS entry_id',
                  'classifiedoffer_entries.title AS title',
                  'classifiedoffer_entries.content AS content',
                  'classifiedoffer_entries.category_id AS category_id',
                  'classifiedoffer_entries.user_id AS user_id',
                  'classifiedoffer_entries.image AS image',
                  'classifiedoffer_entries.price AS price',
                  'classifieds_categories.category_name AS category_name'
                  );
            $offer = $offer->where('classifiedoffer_entries.id', '=', $id)   -
>first();
            return json_encode($offer);
 }
```

Kada se podaci vrate natrag u mobilni kontroler, onda se prosljeđuju u HTML predložak:

```
<ion-view title="Pregled oglasa">
    <ion-nav-buttons side="left">
        <a href="#/menu/categories-demand" class="button button-icon icon backButton"><i class="ion-ios7-arrow-thin-left"></i></a>
    </ion-nav-buttons>
    <ion-nav-buttons side="right">
        <a href="#/home" class="button button-icon icon homeButton"><i class="ion-ios7-keypad-outline"></i></a>
    </ion-nav-buttons>
    <ion-content class="has-header gradient">
        <div class="card pt0">
            <div class="item-text-wrap">
                <img ng-src="http://classifiedapp.com.hr/uploads/modules/classifiedoffer/{{offerbyID.image}}" />
                <div class="padding">
                    <h5>{{offerbyID.title}}</h5>
                    <p>{{offerbyID.content}}</p>
                </div>
            </div>
        </div>
        <br>
    </ion-content>
</ion-view>
```

Krajnji rezultat je prikaz korisniku (slika 5.5.)

Slika 5.5. Prikaz jednog oglasa

5.4 Priprema aplikacije za mobilne uređaje

Nakon što su napravljena i istestirana cijela aplikacija, izrada instalacijske datoteke za Android je relativno jednostavna.

Potrebno je pokrenuti CMD prozor unutar direktorija gdje se nalazi Ionic aplikacija. Prvo uredimo config.xml datoteku i postavimo verziju aplikacije, primjerice 0.0.1.

Zatim se izvršavaju redom slijedeće naredbe:

1. Dodavanje android platforme:

```
ionic platform add android
```

2. Debug konzola se briše:

```
cordova plugin rm org.apache.cordova.console
```

3. Izrada potpisnog ključa:

```
keytool -genkey -v -keystore malioglasi-release-key.keystore -alias malioglasi
-keyalg RSA -keysize 2048 -validity 10000
```

4. Kreiranje android verzije:

```
ionic build --release android
```

Sada je kreiran nepotpisana APK datoteka koja se nalazi u platforms/android/bin

5. Potrebno je i potpisati APK datoteku sa ključem koji smo kreirali

```
jarsigner -verbose -sigalg SHA1withRSA -digestalg SHA1 -keystore malioglasi-
release-key.keystore android-release-unsigned.apk malioglasi
```

6. Zadnji korak je optimizacija APK datoteku

```
zipalign -v 4 Malioglasi-release-unsigned.apk Malioglasi.apk
```

Aplikacija je u ovom trenutku gotova, i može se instalirati na uređaje sa Android operativnim sustavom, ili objaviti na Google Play trgovini aplikacija.

6. ZAKLJUČAK

Razvoj web i mobilnih platformi je došao do točke kada je razvoj aplikacija brz i jednostavan. Uz pomoć gotovih rješenja, kao što su Bootstrap i Laravel za web dio te Ionic za mobilni dio, čak i relativno kompleksne aplikacije se mogu napraviti u kratkom vremenu.

Laravel je PHP okvir koji slijedi MVC obrazac. MVC je omogućio da su prikazi podataka i njihova manipulacija odvojeni. Ovo se općenito smatra jako dobrim jer omogućuje da je kod modularan, da se može ponovno upotrijebiti i omogućuje više sučelja na kojima se može primjenjivati. Prednost razvoja aplikacija na Laravel PHP okviru (pa i ostalima) je to što već postoji veliki broj razvijenih gotovih komponenti.

Za razvoj mobilnog dijela rada korišten je Ionic okvir koji se i inače koristi za razvoj hibridnih mobilnih aplikacija. Ionic je jednostavan za upotrebu, ima jako puno gotovih komponenti i lagana je integracija sa vanjskim API-jem na kojeg se veže i dohvaća podatke.

Mobilna i web aplikacija čine jednu sinergiju, jer ono što se objavi na web dijelu se prikazuje i u mobilnoj aplikaciji. Razvoj ovih aplikacija je pomogao za bolje razumijevanje razvoja za web i mobilne platforme.

Na kraju je važno napomenuti da praktični dio ovog završnog rada, iako je funkcionalan sadrži osnove sa kojima se može prikazati demonstracija tehnologija te postoji mogućnost za daljnji razvoj i širi broj dodatnih mogućnosti koje se mogu integrirati.

7. LITERATURA

Knjige:

[1] J. Lockhart, Modern PHP: New Features and Good Practices, O'Reilly Media Inc., Sebastopol, 2015.

[2] L. Welling, L. Thomson, PHP and MySQL Web Development, Sams Publishing, Indiannapolis, 2005.

[3] M. Stauffer, Laravel: Up and Running: A Framework for Building Modern PHP Apps, O'Reilly Media Inc., Sebastopol, 2015.

[4] A. Ravulavaru, Learning Ionic - Build Hybrid Mobile Applications with HTML5, Pact Publishing, Birmigham, 2015.

[5] D. Rees, Laravel: Code Bright, Leanpub, 2014.

[6] K. Shotts, Mastering PhoneGap Mobile Application Development, PACKT Publishing, Birmigham, 2016

Članci:

[1] I. Malavolta, S. Ruberto, T. Soru, V. Terragni, Hybrid Mobile Apps in the Google Play Store: An Exploratory Investigation, Mobile Software Engineering and Systems (MOBILESoft), 2015 2nd ACM International Conference, 16. - 17. svibanj 2015.

[2] S. Aghaee, C. Pautasso, Mashup development with HTML5, Mashups '09/'10 3rd and 4th International Workshop on Web APIs and Services Mashups, Ayia Napa, Cipar — 01. prosinac 2010.

[3] N. Serrano, J. Hernantes, G. Gallardo, Mobile Web Apps, IEEE Software, vol. 30., 03. rujan 2013.

[4] Kako programirati za Android ili iOS, Bug, broj 252, studeni 2013.

Internetski izvori:

[1] MVC Xerox Parc – Internet http://heim.ifi.uio.no/~trygver/themes/mvc/mvc-index.html (01.06.2016)

[2] Models-Views-Controller, Trygve Reenskaug http://heim.ifi.uio.no/~trygver/1979/mvc-2/1979-12-MVC.pdf (01.06.2016)

[3] MVC Originals http://heim.ifi.uio.no/~trygver/2007/MVC_Originals.pdf (01.06.2016)

[4] JavaScript – JSP Pages http://www.kirkdorffer.com/jspspecs/jsp092.html (01.06.2016)

[5] Structure of Laravel Applications http://laravelbook.com/laravel-architecture/ (01.06.2016.)

[6] Structure od Ionic App http://mcgivery.com/structure-of-an-ionic-app/ (01.06.2016.)

[7] Controllers in Ionic Framework http://mcgivery.com/controllers-ionicangular/ (01.06.2016.)

[8] Ionic preface http://ionicframework.com/docs/guide/preface.html (01.06.2016.)

SAŽETAK

Ključne riječi

U svojim počecima, Internet i mobilne aplikacije su služile samo za prezentaciju informacija. Razvoj tehnologije i programskih jezika je došao do točke da se u relativno kratkom vremenskom roku može napraviti aplikacija koja se oslanja na udaljeni servis te radi na mobitelima i tabletima. Tako da oglasnik ponude i potražnje, umjesto samo u stvarnom svijetu, možemo imati i na internetu. Postoji nekoliko različitih načina izvedbe. U svojoj srži, aplikacija koja se nalazi na mobitelu i pokreće se je tzv. klijent dio, a mrežna stranica je poslužiteljska strana sustava. Cijeli sustav je koncipiran tako da postoji jednostrana komunikacija, i klijent šalje zahtjev za podacima na poslužiteljski dio. Poslužitelj vraća oblikovane podatke, i oni se prikazuju unutar aplikacije. Obje strane se sastoje od sličnog principa – imamo središnju funkcionalnost koja je kontroler, i koja dolazni zahtjev usmjerava prema dohvatu, obradi i slanju ili prikazu podataka. Princip rada se zove MVC obrazac. Poslužiteljska strana šalje podatke, a u klijentskom dijelu se ti podaci samo prikazuju. Poslužiteljska strana još ima i dodanu mogućnost da može i prikazati podatke. Konačni proizvod je sinergija klijentske i poslužiteljske aplikacije, koje koristeći moderne tehnologije i načine prezentacije i obrade podataka, čine jedan cjelovit proizvod.

Ključne riječi: oglasnik, hibridna aplikacija, MVC obrazac, poslužitelj, klijent

ABSTRACT

Keywords

Developing classified application for supply and demand of products

In their early years, Internet and mobile applications were capable only for presenting various data. During years, development of technology, and programming languages offered us to develop application in short time-span. In other words, we can develop not only classifieds in papers, we can have them in digital world too. There are several ways of accomplishing that task. In its core, there is mobile application, which is client side, and there is website which represents server-side of whole system. Idea behind this application is that there is one-way communication, meaning that client sends requests to the server. Server responds with formatted data, and they are shown inside application. Both sides consists of similar arhitecture – there is main functionality which is controller for collecting, and transfering requests. Controller communicates to database, and collects, edits, and shows data. This arhitecture is called MVC. Server application can send and show data, but client side can only show data. Final product is synergy between client and server side. By using modern technology and modern ways of presenting and processing data, they are one, whole product.

Keywords: classified, hybrid application, MVC paradigm, server, client

ŽIVOTOPIS

Nikola Papratović rođen je 22.12.1990. u Slavonskom Brodu. Završio je srednju školu gimnazija „A. G. Matoš“ u Đakovu i Fakultet elektrotehnike, računarstva i informacijskih tehnologija u Osijeku – stručni studij Elektrotehnike – smjer Informatika. Tokom 2011. radio je u 'HEP-ODS d.o.o. Elektroslavonija Osijek', Odjel poslovne informatike, Osijek, na poslovima sistemskog administratora. Tokom 2012. i 2013. radi u 'Hrvatski Telekom d.d.' kao agent tehničke podrške privatnim korisnicima. Tokom 2015. i 2016. radi u 'Betaware d.o.o.', Osijek, kao web dizajner. Od srpnja 2015. je član društva 'Culex d.o.o.' sa sjedištem u Grabovcu. Aktivno govori engleski i njemački jezik. Od tehničkih znanja i vještina, stručnjak je za web i mobilne tehnologije, i poznaje nekoliko programskih jezika.

PRILOZI

Popis slika:

Popis korištenih oznaka i kratica

engl.	Engleski
DBMS	Sustav za upravljanje bazama podataka (engl. Database Management System)
OOP	Objektno orijentirano programiranje (engl. Object Oriented Programming)
PHP	U početku Personal Home Page, kasnije PHP: Hypertext Preprocessor
MVC	engl. Model-View-Controller
PDO	PHP objekti podataka (engl. PHP Data Objects)
AJAX	Asinhroni JavaScript i XML (engl. Asynchronous JavaScript and XML)
SQL	Strukturirani upitni jezik (engl. Structured Query Langauge)
CRUD	Funkcionalnosti izrade, čitanja, ažuriranja i brisanja
HTTP	engl. HyperText Transfer Protocol
HTML	Prezentacijski jezik za izradu web stranica (engl. HyperText Markup Language)
CSS	Kaskadne stranice stilova (engl. Cascading Style Sheets)
XML	Jezik za proširivo označavanje podataka (engl. Extensible Markup Language)
URL	jedinstveni lokator resursa (engl. Uniform Resource Locator)
GUI	engl. Graphical User Interface
API	engl. Application Programming Interface