Web servis za rezervaciju termina temeljen na Javascript tehnologijama

Skolan, Marija

SVEUČILIŠTE U ZAGREBU
GRAFIČKI FAKULTET

MARIJA SKOLAN

WEB SERVIS ZA REZERVACIJU TERMINA TEMELJEN NA JAVASCRIPT TEHNOLOGIJAMA

DIPLOMSKI RAD

Zagreb, 2017.

Sveučilište u Zagrebu
Grafički fakultet

MARIJA SKOLAN

WEB SERVIS ZA REZERVACIJU TERMINA TEMELJEN NA JAVASCRIPT TEHNOLOGIJAMA

DIPLOMSKI RAD

Mentor:
Prof.dr.sc. Klaudio Pap

Student:
Marija Skolan

Zagreb, 2017.

Sažetak

Tema ovog diplomskog rada je istraživanje izrade web servisa za naručivanje klijenata sa multimedijalnim sadržajem. Web servis će se sastojati od odabira datuma i vremena željenog termina, mogućnosti stavljanja opisne fotografije i zagovaranja termina. Jedna od dodatnih opcija će biti slanje podsjetnika dan prije korisniku na zakazani pregled putem e-pošte. Istražiti će se mogućnost izrade logika aplikacije na strani poslužitelja pomoću Node.js JavaScript tehnologije kao i izrada sučelja na klijentskoj strani s jQuery Javascript bibliotekom čime bi se istražile pogodnosti i prednosti tih tehnologija za opisivanje i stiliziranje aplikacije u svim modernim internetskim preglednicima. Za potrebe ovoga rada napraviti će se baza podataka (temeljena na nekomercijalnim tehnologijama) koja će služiti za pohranu korisničkih podataka sa web servisa. Ova web aplikacija biti će namjenjena prvenstveno desktop računalima, ali ostaviti će se mogućnost da se uz male preinake koda aplikacija prilagodi i za druge uređaje. U izradi fokus će se staviti na programsku logiku i izradu funkcionalnosti koje će proširiti aplikaciju i to će činiti programski dio diplomskog rada. U drugom planu biti će dizajn korisničkog iskustva i dizajn sučelja kojim će se korisnik koristiti.

Sadržaj

1 Uvod

U drugom poglavlju biti će opisane tehnologije koje su se koristile za izradu ovog rada. Esperimentalni dio se bazira na JavaScript tehnologijama stoga će najviše biti riječi o platformama i alatima odgovornim za izvršavanje JavaScript koda. U prvom potpoglavlju biti će opisane glavne značajke internetskih aplikacija, čemu služe i od čega se sastoje. Drugo potpoglavlje biti će posvećeno internetskom poslužitelju. Opisati će se što je internetski poslužitelj, čemu služi i na koji način poslužitelj komunicira sa poslužiteljem baza podataka i klijentom. Kao dio poslužitelja opisati će se tehnologije i platforme korištene u eksperimentalnom dijelu rada: Node.js i njegova razvojna cjelina Express. Internetski klijent i njegove značajke obrađene su u trećem potpoglavlju. JavaScript je temeljna tehnologija korištena u eksperimentalnoj izradi stoga će se opisivati njegova funckija i mogućnosti na strani klijenta. HTML je opisni jezik korišten za izradu internetske aplikacije i opisati će se njegovi elementi, atributi i sintaksa kojom se koristi, a CSS je norma kojom se sve internetske aplikacije služe i opisat će se na koji način se korištenjem CSS može utjecati na izgled korisničkog sučelja. Ukratko će se opisati i Jade mehanizam za predloške i skup alata Bootstrap koji su također korišteni prilikom razvoja internetskog poslužitelja. Treće potpoglavlje usporediti će relacijske i nerelacijske baze podataka i navesti će primjere pisanja upita za obje vrste baza podataka. MongoDB baza podataka spada pod skup nerelacijskih baza podataka i koristiti će se u eksperimentalnom dijelu ovoga rada, pa će se opisati i njene značajke i prednosti.

Treće poglavlje opisuje ekspreimentalnu izradu internetske aplikacije. U prvom potpoglavlju opisati će se instalacija tehnologija potrebnih za eksperiment budući da računalo koje ima ulogu internetskog poslužitelja u ovom slučaju je i sami klijent. Sljedeće potpoglavlje opisivati će na koji je način strukturiran internetski poslužitelj korišten za izradu internetske aplikacije. Programiranje internetskog poslužitelja biti će podijeljeno u tri odlomka. Prvi će se odnositi na programiranje početne stranice aplikacije i njen prikaz, a drugi će opisivati kako je isprogramiranja forma za naručivanje i na koji način je u komunikaciji sa bazom podataka. Treći će odlomak opisivati na koji način su prikazani podatci zapisani u

bazi na internetskoj stranici. U drugom potpoglavlju opisati će se struktura korištene baze podataka i na koji način se može pristupiti bazi, a treće potpoglavlje opisuje programiranje korisničkog sučelja internetske aplikacije. Opisati će se početna stranica, od čega se sastoji i kako izgleda i na koji način je u nju uključen Google Maps API. U drugom odlomku opisati će se popis klijenata, njegova svrha, izgled i struktura opisnog koda u pozadini.

2 Teorijski dio

Potrebno je osvrnuti se na tehnologije koje su korištene za izradu ovog web servisa. JavaScript je skriptni jezik koji se koristi za dodavanje interaktivnosti i dinamičnosti web stranicama, a podržavaju ga svi poznatiji preglednici. Javno je raspoloživ što ga čini lako dostupnim i popularnim.

2.1 Internetske aplikacije

Web aplikacije su softverski programi koji se pokreću na internetskom poslužitelju. Web aplikacijama se pristupa putem internet pretraživača, za razliku od tradicionalnih desktop aplikacija koje se pokreću preko operacijskog sustava. Web aplikacije imaju nekoliko prednosti u odnosu na desktop aplikacije. Budući da se one pokreću unutar internetskog preglednika programeri ne moraju razvijati web aplikacija za više platformi, kao što je to slučaj sa desktop aplikacijama. Na primjer, aplikacija koja se pokreće unutar Chrome preglednika raditi će na Windows operativnom sustavu jednako kao i na OS X ili Linux operativnim sustavima. Također, nije potrebno distribuirati korisnicima nove verzije aplikacija sa novim nadogradnjama već je dovoljno nadograditi web aplikaciju na poslužitelju kojim se ona koristi i svi će korisnici imati pristup toj novoj verziji. Budući da se web aplikacije ne pokreću izravno sa operacijskih sustava, one imaju ograničen pristup sistemskim komponentama kao što su CPU (*eng.* Central processing unit), memorija i podatkovni sustav. Zahvaljujući tome, aplikacije visokih razina kao što su aplikacije za video produkciju i sl. izvoditi će se bolje kao desktop aplikacije. Web aplikacije su u potpunosti ovisne o internet pretraživaču. Ukoliko dođe do pogreške u radu internet pretraživača, vrlo je vjerojatno da će sav rad koji nije spremljen biti izgubljen. Probleme mogu stvarati i moguće nadogradnje pretraživača koje mogu biti nekompatibilne sa nekim aplikacijama i na taj način stvarati komplikacije [1].

Sve web aplikacije koriste neko računalo na kojem je spremljen njihov kod, ta računala su poznata kao poslužitelji. Web poslužitelji su zapravo samo računala

koja imaju instalirane određene softverske programe koji im omogućuju da 'posluže' internetske stranice (ili pošalju podatke korisniku kada on to zatraži). Kada se otvori internet pretraživač i upiše se URL (*eng.* Uniform Resource Locator) tada je taj zahtjev odmah poslan internetskom poslužitelju. Poslužitelj tada zapakira sve potrebne informacije i dostavlja ih internetskom pretraživaču koristeći HTTP (*eng.* Hypertext Transfer Protocol) protokol koji se koristi za sve internetski bazirane komunikacije [2]. Kada poslužitelj zaprimi zahtjev za slanjem regularne web stranice on pošalje web stranicu pretraživaču (statičke stranice). Web poslužitelj reagira drugačije kada je riječ o dinamičkim stranicama. U tom slučaju šalje stranicu posebnoj softverskoj ekstenziji koja je odgovorna za završavanje stranice. Taj posebni softver je poslužitelj aplikacija (*eng.* app server). Poslužitelj aplikacija čita kod na stranici i završava stranicu prema uputama u kodu, a onda uklanja kod iz stranice. Rezultat je opet statična stranica koju poslužitelj aplikacija pošalje natrag web poslužitelju koji onda šalje stranicu pretraživaču koji ju je zatražio. Ono što pretraživač na kraju dobije je kod napisan u HTML-u [3].

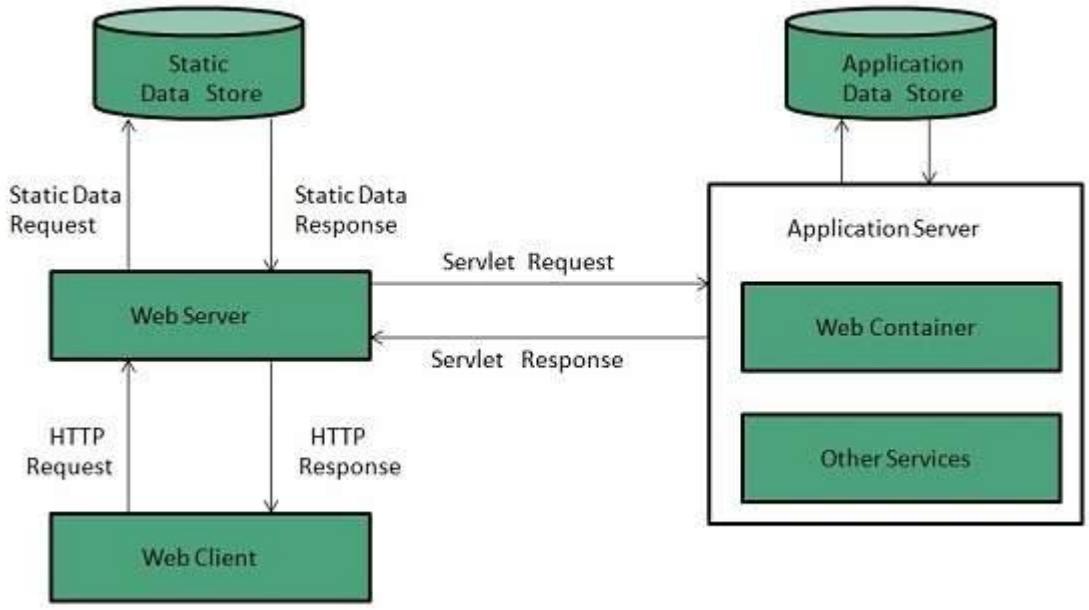
2.2 Tehnologije korištene za implementaciju poslužitelja web servisa za naručivanje

Internetski poslužitelj je sustav koji dostavlja sadržaj krajnjem korisniku putem Interneta. Internet poslužitelj se sastoji od fizičkog poslužitelja, poslužiteljskog operacijskog sustava (OS) i programa koji olakšava HTTP komunikaciju [4]. Poslužitelji najčešće sadrže internetske stranice, ali postoje i razni drugi poslužitelji (računalne igre, podatci, itd.)

Internetski poslužitelj odgovara na zahtjev klijenta na dva načina: slanjem datoteke klijentu koji je povezan sa zatraženim URL-om ili pokretanjem skripte i komuniciranjem sa bazom podataka koji onda generiraju odgovor.

Kada klijent pošalje zahtjev za internetskom stranicom, internetski poslužitelj traži tu stranicu i ako ju pronađe poslat će ju sa HTTP odgovorom. Ukoliko poslužitelj nije pronašao traženu stranicu on će tada poslati HTTP poruku. „Error 404 Not found“. Ukoliko je klijent zatražio i neke druge podatke, internetski

poslužitelj će tada kontaktirati poslužitelja aplikacija (*eng.* app server) i bazu podataka kako bi konstruirao točan HTTP odgovor [5].



Slika 1. *Način rada internetskog poslužitelja*

Neki od najpopularnijih internetskih poslužitelja danas su:

- Apache HTTP Server – najpopularniji poslužitelj. Razvila ga je tvrtka Apache Software Foundation. Ovaj poslužitelj je besplatan softver i može biti instaliran na gotovo sve operacijske sustave uključujući i Linux, UNIX i FreeBSD. Apache poslužitelj pokreće gotovo 60% internetskih poslužitelja danas
- Internet Information Services (IIS)- kreirao ga je Microsoft. Ovaj poslužitelj pokreće Windows NT/2000 i 2003 operacijski sustav. Budući da je usko povezan sa operacijskim sustavom to ga čini lakim za korištenje.
- Lighttpd – također besplatni internetski poslužitelj koji se distribuira sa FreeBSD operacisjkim sustavom. Ovaj besplatni poslužitelj je brz, siguran i zahtjeva malo CPU (*eng.* Central processing unit) snage.

2.2.1 Node.js

Node.js je platforma temeljena na besplatnim tehnologijama koja se koristi za egzekuciju JavaScript koda na poslužiteljskoj strani. Jezgra Node.js-a sastoji se od C/C++ koda u kojem je pisan i brze JavaScript implementacije koja jako brzo kompajlira JavaScript kod prije izrvšavanja osnovnom strojnom kodu. Osim toga, Node.js pokriva i integrirane module kao npr. HTTP modul kako bi 'ugostio' (*eng.* host) internetskog poslužitelja. Dodatni moduli lako mogu biti dodatno instalirani sa 'svakodnevno korištenim' NPM-om (Node Package Manager). Korištenjem Node-a moguće je osigurati internetske aplikacije visokih performansi koje mogu komunicirati sa internetskim pretraživačima (uz pomoć WebSocket konekcije) u stvarnom vremenu. Budući da moderni internetski pretraživači koriste JavaScript, baš kao i Node, kod može biti dostavljen objema stranama (klijentska i poslužiteljska) i dijelom korišten zajedno. Naravno, potrebno je napomenuti da je uporabom Node-a potrebno znati samo jedan programski jezik kako bi se izradila internetska aplikacija [6].

Također važna komponenta Node-a su baze podataka. Ne iznenađuje činjenica da su dostupna korisnička sučelja za neke od najpoznatijih baza podataka (MySQL, MariaDB, PostgreSQL, Oracle, SQL Server), ali prednost Node-a je ta da ima razvijen pristup i drugačijim vrstama baza podataka, NoSQL bazama podataka. Tako Node zajedno sa Expressom, Angularom i MongoDB-om (NoSQL) čini skup kompatibilnih komponenti , takozvani MEAN (Mongo, Express, Angular, Node) [7].

Node.js koristi npm upravitelj paketima, njegove dvije glavne odgovornosti su instaliranje paketa i upravljanje dodatnim alatima. Budući da je npm brz, sposoban i jednostavan upravitelj, on je dijelom zaslužan za rast popularnosti Node-a.

Filozofija koja stoji iza Node-a je programiranje eventima (*eng.* event-driven). Kao primjer se može navesti implementacija korisničkog sučelja, kada korisnik nešto klikne ili odabere tada se 'okine' definirani događaj. To bi značilo da programer nema kontrolu nad time kada i hoće li korisnik nešto kliknuti što čini takav način programiranja intuitivnim.

```
var http = require('http');

http.createServer(function(req,res){
        res.writeHead(200, { 'Content-Type': 'text/plain' });
        res.end('Hello world!');
}).listen(3000);

console.log('Server started on localhost:3000; press Ctrl-C to terminate....');
```

Slika 2. *'Hello World' poruka*

Primjer koda koji ne poslužuje HTML već samo odašilje poruku 'Hello World' pregledniku kao običan tekst. U ovom slučaju događaj koji je postavljen je implicitan, događaj koji se treba izvršiti je HTTP zahtjev. Metoda *http.createServer* zaprima funkciju kao argument koja će biti pozvana svaki puta kada se postavi HTTP zahtjev. Ovaj jednostavni program postavlja sadržaj kao jednostavni čisti tekst i šalje *string* 'Hello World'.

2.2.2 Express

Node.js u pravilu koristi Express razvojnu cjelinu koji je podloga za razne ostale Node razvojne cjeline. Express osigurava konfiguraciju portova korištenih za konekciju i definira lokacije predložaka korištenih za generiranje poruka. Sam Express je minimalističan, ali programeri su razvili kompatibilne dodatke za rješavanje skoro svake programerske komplikacije. Tako se mogu pronaći biblioteke za rad sa 'kolačićima', sesijama, URL parametrima, POST porukama i još mnogo mogućnosti. Za Express razvojnu cjelinu bi se moglo reći da je popularna u zajednici programera što je iznimno važno jer to je pokazatelj da li će se i dalje raditi na poboljšanju i održavanju te tehnologije. Dostupnost dokumentacije, nadopunjavanje biblioteke i tehnička podrška također ovise veličini zajednice koja ju koristi. Express također spada u skupinu neodređenih (*eng.* unopinionated) razvojnih cjelina što znači da takve cjeline imaju manje pravila kako komponente složiti zajedno da bi se ostvario cilj, ne postoji 'pravi način'. Programerima je takve cjeline lakše koristiti i ukomponirati najbolje alate kako bi se izvršio neki zadatak iako to znači da do tih rješenja treba doći samostalno.

Mehanizam za predloške omogućava da se koriste datoteke koje služe kao predlošci u aplikaciji. Tijekom svog rada, mehanizam za predloške zamjenjuje

varijable u datoteci koja služi kao predložak sa stvarnim vrijednostima i pretvara predložak u HTML datoteku koja se potom šalje klijentu. Ovakav pristup omogućava da se lakše izradi HTML stranica.

Neki od popularnih mehanizama za predloške s kojima Express može raditi su Pug (u starijim verzijama zvan Jade), Mustache i EJS. Express generator aplikacija koristi Jade kao unaprijed postavljeni mehanizam, ali također podržava i nekoliko ostalih mehanizama.

Kako bi Express mogao koristiti mehanizme za predloške potrebno mu je konfigurirati putanju na kojoj se nalaze predlošci kao i tip mehanizma za predloške koji se želi koristiti, npr. Pug.

2.3 Tehnologije korištene za implementaciju klijenta web servisa za naručivanje

Internetski klijent je aplikacija koje komunicira sa internetskim poslužiteljem. Klijent u svojoj komunikaciji sa poslužiteljom koristi Hypertext Transfer Protocol (HTTP). To je protokol koji stoji iza World Wide Web-a (www).HTTP stoji iza svakog zahtjeva za internetskim dokumentom ili multimedijskim sadržajem.

Klijentske Web tehnologije su one koje omogućuju prikaz sadržaja klijentskim mašinama. Za razliku od serverskih Web tehnologija, mnogo je veći naglasak na Web standardima, ponajviše zbog toga što je prilikom odabira serverske tehnologije sloboda skoro potpuna sve dok je na izlazu HTML, a kao ulaz se koristi standardno sučelje prema bazi podataka. S druge strane, sve što znamo o klijentu je da koristi preglednik, koji prikazuje sadržaj zavisno o standardima. Za standardizaciju Web formata zadužen je W3C.

Klijentske tehnologije mogu se grubo podeliti na:

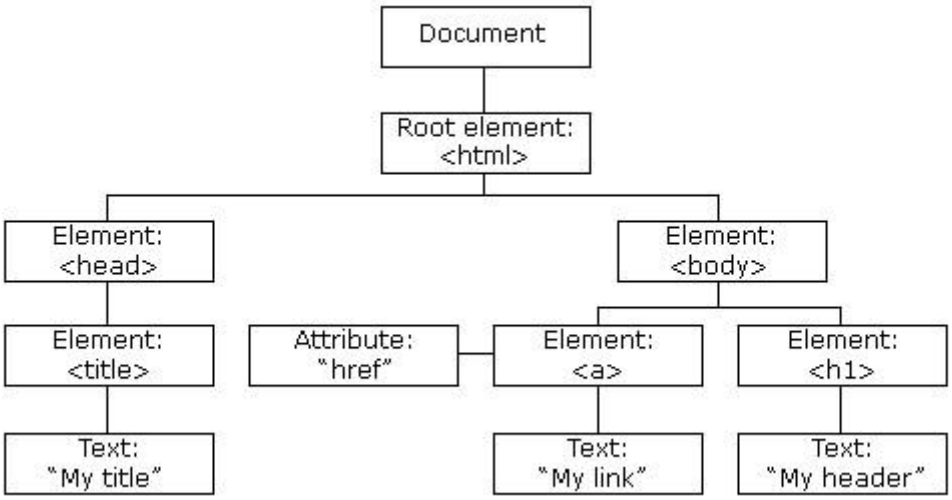
- opise sadržaja i prezentacije (HTML, XHTML, XML, CSS…)
- skriptne jezike (JavaScript, Jscript, VBScript…)
- DOM (Document Object Model)
- dodatne objekte (npr. slikovni objekti – GIF, JPG, PNG datoteke)

2.3.1 JavaScript

JavaScript se pokreće na klijentskoj strani i koristi se za definiranje ponašanje stranice u interakciji sa korisnikom, ali zahvaljujući Node-u JavaScript je postao i jezik poslužitelja. Javascript je skriptni jezik ugrađen u preglednik. Sintakse su inspirirane programskim jezicima C i Java. Omogućuje dinamičke promjene elemenata stranice i reagiranje na događaje. Najvažnije je da se JavaScript izvodi unutar preglednika *Weba* i to na svim novijim preglednicima, ali pri tome ima neka sigurnosna ograničenja. Jedno od njih je da se ne može pisati po disku (osim kolačića) i to se naziva *Sandbox.* JavaScript se interpretira, a ne prevodi.

JavaScript se može uključiti kao i CSS direktno u HTML i to u zaglavlje dokumenta ili u tijelo dokumenta i to se definira pomoću <script> oznake. Osim na taj način JavaScript se može uključiti i kao vanjska datoteka čije se funkcionalnosti onda pozivaju prilikom korištenja HTML stranice. Prednosti prilikom uključivanja vanjskih datoteka su da se mogu skripte spremiti u *cache* i time se brže učitavaju stranice i osim toga ista skripta se može uključivati i na drugim domenama, time se radi podloga za različite *web* API-je.

Ono što JavaScriptu daje moć je mogućnost da dinamički mijenja sadržaj i strukturiranost HTML dokumenta po modelu DOM *(Document Object Model).* DOM zapravo predstavlja organiziranu strukturu dokumenta u obliku stabla [8].



Slika 3. *Prikaz DOM strukture HTML-a*

JavaScript u takvoj strukturi može:

- Promijeniti sve HTML elemente na stranici
- Promijeniti sve HTML atribute na stranici
- Promijeniti sve CSS stilove na stranici
- Ukloniti sve postojeće HTML elemente i atribute
- Dodati novi HTML element i atribut
- Reagirati na sve postojeće HTML događaje na stranici
- Kreirati novi HTML događaj na stranici

2.3.2 HTML

Osnova svake *web* stranice je html. Html je kratica koja dolazi od HyperText Markup Language što predstavlja najrašireniji i najpoznatiji jezik za opis sadržaja na *Webu*. Jedan html dokument predstavlja zapravo jednu *web* stranicu.

Osnovu informacija na *Webu* predstavlja hipertekst. Umjesto da se linearno čita tekst, skače se s jednog dokumenta na drugi koji detaljnije obrađuje određenu temu, zatim na novu temu itd.

Prilikom prikaza stranice pregledniku su najvažnije oznake da zna kako prikazati pojedine cjeline, a kako bi također znao koju verziju HTML koristi potrebno je to i u početku dokumenta naznačiti, tj. po kojoj gramatici (DTD) će se to tumačiti. Oznake su omeđene šiljastim zagradama <...> i najčešće dolaze u parovima, npr. i pri čemu <u> predstavlja početnu oznaku (početak djelovanja imenovane oznake), a </u> predstavlja završnu oznaku (prestanak djelovanja imenovane oznake). Tekst između početne i završne oznake predstavlja sadržaj oznake.

Svaki označni jezik pa tako i HTML sadrži sljedeće ključne komponente:

- elementi
- atributi
- tipovi podataka
- DTD *(Document Type Definition)*

Elementi definiraju „strukturu" dokumenta. Elementi se sastoje od dva osnovna dijela, a to su atributi i sadržaj, a neki elementi nemaju sadržaj kao npr.
. Neki primjeri elemenata mogu biti strukturne oznake (opisuju svrhu odnosno funkciju teksta) kao što je oznaka za naslov <h4>, prezentacijske oznake (opisuju izgled teksta) kao što je oznaka za tekst u kurzivu <i> te oznake hiperteksta (link na druge dokumente) kao što je <a href="http://www.google.com">.

Atributi predstavljaju svojstvo elemenata. Atributi su zapravo parovi ime="vrijednost" i pišu se unutar početne oznake elementa. Primjeri najčešćih atributa u HTML-u su id (jedinstveni identifikator elementa), class (grupiranje sličnih elemenata), style (stil prikazivanja). Primjer korištenja atributa može izgledati ovako:

<p id="prvi" class="druga" style="color:red;">

HTML dokument se sastoji od 2 glavna dijela: zaglavlje i tijelo. U zaglavlju se najčešće koriste oznake: title (naslov dokumenta), link (link na vanjski dokument), meta (meta informacije), base (osnovna lokacija za linkove u

dokumentu), style (definira specifični CSS za dokument), script (opisuje upotrebu skripata).

Sadržaj HTML datoteke je tekst koji možemo podijeliti na dio koji predstavlja informaciju i dio (kombinacija znakova unutar <...>) koji predstavlja oznaku (upute). Položaj i broj praznih mjesta u datoteci je nevažan pa tekst možemo pisati u slobodnom obliku. Osnovni moduli (skupovi oznaka) vezani su za:

- tekst
- veze
- slike
- liste
- tablice
- obrasce

2.3.3 CSS

CSS je norma *Weba* definirana od strane W3C-a. Svrha CSS-a je vizualizacija sadržaja opisanog HTML-om, ali je sintaksa različita od HTML-a.

CSS olakšava dizajn i redizajn, tj. odvajanje sadržaja i prezentacije. Jedna datoteka za izgled i raspored više *web* stranica. Također je omogućeno da se promjeni stil svim elementima istog razreda odjednom. Novi i detaljniji atributi se mogu koristiti za razliku od onog što nudi HTML. Uz to kod upotrebe CSS-a su performanse bolje i manja je količina HTML koda, a vanjski CSS-ovi se spremaju u *(cache)* na klijentu. U današnje vrijeme važno je također da se stranica može prikazati odgovarajuće na različitim uređajima kao što su *smartphoneovi* i *tableti* a to se upravo može postići upotrebom CSS-a koji omogućuje različite stilove za različite uređaje, a što je važno jer personalizacijom sadržaja se privlače korisnici.

Nedostaci koji se mogu javiti prilikom korištenja CSS su da različiti preglednici različito implementiraju neka pravila CSSa (ionako je to više problem preglednika,

kao što je Internet Explorer) te slabo ili uopće nije podržan od starijih inačica *web* preglednika.

Sam CSS se može koristiti na dva načina i to izvan dokumenta što predstavlja zasebnu datoteku referenciranu iz dokumenta ili unutar samo dokument gdje može biti integriran u zaglavlje HTML dokumenta *(Embedded styles)* ili umetnut izravno u "style" atribut nekog elementa *(Inline styles).* Najbolji izbor je kada se ima vanjski dokument koji se može pozivati u više *web* stranica s time da se vanjske datoteke CSS-a označavaju sa .css.

Sintaksa CSS-a zapravo predstavlja skup pravila (definicija) koja govore kako pojedini elementi u dokumentu trebaju biti prikazani. Svaku definiciju tvore selektor (HTML element koji se uređuje) i deklaracijski blok naveden u vitičastim zagradama. Deklaracijski blok se može sastojati od jedne ili više deklaracija, a deklaraciju čine naziv svojstva i vrijednosti koja mu se želi dodijeliti: selektor {svojstvo: vrijednost; svojstvo: vrijednost;...Primjer: h4 {color: red; font-size: 12px;}, pri čemu je h4 selektor, a color i font-size su svojstva, a red i 12px su vrijednosti koja ta svojstva poprimaju.

Ako se želi utjecati samo na neke elemente onda ih se grupira u razrede pomoću atributa 'class'. Stil jednog razreda može se definirati npr: .korisniLinkovi {color: red; font-weight: bold;}. Kod identifikatora je važno da se može dodijeliti samo jednom elementu u dokumentu (id oznaka u atributima elementa) i onda CSS bi izgledao ovako:

#navigacija { backgorund-color: black; color: #FF00CC; width: 15;}

Prilikom selektiranja elementa koji je potrebno selektirati može se selektirati elemente nekog tipa ili samo one koji imaju postavljen određeni atribut [9].

2.3.4 Jade

Jade je mehanizam za predloške s visokim performansama na koji je jako utjecao Haml i impementirano je s JavaScriptom za Node.js i preglednike. Jade je čista sintaksa osjetljiva na korištenje praznina pomoću kojih se piše HTML [10].

2.3.5 Bootstrap

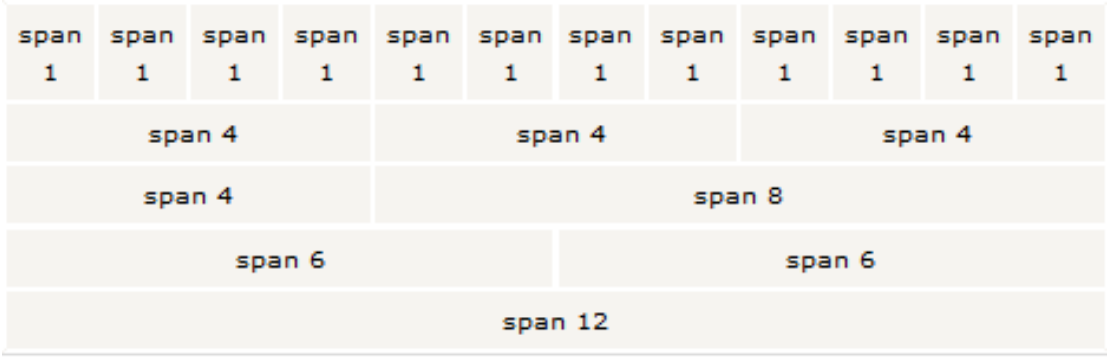
Bootstrap je skup alata otvorenog koda koji se koristi prilikom razvoja responzivnih projekata na Webu koristeći HTML, CSS i Javascript.

Bootstrap se može raslojiti na 3 glavne komponente:

- bootstrap.css – skup CSS funkcionalnosti
- bootstrap.js – skup Javacript/jQuery funkcionalnosti
- glyphicons – set ikona i fontova

Osim ove tri komponente, Bootstrap također koristi jQuery kako bi sve komponente mogle funkcionirati. jQuery je jako popularna i široko korištena JavaScript funkcionalnost koja pojednostavljuje i dodaje kompatibilnost JavaScripta na različitim preglednicima.

Mreža (*grid*) je vjerojatno jedan od najvažnijih aspekata Bootstrap funkcionalnosti. Ona je temelj na kojoj je kreiran cijeli raspored. Mreža omogućava do 12 stupaca kroz čitavu stranicu. Ukoliko se ne želi upotrijebiti svih 12 stupaca zasebno može ih se grupirati zajedno u šire stupce:



Slika 4. *Prikaz mreže za raspored elemenata u Bootstrap-u*

Bootstrapov mrežni sustav je responzivan i stupci će promijeniti raspored ovisno o veličini ekrana. Na velikom ekranu može izgledati bolje ukoliko je sadržaj organiziran u 3 stupca, ali na manjim ekranima može biti bolje ukoliko elementi od kojih je sastavljena stranica budu naslagani jedan povrh drugoga. Zbroj stupaca unutar jednog reda mora biti 12, ukoliko bi zbroj prelazio tu vrijednost onda će se stupci slagati jedan povrh drugoga bez obzira na veličinu ekrana.

Bootstrapov mrežni sustav se sastoji od 4 skupine:

- xs (za telefone)
- sm (za tablete)
- md (za srednje *desktop* ekrane)
- lg (za velike *desktop* ekrane)

Gore navedene skupine se mogu kombinirati kako bi se mogli kreirati dinamičniji i fleksibilniji rasporedi stranica.

Osim same mreže Bootstrapov temeljni CSS će također omogućiti dodavanje korisnih stilova, formi, tablica, gumbova, listi i slika, kao i potpuno funkcionalne navigacijske trake, dok će Javascript kompomenta Bootstrapa omogućiti dodavanje korisnog koda za kreiranje skočnih prozora, padajućih izbornika i drugih interaktivnih web elemenata [11].

Velika prednost Boostrapa je upravo njegova otvorenost i činjenica da se puno ljudi koristi njime kako bi izradila svoje web stranice. Osim kreiranje stranica pomoću Bootstrapa od početka mogu se koristiti i gotovi predlošci koji se mogu besplatno preuzeti te ih uređivati po vlastitim željama. Na taj način se uvelike olakšava izrada stranica i još više širi upotreba Boostrapa. Neki predlošci se mogu vidjeti na slijedećem povezniku: https://startbootstrap.com/template-categories/all/ [12].

2.4 Baze podataka

U svijetu baza podataka postoje dvije vrste baza: SQL i NoSQL (relacijske i nerelacijske baze podataka). Iako su realcijske baze podataka danas još uvijek prevladavajući standrad za spremanje različitih zbirki podataka, u praksi se iz različitih razloga pojavljuju njihove alternative (NoSQL baze podataka). Te su alternative često po svojim karakteristikama dijametralno suprotne relacijskim bazama podataka. Ono što ih čini različitima je način na koji su izgrađene, tipovi spremljenih podataka i način pohrane.

2.4.1 Usporedba relacijske i nerelacijske baze podataka

Kod relacijskih baza podataka podaci su organizirani u tablice na kojima su definirani primarni i vanjski ključevi, za brže pretraživanje podataka koriste se indeksi, a između tablica u bazi podataka uspostavljene su relacije. U dobro organiziranoj relacijskoj bazi podataka svi podatci bi trebali biti u takozvanom normaliziranom obliku. Kod NoSQL baza podataka podaci u bazi podataka mogu biti dobro organizirani na različite načine ovisno o tome kakvu vrstu podataka podržava konkretni sustav za upravljanje bazom podataka. Neke od poznatih implementacija NoSQL baza su: Document Databases, Graph Databases, Columnar Databases i ostale.

Standardni jezik upita na relacijskim bazama podataka je SQL (Structured Query Language), premda pojam 'standardni' treba uzeti sa određenom dozom rezerve. Iako postoje definirani standardi što bi SQL trebao podržavati, obično svaki proizvođač relacijskih baza podataka dodaje vlastita proširenja kako bi njegov sustav što djelotvornije mogao rješiti problem iz prakse. Tu prestaje priča o potpunoj standardizaciji i kompatibilnosti. Primjer SQL upita:

```
SELECT user_id,status
FROM users
WHERE status = "A"
```

Primjer NoSQL upita:

```
Db.users.find(
{status:"A"},
{user_id:1, status:1, _id:0})
```

Jedna od najvažnijih karakteristika relacijskih baza podataka je provjera i sprječavanje upisa neispravnih/nepotpunih podataka u bazu podataka. Za to postoje raližiti mehanizmi poput relacija, ograničenja postavljenih na pojedine stupce tablica, primarnih i jedinstvenih ključeva, transakcija i slično. Iako nabrojene mogućnosti bitno unaprjeđuju ispravnost podataka upisanih u bazu podataka, istovremeno zahtjevaju dosta resursa tijekom korištenja (često

nepotrebno). Kod NoSQL baza podataka dozvoljen je znatno slobodniji način upisa podataka pa sam sustav upravljanje bazom podataka ne brine previše o njihovoj ispravnosti, ali zato troši manje resursa od relacijskog modela. Slobodniji način zapisa podataka istovremeno omogućava spremanje vrlo kompleksnih oblika nestrukturiranih podataka, često nepogodnih za čuvanje u relacijskom obliku.

Iako suvremeni sustavi za upravljanje relacijskim bazama podataka omogućuju upis dosta velike količine podataka uz istovremeno korištenje većeg broja procesora pa čak i računala za njihovu obradu, upravo zbog manje striktne provjere neispravnih i nepotpunih podataka, NoSQL baze podataka su u pravilu jednostavnije i podržavaju distribuiranu obradu podataka, što je posebno važno kod ogromne količine podataka [13].

2.4.2 MongoDB baza podataka

MongoDB pripada skupinu NoSQL baza podataka. MongoDB (DB označava bazu podataka, eng. *data base*) je besplatan i *open-source* program koji može raditi na više različitih platformi (linux, windows i dr.) te se bazira na pohrani dokumenata (*document-oriented database*).

Glavne fukncionalnosti MognoDB-a su:

- Fleksibilni dokumenti slični JSON strukturi sa odgovarajućim shemama se koriste za pohranu podataka što omogoćuva da se polja koja se sadrže informaciju mogu varirati od dokumenta do dokumenta te se sama strukura podataka može promjeniti tijekom vremena.
- Modeli dokumenata mapiraju objekte u kod aplikacije čineći podatke jednostavnim za korištenje
- 'Ad hoc' upiti, indeksiranje i sakupljanje u stvarnom vremenu omogućuju razne načine pristupu i analizi podataka
- MongoDB je besplana baza podataka dostupna pod licencom GNU Affero General Public License [14]

3 Eksperimentalni dio

Izrađena internetska aplikacija se sastoji od dvije stranica od kojih glavna sadrži glavne informacije o tvrtki, a druga omogućava pregled rezervacija. Dio aplikacije je i forma za rezervaciju termina koja se nalazi na glavnoj stranici. Forma funkcionira tako da se upiše ime i prezime, kontakt broj, komentar i po želji je moguće dodati fotografiju. Aplikacija također ima implementiran Google Maps API.

3.1 Instalacija kostura Node.js, alata npm i MongoDB baze podataka

Prije izrade same internetske aplikacije bilo je potrebno instalirati tehnologije koje će biti potrebne za izradu poslužiteljskog dijela. Prvi je na redu bio Node koji je zadužen za izvođenje JavaScript koda na poslužiteljskoj strani. Instalacija Node-a je vrlo jednostavna, prvo je potrebno otići na službenu stranicu Node.js-a i skinuti instalacijski paket. Stranica detektira za koji je operacijski sustav potrebna instalacija (u ovom slučaju: Windows 10) i skine instaler. Njega je samo potrebno pokrenuti na računalu i instalacija će se izvršiti sama. Na taj način je moguće instalirati Node i NPM (Node Package Manager) koji je jednako važan budući da omogućava lakše i brže korištenje Node-a. Također treba odrediti i direktorij u kojem će se nalaziti internet aplikacija, u Windows-ovom terminalu (command prompt) pomoću *cd* naredbe se odabire željeni direktorij.

Sljedeći je korak instalacija Express-a. Express je taj koji je zaslužan za to da se Node ponaša kao poslužiteljska aplikacija na kakvu su svi naviknuli. Potrebno je instalirati Express-Generator koji zapravo nije sami Express već konstrukcijska aplikacija koja kreira kostur za stranice pogonjene Express-om. Instalacija se radi preko terminala unošenjem naredbi:

```
C:\node>npm install -g express-generator
```

Ovaj generator bi se trebao instalirati automatski budući da je smješten unutar NPM direktorija

Kreiranje Express projekta je jednostavno. Potrebno je odrediti direktorij u kojem bi se projekt trebao nalaziti i pomoću naredbe stvoriti projekt:

C:\node>express nodetest1

Kada se pokrene ova naredba računalo će u svom terminalu ispisati koji su sve direktoriji kreirani i njihove putanje. Upravo je to zadaća Express Generatora.

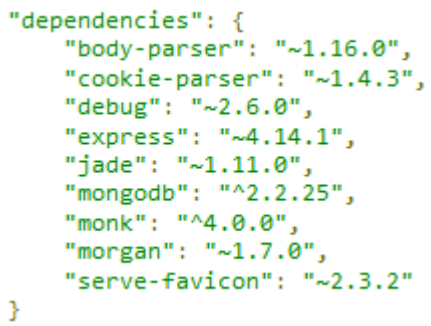
```
C:\node>express nodetest1
   create : nodetest1
   create : nodetest1/package.json
   create : nodetest1/app.js
   create : nodetest1/public
   create : nodetest1/public/javascripts
   create : nodetest1/public/images
   create : nodetest1/public/stylesheets
   create : nodetest1/public/stylesheets/style.css
   create : nodetest1/routes
   create : nodetest1/routes/index.js
   create : nodetest1/routes/users.js
   create : nodetest1/views
   create : nodetest1/views/index.jade
   create : nodetest1/views/layout.jade
   create : nodetest1/views/error.jade
   create : nodetest1/bin
   create : nodetest1/bin/www

   install dependencies:
    > cd nodetest1 && npm install

   run the app:
    > SET DEBUG=nodetest1b:* & npm start
```

Slika 5. *Ispis svih kreiranih direktorija koji su dio Express-a*

Kada je aplikacija izrađena ona ima svoj kostur i ima instalirane određene dodatke. Jade uređivač nije potrebno dodatno instalirati, ali MongoDB i Monk se moraju ručno dodati. Prilikom instalacije express-generator je kreirao dokument package.json unutar nodetest1 direktorija. Taj dokument je potrebno otvoriti u uređivaču teksta i ručno upisati pomoćne resurse (eng. *dependencies*) za Mongo i Monk:



Slika 6. *Popis svih potrebnih dodataka projekt*

Nakon definiranja željenih dodataka potrebno ih je i instalirati. U Windows trerminalu je potrebno pokrenuti naredbu *npm install* unutar direktorija gdje se nalazi projekt:

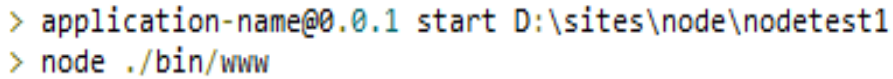
C:\node\nodetest1>npm install

Računalo će pročitati *json* dokument i instalirati sve navedene željene pomoćne resurse uključujući Express (express generator je instaliran, ali je potrebno instalirati modul unutar određenog projekta). Nakon što NPM odradi svoj zadatak, kreirati će direktorij *node_modules* koji sadrži sve dodatke koji su potrebni za ovaj projekt. Za instaliranje baze potrebno je izraditi i novi direktorij unutar *nodetest1* projekta koji će se zvati *data.* To je važno jer ukoliko taj direktorij ne postoji, poslužitelj baze podataka će se srušiti prilikom pokušavanja upisa ili čitanja iz baze.

Kada je poslužiteljska aplikacija instalirana, potrebno ju je još samo pokrenuti. Aplikacija se pokreće iz direktorija u kojem se projekt nalazi naredbom:

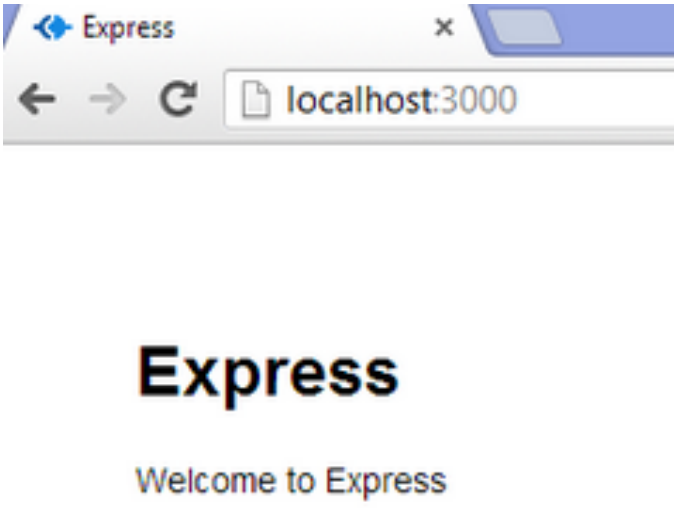
C:\node\nodetest1>npm start

Konsola bi trebala ispisati putanju i ime projekta.



Slika 7. *Ispis Node konzole prilikom pokretanja poslužitelja*

Ova aplikacija se pokreće na portu 3000.



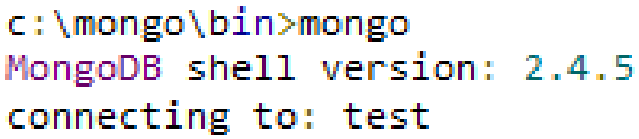
Slika 8. *Pokrenut NodeJS internetski poslužitelj na portu 3000*

MongoDB bazu podataka je potrebno prvo skinuti i instalirati lokalno. Ona će služiti kao poslužiteljska baza podataka. Instalacijski paket za Mongo se može pronaći na službenim stranicama MongoDB-a. Na stranici se može pronaći MSI datoteka koju je potrebno pokrenuti kroz standradnu Windows instalaciju. Mongo će se po pravilu instalirati u direktorij Program Files, ali za potrebe ove aplikacije je lokacija promjenjena u C:\Mongo datoteku. Nije potrebno puno pažnje tome pridodavati jer je Mongo memorijski malo zahtjevan, a podatke će spremati u *nodetest1* direktoriju. Još je samo potebno kopirati datoteke u *bin* datoteci i zaljepiti ih u direktorij u kojem će Mongo 'živjeti', u ovom slučaju je to *data* datoteka unutar *nodetest1* direktorija. Nakon instalacije, Mongo je potrebno i pokrenuti. U Windows terminalu treba navigirati do *bin* direktorija i iz tog direktorija pokrenuti mongod. Mongod je primarni *deamon* proces za MongoDB sustav. On rukuje sa zahtjevima podataka, kontrolira pristup podatcima i izvodi pozadinske kontrolne operacije [15]. Naredba za pokretanje mongod-a:

```
mongod --dbpath c:\node\nodetest1\data\
```

Windows terminal pokazuje da je Mongo poslužitelj pokrenut. Ukoliko se on pokreće po prvi puta, to će trajati malo dulje vremena u usporedbi sa svakim sljedećim pokretanjem budući da mora prelocirati slobodan prostor i odraditi nekoliko drugih zadataka. Kada Mongo vrati poruku: [initandlisten] waiting for connections on port 27017 znači da je uspješno pokrenut. Nakon toga potrebno je još pokrenuti i Mongo bazu koja se koritsti u aplikaciji. Otvaranjem drugog

command prompt se navigira u Mongo instalacijski direktorij *bin* i upisuje se naredba *mongo*. Mongo konsola će tada ispisati koja je verzija pokrenuti i lokaciju.



Slika 9. *Ispis Mongo konsole prilikom pokretanja naredbe mongo*

Ovim postupkom je MongoDB pokrenut i povezan sa poslužiteljem.

3.2 Poslužitelj web servisa za naručivanje

U ovom poglavlju biti će opisana poslužiteljska strana, od čega se sastoji i način na koji je izvedena. Opisati će se i struktura direktorija i datoteka koje čine kostur poslužitelja.

3.2.1 Konfiguracija pozadinske logike

Za izradu ove aplikacije odabran je Notepad++ tekstualni uređivač. U ovom slučaju je tekstualni uređivač odabran prema osobnim referencama, nema pravila koji bi se uređivač trebao koristiti. Prvi izbor uređivača je bio Atom, ali uslijed otežanog pokretanja i određenih *bugova* se odustalo od njegovog korištenja.

Jezgra aplikacije je zapravo sadržana u datoteci *app.js.* Ona kreira JavaScript varijable i povezuje ih sa određenim paketima, pomoćnim resursima, node funkcionalnostima i rutama. U ovoj konfiguraciji su rute kombinacija modela i kontrolera, one usmjeravaju promet i ujedno sadrže dio programske logike. To je sve kreirao Express prilikom kreiranja projekta.

```
var express = require('express');
var path = require('path');
var favicon = require('serve-favicon');
var logger = require('morgan');
var cookieParser = require('cookie-parser');
var bodyParser = require('body-parser');

var mongo = require('mongodb');
var monk = require('monk');
var db = monk('localhost:27017/appointment');

var index = require('./routes/index');
var users = require('./routes/users');

var app = express();
```

Slika 10. *Kreirane JavaScript varijable u app.js datoteci*

Označena varijabla *app* je izrazito važna. Ona instancira Express i dodijeljuje joj varijablu *app*. Ova datoteka također govori koji će se *engine* koristiti za ispis html koda, u ovom slučaju je to Jade, i poziva metode koje dohvaćaju i pokreću tu konfiguraciju. Datoteka također govori Express-u da posluži statične objekte iz /public/dir, ali da ih prikaže kao da dolaze od razine iznad. Primjer: direktorij sa slikama je smješten u c:\node\nodetest1\public\images, ali pristupa mu se putem http://localhost:3000/images

```
var formidable = require('formidable');

// view engine setup
app.set('views', path.join(__dirname, 'views'));
app.set('view engine', 'jade');

// uncomment after placing your favicon in /public
//app.use(favicon(path.join(__dirname, 'public', 'favicon.ico')));
app.use(logger('dev'));
app.use(bodyParser.json());
app.use(bodyParser.urlencoded({ extended: false }));
app.use(cookieParser());
app.use(express.static(path.join(__dirname, 'public')));

// Make our db accessible to our router
app.use(function(req,res,next){
    req.db = db;
    next();
});
```

Slika 11. *Dio postavljene konfiguracije za ispis stranica*

U aplikaciji su također postavljeni upravljači greškama koji su korisni za razvoj i produkciju. Tijekom razvoja aplikacije ispisi grešaka će dati više informacija o tome što se dogodilo.

```
// catch 404 and forward to error handler
app.use(function(req, res, next) {
  var err = new Error('Not Found');
  err.status = 404;
  next(err);
});

// error handler
app.use(function(err, req, res, next) {
  // set locals, only providing error in development
  res.locals.message = err.message;
  res.locals.error = req.app.get('env') === 'development' ? err : {};

  // render the error page
  res.status(err.status || 500);
  res.render('error');
});
```

Slika 12. *Provjera nekih mogućih grešaka prilikom ispisa stranica*

Error 404 je jedna od najučestalijih grešaka, ona govori da izvor na traženoj lokaciji nije pronađen. Događa se najčešće kada je krivo imenovan *view* u ruti ili kada on uopće ne postoji u projektu. Jezgra Node-a je ta da zapravo svi moduli eksportiraju objekt koji može lako biti pozvan kasnije u kodu. Tako glavna aplikacija u projektu eksportira svoj *app* objekt.

```
module.exports = app;
```

Slika 13. *Stvaranje objekta app*

3.2.2 Programiranje početne stranice

Express kostur je već definirao varijablu *routes* koja pokazuje u *index.js* dokument. U tom dokumentu se definiraju željene rute i HTTP poruke. Dvije najučestalije HTTP metode su GET i POST. GET dohvaća podatke iz specificiranog izvora, a POST prenosi podatke specificiranom izvoru. U ovom slučaju izvor je Mongo baza podataka. HEAD, DELETE, PUT su još neke od korištenijih HTTP metoda [16].

GET metodom se poziva početna stranica (Home page) koja dohvaća podatke i prikazuje ih klijentu. Metoda poziva *layout.jade* dokument koji sadrži HTML kod.

```
/* GET Layout page. */
router.get('/layout', function(req, res) {
    console.log('Someone made a request jade!');
    res.render('layout', { title: 'Layout!' });
});
```

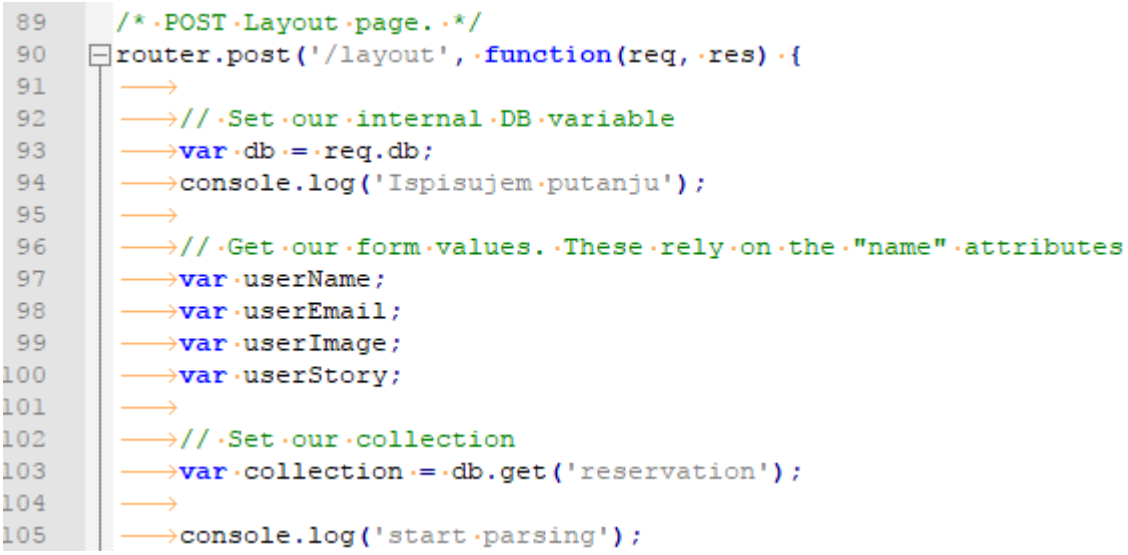
Slika 14. *GET metoda koja prikazuje početnu stranicu*

Za definiranje route u parametrima rute potrebno je specifcirati lokaciju dokumenta kojega je potrebno renderirati. U prvoj liniji koda se navigira rutu da iz trenutnog direktorija (*views* direktorij) pokazuje na *layout* dokument. Metoda *console.log* nije krucijalna za izradu ove aplikacije. Ta metoda ispisuje običnu jednostavnu poruku unutar *loga*, takve poruke su korisne kod većih aplikacija kako bi se lakše pratili pozadinski procesi i lakše otklonili eventualni kvarovi.

res. metode mogu poslati odgovor (eng. *response*) klijentu i završiti zahtjev-odgovor ciklus. Ako ni jedna takva metoda nije pozvana iz rukovoditelja rutom, klijentski zahtjev ostati će 'visjeti' i neće biti dovršen što znači da korisnik na kraju neće vidjeti nikakvu stranicu [17]. Metoda koja je korištena je *res.render* koja renderira HTML predložak. U ovom slučaju će renderirati *layout.jade* dokument. Kreiranje GET metode za renderiranje stranice bez dodatnih zahtjeva je poprilično jednostavno, a Node nudi pregršt *res.* metoda.

3.2.3 Programiranje forme za naručivanje

Budući da korisnik putem forme za naručivanje može poslati neku poruku (upisati u bazu) GET metoda ne može izvršiti tu funkciju. Za to je potrebno implementirati POST metodu koja definira po kojim uvjetima će se podatci upisivati u bazu podataka. Ova POST metoda je nešto kompleksnija od obične GET metode za renderiranje stranice jer ima za zadatak povezati se sa bazom podataka i po njenim uvjetima upisivati podatke u bazu. Ovdje korištena POST metoda također locira dokument i otvara funkciju *function* sa parametrima *req* i *res*.

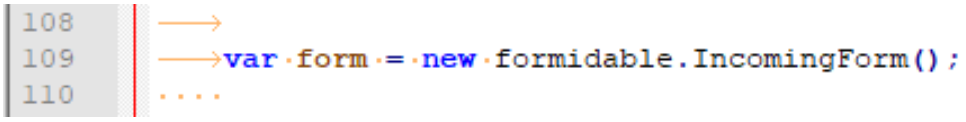


Slika 15. *Kreirane varijable korištene u metodi*

Ova funkcija sadrži lokalne varijable što znači da one 'postoje' samo unutar te funkcije i ne mogu biti korištene izvan njene domene. Varijabla *db* sprema zahtjev za povezivanjem sa bazom podataka i podatcima koji se nalaze unutar *nodetest1* direktorija. Cilj spremanja zahtjeva u varijable je lakše korištenje kasnije u kodu. Podatci koji se unose u formu na strani klijenta su spremljeni unutar atributa *name* koji se dodjeljuje formi unutar HTML koda. Na taj način je moguće povezati pozadinsku logiku i podatke unesene na strani klijenta. Ti su podatci spremljeni u istoimene varijable kako bi se kasnije lakše mogli koristiti.

Budući da se nerelacijska baza, za razliku od relacijskih, ne sastoji od više tablice već kao ekvivalent tome koristi kolekcije i njih je potrebno definirati. Metoda *db.get()* povezuje funkciju sa definiranom kolekcijom u koju će se upisivati podatci. U ovom slučaju to je kolekcija *reservation*. Ovaj komad koda također upisuje u *logove* usputne poruke koje je programer odredio kako bi se poslije olakšala detekcija grešaka i kako bi se znali na kojem koraku je program stao.

Za potrebe ove aplikacije dodatno je instaliran modul Formidable koji ne dolazi u početnoj instalaciji projekta. To je modul Node.js-a za parsiranje podataka unutar forme, uključujući i postavljanje datoteka na poslužitelj (eng. *upload*). Express-formidable je nešto poput mosta između Express-a i Formidable-a. U ovoj aplikaciji je korišten upravo za to, za *upload* podataka (slika) na računalo.

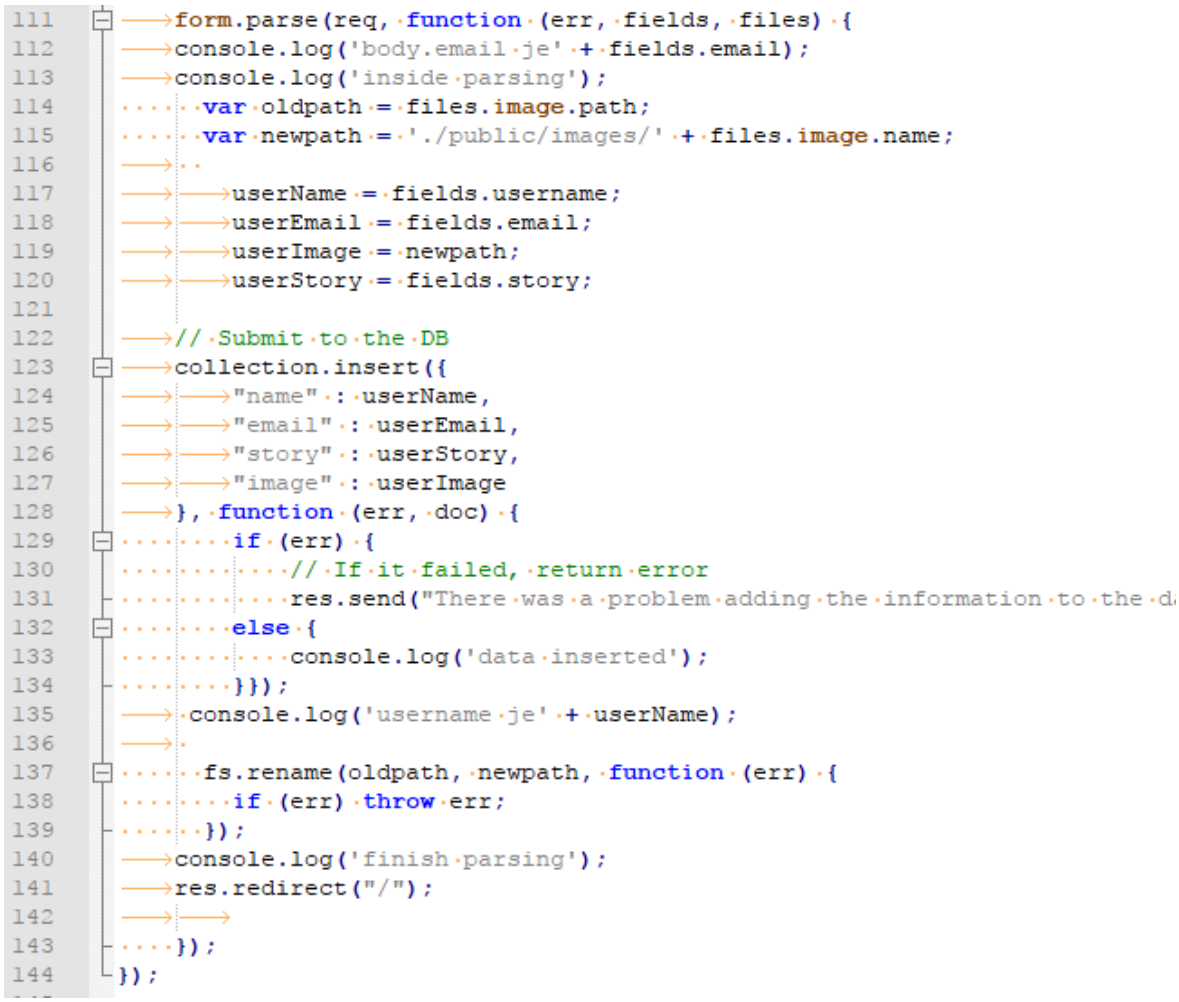


```
108
109     var form = new formidable.IncomingForm();
110
```

Slika 16. *Kreiranje varijable za Formidable metodu za parsiranje*

Kreirana je varijabla *form* koja sprema dohvaćene podatke iz HTML forme za unos. Ti podatci će dalje u kodu biti korišteni za parsiranje i spremanje.

Ostatak POST metode je zapravo pozivanje izvršavanja *parse()* metode nad *form* varijablom koja ima spremljene dohvaćene podatke unešene na strani klijenta.



```
111     form.parse(req, function (err, fields, files) {
112     console.log('body.email je' + fields.email);
113     console.log('inside parsing');
114         var oldpath = files.image.path;
115         var newpath = './public/images/' + files.image.name;
116     
117         userName = fields.username;
118         userEmail = fields.email;
119         userImage = newpath;
120         userStory = fields.story;
121
122     // Submit to the DB
123     collection.insert({
124         "name" : userName,
125         "email" : userEmail,
126         "story" : userStory,
127         "image" : userImage
128     }, function (err, doc) {
129         if (err) {
130             // If it failed, return error
131             res.send("There was a problem adding the information to the d
132         else {
133             console.log('data inserted');
134         }});
135     console.log('username je' + userName);
136     
137         fs.rename(oldpath, newpath, function (err) {
138         if (err) throw err;
139         });
140     console.log('finish parsing');
141     res.redirect("/");
142     
143     });
144 });
```

Slika 17. *Formidable metoda parse()*

Metoda *parse* zaprima dva argumenta, prvi je *req* dok je drugi *callback* funkcija koja će se pozvati kada se *parse* funkcija izvrši. Proslijeđena *callback* funkcija prima uobičajeno tekstualno polje u *fields* argumentu i snima slike u željenu datoteku koja je data u obliku putanje argumentu *files*. Varijable *oldpath* i *newpath*

spremaju staru putanju slike koja se sprema i novu putanju na koju će se spremiti. Kao nova putanja odabran je direktorij *images* koji je unutar *public* direktorija, a naredbom *files.image.name* dodano mu je i ime slike koja se sprema. Ta nova putanja je spremljena u novu varijablu *userImage* kao i ostali podatci. Naredbom *fields* se dohvaćaju podatci sukladno sa imenovanim *name* atributom u HTML kodu. Tako se polje za unos imena na klijentskoj strani kojem *name="username"* dohvaća i nadopunjava *userName* varijablu ovom linijom koda: userName = fields.username;. Na taj način su dohvaćena i ostala polja za unos podataka: *email, story*.

Slijedeće na redu je upisivanje podataka u Mongo bazu. Naredba za upisivanje u nerelacijsku bazu podataka je *insert()*. Ona upisuje jedan dokument ili niz od više njih (eng. *array*) njih u Mongo bazu podataka. Ta funkcija može zaprimati tri argumenta, ali u ovom slučaju korištena su dva. Prvi je argument *docs* koji se prosljeđuje u obliku niza podataka koji su dohvaćeni sa klijentske strane, a drugi argument je definiranje *callback* funkcije koja provjerava da li je došlo do greške. Ona sadrži jednu if-else petlju koja kao uvijet uzima *err*. Ukoliko je *err=true* tj. ukoliko dođe do pogreške i podatci ne budu upisani klijentu će se prikazati poruka „There was a problem adding the information to the database.“.U ostalim slučajevima će program u konzolu ispisati „data inserted“. Kolekcija u koju će se upisivati je ona koja je prethodno definirana, kolekcija *reservation*.

Funkcija *rename* poziva se nad imenom slike, ona kao argument zaprima staro ime slike i novo ime slike, a kao treći argument dobiva *callback* funkciju. Ta funkcija samo provjerava sa jednostavnom *if* petljom da li je došlo do pogreške i ukoliko je došlo do pogreške JavaScript će stati i generirati poruku o pogrešci. JavaScript će izbaciti iznimku (*throw an error*) i kreirati će *Error* objekt koji ima dva svojstva: ime i poruku.

Na kraju parsiranja se poziva *redirect* funkcija. Ta funkcija ima dva parametra. Prvi je parametar status, to je pozitivan cijeli broj koji odgovara HTTP statusu, ukoliko taj argument nije definiran (kao u ovom slučaju) onda će se status automatski postaviti na '302 Found'. Kao drugi parametar ima putanju (eng. *path*) gdje mu je u ovom slučaju ostavljena ista putanja kako bi ta funkcija učitala ponovno istu stranicu. Korisnik to vidi kao ponovno učitavanje iste početne stranice i mogućnost

da napravi novi upis u bazu. Time je završeno parsiranje unešenih podataka, a kolekcija *reservation* ima novi zapis.

3.2.4 Programiranje liste klijenata

Statična stranica koja ispisuje popis klijenata u obliku tablice zapravo ispisuje sve podatke unutar kolekcije, bez iznimke. U pozadini je to još jedna GET poruka koja se u ovom slučaju prvo povezuje sa poslužiteljem baze podataka.

```
145
146   /* GET Client List page. */
147   router.get('/clientlist', function(req, res) {
148       var db = req.db;
149       var collection = db.get('reservation');
150       console.log(collection);
151       collection.find({},{},function(e,docs){
152           console.log(docs);
153           res.render('clientlist',{
154           "clientlist" : docs
155           });
156       });
157   });
158
```

Slika 18. *GET metoda korištena za ispis spremljenih podataka u bazi*

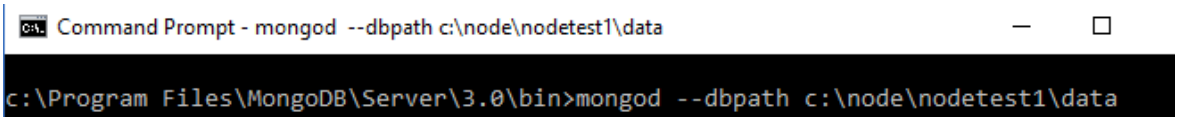
Unutar ove metode je ponovno potrebno dohvatiti bazu koja se koristi, tako da je ona spremljena u *db* varijablu. Isto je potrebno napraviti sa kolekcijom kojoj se želi pristupiti (kolekcija *reservation*). Funkcija koja dohvaća podatke iz nerelacijske baze je *collection.find*, ona kreira kursor za upit u bazu koji može bit korišten za iteriranje rezultata iz Mongo baze. Prvi parametar joj je neki selektor ili upit koji je u ovom slučaju prazan. Drugi parametar se odnosi na polja koja se žele ispisati (eng. *fields*), u ovom slučaju je taj parametar također ostao prazan jer je cilj ispisati sve podatke koji su spremljeni, a ova aplikacija koristi bazu koja ima samo jednu kolekciju što znači da nije potrebno specificirati koju kolekciju se želi pozvati. Kao treći parametar funkcija *find* uzima *callback* parametar koji ima oblik funkcije koja će se pozvati nakon izvršavanje metode. Kao *callback* parametar se prosljeđuje i definira anonimna funkcija sa dva nova parametra. Prvi je parametar *e* koji će sadržavati *Error* objekt ukoliko je došlo do pogreške, a drugi parametar sadrži rezultat iz *find* metode ili vrijednost *null* ukoliko je došlo do greške. Parametar *docs* je tako kursor kojeg je kreirala *find* metoda, ali se dalje prosljeđuje kao parametar.

Pozivanjem funkcije *res.render* prosljeđuju joj se argumenti koje će ta funkcija koristiti, a to su ime dokumenta (*view*) koji će se renderirati i podatci koji su spremljeni u kursoru (*docs*) kojem je dan ključ (*key value*) *clientlist*.

3.3 Struktura korištene baze podataka

Mongo baza prilikom instaliranja nema stvorenu nikakvu bazu. Ukoliko se napravi upis podataka ručno kroz komandni prozor tada će Mongo kreirati *data* bazu podataka i u nju će spremiti taj novi zapis. Ali moguće je kreirati personaliziranu bazu. Jedna od prednosti Mongo baze podataka je ta da koristi JSON strukturu podataka koja je lako razumljiva ljudima.

Mongo baza se u ovoj aplikaciji pokreće iz Mongo konzole. Prije svakog korištenja baze potrebno je navigirati Mongu putanju u kojoj se nalaze podatci.



Slika 19. *Pokretanje Mongo poslužitelja*

Na taj se način pokreće Mongo poslužitelj. Potrebno je još otvoriti novu Mongo konzolu i navigirati u Mongo instalacijski direktorij i naredbom mongo upogoniti bazu.

```
c:\Program Files\MongoDB\Server\3.0\bin>mongo
MongoDB shell version: 3.0.15-9-g0513806
connecting to: test
Server has startup warnings:
2017-08-26T12:10:51.403+0200 I CONTROL  [initandlisten]
2017-08-26T12:10:51.403+0200 I CONTROL  [initandlisten] ** NOTE: This
is a 32 bit MongoDB binary.
2017-08-26T12:10:51.408+0200 I CONTROL  [initandlisten] **       32 bi
t builds are limited to less than 2GB of data (or less with --journal)
.
2017-08-26T12:10:51.410+0200 I CONTROL  [initandlisten] **       Note
that journaling defaults to off for 32 bit and is currently off.
2017-08-26T12:10:51.412+0200 I CONTROL  [initandlisten] **       See h
ttp://dochub.mongodb.org/core/32bit
2017-08-26T12:10:51.413+0200 I CONTROL  [initandlisten]
>
```

Slika 20. *Naredba mongo i njen rezultat*

Budući da je moguće imati više baza, aplikaciji je potrebno dati informaciju koju bazu će koristiti. To je napravljeno u *app.js* dokumentu.

```
8   var mongo = require('mongodb');
9   var monk = require('monk');
10  var db = monk('localhost:27017/appointment');
```

Slika 21. *Definiranje korištene baze*

Na liniji 10 poziva se *monk* funkcija. Monk je mali sloj koji omogućava jednostavna poboljšanja za korištenje MongoDB baze unutar Node.js. ona kao parametre ima koji se lokalni domaćin (eng. *local host*) koristi i ime baze koja se koristi. U ovom slučaju koristi se baza *appointment*, a korišteni port je 27017. Baze sadržavaju jednu ili više kolekcija. Korištena baza koristi već spomenutu *reservation* kolekciju. Kao dio ovoga rada napravljeni su API-i za upisivanje i čitanje iz baze, ali ona ima puno više drugih mogućnosti. Bazi se može pristupiti i kroz konzolu i isto tako joj prosljeđivati upite.

Putem konzole MongoDB baze moguće je vidjeti cijelu strukturu baza podataka. Naredbom use DatabaseName se navigira u željenu bazu. Naredbom show collections mogu se ispisati sve postojeće kolekcije unutar baze.

```
> show collections
reservation
system.indexes
>
```

Slika 22. *Popis postojećih kolekcija*

Ova baza trenutno ima samo jednu kolekciju, to je ona koja se koristi za potrebe ove aplikacije. Naredbom db.reservation.find().pretty() se mogu ispisati svi podatci koji su spremljeni u u toj kolekciji. Metoda *find()* je ta koja vraća kursor sa podatcima, a metoda *pretty()* te podatke čini lakše čitljivima (razdvaja ih u redove).

Slika 23. *Rezultat koji vraća find() metoda*

Ova kolekcija trenutno ima četiri dokumenta. Dokumenti u MongoDB bazi su složeni kao parovi polja-vrijednosti (eng. *field-and-value*), njihova struktura izgleda ovako:

```
{
    field1: value1,
    field2: value2,
    field3: value3,
    ...
    fieldN: valueN
}
```

Vrijednost polja može biti bilo koji BSON (binarni zapis JSON dokumenta, sadržava više tipova podataka u odnosu na JSON) tip podatka, uključujući neki drugi dokument, niz i niz dokumenata.

Ime polja je tip stringa. Tip stringa ima neke restrikcije:

- Ime polja nemože započeti znakom '$'
- Ime polja nemože sadržavati znak '.'
- Ime polja nemože sadržavati redoslijed znakova 'null'
- Ime polja _id je rezervirano za korištenje primarnog ključa, vrijednost tog polja mora biti jedinstvena unutar kolekcije, nepromjenjiva i može biti bilo koji tip podatka osim niza

BSON dokumenti mogu imati više polja jednakog imena iako većina Mongo sučelja to ne dozvoljava. U ovoj aplikaciji ta opcija nije korištena.

U Mongo bazi podataka svaki dokument pohranjen u kolekciji zahtjeva svoj jedinstveni _id koji se ponaša kao primarni ključ. Ukoliko dokument koji se upisuje u bazu nema definirano to polje, MongoDB *driver* će automatski generirati ObjectId za _id polje. ObjectId su mali, jedinstveni, posloženi i brzi za generiranje. Sastoje se od 12 bitnih vrijednosti gdje prva četiri bita sadržavaju informaciju kada je ObjectId kreiran.

Prilikom upisa dokumenata u ovoj kolekciji nije zadana vrijednost _id polja, pa ju je Mongo poslužitelj sam kreirao. Struktura dokumenata ove kolekcije je:

```
{
    "_id": ObjectId,
    "name": string,
    "email": string,
    "story": string,
    "image": string
}
```

Podatci se mogu u Mongo konzoli dodati i ručno. Mongo naredba za dodavanje jednog dokumenta je db.collection.insertOne(). Ukoliko se ručno ne unese _id polje, Mongo će ga kreirati automatski.

```
> db.reservation.insert({name: "Test4", email: "test4@mail"
, story: "Poruka 4", image: "./public/images/index4.jpg"})
WriteResult({ "nInserted" : 1 })
>
```

Slika 24. *Mongo metoda insert()*

Mongo naredba za brisanje dokumenata je *remove().* Moguće je naredbom db.reservation.remove({}) izbrisati sve dokumente unutar te kolekcije.

```
> db.reservation.remove({})
WriteResult({ "nRemoved" : 3 })
```

Slika 25. *Mongo metoda remove()*

Metodama je moguće proslijediti uvijet koji moraju ispuniti prilikom izvođenja. Ukoliko se želi pronaći neki dokument koji zadovoljava neki kriterij onda je potrebno npr. metodi find( ) kao parametar zadati neki upit (eng. *query*). Upit je opcionalan, a specificira filter koji koristi operator. Primjer jednog upita je db.reservation.find({name: 'Test1'}).pretty(). Ovaj upit traži dokumente u kojima je vrijednost *name* polja jednaka stringu "Test1". U ovoj kolekciji postoji samo jedan takav dokument:

```
> db.reservation.find({ name: "Test1"}).pretty()
{
        "_id" : ObjectId("59a060fdb672171cdc03b78e"),
        "name" : "Test1",
        "email" : "test1@mail",
        "story" : "Poruka 1",
        "image" : "./public/images/index2.jpg"
}
>
```

Slika 26. *Rezultat find() metode sa zadanim argumentom*

Dokumente u kolekciji je moguće i urediti (eng. *update*) naknadno, nakon što je zapis već napravljen. To izvršava metoda *update()*. Ona modificira već postojeći dokument u kolekciji ili više njih. Metoda može modificirati specifično polje

dokumenta ili zamjeniti postojeći dokument u potpunosti, ovisno koji je argument zadan *update* parametru. Metoda *update()* ima tri parametra:

- Uvjet
- Pomijena
- Opcije

```
> db.reservation.update({name: "Test 3"}, {imag
e: "./images/index4.jpg", name: "Test3", email:
 "test3@mail", story: "Poruka 3"})
WriteResult({ "nMatched" : 1, "nUpserted" : 0,
"nModified" : 1 })
```

Slika 27. *Primjer update() metode*

Update() metoda na slici 27 mjenja vrijednosti unutar dokumenta za kojeg je postavljen uvjet da je name: 'Test 3'. Kao drugi argument zadan je niz polja kojima se vrijednost mijenja:

image: './images/index4.jpg'

name: 'Test3'

email: 'test3@mail'

story: 'Poruka 3

'

Nakon izvršene naredbe, Mongo u svojoj konzoli vraća povratnu informaciju da je midificiran 1 dokument ('nModified' : 1).

3.4 Implementacija klijenta web servisa za naručivanje

Klijentska strana se sastoji od dvije stranice. Jedna dinamična koja ima ulogu početne stranice i jedna statična koja ima ulogu samo prikazati popis svih klijenata koji su zapisani u bazi. Početna stranica je napravljena kao *single page* internetska stranica. Njena svrha je da omogući korisniku rezervaciju termina za određenu uslugu, predstavi tvrtku u nekoliko kratkih informacija i da prikaže njenu

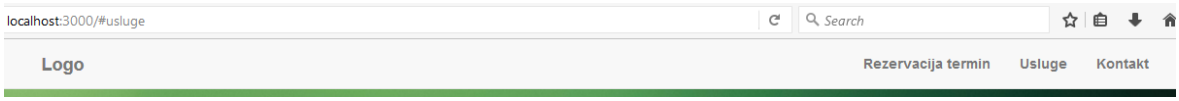
lokaciju kao i kontakt broj. Namjena statične stranice je da omogući tvrtci da pregleda sve dosadašnje zapise poredane kronološki.

3.4.1 Početna stranica

Početna stranica ima tri dijela:

- Rezervacija termina
- Usluge
- Kontakt

Budući da je ova aplikacija *single page* internetska stranica, sve su kategorije smještene na jednoj stranici i može im se pristupiti pomicanjem *scroll* gumba na mišu ili odabirom kategorija u izborniku.



Slika 28. *Prikaz izbornika*

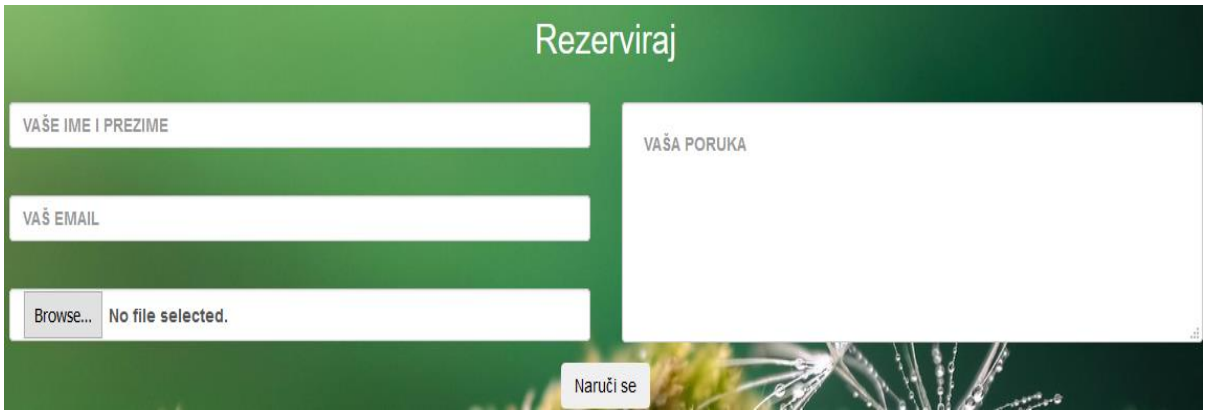
Opcija sa odabirom kategorije klikom na miš je napravljena tako da su stavljene poveznice na određene dokumente. Svaka kategorija je spremljena u svoju formu i njoj je dan atribut name=" ", a u izborniku je ta određena sekcija pozvana.

```
55        <div class="collapse navbar-collapse" id="bs-example-navbar-collapse-1">
56          <ul class="nav navbar-nav navbar-right">
57            <li>
58              <a href="#narucivanje">Rezervacija termin</a>
59            </li>
60            <li>
61              <a href="#usluge">Usluge</a>
62            </li>
63            <li>
64              <a href="#kontakt">Kontakt</a>
65            </li>
66          </ul>
67        </div>
```

Slika 29. *Implementacija navigacijske trake*

Izbornik je napravljen kao niz ugniježdenih *div* elemenata kojima su pozvane predefinirane Bootstrap klase koje olakšavaju uređivanje stranica. Dio njih je i

jedna neodređena lista (<*ul*>) sa nekoliko elemenata. Unutar svakog retka liste zadan je *href* atribute. On specificira lokaciju linka, u ovom slučaju destinacija je samo drugi *div* element unutar istog dokumenta. Lokacija može biti zadana kao absolutni ili relativni URL ili može biti zadan i neki drugi protokol (https://, ftp://,...) ili čak neka skripta (npr. href="javascript:alert('Hello');").



Slika 30. *Forma za rezervaciju termina*

Sekcija „Rezerviraj“ čini glavni dio ove aplikacije. To je jedan *form* element ugnježden unutar nekoliko *div* elemenata. Sastoji se od četiri polja:

- Unos imena i prezimena
- Unos kontakt email-a
- Slanje fotografije
- Poruka ili komentar

```
<form name="sentMessage" id="contactForm" novalidate method="post" action=
    <div class="row">
        <div class="col-md-6">
            <div class="">
                <input type="text" name="username" class="form-control" pl
                <p class="help-block text-danger" style="height:3%"></p>
            </div>
            <div class="">
                <input type="text" name="email" class="form-control" placel
                <p class="help-block text-danger" style="height:3%"></p>
            </div>
            <div class="" style="height:6%">
                <input type="file" name="image" style="height:100%" class=
                <p class="help-block text-danger"></p>
            </div>
        </div>
        <div class="col-md-6">
            <div class="form-group">
                <textarea name="story" class="form-control" placeholder="V
                <p class="help-block text-danger"></p>
            </div>
        </div>
        <div class="clearfix"></div>
        <div class="col-lg-12 text-center">
            <div id="success" style="padding: 0px 12px"></div>
            <button type="submit" class="btn btn-xl">Naruči se</button>
        </div>
    </div>
</form>
```

Slika 31. *Impementacija forme za rezervaciju*

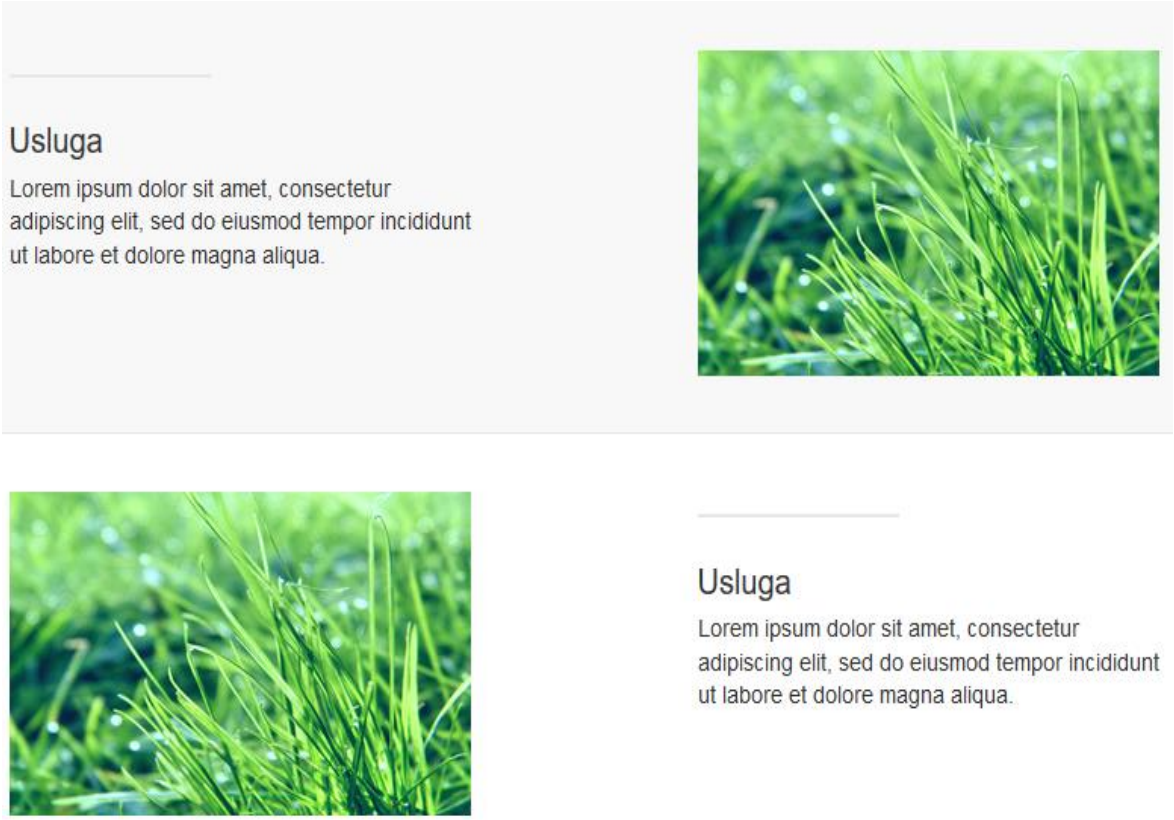
Svako polje za unos je *input* tip elementa kojem se mogu zadati oblici dokumenata kakvi će se unositi. On specificira polje unosa gdje korisnik može unijeti neku vrstu podatka. *Input* elementi se koriste unutar *form* elemenata kako bi se deklarirale kontrole koje dopuštaju korisniku da unese podatke. Polje za unos može varirati na mnoge načine, ovisno koji tip atributa zaprima. Sa oznakom *type* mu se definira koji oblik podataka može zaprimati. Unutar forme za naručivanje koriste se dva tipa *input* elemenata: *text* i *file*. Tip elementa *text* omogućava korisniku da unese neki string kao tip podatka dok *file* tip dopušta odabir nekog dokumenta koji je spremljen lokalno.

Unutar ove forme korišten je i element *textarea*. On omogućuje kontrolu teksta u više redaka. Ovaj element može sadržavati neograničen broj znakova (uvjetno

rečeno), a tekst se renderira veličinom fonta koji je fiksno definiran (obično Courier fontom). Veličina polja za unos teksta može biti specificirana atributima za kolone i redove ili eksplicitno: definiranjem visine i širine unutar CSS-a.

Svim poljima za unos dodan je atribut *name*. Atribut *name* može biti korišten kao referenca na neki element unutar JavaScript funkcije, kao referenca kada su podatci predani i čak može sadržavati meta podatke. Ovaj atribut je moguće koristiti kao atribut koji će spremiti vrijednosti polja za unos, kao što je slučaj u ovoj formi. Tim atributima kasnije pristupa poslužitelj kada te podatke prosljeđuje poslužitelju baze. Na taj je način moguće poslati podatke na poslužitelj koristeći jednostavne oznake.

Sekcija Usluge ima svrhu predstaviti tvrtku u nekoliko točaka, to mogu biti usluge koje ta tvrtka pruža, ciljevi koje tvrtka ima ili nešto drugo. Trenutno postoje dvije takve sekcije koje opisuju tvrtku u nekoliko točaka, a brzom i jednostavnom nadogradnjom je moguće dobiti još nekoliko sekcija.



Slika 32. *Sekcija sa informacijama o tvrtki*

Jedna sekcija se sastoji od *p* i *h2* elementa koji su ugnježđeni unutar *div* elemenata. Element *p* definira paragraf, a internetski pretraživači automatski dodaju margine prije i nakon svakog *p* elementa, a margine se mogu modificirati unutar CSS dokumenta. Element *h* se koristi za definiranje HTML naslova (eng. *headings*). Element *h* ima raspon od 1 (*h1*) do 6 (*h6*) gdje je *h1* tip naslova najvažniji, a *h6* tip naslova najmanje važan tip naslova. Element *div* definira odjeljak ili sekciju unutar HTML dokumenta, on se koristi kako bi se grupirali blokovi elemenata i lakše uređivali koristeći CSS.

```
    <div class="container">
      <div class="row">
        <div class="col-lg-5 col-sm-6">
          <hr class="section-heading-spacer">
          <div class="clearfix"></div>
          <h2 class="">Usluga</h2>
          <p class="lead">Lorem ipsum dolor sit amet, cons
          </div>
          <div class="col-lg-5 col-lg-offset-2 col-sm-6">
            <img class="img-responsive" src="/images/ipad.
          </div>
      </div>

    </div>
    <!-- /.container -->

</div>
<!-- /.content-section-a -->

<div class="content-section-b">
    <div class="container">
      <div class="row">
        <div class="col-lg-5 col-lg-offset-1 col-sm-push-6
          <hr class="section-heading-spacer">
          <div class="clearfix"></div>
          <h2 class="">Usluga</h2>
          <p class="lead">Lorem ipsum dolor sit amet, cons
        </div>
        <div class="col-lg-5 col-sm-pull-6  col-sm-6">
          <img class="img-responsive" src="/images/dog.png'
        </div>
      </div>
    </div>
```

Slika 33. *Izvedba sekcije za informacije*

Div elementima koji služe kao okvir jednog odjeljka dana je klasa *containter*. Bootstrap zahtjeva da se elementi zamotaju i prilagode sistemu mreže. Klasa *container* se koristi kako bi se postigla responzivnost visine i širine te sekcije. *Div*

elementima je također pridodana *row* klasa koja je dio Bootstrap paketa koja određuje da taj div bude rasprostranjen u jednom redu.

Za prikaz lokacije tvrtke preko Google Maps potrebno je bilo implementirati API za Google Maps.

Slika 34. *Implementacija GoogleMaps API-a*

Postoji više načina na koje se može ugraditi Google Maps API unutar HTML koda, jedan od načina je i koristeći JavaScript funkciju. U ovoj aplikaciji je korišten upravo taj način. Za njega je bilo potrebno kreirati JavaScript funkciju koja će definirati način na koji će se prikazivati karta.

```
script.

    function myMap(){
        var mapOptions = {
        center: new google.maps.LatLng(45.815011, 15.981919), zoom: 10, mapTypeId: google.maps.MapTypeId.HYBRID
    }
    ...
    var map = new google.maps.Map(document.getElementById("map"), mapOptions);}

script(src="https://maps.googleapis.com/maps/api/js?key=AIzaSyAtVs9JIaVxCwAHDklIllcQyw2VqqOHEv4")
script(src="https://maps.googleapis.com/maps/api/js?callback=myMap")
```

Slika 35. *JavaScript funkcija myMap*

Pomoću te funkcije moguće je namjestiti opcije kao na primjer gdje će karta biti centrirana. U ovom slučaju su zadane koordinate grada Zagreba, ali moguće je dodati koordinate bilo kojeg mjesta. Moguće je i dodati marker na kartu, ali ta opcija ovdje nije korištena. Ta funkcija mora biti povezana sa nekim HTML elementom. Za to je bilo potrebno definirati neki *div* unutar kojega će se karta renderirati sa zadanim parametrima.

```
<div id="map" class="col-lg-8 col-lg-offset-2 col-sm-6"
style="width:400px; height:250px">
</div>
```

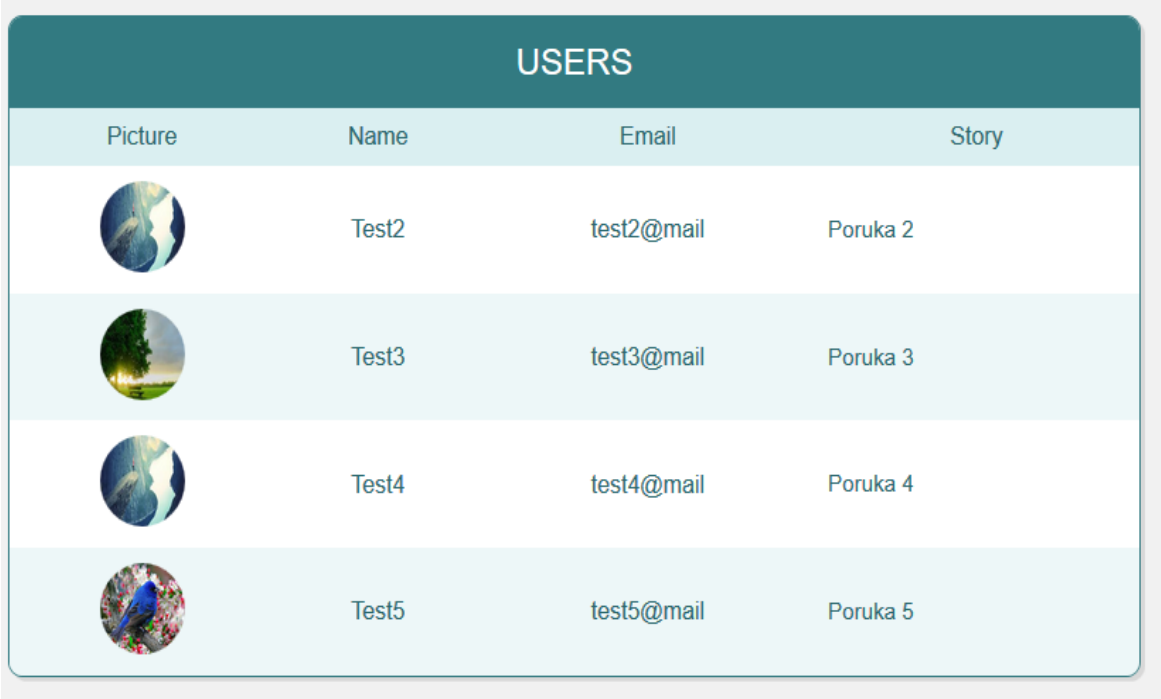
Slika 36. *div element unutar kojega se renderira karta*

Divu su definirani visina i širina kao i pozicije na Bootstrap mreži. Tom *divu* je dodijeljen id=“map“ preko kojega mu se pristupa kroz JavaScript funkciju myMap. Za prikaz karte je još bilo potrebno povezati sami API putem URL adrese unutar skriptnog dijela HTML-a. Google omogućava besplatno korištenje API-a, ali za njega je potrebno definirati ključ koji je dio URL-a. Za generiranje API ključa potrebno se samo registrirati na njihovoj stranici.

Kao što je već spomenuto, cijela internetska aplikacija napravljena je na jednoj stranici koja je podjeljena u nekoliko sekcija koji se mogu prilagoditi ovisno za koje svrhe se stranica koristi.

3.4.2 Stranica za prikaz popisa klijenata

U svrhu prikazivanja popisa klijenata koji se nalazi u bazi podataka izrađena je jedna statična HTML internetska stranica. Ona u obliku tablice prikazuje sve klijente i njihove stavljene slike po kronološkom redu.



Slika 37. *Tablica koja prikazuje zapise iz baze podataka*

Za kodiranje ove stranice korišten je Jade mehanizam za predloške. Pomoću njega je brže i lakše pisati HTML kod pritom misleći na uvlačenje redaka.

```
 9  body
10      .table-users
11          .header Users
12          table(cellspacing='0')
13              tr
14                  th Picture
15                  th Name
16                  th Email
17                  th(width='230') Story
18  
19                  each client, i in clientlist
20                      tr
21                          td
22                              img(src= '#{client.image}', alt='')
23                          td #{client.name}
24                          td #{client.email}
25                          td #{client.story}
26
```

Slika 38. *HTML kod koji generira tablicu*

Unutar *body* sekcije kreirana je tablica. *Body* element definira 'tijelo' dokumenta. On sadrži cijeli sadržaj HTML dokumenta kao što su: tekst, poveznice, slike,

tablice, itd. Oznaka *tr* označava red tablice (eng. *table row*). Budući da će veličina tj. broj redova u tablici biti dinamičan i ovisiti će o broju zapisa u datoteci bilo je potrebno napraviti jednu *each* petlju. Već je ranije, u opisu implementacije poslužitelja, objašnjeno kako u svojoj GET metodi Node prosljeđuje dokumente sa oznakom *clientlist*. Ta je oznaka ovdje korištena i ona sadrži sve zapise iz baze koji se žele prikazati, a *each* petlja prolazi po zapisima i za svaki dokument kreira jedan red sa vrijednostima unutar tog dokumenta. Na taj način je postignuto da se veličina tablice mijenja dinamično.

4 Zaključak

U ovom radu dokazalo se da izrada internetskih aplikacija postaje jednostavnija, ali ne i manje zahtjevna. Budući da postoji mnoštvo uređaja sa kojih se putem internetskih pretraživača može pristupiti internetskim aplikacijama, tako isto postoji mnogo ograničenja i pravila kojih se programeri moraju držati. Prilikom izrade aplikacije potrebno je omogućiti pokretanje internetske aplikacije na svim internetskim pretraživačima (ili barem onim najpopularnijima) i prilagoditi izgled sučelja raznim veličinama ekrana (učiniti ih responzivnima). Današnje tehnologije napreduju upravo u tom smjeru, a jedne od najkorištenijih su upravo one koje čine MEAN stog.

Ovaj rad je istraživao mogućnosti MEAN stoga. Node.js nudi mnoge dobre i jednostavne alate, jedan od njih je i korišteni Formidable koji olakšava parsiranje datoteka i bez kojega bi izrada ove aplikacije bila znatno kompliciranija. Vidjelo se da Node.js također nudi jednostavno i lako povezivanje sa MongoDB bazom podataka koju je jednostavno instalirati i pokrenuti.

Dokazalo se i da ne podržavaju svi internetski pretraživači tehnologije na isti način, kao npr. kada je riječ o HTML elementima. Tako se moralo izostaviti popravljanje grešaka prilikom pokretanja internetske aplikacije na velikom spektru pretraživača i usmjeriti se na samo jedan (Mozzila Firefox), ali postoji prostor da se u budućnosti i to unaprijedi. Uvidjelo se i da je JavaScript prerastao razinu skriptnog jezika. Koristeći JavaScript na klijentskoj i poslužiteljskoj strani vidjelo se da korištenje iste sintakse znatno olakšava pisanje programskog koda. Istraživanje mogućnosti raznih jezika može uzeti neko vrijeme, a pisanje u više programskih jezika od jednom može katkada dovesti do sintaktičkih grešaka.

Prilikom pisanja rada vidjelo se da Node-js koristi dobru i jednostavnu strukturu aplikacije na poslužiteljskoj strani što čini pisanje ruta nešto jednostavnijim u odnosu na druge jezike (npr. PHP). Vidjelo se da je JavaScript učinkovit i lagan za korištenje kada je riječ o pisanju pozadinske logike. JavaScript kao podloga Node.js-a omogućava jednostavnu komunikaciju poslužitelja i klijenta i zato ne čudi činjenica da mu popularnost ne opada, već upravo suprotno, i dalje se ulaže u razvoju tehnologija temeljenih na JavaScriptu.

5 Literatura

1. ***Tech Terms, https://techterms.com/definition/web_application, 15.08.2017.
2. ***Self-taught coders, https://selftaughtcoders.com/how-web-apps-work/, 15.08.2017.
3. ***St-andrews, https://alb.host.cs.st-andrews.ac.uk/webdatabases/howwebapp.htm, 18.08.2017.
4. S. Powers, Getting Started with the Web, Cambridge, O'Reilly, 2011.
5. K. Williamson, Naučite AngularJS, Cambridge, O'Reilly, 2015.
6. J. Krause:Progrmming Web Applications with Node, Express and Pug, Berlin, 2017
7. E. Brown: Modern JavaScript, Cambrige, O'Reilly Media, 2015.
8. S. Powers: Naučite JavaScript, Cambridge, O'Reilly, 2009.
9. V. Skolan, (2014). Programski paket za ispitivanje metoda analize slike, Diplomski rad, Fakultet elektrotehnike i računarstva.
10. ***Npmjs, https://www.npmjs.com/package/jadeWilliams, 18.08.2017.
11. ***W3 Schools, https://www.w3schools.com/bootstrap/bootstrap_grid_system.asp, 20.08.2017.
12. ***Taniarasica, https://www.w3schools.com/bootstrap/bootstrap_grid_system.asp, 19.08.2017.
13. Crnko N. (2016). VIDI, br. 249, (prosinac, 2016.), 110. str
14. ***MongoDB, https://www.mongodb.com/what-is-mongodb, 20.08.2017.
15. ***MongoDB, https://docs.mongodb.com/manual/reference/program/mongod/, 20.08.2017.
16. ***W3 Schools, https://www.w3schools.com/tags/ref_httpmethods.asp, 13.08.2017.
17. ***Express, https://expressjs.com/en/guide/routing.html, 16.08.2017.

6 Popis slika