Table of Contents

JS developers guide 1.1
Introduction 1.2
Reference Types 1.3
Object oriented programming in JS 1.4
- Basics 1.4.1
- Object creation 1.4.2
- Inheritance 1.4.3

Event handling 1.5
JSON handling with JS 1.6
Asynchronous JS 1.7
- AJAX 1.7.1
- Web Sockets 1.7.2

Error handling and debugging 1.8
Javascript i HTML5 1.9
Advanced JS 1.10
Patterns in JS 1.11
Useful JavaScript libraries 1.12
JavaScript tools 1.13
Introduction to JS frameworks 1.14
Developing our JS framework 1.15

JS developers guide

This file serves as your book's preface, a great place to describe your book's content and ideas.

Introduction

Referentni tipovi

Tipovi podataka

U JavaScriptu postoji nekoliko tipova podataka. Formalni tipovi podataka su:

• **String** – niz znakova unutar navodnika

• **Number** – bilo koji broj koji nije pod navodnicima

• **Boolean – true/false** – logičko istinito ili neistinito

• **Null** - lišeno vrednosti, nepoznata vrednost

• **Object** – obuvata sva svojstva i metode koji pripadaju objektu

• **Array** - niz koji skladišti više vrednosti

• **Date** - objekat koji radi sa datumom i vremenom

• **Function** – definicija funkcije

Java Script varijable mogu da sadrže različite tipove podatake koje smo gore naveli. Među njima, primitivni tipovi podataka su String, Number, Boolean.

String

String je niz karaktera sačinjen od nula ili više znakova. String može biti bilo koji tekst koji se piše između navodnika. Navodnici mogu biti jednostruki ili dvostruki.

Npr:

`var ime = “Marko Marković”;`
//korišćenje dvostrukih navodnika

`var ime = ‘Marko Marković;`
//korišćenje jednostrukih navodnika

Takođe, često se javlja potreba da se koriste navodnici unutar stringa. To se može uraditi korišćenjem escape karaktera (kose crte \ ).

Npr:

`var tekst = “Njegovo ime je \”Marko Marković\”.”;`

U stringovima je dozvoljeno koristiti sledeće specijalne karaktere:

• \b = pomeraj za jedno mesto ulevo (backspace)

• \f = pomeraj jedan red dole (form feed)

• \n = na početak novog reda (new line character)

• \r = return (carriage return)

• \t = tabulator (tab).

JavaScript je slabo tipiziran jezik. Tipovi podataka će biti automatski konvertovani zavisno od mesta njihove upotrebe u programu.

Tako, na primer možemo definisati i inicijalizovati sledeću promenljivu:

```
var promenjiva = 7;
```

a kasnije joj dodeliti neku drugu vrednost, odnosno možemo je predefinisati.

```
var promenljiva = "Java Script";
```

Kada govorimo o kombinaciji broja i stringa, Java Script konvertuje broj u string.

Npr:

```
var primer = "Ovo je primer " + 15;
```

ili

```
var primer = 13 + "primer";
```

Za konverziju stringa u broj koriste se sledeće funkcije:

• EVAL – ocenjuje string i pretvara ga u broj ako je moguće

• parseINT – konvertuje string u Integer,ako je moguće

• parseFLOAT – konvertuje string u Float tip ,ako je moguće

Number

Celi brojevi u JavaScriptu mogu biti predstavljeni u tri osnove:

u decimalnom (baza 10), u oktalnom (baza 8) i heksadecimalnom (baza 16) formatu.

Decimalni celi brojevi se predstavljaju kao niz cifara (0-9) bez vodeće nule.

Oktalni celi brojevi se predstavljaju kao niz cifara (0-7) predvođen sa nulom ("0").

Heksadecimalni celi brojevi se predstavljaju kao niz cifara (0-9) i slova (a-f i A-F) predvođen sa nulom koju sledi slovo x ("0x" ili "0X").

Primer prestavljanja celog broja deset (10) u tri brojna sistema:

• decimalnom: 10.

• oktalnom: 012.

• heksadecimalnom: 0xA.

Brojevi u pokretnom zarezu Brojevi u pokretnom zarezu imaju sledeće delove: decimalni ceo broj, decimalnu tačku ("."), deo iza decimalnog zareza (decimalni ceo broj), eksponent ("e" ili "E", praćen decimalnim celim brojem).

Primeri brojeva u pokretnom zarezu:

• 1.1234

• .1E23

• -1.1E12

• 2E-10

Infinity (ili -Infinity) je vrednost koju će da vrati Java Script ako radimo sa brojevima koji su veći od najvećeg mogućeg broja. Deljenje sa nulom će da generiše Infinity vrednost.

```
var x = 2/0;
```

// x će imati vrednost Infinity

```
var x = -2/0
```

// x će imati vrednost –Infinity

Infinity je broj : typeOf Infinity će da vrati number.

typeOf Infinity; //vraća number

Boolean

Promenljive tipa Boolean mogu imati dve vrste vrednosti: true (tačno) ili false (netačno).

```
var x = true;
```

```
var y = false;
```

Boolean funkcija se može koristiti da utvrdimo da li je neki izraz tačan ili ne.

`Boolean(10>9)` //vraća true

Array objekat

Array objekat se koristi za čuvanje seta vrednosti u jednom imenu promenljive. Niz je kao i u drugim programskim jezicima uređena kolekcija podataka, međutim za razliku od drugih programskih jezika, JavaScript dozvoljava da se u jednom nizu nalaze podaci različitog tipa

(zbog slabe tipiziranosti JavaScript-a). Svakom elementu niza dodeljuje se vrednost indeksa preko koga mu se može direktno pristupiti. Indeksiranje u nizu počinje od nule. Sledeća linija koda definiše objekat array koji je nazvan myArray:

```
var myArray = new Array;
```

Ključna reč *new* se koristi da bi se dinamički kreirao objekat tipa Array. On poziva konstruktor objekta Array() pri čemu se kreira novi objekat tipa *Array*. Veličina niza može biti prosleđena kao argument konstruktora.

```
var myArray = new Array(10);
```

U gornjem primeru je kreiran objekat tipa Array koji ima 10 elemenata. Postoje dva načina za dodavanje vrednosti nizu:

• moguće je dodati koliko god je potrebno vrednosti za definisanje promenljivih koje su potrebne

```
var mojakola=new Array();
```

```
mojakola[1]="Golf";
mojakola[2]="BMW";
```

• moguće je dodati ceo sadržaj za kontrolisanje

```
var mojakola=new Array("Golf", "BMW")
```

Pristup nizu

Moguće je obratiti se posebnom elementu u nizu, obraćajući se imenu niza i broju indeksa. Indeks počinje od 0.

```
document.write(mojakola[0]);
```

Kao rezultat ovog koda pojaviće se: "Golf".

Asocijativni nizovi

Asocijativni niz je niz koji umesto numeričkih vrednosti koristi stringove za indeksiranje vrednosti. Postoji asocijacija između indeksa i vrednosti sačuvane na toj lokaciji. Indeks se obično zove i key a dodeljena vrednost value. Ovi *key/value* parovi su uobičajen način za čuvanje i pristup podacima. U sledećem nizu drzave postoji asocijacija između vrednosti indeksa (skraćenice za državu) i sačuvane vrednosti (celog imena države). Za iteraciju kroz elemente ovakvog niza možemo da koristimo for petlju.

Niz indeksiran stringovima

Rezultat ovog ovog JavaScripta je spisak skraćenica i celih naziva država.

Svojstva Array objekta

Objekat Array ima samo tri svojstva. Najčešće korišćeno svojstvo je length koji određuje broj elemenata niza, tj. njegovu dužinu.

Svojstvo	Opis
constructor	referencira konstruktor objekta Array
length	vraca broj elemenata niza
prototype	prosiruje definiciju niza dodajuci mu svojstva i metode

Sledi primer u kom je prikazana primena svojstava Array objekta.

Svojstva niza

Niz knjiga ima 3 elemenata
Rezultat ovog JavaScripta će biti “Niz knjiga ima 3 elemenata”.

Metode Array objekta

Metoda	Opis
concat()	Dodaje elemente jednog niza drugom nizu
join()	Spaja elemente niza odvojene separatorom i formira string
pop()	Briše i vraća poslednji element niza
push()	Dodaje elemente na kraj niza
reverse()	Obrće raspored elemenata u nizu
shift()	Briše i vraća prvi element niza
slice()	Kreira novi niz od elemenata postojećeg niza
sort()	Sortira niz po abedecnom ili numeričkom redosledu
splice()	Uklanja i/ili zamenjuje elemente niza
toLocaleString()	Vraća niz predstavljen stringom u lokalnom formatu
toString()	Vraća niz predstavljen stringom
unshift()	Dodaje elemente na početak niza

Tabela 10 Metode objekta Array

Concat() metoda

Metoda *concat()* dodaje elemente prosleđene kao argumente u postojeći niz i vraća novoformirani niz.

noviNiz = stariNiz.concat(new elementi);

Primer upotrebe:

Pop() metoda

Pop() metoda briše poslednji element niza i kao rezultat izvršavanja vraća taj skinuti element niza.

var poslednja_vrednost = mojNiz.pop();

Sledi primer upotrebe:

Kao rezultat izvršavanja ovog programa dobićemo da originalni niz ima elemente: Ana, Dragan, Maja, da je skinut element Maja, a novi niz ima elemente Ana,Dragan.

Push() metoda

Metoda *push()* dodaje nove elemente na kraj niza, pri čemu se alocira potrebna memorija.

mojNiz.push(new elementi);

Primer upotrebe metode push():

Kao rezultat izvršavanja gornjeg primera dobićemo ispisane elemente prvobitnog niza: Ana, Vlada, Dragan, Maja i dobićemo ispisane elemente novog niza: Ana, Vlada, Dragan, Maja, Sandra, Tanja.

Metode shift() i unshift()

Metoda *shift()* uklanja prvi element niza i kao rezultat vraća uklonjenu vrednost, dok *unshift()* metoda dodaje element na početak niza. Ove metode su kao pop() i push() sem što se odnose na početak niza a ne na njegov kraj.

Metoda slice()

Metoda *slice()* kopira elemente jednog niza u drugi i ima dva argumenta: prvi je broj koji predstavlja početni element od koga će krenuti kopiranje, a drugi element predstavlja poslednji element (on se ne uključuje). Pri tome treba imati u vidu da je prvi element niza na poziciji " 0". Pritom originalni niz ostaje nepromenjen.

var noviNiz = mojNiz.slice(indeks prvog elementa, indeks poslednjeg elementa);

Primer upotrebe metode slice():

slice() metoda Kao rezultat izvršavanja gornjeg programa dobićemo ispisane elemente

prvog niza: Ana, Vlada, Dragan, Maja, Sandra, kao i elemente novog niza dobijenog korišćenjem slice metode: Dragan, Maja. Metoda splice() Metoda splice() uklanja određeni broj elemenata od neke startne pozicije i dozvoljava zamenu izbačenih elemenata novim. mojNiz.splice(indeks elementa, broj elemenata, [elementi zamene]); Primer upotrebe metode splice():
Kao rezultat izvršavanja gornjeg programa dobićemo ispisane elemente početnog niza: Ana, Vlada, Dragan, Maja, kao i ispisane elemente novog niza: Ana, Sandra, Tanja, Pera, Maja.

Objekat Date

Date objekti se koriste za rad sa datumom i vremenom. Postoji veliki broj metoda za dobijanje informacija vezanih za datum i vreme. Datum je zasnovan na UNIX-ovom datumu počev od 1.januara 1970. i ne podržava datume pre njega. S obzirom da se JavaScript izvršava u čitaču Date objekat vraća vreme i datum lokalnog računara, a ne servera. Ako vreme nije dobro podešeno na korisničkom računaru onda se ni vrednosti neće dobro prikazivati.

Primeri kreiranja Date objekta:

var Date = new Date();

// Konstruktor koji vraća Date objekat

var Date = new Date("July 4, 2004, 6:25:22");

var Date = new Date("July 4, 2004");

var Date = new Date(2004, 7, 4, 6, 25, 22);

var Date = new Date(2004, 7, 4);

var Date = new Date(Milliseconds);

Svojstva Date objekta

Objekat Date ima samo jedno svojstvo prototype.

Metode Date objekta

Date objekat ima veliki broj metoda. U tabeli su prikazane neke od metoda.

Metoda	Opis
getDate	Vraća dan u mesecu
getDay	Vraća dan u nedelji
getFullYear	Vraća godinu ispisanu sa sve 4 cifre
getHours	Vraća sat u danu
getMonth	Vraća broj meseca (Januar – 0)
getTime	Vraća broj milisekundi počev od 1.januara 1970.
parse()	Konvertuje string u date objekat
setDate(value)	Postavlja datum u mesecu
setFullYear()	Postavlja godinu
setHours()	Postavlja sat u danu
setTime()	Postavlja vreme od 1.januara 1970. u milisekundama
toString()	Vraća string objekat koji predstavlja datum i vreme

Primer izvršenja gornjeg koda je:

Lokalno vreme: Mon Feb 29 23:00:31 UTC+0100 2016

Godina je 2016

Vreme je: 23:0:31

Vrednost lokalnog vremena će zavisiti od trenutka izvršavanja programa i biće prikazano vreme na klijentu.

Object type

Kreiranje objekata

Kreiranje objekta se odvija u dva koraka:

1. Definisanje tipa objekta preko pisanja funkcije,
2. Kreirajući instancu objekta sa new. Da bi definisali tip objekta moramo napisati funkciju koja određuje njegovo ime, njegove osobine i metode. Na primer, pretpostavimo da želimo da kreiramo tip objekta za studente. Neka se objekat zove student i neka ima osobine ime, srednje_ime, prezime i indeks. Da bi to uradili potrebno je da napišemo sledeću funkciju:

function student(ime, srednje_ime, prezime, indeks) { this.ime=ime; this.ime_roditelja = srednje_ime; this.prezime=prezime; this.Indeks=indeks; }

Sada možemo da kreiramo objekat student1 pozivajući funkciju student preko new:

student1=new student(“Pera”, “Petar”,”Perić”, ”2012/99”);

Objektu možemo da dodamo i metode. Recimo, funkcija prikaziProfil() koja ispisuje sve podatke o studentu:

function prikaziProfil() { document.write("Ime: " + this.ime + "
"); document.write("Srednje ime: " + this.ime_roditelja + "
"); document.write("Prezime: " + this.prezime + "
"); document.write("Indeks: " + this.Indeks + "
"); }

Sada ćemo ponovo napisati funkciju student() preko koje definišemo objekat.

function student(ime, srednje_ime, prezime, indeks) { this.ime=ime; this.ime_roditelja = srednje_ime; this.prezime=prezime; this.Indeks=indeks; this.prikaziProfil=prikaziProfil; }

Kada definišemo objekat student1, funkciji prikaziProfil() se pristupa preko:

student1.prikaziProfil();

Objekat može da ima svojstvo koje je i samo objekat. Na primer, objekat tipa student, može u sebe uključivati svojstva profesor i predmet, koja su i sama objekti. I tako redom.

Objektu Student možemo dodeliti svojstva pod nazivima ime, ime_roditelja, prezime i Indeks na sledeći način:

*Student.ime=”Pera”; Student.ime_roditelja=”Petar”; Student.prezime=”Perić”; Student.Indeks=”100/99”;***

Ovo nije jedini način pristupa svojstvima objekta. Nizovi su skup vrednosti organizovan po brojevima kome je dodeljeno jedno ime promenljive. Osobine objekta i nizovi su u JavaScriptu direktno povezani, oni zapravo predstavljaju različit način pristupa istoj strukturi podataka. Tako, na primer, osobinama objekta Student možemo pristupiti i na sledeći način, potpuno ekvivalentan prethodnom primeru:

Student[”ime”]=”Pera”; Student[”ime_roditelja”]=”Petar”; Student[”prezime”]=”Perić”; Student[”Indeks”]=”100/99”;

Takođe je moguće prići svakoj od osobina niza i preko indeksa. Tako je potpuno ekvivalentno gornjim načinima ekvivalentan i pristup preko indeksa:

Student[0]=”Pera”; Student[1]=”Petar”; Student[2]=”Perić”; Student[3]=”100/99”;

Metoda je funkcija pridružena objektu. Programer definiše metod na isti način kao što definiše funkciju. Potom tu funkciju pridružuje objektu na sledeći način:

NazivObjekta.NazivMetode = NazivFunkcije

gde je NazivObjekta postojeći objekat, NazivMetode je naziv koji dodeljujemo metodi, a NazivFunkcije je naziv funkcije koju povezujemo sa objektom. Potom možemo pozivati metodu u kontekstu objekta kao:

NazivObjekta.NazivMetode(parametri);

JavaScript ima specijalnu rezervisanu reč, *this*, koja se koristi za referenciranje na trenutni objekt.

Na primer, ako imamo funkciju *Proveri()* koja proverava da li se vrednost osobine objekta nalazi u propisanim granicama:

function Poveri(obj, donja, gornja){ if((obj.value < donja)||(obj.value > gornja)) alert(“Pogrešna vrednost”) }

Objekat Math

JavaScript obezbeđuje mnoštvo matematičkih mogućnosti. Sve ove mogućnosti sadržane su u matematičkom objektu *math*. Ovaj objekt se razlikuje od drugih po tome što se za njegovu upotrebu ne pravi njegova kopija. Skriptovi direktno pozivaju svojstva i metode objekta math i on je deo reference.

round() - metod zaokružuje broj na najbliži integer.

Sintaksa

Math.round(x)

random() - metod vraća nasumice izabran broj između 0 i 1.

Sintaksa

Math.random()

max() metod vraća broj sa najvećom vrednošću od dva specifičbna broja. Sintaksa

Math.max(x,y)

min() - metod vraća broj sa najmanjom vrednosti izmađu dva specifična broja.

Sintaksa

Math.min(x,y)

Function

Java Script Function je deo koda koji izvršava neki poseban zadatak i izvršava se kad se desi neki događaj, kada je funkcija pozvana od strane Java Script koda ili automatski (self-invoked funkcija). Sintaksa za kreiranje funkcije Sintaksa za kreiranje funkcije je:

function imefunkcije(var1,var2,...,varX) { Neki kod }

Var1,var2,..,varX su promenljive ili vrednosti prosleđene funkciji.

Zagrade { } definiše početak i kraj funkcije.

Funkcije bez parametara moraju imati zagrade { } posle naziva imena funkcije. Ime funkcije je praćeno (). Ime funkcije može da sadrži slova, brojeve, donju crtu i znaka za dolar ($). Ne treba zaboraviti značaj velikih slova u JavaScript-u. Reč function mora biti napisana malim slovima, u suprotnom će se desiti greška. Takođe, kada se poziva funkcija njeno ime mora biti navedeno identično kao kada se ona definiše.

Naredba *return* se koristi kada je potrebno odrediti vrednost koju funkcija vraća, i sve funkcije koje vraćaju neku vrednost moraju imati return naredbu.

function proizvod(a,b) { return a*b }

Prethodna funkcija vraća proizvod dva broja. Da bi se pozvala navedena funkcija moraju se proslediti i dva parametra:

product=proizvod(4,5)

Rezltujuća vrednost ove prod() funkcije je 20, i ona će biti smeštena u promenljivu nazvanu product.

Značaj definisanja funkcija se ogleda u tome što funkciju definišemo samo jednom, a možemo je koristiti koliko hoćemo puta (reusable code). Isti kod, sa različitim argumentima će proizvesti različit rezultat.

U Java Scriptu možemo da koristimo funkcije na isti način kao što koristimo varijable.

Npr. var tekst = “Proizvod brojeva 3 i 4 je : “ + proizvod(3,4);

Object oriented programming in JS

Basics

Property types

Defining and reading properties

this explanation

Upotreba ključne reči this u programskom jeziku Javascript u nekim slučajevima razlikuje od upotrebe u drugim porgramskim jezicima (Java, objektni PHP, C# itd.). U većini slučajeva vrednost ključne reči this se razrešava načinom na koji je neka funkcija pozvana. ECMAScript standard definiše this kao ključnu reč čija je vrednost evaluirana u odnosu na trenutni kontekst izvršavanja programa.

Pre svega potrebno je znati da u Javascriptu postoji tri vrste izvršnog koda. To su: globalni kod, kod funkcije i kod evaluacije. Grubo rečeno, globalni kod predstavlja kod koji se nalazi izvan funkcija, kod funkcije je od unutar deklaracije funkcije. Eval kod je kod je pozvan za evaluaciju. Objekat koji je referenciran ključnom rečju this se uvek odredjuje kada kontrola izvršenja programa predje u drugi kontektst. Stoga, vrednost ključne reči this je odredjena sa vrstom koda koja se izvršava i pozivaocem tog koda.

Kada se programski kod izvršava u globalnom kontekstu, vrednost reči this je globalni objekat. U brauzeru globalni objekat je objekat window, dok je u Node.js program globalni objekat jednostavno global object.

Kada se kontekst izvršavanja programa promeni postoji tri slučaja kada se menja vrednost ključne reči this. To su: poziv metode, poziv funkcije ključnom rečju new i poziv funkcija korišćenjem apply i call.

Kada pozovemo funkciju kao telo polje objekta, vrednost reči this se odnosi na taj objekat. Medjutim ukoliko polje kao funkciju prosledimo u novu varijablu i onda funkciju pozovemo preko te varijable onda se vrednost ključne reči this ne menja. Sledeći primer prikazuje ova dva slučaja.

```
var student = {
      brojIndeksa: "11/11"
      prikazi: function() {
          console.log("Student sa brojem indeksa:
"+this.brojIndeksa);
      }
}
student.prikazi();
var prikaz = student.prikazi;
prikaz();
```

U prvom slučaju vrednost reči this je objekat student, dok se u drugom slučaju vrednost ove reči ne menja te je dalje globalni objekat. Na početku izvršavanja programa, vrednost reči this je uvek globalni objekat.

Ukoliko koristimo ključnu reč new kao konstruktor tada se kreira novi objekat i postavlja this na vrednost novokreiranog objekata unutar funkcije koja je pozvana ključnom rečju this.

```
function Student(brojIndeksa){
    this.brojIndeksa = brojIndeksa;
 }
var s1 = new Student("11/11");
var s2 = Student("11/11");
```

U prvom slučaju vrednost reči this unutar funkcije Student je novokreirani objekat, dok je u drugom slučaju reč o običnom pozivu funkcije pa se this odnosi i dalje na globalni objekat.

Javascript ugradjene funkcije call i apply postavljaju eksplicitno vrednost reči this na objekat koji je prosledjen kao prvi argument ove funkcije, dok je drugi argument niz argumenata objekta funkcije na koju this pokazuje. Funkcija apply se ponaša na isti način sa izuzetkom da je da se argumenti prenose pojedinačno a ne kao niz.

Kada direktno pozivamo funkciju eval() vrednost reči this ostaje neizmenjena. U drugom slučaje se odnosi na globalni objekat.

Postoji nekoliko izuzetaka. Na primer, ukoliko se this ključna rec koristi u Funciton.prototype funkcijama onda se this odnosi na vrednost prosledjenu kao argument u tim funkcijama.

Prototype concept

Javascript objekti pored svoji polja imaju i još jedno dodatno polje. To je pokazivač na neki drugi objekat. Ovaj pokazivač nazivamo prototype objekat. Ukoliko pokušamo da pronadjemo neko polje u okviru nekog objekta i to polje se ne pronadje u okviru objekta, Javascript će potražiti to polje u prototype-u. Izvršno okruženje će pratiti lanac uvezanih prototipa sve dok ne naidje na null vrednost. Tada vraća undefined kao povratnu vrednost.

Kada kreiramo objekat korišćenjem Object.create(prot) narednom parameter prot se odnosi na prototype upravo kreiranog objekta. Možemo potražiti vrednost prototype-a nekog objekta naredbom Object.getPrototypeOf. Naredni primer prikazuje kreiranje objekta i prosledjivanje prototype-a.

```
function student(){
        this.fakultet = "FON";
        this.prikazi = function(){
            alert("Ja sam student FON-a");
        }
    }
    var s = Object.create(Student.prototype);
           Object.getPrototypeOf(s);
```

Ispod je prikazan način na koji jednu funkciju svim objektima kreiranim pomoći ključne reči new.

```
student.prototype.getFakultet = function(){
        return this.fakultet;
  };
```

U Javascriptu funkcije su objekti. Objekti kao funkcije su povezani sa Function.prototype-om. Svaka objekat funkcije je kreirana sa poljem prototype. Njegova vrednost je objekat sa poljem constructor čija vrednost je funkcija. Pošto su funkcije objekti one se mogu koristiti kao i svi drugi objekti (paramteri drugih funkcija, mogu biti povratna vrednost itd.).

Ukoliko kreiramo novi objekat korišćenjem ključne reči new, taj objekat će posedovati polje **proto**. Vrednost ovog polja je prototype vrednost funkcije kojom je kreiran objekat. Korišćenjem jedne reči možemo kreirati vise objekata. Svi ti objekti će deliti jedan prototype. Na taj nači menjanjem prototype, promene će automatski naslediti i svi objekti. Prototypes se koriste za realizaciju nasledjivanja u Javascript programskom jeziku.

Object creation

Factory pattern

Constructor pattern

Prototype pattern

Inheritance

Prototype chaining

Parasitic inheritance

Prototypal inheritance

Combined inheritance

Event handling

JavaSrcipt se, kao i mnogi drugi skriptni jezici, izvršava sekvencijalno. Sa druge strane, kako pre svega služi za upravljanje ponašanjem web strana, sadrži i mehanizme koji omogućavaju odgovor na interakciju korisnika ili samog pretraživača sa stranom, u onom trenutku kada do te interakcije dođe. Ti mehanizmi nazivaju se „event handlers” (upravljači događajima).

Događaj je, kako je već rečeno, bilo koja interakcija pretraživača ili korisnika sa internet stranom. Kako se svi dešavaju nad elementima DOM stabla, nazivaju se i događajima DOM-a („DOM events”). Mogu se podeliti na:

```
1. Mouse events
2. Keyboard events
3. Frame/Object events
4. Form events
5. Drag events
6. Clipboard events
7. Print events
8. Media events
9. Animation events
10. Transition events
11. Server-sent events
12. Misc events
13. Touch events
```

Mouse events su događaji koji se realizuju pri interakciji strelice miša ili „scroll”-a (točkića) i internet strane. **Keyboard events** se realizuju pri interakciji tastature i strane. Predstavljaju odziv strane u odnosu na pritiskanje, držanje ili sklanjanje prsta sa nekog dugmeta na tastaturi. **Frame/Object events** tipično predstavljaju događaje koji ne moraju dolaziti od korisnika, kao što su kraj učitavanja nekog DOM elementa, greška u učitavanju spoljnog fajla, kada se zaustavi učitavanje resursa... ali se odnose i na neke događaje koji jesu vezani za korisničku interakciju, kao što je promena veličine prozora ili „skrolovanje”. **Form events** se događaju pri interakciji korisnika sa nekom formom, poput promene sadržaja input elemenata, resetovanja forme ili slanja („submit”). **Drag events** se događaju kada se element „vuče” mišem. **Clipboard events** omogućavaju strani da reaguje na kopiranje, sečenje i lepljenje („copy”, „cut”, „paste”) sadržaja unutar elemenata. Print events se dešavaju odmah pre nego što strana počne da se štampa, ili u trenutku kada započne sa

štampanjem. **Media events** su reakcije na različite promene stanja media fajlova (zvučni fajlovi, video fajlovi...). **Animation events** se realizuju pri izvršavanju CSS animacija. Mogu se vezati za početak, ponavljanje i kraj animacije. **Transition events** sastoje se, zapravo, samo iz jednog događaja – kraja CSS tranzicije. **Server-sent events** izvršavaju se u tri slučaja: kada se desi greška na izvoru događaja, kada dođe do prijema poruke sa izvora događaja i kada se otvori konekcija sa izvorom događaja. **Misc events** su razni nesvrstani događaji. **Touch events** predstavljaju događaje izazvane interkacijom korisnika i ekrana osetljivog na dodir („touch screen”) i tipično se dešavaju kada se internet strana pregleda na nekom pametnom telefonu, tabletu i sl.

Dalje će biti dati primeri najčešće korišćenih događaja, kao i neke naprednije teme poput događaja baziranih na vremenu, propagacije događaja i korišćenja „callback” funkcija (o kojima će biti reči u kasnijim poglavljima).

Onload event handler

Prvi event handler koji će biti obrađen je onload. Događaj koji mu odgovara je kraj učitavanja nekog DOM elementa. Posebno je koristan kada želimo da „odložimo” izvršavanje skripte na trenutak kada je web strana završila učitavanje, a tada se najčešće i koristi.

Naime, CSS kod je uputno staviti na početak HTML dokumenta, jer ne bi imalo puno smisla da učitavamo stranu bez stilova i „layout”-a, pa da ih tek nakon toga primenjujemo. Iz tog razloga se link tagovi obično stavljaju unutar head taga. Sa druge strane, većina funkcionalnosti i animacija koje izvršava JavaScript, dešavaju se u već učitanoj strani. Jedan pristup je da se script tag nalazi na samom dnu body elementa. Kako se, zbog čitljivosti, i JavaScript kod ili link ka JavaScript dokumentu nalaze u head-u, kod se enkapsulira u onload event handler. Time se kraj učitavanja svih HTML elementa i CSS pravila eksplicitno određuje kao početni trenutak izvršavanja skripte.

Ovo se može uraditi na više načina. Prvi primer pokazuje korišćenje event handler HTML atributa:

```
<!DOCTYPE html>
  <head>
    <title>Onload event</title>
  </head>
  <body onload="alert('Učitavanje stranice je završeno!');">
  </body>
</html>
```

Drugi primer je korišćenje onload atributa window elementa direktno iz JavaScript-a:

```
<!DOCTYPE html>
  <head>
    <title>Onload event</title>
    <script >
          window.onload = function () {
              alert('Učitavanje stranice je završeno!');
          }
      </script>
  </head>
  <body>
  </body>
</html>
```

Oba primera imaće isti rezultat – kada se strana učita, prikazaće se alert box sa porukom „I have loaded!”. Naravno, u realnoj situaciji, unutar anonimne fuknkcije koja se poziva bio bi obuhvaćen sav JavaScript kod na toj strani, ili barem onaj deo koji je potrebno pozivati nakon što je strana učitana.

Mouse events

Kao što je prethodno napomenuto, Mouse events podrazumevaju najrazličitije akcije izvršene upotrebom miša. Onclick event handler podrazumeva funkciju koja će se izvršiti u slučaju da korisnik jednom klikne mišem na unapred definisan element na stranici. Opšti oblik upotrebe:

```
<button onclick="funkcija()">Klikni</button>
```

Sada pogledajmo dva primera koji ilustruju rad ovog handler-a.

```
<!DOCTYPE html>
<html>
<body>

    <p>Za priakz trenutnog vremena kliknite na dugme ispod.</p>

    <button onclick="getElementById('vreme').innerHTML=Date()">Koliko je sati?</button>

    <p id="vreme"></p>

</body>
</html>
```

U ovom primeru, obradili smo događaj klika na dugme, tako što smo napisali inline funkciju kojom pristupamo paragrafu čiji je id vreme i menjamo mu sadržaj. Za tu svrhu iskristili smo svojstvo innerHTML.

Naravno, ovaj handler možemo iskoristiti na bilo kom elementu na našoj stranici.

```
<!DOCTYPE html>
<html>
<body>

    <p id="demo" onclick="myFunction(this, 'red')">Oboj me.</p>

    <script>
        function myFunction(elmnt,clr) {
            elmnt.style.color = clr;
        }
    </script>

</body>
</html>
```

U ovom slučaju, pozivamo funkciju koju smo definisali u okviru script tagova a koja ima dva ulazna parametra. Prvi je element na koji će se akcija primeniti, u našem primeru to je upravo paragraf na koji je potebno kliknuti, pa zato prosleđujemo this. Drugi predstavlja boju kojom ćemo obojiti dati paragraf.

Javascript, naravno, podržava i događaj dvostrukog klika mišem. Celokupna primena se izvodi analogno situaciji kada smo implementirali onclick event handler. Jedina razlika je što bismo u ovoj situaciji koristili ondblclick handler. Sledeća dva događaja koja ćemo obraditi su onmouseenter i onmouseleave. Oni se najčešće pojavljuju u paru i to u situacijama kada želimo da na neki način promenimo element preko koga korisnik pređe kursorom. Da bi detaljnije ovo pojasnili, pogledajmo sledeći primer:

```
<!DOCTYPE html>
<html>

<body>

    <div id="demo" style="width:200px; height:200px; background-color:blue;"
    onmouseenter="enter()" onmouseleave="leave()"></div>

    <script type="text/javascript">
        function enter() {
            document.getElementById("demo").style.backgroundColor = 'green';
        }

        function leave() {
            document.getElementById("demo").style.backgroundColor = 'blue';
        }

    </script>

</body>
</html>
```

U prethodnom primeru smo definisali jedan div ciji je id demo, a koji ćemo koristiti kako bi pristupili baš tom elementu u funkcijama. Inicijalno, boja tog diva je plava. Preko funkcija koje odgovaraju događajima onmouseenter i onmouseleave menjaćemo boju tog diva.

Keyboard events

Sada ćemo obraditi događaj onkeypress. Odnosno, prikazaćemo upotrebu sledeća dva događaja: • Onkeydown • Onkeyup Razlog je u tome što kombinacija upravo ova dva događaja predstavlja prvobitno navedeni onkeypress.

```
<!DOCTYPE html>
<html>
<body>


    <input type="text" id="demo" onkeydown="keydownFunction()" onkeyup="keyupFunction()">

    <script>
        function keydownFunction() {
            document.getElementById("demo").style.backgroundColor = "red";
        }

        function keyupFunction() {
            document.getElementById("demo").style.backgroundColor = "green";
        }
</script>

</body>
</html>
```

Naravno, veoma često se susrećemo sa različitim prečicama koje možemo iskoristiti u mnogim veb aplikacijama. Prirodno, postavlja se pitanje kako obraditi događaj kada korisnik pritisne kombinaciju dva ili više dugmeta. Upravo u sledećem primeru, obradićemo takvu situaciju.

```
<!DOCTYPE html>
<html>
<head>
    <title>Untitled Document</title>
</head>
<body>

    <script type='text/javascript'>
        function kombinacija(e) {
            var evtobj = window.event? event : e

            if (evtobj.keyCode == 90 && evtobj.ctrlKey) alert("Ctrl+z");
        }

        document.onkeydown = kombinacija;

    </script>

</body>
</html>
```

Počnimo od samog poziva funkcije. Ova notacija, `document.onkeydown = kombinacija;` , samo je još jedan način za aktiviranje event handlera, kada želimo da to izvršimo u okviru neke skripte. Analogno tome, mogli smo u prethodnim primerima napisati i object.onkeydown=function. Sledeće što možemo primetiti je kreiranje objekta ovog događaja, evtobj. Ovo činimo kako bi saznali koja kombinacija dugmića je pritisnuta. Neki od njih su unapred definisani, kao što su ctrl, alt, shift, a ostalima se pristupa preko svojstva keyCode koje ima različite vrednosti u zavisnosti od toga koje je dugme u pitanju. U našem slučaju, ispituje se da li je vrednost tog svojstva 90 jer je upravo to kod koji odgovara slovu z na tastaturi. Ceo spisak ovih kodova možete pogledati ovde.

Još jedan često korišćeni handler je onsubmit. Upravlja ponašanjem strane nakon potvrde forme, tj. „submit”-ovanja:

```
<!DOCTYPE html>
  <head>
    <title>Onload event</title>
    <script>
      function printContent() {
        var content = document.getElementById('text-input').value;
        document.getElementById('content-container').innerHTML = content;
        alert('Strana će biti ponovo učitana.');
      }
    </script>
  </head>
  <body onsubmit="printContent()">
    <form>
      <fieldset>
        <input type="text" id="text-input">
        <button type="submit">Završi</button>
      </fieldset>
    </form>
    <p id="content-container"></p>
  </body>
</html>
```

Kada korisnik klikne na dugme „Završi”, koje je tipa submit, „okida” se onsubmit event handler. Iz text box-a se uzima uneti tekst i prikazuje unutar

elementa ispod forme. Kako se, po „default”-u, stranica ponovo učitava pri submit event-u, poziva se i funkcija alert() koja zadržava to stanje kako bi efekat bio očigledan.

U realnoj situaciji, nakon ovog događaja bi se pozivala funkcija koja upravlja validacijom podataka, obaveštavanjem korisnika o eventualnim greškama i slanja podataka serveru tj. preuzimanja podataka sa servera.

JSON handling with JS

Istorija

```
  **JSON** ja nastao iz potrebe da se real-time komunikacija
između brauzera i klijenta odvija bez korišćenja plugin-ova kao
što su Flash ili Java apleti koji su se koristili u ranim 2000-
im. Daglas Krokford je prvi precizirao i popularizovao JSON
format. JSON.org web sajt je osnovan 2002. godine, a 2005. Yahoo
je počeo da nudi neke svoje servise u JSON formatu. Google je
2006. počeo da nudi JSON feed za svoj GData protokol.
 ** JavaScript Object Notation** predstavlja sintaksu za čuvanje
i razmenu podataka. Ova notacija ima jako jednostavnu strukturu
gde se podaci predstavljaju po principu ključ: vrednost. JSON je
nastao kao alternativa XML standarda za razmenu podataka. Danas
se dosta više koristi JSON u odnosu na XML zato što je
jednostavniji i lakši za korišćenje. U nastavku teksta biće dat
primer nekog jednostavnog JSON fajla i biće dat primer istih
podataka predstavljenih u XML-u, pa je na taj način najlakše
sagledati razliku između ova dva formata.
  JSON može raditi sa različitim tipovima podataka. Neki od
podržanih tipova podataka su: boolean, number, string, object,
array, null.
```

Najjednostavniji primer JSON fajla može izgledati ovako: { "ime":"Milan", "prezime":"Kalinić" } Oznaka koja se koristi za definisanje niza vrednosti u JSON fajlu su uglaste zagrade([]). Primer nekog jednostavnog niza JSON vrednosti nalazi se ispod: "zaposleni":[{"ime":"Ivan","prezime":"Zeljković"}, {"ime":"Katarina","prezime":"Šišmanović"}, {"ime":"Sanja","prezime":"Marinković"}]

```
  Sličnost izmedju XML-a i JSON-a je u tome što su oba ova
standarda samoopisjuća. I jedan i drugi standard predstavljaju
hijerhijske strukture, vrednosti se mogu ugnježdavati unutar
drugih vrednosti. I XML i JSON se mogu parsirati od strane
većine programskih jezika, a takodje i jednom i drugom se može
pristupiti preko XMLHttpRequesta.
  Ipak, postoje i značajne razlike između ova dva standarda za
razmenu podataka. JSON zapis je kraći, a istovremeno i brži za
čitanje i pisanje. JSON ne koristi kao oznaku
krajnji(zatvarajući tag). Za razliku od XML-a JSON može da
koristi nizove. Ipak ono što se često navodi kao najveća razlika
je da za XML da bi se parsirao mora postojati XML parser, a kod
JSON-a je to obična JavaScript funkcija.
```

Sama JSON sintaksa predstavlja podskup od JavaScript sintakse. JSON sintaksa je izvedena iz JavaScript notacije za kreiranje objekate. Zbog toga je jako jednostavno raditi sa JSON-om u okviru JavaScript-a. Na slici ispod je primer kreiranja varijable koja prima neki niz vrednosti kao ilustracije veze izmedju JSON-a i JavaScripta.

```
  Elementima ovog niza može se pristupiti na više načina.

  ![](slika11.bmp)



  Ovakav pristup će nam vratiti prvi element niza. Isti efekat
će biti ako napišemo sledeći kod:

  Naravno, na isti način se mogu menjati vrednosti nekim
elementima niza.
```

Primer ispod ilustruje neke od razlika koje su navedene kao najbitnije između JSON i XML formata. Vidi se da je JSON format lakši za čitanje i razumevanje.

```
  Zašto koristiti JSON? – Za aplikacije koje koriste AJAX, JSON
je dosta lakši i brži. Najčešća primena je kod aplikacija koje
čitaju neke podatke sa web servera i prikazuju te podatke na web
stranici.
  AJAJ(Asynchronous JavaScript and JSON) je ista metodologija
kao Ajax samo što se umesto XML-a, koristi JSON kao format
podataka. AJAJ je web development tehnika koja omogućava da web
stranica zahteva nove podatke od servera nakon što je učitana
putem brauzera. Na primer kada korisnik nešto kuca kao
kriterijum pretrage, klijentski kod aplikacije to šalje serveru
asinhrono koji na to odgovora odmah sa nekim predlozima
pretrage. Na slici ispod je ilustrovan ovaj koncept tako što je
prikazan kod klijentske aplikacije koja koristi XMLHttpRequest
da bi pokupila podatke sa servera u JSON formatu.
```

Obrada JSON-a u JavaScript-u Da bismo dobili JSON objekat iz stringa, možemo koristiti JSON.parse() metodu. Sintaksa metode: JSON.parse(text[, reviver]) Text– obavezni parametar, predstavlja JSON u tekstualnom formatu. Reviver– opcioni parametar, predstavlja funkciju koja može transformisati vrednost parametara pre nego što se dobije JSON objekat kao povratna vrednost metode JSON.parse(). Povratna vrednost metode je JSON objekat. U slučaju da se ne može generisati JSON objekat od teksta koji je prosleđen metodi, baciće se SyntaxError izuzetak koji označava da uneti string nije validan JSON. Da bismo dobili string od JSON objekta, možemo koristiti JSON.stringify() metodu. Sintaksa metode: JSON.stringify(value[, replacer[, space]]) Value – obavezni parametar, predstavlja JSON objekat koji želimo da konvertujemo u string. replacer – opcioni parametar, može biti funkcija ili niz. Kao funkcija prima dva parametra, ključ i vrednost parametra. Povratna vrednost je vrednost parametra koju želimo uključiti u rezultat. Primer:

Promenljiva stringJson će imati vrednost:

Svi parovi ključ-vrednost čija vrednost je string su isključeni iz rezultata. Ukoliko replacerprosledimo kao niz čije su vrednosti nazivi ključeva, to znači da će se samo ti kluč-vrednost parovi javiti u rezultatu.

Sada, promenljiva stringJson će imati vrednost:

Space – opcioni parametar, predstavlja nivo indentacije pri štampanju rezultata. Može imati vrednost do broj 10, što predstavlja indentaciju od 10 karaktera (praznih mesta). Za simulaciju prettyPrint-a (lepog i čitljivog ispisa JSON-a), može se proslediti ‘\t’.   JSON Schema JSON Schema predstavljaspecifikaciju JSON formata. Koristi se za:

1. Opisivanje tipa podataka

2. Jasnu i čitljivu dokumentaciju za ljude, kao i za mašine
3. Kompletna strukturna validacija, pogodna za validiranje podataka koji se nalaze u datom JSON-u Primer JSON Schema-e:

Na sajtu http://json-schema.org/mogu se naćisveključnerečikoje se mogukoristitiprikreiranjujedne JSON Schema-e. Uzpomoć ove šeme bi mogao da se validira sledeći JSON dokument:

Ukoliko bi brojGodina imao negativnu vrednost, navedeni JSON dokument ne bi prošao validaciju. Postoje razne biblioteke koje se mogu koristiti za validaciju i pomoći da aplikacija koju pravite bude bolja. Primer poziva validacije nekog JSON dokumenta po nekoj JSON Schema-i:

Na ovaj način bi smo prošli kroz JSON dokument koji sadrži podatke o našim korisnicima, i proverili da li su svi podaci o našim korisnicima validni.

Asynchronous JS

AJAX, Web sockets

Uvod u AJAX

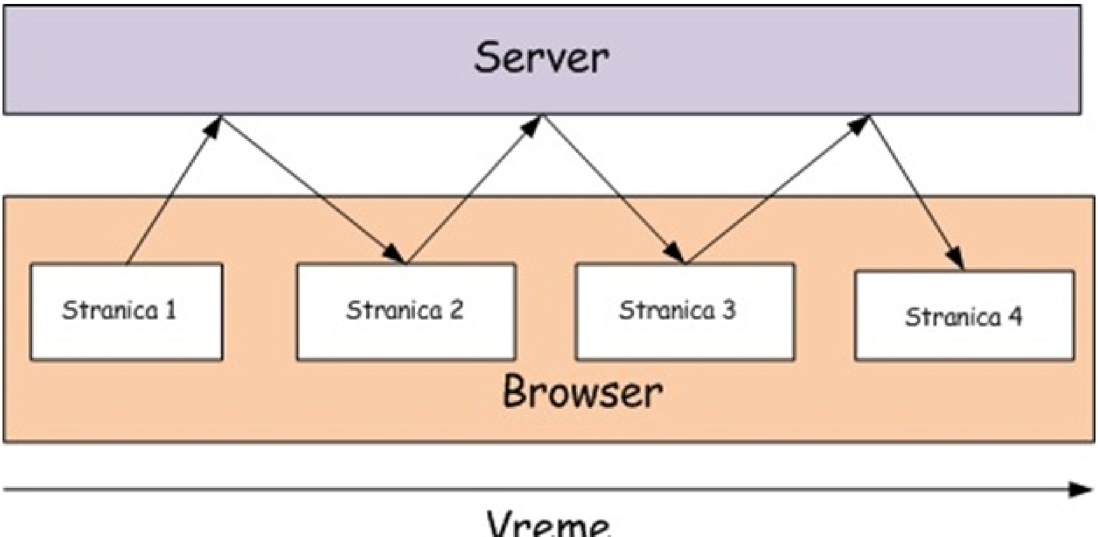
AJAX je skraćenica za Asynchronous JavaScript and XML. Prosto rečeno, AJAX omogućava prikaz podataka na web stranici bez potrebe za ponovnim učitavanjem iste. Svakodnevno imamo priliku da vidimo razne primere upotrebe AJAX-a na najpoznatijim sajtovima poput Facebook-a, Gmail-a, Twitter-a.

Primer sa kojim su se bez sumnje susreli svi je Google auto suggest koji nakon nekoliko unetih slova predlaže reči koja sadrže ta slova kako bi nam ubrzao pretragu. Naravno, to se čini bez učitavanja nove stranice, već nam se podaci dinamički prikazuju.

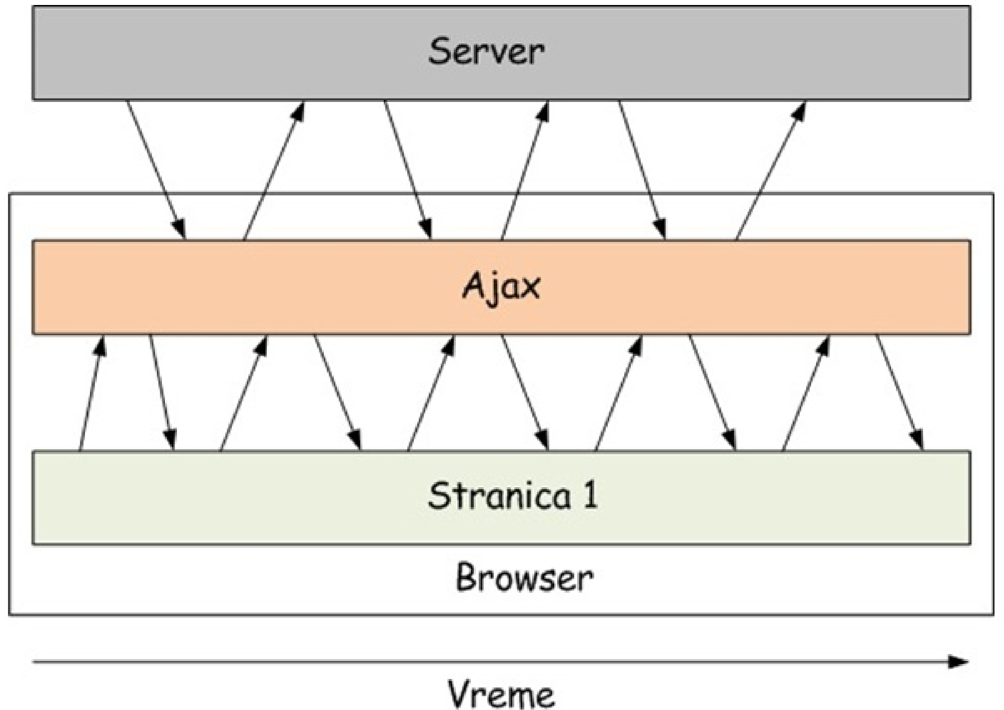
Da bi se shvatio koncept AJAX tehnologija, neophodno je objasniti razliku između sinhronog i asinhronog prenosa podataka između klijenta i servera.

U slučaju sinhronog prenosa podataka sa klijenta se sekvencijalno šalju zahtevi. Dakle, zahtev poslat sa klijenta stižu na server koji ih zatim obrađuje, a nakon toga ih vraća klijentu. // Kod sinhronog načina rada procesi se izvode sekvencijalno. Dakle, osnovni proces komunikacije između klijenata i servera može se opisati na sledeći način: klijent šalje zahtev ka serveru, podaci se prenose ka serveru, server obrađuje podatke i zatim ih vraća klijentu. Ono što je primetno sa slike ispod je da postoji vremenski interval između pravljenja zahteva i odgovora na isti. Za vreme tog intervala klijent čeka, bez mogućnosti da pravi nove zateve ili nastavi svoje korišćenje web aplikacije.

With traditional web pages and applications, every time a user clicks on something, the browser sends a request to the server, and the server responds with a whole new page. Even if your user’s web browser is smart about caching things like images and cascading style sheets, that’s a lot of traffic going back and forth between their browser and your server... and a lot of time that the user sits around waiting for full page refreshes.



Kod asinhronog prenosa podataka između klijenata i servera karakteristično je da se korisnička upotreba web aplikacije nikada ne prekida. Dakle, korisnik ne čeka da se učita nova stranica već dobija odgovore u okviru date stranice



Callbacks

S obzirom da su funkcije u Javascriptu objekti, možemo ih čuvati u varijablama, prosleđivati kao argument funkcijama, kreirati ih unutar funkcija i vraćati ih kao povratnu vrednost funkcije. Ako prosledimo funkciju kao argument drugoj funkciji, mi kasnije možemo izvršiti

prosleđenu funkciju, ili je vratiti kao rezultat kako bi se izvršila kasnije. Ovo je suština korišćenja callback funkcija u JavaScriptu.

Callback funkcija, poznata i kao funkcija višeg reda, je funkcija prosleđena drugoj funkciji kao parametar, a koja će biti izvršena unutar te druge funkcije.

Evo čestog primera korišćenja callback funkcija u JQuery-ju:

```
$("#btn1").click(function() {
  alert("Na btn1 je kliknuto!");
});
```

Primećujete da parametar koji je prosleđen metodi "click" nije varijabla, već funkcija. Prosleđena funkcija biće izvršena unutar metode "click".

Primer iz čistog JavaScripta:

```
var prijatelji = ["Stefan", "Đole", "Milan", "Tamara"];

prijatelji.forEach(function (ime){
console.log(ime);
// "Stefan" "Đole" "Milan" "Tamara"
});
```

Ponovo, primetite da smo prosledili anonimnu funkciju (funkciju bez imena) forEach metodi kao parametar.

Prosleđeni parametar nije pozivanje funkcije, već samo njena definicija. Ona neće biti pozvana odmah, već kasnije, pa je tako i dobila ime (called back). Iz prvog primera, funkcija se poziva svaki put kad neko klikne na dugme "btn1".

Pored anonimnih, moguće je proslediti i imenovane funkcije. Sledeći primer je malo komplikovaniji.

```
var prijatelji = ["Stefan", "Đole", "Milan", "Tamara"];

function ispisi(podaci){
    podaci.forEach(function(podatak){
        console.log(podatak);
    });
}

function ulaz(data, callback){
    callback(data); // ispisi(prijatelji)
}

ulaz(prijatelji, ispisi);
```

Funkcija "ulaz" kao parametar prima niz stringova i funkciju. Ona poziva funkciju koja joj je prosleđena sa podacima kao argumentom.

Promise

Promise u JavaScriptu predstavlja rezultat asinhrone operacije. U njemu će se sačuvati rezultat uspešnog izvršenja, ili razlog neuspeha.

Promise donosi jednostavniji način za izvršenje, sastavljanje i upravljanje asinhronim operacijama u poređenju sa tradicionalnim pristupom baziranim na callback funkcijama. Takođe, uz pomoć njih moguće je upravljati asinhronim greškama na način koji je sličan tradicionalnom `try/catch` .

Promise može biti u jednom od tri stanja:

- Pending (na čekanju): Ishod promise-a još nije poznat, zato što asinhrona operacije nije još završila.
- Fulfilled (ispunjen): Asinhrona operacija je završena, i promise je dobio vrednost.
- Rejected (odbijen): Asinhrona operacija nije uspela, i promise nikada neće biti ispunjen. U ovom stanju, promise ima reason(razlog) zašto operacija nije uspela.

Kada je promise u stanju pending, on može preći u stanje fulfilled ili rejected. Kada jednom promeni stanje u fulfilled ili rejected, on više nikada neće moći da ponovo promeni stanje.

Primena promise-a

Primarna metoda za promise je `then`, koja registruje callback metode koje će primiti ili uspešnu vrednost izvršenja, ili razlog zašto se promise ne može ispuniti.

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

    var data = $.get('http://localhost:60139/AJAX/data.json',
function(data){
        if (data.success) {
            resolve(data);
        }
        else {
            reject(Error("It broke"));
        }
    });
});

promise.then(function(result) {
    console.log(result); // "Stuff worked!"
}, function(err) {
    console.log(err); // Error: "It broke"
});
```

AJAX polling

JSONP and CORS

```
{
  "isChanged" : true,
  "content" : "Koji sam ja meni kralj",
  "newText" : "Changed me!"
}
```

Uvod u WebSocket-e

WebSocket predstavlja protokol koji je razvijen kao deo HTML5 inicijative, kako bi se definisao JavaScript interfejs, kojim bi se omogućila perzistentna veza između klijentske Web stranice i servera preko jednog soketa, gde se poruke mogu razmenjivati u oba smera u svakom trenutku (full duplex bidirectional communication over a single TCP connection).

Perzistentna veza znači da je veza otvorena i nakon razmene prve poruke. Kao što znamo, HTTP protokol funkcioniše na principu pitanja i odgovora, gde klijent pošalje zahtev serveru za podacima, a server odgovara u vidu podataka klijentu, nakon čega se veza zatvara. Za svako pitanje-odgovor potrebna je jedna TCP veza.

Web soketi nam omogućavaju da razmenjujemo podatke sa serverom preko jedne TCP veze. I to ne samo jednu poruku, već koliko god želimo, gde nam server može poslati podatke kada kod želi, bez eksplicitne potrebne da ih klijent zatraži. Ovo predstavlja glavnu benefit Web soketa, jer preko jedne veze možemo razmenjivati više poruka, dok nam server može slati podatke kada god želi, bez ikakvih ograničenja.

WebSocket JavaScript klijent - DEMO

Napravite folder sa nazivom Chat-App. U okviru tog foldera, napravite još dva foldera: css i js. Takođe napravite i fajl `index.html`.

U folderu css smestite `bootstrap.min.css` fajl. Takođe napravite fajl `style.css`.

U folderu js napravite fajl app.js. U tom fajlu pisaćemo sav JavaScript kod. Takođe napravite folder lib, i u njega smestite `bootstrap.min.js` fajl, kao i `jqery.min.js` fajl.

Klijent je implementiran uz korišćenje JavaScript jezika, uz pomoć biblioteka Jquery. Za skroman izgled stranice je korišćen Bootstrap.

Kod `index.html` stranice:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Chat app</title>
    <link rel="stylesheet" type="text/css"
href="css/bootstrap.min.css">
    <link rel="stylesheet" type="text/css" href="css/style.css">
    <script src="js/lib/jquery-1.11.3.min.js"></script>
    <script src="js/lib/bootstrap.min.js"></script>
</head>
<body>
    <div class="container">
        <div class="row">
            <div class="col-md-10 col-md-offset-1">
                <h1><span class="label label-
default">Chat</span></h1>
                <form class="form-group">
                    <div class="form-control"
id="chatResponses"></div>
                    <div class="row">
                        <div class="col-md-2">
                            <input class="form-control input-lg"
id="name" type="text">
                        </div>
                        <div class="col-md-10">
                            <input class="form-control input-lg"
id="chatInput" type="text">
                        </div>
                    </div>
                </form>
            </div>
        </div>
    </div>
</body>
<script src="js/app.js"></script>
</html>
```

U head delu referenciramo sve potrebne biblioteke. Samu aplikaciju referenciramo na dnu stranice, ispod body dela. Stranica sadrži deo za prikaz poruka od svih klijenata, deo za unos nadimka, kao i deo za unos teksta za slanje.

Kod `style.css` fajla:

```
#chatResponses {
    max-width: 100%;
    height: 300px;
    border: 1px solid #ccc;
    padding: 5px;
    overflow: auto;
    overflow-x: hidden;
    text-wrap: normal;
    margin-bottom: 10px;
}

p {
    font-family: Verdana, Geneva, sans-serif;
    font-size: 14px;
    font-style: normal;
    font-variant: normal;
    font-weight: 400;
    line-height: 20px;
}
```

Da biste koristili WebSocket-e, potrebno je inicijalizovati promenljivu korišćenjem WebSocket konstruktora, koji kao ulazni parametar prima url endpointa koji se nalazi na serveru. U fajl `app.js` , napišite sledeću liniju koda:

```
var socket = new WebSocket('ws://localhost:8080/chat-
server/chat');
```

URL endpointa počinje sa ws, koji označava WebSocket protokol. Postoji mogućnost slanja preko sigurno konekcije, i u tom slučaju bi prefiks bio wss. WebSocket protokol predstavlja apstrakciju nad HTTP protokolom.

Nakon oznake protokola sledi adresa i port servera, i na kraju, tačna adresa WebSocket endpointa na pomenutom serveru.

Nad promenljivom socket moramo da implementiramo određene metode koje su nam neophodne za rad sa WebSocket-ima (`onOpen` , `onClose` , `onError` , `onMessage`). Ove metode kao ulazni argument primaju neki event, koji se u okviru metode obrađuje.

OnOpen metoda

Ako se pri kreiranju soketa uz pomoć konstruktora `WebSocket('url')` uspešno uspostavi konekcija, izvršiće se ova metoda.

```
socket.onopen = function open (e) {
    console.log('Veza sa serverom je uspesno uspostavljena:
WebSocket Open event');
};
```

OnClose metoda

Pri zatvaranju konekcije između klijenta i servera, izvršiće se ova metoda.

```
socket.onclose = function close (e) {
    console.log('Veza sa chat serverom je prekinuta: WebSocket
Close event');
};
```

OnError metoda

Ukoliko dođe do neke greške u konekciji, izvršiće se ova metoda.

```
socket.onerror = function error(e) {
    console.log('Doslo je do greske: WebSocket Error event');
};
```

OnMessage metoda

Kada stigne poruka sa servera, izvršiće se ova metoda.

```
socket.onmessage = function message (e) {
    var data = JSON.parse(e.data);
    var node = document.createElement("P");
    var text = document.createTextNode(data.name + ' | ' +
data.chatText);
    node.appendChild(text);
    document.getElementById("chatResponses").appendChild(node);
};
```

U ovom primeru, poruke koje će dobijati klijent od servera su u JSON formatu i sadrže nadimak klijenta i tekst koji je klijent otkucao.

Primer ulaza:

```
{
    name: "PETAR",
    chatText: "Hello World!!! "
};
```

Kako se ulaz preko eventa e dobija u tekstualnom formatu, potrebno ga je konvertovati u JSON objekat metodom `JSON.parse()`. Svim informacije koje se šalju putem WebSocket-a se nalaze u `e.data` promenljivoj.

Zatim, kreiramo node element (HTML `<p></p>` tag), kao i text element (tekst koji ćemo smestiti u okviru node elementa). Text element smeštamo u okviru node elementa metodom `appendChild()`. Takođe, istom metodom dodajemo node element div elementu na `index.html` stranici koji ima `id=″chatResponses“`.

Slanje podataka putem WebSocket-a

U ovoj demo aplikaciji, korisnik u input polje koje ima `id=″name″` unosi svoj nadimak. Takođe, u okviru input polja koje ima `id=″chatInput″` unosi tekst koji želi da pošalje. Pritiskom na taster `Enter`, uneto ime i tekst poruke se pretvaraju u JSON objekat i šalju na server putem metode `send()`.

```
$("#chatInput").keyup(function (e) {
    if (e.keyCode == 13) {
        var nickname = document.getElementById("name").value;
        if(nickname == "" || nickname == null) {
            alert('Unesi ime!');
            return;
        }
        var text = document.getElementById("chatInput").value;
        if (text == "" || text == " " || text == null){
            return;
        }

        var jsonData = {
            name: nickname,
            chatText: text
        };

        socket.send(JSON.stringify(jsonData));
        document.getElementById("chatInput").value = "";
    }
});
```

Korišćenjem Jquery biblioteke, dohvatamo element sa `index.html` stranice koji ima `id=”chatInput”`, i nad njim pozivamo metodu keyup, koja reaguje na svako pritiskanje tastera. Ako je šifra tastera jednaka broju 13, to znači da je korisnik pritisnuo `Enter` na svom računaru.

U promenljivoj `nickname` ćemo smestiti nadimak klijenta, ako ga je uneo. Ako nadimak nije unet, ispisaće se upozorenje o grešci. Ako je input polje za unos teksta prazno, izvršavanje metode će se prekinuti.

Kada su uneti i nadimak i tekst, smeštaju se u okviru JSON objekta sa nazivom jsonData.

Kako bismo poslali na server taj JSON objekat, potrebno je da nad soketom pozovemo metodu `send()` i prosledimo joj taj JSON objekat u vidu teksta (uz pomoć metode `JSON.stringify()`). Nakon slanja podataka na server, brišemo tekst iz polja za unos teksta.

Napomena:

Pomoću `send()` metode možemo slati tri tipa podatak:

1. Text (UTF)

2. BLOB
3. ArrayBuffer

To takođe znači da možemo i prihvatiti `onMessage()` ove tipove podataka. Da bismo mogli da svaki tip obradimo na odgovarajući način, potrebno je da proverimo koji tip podatak nam je stigao putem WebSocket-a.

Načini za proveru:

```
e.data instanceof ArrayBuffer

e.data instanceof Blob

typeof e.data === "string"
```

Korisni linkovi:

1. http://kaazing.com/websocket/
2. https://html.spec.whatwg.org/multipage/comms.html#network
3. https://os.alfajango.com/websockets-slides/#/
4. http://www.html5rocks.com/en/tutorials/websockets/basics/
5. https://tools.ietf.org/html/rfc6455

Error handling and debugging

Javascript i HTML5

Events, Web workers, Web storage, IndexedDB, etc.

Advanced JS

Custom events, drag and drop, function bindings, closures and callbacks...

Patterns in JS

Kristofer Aleksander je dao sledeću definiciju paterna: *„Svaki patern je trodelno pravio, koje uspostavlja relaciju izmedju izmedju nekog problema, njegovog rešenja i njihovog konteksta. Patern je u isto vreme i stvar koja se dešava u stvarnosti, i pravilo koje govori kada i kako se kreira navedena stvar.“*

Paterni predstavljaju gotova rešenja koja nalaze primenu prilikom projektovanja i implementacije softvera. Paterni projektovanja su paterni koji su nezavisni od bilo kakve implemntacione tehnologije. Dakle paterni predstavljaju opšta, apstraktna rešenja odredjenih problema koja se javljaju prilikom projektovanja objektno-orijentisanog softvera i koja su predložena rešavanju konkretnih problema. Odredjeni patern je definisan sa imenom, problemom, rešenjem i posledicama. Ime se upotrebljava za opis nekog paterna u nekoliko kratkih rečenica. Problem opisuje situaciju i kontekst prilikom kojeg se patern koristi. Rešenje opisuje opšte odnose medju elementima koji uzimaju učešća u rešavanju problema. Posledice opisuju rezultate primene paterna, ali takodje i opisa opravdanosti upotrebe paterna. Rešenja iz paterna se mogu ponovo koristiti.

Paterni su proverena rešenja, rešenja iz paterna se mogu ponovo koristiti što je i cilj paterna i paterni su pogodni jer imaju jasno vidljivu strukturu problema i rešenja. Paterni projektovanja se dele na kreacione paterne, strukturne paterne i paterne ponašanja. Kreacioni paterni su usesredjeni na kreiranje objekata koji su kreirani prikladnu za situaciju u kojoj se nalazimo (Constructor, Factory, Abstract, Prototype, Singleton i Builder). Strukturni paterni se odnose na strukturu svakog objekta i pronalaženje načina za ostvarivanje veze izmedju njih (Decorator, Facade, Flyweight, Adapter, Proxy itd.). Paterni ponašanje se odnose na unapredjenje komunikacije izmedju objekata (Iterator, Mediator, Observer, Visitor itd.).

Kratak prikaz Javascript „klasa“

Koncept paterna je blisko povezan sa konceptom klasa u objektno orijentisanom razvoju softvera. Najpopularniji pristup jeste definisanjem funkcija i kreiranje objekata korišćenjem ključne reči new uz naziv funkcije. Ključna reč this se može koristiti u okviru funkcija za definisanje osobina i metoda za objekte. U narednom delu je prikazana funkcija Drzava koja ima ulogu klase, zatim njeno „instanciranje“ i pozivanje metoda.

```
function Drzava(brojstanovnika) {

    this.naziv = "Srbija";
    this.glavnigrad = "Beograd";
    this.povrsina = 88361;
    this.brojstanovnika = brojstanovnika;

    this.prikazi = function() {
        return this.nastavnik + " " + this.glavnigrad + " " +
this.povrsina + " km na kvadrat" + this.brojstanovnika;
    }
}

var srb = new Drzava(7164824);
document.write(srb.prikazi());
```

Rezultat izvršenja navedenog progama možemo pogledati na sledećoj slici.

Srbija Beograd 88361 km na kvadrat 7164824

Builder patern

Kada radimo sa DOM elementima, često želimo da kreiramo te elemente dinamički a ne odmah na početku. Ovaj proces se može znatno usložniti povećanjem kompleksnosti aplikacije. Builder patern omogućava kreiranje objekata samo specifirajući sadržaj nekog objekata., nezavisno od toga kako se objekat kreira, razdvajajući kreiranje od reprezentacije nekog objekta. Mi dakle definišemo strukturu nekog objekta bez potrebe za kreiranjem samog objekta. Naredni primer prikazuje primer korišćenja Builder paterna.

```
var Builder = function() {
  var korisnickoIme = "nenad";
  var sifra = "nenad";

  return {
      promenaKorisnickogImena : function(korIme) {
        korisnickoIme = korIme;
        return this;
      },
      promenaSifre : function(pass) {
        sifra = pass;
        return this;
      },
      build : function() {
        return "Korisnicko ime je: " + korisnickoIme +", Sifra
je: " + sifra;
      }
  };
};

var builder = new Builder();

console.log(builder.build());

var promena1 =
builder.promenaKorisnickogImena("pejovic").promenaSifre("pejovic"
).build();

var promena2 = builder.promenaSifre("admin").build();
```

Predhodni primer prikazuje primer “klase” Builder koja ima metodu build, promenaKorisnickogImena i promenaSifre. Pozivanjem metode build ispisuje se vrednosti polja korisničko ime i šifra koja imaju svoju podrazumevanu vrednost. Pozivanjem odredjenih metoda menjaju se ova polja vrednostima prosledjenim kroz parametre te ponovno pozivanje metode build daje različite rezultate sa novom šifrom i korisničkim imenom.

Singleton patern

Ovaj patern obezbedjuje klasi jedinstveno pojavljivanje i jedinstven globalni pristup do tog jedinstvenog pojavljivanja. Jedinstvena instanca se naziva singleton. Ovaj patern je pogodan u situacijama kada neke sistemske akcije moraju biti koordinisane sa jednog mesta. Primer je klasa koja je zadužena sa ostvarivanje konekcije prema bazi podataka. Sledeći primer prikazuje implementaciju Singleton paterna u Javascript programskom jeziku.

```
var Singleton = (function() {

    var instanca;

    function inicijalizacija() {

        function metoda1() {
            alert("Singleton patern!");
        }

        function metoda2() {
            alert("Kreacioni paterni!");
        }
        return {
            prikaziNazivPaterna: metoda1,
            prikaziKategorijuPaterna: metoda2
        };
    }

    function vratiInstancu() {
        if (!instanca) {
            instance = inicijalizacija();
        } else {
            return instanca;
        }
    }
    return {
        vratiInstancu: vratiInstancu;
    }
})();

Singelton.vratiInstancu.prikaziNazivPaterna();
```

Navedeni primer prikazuje klasu Singleton koja ima ima metodu vratiInstancu i polje instanca. Ova metoda, ukoliko je polje instanca nedefinisano, poziva u sebi metodu inicijalizuje i povratnu vrednost ove metode stavlja u promenljivu instanca. Povratna vrednost metode inicijalizuj je objekat sa dve definisane metode.

Decorator patern

Decorator patern produžuje dodatne odgovornosti do objekta dinamički. Ovaj patern obezbedjuje fleksibilnost u izboru klasa koje proširuju funkcionalnost. Klase koje pružaju funkcionalnost dekoracije nisu esencijalne sa tačke gledišta sistema. U našem javascript primer umesto dodavanja podklase, dodavaćemo polja i metode već kreiranim objektima.

```
function Plata() {
    this.osnovnaPlata = function(){
        return 15000;
    };
}

function bonusi(plata) {
    var op = plata.osnovnaPlata();
    plata.osnovnaPlata = function() {
        return op + 5000;
    };
}

function topliObrok(plata) {
    var op = plata.osnovnaPlata();
    plata.osnovnaPlata = function() {
        return op + 2000;
    };
}

var plata = new Plata();
bonusi(plata);
topliObrok(plata);
console.log(plata.osnovnaPlata());
```

U ovom primeru je izvšen override objekta osnovnaPlata super-objekta Plata u funkcijama u kojima smo dodavali bonuse i topli obrok na vrednost osnovne plate.

Adapter patern

Adapter patern konvertuje interfejs neke klase ili objekta u drugi interfejs koji klijent očekuje. Adapter patern omogućuje zajednički rad klasama i interfejsima koji inače ne bi mogli da funkcionišu zajedno zbog nekompatibilnih interfejsa. Adapter patern uzima objekat koji smo kreirali, i umotava ga u drugi objekat koji odgovara nekom trećem objektu sa kojim prvi objekat treba da komunicira.

```
function pretraga() {
    this.pronadjiKorisnika = function(username, pass) {
        return "Nenad Pejovic"
    }
}

function naprednaPretraga() {
    this.postaviUsername(username) {};
    this.postaviPassword(password) {};
    this.finalnaPretraga() {
        return "Nenad Pejovic";
    }
}

function Adapter(username, password) {
    var pretraga = naprednaPretraga;
    return {
        pronadjiKorisnika: function(username, password) {
            pretraga.postaviPassword(password);
            pretraga.postaviUsername(username);
            return pretaga.pronadjiKorisnika();
        }
    }
}

function Client() {
    var pretraga = new pretraga();
    var username = "admin";
    var username = "admin";

    var adapter = new Adapter(username, password);
    var pr = pretraga.pronadjiKorisnika(username, password);
    var naprPr = adapter.pronadjiKorisnika(username, password);
}
```

Adapter prilagodjava objekat pretraga objektu naprednaPretraga

Chain of responsibility pattern

Izbegava čvrsto povezivanje izmedju pošiljaoca zahteva i njegovog primaoca, obezbedjuje lanac povezanih objekata, koji će da obradjuje zahtev sve dok se on ne obradi. Klijent objekat pokreće zahtev za obradom, dok handler objekti obradjuju zahteve. Sledeći primer prikazuje primer korišćenja Chain of responsibility paterna na primeru automata koji vraća kusur u apoenima za 20, 10, 5, 2 i 1.

```
var vracanjeKusura = function(kusur) {
    this.kusur = kusur;
}

vracanjeKusura.prototype = {
    vratiApoene: function(apoen) {
        var apoeni = Math.floor(this.kusur / apone);
        this.kusur -= apoeni * apoen;
        console.log(apoeni);
        return this;
    }

}
};

function Klijent() {
    var vracanjeKusura = new vracanjeKusura(214);
}
vracanjeKusura.vratiApoene(20).vratiApoene(10).vratiApoene(5).vr
atiApoene(5).vratiApoene(1);

}
```

State patern

State patern dopušta objektu da promeni ponašanje kada se menja njegovo interno stanje. Svaki objekat predstavlja specifično stanje.

```
var ukljuci = function(){

    var ukljucen = "ne";

};

promeniKanal.prototype = function(){
    if(this.ukljucen === "ne"){
        this.ukljucen = "da";
    } else{
        this.ukljucen = "ne"
    }

}
```

Ovaj primer prikazuje promenu stanja nekog uredjaja koja u zavisnosti od toga da li je uredjaj uključen ili ne, vrši se promena njegovog stanja.

Useful JavaScript libraries

JavaScript je skriptni programski jezik, koji se izvršava u web pretraživaču na strani korisnika, umesto na strani servera (dakle, u browseru, a ne na web serveru). JavaScript može menjati sadržaj prikazane web stranice i kontrolisati pretraživač. Može se koristiti za animacije, navigaciju, jednostavne igre, kalkulatore, validaciju formi... Dok su stranice bile čisti HTML, kad bi kliknuo na neki deo stranice, ponovno bi se učitavao celi HTML od sledeće stranice. Danas, sa javascriptom, to nije potrebno. Umesto dugogodišnjeg slogana “World Wide Wait” koja je upućivala na period frustracije nekoliko miliona korisnika koji su čekali da server obradi zahtevanu stranicu u potpunosti pre njenog prikaza na ekran računara, AJAX tehnologija je, ukratko, ponudila mogućnost parcijalnog ažuriranja nekog segmenta web stranice a da se pri tome njen kompletan sadržaj ne učitava iznova. Na ovaj način su funkcionalnosti web aplikacija podignute na viši nivo uz efikasnije korišćenje mrežnih resursa.

Programeri koriste JavaScript za puno različitih web aplikacija zato što on stvarno može da učini da strana klijenta – odnosno, strana korisnika - izgleda lepo i dobro funkcioniše. Ako uvek dodajete sve više i više kodova da bi JavaScript funkcionisao u više različitih pretraživača može da postane veoma velik i zbunjujuć. Tako je programiranje popularnih funkcionlanosti web stranica na klijentskoj strani pomoću JavaScript jezika iz godine u godinu postojalo sve teže. Pojava novih igrača na tržištu internet pretraživača je dodatno zakomplikovala postupak programiranja internet aplikacija s obzirom na činjenicu da su postojale međusobne razlike u interpretiranju web stranica pa i samog koda koristeći JavaScript. Danas, uz pomoć JavaScript biblioteka se ruše ove razlike među pretraživačima. Gotovo da svaka JavaScript biblioteka danas poseduje podršku za lakšu realizaciju AJAX zahteva, manipulaciju odgovorima servera i jednostavnije parcijalno ažuriranje web stranice. U zavisnosti od nivoa kompleksnosti efekata koje želite da postignete nad elementima neke web stranice, realizacija tih efekata je u “čistom” JavaScript programiranju ponekad zahtevala i na desetine redova programskog koda umesto samo par linija koda pomoću neke JavaScript biblioteke.

Biblioteka je skup već napisanih JavaScript funkcija (malih programa) koji vam omogućavaju da brže pišete JavaScript kod. Korišćenjem raznih biblioteka zasnovanih na JavaScript jeziku unosite strukturu u vaš JavaScript kod i održavate ga organizovanim. JavaScript biblioteke su rešenja otvorenog koda pa se konstantno poboljšavaju. Zahvaljujući JavaScript bibliotekama, upravljanje DOM-om i njegovim elementima je uštedelo vreme mnogim web programerima da realizuju određene funkcionanosti web stranica. Tako se npr. pomoću JavaScript biblioteka na jednostavniji način vrši selekcija i pristup željenim elementima stranice, iščitavanje i izmena njihove vrednosti i vrednosti njihovih atributa, njihovo

dodavanje ili uklanjanje u/iz DOM stabla, promena izgleda elemenata manipulacijom njihovog CSS koda itd.. Takođe, jedna od podržanih oblasti JavaScript biblioteka jeste upravljanje događajima. JavaScript je po prirodi event-driven skript jezik, odnosno skript jezik koji zagovara interaktivnost između korisnika i sadržaja neke web strane. Ta interakcija se najčešće zasniva na korišćenju računarskih periferija poput miša i tastature gde se kroz niz mogućih događaja (klik miša, prevlačenje krusora miša preko nekog elementa stranice, pritiskom na određeno dugme tastature itd.) mogu izvršavati napisani JavaScript kodovi. JavaScript biblioteke u tom pogledu teže da omoguće pravilno registrovanje pomenutih događaja i izvršavanju određenog koda ukoliko se dati događaj okine.

JS Biblioteke

Neke od najčešće korišćenih opštih biblioteka među programerima su jQuery, jQueryUI, MooTools, YUI, Dojo, Prototype, React i mnoge druge. Sve imaju svoje prednosti i mane a razlike se ogledaju u razvoju određenih funkcionalnosti. Sve biblioteke teže da olakšaju programiranje u JavaScript programskom jeziku ali neke su više okrenute ka jednoj od osnovnih funkcionalnosti biblioteka koje smo gore pomenuli dok druge teže da olakšaju i unaprede rad nekih drugih aspekata. Neke biblioteke su okrenute ka DOM-u i manipulaciju sa HTML elementima dok neke teže da unaprede rad sa JSON objektima.

Na internetu se mogu naći mnogi pluginovi koji olakšavaju rad programerima jer imaju već predefinisane neke funkcionalnosti i stilove sa kojima je lako manipulisati i prolagođavati ih sebi i datoj situaciji. Pluginovi služe za rad sa slikama, video zapisima, datumima, formama, animacijama, elementima na samoj strani, prikzu određenih stvari… Oblast gde se dosta koriste razni pluginovi je i prikaz i rad sa tabelarnim prikazom podataka na samoj web stranici. Najkorišćeniji pluginovi za te svrhe su jqGrid, Flexigrid, Gridster, OmniGrid, TableKit, Gridsteck.

Njihova prednost je u tome što već imaju implementirane funkcionalnosti za tabelarni prikaz podataka, manipulaciju sa podacima, stilizovan prikaz i dr. za šta bi vi izgubili dosta vremena da radite sami na svakom projektu. Već prilikom inicijalizacije ovakvog plugin-a dobijate osnovni prikaz i osnovne njegove karakteristike. Naravno kod je potpuno otvoren i ostavljena vam je mogućnost personalizacije i prilagođavanja svojim potrebama i što se tiče samog izgleda, boja, kolona, redova, veličine, fonta i dr. ali i što se tiče samog funkcionisanja tabela i nekih dodatnih opcija.

Implementira se veoma lako tako što downloadujete fajl (može da sadrži i .js i .css fajlove), ubacite u svoj folder i pozovete na samoj web strani unutar head taga.

```
<link rel="stylesheet" href="http://YOURJQGRIDURL/jqGrid-4.5.2/css/ui.jqgrid.css" type="text/css" />
```

```
<script type="text/javascript" src="http://YOURJQGRIDURL/jqGrid-4.5.2/js/jquery.jqGrid.src.js" />
```

Pored navedenih biblioteka, postoji i čitav niz biblioteka koje se koriste za vizuelizaciju podataka i crtanje različitih vrsta grafikona, kao što su: D3.js, Chart.js, FusionCharts, jqPlot, Dimple, Highcharts, Plotly i mnoge druge. Upotreba bilo koje od ovih biblioteka zavisi kako od problema koji je pomoću njih potrebno rešiti, tako i od vrste i tipa podataka koje je potrebno vizuelizovati, ali i ličnih preferenci programera. Velika većina ovih biblioteka omogućava razvoj prilično komplikovanih i zahtevnih vizuelizacija bez posedovanja bilo kakvog naprednog programerskog znanja, što predstavlja njihovu ogromnu prednost.

jQuery

Jedna od najkorišćenijih JavaScript biblioteka je jQuery. jQuery je JavaScript biblioteka koja pojednostavljuje korištenje JavaScripta kod komplikovanih stvari kao što su AJAX pozivi (Asynchronous JavaScript and XML) ili manipulacija DOM-om (Document Object Model). Omogućava vam da web sajtu dodate funkcionalnosti koje sa čistim html/css-om ne biste mogli da uradite. jQuery slogan „write less, do more“ dosta govori – sa jednom linijom jQuery koda možete pozvati funkcionalnosti za koje bi vam trebalo mnogo više rada i truda, ako bi koristili klasični JavaScript metod kodiranja. jQuery obuhvata mnoge zadatke koji zahtevaju veliki broj linija JavaScript-a i omotava ih u metode koje možete pozvati sa samo jednom linijom. Njegova najveća prednost sa funkcionalne strane je da je potpuno cross-browser kompatibilan, tj. jednako ga ispravno tumače i Internet Explorer i Mozzilla kao i Chrome pa je bojazan od grešaka značajno umanjen.

Funkcionalnosti jQuery-ja:

- HTML/DOM manipulacija
- CSS manipulacija
- HTML event metodi
- Efekti i animacije
- AJAX

Neki od bitnih prednosti su:

- Drastično smanjuje količinu potrebnog koda u odnosu na čist JavaScript što ga čini lakim za čitanje
- Poboljšava performanse vašeg sajta
- Lakši za učenje u odnosu na JavaScript
- Organizovana dokumentacija i prilično velika zajednica za podršku
- Mnogo gotovih stvari koje možete odmah dodati na web stranu
- Smanjuje vreme potrebno za razvoj aplikacije
- Lakše optimizovanje sajta za različite web pretraživače

Dodavanje jQuery biblioteke na web stranice je jednostavno i sastoji se od dva koraka:

- Preuzimanje jQuery biblioteke sa sajta jQuery.com
- Pozivanje u head delu HTML koda. jQuery biblioteka je samo jedan JavaScript fajl i pozivate ga u okviru script tag-a unutar head sekcije. `<script type="text/javascript" src="jquery.js"></script>`
- Pozivanjem same biblioteke sa nekog od servera i tada ne morate imati sam JavaScript fajl u svom projektu. `<script src="//code.jquery.com/jquery-1.12.0.min.js"></script>`

Postoje dve verzije dostupne za preuzimanje (production i development).

- verzija za produkciju – verzija koda za vaš web sajt koji je već objavljen, zbog toga što je minimalizovana i kompresovana.
- development verzija – verzija za testiranje i razvoj (nekompresovan i čitljiv kod).

Jedna od velikih prednosti jQuery biblioteke nad ostalim bibliotekama JavaScript jezika je u tome što je to poprilično "živ" projekat. Pod tim podrazumevamo da veliki tim ljudi radi na projektu i da se stalno usavršava i unapređuje i dorađuje. Tim ljudi koji radi na razvoju jQuery-ja je toliko posvećen tome da se ova biblioteka razvija kroz dva projekata tj. kroz dve verzije koje se stalno dopunjuju. Verzija 1.X je verzija jQuery-ja koja podržava i starije web pretraživače kao što su Internet Explorer u verzijama 6,7 i 8 i starije verzije drugih pretraživača dok druga verzija 2.X nema mogućnosti podrške na ovim starijim verzijama web pretraživača. Ovo je iz razloga što se time smanjuje sam kod jer se neke funkcije razlikuju od pretraživača do pretraživača pa kako se ovi pretraživača polako izbacuju iz upotrebe tako se i kod i funkcionalnosti prilagodjavaju samo potrebama koje se javljaju u praksi i time dobijamo struktuiran i pregledan kod bez nepotrebnih funkcionalnosti. Pa tako implementirate verziju prema korisnicima za koje smatrate da će koristiti vašu web stranicu .

Navešćemo primer kako jQuery kod olakšava samo programiranje. Da bi JavaScript sintaksom pristupili nekom elementu sa ID-jem moj_div i pridružili mu određenu CSS klasu, kreiraćemo sledeći izraz:

```
document.getElementById('moj_div').className='neka_klasa';
```

Isto ovo u jQuery sintaksi postizemo izrazom:

```
$('#moj_div').addClass('neka_klasa');
```

jQueryUI

jQuery je već korišćen da bi se pravili neki impresivni efekati i pluginovi, od kojih su neki bili dovoljno korisni da se opravda njihovo uključivanje u osnovni paket jQuery biblioteke. Međutim, tim programera koji radi na usavršavanju biblioteke je odlučio da u cilju

zadržavanja fokusa jezgra biblioteke izdvoji te dodatne funkcionalnosti na višem nivou i upakuje ih u jednu dodatnu biblioteku koja se nalazi na vrhu jQuery bibliteke.

Ta biblioteka je nazvana jQuery User Interface (obično skraćeno na samo jQueryUI), i obuhvata skup korisnih efekata i naprednih vidžeta koji su dostupni i visoko prilagodljivi kroz korišćenje tema. jQueryUI je kolekcija grafičkih vidžeta, animiranih, vizuelnih efekata i tema realizovanih na jQuery-ju. Slajderi, text boksevi, datum polja, selektori, i drugi - svi spremni da se koristi. Možete da potrošite gomilu vremena na pravljenje ovih funkcionalnosti u jQuery-ju ali jQueryUI kontrole su prilagodljive i dovoljno sofisticirane da Vaše vreme možete bolje potrošiti radeći nešto drugo.

D3.js

Prvo ime na koje većina programera pomisli kada se spomene softver za vizuelizaciju podataka je **D3.js (Data-Driven Documents).** On koristi HTML, CSS i SVG da renderuje zaista sjajne grafikone i dijagrame. Bilo kakvu vizuelizaciju koju možete zamisliti, možete napraviti koristeći D3.js biblioteku. D3 olakšava povezivanje podataka sa DOM-om. U suštini, to je skup gotovih funkcija za iteraciju i obradu elemenata DOM stabla web stranice. D3 omogućava dinamičko kreiranje, izdvajanje i obradu elemenata u DOM-u. O Ono što je takođe bitno je činjenica da su skoro sve njegove komponente open-source. Sa druge strane, dve izuzetno bitne činjenice o kojima treba voditi računa prilikom upotrebe D3.js biblioteke na nekom projektu su prilično **strma kriva učenja** i njegova **kompatibilnost sa modernim verzijama browser-a**. Tako da je preporuka da ovu biblioteku koristite samo kada imate dovoljno vremena i nije vam bitno da li se vaši grafikoni mogu prikazati u starijim verzijama browser-a ili ne.

Veoma ga je lako implementirati, preuzimanjem .zip arhive sa D3 web stranice ili direktnog linkovanja sa iste: `<script src="https://d3js.org/d3.v3.min.js"> </script>`

Prikazaćemo i jednostavan primer korišćenja D3 selekcija, a ostale primere možete pronaći na https://d3js.org. Naime, modifikovanje elemenata koristeći W3C DOM API je prilično naporno. Da bismo npr. samo promenili boju teksta naslova nekog grafika kojeg pravimo, potrebno je napisati sledeći kod u javascript-u:

```
var paragraphs = document.getElementsByTagName("h1");
for (var i = 0; i < paragraphs.length; i++) {
    var paragraph = paragraphs.item(i);
    paragraph.style.setProperty("color", "white", null);
}
```

Koristeći D3 selekcije prethodni kod možemo napisati na sledeći način:

```
d3.selectAll("h1").style("color", "white");
```

Plotly

Kao što smo već napomenuli, D3.js može uraditi skoro sve što možete zamisliti, ali je i činjenica da je potrebno dosta vremena utrošiti na učenje ukoliko želimo postići takve rezultate. **Plotly** je biblioteka koja omogućava mnogo brži razvoj i dosta je lakša za učenje od D3.js biblioteke. Plotly je takođe biblioteka za vizuelizaciju podataka napravljena na D3.js biblioteci i stack.gl-u. Plotly poseduje više 20 različitih vrsta grafika, uključujući 3D i statističke grafike, kao i SVG mape.

Implementacija je takođe jednostavna, bilo da preuzimate minifikovani plotly.js izvorni kod koji nakon toga uključujete u html stranicu: `<script src="plotly-latest.min.js"></script>` ili umesto toga koristite plotly.js CDN link: `<script src="https://cdn.plot.ly/plotly-latest.min.js"></script>`

Da biste napravili grafikon, potrebno je u HTML dokumentu kreirati prazan DIV u koji ćete ucrtati grafik:

```
<div id="tester" style="width:600px;height:250px;"></div>
```

A nakon toga pravite interaktivni plotly.js grafik koristeći Ploty.plot() funkciju.

```
<script>
tester = document.getElementById('tester');
Plotly.plot( tester, [{
    x: [1, 2, 3, 4, 5],
    y: [1, 2, 4, 8, 16] }],
    { margin: { t: 0 } } );
</script>
```

Upotreba JavaScript-a pri izradi aplikacija za mobilne uređaje

Jedna od ključnih prednosti korišćenja web tehnologije za pravljenje aplikacija je prenosivost. Korišćenje programskog prevodioca, kao što je PhoneGap, može se preneti aplikacija i instalirati različitim platformama. Postoji veliki broj biblioteka i okvira koji to omogućavaju. PhoneGap programski prevodioc obezbeđuje niz JavaScript API-ja koji se povezuju sa izvornim funkcijama mobilnih uređaja kao što su kamera, kompas, kontakti i

lokacije i omogućava nam da napravimo mobilnu aplikaciju bez korišćenja nativnog programskog jezika platforme na kojoj uređaj radi. Umesto nativnih jezika možemo koristiti okvire i biblioteke JavaScript jezika za pravljenje mobilnih aplikacija. Najveća prednost hibridnih mobilnih aplikacija se zasniva na pristupu "napravi jednom a pokreni svuda".

jQuery Mobile

jQuery Mobile je jedan od paketa koji se koriste za izradu hibridnih aplikacija. Fanovi jQuery zajednice su izgradili ovaj paket na temeljima čvrstog jQuery i jQueryUI. Cilj jQuery Mobile okvira je da obezbedi programerima da prave web aplikacije i mobilne aplikacije koje rade besprekorno i na mobilnim telefonima, tabletima i računarima. Ne fokusira se mnogo na pružanje nativnog izgleda i osećaja aplikacije individualnim platforme kao što su iOS ili Android. jQuery Mobile podržava širok spektar različitih platformi, od stonih, pametni telefon, tableta ili uređaja E-čitača kao što su Nook i Kindle. Slično kao i jQueryUI, jQuery Mobile obuhvata niz grafičkih kontola i komandi koje su optimizovane za mobilne i uređaja sa ekranima osetljivim na dodir.

Mobile AngularUI

Uz ovaj okvir dobijate najbolje fukncionalnosti iz Bootstrapa 3 i AngularJS okvira za pravljanje mobilnih aplikacija korišćenjem HTML5 standarda. Mobile Angular UI koristi Fastclick.js i Overthrov.js za kontinuirano i bolje korišćenje mobilnoih aplikacija. Omogućava kontrole za pravljenje korisnićkih komponenti kao što su preklapanja, dugme, sidebar i apsolutna pozicioniranja navigacije bez odskakanja od koncepta prilikom skrolovanja. To su komponente koje nedostaju Bootstrapu 3 za izgradnju mobilnih aplikacija.

JavaScript tools

U ovom poglavlju predstavljeno je nekoliko, danas najpopularnijih, JavaScript alata koji mogu biti jako korisni kako početnicima tako i iskusnijim JavaScript developer-ima.

O svakom alatu napisano je kratko objašnjenje čemu služi i postavljeni su linkovi na kojima se može više saznati o alatu, način korišćenja, razlozi korišćenja i slično, ili gde se mogu download-ovati isti. Treba pratiti nove trendove i nove alate koji se danas jako često i brzo razvijaju jer se i oni sami prave kako bi rešili neke određenje probleme sa kojima se programeri susreću i na taj način mogu umnogome pomoći. Sa druge strane, ne treba preterivati sa njihovim korišćenjem i upotrebljavati ih tamo gde im nije mesto i treba izbegavati slučajeve kada njihovo koriščenje nepotrebno narušava performanse projekta.

Za svaki od predstavljenih alata, kao i za sve buduće alate koje ćete koristi, predlaže se testitranje i isprobavanje na nekim vašim test fajlovima i projektima kako bi se detaljije upoznali sa alatom i razumeli njegov rad i uvideli da li vam je zaista potreban. Biće alata o kojima ćete čitati ali dok ih ne isprobate u praksi nećete biti svesni njihove „moći“ i dobrobiti njihovog korišćenja, zato istražujte i kucajte...

Verzioniranje koda (Code Versioning)

Sistemi za kontrolu verzija koda (version control systems) se baziraju na konceptu praćenje promena koje se dešavaju unutar direktorijuma ili fajlova čije promene želite da pratite. Zavisno od sistema koji se koristi, ovo može da varira od informacija o tome koji fajlovi su imali izmene do informacija o konkretnoj liniji koda koja je izmenjena.

Najpopularniji su Git i Subversion. Njihovo korišćenje vam omogućava da imate nekoliko verzija koda, što vam daje mogućnost lakog povratka na prethodnu verziju koda ukoliko dođe do neke veće greške ili niste zadovoljni rešenjem. Mogu se kreirati grane (branch) koje su kopije izvornog koda i koje vam omogućavaju rad i implementaciju novih funkcionalnosti bez rizika uništavanja postojećeg izvornog koda (codebase).

Takođe, možete koristi servise kao što su GitHub, Beanstalk ili Bitbucket, kako bi čuvali vaš kod u cloud-u, što je izuzetno korisno kada je u pitanju grupni rad na projektu.

https://bitbucket.org/
https://git-scm.com/
https://github.com/

Node.js – npm

Node Package Manager (npm) omoguća dve glavne funkcionalnosti:

1. Online repozitorijum za node.js pakete/module koji se mogu pronaći na https://www.npmjs.com/
2. Mogućnost da se koristeći komandnu liniju instaliraju gore pomenuti Node.js paketi kao i upravljanje verzijama i međusobnim zavisnostima Node.js paketa.

Sam **npm** dolazi zajedno sa Node.js instalacijom. Više od Node.js-u i njegovu instalaciju možete pronaći na https://nodejs.org/en/ . Npm omogućava jako brzo i jasno instaliranje (uvođenje u vaš projekat) mnoštvo biblioteka (dependencies) i modula koji vam mogu ubrzati i olakšati rad. Sama instalacija i deinstalacija modula je jako jednostavno, jednom komandnom linijom:

```
$ npm install <Modul Name>
$ npm uninstall <Modul Name>
```

Treba voditi računa o delokrugu (scope) modula. Njih je moguće instalirati Globalno i Lokalno. Ukoliko se ne naglasi pri instalaciji da se modul treba instalirati globalno, on će se instalirati lokalno. Pod lokalno podrazumeva se `node_modules` direktorijum koji se nalazi u folderu (projektu) gde je Node pokrenut. Lokalno instaliranom modulu u projektu se može pristupiti require() metodom. Globalno instalirani paketi se smeštaju u sistemski direktorijum i moguće ih je koristiti u CLI (Command Line Interface), međutim nije ih moguće uključivati u projekat required() metodom.

Da bi koristili npm potrebno je da aplikacija sadrži manifest fajl (sa meta podacima) koji se naziva package.json. On generalno sadrži sve potrebne informacije vezane za aplikaciju kao i listu modula koji su uključeni i potrebni aplikaciji. Primer package.json fajla :

```
{
  "name": "My new App",
  "version": "1.0.0",
  "description": "My new App description",
  "author": "Faculty of Organizational Sciences" ,
  "main": "index.js",
  "scripts" {
    "start": "webpack --progress --colors --watch"
  },
  "dependencies": {
    "react": "^0.14.5",
    "react-dom": "^0.14.5"
  },
  "devDependencies" : {
    "babel-core": "^6.3.26",
    "babel-loader": "^6.2.0",
    "babel-preset-es2015": "^6.3.13",
    "babel-preset-react": "^6.3.13",
    "webpack": "^1.12.9"
  }
}
```

Ovaj fajl nije neophodan za rad, međutim ukoliko aplikacija treba biti javna, ili ukoliko se treba više ljudi uključiti na projekat, package.json olakšava mnoge stvari jer njega je potrebno smestiti u root folder i jednostavnom komandom:

```
$npm install
```

automatski će se instaliti svi moduli koji su navedeni u samom package.json fajlu.

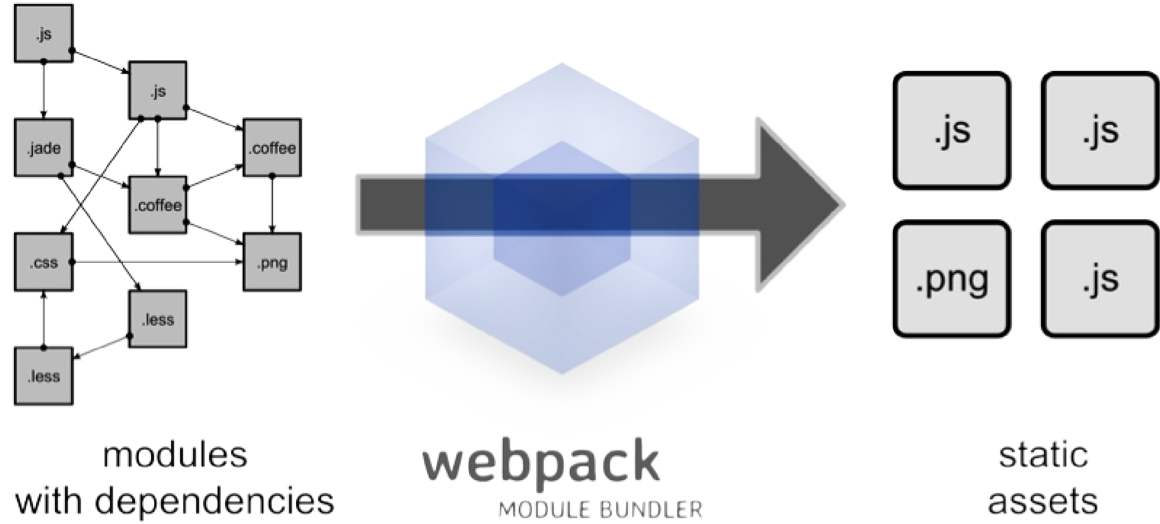
Više o samom npm-u : https://www.npmjs.com/

O načinu pisanja `package.json` -a i njegovim mogućim atributima : https://docs.npmjs.com/files/package.json

Webpack

Webpack je modularni bundler (module bundler-svežanj), koji omogućava napredne fuknconalnosti kao što su :

1. Module Reloading
2. Lazy Loading – učitava pakete kada su potrebni
3. Hashing
4. Code splitting
5. Source maps – koje olakšavaju debug-ovanje minifikovanih verzija



Kao što se može videti sa slike, Webpack generiše statičke aktive koje predstavljaju međusobno zavisne module i pretvaraju ih u nešto što je, može se reći konkretno.

Webpack omogućava mnogo produktivniji i prijatniji rad, jer sa Module Reloading-om ukida potrebu da se nakom promene koda osvežava strana browser-a kako bi se te promene videle, već automatski to čini. Kako se danas zbog performansi uvek vrši minifikacija koda, Source map u velikoj meri pomaže za brzi i jednostavniji debug.

Pored Webpack-a, postoji još alata koje omogućavaju slične stvari kao što su Gulp i Grunt, međutim Webpack je danas u određenoj prednosti jer je mnogo bolji za rad na velikim projektima, pogotovu na danas veoma popularnim SPA (Single page applications) i pored toga rešava neke probleme sam dok je uz ostale potreban dodatni manualni rad.

Webpack možete instalirati koristeći npm : https://webpack.github.io/docs/installation.html

Više o webpack-u :
http://webpack.github.io/docs/
https://medium.com/@dabit3/beginner-s-guide-to-webpack-b1f1a3638460#.61exe0s7b

JSLint

JSLint je Javascript parser i “code quality checker”, to jest alat koji proverava ispravnost ispisanog javascript koda. On nas upozorava ukoliko koristimo javascript funkcije koje su problematične, koristimo promenljive koje se ne podudaraju sa promenljivim koje smo deklarisali i upozorava nas ako smo ostavili nepotrebni zarez, nismo zatvorili zagrade petlji kao što su if, for while itd, označava kod koji se nikada ne izvršava zbog return, throw, continue ili break-a, slučaj u switch-u koji nema break izraz i slične greške. Pored ovih jednostavnih, JSLint rešava i neke druge, ne tako česte greške.

Njegov način delovanja i razloga oko kojih će upozoravati se mogu modifikovati i menjati po potrebi, kako postoji mnogo opcija koje se mogu konfigurisati, treba biti jako pažljiv kako neki zahtevi ne bi bili kontradiktorni.

JSLint naravno ne moze pronaći i rešiti sve greške, ali nam može pomoći da brže i jednostavnije pišemo čistiji kod, pri čemu ćemo se rešiti čestih grešaka pre nego što pokrenemo skriptu ili otvorimo web stranu.

JSLint možete instalirati unutar svog projekta preko npm-a :
https://www.npmjs.com/package/jslint .

U slučaju da želite samo da proverite neke delove koda to se može uraditi na
http://www.jslint.com/ .

JavaScript Kompajleri

Želja web programera da se reše nekih nedostataka JavaScript-a, dovelo je do stvaranja mnogih jezika koji se kasnije kompaliraju u JavaScript. To su jezici koji na primer omogućavaju jednostavnije pravljenje klasa, ili su tipizIraniji jezici. Neki od njih su TypeScript, CoffeeScript ili LiveScript.

I danas sve popularniji ES6 (ECMAScript 2015) standar i sam u sebi ima implementirano mnogo koncepata iz ovih jezika, međutim i ako ES6 postaje sve popularniji, mnogi browser-i još uvek nisu spremni za njega tako da je su potrebni transpajleri koji će taj kod kompajlirati u ES5 koji je čitljiv za većinu browsera, uključujući i stare kao sto je Internet Explorer 9. Takvi transpajleri su Babel ili es6-transpiler.

Instalacija babela je naravno moguća preno npm-a : https://www.npmjs.com/package/babel

Više o ES6(na šta treba obratiti posebnu pažnju) i babel-u : http://es6-features.org/#Constants
https://babeljs.io/

Dodatni alati

JavaScript zajednica je jedna od najboljih zajednica kada je u pitanju količina slobodnog koda, i zista je puna mnogih biblioteka koje će vam u mnogome olakšati rad.

Jedan od zahtevnih stvari u JavaScriptu je rad sa datumima, ali zahvaljujući nekim JavaScript guruima imamo Moment.js koji vam omogućava kreiranje, manipulisanje i formatiranje datuma na koji god način želite.

Ukoliko imate posla sa brojevima koji imaju puno decimala i potrebna vam je velika preciznost, pogledajte Accounting.js, verovatno će vam pomoći.

Kada dođe vreme za povećavanje performansi i minifikacija koda imate UfligyJS.

JS svet je pun predobrih i veoma korisnih alata. Iskoristite ih.

Introduction to JS frameworks

Developing our JS framework