2. HTML

HTML (eng. *Hypertext Markup Language*) je *markup* jezik koji specifikuje strukturu *web* stranica. Može se reći da je HTML jezik za opis *hypertext*-a, gdje *hypertext* predstavlja tekst koji sadrži veze ka drugim stranicam ili ka segmentima stranice na kojoj se nalaze. HTML je prilično jednostavan jezik sastavljen od elemenata koji se mogu primjeniti na dijelove teksta kako bi im se dalo drugačije značenje u dokumentu, strukturiraju dokument u logičke cjeline i ugrađuju sadržaj kao što su slike i video zapisi na stranicu. [[1](#Lea19)]

HTML dokument ili stranica se sastoji od teksta i tagova (eng. *markups*) kojima se *web* čitaču daju instrukcije kako da prikaže HTML stranicu. Svaki tag ima svoje ime i odgovarajuće atribute i navodi se pomoću para tagova, otvarajućeg taga *<ime\_taga>* i zatvarajućeg taga *</ime\_taga>*. Takođe postoje i prazni tagovi koji se navode na sljedeći način *<ime\_taga/>*. Tag zajedno sa sadržajem koji se navodi između otvarajućeg i zatvarajućeg taga predstavlja element HTML dokumenta. Atributi taga se navode unutar otvarajućeg taga i njihove vrijednosti se uokviruju navodnicima. Atributi nam omogućavaju korištenje elemenata na različite načine zavisno od okolnosti. Na Slici 2.1 prikazan je primjer korištenja atributa.

*<html>*

*<head>*

*<title>Primjer</title>*

*</head>*

*<body>*

*<img src=”img.jpg” alt=”slika”/>*

*</body>*

*</html>*

*Slika 2.1*

Tag *<img/>* u HTML datoteci definiše prikazivanje slike, gdje atribut *src* definiše ime slike ili putanju do slike koja se treba prikazati, a atribut *alt* definiše tekste koji će biti ispisan ukoliko slika iz nekog razloga nije učitana.

Osnovnu strukturu HTML dokumenta čine sjedeči tagovi: *<html>*, *<head>* i *<body>*. Svaki HTML dokument počinje *<html>* tagom u okviru kojeg se nalaze ostali tagovi i završava *</html>* tagom. Tag *<head>* sadrži metapodatke (podatke o HTML datoteci: ko je autor, ključne riječi, itd.), veze ka CSS stranicama, tag *<title>* koji definiše naslov stranice. Sve što se nalazi u tagu *<head>,* osim sadržaja taga *<title>* neće biti prikazano kada *web* čitač učtita stranicu. Sadržaj taga *<body>* je sve ono što će biti prikazano kada se stranica učita. Može imati atribute kojima će se specifikovati karakteristike dokumenta kao što su: boja teksta, boja pozadine ili slika pozadine, akcije koje će se izvršiti kada se dokument učita ili ne učita i slično. [[2](#ETi)]

*Web* čitač ne prikazuje HTML tagove, nego ih koristi da bi odlučio kako da prikaže sadržaj dokumenta i prepozna paragraf, listu, link, zaglavlje i ostale elemente HTML dokumenta.

Kao što je rečeno, postoje elementi koji se označavaju parom tagova i kaže se da takvi elementi definišu određen sadržaj u stranici. Takođe, postoje elementi sa praznim tagom i takvi elementi se koriste za ubacivanje nekog sadržaja u stranicu. Sljede primjeri ovakvih elemenata.

Parovi tagova:

* *<p>Ovo je pasus.</p>* - Pasusi se navode između *<p>*tagova. Unutar početnog taga može se navesti atribut *align* kojim se definiše horizontalno poravnanje pasusa i može imati jednu od sljedećih vrijednosti: *left*, *center*, *right*, *justify*.
* *<font size=”10”>Tekst veličine 10.</font>* - Omogućuje promjenu veličine, boje i vrste fonta.
* *<ul>* - Neoznačena (eng. *Unordered*) lista kojom se definiše lista elemenata ispred kojih se nalazi *bullet*. Svaki element definiše se tagom *<li>*. Ukoliko se umjesto *<ul>* koristi tag *<ol>*, radi se o numerisanoj (eng. *Ordered*) listi. Postoje i definicione liste koje se definišu pomoću para tagova *<dl>* i sastoje se iz dva dijela: *termina* i *definicije*. Na slici je primjer korištenja liste.
* *<a href=”https://etf.unibl.org/”>ETF</a>* - Linkovi se definišu pomoću para tagova *<a>*. Atributom href se definiše pozicija na koju se stiže ukoliko se klikne na dati link.
* *<iframe src=”https://etf.unibl.org/”></iframe>* - Ovim tagom se definiše *inline frame*, kojim se omogućava ubacivanje drugog dokumenta unutar trenutnog.
* *<form>* - Ovaj par tagova definiše formu koja omogućava korisniku da unese neke informacije (npr.: unos korisničkog imena, lozinke, izbora elementa iz padajuće liste i slično). Atributom *method* se definiše način prenosa informacija koje su unešene (GET ili POST), dok se atributom *action* definiše adresa stranice (URL) na koju će informacije biti prenešene. Najčešće korišten tag unutar *<form>* taga je *<input>* kojim se definiše element forme. Tip unosa određuje se pomću atributa *type* i tipovi mogu biti *text*, *password*, *image*, *file*, *select*, *radio*, *checkbox*, *submit*, *button*.

Prazni tagovi:

* *<img/>* - Ovo je tag pomoću kojeg se ubacuje slika u stranicu.
* *<br/>* - Koristi se za dodavanje nove linije.
* *<hr/>* - Slično prethodnom, koristi za razdvajanje teksta, s tim što dodaje i jednu horizontalnu liniju ispred i iza koje se nalazi prazan red. Pomoću atributa *size* može se definisati visina, a širina pomoću *width*.
* *<meta>* - Definišu dodatne informacije koje *web* čitač ne prikazuje.

HTML se od svog postojanja razvijao, te postoji više verzija ovog jezika. Danas se svakako koristi najakutelnija, a to je *HTML 5* koji se pojavio 2014. godine. Slijedi nešto opširnije o ovoj verziji.

HTM5 je najnovija verzija HTML standarda koja nudi nove karakteristike (eng. features), poboljšava podršku za kreiranje web aplikacija za komunikaciju sa klijentima, sa njihovim lokalnim podacima i serverima na jednostavnji i efikasnije način nego što je to bilo ranije. [[3](#KFö)] Preporučen je od strane W3C (*Wordl Wide Web Consortium)*.

Svaki HTML5 dokument počinje deklaracijom tipa dokumenta *<!DOCTYPE html>*, koja je obavezna i dosta kraća od onih u prethodnoj verziji. Takođe u ovoj verziji se primjenjuje kraći način za određivanja kodovanja *<meta charset=”UTF-8”>*. Pored ovoga HTML5 ima dosta novih tagova i atributa ali i onih koji se u ovoj verziji više ne koriste. Slijede neki od najznačajnih karakteristika.

Struktura i semantika dokumenta:

* <header> - Ovaj tag odnosi se na kontejner za naslov i dodadni uvodni sadržaj. Ne odnosi se samo na zaglavlje na vrhu stranice, već se mogu koristi i u ostalim dijelovima stranice.
* <article> - Reprezentuje nezavisno područje web dokumenta, za npr. novosti, blog ili sličan sadržaj.
* <footer> - Ovaj element sadrži dodatne informacije o sekciji na koju se odnosi, kao što su: autor, povezane stranice, autorska prava i slično. Može da se nalazi unutar <article> elemeta.
* <nav> - Ukolilo je potrebna kreirati navigacione blokove preko kojih se prelazi na neku labelu unutar dokumenta ili na neku drugu povezanu stranicu.
* <aside> - Koristi se za sadržaj koji je sporedan u odnosu na najbitniji sadržaj stranice i treba da bude vezan za taj njega.
* <section> - Sadrži sekciju dokumenta kao što je poglavlje eseja, u sklopu zaglavnja ili *footer*-a.
* <figure> - Koristi se za prikazivanje nezavisnik slika, grafika, dijagrama i unutar njega se može nalaziti elemet <figcatption> za navođenje naslova.
* <meter> - Element za grafičko predstavljanje vrijednosti u zadatom opsegu.
* <progress> - Grafički predstavlja napredak završetka zadatka.

Pametne forme:

HTML5 specifikacija poboljšava <input> element dozvoljavajući nove vrijednosti za *type* atribut, a to su: tel – ulazna vrijednost je telefonski broj, search – ulazno polje je search, url – ulazna vrijednost je url, email – ulazna vrijednost je email, datetime, date, month, week, time, datetime-local, number, range, color – u heksadecimalnom zapisu. Takođe, HTML5 nudi nekoliko novih atribua za elemente forme: autofocus, placeholder, required.

Novi elementi forme:

* *<datalist>* - Padajuća lista opcija za ulaznu vrijednost. Input element koristi atribut *list* za povezivanje sa *datalist*.
* *<keygen>* - Koristi se za generisanje para kriptografiskih ključeva. Javni ključ se šalje zajedno sa svim podacima forme kako bi bio dostupan serveru, dok privatni ključ ostaje na strani klijenta.
* *<output>* - Predstavlja rezultat izračunavanja, gdje izračunavanje npr. može biti rezultat množenja unešenog broja u input polje sa nekom vrijednošću.

Video i Audio:

Pomoću taga *<video>* u okviru pretraživača se može pogledati video. Kako bi bila omogućena podrška za različite čitače i uređjaje, video treba da bude enkodovan više puta. Na Slici 2.2 je prikazan primjer upotrebe taga:

*<!DOCTYPE HTML>*

*<html>*

*<head>*

*<title>Video</title>*

*</head>*

*<body>*

*<video width=”320” height=”240” controls>*

*<source src=”movie.mp4” type=”video/mp4”>*

*<source src=”movie.ogg” type=”video/ogg”>*

*<source src=”movie.webm” type=”video/webm”>*

*Your browser does not support the video tag.*

*</video>*

*</body>*

*</html>*

*Slika 2.2*

Kao što se vidi na slici, unutar taga *<video>*, navode se taogovi *<source>* kojima se obezbjeđuje kompatibilnost za različite *web* čitače. Navođenjem atributa *controls* obezbjeđuje se prikazivanje kontrolnih elemenata za video.

Za uključivanje audio fajla u stranicu koristi se tag *<audio>*. Slično kao kod *<video>*, elementom *<source>* se obezbjeđuju različiti audio fajlovi, kako bi *web* čitač izabrao odgovarajući. [[3](#KFö)]

Canvas:

Jedan od najinteresantnijih elemenata u HTML5 je *<canvas>*. Njime je obezbjeđeno iscrtavanje grafičkih elemenata. Iscrtavanje se vrši pomoću *JavaScript*-a, a element *<canvas>* predstavalja kontejner. Na slici je primjer.

*Canvas* element se pronalazi u *DOM*-u pomoću id-a, a onda se nad tim elementom poziva metoda *getContext()* koja vraća *drawing context*. Sada kada je *drawing context* definisan, moguće je iscrtavanje elemenata.

Geolocation API:

*Geolocation API* omogućava korisniku da web aplikaciji obezbjedi svoju lokaciju, ukoliko to želi. Kako bi se obezbjedila privatnost, od korisnika se traži dozvola za dobijanje informacija o lokaciji. *Geolocation API* je dostupan putem  *navigator.geolocation* objekta. Lokacija korisnika se može dobiti pozivom metode *getCurrentPosition()*. Lokacijske informacije se mogu dobiti pomoću različitih izvora: *IP* adrese, *WiFi*, *GPS*, korisničkog unosa. [[4](#19Fe)]

Web Storage i Offline Web Applications:

*Web Storage* predstavlja interfejs koji definiše atribute i metode za perzistento čuvanje podataka na klijentskoj strani. Podaci se čuvaju kao parovi ključ-vrijednost i moguća su dva mehanizma za njihovo skladištenje. Prvi mehanizam je *session storage* - aplikacije smještaju podatke u ovaj *storage* i biće dostupni samo od strane prozora koji ih je kreirao. Po zatvaranju prozora podaci se brišu iz *session storage*-a. *Local storage* je drugi mehanizam u kome se podaci čuvaju i po završetku sesije. Ukoliko korisnik u istom čitaču otvori novi prozor istog *web* sajta i izmjeni vrijednost određenog ključa u *local storage*-u, ova vrijednost biće izmjenjena i za prvi prozor istog sajta. Ovo je osnovni nedostatak čuvanja podataka u *local storage*-u u odnosu na *session storage*. [[3](#KFö)]

*Offline Web Applications*-kako bi klijent mogao pokrenuti aplikaciju bez pristupa mreži, potrebno je da se HTML, JavaScript i multimedijalni fajlovi čuvaju na klijentskoj mašini. *Web* čitač će potrebne podatke čuvati u lokalnoj *cache* memoriji. Kako bi aplikacija radila offline, potrebna je *manifest* datoteka kojom se čitaču govori šta je potrebno čuvati u *cache* memoriji. U situacijama kada klijent nema mrežnu konekciju ili pokušava pristupiti resursu koji se ne nalazi u localnom (offline) *cache*-u, HTML5 može da specifikuje *fallback* slučaj kojim se klijentu uslužuje neki drugi resurs. [[5](#Bru)]

Web Sockets:

*Web Socket* predstavlja najmoćnije svojstvo komunikacije u HTM5 specifikaciji. Definiše *full-duplex* komunikacijski kanal koji radi preko jednog *socket*-a preko internet mreže, što znači da klijent i server mogu slati podatke u bilo kojem trenutku.

Web Workers:

*Web Workers* obezbjeđuju izvršavanje zadataka u pozadini tako da web stranica može nastavi sa radom. *Web Worker* i pozivajuća skripta komuniciraju na sljedeći način: pozivajuća skripta kreira objekat *Worker* i u konstruktoru prosljeđuje naziv skripte koju *worker* treba da izvši. Sada pozivajuća skripta može da pošalje poruku *worker*-u pozivajući *postMessage()* metodu. U *worker*-u je definisana *event handler* *onmessage* koji će obraditi podatke poslane kroz *postMessage()* metodu. Na isti način worker može poslati poruku kao odgovor. *Web Worker* nema pristup *DOM* elemetima i time se potencijalno sprečava da dvije skripte pokušaju pristupiti istom elemetu.

Microdata:

*Microdata* predstavlja način da se *meta* podaci smještaju unutar sadržaja *web* stranice. *Search engines*, *web crawlers* i *browsers* mogu izdvojiti i pretražiti *microdata* i time korisnicima obezbjediti bogatiji i relevantniji pregled stranice.

Globalni atributi:

Globalni atribut su atributi zajednički za sve *HTML* elemente, mogu se koristit za sve elemente, mada na neke elemente nemaju uticaj. Neki od zajedničkih atributa: *accesskey*-definiše prijedlog za generisanje prečice na tastaturi za trenutni element, *autocapitalize*-definiše da li i na koji način se unešeni tekst automatski piše velikim slovom, *class*-klase dozvoljavaju *CSS*-u i *JavaScript*-u da pristupe određenim elementima preko selektora klasa, *contenteditable*-atribut koji definiše da li se dati element može uređivati od strane korisnika, *id*-jedinstveni identifikator elementa koji je jedinstven u čitavom dokumentu, njegova svrha je da dozvoli *CSS*-u i *JavaScript*-u pristup datom elementu, *style*-sadrži *CSS* deklaracije stila koje treba da se primjene na element.

CSS3

CSS (*Cascading Style Sheets*) se koristi za stil i izgled web stranice. Stilovi se definišu za HTML elemente i mogu da promijene boju, font ili pozadinu elementa. Korištenjem CSS kao eksternog dokumenta razdvaja se sadržaj i formatiranje stranice, pa se *web* stranice brže učitavaju. Takođje, ovim je omogućeno ažuriranje više različitih stranica.

CSS3 predstavlja posljednju verziju evolucije CSS-a i u potpunosti je kompatibilan sa prethodnim verzijama. Najveća promjena u CSS3 verziji su moduli kojima se postiže efikasniji i laksi rad. Neki od najvažnijih modula su: selektori, tekstualni efekti, pozadine i ivice, 2D i 3D transformacije, animacije i drugi.

3. JavaScript

3.1 Uvod

JavaScript je platformski nezavisan, objektno orijentisan programski jezik koji se koristi u svrhu kreiranja dinamičkih *web* stranica. JavaScript može funkcionisati kao proceduralni jezik ili kao jezik baziran na protipovima objekata. Objekti se kreiraju programski zajedno sa pripadajućim metodama i atributia. Jednom kreiran objekat može se koristiti kao prototip za kreiranje sličnih objekata.

JavaScript se sve češće koristi u razvojnim okruženjima koji su zasnovani na *JavaScript*-u za razvoj serverske strane. Jedno od takvih okruženja je *NodeJS* koje je opisano u poglavlju 5.

JavaScript je nastao u kompaniji *Netscape*, koji je predstavio ovaj jezik *ECMA (Europen Computer Manufacturers Association)* organizaciji za standarde. *ECMA* je razvila standard i preimenovala jezik u *ECMAScript*. U praksi se ipak češće koristi naziv JavaScript.

3.2 Osobine *JavaScript*-a

U JavaScript-u nije dozvoljeno deklarisanje tipova podataka promjenjivih. JavaScript *interpreter* prepoznaje tip podatka i dinamički ga dodjeljuje promjenjivoj. JavaScript tipovi mogu biti podijeljeni u dvije kategorije: primitivni i objekti. Primitivni tipovi mogu bit: *boolean*, *null*, *undefined*, *number*, *string*, *symbol*. Objekti se mogu posmatrati kao kolekcija atributa. Vrijednosti atributa mogu biti bilo kojeg tipa, i mogu se dinamički dodati ili ukloniti. Na slici je prikazan jedan od načina kreiranja i korištenja objekta u JavaScript-u:

Slika 3.1

function Person(first\_name, last\_name, age){

this.first\_name = first\_name;

this.last\_name = last\_name;

this.age = age;

}

var person1 = new Person(“John”, “Doe”, 25);

document.getElementById(“demo”).innerHTML =

“My name is ” + person1.first\_name;

Na slici je primjer funkcije koja kreira objekat.

Funkcije se posmatraju kao objekti, jer mogu imati svoje atribute i metode kao i svaki objekat. Razlika u odnosu na objekte je ta da funkcije mogu biti pozvane. Funkcija se može proslijediti kao argument neke druge funkcije, može biti povratna vrijednost druge funkcije i može biti dodjeljena kao vrijednost određenoj varijabli. Definicija funkcije je prikazana na slici:

Slika 3.1

function myFunction(param1, param2){

statements

}

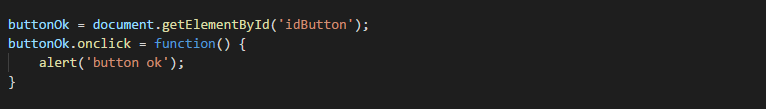
3.3 *Client Side* i *Server Side JavaScript*

Izvršanje *JavaScript* koda na klijentskoj strani u okviru čitača podrazumjeva da taj *web* čitač ima ugrađen *JavaScript* prevodilac (eng. *interpreter*). U HTML stranici koja će biti učitana, pomoću taga *<script>* *web* čitaču se daje naznaka da sve što se nalazi unutar ovog taga treba biti tumačeno kao *JavaScript* skripta. Tada skripta može dinamički da mijenja sadržaj učitane stranice, koristeći *DOM* (eng. *Document Object Model*) kojim se omogućava pristup objektima stranice.

*JavaScript* se za razvoj serverske strane može koristi za komunikaciju sa bazom podataka, prihvatanje i obradu klijentskih zahtjeva, kreiranje i slanje odgovora klijentu ili manipulaciju fajlovima.

3.4 Događaji

Ranije su se *JavaScript* programi mogli izvršavati isključivo sinhrono, što znači da se u jednom trenutku može izvšiti samo jedna operacija. Ovakav način izvršavanja je neuobičajan i nepraktičan danas. Danas se pišu programi tako da se za svaki događaj poziva *event handler* ili *event listener* funkcija. Ove funkcije se pozivaju asinhrono kada se desi događaj za koju je funkcija napisana. Događaji mogu biti učitavanje stranice, klik na dugme, unos teksta u polje i slično. Svaki događaj ima svoj tip i ciljani objekat, gdje ciljani objekat predstavlja *DOM* element. Tipovi događaja mogu biti *click*, *change*, *load*, *mouseover*, *keypress* ili *readystatechange*. Ukoliko želimo da naš program odgovori na događaj klik na dugme pomoću funkcije *event handler*, treba da registrujemo, odnosno napišemo ovu funkciju na način da tipu dogašaja dodamo prefiks *on.* Slijedi primjer:



buttonOk = document.getElementById(‘idButton’);

buttonOk.onclick = function() {

alert(‘button OK’);

}

U JavaScript-u se za asinhrono izvršavanje i vraćanje povratne vrijednosti kada je rezultat spreman još koriste i *Callback* funkcije. Callback funkcije se izvršavaju nakon što se neka druga funkcija izvrši. Kako su funkcije u JavaScript-u objekti prve klase, Callback funkcija se proslijeđuje kao argument nekoj drugoj funkciji iz koje će se Callback funkcija pozvati. Na slici slijedi primjer:

{

function greeting(name){

alert(‘Hello ’ + name);

}

function processUserInput(callback){

var name = prompt(‘Please enter your name.’);

callback(name);

}

processUserInput(greeting);

}

Slika 3.1

Poziva se funkciju processUserInput*()* i kao argument joj se prosljeđuje *Callback* funkcija *greeting*. Kada se prosljeđuje *Callback* funkcija kao argument, prosljeđuje se samo njena definicija. Funkcija processUserInput*()* od korisnika zahtjeva unos imena. Kada korisnik unese ime, poziva se *Callback* funkcija i prosljeđuje joj se unešeno ime koje je upisano u varijablu *name*.

Pored *Callback* funckija koriste se i *Promise*-e. *Promise* je objekat koji predstavlja eventualni uspjšno ili neuspješno izvršavanje asinhrone operacije. – OVJE REFERENCU-

*Promise* obezbjeđuje povezivanje rukovaoca sa uspješnim izvršavanjem ili neuspješnim izvršavanjem. Ovim se omogućava da asinhrona metoda vrati „obećanje“ da će vratiti vrijednost u nekom budućem trenutku. *Promise* može da ima jedno od sljedeća tri stanja: *peding* – inicijalno stanje, *fulfilled* – znači da je operacija uspješno izvršena, *rejected* – označava da je operacija neuspješna. Kada se desi jedna od navedenih opcija, pridruženi rukovaoci koji su stavljeni u red čekanja *then* metode *Promise*-e se pozivaju. Na slici slijedi primjer kreiranja i korištenja *Promise. MDN*

Slika 3.1

var promise = new Promise(function(resolve, reject){

mysql.getConnection(function(err, connection){

if(err) reject(err);

else{

connection.query('SELECT \* FROM restaurant', function(err, rows){

resolve(JSON.stringify(rows));

});

}

});

});

promise.then(data=>{

console.log(data);

}).catch(еrr=>{

console.log(err);

})

4. NodeJs

4.1 Uvod

NodeJS je razvojno okruženje koje se koristi za razvoj brzih i skalabilnih serverskih aplikacija. Dizajniran je tako da bude dobar za intezivne I/O aplikacije koje koriste *event-driven* arhitekturu. *Event-driven* modelom programiranja se obezbjeđuje da NodeJs pokreće samo jednu nit.

NodeJs se najčešće izvršava asinhrono. To znači da kada se desi događaj, poziva se callback funkcija, registrovana za taj događaj. Dok se čeka izvršavanje ove funkcije, sljedeći događaj ili funkcija je u redu za izvršavanje. Na serverskoj strani, događaj se može definisati kao asinhrone ulazno/izlazne (*asynchronous I/O*) poruke.

Postoji više razloga zašto je NodeJs sve više pogodniji za razvoj serverske strane. Jedan od njih je JavaScript kao najčešće korišten programski jezik web aplikacija. Drugi razlog je jednostavnost NodeJs-a. Osnovne funkcionalnosti su svedene na minimum. Još jedan od razloga je veomam jednostavno preuzimanje i instaliranje, a zatim pokretanje u svega nekoliko minuta.

4.2 Event-Driven model programiranja

Event-Driven programiranje predstavlja stil programiranja kod koga je tok izvršavanja programa određen događajima. Događajima se rukuje pomoću callback funkcija. Ovaj stil programiranja, gdje se umjesto povratne vrijednosti, definišu callback funkcije koje će biti pozvane kada se događaj desi i koje će obraditi zahtjeve i vratiti odgovor, još naziva asinhrono programiranje. Ovo je jedna od karakteristika koje definišu NodeJs. Na Slici je prikazan poziv callback funkcije koja dobavlja podatke iz baze:

Slika 4.3

function getRestaurants(){

database.getRestaurantsDB(function(data){

console.log(data);

});

}

//database

function getRestaurantsDB(callback){

mysql.getConnection(function(err, connection){

if(err) callback(err);

else{

connection.query('SELECT \* FROM restaurants',

function(err, rows){

callback(rows);

})

}

})

}

Unutar *getRestaurants()* poziva se funkciju *getRestaurantsDB()* i kao argument joj prosljeđuje *Callback* funkcija *function(data)*. Funkcija *getRestaurantsDB()* izvršava upit nad bazom, ukoliko je došlo do greške pozvaće se *callback* i kao argument se prosljeđuje *err* (greška nastala pri izvršavanju funkcije *getConnection*). U slučaju da je upit uspješan, funkciji *callback* se kao argument prosljeđuje *rows* (rekordi koji se dobiju izvršavanjem upita). U oba slučaja, po pozivu, funkcija callback će ispisati prosljeđeni argument (*err* ili *rows*).

Kod Event-Driven modela programiranja trenutni proces ne blokira I/O operacije, pa se zbog toga može pojaviti više I/O operacija u paraleli za koje će biti pozvane odgovarajuće callback funkcije.

Event-Driven stil programiranja praćen je *event-loop*-om. *Event-loop* je konstrukcija koja obavlja dvije funkcije u neprekidnoj petlji - *event detection* (funkcija koja detektuje događaj) i *event handler triggering* (funkcija za poziva rukovaoca događaja). *Event-loop* mora da detektuje događaj kada se desi i da odredi odgovarajuću *callback* funkciju i pozove je. *Event-loop* je samo jedna nit koja se izvodi unutar jednog procesa, što znači, da kada se desi događaj, *event handler* može da radi bez prekida. To znači da u svakom trenutku postoji samo jedan *event handler* i *event handler* koji je pokrenut biće izvršen bez prekidanja. Ovo je razlog zašto programeri ne moraju brinuti o izvršavanju istovremenih niti.

4.3 Moduli

Kao što je ranije rečeno NodeJs je jednostavan i osnovne funkcionalnosti su svedene na minimum. Kada je potrebno kreirati kompleksniju aplikaciju, tada se na jednostavan način mogu uključiti NodeJs moduli. Pomoću modula postiže se bolja organizacija koda i izbjegavaju se greške i sigurnosni problemi.

U NodeJS-u moduli se referenciraju po putanji ili po imenu. Module možemo podijelit u tri grupe. Postoje *core* moduli koji se učitavaju pri pokretanju NodeJS procesa. Drugi tip modula su *third-party* koji se instaliraju u aplikaciji. I treći tip modula su *local* moduli koje kreira programer.

*Core* ili osnovni moduli su kompajlirani u binarne distribucije i iako se automatski učitavaju pri pokretanju NodeJS-a, potrebno ih je prvo uvesti kako bi se mogli koristiti. Neki od osnovnih modula:

* *http* – *http* moduo uključuje klase, metode i događaje za kreiranje Node.js *http* servera.
* *url* – *url* moduo uključuje metode za URL rezoluciju i parsiranje.
* *fs* – *fs* moduo uključuje klase, metode i događaje za rad sa fajlova

Core moduli se referenciraju samo pomoću imena, a ne putanje i učitavaju se iako postoji *third-party* moduo sa istim imenom. Učitavanje *http* core modula prikazana je na Slici:

var http = require('http');

Slika 4.3

Funkcija require vraća http objekat koji implementira API (*Application programming interface*) pomoću kojeg NodeJs prenosi podatke preko HTTP (*HyperText Transfer Protocol*) protokola.

*Third-party* ili eksterni moduli instaliraju se u aplikaciji pomoću NPM (*Node Package Manager*). NPM predstavlja repozitorijum NodeJs modula i dolazi zajedno sa instalacijom NodeJs-a. Na slici je prikazano kako se instalira modul pomoću NPM-a:

Slika 4.3

$ npm install <naziv modula>

Ovako kreirani moduli smještaju se u lokalni direktorijum *node\_module* i mogu se učitati pomoću funkcije *require* i prosljeđivanjem imena modula, kao na Slici 4.3.2.

*Local* modul se kreira na način da se kreira JavaScript datoteka koja izvozi objekat, koji predstavlja API modula. Ovi moduli se mogu distribuirati putem NPM-a, tako da ih NodeJs zajednica može koristiti. Referenciranje *local* modula vrši se pomoću apsolutne ili relativne putanje. Na Slici je prikazano kreiranje local modula:

// message.js

function info(info){

console.log(‘Info: ’ + info);

}

function error(error){

console.log(‘Error: ’ + error);

}

module.exports = {

info,

error

}

Slika 4.3

4.4 Express

Za neke programske jezike i platforme postoje razvojna okruženja koja imaju za cilj da pojednostave i strukturiraju izradu HTTP baziranih aplikacija. Postoje različiti moduli i razvojna okruženja za Node.js, a jedan od njih je Express.js.

Express.js je najpopularnije razvojno okruženje za Node.js i osnovna biblioteka za brojna druga razvojna okruženja. MDN

Nudi sljedeće mehanizme:

* pisanje rukovaoca za korisničke zahtjeve sa različitim HTTP metodama i URL putanjama
* integrisanje sa „view“ mehanizmima za prikazivanje, kako bi se generisali odgovori ubacivanjem podataka u šablone (*templates*)
* postavljanje web postavki aplikacije kao što su: port koji se koristi za povezivanje, lokaciju šablona koji se koriste za prikazivanje odgovora
* dodavanje „middleware“ za dodatnu obradu korisničkih zahtjeva

Express.js se instališe pomoću NPM, naredbom koja je prikazana na Slici:

$ npm install express --save

Slika 4.4

Na ovaj način će Express.js biti instaliran i sačuvan u listi zavisnosti (*dependencies*). Instaliranje Express.js-a dobijaju se datoteke na osnovu kojih se bazira nova aplikacija. Prva je manifest datoteka *package.json*. Osnovna namjena ove datoteke je ta da sadrži listu zavisnosti modula koju NPM može koristiti da bi ih instalirao. Kada se instaliraju ove zavisnosti smještaju se u *node\_modules* folder. Sljedeći je *public* folder, koji sadrži foldere za statičke datoteke koji mogu biti potrebni aplikaciji. Neki od standardnih foldera su: images, javascripts, stylesheets sa svojim datotekama koje korisnici mogu mijenjati. Pored ovoga, Express.js kreira datoteku app.js koja sadrži inicijalizacioni kod servera. Na Slici je prikazana jednostavna Express.js aplikacija koja pokreće server i sluša na portu 5000.

Slika 4.4

var express = require(‘express’);

var app = express();

app.get('/', function(req, res){

res.send('Hello');

})

var server = app.listen(5000, function(){

var host = server.address().address;

var port = server.address().port;

console.log('App listening at http://%s:%s', host, port);

})

4.5 Middleware

Inicijalizacioni kod app.js datoteke dodaje *middleware* funkcije. Express.js aplikacija je u suštini serija poziva middleware funkcija.

*Middleware* funkcije su funkcije koje imaju pristup *request* objektu, *response* objektu i sljedećoj *middleware* funkciji u *request-respons* ciklusu aplikacije. Sljedeća funkcija *middleware* je označena varijablom *next*. *Middleware* funkcije mogu obavljati sljedeće zadatke: izvršavanje nekog koda, pormjenu *request* i *response* objekta, završiti *request-response* ciklus, poziv sljedeće *middleware* funkcije sa *stack*-a. Ukoliko tekuća middleware funkcija ne završi *request-response* ciklus, mora pozvati sljedeću middleware funkciju kojoj će proslijediti kontrolu.

Express aplikacije mogu da koriste sljedeće tipove middleware-a:

* Application-level middleware – povezuje se application-level middleware sa objektom aplikacije koristeći app.use() i app.METHOD() funkcije, gdje METHOD može biti GET, PUT ili POST
* Router-level middleware – radi na isti način kao prethodna, s tim što se veže za instancu express. Router()
* Error-handling middleware - funkcija se definiše isto kao i druge middleware funkcije, ali ima još jedan argument *err*
* Built-in middleware – Express ima sljedeće build-in middleware funkcije: *express.static*, *express.json*, *express.urlencoded*
* Third-party middleware – koristi se za dodavanje novih funkcionalnosti Express. aplikaciji

Rutiranje zahtjeva – različiti dijelovi aplikacije treba da aktiviraju različito ponašanje servera. Npr. GET i POST zahtjevi se trebaju smatrati različito i različiti URL-ovi takođe aktiviraju različite odgovore. Ukoliko želimo da neka funkcija vraća korisničke profile koji se čuvaju npr. u bazi podataka, različitom kombinacijom HTTP metoda i URL-ova dobijaju se različiti odgovori:

* GET/users – za prikazivanje liste korisničkih profila
* GET/users/:username – za prikazivanje profila korisnika sa proslijeđenim korisničkim imenom
* POST/users – za kreiranje korisničkog profila
* PUT/users/:username - za ažuriranje profila korisnika sa proslijeđenim korisničkim imenom

Prethodna lista predstavlja listu rutiranja, gdje se kombinacije metoda i URL-ova mapiraju u odgovarajuće akcije, odnosno callback funkcije. Callback funkcije su najčešće definisane u odvojenim modulima, koji se nazivaju kontroleri (*controllers*). Ukoliko je potrebno kreirati funkciju koja šalje odgovor na zahtjev *GET/users* iz prethodne liste, kreira se modul *controllers/user.controler.js* koji je prikazan na Slici:

Slika 4.4

// *controllers/user.controller.js*

var db = require(‘./database/user.database’);

function getAllUsers(req, res){

db.getAllUsers(function(data){

res.send(data);

})

}

U kodu na prethodnoj slici, pozivom *require(‘./database/user.database’)* uvezen je moduo sa funkcijama koje dobavljaju podatke iz baze podataka. Na sličan način se kreiraju callback funkcije za zahtjeve iz liste.

Dobra je praksa da su rute za različite API-je (npr.: Users API, Tasks API) definisane u različitim modulima. Moduli se kreiraju u folderu routes. Na Slici je primjer:

Na serverskoj strani događaji se mogu definisati kao asinhrone ulazno/izlazne (*asynchronous I/O*) poruke. Kada server dobije I/O zahtjev proslj

x

|  |  |
| --- | --- |
| [1] | Learning HTML: Guides and tutorials. [Online], <https://developer.mozilla.org/en-US/docs/Learn/HTML/Introduction_to_HTML> |
| [2] | E. Tittel, J. Noble, *HTML, XHTML and CSS for Dummies*.: Wiley Publishing Inc., Indianapolis 2011. |
| [3] | B. Öggl K. Förster, *HTML5 Guidelines for Developers*.: Pearson Education, Inc. 2011. |
| [4] | Geolocation API. [Online], <https://developer.mozilla.org/en-US/docs/Web/API/Geolocation_API>, posjeceno: 2019, February |
| [5] | R Sharp B. Lawson, *Introducing HTML5*., 2019. |

x

x

|  |  |
| --- | --- |
| [1] | Learning HTML: Guides and tutorials. [Online], <https://developer.mozilla.org/en-US/docs/Learn/HTML/Introduction_to_HTML> |
| [2] | E. Tittel, J. Noble, *HTML, XHTML and CSS for Dummies*.: Wiley Publishing Inc., Indianapolis 2011. |
| [3] | B. Öggl K. Förster, *HTML5 Guidelines for Developers*.: Pearson Education, Inc. 2011. |
| [4] | [Online], <https://developer.mozilla.org/en-US/docs/Web/API/Geolocation_API>, posjeceno: 2019, February |

x

Web Workers - Klaus Forster(knjiga) i <https://developer.mozilla.org/en-US/docs/Web/API/Web_Workers_API/Using_web_workers>

* Web Sockets - Peter Lubbers
* Brian Albers
* Frank Salim, Pro HTML5 Programming(knjiga)

Web Storage i Offline Web Applications - Klaus Forster(knjiga)

Sve prije canvas itd - Klaus Forster(knjiga)

Microdata - <https://developer.mozilla.org/en-US/docs/Web/HTML/Microdata>

Geolocation - <https://developer.mozilla.org/en-US/docs/Web/API/Geolocation_API>

Globalni atributi - https://developer.mozilla.org/en-US/docs/Web/HTML/Global\_attributes