Fakultet elektrotehnike, računarstva
i informacijskih tehnologija Osijek

Web programiranje

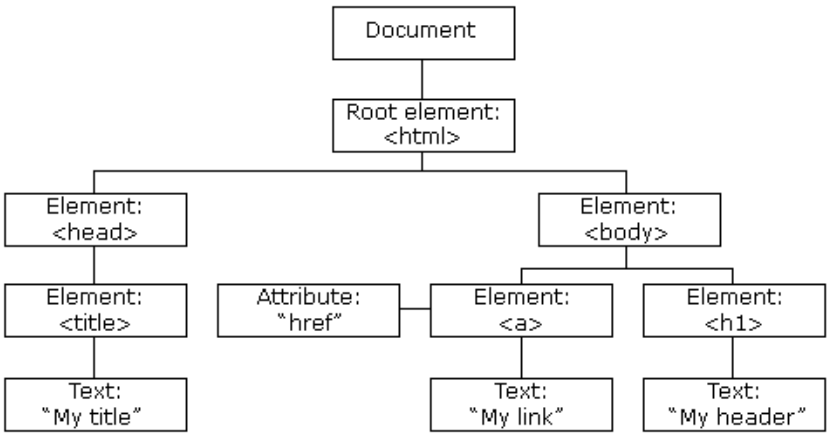
LV3: Web API - DOM i BOM

2019-2020 | ver 1.0 | Autor: Luka Omrčen

Sadržaj

1 Document Object Model (DOM)

Model dokumenta objekta (Document Object Model - DOM) povezuje i modelira web dokumente, skripte i programske jezike u podatkovnu strukturu nalik stablu. Navedeno stablo predstavlja sadržaj dokumenta i odnos između čvorova (objekta) (Slika 1). Strukture i modele unutar stabla opisuju se pomoću Javascripta, iako modeliranje i manipuliranje s HTML, SVG ili XML dokumentima kao objektima ne ulazi u osnovne dijelove JavaScript jezika.



Slika 1: Prikaz DOM strukture stabla

1.1 Tipovi čvorova

Prema slici 1 moguće je vidjeti da postoje više vrsta čvorova. Većina je definirana kao **Element**, dok su neki označeni kao **Attribute** ili **Text**. DOM definira različite vrste čvorova koji implementiraju različita sučelja.

Najčešće korišćeni čvorovi su:

- **Document** - predstavlja korijenski čvor svih XML (HTML) dokumenata.
- **DocumentType** - predstavlja definiciju vrste dokumenta koja se koristi na stranici (oznaka doctype)
- **Element** - predstavlja oznaku (najčešće HTMl element)
- **Attr** - ključ-vrijednost koji predstavlja atribut unutar oznake (najčešće HTML element).
- **Text** - predstavlja sadržaj čvora.
- **Comment** - predstavlja XML / HTML komentare.

1.2 Node objekt

Node objekt predstavlja sučelje koje različite vrste DOM objekata nasljeđuju, omogućava da se određeni objekti tretiraju na sličan način i da posjeduju zajedničke karakteristike. Sučelja koja nasljeđuju

metode i svojstva Node objekta su: Document, Element, Attr, Text, Comment itd. Node objekt implementira i izlaže svoja svojstva i metode koje se onda mogu nasljeđivati. Popis svih metoda i svojstava moguće je pronaći na poveznici [9]. Slika 2 prikazuje strukturu Node objekta s definirajućim EventTarget DOM sučeljem koje implementiraju objekti za primanje i osluškivanje određenih događaja i njegovih promjena.

Slika 2: Struktura Node objekta

Neke od navedenih svojstava i metoda Node objekta su:

- nodeName
- nodeValue
- nodeType
- ownerDocument
- firstChild
- lastChild
- childNodes
- previousSibling
- nextSibling
- hasChildNodes()
- attributes
- parentNode
- parentElement
- appendChild(node)
- removeChild(node)
- replaceChild(newNode, previousNode)
- insertBefore(newNode, previousNode)

1.3 Pristupanje čvorovima

Najlakši način za dohvaćanje čvorova je njihovo izravno pozivanje pomoću jedne od sljedeće navedenih metoda unutar koda 1

```
// Vraca referencu na element prema ID-u.
document.getElementById("someId");
// Vraca objekt slican nizu svih podredenih elemenata koji imaju sva navedena imena klase.
document.getElementsByClassName("someClass");
// Vraca HTMLCollection elemenata s danom nazivom oznake.
document.getElementsByTagName("li");

// Novija Sintaksa _________________________

// Vraca prvi element u dokumentu koji odgovara navedenoj grupi selektora.
document.querySelector(".someClass");
document.querySelector ("#someId");
//Vraca popis elemenata u dokumentu (koristeci dubinu prvog unaprijed definiranog cvora
    dokumenta) koji odgovaraju navedenoj grupi selektora.
document.querySelectorAll ("div.note, div.alert");
```

Kod 1: Primjer dohvaćanja čvorova navedenim metodoma

Metoda **.querySelectorAll** vraća NodeList tip podatka - koja ne predstavlja polje. To znači da uobičajene metode za rad s poljima nisu dostupne izravno. Kako bi manipulirali s vrijednostima iz metode potrebno je prvo pretvoriti NodeList niz pomoću metode Array.from u polje (kod 2).

```
let myElements = document.querySelectorAll(".list li"); // NodeList
Array.from(myElements).forEach(...)
```

Kod 2: Pretvorba NodeList u polje

Također možemo koristiti većinu ovih metoda (osim getElementById i getElementByName) na bilo kojem čvoru tipa Element koji djeluje kao korijenski element za naš upit (kod 3).

```
// dohvacamo element s id wrapper
const wrapper = document.querySelector("#wrapper");
//unutar elementa s id wrapper dohvacamo element koji je paragraf
wrapper.getElementsByTagName("p");
//unutar elementa s id wrapper dohvacamo element koji ima klasu active
wrapper.getElementsByClassName("active");

// Drugi nacin _________
const wrapper = document.querySelector("#wrapper");
const p = wrapper.querySelector("p");
const activeElement = wrapper.querySelector(".active");
```

Kod 3: Dohvaćanje čvorova na elementu koji nije Document objekt

Osim direktnog pristupanja elementu pomoću querySelector-a moguće je dohvatiti element i njegovu

djecu unutar DOM-stabla sljedećim metodama - navedene metode naslijeđene su iz Node objekta.

```
let element = document.getElementById("someId");
// Dohvati svu djecu iz elementa s id-om "someId"
let children = element.childNodes;
// Dohvati roditeljski cvor
let parent = children.parentNode;
// Metode za manipulaciju cvorovima unutar roditeljskog cvora
myElement.children;
myElement.firstElementChild;
myElement.lastElementChild;
myElement.previousElementSibling;
myElement.nextElementSibling;
myElement.firstChild;
myElement.lastChild;
myElement.previousSibling;
myElement.nextSibling;
myElement.parentNode;
myElement.parentElement;
```

Kod 4: Dodatne metoda za dohvaćanje elemenata unutar roditeljskog čvora

1.4 Kreiranje, izmjena i brisanje čvorova

Postoje razne nativne DOM metode koje možemo koristiti za stvaranje čvorova različitih vrsta. Najčešće metode prilikom stvaranja novih čvorova su:

- document.**createElement**("element name") - za kreiranje novog elementa
- document.**createAttribute**("attribute name") - za kreiranje novog atributa
- document.**createTextNode**("text name") - za kreiranje nove tekstualne vrijednosti unutar dokumenta
- document.**createComment**("comment name") - za kreiranje komentara

U primjeru koda 2 prikazane su metode za kreiranje čvorova unutar DOM-a.

```
// kreiranje elementa
let newHeading = document.createElement("h1");
let newParagraph = document.createElement("p");

// kreiranje tekstualnog cvora sa vrijednosti
let h1Text = document.createTextNode("Ovo je naslov");
let pText = document.createTextNode("Ovo je paragraf");

// umetanje tekstualnog cvora unutar novog kreiranog elementa
newHeading.appendChild(h1Text);
newParagraph.appendChild(pText);
```

```
// dohvacamo element pomocu query selectora
let firstHeading = document.querySelector("#firstHeading");

// unutar firstHeading-a elementa ubacujemo newHeading i newParagraph elemente
firstHeading.appendChild(newHeading);
firstHeading.appendChild(newParagraph);

// dohvacamo roditeljski element firstHeading-a
let parent = firstHeading.parentNode;

// postavljamo newHeading element prije firstHeading elementa - osim meteode insertBefore
    moguce je koristiti metode .append, .appendChild, .insertAdjancentElement
parent.insertBefore(newHeading, firstHeading);
```

Kod 5: Kreiranje novog elementa i tekstualnog čvora unutar DOM-a

Svaki element također ima nasljeđene metode kao što **.innerHTML** i **.textContent** (kao i .innerText, koji je sličan **.textContent**). Navedene metode upravljaju HTML i tekstualni sadržajem unutar elemenata. Više o navedenim metodama na poveznici [7, 10].

Za uklanjanje čvorova potrebno je koristiti metodu **removeChild**. Prije nego što se koristi metoda removeChild potrebno je selektirati roditelja kojega se želi brisati. Postoji i metoda **remove()** koja se također može upotrebljavati na elementima, iako nije podržana u Internet explorer preglednicima i potrebno je definirati Polyfill-e (Kod 6).

```
const element = document.querySelector('#title');
// dohvacamo roditelja navedenog elementa i s njegove pozicije brisemo navedeni element
element.parentNode.removeChild(element);
// brisemo direktno element iz DOM-a - ne radi u IE
element.remove();
```

Kod 6: Brisanje čvora iz DOM-a

Za izmjenu čvorova potrebno je koristiti metodu **replaceChild()**. Postupak izmjene je isti kao i kod brisanja čvora - potrebno je selektirati roditeljski čvor, a zatim pozvati metodu koja za prvi parametar ima novi element, dok za drugi parametar predaje se element koji će zamjeniti stari element (Kod 7).

```
const oldElement = document.querySelector('#title');
const newElement = document.createTextNode('ovo je novi naslov');
const replacedNode = oldElement.parentNode.replaceChild(newElement, oldElement);
```

Kod 7: Zamjena čvorova unutar DOM-a

1.5 Atributi

DOM čvorovi tipa Element izlaže svoje svojstvo atributa kao zbirku svih atributa za određeni element. Postoje dvije metode dohvaćanja atributa unutar js-a:

```
const element = document.querySelector('#title');
// tocka "dot" notacija - dohvacamo atribut class
const className = element.attributes.class;
// standardna notacija preko polja - dohvacamo atribut pod indexom 0
const firstAttribute = element.attributes[0];
```

Kod 8: Metode pristupanja atributima unutar DOM-a

Atribute je moguće dohvatiti, postaviti i ukloniti pomoću navedenih metoda:

- element.getAttribute('class') - dohvaća vrijednosti iz atributa class
- element.setAttribute('class', 'new-classname'); - postavlja novu vrijednost unutar atributa class
- element.setAttributeNode(attributeNode); - postavlja novi Arr čvor na navedeni element - više na poveznici [1].
- element.removeAttribute('class'); - briše navedeni atribut iz elementa

Koristeći naziv atributa DOM HTML specifikacija omogućuje pristup i izmjenu svih atributa izravno dodjeljivanjem novih vrijednosti (Kod 9).

```
<img src="slika.jpg" alt="Logo" class="logo logo-sm" />
//-------------------------------------------------------
const image = document.querySelector(".logo");
// vraca vrijednost "image.jpg"
image.src;
// azurira vrijednost atributa alt
image.alt = "Ovo je novi alt naslov";
// Osim direktnog pristupanja atributima pomocu dot notacije moguce je pristupiti i preko polja
    s koristenjem metode value
image.attributes[0].value
// moguce je postavljanje i izmjenjivanje vise atributa istovremeno koristeci metodu
    Object.Assign
Object.assign(image, {
  className: "nova-klasa",
  id: "novi id"
});
```

Kod 9: Izravno pristupanje i manipuliranje atributima unutar DOM-a

Mogućnosti navedene u gornjim primjerima vrijede za sve atribute, osim klase. Klasa (**class**) predstavlja rezerviranu ključnu riječ koja se stavlja u upotrebu od korištenje ES6. Kako bi koristili atribut za klasu potrebno je koristiti ključnu riječ className. Isto vrijedi i za atribut **for** unutar label elementa, potrebno je koristiti alternativni naziv atributa htmlFor (Kod 10).

```
<img src="slika.jpg" alt="Logo" class="logo logo-sm" />
//-------------------------------------------------------
image.className; // "logo logo-sm"
image.class; // undefined
});
```

Kod 10: Korištenje className ključne riječi

Kako bi olakšali developerima manipulaciju klasa na frontendu, postoji svojstvo classList - zbirka tipa DOMTokenList koja vraća sve vrijednosti klasa koje sadržava pojedini element. Sučelje implementira metode za manipuliranje klasama (Kod 11).

```
let firstHeading = document.querySelector ("#main-heading");

// ukloni klasu foo iz elementa
firstHeading.classList.remove ( "foo");
// dodaj klasu foo unutar elementa
firstHeading.classList.add ("new-class-foo");
// dodavanje ili uklanjanje vise klasa
firstHeading.classList.add ("foo", "bar");
firstHeading.classList.remove ("foo", "bar");
// ukoliko postoji navedena klasa obrisi ju, u suprotnom slucaju dodaj ju
firstHeading.classList.toggle ( "active");
// vraca true ili false ovisno dali je klasa vidljiva ili ne
firstHeading.classList.contains ("foo");
```

Kod 11: Korištenje classList svojstva i njegove metode

Osim svojstva za manipuliranje klasama, Web API definirao je metode za dinamičku promjenu vizualne prezentacije HTML dokumenata pomoću Javascripta za izravno stiliziranje (**Inline CSS**). Inline stilovi primjenjuju se izravno na određeni HTML element koristeći atribut **style**. U JavaScript-u svojstvo **style** koristi se za dohvaćanje ili postavljanje inline stila unutar elementa (Kod 12).

```
let ele = document.querySelector(".box");
// Postavljanje stilova unutar jedne naredbe
ele.style.cssText = "color: blue; border: 1px solid black";
// Postavljanje stilova pomocu setAttribute metode
ele.setAttribute("style", "color:red; border: 1px solid blue;");
// Postavite pojedinacnog stila, a ostale vrijednosti stilova ostaju ne promijenjene
ele.style.color = "blue";
```

Kod 12: Primjer implementacije stilova pomoću Javascripta

Mnoga CSS svojstva, poput veličine fonta, pozadinske slike, ukrašavanja teksta - sadrže crtice (-) u svojim imenima. Budući da je u JavaScript crtica rezervirani operator i ona se tumači pomoću znaka minus, potrebno je koristiti **camelCase** sintaksu prilikom definiranja svojstva stila (Kod 13).

```
let ele = document.querySelector(".box");
ele.style.color = "blue";
ele.style.fontSize = "18px";
ele.style.fontWeight = "bold";
});
```

Kod 13: Korištenje camelCase sintakse prilikom definiranja stilova

Svojstvo stila nije korisno u situaciji kada je potrebno dohvatiti sve podatke o stilu elementa - jer vraća samo ona pravila koja su postavljena unutar atributa stila, dok ona vanjska ne dohvaća (kao što su vanjski izvori stilova -eksterni CSS). Kako bi dohvatili vrijednost svih CSS svojstva navedenog elementa potrebno je koristiti metodu .getComputedStyle() u kombinaciji s window objektom: **window.getComputedStyle(trazeniElement)**, kako je prikazano u primjeru koda 14.

```
let elem = document.getElementById("intro");
let styles = window.getComputedStyle(elem);
styles.getPropertyValue("color");
styles.getPropertyValue("font-size");
// Ili na drugi nacin
let width = styles.width;
```

Kod 14: Primjer definiranja getComputedStyle metode

Više o navedenim metodama moguće je pronaći na poveznici [2, 5].

HTMl5 ima osmišljeni koncept proširivanja podataka koji su povezani s određenim elementom, ali ne moraju imati određeno ili smisleno značenje. Atribut **data-*** omogućuje korisniku pohranu i manipulaciju dodatnih informacija unutar standardnih, semantičkih HTML elemenata (Kod 15).

```
<article
id="motori"
data-columns="3"
data-index-number="12314"
data-parent="vozila"
data-object='{"id":42,"name":"Yamaha"}'>
...
</article>
```

Kod 15: Definiranje korisnikovih data atributa

Za čitanje vrijednosti proširenih atributa unutar Javascript-a može se koristiti metoda getAttribute(), ali standard definira jednostavniji način pristupanja - unutar tipa DOMStringMap koristeći svojstvo **dataset**. Kako bi dohvatili data- atribut, potrebno je pozvati svojstvo **dataset** i pomoću operatora pridruživanja pridružiti traženi atribut (Kod 16). Svako pridruženo svojstvo predstavlja niz koje se može čitati i pisati.

```
const motors = document.querySelector('#motori');
motors.dataset.columns // "3"
motors.dataset.indexNumber // "12314"
motors.dataset.parent // "vozila"
// Refaktoriranje data atributa
motors.dataset.columns = 5;
```

Kod 16: Pristupanje data atributima

1.6 Događaji

DOM događaji predstavljaju objekt koji se temelji na sučelju događaja (Event) i mogu posjedovati određena svojstva ili metode koje se koriste za definiranje akcija o onome što se dogodilo. Svaki HTML element ima vlastiti popis podržanih vrsta događaja. Osluškivanjem događaja na elementu, čeka se ispunjavanje uvjeta određenog događaja koji vodi do određenih akcija pomoću rukovoditelja događaja (event handler). Postoje više scenarija upravljanja događajima:

- Jedan element može slušati i odgovarati na više vrsta događaja.
- Jednom vrstom događaja može se upravljati s više elemenata.

Određeni događaj moguće je vezati na element pomoću metode **addEventListener()**. Prvi argument metode predstavlja naziv metode, dok drugi parametar definira rukovodioca događaja (najčešće funkcija) koji izvršava određeni set akcija nakon što se događaj dogodi (Kod 17).

```
const button = document.querySelector('#button');
button.addEventListener('click', handleClick);
function handleClick(event) {
  console.log('clicked');
}
//---------Drugi nacin preko anonimne metode
button.addEventListener('click',(event) => {
  console.log('clicked');
});
```

Kod 17: Definiranje događaja

Kako bi prestali slušati određeni događaj na elementu, potrebno je koristiti metodu **removeEventListener** (Kod 18). Kako bi uklonili događaj, potrebno je imati referencu na funkciju koja se izvršava prilikom dohvaćanja događaja - to znači da nije moguće ukloniti određeni događaj ako se anonimne funkcija predaje kao callback metodi addEventListener().

```
button.removeEventListener('click', handleClick);
```

Kod 18: Definiranje događaja

Popisa najčešće korištenih vrsta događaja:

1. Događaji s mišem: click, dblclick, mousedown, mouseout, mouseover, mouseup, mousemove.
2. Događaji na tipkovnici: keydown, keypress, keyup.
3. HTML događaji: load, unload, abort, error, resize, change, submit, reset, scroll, focus, blur.
4. DOM događaji: DOMSubtreeModified, DOMNodeInserted, DOMNodeRemoved.

Referencu na sve događaje moguće je pronaći na poveznici [4].

Unutar funkcije slušatelja, **event.target** predstavlja referencu elementa na kojem je događaj pokrenut. Navedeni slučaj je koristan ako postoje više elemenata istog tipa i gdje je potrebno za svaki element odraditi određenu akciju (Kod 19).

```
// referenciramo sve forme koje postoje unutar DOM-a
const myForm = document.forms[0];
// selektiramo svaki input unutar formi koje postoje
const myInputElements = myForm.querySelectorAll("input");
// iteriramo kroz sve inpute i dodjeljujemo im dogadaj i rukovodioca
Array.from(myInputElements).forEach(el => {
  el.addEventListener("change", function(event) {
    // koristeci event.target ispisujemo vrijednost pojedinog elementa pomocu value
    console.log(event.target.value);
  });
});
```

Kod 19: Definiranje događaja za više specifičnih elemenata

Postoje situacije kada je potrebno spriječiti nativno ponašanje elemenata unutar web preglednika - u tu svrhu potrebno je koristiti metodu .preventDefault(). Navedeni primjer unutar koda 20 prikazuje sprečavanje nativnog ponašanja poveznice eksterni link. Logika koja je definirana poslije .preventDefault() metode predstavlja novo ponašanje navedenog elementa unutar događaja. Navedena metoda najčešće se koristi prilikom provjere valjanosti obrasca na strani klijenta.

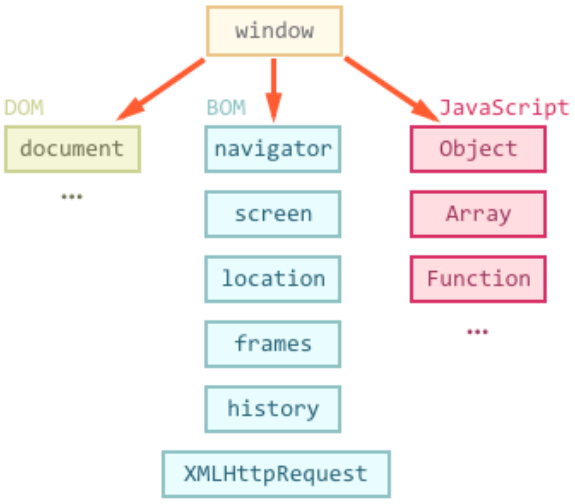
```
const anchor = document.querySelector("a");
anchor.addEventListener("click", (event) => {
  event.preventDefault
  console.log("prevented default behaviour",2+2);
})
// Primjer 2 - forme ------------------------------
myForm.addEventListener("submit", function(event) {
  const name = this.querySelector("#name");
  if (name.value.match(/\d+/g) !== null) {
    alert("Cant use numbers inside name input");
    event.preventDefault();
  }
});
```

Kod 20: Korištenje metode .preventDefault()

Druga važna metoda unutar događaja je **.stopPropagation()**, koja sprječava pojavu "event-bubbling" unutar DOM-a. Termin "event-bubbling" predstavlja način širenja događaja unutar DOM stabla. Kada se događaj dogodi u elementu koji se nalazi unutar drugog elementa, oba elementa su registrirala rukovodioca za taj događaj. Drugim riječima, kada se događaj dogodi na nekom elementu, on prvo pokreće rukovodioca na njemu, zatim nad roditeljem, a zatim i na ostale pretke sve do korijena DOM stabla. Više o "event-bubbling" na poveznici [3].

2 Browser Object Model (BOM)

Model objekta pretraživača (BOM) omogućuje Javascriptu da komunicira s preglednikom o stvarima koje ne predstavljaju sadržaj stranice. Službeni standard i specifikacije za BOM ne postoje, iako su popularni pretraživači primjenili gotovo identične značajke kako bi osigurali kompatibilnost. Slika 3 prikazuje BOM strukturu stabla.



Slika 3: Prikaz BOM strukture

2.1 Globalne varijable i metode

Window objekt predstavlja korjenski element preko kojega su vezani svi ostali elementi - izravno ili neizravno. Svaka kartica unutar preglednika (tab) posjeduje vlastiti window objekt, odnosno ne window objekt je unikatan za svaku karticu unutar prozora.

Važno je napomenuti da sve varijable i funkcije koje se ne vežu na određeni objekt, vežu se automatski na window objekt i tako postaju dio globalne okoline - globalne varijable postaju svojstva window objekta, dok globalne funkcije postaju metode.

Globalni objekt window pruža pristup:

- svojstva koja pružaju informacije na prozoru preglednika
- metode za podešavanje timera i intervala

Kod 21 prikazuje primjere iz navedene liste. Potpuni popis svih svojstava i metoda objekta window nalazi se na poveznici [11].

```
// vanjske dimenzije preglednika
const outerHeight = window.outerHeight;
const outerWidth = window.outerWidth;
// unutarnje dimenzije
const innerHeight = window.innerHeight;
const innerWidth = window.innerWidth;

// definiranje intervala i timera
const timeout = setTimeout(callbackFunkcija, vrijeme); // vrijeme je u ms
const interval = setInterval(callbackFunkcija, vrijeme );

// ponistavanje intervala i timera sljedecim metodama
clearTimeout(timeout);
clearInterval(interval);
```

Kod 21: Metode i svojstva window objekta

2.2 Location objekt

Zbog nedostatka specifikacije, **location** objekt unutar preglednika veže se generalno za document i window objekt (Kod 22).

```
document.location === window.location; // true
```

Kod 22: Location objekt

Location objekt omogućava čitanje i manipuliranje URL-ovima adresne trake web preglednika. Najčešća svojstava i metode koje se koriste su:

- location.hash
- location.host
- location.hostname
- location.**href** - vraća puni URL trenutne stranice. Također, moguće je definirati novu vrijednost na navedeno svojstvo, čime će se preusmjeriti stranica na novi URL.
- location.**pathname** - vraća ono što dolazi nakon imena hosta.
- location.**port** - vraća broj porta, ali samo ako je postavljen na URL.
- location.**protocol** - vraća protokol koji se koristi za pristup stranici.
- location.**search** - vraća ono što dođe nakon znaka ? (znak ? odvaja Odvaja URL niz od niza upita) u URL-u.
- location.**assign(url)** - navigira do URL-a koji je poslan kao parametar.

- location.**replace(url)** - slično kao metoda assign, ali zamijenjeno web mjesto uklanja iz povijesti sesija (session).
- location.**reload()** - ponovno pokreće stranicu (osvježavanje)

Više informacija o location objektu nalazi se na poveznici [8].

2.3 History objekt

History objekt (**window.history**) služi za manipulaciju povijesti sesije preglednika. History objekt omogućava pregled posjećenim stranicama koje su bile aktivne na trenutnoj kartici ili okviru.

Najčešće korištene metode history objekta su:

- history.**length** - vraća cijeli broj koji predstavlja broj elemenata u povijesti sesija, uključujući trenutno učitanu stranicu.
- history.**go(number)** - preusmjerava na stranicu koja se nalazi u povijesti sesije identificiranu prema relativnoj lokaciji.
- history.**back (number)** - preusmjerava na prethodnu stranicu koja se nalazi u povijesti sesije
- history.**forward (number)** - preusmjerava na sljedeću stranicu koja se nalazi u povijesti sesije

Više informacija o history objektu nalazi se na poveznici [6].

2.4 Navigator objekt

Objekt navigator pruža informacije o stanju i identitetu korisničkog agenta (preglednika i OS-a koji korisnik pokreće). Navedeni objekt koristan je u situacijama kada je potrebno prilagoditi stranicu određenom pregledniku ili operativnom sustavu.

Najčešće korištene metode navigator objekta su:

- navigator.**userAgent** - vraća verziju korisnikovog agenta unutar preglednika.
- navigator.**language** - vraća polje koji predstavlja željeni jezik korisnika, obično predstavlja jezik korisničkog sučelja preglednika.
- navigator.**languages** - vraća niz jezika poznatih korisniku, razvrstanih po želji: ["en-HR", "en-US", "en"]
- navigator.**getBattery()** vraća promise koje se nakon rješavanja (resolve) vraća BatteryManager objekt koji pruža informacije o bateriji sustava i događaje pomoću kojega je moguće nadgledali stanje baterije.
- navigator.**plugins** - vraća popis dodataka instaliranih unutar preglednika.
- navigator.**onLine** - vraća logički signal koji prikazuje radi li korisnikova mreža unutar preglednika ili ne.
- navigator.**geolocation** - vraća objekt Geolocation koji omogućava pristup lokaciji uređaja.

Literatura

[1] *MDN - Attribute Node.* URL: https://developer.mozilla.org/en-US/docs/Web/API/Element/setAttributeNode.

[2] *MDN - ElementInlineStyle.* URL: https://developer.mozilla.org/en-US/docs/Web/API/ElementCSSInlineStyle/style.

[3] *MDN - Event Bubbling.* URL: https://javascript.info/bubbling-and-capturing.

[4] *MDN - Events.* URL: https://developer.mozilla.org/en-US/docs/Web/Events.

[5] *MDN - getComputedStyle metoda.* URL: https://developer.mozilla.org/en-US/docs/Web/API/Window/getComputedStyle.

[6] *MDN - History object.* URL: https://developer.mozilla.org/en-US/docs/Web/API/History.

[7] *MDN - InnerHTML.* URL: https://developer.mozilla.org/en-US/docs/Web/API/Element/innerHTML.

[8] *MDN - Location Object.* URL: https://developer.mozilla.org/en-US/docs/Web/API/Location.

[9] *MDN - node.* URL: https://developer.mozilla.org/en-US/docs/Web/API/Node.

[10] *MDN - textContent.* URL: https://developer.mozilla.org/en-US/docs/Web/API/Node/textContent.

[11] *MDN - Window Object.* URL: https://developer.mozilla.org/en-US/docs/Web/API/Window.