

Ajzenhamer Nikola  
Bukurov Anja

# UVOD U VEB I INTERNET TEHNOLOGIJE

24. januar 2019.

# Sadržaj

<b>Predgovor</b>	<b>5</b>
<b>1 Čas 3</b>	<b>7</b>
1.1 Uvod u HTML . . . . .	7
1.1.1 Elementi i atributi, komentari . . . . .	7
1.1.2 Struktura HTML dokumenta . . . . .	7
1.1.3 HTML zaglavje . . . . .	9
1.1.4 HTML telo . . . . .	9
1.1.5 HTML5 semantički elementi ( <code>header</code> , <code>nav</code> , ...)	9
1.1.6 Elementi za naslove ( <code>h1</code> , ..., <code>h6</code> ) . . . . .	10
1.1.7 Elementi <code>p</code> i <code>span</code> . . . . .	10
1.1.8 Blokovski i linijski elementi . . . . .	11
1.1.9 Grupisanje elemenata ( <code>div</code> ) i novi red ( <code>br</code> ) . . . . .	11
1.2 Uvod u CSS . . . . .	13
1.2.1 CSS sintaksa . . . . .	14
1.2.2 Dodeljivanje boje elementu . . . . .	15
1.2.3 Formatiranje teksta . . . . .	16
1.2.4 Dodavanje korisničkih fontova . . . . .	17
1.3 Model kutije . . . . .	20
1.3.1 Sadržaj elementa . . . . .	20
1.3.2 Punjenje elementa . . . . .	22
1.3.3 Ivica elementa . . . . .	23
1.3.4 Pojas elementa . . . . .	26
<b>2 Čas 4</b>	<b>34</b>
2.1 Pozicioniranje elemenata . . . . .	34
2.1.1 Statičko pozicioniranje . . . . .	34
2.1.2 Relativno pozicioniranje . . . . .	35
2.1.3 Apsolutno pozicioniranje . . . . .	36
2.1.4 Fiksno pozicioniranje . . . . .	37
2.1.5 Z-pozicioniranje elemenata . . . . .	38
2.1.6 Svojstvo <code>display</code> . . . . .	39
2.1.7 Svojstvo <code>float</code> . . . . .	43
2.1.8 Svojstvo <code>clear</code> . . . . .	43
2.1.9 Svojstvo <code>overflow</code> . . . . .	47
<b>3 Čas 5</b>	<b>49</b>
3.1 Upravljanje pozadinom elemenata . . . . .	49
3.1.1 Slika kao pozadina . . . . .	49
3.1.2 Gradijentna pozadina . . . . .	54
3.2 Rad sa slikama . . . . .	58
<b>4 Čas 6</b>	<b>62</b>
4.1 Rad sa tabelama . . . . .	62
4.1.1 Razna podešavanja prikaza tabela . . . . .	64
4.1.2 Spajanje kolona i redova . . . . .	66
4.1.3 Stilizovanje redova tabela . . . . .	67
4.2 Liste . . . . .	69
4.2.1 Nenumerisane liste . . . . .	69
4.2.2 Numerisane liste . . . . .	70

4.2.3 Stilizovanje liste . . . . .	73
4.2.4 Ugnježdene liste . . . . .	74
4.2.5 Definicione liste . . . . .	75
4.3 Veze . . . . .	76
4.3.1 Pseudoklase . . . . .	76
4.4 Pseudoklase i pseudoelementi . . . . .	80
<b>5 Čas 7</b>	<b>83</b>
5.1 Pogled . . . . .	83
5.2 Prikazivanje slike . . . . .	83
5.3 Pozicioniranje . . . . .	85
5.4 Bootstrap . . . . .	89
<b>6 Čas 8</b>	<b>94</b>
6.1 Osnovne jezičke strukture . . . . .	94
6.1.1 Programske strukture . . . . .	97
6.2 Izvršavanje JavaScript koda u veb pregledaču . . . . .	100
6.3 Rad sa DOM stablom . . . . .	101
6.4 Rad sa konzolom veb pregledača . . . . .	108
6.5 Pridruživanje osluškivača događaja elementima . . . . .	108
<b>7 Čas 9</b>	<b>111</b>
7.1 HTML formulari . . . . .	111
7.2 Obrada podataka u formularu . . . . .	118
<b>8 Čas 10</b>	<b>127</b>
8.1 Asinhrono programiranje . . . . .	127
8.1.1 Po-potrebi-pozivne funkcije . . . . .	127
8.1.2 XMLHttpRequest objekat . . . . .	128
8.2 Biblioteka jQuery . . . . .	130
8.2.1 Dodavanje jQuery-ja . . . . .	130
8.2.2 Osnovna jQuery sintaksa . . . . .	131
8.2.3 "Dokument je spreman" događaj . . . . .	131
8.2.4 jQuery selektori . . . . .	132
8.2.5 jQuery metodi zasnovani na događajima . . . . .	133
8.2.6 jQuery efekti i animacije . . . . .	136
8.2.7 Korišćenje funkcionalnosti reda . . . . .	137
8.2.8 Po-potrebi-pozivne funkcije i animacije . . . . .	138
8.2.9 Lančanje metoda . . . . .	138
8.2.10 Upravljanje DOM stablom . . . . .	139
8.2.11 Upravljanje stilovima . . . . .	141
8.2.12 Upravljanje modelom kutije . . . . .	143
<b>9 Čas 11</b>	<b>161</b>
9.1 Moduli . . . . .	161
9.1.1 Paketi . . . . .	161
9.1.2 Node upravljač paketima . . . . .	161
9.1.3 CommonJS . . . . .	161
9.2 Kreiranje serverskih aplikacija . . . . .	162
<b>10 Čas 12</b>	<b>180</b>
10.1 Razvojno okruženje Express.js . . . . .	180

<b>11 Čas 13</b>	<b>196</b>
11.1 MongoDB . . . . .	196
11.1.1 Alat MongoDB Compass . . . . .	196
11.1.2 Upiti . . . . .	200
11.2 Razvojno okruženje Mongoose.js . . . . .	202
<b>Dodatak</b>	<b>231</b>
Lista kodova . . . . .	232



## Predgovor

Ovaj tekst predstavlja skriptu iz kursa "Uvod u Veb i Internet Tehnologije", na 2. godini smera Informatika na Matematičkom fakultetu Univerziteta u Beogradu. Skripta je koncipirana na osnovu materijala koleginice [Andelke Zečević](#) sa časova vežbi. Skripta je prateći materijal pre svega studentima koji ovaj kurs slušaju u okviru svojih studija, ali i svima Vama koji biste želeli da se upoznate sa ovom tematikom. Ovaj materijal ne može zameniti pohađanje vežbi niti drugu preporučenu literaturu.

Ovaj tekst je u ranoj fazi formiranja. Ukoliko ste pažljivi čitalac ove skripte, i ukoliko uočite bilo kakvu grešku ili propust, možete se javiti autorima na adresu [nikola\\_ajzenhamer@math.rs](mailto:nikola_ajzenhamer@math.rs). Svi komentari, sugestije, kritike, ali i pohvale vezane za ovaj materijal su dobrodošli.

Autori



# 1 Čas 3

Cilj ovog časa je upoznavanje studenata sa osnovnim elementima jezika za obeležavanje teksta HTML i CSS, kao i sa modelom kutije u jeziku CSS.

## 1.1 Uvod u HTML

HTML (*HyperText Markup Language*) predstavlja najosnovniji blok za izgradnju Veba. Njime se opisuje logički sadržaj stranice<sup>1</sup>. Na primer, u HTML-u zadajemo koji deo stranice čini naslov ili paragraf, ili gde se nalazi slika.

### 1.1.1 Elementi i atributi, komentari

HTML stranica se izgrađuje od velikog broja HTML *elementa*. Neki od njih definišu meta-informacije (odnosno, informacije o samoj stranici), a drugi definišu konkretni sadržaj. Struktura jednog elementa se sastoji od dve *etikete*, otvarajuće i zatvarajuće, između kojih se nalazi nekakav sadržaj (tekst ili ugnezđeni elementi):

```
<p>      <!-- Otvarajuća etiketa za element "p" -->
...        <!-- Sadržaj elementa -->
</p>      <!-- Zatvarajuća etiketa za element "p" -->
```

Takođe, postoje elementi koji imaju samozatvarajuću etiketu. Njihova struktura je oblika:

```
<img>      <!-- Samozatvarajuća etiketa za element "img" -->
```

U nastavku teksta ćemo, kada uvodimo novi element, podrazumevati da je reč o elementu koji ima otvarajuću i zatvarajuću etiketu. Posebno ćemo naglasiti ukoliko je reč o samozatvarajućem elementu.

U prethodnim primerima smo videli i primer HTML komentara. Svaki HTML komentar počinje oznakom `<!--` i završava se oznakom `-->`. Sve što je unutar ove dve oznake neće biti prikazano na stranici, slično kao što komentari u programskom jeziku C ne utiču na kompiliranje i izvršavanje koda.

Elemente možemo dodatno okarakterisati *atributima* oblika `atribut="vrednost"`. Atributi se navode u okviru otvorene etikete. Na primer,

```
<body id="telo">
</body>
```

Postoje generički atributi koji se mogu primeniti na svim elementima, a postoje i specifični atributi za pojedine elemente. Na primer, svi elementi mogu imati atribut `id`, dok samo element `form` može imati atribut `onsubmit`.

### 1.1.2 Struktura HTML dokumenta

Osnovna struktura HTML dokumenta podrazumeva narednu strukturu:

Kod 1: `primeri/uvod_u_html/osnovna_struktura.html`

```
<!DOCTYPE html>
<html>
  <head>
```

<sup>1</sup>HTML stranice se čuvaju kao HTML dokumenti, tj. sa ekstenzijom `html`.

```
<title>Naslov</title>
<meta charset="UTF-8">
</head>
<body>
</body>
</html>
```

Deklaracija dokumenta `<!DOCTYPE html>` predstavlja posebnu vrstu elementa kojom se definiše verzija HTML dokumenta koja se koristi. Konkretno, ovom deklaracijom navodimo da ćemo koristiti HTML5 verziju. Elementom `html` definišemo sadržaj html dokumenta. HTML dokumenti imaju dva glavna dela:

- *zaglavje* — sadrži informacije o samom dokumentu, kao i linkove ka definicijama stilova (CSS) i programskih elemenata (JavaScript). Ove informacije služe da opišu dokument i one nisu vidljive na stranici. Primeri informacija su: naslov, opis, jezik, ključne reči, itd. Zaglavje je definisano elementom `head`.
- *telo* — sadrži vizuelne elemente koji će biti prikazani korisniku. Sve što se nalazi u telu dokumenta je vidljivo u okviru veb pregledača. Takođe, sve što želimo da bude vidljivo, moramo da stavimo u telo dokumenta. Telo je definisano elementom `body`.

Element `title` služi za davanje semantičkog naslova dokumenta. Ovaj naslov će se prikazivati u većini veb pregledača kao ime kartice u kojoj je dokument otvoren, ali i u rezultatima pretrage. Element `meta` nudi različite mogućnosti za dodavanje metainformacija dokumentu. Na primer, atributom `charset="UTF-8"` postavlja se UTF-8 kao kodna shema za dokument. U nastavku ćemo videti još neke primere metainformacija.

U prethodnom primeru takođe vidimo obaveznu strukturu svakog HTML dokumenta, definisanu HTML specifikacijom. Možemo testirati da li naš html dokument predstavlja validno napisan dokument pomoću HTML validatora. Jedan primer validatora je moguće pronaći na adresi <https://html5.validator.nu/>. Na primer, da smo izostavili element `title`, validator bi se pobunio i prikazao nam odgovarajuću poruku:

(X)HTML5 validation results

Validator Input

Text Field > <!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
</head>
<body>
</body>
</html>

Show Image Report
 Show Source

Group Messages

1 Error: Element `head` is missing a required instance of child element `title`.
From line 6, column 2 to line 6, column 8
<head></head>

Content model for element `head`:  
If the document is an [iframe srcdoc document](#) or if title information is available from a higher-level protocol: Zero or more elements of `metadata content`, of which no more than one is a `title` element and no more than one is a `base` element.  
Otherwise: One or more elements of `metadata content`, of which exactly one is a `title` element and no more than one is a `base` element.

### 1.1.3 HTML zaglavljje

Kao što smo rekli, zaglavljje HTML dokumenta predstavljeno je elementom `head`. Takođe, upoznali smo se sa dva važna elementa zaglavljja: `title`, koji definiše naslov dokumenta i `meta` koji zadaje metainformacije o dokumentu. Najveći broj metainformacija se zadaje u obliku para atributa `name="..."` i `content="..."`. U sledećoj tabeli dat je spisak nekih metainformacija i primer njihovog korišćenja:

Naziv meta-informacije	Primer upotrebe
language	<code>&lt;meta name="language" content="sr"&gt;</code>
author	<code>&lt;meta name="author" content="Ajzenhamer N, Bukurov A."&gt;</code>
keywords	<code>&lt;meta name="keywords" lang="sr" content="html, css, uvit"&gt;</code>
description	<code>&lt;meta name="description" content="Primer HTML i CSS datoteke"&gt;</code>
generator	<code>&lt;meta name="generator" content="Visual Studio Code"&gt;</code>

### 1.1.4 HTML telo

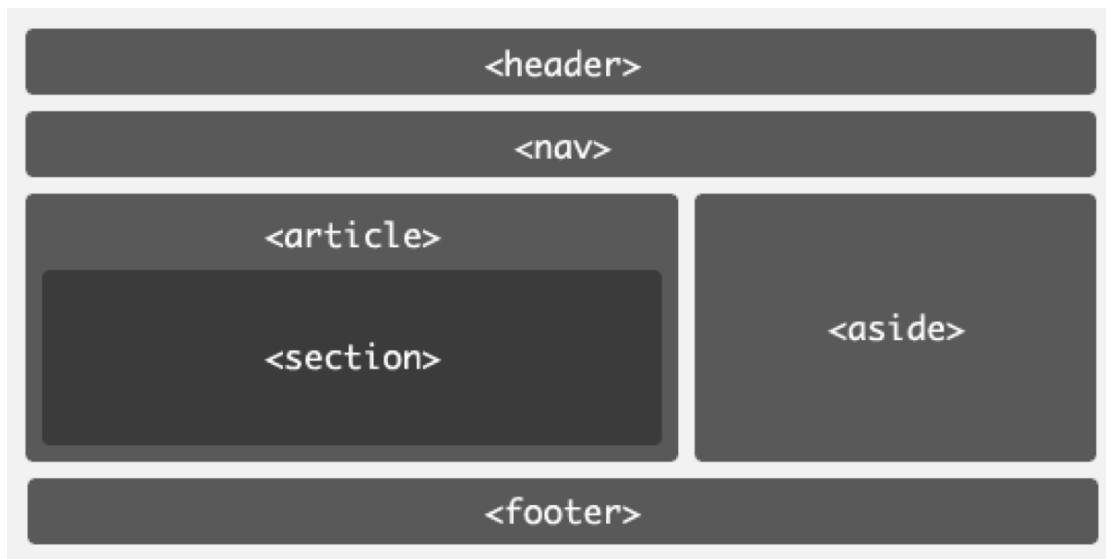
Telo HTML dokumenta je određeno elementom `body`. U okviru njega, potrebno je da vršimo ugnezđavanje elemenata koje želimo da prikažemo korisniku. Elemana ima raznih, i u nastavku ćemo se upoznati sa velikim brojem ovih elemenata.

### 1.1.5 HTML5 semantički elementi (`header`, `nav`, ...)

HTML5 je uveo nove elemente koji označavaju glavne delove HTML dokumenta. Ti elementi su:

- `header` — element za naslovni sadržaj
- `nav` — element za navigaciju
- `main` — glavni sadržaj
- `footer` — element za završni sadržaj
- ...

Ovi elementi definišu isključivo semantičku organizaciju elemenata na stranici, odnosno, oni dodeljuju odgovarajuće značenje elementima koje sadrže i ne podrazumevaju nikakav poseban način prikazivanja tih elemenata. Naredna slika daje primer upotrebe ovih elemenata.



### 1.1.6 Elementi za naslove ( h<sub>1</sub> , ..., h<sub>6</sub> )

Tekst možemo istaknuti u vidu naslova koristeći elemente `<h1>` do `<h6>`. Oni takođe dodeljuju semantički značaj tekstu koji obeležavaju. Takođe, oni podrazumevaju i odgovarajući prikaz teksta koji je dat na sledećoj slici

# Naslov 1

## Naslov 2

### Naslov 3

#### Naslov 4

##### Naslov 5

###### Naslov 6

### 1.1.7 Elementi p i span

Pored naslova, tekst možemo obeležiti i elementima:

- `p` — definiže paragraf teksta. Podrazumevano, tekst obeležen ovim elementom se prostire celom širinom elementa u kojem se nalazi. Tako, na primer, ukoliko se nalazi u elementu `body`, tekst će se prostirati celom širinom prozora veb pregledača:

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Mauris ac ex in eros placerat elementum vel ac purus. Pellentesque iaculis, massa sed sollicitudin tincidunt, dolor lacus fermentum nibh, dictum ornare nulla ipsum a orci. Nulla imperdiet, est ac auctor tincidunt, tellus est condimentum risus, in tincidunt sem lectus eget enim. Nullam ut sapien ut libero tristique blandit. Pellentesque vitae lectus ultrices, placerat odio a, bibendum diam. Sed lobortis, felis in sagittis gravida, lectus ipsum laoreet lorem, sed imperdiet arcu augue id tellus. Aliquam finibus vulputate dui, a gravida felis egestas lacinia. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Duis eu sapien a mauris pellentesque ullamcorper et quis lacus.

- **span** — definiše deo teksta, najčešće jednu reč ili deo rečenice (ali ne nužno). Veličina elementa zavisi od veličine teksta koji se nalazi kao njegov sadržaj.

### 1.1.8 Blokovski i linijski elementi

Prethodno opisani elementi su predstavnici dveju bitnih kategorija elemenata: blokovski i linijski<sup>2</sup>. Veb pregledači *blokovske elemente* tipično prikazuju sa praznim redom pre i nakon elementa. Možemo ih vizualizovati kao naslagane kutije. Blok elementi zauzimaju čitavu širinu elementa u kojem se nalaze. Za razliku od njih, *linijski elementi* mogu da počinju bilo gde u liniji (ne nužno u novom redu), i njihova veličina zavisi od njihovog sadržaja. Blokovski elementi mogu da sadrže linijske elemente i (obično) druge blok elemente, dok linijski elementi mogu da sadrže samo linijske elemente.

Na primeru opisanih elemenata možemo demonstrirati ovo ponašanje. Na nadređenoj slici su obeleženi tekst koji se nalazi u okviru elementa **p** (levo) i tekst koji se nalazi u okviru elementa **span** (desno)

The following paragraph is a  
**block-level element**; its background has been colored to display both the beginning and end of the block-level element's influence.  
The following span is an **inline element**; its background has been colored to display both the beginning and end of the inline element's influence.  
its background has been colored to display both the beginning and end of the inline element's influence.

### 1.1.9 Grupisanje elemenata (**div**) i novi red (**br**)

Grupisanje elemenata najčešće služi za izdvajanje logičkih celina, koje se potom veoma često i vizualno izdvajaju od ostalih celina. HTML5 semantički elementi omogućavaju jedan način grupisanja elemenata, na način na koji smo već videli. Dodatno, na raspolaganju nam je blokovski element **div**, koji takođe ne podrazumeva poseban način prikazivanja (osim što je u pitanju blokovski element, te ima njihova svojstva), već pomoću njega možemo da izdvojimo i, najčešće, imenujemo celine koje ćemo stilizovati na neki način.

Forsiranje prelaska u novi red se, pored korišćenja blokovskih elemenata, može izvesti i upotrebom samozatvarajućeg elementa **br**.

<sup>2</sup>Ove klase su definisane standardom pre HTML5, dok se u verziji HTML5 definišu nove kategorije.

**Zadatak 1.1:**

Napisati HTML datoteku koja odgovara stranici na narednoj slici

**HTML i CSS**

- Uvod -

**HTML**

HTML (енгл. HyperText Markup Language, језик за означавање хипертекста) је описни језик специјално намењен опису веб страница. Помоћу њега се једноставно могу одвојити елементи као што су наслови, параграфи, цитати и слично. Поред тога, у HTML стандард су уgraђени елементи који детаљније описују сам документ као што су кратак опис документа, кључне речи, подаци о аутору и слично. Ови подаци су општепознати као мета подаци и јасно су одвојени од садржаја документа. Актуелна верзија стандарда је HTML 5, а сам стандард одржава Конзорцијум за Веб (W3C, World Wide Web Consortium).

**CSS**

CSS синтакса се састоји од описа изгледа елемената у документу. Опис може да дефинише изглед више елемената, и више описа може да дефинише један елемент. На тај начин се описи слажу један преко другог да би дефинисали коначни изглед одређеног елемента (отуда назив Cascading да би се доčaralo слагање једног стила преко другог у дефинисању коначног изгледа елемента).

15. oktobar 2018.

**Rešenje:**

Kod 2: zadaci/001/strana.html

```
<!DOCTYPE html>

<html>
  <head>
    <title>HTML i CSS</title>
    <meta charset="UTF-8">
    <meta name="author" content="Ajzenhamer N, Bukurov A.">
    <meta name="generator" content="Visual Studio Code">
    <meta name="keywords" content="css, html, uvod, veb, uvit">
    <meta name="description" content="Uvod u HTML i CSS">
  </head>

  <body>
    <!-- primer HTML komentara -->

    <header>
      <h1>HTML i CSS</h1>
      <h4>-- Uvod --</h4>
    </header>

    <main>
      <!-- Deo o HTML-u -->
      <div>
        <h3>HTML</h3>
        <p>
          HTML (engl. HyperText Markup Language, jezik za označavanje hiperteksta) je opisni jezik specijalno namenjen opisu veb stranica. Pomocu njega se jednostavno mogu odvojiti elementi kao što su naslovi, paragrafi, citati i slično. Pored toga, u HTML standard su ugrađeni elementi koji detaljnije opisuju sam dokument као што су кратак опис документа, кључне речи, подаци о аутору и слично. Ови подаци су општепознати као мета подаци и јасно су
        </p>
      </div>
    </main>
  </body>
</html>
```

```
odvojeni od sadržaja dokumenta.

Aktuelna verzija standarda je HTML 5, a sam standard
održava Konzorcijum za Veb (W3C, World Wide Web
Consortium).

</p>
</div>

<!-- Deo o CSS-u -->
<div>
    <h3>CSS</h3>
    <p>
        CSS sintaksa se sastoji od opisa izgleda elemenata u
        dokumentu. Opis može da definiše izgled više elemenata,
        i više opisa može da definiše jedan element. Na taj
        način se opisi slažu jedan preko drugog da bi definisali
        konačni izgled određenog elementa (otuda naziv
        Cascading da bi se dočaralo slaganje jednog stila preko
        drugog u definisanju konačnog izgleda elementa).
    </p>
</div>
</main>

<footer>
    <p>15. oktobar 2018.</p>
</footer>

</body>
</html>
```

## 1.2 Uvod u CSS

CSS (engl. *Cascading Style Sheets*) je jezik koji služi za stilizovanje HTML elemenata<sup>3</sup>. CSS možemo koristiti na tri načina:

1. Navođenjem stilova u odvojenim CSS datotekama (uz njihovo linkovanje u HTML datoteci). Element oblika

```
<link rel="stylesheet" type="text/css" href="dokument.css">
```

navodi se u okviru `head` elementa HTML dokumenta. Ovim se omogućava da više HTML datoteka može da pristupi jednom CSS dokumentu (čime se postiže modularnost i smanjenje redundantnih stilova).

2. Putem HTML elementa `style` (koji se takođe navodi u okviru `head` elementa koji ima atribut `type="text/css"`), u čijem sadržaju se nalaze stilovi. Navedeni stilovi važe za ceo dokument u čijem zaglavlju se nalaze, ali nisu vidljivi drugim HTML dokumentima.

<sup>3</sup>Kaskadni stilski listovi se čuvaju kao CSS dokumenti, tj. sa ekstenzijom css.

3. Korišćenjem generičkog atributa `style` nad elementima<sup>4</sup>. Stilovi definisani na ovakav način su vidljivi i važe isključivo nad pojedinačnim elementom nad kojim su definisani.

Ono što je važno napomenuti jeste da često dolazi do preklapanja stilova. Na primer, moguće je nad celim dokumentom definisati da svi paragrafi budu obojeni crvenom bojom, a zatim posebno definisati za jedan paragraf da je obojen plavom bojom. U tu svrhu, definisana su pravila kojim se određuje koje svojstvo dolazi do izražaja u slučaju preklapanja.

Najveću težinu imaju stilovi definisani korišćenjem atributa `style`, zatim atributi definisani u zaglavlju HTML dokumenta (korišćenjem elementa `style` i ugnezđavanjem eksternih CSS dokumenata), i na kraju, najmanju težinu imaju podrazumevane vrednosti veb pregledača.

Takođe, potrebno je obratiti pažnju na redosled elemenata `style` i `link` (koji vrši ugnezđavanje eksternog CSS dokumenta) u zaglavlju HTML dokumenta. Na primer, pretpostavimo da eksterna CSS datoteka `plavi_naslov.css` definiše da naslov `h1` treba biti obojen u plavu boju. Takođe, neka je dat element `style` u zaglavlju koji definiše da naslov `h1` treba biti obojen u narandžastu boju. Na narednoj slici su dati rezultati kada se element `style` nalazi nakon elementa `link` (levo) i kada se element `link` nalazi nakon elementa `style` (desno). Vidimo da veću težinu ima ono pravilo koje je poslednje pronađeno (tj. ono je kaskadno poslednje složeno).

### This is a heading

The style of this document is a combination of an external stylesheet, and internal style

### This is a heading

The style of this document is a combination of an external stylesheet, and internal style

Ipak, postoji način da se forsira određeni stil, a to je navođenjem `!important` ključne reči u okviru CSS deklaracije. Korišćenjem ove ključne reči, navedena deklaracija će uvek imati prednost nad svim ostalima. Više o deklaracijama u nastavku teksta.

#### 1.2.1 CSS sintaksa

CSS jezik se sastoje od niza *pravila*. Svako pravilo je oblika:

```
selektor {
    svojstvo-1: vrednost_1;      /* deklaracija 1 */
    ...
    svojstvo-N: vrednost_N !important; /* deklaracija N */
}
```

Objasnjimo sada svaku od predstavljenih stavki:

- *Selektor* predstavlja način za definisanje nad kojim HTML elementima će stil biti primenjen. Postoji veliki broj selektora, a mi ćemo predstaviti neke od njih:

<sup>4</sup>Da li razumete razliku između elementa `style` i atributa `style`?

Selektor	Primer	Objašnjenje
<i>Ime elementa</i>	<code>p</code>	Primeniće stil nad svim elementima <code>p</code>
<i>Identifikator</i>	<code>#moj-id</code>	Primeniće stil nad svim elementima koji imaju postavljen generički atribut <code>id="moj-id"</code> . Poželjno je da na nivou dokumenta postoji samo jedan element koji ima svoj identifikator, čime se on jedinstveno određuje.
<i>Klasa</i>	<code>.moja-k1</code>	Primeniće stil nad svim elementima koji imaju postavljen generički atribut <code>class="moja-k1"</code> . Stilovi se definišu pomoću klase kada je potrebno na isti način predstaviti vizualno srodne elemente na stranici, na primer, kartice sa obaveštenjima. Primeniće stil na sve elemente odabrane sa <code>sel2</code> , a koji se nalaze u okviru elemenata odabranih sa <code>sel1</code> .
<i>Ugnežđeni</i>	<code>sel1 sel2</code>	Primeniće stil na sve pasuse (elemente <code>p</code> ) sa atributom <code>class="m-k1"</code> , a koji se nalaze u okviru elemenata <code>div</code> .
<i>Kompozicija</i>	<code>div p.m-k1</code>	Primeniće stil na sve elemente <code>div</code> i sve elemente <code>p</code> .
<i>Grupisanje</i>	<code>div, p</code>	

- *Deklaracija* opisuje način na koji se vrši stilizovanje selektovanih elemenata. Deklaracije se međusobno razdvajaju karakterom `;`.
- Svaka deklaracija sadrži par *svojstvo i vrednost*. Svaki element ima definisana svojstva koja je moguće primeniti nad njime. Takođe, svako svojstvo ima definisan opseg vrednosti koje može uzimati. U nastavku teksta ćemo se upoznati za raznim svojstvima i odgovarajućim vrednostima.

Komentari se u jeziku CSS navode između `/* i */`.

Možemo testirati da li naš CSS dokument predstavlja validno napisan dokument pomoću CSS validatora. Jedan primer validatora je moguće pronaći na adresi <http://jigsaw.w3.org/css-validator/>.

### 1.2.2 Dodeljivanje boje elementu

U CSS jeziku, za dodeljivanje boje elementu možemo koristiti sledeća dva svojstva:

- `color` — zadaje boju teksta koji se nalazi u sadržaju elementa
- `background-color` — zadaje boju pozadine elementa

Vrednosti za boje se mogu specifikovati na više načina, a mi navodimo sledeće:

- Pomoću imena boje, na primer, `Red`, `Orange`, `DodgerBlue`, `Violet`, `MediumSeaGreen`, itd. Spisak svih dostupnih imena boja je dostupan na [ovoj adresi \(w3schools.com\)](http://www.w3schools.com/colors/colors_names.asp).
- Pomoću RGB i RGBA modela. Ovim modelima je potrebno zadati vrednosti iz opsega [0, 255] za crvenu (R), zelenu (G) i plavu (B) boju čijim se "mešanjem" dobija oko 16,7M boja. Varijanta RGBA dodatno omogućava da se podesi prozirnost boje (alfa kanal, A) u opsegu [0.0, 1.0]. Na primer,

```
color: rgb(255, 0, 0); /* crvena boja */
background-color: rgba(0, 255, 0, 0.5); /* zelena boja sa prozirnoscu
od 50% */
```

- Pomoću HEX modela. Ovim modelom se za svaku od crvene, zelene i plave, redom, navodi dvocifreni heksadekadni broj iz intervala [00 – FF]. Na primer,

```
color: #0000ff; /* plava boja */
```

Više o modelima boja se može pronaći na [ovoj adresi \(w3schools.com\)](http://www.w3schools.com).

### 1.2.3 Formatiranje teksta

Sada ćemo se upoznati sa nekim svojstvima kojima je moguće uticati na prikaz teksta:

- font-family** — zadaje familiju fonta koji se koristi. Familija fonta se može zadati po imenu, na primer `Arial` ili `'DejaVu Serif'` (ukoliko ima znak razmaka, stavljaju se navodnici), ili se može nazvati generička familija fontova, pa se ostavlja veb pregledaču da odabere konkretan font iz te familije koji mu je dostupan. Na primer, `Sans-Serif` je familija fontova bez kukica (npr. Arial, Helvetica, ...). Analogno tome, `Serif` je familija fontova sa kukicama (npr. Timew New Roman, Georgia, ...). Takođe, moguće je staviti više fontova odvojenih karakterom `,`, da bi veb pregledač, u slučaju da mu nije dostupan neki font, prikazao odgovarajuću zamenu. Na primer

```
font-family: 'Neki kul font koji mozda nije dostupan', Arial,
Sans-Serif;
```

- font-size** — podešava veličinu slova. Vrednosti se mogu iskazati u pikselima (`px`), u `em` (važi  $1em = 16px$ ), ili u procentima, pri čemu se razmatra okolina teksta da bi se zaključio odnos. Ove jedinice se nazivaju *dužinama*, i više o njima se može pronaći na [ovoj adresi \(w3schools.com\)](http://www.w3schools.com).
- font-style** — kontroliše iskošenosti slova. Vrednosti su: `normal` (uspravna slova), `italic` (iskošena slova) i `oblique` (nakošena slova, slično kao `italic`, ali manje podržano od veb pregledača).
- font-weight** — kontroliše potamnjivanje slova. Vrednosti su: `normal` (podrazumevana debljina), `bold` (deblje), `bolder` (više podebljano), `lighter` (tanje) ili brojevi od `100` do `900` (sa korakom 100), pri čemu broj `400` odgovara vrednosti `normal`, a `700` odgovara vrednosti `bold`.
- text-align** — kontroliše horizontalno poravnjanje teksta. Vrednosti su: `left` (levo poravnato), `right` (desno poravnato), `center` (centrirano) i `justify` (poravnato sa obe strane).
- letter-spacing** — podešava rastojanje između slova. Vrednosti su: `normal` (podrazumevano rastojanje) ili neka od dužina.

- `word-spacing` — podešava rastojanje između reči. Vrednosti su: `normal` (podrazumevano rastojanje) ili neka od dužina.
- `text-indent` — kontroliše nazubljivanje prve linije teksta. Vrednosti su neka od dužina.
- `text-shadow` — dodaje senku tekstu elementa kome postavljamo svojstvo. Ima 2-4 vrednosti. Prva vrednost predstavlja pomeraj po horizontali, druga pomeraj po vertikali u odnosi na tekst. Treća vrednost omogućava zamućenje senke i može se izostaviti. Četvrta vrednost predstavlja boju senke i ona se takođe može izostaviti.

Više o fontovima se može pronaći na [ovoj adresi \(w3schools.com\)](http://www.w3schools.com), dok se više o tekstualnim promenama može pronaći na [ovoj adresi \(w3schools.com\)](http://www.w3schools.com).

#### 1.2.4 Dodavanje korisničkih fontova

Ukoliko želimo da koristimo svoj font, možemo ga definisati u CSS dokumentu pomoću `@font-face` naredbe. Njena struktura je oblika:

```
@font-face {  
    font-family: 'Naziv fonta';  
    src: url('./lokacija/do/mog/fonta/Naziv Fonta.ttf');  
}
```

Svojstvom `font-family` dodelujemo naziv korisničkom fontu. Dalje se font može koristiti pomoću već opisanog postupka korišćenjem svojstva `font-family`. Svojstvom `src` zadajemo putanju ka datoteci koja sadrži informacije o fontu. Ona se može nalaziti u lokalnom računaru ili na nekoj veb lokaciji, ali se u oba slučaja navodi korišćenjem vrednosti `url('...')`.

#### Zadatak 1.2:

Napisati HTML datoteku i CSS datoteku koja odgovara stranici na narednoj slici

## HTML I CSS

- Uvod -

### HTML

HTML (engl. *HyperText Markup Language*, jezik za označavanje hiperteksta) je opisni jezik specijalno namenjen opisu veb stranica. Pomoću njega se jednostavno mogu odvojiti elementi kao što su naslovni, paragrafi, citati i slično. Pored toga, u HTML standard su ugrađeni elementi koji detaljnije opisuju sam dokument kao što su kratak opis dokumenta, ključne reči, podaci o autoru i slično. Ovi podaci su opštepoznati kao meta podaci i jasno su odvojeni od sadržaja dokumenta. Aktuelna verzija standarda je HTML 5, a sam standard održava Konzorcijum za Veb (W3C, World Wide Web Consortium).

### CSS

CSS sintaksa se sastoji od opisa izgleda elemenata u dokumentu. Opis može da definiše izgled više elemenata, i više opisa može da definiše jedan element. Na taj način se opisi slazu jedan preko drugog da bi definisali konačni izgled određenog elementa (otuda naziv Cascading da bi se dočaralo slaganje jednog stila preko drugog u definisanju konačnog izgleda elementa).

15. oktobar 2018.

**Rešenje:**

Kod 3: zadaci/002/strana.html

```
<!DOCTYPE html>

<html>
    <head>
        <title>HTML i CSS</title>
        <meta charset="UTF-8">

        <link rel="stylesheet" type="text/css" href="stil.css">

        <style type="text/css">
            /*
                Primer CSS komentara
            */
        </style>
    </head>

    <body>
        <!-- primer HTML komentara -->

        <header>
            <h1 id="glavni_naslov">HTML i CSS</h1>
            <h4 id="podnaslov">-- Uvod --</h4>
        </header>

        <main>
            <!-- Deo o HTML-u -->
            <div>
                <h3>HTML</h3>
                <p class="intro html">
                    HTML (engl. <span style="color: #DA5FA6;">HyperText Markup
                    Language</span>, jezik za označavanje hiperteksta) je
                    opisni jezik specijalno namenjen opisu veb stranica.
                    Pomocu njega se jednostavno mogu odvojiti elementi kao
                    što su naslovi, paragrafi, citati i slično. Pored toga,
                    u HTML standard su ugrađeni elementi koji detaljnije
                    opisuju sam dokument kao što su kratak opis dokumenta,
                    ključne reči, podaci o autoru i slično. Ovi podaci su
                    opštepoznati kao meta podaci i jasno su odvojeni od
                    sadržaja dokumenta.

                    Aktuelna verzija standarda je HTML 5, a sam standard
                    održava Konzorcijum za Veb (W3C, World Wide Web
                    Consortium).
                </p>
            </div>
            <!-- Deo o CSS-u -->
            <div>
                <h3>CSS</h3>
                <p id="css" class="intro">
```

```
CSS sintaksa se sastoje od opisa izgleda elemenata u dokumentu. Opis može da definiše izgled više elemenata, i više opisa može da definiše jedan element. Na taj način se opisi slažu jedan preko drugog da bi definisali konačni izgled određenog elementa (otuda naziv Cascading da bi se dočaralo slaganje jednog stila preko drugog u definisanju konačnog izgleda elementa).  
    </p>  
  </div>  
</main>  
  
<footer>  
  <p>15. oktobar 2018.</p>  
</footer>  
  
</body>  
</html>
```

Kod 4: zadaci/002/stil.css

```
@font-face {  
  font-family: 'Nas font';  
  src: url('cool_font.ttf');  
}  
  
#glavni_naslov {  
  text-align: center;  
  font-family: 'Nas font', Arial, Sans-Serif;  
  font-size: 50px;  
  text-shadow: 2px 2px grey;  
}  
  
#podnaslov {  
  text-align: center;  
  font-size: 25px;  
  font-family: Sans-Serif;  
}  
  
.intro {  
  color: SlateBlue;  
}  
  
.html {  
  font-size: 14px;  
  font-family: 'Tahoma', 'Arial', sans-serif;  
  letter-spacing: 2px;  
  text-indent: 20px;  
  text-align: justify;  
}  
  
#css {  
  color: rgb(60, 154, 109);  
  font-style: italic;  
}
```

```

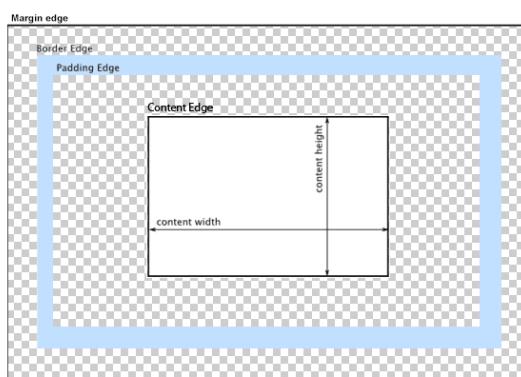
main div h3 {
    font-weight: bold;
}

header h1, header h4{
    background-color: violet;
    color: white;
}

```

### 1.3 Model kutije

Sve HTML elemente možemo posmatrati kao pravougaonike na ekranu, te se stoga zovu ponekad i *kutije*. U jeziku CSS, termin *model kutije* opisuje kutiju koja okružuje sadržaj svakog elementa. Naredna slika demonstrira prikaz modela kutije



Dakle, model kutije se sastoji od sledećih elemenata (elementi su nabrajani tako da svaki sledeći uokviruje sve prethodne):

- *Sadržaj* (engl. *Content*) — kutija u kojoj se prikazuje tekst, slika ili drugi sadržaj HTML elementa.
- *Punjene* (engl. *Padding*) — prostor oko sadržaja, ali koji se još uvek smatra "unutrašnošću" elementa, s obzirom da se nalazi unutar ivice. Punjenje je uvek prozirno.
- *Ivica* (engl. *Border*) — kutija koja deli unutrašnjost elementa od spoljašnjosti (odn. od drugih elemenata).
- *Pojas* (engl. *Margin*) — prostor oko ivice elementa, koji pre svega služi da stvori bezbedni pojaz oko elementa i odvoji ga od okolnih elemenata (otuda i njegov prevod). Pojas je uvek proziran.

#### 1.3.1 Sadržaj elementa

Veličina kutije sadržaja zavisi od dve stvari. Prvo, koja je vrsta elementa u pitanju (linijski i blokovski), a zatim i od postavljenih svojstava `width` (širina) i `height` (visina).

Veličina kutije sadržaja linijskih elemenata zavisi isključivo od veličine njihovog sadržaja. Na linijske elemente nije moguće uticati svojstvima `width` i `height`. Naredni primer ilustruje ovo ponašanje

Kod 5: `primeri/model_kutije/box_model_inline.html`

```
<!DOCTYPE html>
<html>
<head>
    <title>Document</title>
    <meta charset="UTF-8">

    <style type="text/css">
        #s1 {
            /* height i width svojstvo nemaju nikakvog efekta */
            height: 100px;
            width: 1000px;
            background-color: rgb(193, 22, 193);
        }
    </style>
</head>
<body>
    <span id="s1">TEST SADRZAJ</span>
    <span id="s2">TEST SADRZAJ</span>
</body>
</html>
```

Kao što vidimo na narednoj slici, iako smo podesili visinu i širinu, linijski element `span` je zauzeo samo onoliko prostora koliko mu je neophodno za njegov sadržaj.



TEST SADRZAJ TEST SADRZAJ

Za razliku od njih, na veličinu sadržaja blokovskih elemenata se može uticati ovim dvama svojstvima. Oni mogu uzimati naredne vrednosti: `auto` (prepušta se veb pregledaču da izračuna širinu/visinu elementa, što je podrazumevano ponašanje), dužina ili procenat. Naredni primer ilustruje ovo ponašanje

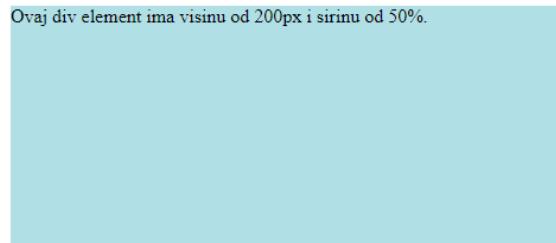
Kod 6: `primeri/model_kutije/block_sirina_visina.html`

```
<!DOCTYPE html>
<html>
<head>
    <title>Document</title>
    <meta charset="UTF-8">

    <style type="text/css">
        div {
            height: 200px;
            width: 50%;
            background-color: powderblue;
        }
    </style>
</head>
<body>
```

```
<div>
    Ovaj div element ima visinu od 200px i sirinu od 50%.
</div>
</body>
</html>
```

Kao što vidimo na narednoj slici, blokovski element `div` ima ispravno podešenu visinu i širinu.



### 1.3.2 Punjenje elementa

Punjene elemente možemo zadati u jeziku CSS pomoću svojstva `padding` koje istovremeno podešava gornje, desno, donje i levo punjenje. Međutim, ukoliko želimo da specifikujemo posebnu veličinu punjenja na nekoj strani, možemo iskoristiti neko od svojstava `padding-top`, `padding-right`, `padding-bottom` ili `padding-left`. Vrednosti za sva opisana svojstva su: dužina ili procenat. Podrazumevana vrednost je 0.

Naredni primer i prateća slika ilustruju postavljanje punjenja za svaku stranu ponaosob. Dodatno, element `div` iz primera sadrži crnu ivicu kako bi se lakše razumeo efekat punjenja. Primetimo da će svojstvo `background-color` takođe obojiti i boju punjenja.

Kod 7: primeri/model\_kutije/block\_padding.html

```
<!DOCTYPE html>
<html>
<head>
    <title>Document</title>
    <meta charset="UTF-8">

    <style type="text/css">
        div {
            border: 1px solid black;
            background-color: lightblue;
            padding-top: 50px;
            padding-right: 30px;
            padding-bottom: 100px;
            padding-left: 280px;
        }
    </style>
</head>
<body>
    <div>
        Ovaj div element ima gornje punjenje od 50px, desno punjenje od 30px,
        donje punjenje od 100px i levo punjenje od 280px.
    </div>
</body>
</html>
```

```
</div>
</body>
</html>
```

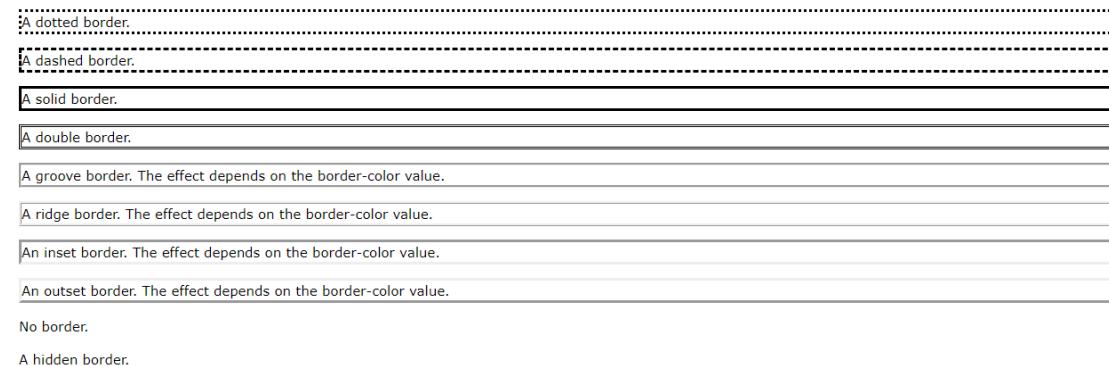
Ovaj div element ima gornje punjenje od 50px, desno punjenje od 30px, donje punjenje od 100px i levo punjenje od 280px.

Punjenje proizvodi isti efekat i za linijske i za blokovske elemente.

### 1.3.3 Ivica elementa

U CSS jeziku, svojstvom `border` je moguće postaviti stil, širinu i boju kutije koja odvaja unutrašnjost elementa od njegovog pojasa (odnosno, od okoliko ukoliko pojas ne postoji). Ipak, prvo ćemo se upoznati sa svojstvima koja postavljaju ove tri stavke odvojeno, pre nego što pokažemo kako ih je moguće ukombinovati pomoću svojstva `border`.

**Stil ivice** Stil ivice definišemo pomoću svojstva `border-style`. Moguće vrednosti su: `dotted`, `dashed`, `solid`, `double`, `groove`, `ridge`, `inset`, `outset`, `none` ili `hidden`. Naredna slika ilustruje svaki od ovih stilova.



Takođe, ukoliko želimo da definišemo različite stилove za svaku ivicu ponaosob, umesto navođenja jedne vrednosti (ćime podešavamo sve četiri ivice), možemo nавести četiri vrednosti koje će postaviti stилove za gornju, desnu, donju i levu ivicu, redom. Na primer, rezultat narednog pravila

```
p.mix {
    border-style: dotted dashed solid double;
}
```

dat je na narednoj slici

A mixed border.

Napomenimo da nijedna od preostala dva svojstva, širina i boja, neće biti vidljiva ukoliko nije podešen stil ivice.

**Širina ivice** Širinu ivice definišemo pomoću svojstva `border-width`. Moguće vrednosti su: dužine, `thin`, `medium` ili `thick`. Poput svojstva za stil ivice, i ovo svojstvo može imati jednu ili četiri vrednosti. Semantika ovih dveju deklaracija je identična kao za prethodno svojstvo. Podrazumevana vrednost za širinu je `1px`.

**Boja ivice** Boju ivice definišemo pomoću svojstva `border-color`. Moguće vrednosti su: ime boje, boja zadata u HEX modelu, boja zadata u RGB ili RGBA modelima ili `transparent`. Kao i prethodna dva svojstva, i ovo svojstvo može imati jednu ili četiri vrednosti, uz identičnu semantiku. Podrazumevana vrednost za boju je `black`.

**Navođenje konkretne ivice** Svako od prethodna tri svojstva ima i varijantu kojom se navodi na koju stranu ivice svojstvo treba da se primeni. Tako, na primer, posred svojstva `border-style`, postoje i svojstva `border-top-style`, `border-right-style`, `border-down-style` i `border-left-style`. Analogno važi i za svojstva `border-width` i `border-color`.

**Svojstvo border** Dakle, ukoliko želimo za ivicu da detaljno specifikujemo izgled, moramo da navedemo tri deklaracije, po jednu za stil, širinu i boju. Navedene tri deklaracije se mogu skratiti u jednu tako što se koristi svojstvo `border` (ili neko od svojstava `border-top`, ..., `border-left` za svaku ivicu ponaosob). Vrednost ovog svojstva čine: (1) vrednost za stil (obavezno), (2) vrednost za širinu i (3) vrednost za boju, odvojenu znakom razmaka. Ukoliko se ne navede neka od (2) ili (3), koristiće se podrazumevane vrednosti. Na primer, CSS pravilom

```
p {  
    border: 5px solid red;  
}
```

proizvodi se



```
Some text
```

dok se CSS pravilom

```
p {  
    border-left: solid 10px;  
    background-color: lightgrey;  
}
```

proizvodi



```
Some text.
```

**Svojstvo border-radius** Svojstvo `border-radius` služi za kontrolu zakriviljenosti čoškova elementa. Moguće vrednosti su: dužina ili procenat. Ukoliko navedemo jednu vrednost, svi čoškovi će se zakriviti. Alternativno, možemo navesti četiri vrednosti koje će zakriviti gornji-levi, gornji-desni, donji-desni i donji-levi čošak, redom. Na primer, naredni kod

Kod 8: primeri/model\_kutije/border\_radius.html

```
<!DOCTYPE html>
<html>
<head>
    <title>Document</title>
    <meta charset="UTF-8">

    <style type="text/css">
        #example1 {
            border: 2px solid red;
            padding: 10px;
            border-radius: 25px;
        }

        #example2 {
            border: 2px solid red;
            padding: 10px;
            border-radius: 50px 20px 50px 50px;
        }
    </style>
</head>
<body>
    <h2>border-radius: 25px:</h2>

    <div id="example1">
        <p>Svojstvo border-radius definise zakriviljenje coskova elementa.</p>
    </div>

    <h2>border-radius: 50px 20px 50px 50px:</h2>

    <div id="example2">
        <p>Mozemo staviti jednu vrednost za sve coskove, ili cetiri vrednosti
           , za svaki cosak od gornjeg-levog do donjeg-levog po jednu.</p>
    </div>
</body>
</html>
```

formatira se u prikaz na narednoj slici

**border-radius: 25px;**

Svojstvo border-radius definise zakrivljenje coskova elementa.

**border-radius: 50px 20px 50px 50px;**

Mozemo staviti jednu vrednost za sve coskove, ili cetiri vrednosti, za svaki cosak od gornjeg-levog do donjeg-levog po jednu.

### 1.3.4 Pojas elementa

CSS svojstvo `margin` koristi se za podešavanje prostora oko elementa, tj. oko njegove ivice. Slično kao i za punjenje elementa, ovo svojstvo može uzimati jednu ili četiri vrednosti, u zavisnosti od toga da li želimo da postavimo jednu vrednost za sva četiri pojasa ili svaku vrednost pojedinačno (u tom slučaju navodimo vrednosti za gornji, desni, donji i levi pojasa, redom). Takođe, moguće je podesiti vrednosti za četiri pojasa korišćenjem svojstava `margin-top`, `margin-right`, `margin-bottom` i `margin-left`.

Vrednosti ovih svojstava mogu biti:

- `auto` — čime se prepušta pregledaču da izračuna marginu
- dužina (px, pt, cm, ...)
- procenat

Naredni primer ilustruje upotrebu ovog svojstva nad blokovskim elementom `p`

Kod 9: primeri/model\_kutije/block\_margin.html

```
<!DOCTYPE html>
<html>
<head>
    <style>
        div {
            border: black solid 1px;
        }

        p {
            border: 1px solid black;
            margin-top: 100px;
            margin-bottom: 100px;
            margin-right: 20%;
            margin-left: 20%;
            background-color: lightblue;
        }
    </style>
</head>
<body>

    <div>
```

```
<p>
    Ovaj element ima gornju marginu od 100px, desnu marginu od 20%,
    donju marginu od 100px i levu marginu od 20%.
</p>
</div>

</body>
</html>
```

Ovaj element ima gornju marginu od 100px, desnu marginu od 20%, donju marginu od 100px i levu marginu od 20%.

**Horizontalno centriranje pomocu svojstva `margin`** Jedan zanimljiv efekat koji se može postići pomoću svojstva `margin` jeste da se element horizontalno centriра, u odnosu na element koji ga sadrži. Postavljanjem deklaracije `margin: auto;`, element će prvo zauzeti odgovarajući prostor svojom širinom, a zatim će se preostali horizontalni prostor podeliti jednakom među levom i desnom marginom. Naredni primer ovo ilustruje

Kod 10: primeri/model\_kutije/margin\_auto.html

```
<!DOCTYPE html>
<html>
<head>
<style>
    body {
        border: 1px solid black;
    }

    div {
        width: 300px;
        margin: auto;
        border: 1px solid red;
    }
</style>
</head>
<body>

<h2>Koriscenje margin:auto</h2>

<div>
    Ovaj element div ce biti horizontalno poravnat jer ima postavljenu
    deklaraciju margin: auto;
</div>

</body>
```

```
</html>
```

### Koriscenje margin:auto

Ovaj element div će biti horizontalno poravnat jer ima postavljenu deklaraciju margin: auto;

**Linijski elementi i svojstvo margin** Za razliku od blokovskih elemenata, linijski elementi imaju specifično ponašanje kada se na njih primeni svojstvo `margin`. Naime, na njih nema uticaja vrednosti koje su postavljene za gornju i donju marginu, ali vrednosti postavljene za levu i desnu marginu se očekivano primenjuju. Naredni primer ovo ilustruje

Kod 11: primeri/model\_kutije/inline\_margin.html

```
<!DOCTYPE html>
<html>
<head>
    <style>
        body {
            border: 1px solid black;
        }

        span {
            margin: 100px 50px 100px 50px;
            border: 1px solid red;
        }
    </style>
</head>
<body>

    <h2>Linijski elementi i svojstvo margin</h2>

    <span>
        Linijski element span nema gornju i donju marginu
    </span>
    <span>
        Linijski element span nema gornju i donju marginu
    </span>

    <br>

    <span>
        Linijski element span nema gornju i donju marginu
    </span>
    <span>
        Linijski element span nema gornju i donju marginu
    </span>

    <br>

</body>
```

```
</html>
```

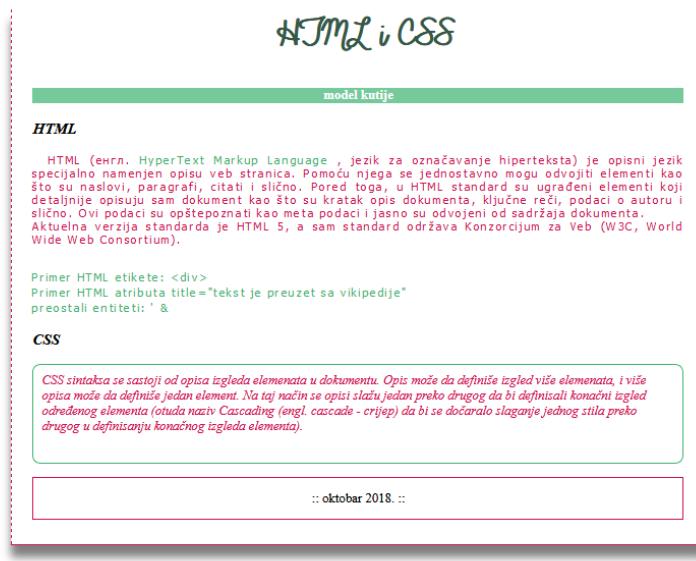
### Linijski elementi i svojstvo margin

Linijski element span nema gornju i donju marginu  
Linijski element span nema gornju i donju marginu

Linijski element span nema gornju i donju marginu  
Linijski element span nema gornju i donju marginu

**Zadatak 1.3:**

Napisati HTML datoteku i CSS datoteku koja odgovara stranici na narednoj slici

**Rešenje:**

Kod 12: zadaci/003/strana.html

```
<!DOCTYPE html>

<html>
  <head>
    <title> Model kutije </title>
    <meta charset='UTF-8'>
    <link rel='stylesheet' href='stil.css'>
  </head>

  <body>

    <header>
      <h1> HTML i CSS </h1>
      <h4> model kutije </h4>
    </header>

    <main>
      <div>
        <h3> HTML </h3>
        <p class='intro html' title='текст је преузет са Википедије'>
          HTML енгл(. <span style='color:mediumseagreen;'> HyperText
          Markup Language </span>, језик за означавање
          хипертекста) је описни језик специјално намењен опису
          веб страница. Помоћу њега се једноставно могу одвојити
          елементи као што су наслови, параграфи, цитати и слично.
        </p>
      </div>
    </main>
  </body>
</html>
```

```
Pored toga, u HTML standard su ugrađeni elementi koji detaljnije opisuju sam dokument kao što su kratak opis dokumenta, ključne reči, podaci o autoru i slično. Ovi podaci su opštepoznati kao meta podaci i jasno su odvojeni od sadržaja dokumenta.  
<br>  
Aktuelna verzija standarda je HTML 5, a sam standard održava Konzorcijum za Veb (W3C, World Wide Web Consortium).  
<br>  
<br>  
</p>  
<span class='example html' title='tekst je preuzet sa  
Vikipedije'>  
Primer HTML etikete: &lt;div&gt;  
<br>  
Primer HTML atributa title="tekst je preuzet sa  
vikipedije";  
<br>  
preostali entiteti: &apos; &amp;  
</span>  
</div>  
  
<div>  
<h3> CSS </h3>  
<p id='css' class='intro'>  
CSS sintaksa se sastoji od opisa izgleda elemenata u dokumentu. Opis može da definiše izgled više elemenata, i više opisa može da definiše jedan element. Na taj način se opisi slažu jedan preko drugog da bi definisali konačni izgled određenog elementa (otuda naziv Cascading (engl. cascade – crijepljivo) da bi se dočaralo slaganje jednog stila preko drugog u definisanju konačnog izgleda elementa).  
</p>  
</div>  
</main>  
  
<footer>  
<p> :: Oktobar 2018. :: </p>  
</footer>  
  
</body>  
</html>
```

Kod 13: zadaci/003/stil.css

```
@font-face{  
    /* Vanessa Bays @ http://bythebutterfly.com */  
    font-family: 'TestFont';  
    src: url('test_font.ttf');  
}
```

```
body{  
    /* leva i desna marga postavljene na auto omogucavaju centriranje –  
     preduslov je promeniti sirinu tako da bude manja od 100% */  
    width: 800px;  
    margin-left:auto;  
    margin-right:auto;  
  
    border-left: 1px dashed rgb(211, 17, 79);  
    border-right: 3px dotted rgb(211, 17, 79);  
    border-bottom: 1px solid rgb(211, 17, 79);  
  
    padding-left: 25px;  
    padding-right:25px;  
  
    box-shadow: 0px 15px 10px 0px gray;  
}  
  
header{  
    text-align: center;  
}  
  
header h1{  
    text-align: center;  
    font-family: 'TestFont', cursive;  
    font-size: 50px;  
    color: rgb(53, 88, 71);  
}  
  
header h4{  
    background-color: mediumseagreen;  
    color: white;  
    /* TODO: eksperimentisati sa razlicitim vrednostima providnosti – od 0  
     do 1 */  
    opacity: 0.7;  
}  
  
.intro{  
    color: rgb(211, 17, 79);  
}  
  
.example {  
    color: mediumseagreen;  
}  
  
.html{  
    font-size: 14px;  
    font-family: 'Tahoma', 'Arial', sans-serif;  
    letter-spacing: 2px;  
    text-indent: 20px;  
    text-align: justify;  
}
```

```
#css{
    border: 2px solid mediumseagreen;
    border-radius: 10px;
    padding: 10px;
    color: rgb(211, 17, 79);
    font-style: italic;
    /* TODO: eksperimentisati sa razlicitim visinama bloka */
    height: 100px;
    overflow: auto;
}

main div h3{
    font-weight: bold;
    font-style: oblique;
}

footer{
    text-align: center;
    text-transform: lowercase;
    border: 2px solid rgb(211, 17, 79);
    margin-bottom: 30px;
}
```

## 2 Čas 4

Cilj ovog časa je upoznavanje studenata sa pojmovima vezanim za *pozicioniranje* elemenata.

### 2.1 Pozicioniranje elemenata

Svojstvo `position` određuje tip metoda za pozicioniranje koje se koristi za određeni element. Moguće vrednosti ovog svojstva su:

- `static`
- `relative`
- `absolute`
- `fixed`

Elementi se pozicioniraju korišćenjem svojstava `top`, `right`, `bottom`, `left`. Ova svojstva neće imati nikakvog efekta na pozicioniranje elemenata ukoliko pre toga nije postavljena vrednost za svojstvo `position`. U zavisnosti od odabranog metoda za pozicioniranje ova svojstva će imati različite efekte na pozicioniranje elementa na stranici, što je opisano u nastavku.

#### 2.1.1 Statičko pozicioniranje

Ovo je podrazumevan način pozicioniranja elemenata. Postavljanje vrednosti za neko od svojstava `top`, `right`, `bottom`, `left` nema uticaj na poziciju elementa sa statičkim pozicioniranjem.

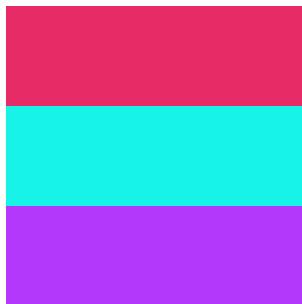
Elementi se redaju na stranici prema normalnom toku stranice. Linijski elementi redaju se jedan pored drugog, dok se blokovski redaju jedan ispod drugog. Na primer, ako na stranici imamo 3 elementa `div`, redom označene identifikatorima "prvi", "drugi" i "treci", i na njih primenjena sledeća CSS pravila

```
#prvi{
    height: 100px;
    width: 300px;
    background-color: rgb(231, 43, 103);
}

#drugi{
    height: 100px;
    width: 300px;
    background-color: rgb(24, 243, 232);
    position:static;
}

#treci{
    height: 100px;
    width: 300px;
    background-color: rgb(180, 55, 252);
}
```

dobijamo



Vidimo da su elementi poređani jedan ispod drugog, kao što se dešava i kad svojstvo `position` nije podešeno.

### 2.1.2 Relativno pozicioniranje

Relativno pozicioniranje elementa je pozicioniranje u odnosu na njegovu normalnu poziciju, odnosno onu poziciju na kojoj bi se element našao da mu je svojstvo `position` postavljeno na `static`. Postavljanje vrednosti za svojstva `top`, `right`, `bottom`, `left` pomeriće element u odnosu na njegovu normalnu poziciju. Ove izmenе nemaju uticaja na preostali sadržaj stranice tako da ako se stvori neka praznina ona neće biti popunjena elementima koji bi mogli da tu stanu. Ako u prethodnom primeru izmenimo CSS za drugi `div` na sledeći način

```
#drugi{
    height: 100px;
    width: 300px;
    background-color: rgb(24, 243, 232);
    position: relative;
    top: 30px;
    left: 80px;
}
```

dobićemo



### 2.1.3 Apsolutno pozicioniranje

Element sa absolutnim pozicioniranjem postavlja se na stranicu u odnosu na prvog pretka koji ima postavljeno pozicioniranje<sup>5</sup>. Ukoliko takav predak ne postoji, element se pozicionira u odnosu na `body`.

Ako drugom elemntu `div` iz prvog primera postavimo sledeća CSS svojstva:

```
#drugi{  
    height: 100px;  
    width: 300px;  
    background-color: rgb(24, 243, 232);  
    position: absolute;  
    top: 30px;  
    left: 80px;  
}
```

dobićemo



Kako ovaj element nema pretka pozicioniran je u odnosu na `body`.

Ako dodamo novi element `div` sa identifikatorom "omotac" u koji ćemo smestiti prethodna 3 elementa dodelimo mu sledeća CSS svojstva:

```
#omotac{  
    height: 400px;  
    width: 400px;  
    background-color: rgb(112, 212, 87);  
    position: relative;  
    top: 50px;  
}
```

dobićemo



<sup>5</sup>Smatramo da je pozicioniranje postavljeno ako svojstvo `position` ima bilo koju vrednost osim `static`.

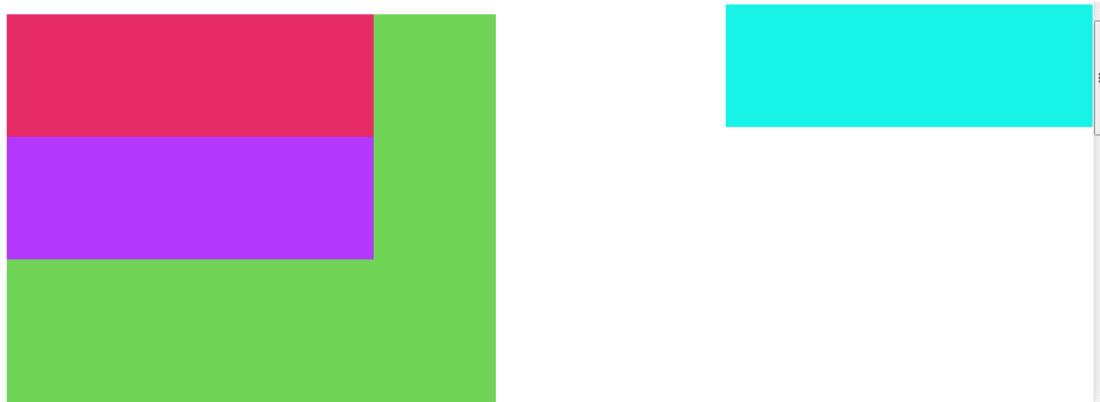
Kako sad element sa absolutnim pozicioniranjem ima pretka sa postavljenim pozicioniranjem, njegovo pozicioniranje određuje se u odnosu na pretka, a ne u odnosu na element `body`, kao u prethodnom primeru.

#### 2.1.4 Fiksno pozicioniranje

Element sa fiksnim pozicioniranjem nalazi se na istoj poziciji na ekranu, čak i kada se stranica skroluje. U slučaju ovakvog pozicioniranja, ne ostaju praznine tamo gde bi se element inače našao. Izmenimo ponovo prethodni primer, tako što ćemo dodati jedan omotač oko tri elementa `div`, a drugi `div` izmeniti tako da sada ima sledeća CSS svojstva:

```
#omotac{  
    height: 900px;  
    width: 400px;  
    background-color: rgb(112, 212, 87);  
}  
  
#drugi{  
    height: 100px;  
    width: 300px;  
    background-color: rgb(24, 243, 232);  
    position:fixed;  
    top: 0px;  
    right: 0px;  
}
```

Time dobijamo



a kad skrolujemo stranicu



### 2.1.5 Z-pozicioniranje elemenata

Kada pozicioniramo elemente može se desiti da se neki preklope. Svojstvo `z-index` određuje kako se elementi slažu jedan preko drugog, odnosno koji će element biti ispred ostalih.

Element može imati pozitivan ili negativan z-indeks. Onaj koji ima veći indeks biće pozicioniran ispred onog sa manjim. Ukoliko se dva elementa vez određenih z-indeksa preklope, onda će ispred biti onaj element koji je poslednji pozicioniran. U primeru sa relativnim pozicioniranjem, drugi i treći `div` se preklapaju, a nismo im odredili z-indekse. Kako je drugi `div` poslednji pozicioniran, on se nalazi ispred.

Ukoliko bismo u tom primeru izmenili svojstva drugog tako da ima određen z-indeks:

```
#drugi{  
    height: 100px;  
    width: 300px;  
    background-color: rgb(24, 243, 232);  
    position: relative;  
    top: 30px;  
    left: 80px;  
    /* kontrola preklapanja elemenata */  
    z-index: -5;  
}
```

dobijamo ovakav raspored elemenata

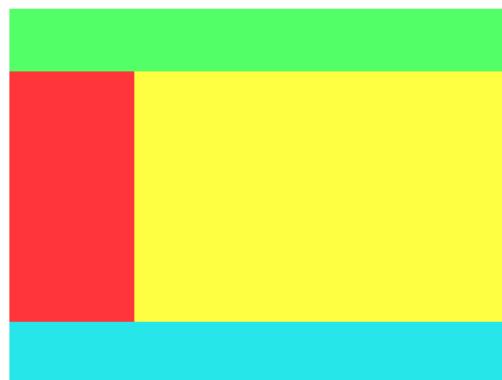


### 2.1.6 Svojstvo `display`

Pomoću svojstva `display` određujemo da li i kako će element biti prikazan. Svaki HTML element ima podrezumevanu vrednost za ovo svojstvo u zavisnosti od tipa elementa. Za većinu elemenata podrazumevana vrednost je `block` ili `inline`.

Podestimo se, blokovski elementi uvek počinju u novom redu i zauzimaju punu širinu koja je slobodna, odnosno prostiru se od leve do desne ivice roditeljskog elementa. U ove elemente spadaju `div`, `h1 - h6`, `p`, `header`, `footer`, itd. Sa druge strane, linijski elementi ne počinju u novom redu i zauzimaju onoliko prostora koliko im je neophodno. Neko od linijskih elemenata su `img`, `span`, `a`, itd.

Podrazumevana vrednost se može izmeniti tako da, na primer, stavke liste budu prikazane sve u istoj liniji ili da veza zauzme celu liniju. Promena svojstva `display` može biti korisna za postizanje odgovarajućeg izgleda stranice, a da se pritom ne naruše standardi Veba. Na primer, česta struktura stranice je da naslovni deo bude na vrhu, zatim navigacija i sadržaj stranice u istoj liniji ispod naslovnog dela, i na dnu futer, kao na slici



Da bismo postigli ovakav raspored elemenata, nepohodno je da crvenom i žutom elementu izmenimo način prikaza, odnosno da svojstvo `display`, koje je za element `div` podrazumevano postavljeno na `block`, izmenimo na vrednost `inline-block`. Tako dobijamo da se blokovski elementi redaju u istoj liniji, pri čemu i dalje zadržavaju svojstva blokovskog elementa. Naredni kod daje ovakav izlged stranice:

Kod 14: primeri/pozicioniranje/inline-block.html

```
<html>
  <head>
    <title> </title>

    <style type="text/css">
      #wrapper{
        margin-left: auto;
        margin-right: auto;
        width: 400px;
        position: relative;
      }

      #header{
```

```
        width: 400px;
        height: 50px;
        background-color: rgb(83, 255, 103);
    }

#nav{
    width: 100px;
    height: 200px;
    background-color: rgb(255, 53, 57);
    display: inline-block;
    vertical-align: top;
}

#content{
    width: 300px;
    height: 200px;
    background-color: rgb(255, 255, 65);
    display: inline-block;
    vertical-align: top;
}

#footer{
    width: 400px;
    height: 50px;
    background-color: rgb(39, 230, 233);
}

</style>

</head>

<body>
    <div id="wrapper">
        <div id="header"> </div>
        <div id="nav"> </div><div id="content"> </div>
        <div id="footer"> </div>
    </div>
</body>
</html>
```

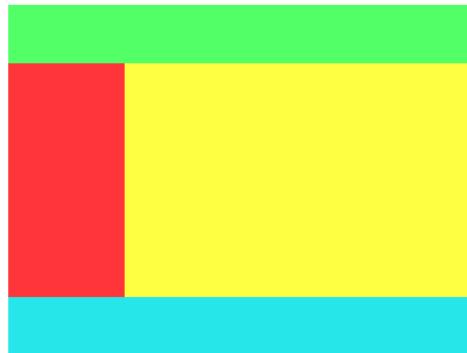
Ako bismo ovim elementima postavili vrednost `inline` umesto `inline-block` oni bi se potpuno ponašali kao linijski elementi, odnosno zauzimali bi samo onoliko prostora koliko im je zaista potrebno.

**Skrivanje elemenata** postiže se postavljanjem vrednosti `none` za svojstvo `display`. Ukoliko element ima podešeno ovo svojstvo on neće biti prikazan na stranici, ali neće ni biti prikazana praznina tamo gde bi se on inače našao, odnosno ne utiče na raspored elemenata koji su vidljivi.

Ako želimo da skriveni element utiče na raspored tj. da se pojavi praznina na njegovom mestu, umesto svojstva `display` možemo iskoristiti svojstvo `visibility` sa vrednošću `hidden`.

**Zadatak 2.1:**

Napisati HTML datoteku koja odgovara stranici na narednoj slici



Napomena: iskoristiti svojstvo `position` za poozicioniranje elemenata.

**Rešenje:** Zadatak zahteva da koristimo svojstvo `position`, ali nije naglašeno koju vrednost treba iskoristiti.

Ukoliko koristimo relativno pozicioniranje potrebno je da žuti element pomerimo nagore za visinu i nalevo za širinu crvenog elementa kako bi ga pozicionirali na odgovarajuće mesto. Međutim, tu posao nije gotov. Pošto je iskorišćeno relativno pozicioniranje, ostaje praznina na normalnoj poziciji elementa, pa je potrebno pomeriti i plavi element nagore.

Kod 15: zadaci/004/relativno.html

```
<!DOCTYPE html>
<html>
    <head>
        <title> </title>
        <meta charset="UTF-8">
        <link rel="stylesheet" href="relativno.css">
    </head>

    <body>
        <div id="wrapper">
            <div id="header"> </div>
            <div id="nav"> </div>
            <div id="content"> </div>
            <div id="footer"></div>
            <!-- oprez: na dnu ostaje prazan prostor kao posledica koriscenja
                relativnog pozicioniranja -->
        </div>
    </body>
</html>
```

Kod 16: zadaci/004/relativno.css

```
#wrapper{
    margin-left: auto;
```

```
margin-right: auto;
width: 400px;
position: relative;
}

#header{
    width: 400px;
    height: 50px;
    background-color: rgb(83, 255, 103);
}

#nav{
    width: 100px;
    height: 200px;
    background-color: rgb(255, 53, 57);
}

#content{
    width: 300px;
    height: 200px;
    background-color: rgb(255, 255, 65);
    position: relative;
    top:-200px;
    left:100px;
}

#footer{
    width: 400px;
    height: 50px;
    background-color: rgb(39, 230, 233);
    position: relative;
    top: -200px;

    /* dodatno css svojstvo koje koristimo ako hocemo da obrisemo prazan
       prostor koji
       ostaje ispod sheme
    */
    /*
    margin-bottom: -200px;
    */
}
```

Drugi način bio bi korišćenje apsolutnog pozicioniranja. Prisetimo da se apsolutno pozicioniranje određuje u odnosu na prvog roditelja sa relativnim pozicioniranjem. Pošto su svi elementi u okviru jednog omotača, njegovo pozicioniranje postavljamo na relativno, ali ga nećemo pomerati. Ostaje još da žutom elementu postavimo pozicioniranje na apsolutno i u odnosu na omotač pomerimo od gornje i leve ivice za odgovarajuće vrednosti. U nastavku je data CSS datoteka sa izmenjenim pozicioniranjem, a HTML ostaje isti.

Kod 17: zadaci/004/apsolutno.css

```
#wrapper{
    margin-left: auto;
```

```
margin-right: auto;
width: 400px;
position: relative;
}

#header{
    width: 400px;
    height: 50px;
    background-color: rgb(83, 255, 103);
}

#nav{
    width: 100px;
    height: 200px;
    background-color: rgb(255, 53, 57);
}

#content{
    width: 300px;
    height: 200px;
    background-color: rgb(255, 255, 65);
    position: absolute;
    top:50px;
    left:100px;
}

#footer{
    width: 400px;
    height: 50px;
    background-color: rgb(39, 230, 233);
}
```

### 2.1.7 Svojstvo `float`

Svojstvo `float` određuje kako elementi treba da plutaju na stranici. Može imati neku od sledećih vrednosti:

- `left` - element pluta prema levoj strani elementa koji ga okružuje
- `right` - element pluta prema desnoj strani elementa koji ga okružuje
- `none` - element ne pluta (podrazumevana vrednost)
- `inherit` - element pluta na isti način kao i roditelj

### 2.1.8 Svojstvo `clear`

Pomoću svojstva `clear` određujemo koji elementi mogu da plutaju pored elementa sa ovim svojstvom i na kojoj strani. Može imati sledeća svojstva:

- `none` - svi elementi mogu da plutaju sa obe strane (podrazumevana vrednost)
- `left` - nisu dozvoljeni plutajući elementi sa leve strane

- `right` - nisu dozvoljeni plutajući elementi sa desne strane
- `both` - nisu dozvoljeni plutajući elementi na obe strane
- `inherit` - nasleđuje vrednost ovog svojstva od roditelja

Ovo svojstvo se najčešće koristi nakon što je svojstvo `float` iskorišćeno na elementu. Plutajući element ostaje da pluta na stranici, međutim, element sa postavljenim svojstvom `clear` pojaviće se ispod plutajućeg.

Ukoliko se na nivou elementa definise `clear` svojstvo, ono onemogućava preklapanje elementa sa elementima koji su pozicionirani pomoću float svojstva. Na primer, ako je na nivou elementa definisano svojstvo `clear:right`; ono onemogućava preklapanje elementa sa elementima koji imaju definisano `float:right`; svojstvo. Ovo je ilustrovano narednim primerom.

Kod 18: primeri/float/clear.html

```
<!DOCTYPE html>
<html>
    <head>
        <title> Clear svojstvo </title>

        <style type='text/css'>
            #d1{
                height: 200px;
                width: 200px;
                background-color: rgb(219, 56, 41);
                float:left;
            }

            #d2{
                height: 350px;
                width: 200px;
                background-color: #F2E422;
                float:left;
            }

            #d3{
                height: 300px;
                width: 200px;
                background-color: rgb(255, 172, 17);
                float:right;
            }

            #d4{
                /*TODO: posmatrati sta se dogadja kada se vrednosti postave na
                   none, left, right i both */
                clear: none;
                height: 150px;
                background-color: rgb(135, 180, 68);
            }
        </style>
    </head>
```

```
<body>
<div id='d1'></div>
<div id='d2'></div>
<div id='d3'></div>
<div id='d4'></div>
</body>
</html>
```

Ovakav kod daje nam sledeći izgled stranice



Kao što možemo videti, zeleni element preklopio se sa ostalim jer je svojstvo `clear` postavljeno na `none`. Ako ga izmenimo na `left`, zeleni element je pomeren ispod ostalih kako se ne bi preklapao sa njima



Isti rezultat bismo dobili i postavljanjem ovog svojstva na vrednost `both` jer je element koji pluta na levu stranu viši. Ako bismo ga izmenili na `right`, zeleni element pomeriće se ispod ostalih, ali sada dolazi do preklapanja sa žutim jer njegovo plutajuće svojstvo ima vrednost `left` i to preklapanje je dozvoljeno.



Raspored elemenata kao u prethodnom zadatku može se dobiti i kombinacijom svojstava `float` i `clear`. Rešenje je dato narednim kodom.

Kod 19: zadaci/004/float.css

```
#wrapper{
    width: 400px;
    margin: auto;
    border: 1px solid black;
}

#header{
    height: 50px;
    width: 400px;
    background-color: #ff9966;
}

#nav{
    float: left;
    height: 200px;
    width: 100px;
    background-color: #00cc99;
}

#content{
    float: left;
    height: 200px;
    width: 300px;
    background-color: #6666ff;
    border: 3px solid white;
    box-sizing: border-box;
}

#footer{
    clear: left;
    height: 50px;
    width: 400px;
    background-color: #993366;
}
```

### 2.1.9 Svojstvo `overflow`

S obzirom da pregledač izbacuje plutajuće elemente iz regularnog toka prikaza, imamo promene u izgledu njihovih roditeljskih elemenata. Može se desiti da element koji smo postavili da pluta u okviru nekog elementa bude veće visine od njega i tada će ispasti iz okvira tog elementa, kao u primeru



Kao rešenje ovog problema može se dodati svojstvo `float` i roditeljskom elementu. To ima smisla ako želimo da elementi budu poravnati sa levom ili desnom ivicom. Drugi način bi bio da dodamo još jedan element koji će imati postavljeno `clear` svojstvo. Treći način je korišćenje svojstva `overflow` sa vrednošću `auto`. Sva tri načina za rezultat imaju sledeći prikaz



Ovaj prikaz postižemo narednim kodom

Kod 20: `primeri/float/parent.html`

```
<html>
  <head>
    <title> Float layout </title>
    <style type='text/css'>

      .wrapper{
        width: 900px;
        border: 10px solid #031028;
        /*float: left;*/
        /*overflow: auto; */
```

```
}

#left_part{
    width: 300px;
    height: 300px;
    background-color: #02564A;
    float: left;
}

#right_part{
    width: 600px;
    height: 300px;
    background-color: #238965;
    float: left;

}

</style>
</head>

<body>
    <div class='wrapper'>
        <div id='left_part'> </div>
        <div id='right_part'> </div>
        <!--
        <div style='clear:both'> </div>
        -->
    </div>
</body>
</html>
```

## 3 Čas 5

Cilj ovog časa je upoznavanje studenata sa upravljanjem pozadinom elemenata i radom sa slikama.

### 3.1 Upravljanje pozadinom elemenata

Kao što videli na prethodnim časovima, elementima možemo pridružiti jednobojnu pozadinu korišćenjem svojstva `background-color`, uz odgovarajuću vrednost boje u nekom od opisanih modela boja ili navođenjem imena boje. Ipak, postoje i naprednije tehnike za postavljanje pozadine sa kojima ćemo se upoznati u nastavku.

#### 3.1.1 Slika kao pozadina

Ukoliko nam je na raspolaaganju neka slika, možemo je postaviti kao pozadinu za proizvoljni element korišćenjem svojstva `background-image`. Vrednost ovog svojstva je putanja ka slici, na primer:

Kod 21: primeri/pozadina/background\_image.html

```
<!DOCTYPE html>
<html>

    <head>
        <title> Slike u pozadini elementa </title>

        <style type='text/css'>
            #test_div {
                height: 600px;
                width: 900px;

                background-image: url('halloween.jpeg');
            }

            #test_div p {
                color: white;
                text-align: center;
                font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
                font-size: 50px;
                margin: 0px;
                padding-top: 1cm;
            }
        </style>
    </head>

    <body>
        <div id='test_div'>
            <p>Ready for Halloween?</p>
        </div>
    </body>
</html>
```



**Ponavljanje pozadine** Podrazumevano, pozadina se ponavlja horizontalno (tj. po x-osi) i vertikalno (tj. po y-osi), što je najjednostavnije videti ukoliko su dimenzije unutrašnjosti elementa veće od dimenzije slike koja se koristi kao pozadina. CSS svojstvo `background-repeat` se koristi za kontrolisanje ponavljanja. Njegove vrednosti su:

- `repeat` — pozadina se ponavlja horizontalno i vertikalno (podrazumevano)
- `repeat-x` — pozadina se ponavlja samo horizontalno
- `repeat-y` — pozadina se ponavlja samo vertikalno
- `no-repeat` — pozadina se ne ponavlja

Ovim CSS svojstvom možemo izbeći neobična ponašanja, kada slika nije namenjena da se ponavlja po nekoj od osa (ili da se ponavlja uopšte). Na primer, naredni kod i prateći prikaz na slici ilustruju jedno takvo ponašanje:

Kod 22: primeri/pozadina/background\_repeat-default.html

```
<!DOCTYPE html>
<html>
<head>
    <style>
        body {
            background-image: url("gradient_bg.png");
        }
    </style>
</head>
<body>
    <h1>background-repeat: repeat; /* Podrazumevano */</h1>
</body>
</html>
```

**background-repeat: repeat; /\* Podrazumevano \*/**

Ovo se može ispraviti postavljanjem odgovarajuće vrednosti za svojstvo `background-repeat`, kao u narednom primeru:

Kod 23: primeri/pozadina/background\_repeat-repeatx.html

```
<!DOCTYPE html>
<html>
<head>
    <style>
        body {
            background-image: url("gradient_bg.png");
            background-repeat: repeat-x;
        }
    </style>
</head>
<body>
    <h1>background-repeat: repeat-x;</h1>
</body>
</html>
```

**background-repeat: repeat-x;**

**Pozicija pozadine** Podrazumevano, slika za pozadinu je postavljena uz gornji levi ugao elementa. Ipak, ovo podešavanje se može izmeniti korišćenjem svojstva `background-position`. On uzima dve vrednosti, jednu za horizontalnu, a drugu za vertikalnu poziciju. Ove vrednosti se mogu navesti na različite načine:

- Za prvu vrednost možemo navesti `left`, `right` ili `center`, dok za drugu vrednost možemo navesti `top`, `center`, `bottom`. Dakle, na ovaj način možemo zadati ukupno 9 vrednosti.
- Procenti — gornji levi ugao je `0% 0%`. Donji desni ugao je `100% 100%`.
- Dužine — gornji levi ugao je `0 0`. Kako vrednosti rastu, to se pozadina pomera udesno (po x-osi) i nadole (po y-osi).

Dajemo naredni primer (na prikazu je naknadno dodata ivica radi prikaza ivica prozora veb pregledača):

Kod 24: primeri/pozadina/background\_position.html

```
<!DOCTYPE html>
<html>
<head>
    <style>
        body {
            background-image: url("halloween_card.jpg");
            background-repeat: repeat-y;

            background-position: right top;
        }
    </style>
</head>
<body>
    <h1>Happy halloween!</h1>
    <p>
        From: _____ <br>
        To: _____
    </p>
</body>
</html>
```



**Veličina pozadine** Podrazumevano, veličina pozadine će biti postavljena na dimenzije slike koja se koristi za pozadinu. Ovo može predstavljati problem ukoliko je slika velikih dimenzija, a element manjih dimenzija. Kontrola veličine pozadine se ostvaruje pomoću svojstva `background-size`. Njegove vrednosti mogu biti:

- `auto` — podrazumevano ponašanje.
- dužina — navode se dve vrednosti: prva za širinu pozadine, a druga za visinu pozadine.
- procenat — navode se dve vrednosti: prva za širinu pozadine, a druga za visinu pozadine. Ove vrednosti se posmatraju u odnosu na roditeljski element.
- `cover` — vrši se automatsko skaliranje slike tako da prekrije ceo element. U ovom slučaju može doći do "razvlačenja" slike ili odsecanja po nekoj ivici.
- `contain` — vrši se automatsko skaliranje slike tako da bude cela vidljiva.

Naredni primer ilustruje kako se menja ponašanje kada se menjaju opisane vrednosti za svojstvo `background-size`:

Kod 25: `primeri/pozadina/background_size.html`

```
<!DOCTYPE html>
<html>
<head>
<style>
div {
    background-image: url("halloween_cartoon.jpeg");
    background-repeat: no-repeat;
    width: 40%;
    height: 200px;
    border: 1px solid black;
    position: absolute;
}

#halloween1 {
    top: 0px;
    left: 0%;
    background-size: auto;
}
#halloween2 {
    top: 0px;
    left: 50%;
    background-size: 350px 150px;
}
#halloween3 {
    top: 250px;
    left: 0%;
    background-size: 90% 90%;
}
#halloween4 {
    top: 250px;
    left: 50%;
    background-size: cover;
}
```

```

        }
        #halloween5 {
            top: 500px;
            left: 0%;
            background-size: contain;
        }
    </style>
</head>
<body>
    <div id="halloween1"></div>
    <div id="halloween2"></div>
    <div id="halloween3"></div>
    <div id="halloween4"></div>
    <div id="halloween5"></div>
</body>
</html>

```



### 3.1.2 Gradijentna pozadina

CSS gradijenti omogućavaju prikazivanje ravnomerne tranzicije između dve boje ili više boja. U jeziku CSS, definisane su dve vrste gradijenata:

- linearni gradijent
- radijalni gradijent

Gradijenti se postavljaju korišćenjem svojstva `background-image`. U zavisnosti od toga koja se CSS funkcija koristi, biće postavljen linearni ili radijalni gradijent.

**Linearni gradijent** Funkcija koja se koristi za definisanje linearnog gradijenta ima narednu definiciju:

```
linear-gradient(direction, color-stop1, color-stop2, ...);
```

Da bismo kreirali linearni gradijent, potrebno je da specifikujemo:

- Nagib prave u čijem pravcu se gradijent prostire (`direction`), korišćenjem:
  - ključnih reči oblika `to right`, `to bottom right`, i sl.

- navođenjem ugla nagiba, na primer, `60deg`
- Boje na osnovu kojih će se gradijent napraviti. Možemo navesti proizvoljan broj boja, odvojenih karakterom `,`, u bilo kom modelu boja (`color-stop1`, `color-stop2`, ...).

Ukoliko želimo da preciznije definišemo prelaska iz jedne boje u drugu, možemo nakon zadavanja boje dodati procenat kojim govorimo do koje tačke u gradijentu treba ta boja da se pronađe. Na primer, ukoliko stavimo

```
background-image: linear-gradient(to right, black 75%, white);
```

rezultat će biti takav da će u prvih 75% gradijenta vladati crna boja, a zatim će se do kraja vršiti prelaz do bele boje. Ovim se mogu postići neki zanimljivi efekti (na primer, pojavljivanje štafti, što naredni primer ilustruje):

Kod 26: `primeri/pozadina/linear_gradient.html`

```
<!DOCTYPE html>
<html>
    <head>
        <title>Gradijenti</title>

        <style type='text/css'>
            div#gradijent{
                height: 200px;
                width: 1000px;
                background-image: linear-gradient(to bottom, #4FB2B2, #FFA3E7,
                    #8AFFFF);
            }

            div#strafte{
                height: 200px;
                width: 1000px;
                background-image: linear-gradient(to right, #4FB2B2 0%, #4
                    FB2B2 25%, #FFA3E7 25%, #FFA3E7 50%, #8AFFFF 50%, #8AFFFF
                    100%);
                margin-top: 50px;
            }
        </style>
    </head>

    <body>
        <div id='gradijent'>
        </div>

        <div id='strafte'>
        </div>
    </body>
</html>
```



**Radijalni gradijent** Funkcija koja se koristi za definisanje radijalnog gradijenta ima narednu definiciju:

```
radial-gradient(shape size at position, start-color, ..., last-color);
```

Da bismo kreirali linearни gradijent, potrebno je da specifikujemo:

- Oblik (`shape`), čije vrednosti mogu biti:
  - `ellipse` (podrazumevana vrednost)
  - `circle`
- Veličinu (`size`), čije vrednosti mogu biti:
  - `farthest-corner` (podrazumevana vrednost)
  - `closest-side`
  - `closest-corner`
  - `farthest-side`
- Poziciju (`position`), čije vrednosti mogu biti:
  - ključne reči oblika `center`, `top right`, i sl.
  - Dužine (po dve vrednosti za horizontalnu i vertikalnu poziciju)
  - Procenti (po dve vrednosti za horizontalnu i vertikalnu poziciju)
- Boje na osnovu kojih će se gradijent napraviti. Možemo navesti proizvoljan broj boja, odvojenih karakterom `,` u bilo kom modelu boja (`start-color`, ..., `last-color`).

Naredni primer ilustruje korišćenje nekih od opisanih vrednosti:

Kod 27: `primeri/pozadina/radial_gradient.html`

```
<!DOCTYPE html>
<html>
  <head>
```

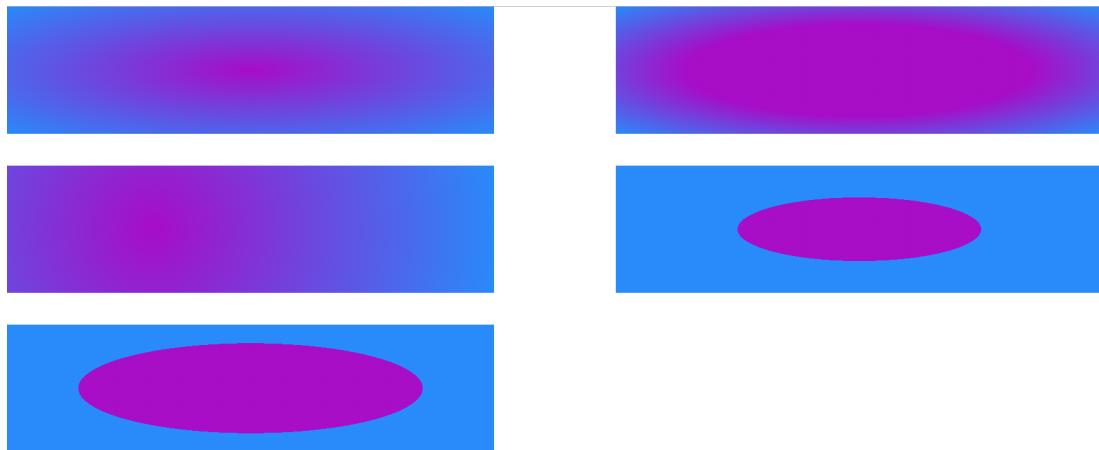
```
<title>Gradijenti</title>

<style type='text/css'>
    div {
        position: absolute;
        height: 200px;
        width: 40%;
    }

    div#radijalni-ravnomerni{
        top: 0px;
        left: 0%;
        background-image: radial-gradient(rgb(168, 13, 199),
                                           rgb(41, 139, 250));
    }
    div#radijalni-neravnomerni{
        top: 0px;
        left: 50%;
        background-image: radial-gradient(rgb(168, 13, 199) 50%,
                                           rgb(41, 139, 250));
    }
    div#radijalni-krug{
        top: 250px;
        left: 0%;
        background-image: radial-gradient(circle at 30% 50%,
                                           rgb(168, 13, 199), rgb(41, 139, 250));
    }
    div#radijalni-size1{
        top: 250px;
        left: 50%;
        background-image: radial-gradient(closest-side,
                                           rgb(168, 13, 199) 0%, rgb(168, 13, 199) 50%,
                                           rgb(41, 139, 250) 50%, rgb(41, 139, 250) 100%);
    }
    div#radijalni-size2{
        top: 500px;
        left: 0%;
        background-image: radial-gradient(farthest-corner,
                                           rgb(168, 13, 199) 0%, rgb(168, 13, 199) 50%,
                                           rgb(41, 139, 250) 50%, rgb(41, 139, 250) 100%);
    }
</style>
</head>

<body>
    <div id='radijalni-ravnomerni'>
    </div>
    <div id='radijalni-neravnomerni'>
    </div>
    <div id='radijalni-krug'>
    </div>
    <div id='radijalni-size1'>
    </div>
```

```
<div id='radijalni-size2'>
</div>
</body>
</html>
```



### 3.2 Rad sa slikama

Slike često predstavljaju nezaobilazni element stranica na vebu. Najosnovniji način za uključivanje slika u HTML dokumenta je pomoću samozatvarajućeg elementa `img`. Kako bismo naveli koji resurs želimo da uključimo kao sliku, potrebno je da zadamo lokaciju resursa kao vrednost atributa `src`. Na primer,

Kod 28: primeri/rad\_sa\_slikama/img.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Rad sa slicicama</title>
</head>
<body>
    
</body>
</html>
```



U prethodnom primeru vidimo da je upotrebljen i atribut `alt`. Njegova uloga je da prikaže alternativni tekst, ukoliko iz bilo kog razloga klijent nije u stanju da prikaže željeni resurs (na primer, ako ne može da mu pristupi, najčešće zato što resurs ne postoji na traženoj lokaciji ili ako korisnik koristi "čitače ekrana"). Korišćenje ovog atributa se smatra dobrom praksom, te ćemo ga mi koristiti. Osim što predstavlja dobru praksu, HTML standard zahteva korišćenje ovog atributa i u suprotnom, stranica neće biti validirana.

Veličina slike se može podesiti atributima `width` i `height`, čije su vrednosti celi brojevi koji predstavljaju veličine u pikselima. Naravno, dimenzija elementa `img` se može podesiti i u jeziku CSS, zadavanjem svojstava `width` i `height`, čije veličine mogu biti bilo koja od dužina.

Ipak, smatra se poželjnim navođenje vrednosti ovih atributa, da bi veb pregledač pri konstrukciji izgleda stranice, znao koliko prostora mu je potrebno da odvoji za ovaj element, čak i pre nego što se slika učita (pogotovo ako je veličina resursa velika ili se učitava na sporoj konekciji).

Naravno, ukoliko želimo da se zaštитimo od toga da neka druga CSS pravila (na primer, ona koja dolaze iz neke eksterne datoteke) utiču na prikaz slika u našem dokumentu, onda je poželjnije koristiti CSS svojstva `width` i `height` u okviru `style` atributa elementa `img`. Naredni primer ilustruje ovo ponašanje:

Kod 29: primeri/rad\_sa\_slikama/img\_override.html

```
<!DOCTYPE html>
<html>
<head>
    <link rel="stylesheet" type="text/css" href="img_override.css">
</head>
<body>

<h2>Stilizovanje slicica</h2>

<p>Slika ispod ima vrednost atributa width postavljenu na 128 piksela, ali
    je ta vrednost pregazena od strane eksterne CSS datoteke, koja postavlja
    CSS svojstvo width na 100%.</p>


<p>Slika ispod koristi style atribut, u kojem je CSS svojstvo width
    postavljeno na 128 piksela. S obzirom da ovakva upotreba ima najveći
    prioritet, vrednost CSS svojstva width od 100% je pregazena.</p>


</body>
</html>
```

Kod 30: primeri/rad\_sa\_slikama/img\_override.css

```
img {
    width: 100%;
}
```

### Stilizovanje slicica

Slika ispod ima vrednost atributa width postavljenu na 128 piksela, ali je ta vrednost pregazena od strane eksterne CSS datoteke, koja postavlja CSS svojstvo width na 100%.



Slika ispod koristi style atribut, u kojem je CSS svojstvo width postavljeno na 128 piksela. S obzirom da ovakva upotreba ima najveći prioritet, vrednost CSS svojstva width od 100% je pregazena.



**Figure i naslovi** Jedan zanimljiv blokovski element koji služi za anotaciju drugih elemenata, pre svega slika, ilustracija, kodova, ali i drugih elemenata, naziva se **figure**. U njemu možemo staviti bilo koji element koji predstavlja nekakav sadržaj, a njegov značaj se odgleda u podelementu **figcaption**, čija je uloga anotacija sadržaja. Naredni primer ilustruje upotrebu ovih elemenata:

Kod 31: primeri/rad\_sa\_slikama/figure.html

```
<!DOCTYPE html>

<html>
<head>
    <meta charset='UTF-8'>
    <title>Figure i naslovi</title>

    <style type='text/css'>
        figure {
            width: 500px;
            /* Centriranje figure u odnosu na roditelja (body) */
            margin: auto;
            border: 1px solid red;
        }

        figcaption {
            text-align: center;
        }
    </style>
</head>

<body>
    <figure>
        <img src='jesen.jpg' alt='Jesen' style="width: 500px; height: 375px;">
        <figcaption>
            <p>Slika 1: Jesen, šareno godišnje doba</p>
        </figcaption>
    </figure>
</body>
</html>
```



Slika 1: Jesen, šareno godišnje doba

Naravno, anotacija pomoću elementa **figcaption** je opcionalna. Dodatno, možemo staviti prvo sadržaj, pa zatim anotaciju (kao što je urađeno u primeru), a možemo i staviti prvo anotaciju, pa zatim sadržaj.

## 4 Čas 6

Cilj ovog časa je upoznavanje studenata sa radom sa tabelama, zatim korišćenjem lista, povezivanjem stranica i pseudoklasama koje se mogu dodeliti elementima.

### 4.1 Rad sa tabelama

Tabele predstavljaju najjednostavniji način za prikazivanje podataka u više dimenzija. Tabele imaju redove i kolone u čijim presecima se nalaze ćelije. U HTML jeziku, tabele posmatramo kao redove ćelija, što će se prirodno preslikati u odgovarajuće elemente.

Osnovni element od kojeg polazimo kada formiramo tabelu jeste `table`. Njegov sadržaj čine, kao što smo rekli, redovi, koji se preslikavaju u elemente `tr` (skr. *table row*). Redovi se sastoje od ćelija, koje mogu biti predstavljene elementima:

- `th` (skr. *table header*) — koriste se za opis zaglavlja, ukoliko tabela ima zaglavlj. Podrazumevano, zaglavla su podebljana i centrirana.
- `td` (skr. *table data*) — koriste se za prikaz jedne ćelije, u kojoj se nalazi neki podatak (tekst, slika, lista, druga tabela, i dr.).

Pored logičkog smeštanja podataka u tabelu i najjednostavnijeg tabelarnog prikaza podataka, tabele nemaju posebne prikaze. Ukoliko želimo da postavimo ivicu za celu tabelu, red ili ćeliju, moramo da podesimo CSS svojstvo `border` (ili njemu slična svojstva). Naredni primer ilustruje tabelu bez posebnih stilova, a zatim dodavanje jednostavne ivice za celu tabelu i svaku pojedinačnu ćeliju:

Kod 32: primeri/tabele/table.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Jednostavne tabele</title>

    <style type="text/css">
        div table, div th, div td {
            border: 1px solid black;
        }

        div {
            margin-top: 50px;
        }
    </style>
</head>
<body>
    <table>
        <tr>
            <th>Identifikator</th>
            <th>Ime studenta</th>
        </tr>
        <tr>
            <td>1/2076</td>
```

```

        <td>Anja</td>
    </tr>
    <tr>
        <td>2/2076</td>
        <td>Nikola</td>
    </tr>
</table>

<div>
    <table>
        <tr>
            <th>Identifikator</th>
            <th>Ime studenta</th>
        </tr>
        <tr>
            <td>1/2076</td>
            <td>Anja</td>
        </tr>
        <tr>
            <td>2/2076</td>
            <td>Nikola</td>
        </tr>
    </table>
</div>
</body>
</html>
```

**Identifikator Ime studenta**

Identifikator	Ime studenta
1/2076	Anja
2/2076	Nikola

Identifikator	Ime studenta
1/2076	Anja
2/2076	Nikola

Ukoliko želimo da svaka čelija ima ivicu, ali ne da budu zasebne, već da dve susedne čelije dele ivicu, to možemo uraditi postavljanjem CSS svojstva `border-collapse` na vrednosti `collapse`, kao u narednom primeru:

Kod 33: primeri/tabele/table\_collapsed.html

```

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Jednostavne tabele</title>
```

```

<style type="text/css">
    table, th, td {
        border: 1px solid black;
        border-collapse: collapse;
    }
</style>
</head>
<body>
    <table>
        <tr>
            <th>Identifikator</th>
            <th>Ime studenta</th>
        </tr>
        <tr>
            <td>1/2076</td>
            <td>Anja</td>
        </tr>
        <tr>
            <td>2/2076</td>
            <td>Nikola</td>
        </tr>
    </table>
</body>
</html>

```

Identifikator	Ime studenta
1/2076	Anja
2/2076	Nikola

#### 4.1.1 Razna podešavanja prikaza tabela

**Anotiranje tabela** Poput elementa `figure`, i tabele mogu biti anotirane. Za anotiranje tabela koristi se element `caption`. Za razliku od elementa `figcaption` koji se može pojaviti bilo ispred ili nakon glavnog sadržaja, element `caption` mora biti prvi naslednik elementa `table`. Sama pozicija anotacija se postavlja CSS svojstvom `caption-side`, čije su vrednosti `top` ili `bottom`.

**Razmak između celija i prikazivanje praznih celija** Ukoliko želimo da definisemo globalni razmak između celija, možemo nad elementom `table` podesiti CSS svojstvo `border-spacing` čija je vrednost neka od dužina.

Ukoliko tabela sadrži prazne celije, možemo izbeći njihovo prikazivanje podešavanjem CSS svojstva `empty-cells`. Njegove vrednosti su: `hide` ili `show`.

**Dimenzije redova i celija** Elementu `tr` možemo pridružiti svojstvo `height`, dok elementima `td` i `th` možemo pridružiti svojstva `height` i `width`, da bismo im po-

desili željenu dimenziju. Ono na šta treba obratiti pažnju jeste da se visina vrste prilagođava visini najviše celije, dok se širina kolone prilagođava širini najduže celije.

Naredni primer ilustruje upotrebu anotacije tabele, razmaka između celija, prikazivanje praznih celija i dimenzije redova i celija:

Kod 34: primeri/tabele/table\_misc.html

```
<!DOCTYPE html >

<html>
    <head>
        <title>Tabelice – razna podešavanja</title>
        <meta charset="UTF-8">
        <style type="text/css">
            caption {
                caption-side: bottom;
                text-align: center;
            }

            table {
                margin: auto;
                border: 5px solid violet;
                border-collapse: separate;
                border-spacing: 10px;
                empty-cells: hide;
            }

            tr {
                height: 50px;
            }

            td {
                padding: 5px;
                border: 1px solid slateblue;
                width: 150px;
            }

            /*
                Visina vrste se prilagodjava visini najveće celije.
                Sirina kolone se prilagodjava sirini najduže celije.
            */
            td#druga_celija {
                background-color: cyan;
                vertical-align: middle;
                text-align: center;
                width: 300px;
                height: 100px;
            }
        </style>
    </head>
    <body>
        <table>
            <caption>Ovo je naslov tabele</caption>
```

```
<tr>
    <td>1</td>
    <td id="druga_cevija">2</td>
    <td></td>
</tr>
<tr>
    <td>4</td>
    <td>5</td>
    <td>6</td>
</tr>
</table>
</body>
</html>
```

1	2	
4	5	6

Ovo je naslov tabele

#### 4.1.2 Spajanje kolona i redova

Ukoliko želimo, možemo spajati više celija u jednu, i to korišćenjem sledećih atributa nad elementom celije:

- `rowspan` — vrednost ovog atributa je broj redova kroz koji će se celija prostirati
- `colspan` — vrednost ovog atributa je broj kolona kroz koji će se celija prostirati

Naredni primer ilustruje upotrebu ovih atributa:

Kod 35: primeri/tabele/table\_span.html

```
<!DOCTYPE html>

<html>
<head>
    <title>Tabelice – spajanje vrsta i kolona</title>
    <meta charset="UTF-8" />
    <style type="text/css">
        table.spojena{
            margin: auto;
            border: 3px dashed rgb(155, 29, 214);
            text-align: center;
        }
    </style>
</head>
<body>
    <table border="1" class="spojena">
        <tr>
            <td>1</td>
            <td>2</td>
            <td>3</td>
        </tr>
        <tr>
            <td>4</td>
            <td>5</td>
            <td>6</td>
        </tr>
        <tr>
            <td>7</td>
            <td>8</td>
            <td>9</td>
        </tr>
        <tr>
            <td>10</td>
            <td>11</td>
            <td>12</td>
        </tr>
        <tr>
            <td>13</td>
            <td>14</td>
            <td>15</td>
        </tr>
        <tr>
            <td>16</td>
            <td>17</td>
            <td>18</td>
        </tr>
        <tr>
            <td>19</td>
            <td>20</td>
            <td>21</td>
        </tr>
        <tr>
            <td>22</td>
            <td>23</td>
            <td>24</td>
        </tr>
        <tr>
            <td>25</td>
            <td>26</td>
            <td>27</td>
        </tr>
        <tr>
            <td>28</td>
            <td>29</td>
            <td>30</td>
        </tr>
    </table>
</body>
</html>
```

```

1, 4</td>
        <td colspan="2">2, 3</td>
    </tr>
    <tr>
        <td>5</td>
        <td>6</td>
    </tr>
</table>
</body>
</html>

```

The diagram shows a 2x3 grid of cells. The first column contains two cells: the top one is labeled "1, 4" and the bottom one is labeled "5". The second column contains three cells: the top one is labeled "2, 3" and the bottom two are labeled "6". The third column contains one cell labeled "6". A dashed purple border surrounds the entire grid.

#### 4.1.3 Stilizovanje redova tabela

Pomoću selektora `nth-child(n)` možemo da definišemo kako svako  $n$ -to dete treba da izgleda. U slučaju tabele, definišemo kako će  $n$ -ta vrsta izgledati. U zagradama navodimo vrednost za  $n$ , koja može biti neki broj. Dodatno, ako želimo da definišemo kako će izgledati parne i neparne vrste, možemo iskoristiti vrednosti `even` i `odd`, kao u narednom primeru

Kod 36: primeri/tabele/table\_stilovi.html

```

<!DOCTYPE html>
<html>
<head>
    <title>Tabelice – stilovi</title>
    <meta charset="UTF-8">
    <style>

```

```
table {
    width: 50%;
}
table, th, td {
    border: 1px solid black;
    border-collapse: collapse;
}
th, td {
    padding: 15px;
    text-align: left;
}
table#t01 tr:nth-child(even) {
    background-color: rgba(92, 236, 255, 0.4);
}
table#t01 tr:nth-child(odd) {
    background-color: white;
}
table#t01 th {
    background-color: rgb(63, 107, 226);
    color: white;
}
</style>
</head>
<body>
    <h2>Hari Poter univerzum</h2>

    <table id="t01">
        <tr>
            <th>Ime</th>
            <th>Prezime</th>
            <th>Godine</th>
        </tr>
        <tr>
            <td>Hermiona</td>
            <td>Grejndzer</td>
            <td>17</td>
        </tr>
        <tr>
            <td>Oliver</td>
            <td>Wood</td>
            <td>20</td>
        </tr>
        <tr>
            <td>Pomona</td>
            <td>Spraut</td>
            <td>55</td>
        </tr>
    </table>
</body>
</html>
```

## Hari Potter univerzum

Ime	Prezime	Godine
Hermiona	Grejndzer	17
Oliver	Wood	20
Pomona	Spraut	55

## 4.2 Liste

Liste su blok elementi koji se koriste prilikom nabranjanja. Razlikujemo nenumerisane, numerisane i definicione liste.

### 4.2.1 Nenumerisane liste

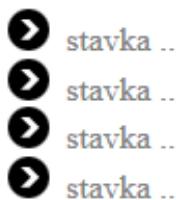
Nenumerisane ili neuređene liste obeležavamo elementom `ul`. Svaka stavka liste treba da se nađe između otvarajućeg i zatvarajućeg taga ovog elementa, a jednu stavku obeležavamo elementom `li`. Ovaj element može, i ne mora, da ima zatvarajući tag. Stvake mogu biti obeležene različitim znacima. Pomoću CSS svojstva `list-style-type` možemo odrediti koji znak će biti prikazan. Vrednosti za ovo svojstvo su:

- `disc` - popunjjen krug (podrazumevana vrednost)
- `square` - popunjjen kvadrat
- `circle` - prazan krug
- `none` - nema prikaza znaka

Ako bismo postavili svojstvo na vrednost `square` dobili bismo ovakvu listu

- stavka ..
- stavka ..
- stavka ..
- stavka ..

Pored toga, možemo koristiti i slike za obeležavanje stavki. Za to koristimo svojstvo `list-style-image` koji za vrednost treba da dobije putanju do slike koju koristimo, a navodimo je pomoću `url("putanja/do/slike")`. Evo jednog primera



Naredni kod daje nam stranicu sa prethodne dve liste.

Kod 37: primeri/liste/nenumerisana.html

```
<!DOCTYPE html >

<html>
  <head>
    <title> :: Nenumerisane liste :: </title>
    <meta charset='UTF-8' />
  </head>

  <body>
    <ul style="list-style-type: square;">
      <li> stavka .. </li>
      <li> stavka .. </li>
      <li> stavka .. </li>
      <li> stavka .. </li>
    </ul>

    <!-- pocetni simbol moze biti i slika -->
    <ul style="list-style-image: url('arrow.png'); color: gray;">
      <li> stavka .. </li>
      <li> stavka .. </li>
      <li> stavka .. </li>
      <li> stavka .. </li>
    </ul>
  </body>
</html>
```

#### 4.2.2 Numerisane liste

Numerisane ili uređene liste obeležavamo elementom `ol`. Za obeležavanje stavki takođe koristimo element `li`, kao i kod nenumerisane liste. Stvake možemo numerisati različitim simbolima, korišćenjem svojstva `list-style-type` koje za numerisane liste može imati neku od sledećih vrednosti:

- `decimal` - dekadni brojevi
- `lower-alpha` - malo slovo abecede
- `upper-alpha` - veliko slovo abecede

- `lower-roman` - mali rimski brojevi
- `upper-roman` - veliki rimski brojevi

Na primer, ako postavimo vrednost ovog svojstva na `lower-alpha` dobićemo ovakvu listu

- a. Prva stavka...
- b. Druga stavka...
- c. Treća stavka...
- d. Četvrta stavka...

Ako je izmenimo na `decimal` dobićemo

- 1. Prva stavka
- 2. Druga stavka
- 3. Treća stavka
- 4. Četvrta stavka

Naredni kod daje nam stranicu sa prethodne dve liste.

Kod 38: primeri/liste/numerisana.html

```
<!DOCTYPE html >

<html>
    <head>
        <title> :: Numerisane liste :: </title>
        <meta charset='UTF-8' />
    </head>

    <body>
        <ol style="list-style-type: lower-alpha;">
            <li> Prva stavka... </li>
            <li> Druga stavka... </li>
            <li> Treća stavka... </li>
            <li> Četvrta stavka... </li>
        </ol>

        <ol style="list-style-type: decimal; color:#7AC5CD;">
            <li> Prva stavka </li>
            <li> Druga stavka </li>
            <li> Treća stavka </li>
            <li> Četvrta stavka </li>
        </ol>
    </body>
</html>
```

I za ove liste možemo koristiti slike za obeležavanje stavki, s tim da tada gubimo svojstvo uređenosti jer će svaka stavka biti obeležena istom slikom.

**Pozicioniranje stavki** Stavke liste, numerisane ili nenumerisane, možemo pozicionirati tako da bude izvan ili unutar okvira predviđenog za tu stavku. Svojstvo `list-style-position` može imati vrednosti `outside` (podrazumevano) i `inside`. Sledeći primer ilustruje primenu ovog svojstva

#### List-style-position: outside

1. Prva stavka...
2. Druga stavka...
3. Treća stavka...
4. Četvrta stavka...

#### List-style-position: inside

1. Prva stavka...
2. Druga stavka...
3. Treća stavka...
4. Četvrta stavka...

Kod 39: primeri/liste/pozicioniranje.html

```
<!DOCTYPE html >

<html>
    <head>
        <title> :: Pozicioniranje stavki :: </title>
        <meta charset='UTF-8' />
        <style type="text/css">

            ol.outside li{
                border: 1px solid #66CD00;
                width: 40%;
                list-style-position: outside;
            }

            ol.inside li {
                border: 1px solid #66CD00;
                width: 40%;
                list-style-position: inside;
            }
        </style>
    </head>

    <body>
        <!-- vodeći simbol liste može biti izvan ili unutar box-a
        predviđenog za stavku liste -->
        <h3> List-style-position: outside </h3>
        <ol class="outside">
            <li> Prva stavka... </li>
            <li> Druga stavka... </li>
        
```

```
<li> Treća stavka... </li>
<li> Četvrta stavka... </li>
</ol>

<h3> List-style-position: inside </h3>
<ol class="inside">
    <li> Prva stavka... </li>
    <li> Druga stavka... </li>
    <li> Treća stavka... </li>
    <li> Četvrta stavka... </li>
</ol>
</body>
</html>
```

#### 4.2.3 Stilizovanje liste

Selektor `nth-child(n)` možemo primeniti i na liste:

stavka...  
stavka....  
stavka....  
stavka....  
stavka....

Kod 40: primeri/liste/nth-child.html

```
<!DOCTYPE html >

<html>
    <head>
        <title> :: Selektor nth-child :: </title>
        <meta charset='UTF-8' />
        <style type="text/css">

            ul.zebra{
                list-style-type:none;
                padding-left: 0px;
            }

            ul.zebra li:nth-child(even){
                color: black;
            }

            ul.zebra li:nth-child(odd){
                color: white;
                background-color: gray;
            }

        </style>
    </head>
```

```

<body>
  <ul class='zebra'>
    <li> stavka... </li>
    <li> stavka....</li>
    <li> stavka....</li>
    <li> stavka....</li>
    <li> stavka....</li>
  </ul>
</body>
</html>

```

**Skraćena forma pridruživanja stila** Stil liste možemo skraćeno zapisati korišćenjem svojstva `list-style` i navođenjem vrednosti za tip, poziciju i sliku.

#### 4.2.4 Ugnježdene liste

Nekada nam je potrebno da imamo nabranje unutar nabranja, odnosno dodatne stavke koje se odnose na jednu stavku. Možemo ugnezdati dve ili više numerisanih, nenumerisanih ili nekih njihovih kombinacija. U narednom primeru imamo nenumerisanu listu ugnježdenu u numerisanu:

1. proleće
  -  mart
  -  april
  -  maj
2. leto
  -  jun
  -  jul
  -  avgust

Kod 41: primeri/liste/ugnjezdene.html

```

<!DOCTYPE html >

<html>
  <head>
    <title> :: Ugnježdene liste :: </title>
    <meta charset='UTF-8' />
  </head>

  <body>
    <ol>
      <li> <span style="color:green"> proleće </span>
        <ul style="list-style-image: url('flower.png')">
          <li> mart </li>
          <li> april </li>
          <li> maj </li>
        </ul>
      </li>
    </ol>
  </body>
</html>

```

```
<li> <span style="color:orange">leto </span>
    <ul style="list-style-image: url('sun.png')">
        <li> jun </li>
        <li> jul </li>
        <li> avgust </li>
    </ul>
</li>
</ol>
</body>
</html>
```

#### 4.2.5 Definicione liste

Definicione liste obeležavaju se elementom `dl`. Ove liste predstavljaju listu pojmove, koje obeležavamo elementom `dt`, i njihovih opisa, obeleženih elementom `dd`. Svaki od ovih elemenata ima otvarajući i zatvarajući tag. Evo jednog primera ovakve liste

HTML

HyperText Markup Language  
Jezik za obelezavanje hiperteksta

CSS

Cascade Style Sheets

Kod 42: primeri/liste/definiciona.html

```
<!DOCTYPE html >

<html>
    <head>
        <title> :: Definicione liste :: </title>
        <meta charset='UTF-8' />
    </head>

    <body>
        <dl>
            <dt title="pojam koji se definise"> HTML </dt>
            <dd title="objasnjenja za pojam koji se definise">     HyperText
                Markup Language
            </dd>
            <dd title="objasnjenja za pojam koji se definise">     Jezik za
                obelezavanje hiperteksta
            </dd>

            <dt> <abbr title='Cascade Style Sheets'> CSS </abbr> </dt>
            <dd> Cascade Style Sheets </dd>
        </dl>
    </body>
</html>
```

## 4.3 Veze

Veze su linijski elementi koji služe povezivanju resursa na vebu, mahom veb strana ili njihovih fragmenata. Za dodavanje veza na stranicu koristi se element `a`. Neki od atributa koje možemo dodati ovom elementu su:

- `href` - ovo je obavezan atribut koji za vrednost prima putanju do dokumenta do kog vodi veza
- `target` - određuje gde će biti otvoren dokument. Neke od vrednosti:
  - `_blank` - otvori dokument u novoj kartici
  - `_self` - otvori dokument u trenutnoj kartici (podrazumevana vrednost)
- `hreflang` - označava jezik koji je korišćen u dokumentu do kog ova veza vodi
- `type` - označava tip dokumenta u MIME formatu

**Spoljašnje veze** U spoljašnje veze spadaju veze koje vode do dokumenata koji se nalaze na drugom veb serveru. Možemo ih dodati na stranu korišćenjem elementa `a` sa atributom `href` postavljenim na apsolutnu putanju koja vodi do tog dokumenta.

Da bismo koristili relativne putanje, u okviru zaglavila dokumenta treba dodati element `base` sa atributom `href` postavljenim na vrednost u odnosu na koju su putanje relativne.

**Unutrašnje veze** U unutrašnje veze ubrajamo veze koje vode do dokumenata koji se nalaze na istom veb serveru. Dodajemo ih na isti način kao i spoljašnje, korišćenjem elementa `a` sa atributom `href` s tim da kod ovog tipa veze možemo koristiti i relativne putanje. Treba biti oprezan kada se radi sa relativnim putanjama ukoliko je postavljen element `base` jer se on odnosi na sve veze, a ne samo na spoljašnje!

Unutrašnje veze koriste se za povezivanje stranica u okviru jedne veb prezentacije, ali i za povezivanje delova stranica. Na primer, možemo na dnu strane da postavimo vezu sa vrhom stranice kako bismo pošteli korisnika skrolovanja nazad.

Da bismo to postigli, potrebno je da se na vrhu stranice nađe neki element kome ćemo dodeliti neki identifikator, npr. "vrh". Na dnu stranice, ili na nekom drugom pogodnom mestu, postavićemo vezu koja za vrednost atributa `href` ima identifikator elementa na vrhu. Ovakva veza nas vodi sa jednog na drugi deo iste stranice.

### 4.3.1 Pseudoklase

Veze možemo urediti tako da izgledaju različito u zavisnosti od akcije koja je preduzeta. To postižemo korišćenjem pseudoklasa koje određuju stanja veze. Ova stanja koristimo prilikom stilizovanja elemenata i dodajemo ih u formatu `selektor:pseudoklasa`. Postoje 4 stanja:

- `link` - definiše izgled veze koja vodi do dokumenta koji još uvek nije posećen. Sledeći CSS segment neposećenim vezama dodeljuje plavu boju teksta, a pomoću svojstva `text-decoration` možemo odrediti da li će tekst biti podvučen ili ne (podrazumevano je `underline`).

```
a:link{  
    color: blue;  
    text-decoration: none;  
}
```

[klikni ovde](#)

- **visited** - definiše izgled veze koja vodi do dokumenta koji je posećen. Naredni CSS segment menja boju veza kada su posećene na nijansu zelene.

```
a:visited{  
    color: mediumgreen;  
    text-decoration: none;  
}
```

[klikni ovde](#)

- **hover** - definiše izgled veze kada cursorom pređemo preko veze. Naredni CSS kod menja boju teksta i pozadine veze iznad koje je cursor, i menja se dekoracija na podvučeno. Dodatno, menja se izgled na ruku umesto strelice, što je podrazumevano ponašanje za ovo stanje. Da bismo sami kontrolisali izgled cursora koristimo svojstvo **cursor**<sup>6</sup>.

```
a:hover{  
    color: darkmagenta;  
    background-color: silver;  
    text-decoration: underline;  
}
```

[klikni ovde](#)

- **active** - definiše izgled veze dok je aktivna. Veza je aktivna od trenutka kad kliknemo na nju do trenutka kada je otvoren dokument.

```
a:active{  
    color: yellow;  
    background-color: silver;  
    font-size: 17px;  
}
```

[klikni ovde](#)

<sup>6</sup>Detaljnije o ovom svojstvu možete pročitati na [https://www.w3schools.com/cssref/pr\\_class\\_cursor.asp](https://www.w3schools.com/cssref/pr_class_cursor.asp)

Kako bi pseudoklase `hover` i `active` imali desjtva, `link` i `visited` moraju da se definišu pre `hover`, a `active` nakon `hover`. U nastavku dajemo HTML i CSS kod koji objedinjuju priču o vezama u jednu stranicu.

Kod 43: primeri/veze/veze.html

```
<!DOCTYPE html>

<html>

    <head>
        <title> :: veze :: </title>

        <meta charset='utf-8' >

        <link rel="stylesheet"
              href="veze.css"
              type="text/css" />

        <!-- <base href="http://poincare.matf.bg.ac.rs" /> -->

    </head>

    <body>
        <span id="vrh">
        </span>

        <h2> Spoljne veze </h2>
        <a class="spoljna_veza"
           href="http://www.matf.bg.ac.rs">
            Matematički fakultet
        </a>

        <h2> Relativne veze </h2>
        <a href="../liste/definiciona.html"> Prethodna strana </a>

        <br/>
        <h2> Veze sa propratnim atributima </h2>
        <a class="veza_sa_atributima"
           href="http://en.wikipedia.org/wiki/Hyperlink"
           hreflang="en"
           type='text/html'
           target='_blank'>
            pročitajte više na Vikipediji
        </a>

        <div class="sidro">
            <a href="#vrh"> skok na vrh tekuće strane </a>
        </div>
    </body>
</html>
```

Kod 44: primeri/veze/veze.css

```
body {
    height: 2000px;
    padding-left: 30px;
}

a{
    font-size: 18px;
}

a:link{
    color: blue;
    text-decoration: none;
}

a:visited{
    color: mediumseagreen;
    text-decoration: none;
}

a:hover{
    color: darkmagenta;
    background-color: silver;
    text-decoration: underline;
}

a:active{
    color: yellow;
    background-color: silver;
    font-size: 17px;
}

a.spoljna_veza:link{
    color: rgb(255, 0, 180);
}

a.spoljna_veza:visited{
    color: rgb(255, 0, 180);
}

a.spoljna_veza:hover{
    color: rgb(0, 225, 255);
}

a.spoljna_veza:active{
    color: black;
    text-decoration: overline;
}

a.veza_sa_atributima:link, a.veza_sa_atributima:visited{
    color: royalblue;
}
```

```
a.veza_sa_atributima:hover{
    text-decoration: none;
}

a.veza_sa_atributima:active{
    color: yellow;
}

div.sidro {
    text-align: center;
    width: 100%;
    position: fixed;
    bottom: 0px;
    background-color: darkmagenta;
}
```

## 4.4 Pseudoklase i pseudoelementi

Pored pseudoklasa koje možemo primeniti na linkove, postoje i one koje se mogu primeniti na sve elemente:

- `lang` - HTML elementima možemo dodeliti atribut `lang`. Pseudoklasa `lang(language)` omogućava nam da definišemo različita svojstva za različite jezike. Pomoću selektora `selector:lang(language)` dohvatamo sve elemente koji su obuhvaćeni selektorom `selector` i imaju podešen atribut `lang` na vrednost `language`.
- `hover` i `active` - Ove pseudoklase ponašaju se isto kao i kod linkova.
- `kombinacije` - Možemo kombinovati više pseudoklasa. Na primer, selektor `p:lang(en):hover` određuje stil svih paragrafa koji su napisani na engleskom jeziku kada se mišem pređe preko njih.

Pored pseudoklasa, postoje i takozvani pseudoelementi koje možemo uređivati posebno. Neki od pseudoelemenata su:

- `first-line` - Pseudoelement `first-line` označava prvu liniju elementa. Pomoću selektora `selector::first-line` dohvatamo sve elemente koji su obuhvaćeni selektorom `selector`, a svojstva koja navedemo odnose se isključivo na prvu liniju elementa.
- `first-letter` - Ovaj pseudoelement omogućava zasebno stilizovanje prvog slova u tekstu. Selektorom `selector::first-letter` dohvatamo sve elemente koji su obuhvaćeni selektorom `selector`, a svojstva koja navedemo odnose se isključivo na prvo slovo teksta koji je smešten u elementu.

Naredni kod obuhvata primenu svih navedenih pseudoklasa i pseudoelemenata:

Kod 45: `primeri/pseudoklasa/pseudoklase.html`

```
<html>
```

```
<head>
    <title> :: pseudoklase ::</title>
    <meta charset="UTF-8">

    <style type="text/css">
        p:lang(en){
            color:rgb(255, 90, 0);
        }

        p.dugi_pasus::first-line{
            color: blue;
            font-style: italic;
        }

        p.treci_pasus::first-letter{
            font-size: 25px;
            color: red;
        }

        #intro:hover{
            background-color:rgb(205, 122, 122);
            color:rgb(0, 86, 94);
        }

        /* kombinovanje pseudoklasa */
        p:lang(sr):hover{
            background-color: yellow;
        }

    </style>
</head>

<body>
    <p lang="en"> The first paragraph. </p>
    <p> Drugi pasus. </p>
    <p class="dugi_pasus">
        Prva linija.
        <br/>
        Druga linija.
        <br/>
        Treća linija.
    </p>

    <p class="treci_pasus">
        Treći pasus.
    </p>

    <p id="intro">
        Pseudoklase :hover i :active se mogu primenjivati i na elemente
        koji nisu linkovi.
    </p>

    <p lang="sr">
```

```
Možemo kombinovati više pseudoklasa.  
</p>  
</body>  
</html>
```

## 5 Čas 7

Cilj ovog časa je upoznavanje studenata sa *responsive* dizajnom, odnosno sa podešavanjem izgleda stranica u zavisnosti od veličine uređaja na kom se prikazuju.

### 5.1 Pogled

Kada god pravimo responsive dizajn moramo navesti informacije o prostoru u okviru kojeg pregledač prikazuje sadržaj — *pogled* (engl. *viewport*). Kroz `meta` element možemo navesti informacije o pogledu i tako kontrolisati način prikaza sadržaja. Obično podešavamo da:

- širina prikaza bude jednaka širini uređaja postavljanjem vrednosti `width` na `device-width`
- veličina prikaza nije skalirana (uvećana ili umanjena), što se često dešava na mobilnim telefonima i tabletima, postavljanjem vrednosti `initial-scale` na `1.0`
- korisnik može da skalira (uvećava ili umanjuje) prikaz do neke granice (na primer, ne manje od 0.7) postavljanjem vrednosti `user-scalable` na `yes` i vrednosti `minimum-scale` na `0.7`.

Pregledač širinu okvira u CSS pikselima računa tako što se fizička dimenzija uređaja podeli *odnosom fizičkih i CSS piksela* (engl. *device pixel ratio*, skr. DPR). Na primer, ako je fizička širina ekrana 800px, a DPR ima vrednost 2, pogled će biti širine 400px.

### 5.2 Prikazivanje slike

Možemo kontrolisati i prikaz slike u zavisnosti od veličine uređaja. Postavljanjem css svojstva `max-width` na `100%` postižemo da se slika smanjuje ukoliko mora tako da cela bude prikazana na ekranu. Ukoliko je ekran dovoljno veliki da se cela slika prikaže na njemu, slika će biti originalne veličine.

U nastavku je dat primer koji ilustruje način upotrebe opisanih html atributa i css svojstava. Eksperimentisati sa različitim veličinama pregledača i posmatrati kako se menja prikaz stranice.

Kod 46: primeri/responsive/viewport.html

```
<!DOCTYPE html>
<html>
    <head>
        <title> :: prikaz :: </title>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width, initial-scale=1
            .0, user-scalable=yes, minimum-scale=0.7">
        <style type="text/css">
            #wrapper{
                color: rgb(255, 255, 255);
                background-color: #cf66ff;
            }
        </style>
    </head>
    <body>
        <h1>Prikaz slike</h1>
        <img alt="Slika prikazana na ekranu" width="100%">
    </body>
</html>
```

```
header {
    text-align: center;
}

img {
    max-width: 100%;
    border: 1px solid lightblue;
}

h2 {
    padding-top: 30px;
}

#content {
    margin: 30px;
}
</style>
</head>

<body>
<div id="wrapper">
    <header>
        <h2> Cascading style sheets </h2>
        <br>
        
    </header>
    <div id="content">
        <br>
        CSS sintaksa se sastoji od opisa izgleda elemenata u dokumentu

        Opis može da definiše izgled više elemenata, i više opisa može
        da definiše jedan element.
        Na taj način se opisi slažu jedan preko drugog da bi
        definisali konačni izgled određenog elementa
        (otuda naziv Cascading (engl. cascade – crep) da bi se
        dočaralo slaganje jednog stila preko drugog u definisanju
        konačnog
        izgleda elementa).
        <br>
        Svaki opis se sastoji od tri elementa:
        <ul>
            <li> definicija ciljnih elemenata</li>
            <li> svojstva</li>
            <li> vrednosti</li>
        </ul>
        Nakon što definišemo ciljne elemente, tj. elemente na koje će
        se trenutni opis odnositi, nizom parova
        svojstvo-vrednost definišemo izgled svakog ciljnog elementa.
    </div>
</div>
</body>
</html>
```

### 5.3 Pozicioniranje

Da bismo pozicioniranje elemenata stranice prilagodili veličini uređaja koristimo takozvane *medija upite* (engl. *media queries*). Za njih važi sledeća sintaksa:

```
@media not|only tip_medija and (svojstvo_medija: vrednost) {
    /* nasacss-pravila */
}
```

Umesto `tip_medija` možemo navesti neki od sledećih tipova:

- `all` - odnosi se na sve tipove uređaja (podrazumevana vrednost)
- `screen` - odnosi se na ekrane računara, tableta, pametnih telefona, ...
- `print` - odnosi se na štampače
- ...

Kao `svojstvo_medija` možemo koristiti svojstva:

- `width` - širina pogleda
- `height` - visina pogleda
- `max-width` - najveća širina prostora za prikaz
- `max-height` - najveća visina prostora za prikaz
- `min-width` - najmanja širina prostora za prikaz
- `min-height` - najmanja širina prostora za prikaz
- `aspect-ratio` - odnos širine i visine pogleda
- `orientation` - orijentacija pogleda (`landscape` ili `portrait`)
- ...

U narednom primeru možemo videti kako možemo da izmenimo širinu elemenata u zavisnosti od veličine ekrana. Struktura stranice podrazumeva naslovni deo na vrhu, u sredini navigacioni deo i sadržaj, i podnožje stranice. Elemente u sredini raspoređujemo prema veličini ekrana na sledeći način:

- Za ekrane širine najmanje 600px, ređamo elemente jedne do drugog tako da širina navigacije bude 10%, a tri kolone sa sadržajem po 30%. Da bismo to postigli biramo ključnu reč `only` i za tip medija biramo `screen`.
- Za ekrane širine najviše 599px želimo da prilagodimo prikaz tako da se svi elementi lepo vide. Svakom od ovih elemenata postavićemo širinu na 100% tako da budu vidljivi na malom ekranu.

Kod 47: primeri/responsive/pozicioniranje.html

```
<!DOCTYPE html>
<html>
    <head>
        <title> :: primer pozicioniranja :: </title>
```

```
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0
  ">
<style type="text/css">
  * {
    box-sizing: border-box;
  }

#header{
  height: 50px;
  max-width: 100%;
  min-width: 100%;
  background-color: #ff9966;
}

#nav{
  display: inline-block;
  vertical-align: top;
  height: 400px;
  background-color: #00cc99;
}

#col1{
  display: inline-block;
  vertical-align: top;
  height: 400px;
  background-color: #ffff66;
}

#col2{
  display: inline-block;
  vertical-align: top;
  height: 400px;
  background-color: #aa55dd;
}

#col3{
  display: inline-block;
  vertical-align: top;
  height: 400px;
  background-color: #55aadd;
}

@media only screen and (max-width: 599px) {
  #nav, #col1, #col2, #col3 {
    width: 100%;
  }
}

@media only screen and (min-width: 600px) {
  #nav {
    width: 10%;
  }
}
```

```

        #col1, #col2, #col3 {
            width: 30%;
        }
    }

    #footer{
        height: 50px;
        background-color: #993366;
        width: 100%;
    }

```

</style>

</head>

<body>

<div id="header"> </div>

<div id="nav"> </div><div id="col1"> </div><div id="col2"> </div><div id="col3"> </div>

<div id="footer"> </div>

</body>

</html>

Vrednosti za širinu ili visinu ne moraju isključivo biti ograničene samo odozdo ili odozgo već možemo navoditi intervale veličina za koje želimo odgovarajuća svojstva. Tako možemo istovremeno postaviti i najmanju i najveću širinu ili visinu tako što ćemo svako svojstvo koje uključimo nadovezati u zagradi nakon ključne reči **and**, kao u primeru:

Kod 48: primeri/responsive/min\_max\_width.html

```

<!DOCTYPE html>
<html>
    <head>
        <title> :: max-width i min-width pojasnjenja :: </title>
        <meta charset="utf-8">
        <meta name="viewport" content="width=device-width, initial-scale=1">
        <link rel="stylesheet" type="text/css" media="screen and (max-width:
            499px)" href="small_devices.css">
        <style type="text/css">
            div{
                width: 100px;
                height: 100px;
                border: 5px solid red;
                box-sizing: border-box;
            }

            @media screen and (min-width: 500px) and (max-width:999px){
                div{
                    background-color: green;
                }
            }

            @media screen and (min-width: 1000px){
                div{
                    background-color: orange;
                }
            }
        </style>
    </head>
    <body>
        <div> </div>
    </body>
</html>

```

```

        }
    }
</style>
</head>

<body>
    <div> </div>
</body>

</html>

```

Kod 49: primeri/responsive/small\_devices.css

```

div{
    background-color: red;
}

```

Radi veće preglednosti, možemo izdvojiti podešavanja za različite tipove uređaja u različite `.css` datoteke u kojima ćemo pisati specifična svojstva za elemente u zavisnosti od veličine ili orientacije i slično. To možemo postići dodavanjem vrednosti atributu `media` u okviru elementa `link`, kao u primeru iznad. Vrednost ovog atributa postavljamo u obliku `tip_medija and (svojstvo_medija: vrednost)`.

### Zadatak 5.1:

Napraviti stranicu na kojoj je ispisan tekst "Zdravo svima", a element koji ga sadrži uređen je na različite načine, u zavisnosti od veličine ekrana i to:

- za uređaje do 400px postaviti crvenu boju slova
- za uređaje do 700px postaviti narandžastu boju slova
- za uređaje do 1000px promeniti pozadinu elementa na sivu
- za veće uređaje postaviti veličinu fonta na 30px

### Rešenje:

Kod 50: zadaci/005/media.html

```

<!DOCTYPE html>

<html>
    <head>
        <title> :: media queries demo :: </title>
        <meta charset="utf-8">
        <style type="text/css">
            @media screen and (max-width: 400px){
                #test{
                    color: red;
                }
            }

            @media screen and (min-width: 401px) and (max-width: 700px){
                #test{
                    color: orange;
                }
            }
        </style>
    </head>
    <body>
        <div id="test">Zdravo svima!</div>
    </body>
</html>

```

```
        }

        @media screen and (min-width: 701px) and (max-width: 1000px) {
            #test{
                background-color: gray;
            }
        }

        @media screen and (min-width: 1001px) {
            #test{
                font-size: 30px;
            }
        }
    </style>
</head>

<body>
    <div id="test">
        Zdravo svima!
    </div>
</body>
</html>
```

## 5.4 Bootstrap

Bootstrap je besplatno okruženje za brže i jednostavnije razvijanje veb sadržaja na klijentu. U ovoj sekciji prikazaćemo kako pomoći Bootstrap-a možemo dobiti responsive dizajn. Kako bismo mogli da ga koristimo potrebno je da ga preuzmemos sa adrese <http://getbootstrap.com>.

Pre svega, odredićemo jedan element koji će nam služiti kao omotač i u njega smeštamo sve ostale elemente. Ovom elementu dodajemo klasu `container` (ukoliko želimo omotač fiksirane širine) ili `container-fluid` (ukoliko želimo omotač 100% širine pogleda).

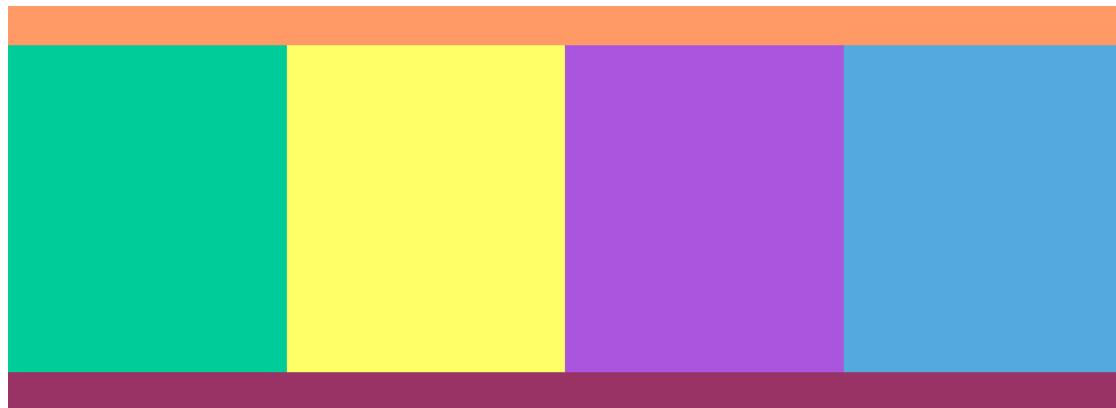
Stranicu delimo na redove. Jedan red je element sa klasom `row`. Red se deli na najviše 12 kolona i treba voditi računa da ukupan broj kolona u jednom redu ne bude veći od 12. Ukoliko ne želimo da koristimo svih 12 kolona pojedinačno, možemo ih grupisati i time dobiti šire kolone. Ovaj sistem je prilagodiv veličini ekrana, tako da ako u jednom redu nema mesta da se smesti svih 12 kolona, one će biti preuređene jedna ispod druge.

Klase koje možemo koristiti za obeležavanje kolona su:

- `col-*` - označava veoma male ekrane, veličine do 576px
- `col-sm-*` - označava male ekrane, od 576px naviše
- `col-md-*` - označava ekrane srednje veličine, od 768px naviše
- `col-lg-*` - označava velike ekrane, od 992px naviše
- `col-xl-*` - označava veoma velike ekrane, od 1200px naviše

Pomoću ovih klasa lako kontrolišemo prilagodivost stranice veličini ekrana. Za svaku veličinu ekrana možemo definisati koliko kolona treba da zauzima element tako što umesto zvezdica navodimo broj kolona. Još jednom da napomenemo — vodimo računa da u jednom redu, koji se označava klasom `row`, ne bude više od 12 kolona.

U narednom primeru možemo videti jedan način organizovanja stranice pomoću bootstrap okruženja:



Kod 51: primeri/bootstrap/pozicioniranje.html

```
<!DOCTYPE html>
<html>

    <head>
        <title> :: primer pozicioniranja – bootstrap :: </title>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width, initial-scale=1.0
            ">
        <link rel="stylesheet" href="bootstrap.css">
        <style>
            #header{
                height: 50px;
                background-color: #ff9966;
            }

            #nav{
                height: 400px;
                background-color: #00cc99;
            }

            #col1{
                height: 400px;
                background-color: #ffff66;
            }

            #col2{
                height: 400px;
                background-color: #aa55dd;
            }
        </style>
    </head>
    <body>
```

```
#col3{  
    height: 400px;  
    background-color: #55aadd;  
}  
  
#footer{  
    height: 50px;  
    background-color: #993366;  
}  
  
</style>  
</head>  
  
<body>  
    <div class="container-fluid">  
        <div class="row">  
            <div id="header" class="col-xs-12 col-sm-12 col-md-12  
                col-lg-12"></div>  
        </div>  
        <div class="row">  
            <div id="nav" class="col-sm-3 col-md-3 col-lg-3"> </div>  
            <div id="col1" class="col-sm-3 col-md-3 col-lg-3"> </div>  
            <div id="col2" class="col-sm-3 col-md-3 col-lg-3"> </div>  
            <div id="col3" class="col-sm-3 col-md-3 col-lg-3"> </div>  
        </div>  
        <div class="row">  
            <div id="footer" class="col-xs-12 col-sm-12 col-md-12  
                col-lg-12"> </div>  
        </div>  
    </div>  
  
</body>  
</html>
```

Takođe, napomenimo da ako neki element zauzima, na primer, 3 kolone na veličini `md`, onda će on podrazumevano zauzimati isto toliko kolona i na svakoj većoj veličini (dakle, `lg` i `xl`).

**Margine** Postoje i posebne klase za kontrolu razmaka između elemenata, odnosno za kontrolu spoljašnjih margina. Ove klase ubrajaju se u kolone, a obeležavamo ih sa `offset-*-*`, gde se prva zvezdica zamjenjuje veličinom uređaja, a druga brojem kolona. Kako se i ove klase ubrajaju u kolone, i njihov broj treba uključiti u ukupan broj kolona u jednom redu koji ne sme biti veći od 12.

**Naslovi** Bootstrap definiše nova svojstva za elemente `h1` do `h6` i sadrži klase sa istim nazivima u kojima dodatno kontroliše izgled ovih elemenata. Razlike između podrazumevanih vrednosti i Bootstrap vrednosti možete pogledati [https://www.w3schools.com/bootstrap/bootstrap\\_typography.asp](https://www.w3schools.com/bootstrap/bootstrap_typography.asp).

Sledeći primer prikazuje upotrebu kolona kao margina.

Kod 52: primeri/bootstrap/margine.html

```
<!DOCTYPE html>
<html>
  <head>
    <title> :: primer pozicioniranja – bootstrap :: </title>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link rel="stylesheet" type="text/css" href="bootstrap.css">
    <style type="text/css">
      header{
        height: 100px;
        background-color: #ff9966;
        margin-bottom: 20px;
      }

      main{
        height: 300px;
        background-color: #ffff66;
        margin-bottom: 20px;
      }

      footer{
        height: 50px;
        background-color: #993366;
      }
    </style>
  </head>

  <body>

    <div class="container">
      <div class="row">
        <header class="col-sm-4 col-md-4 col-lg-4">
          <h1 class="h1 text-center"> Naslov 1 </h1>
        </header>

        <header class="col-sm-6 col-md-6 col-lg-6 offset-sm-2 offset-md-2
          offset-lg-2">
          <h1 class="h1 text-center"> Naslov 2 </h1>
        </header>
      </div>

      <div class="row" >
        <main class="col-sm-10 col-md-10 col-lg-10 offset-sm-1
          offset-md-1 offset-lg-1" >
          Glavni sadržaj
        </main>
      </div>

      <div class="row">
        <footer class="col-sm-12">
          Podnožje strane
        </footer>
      </div>
    </div>
  </body>

```

```
</footer>
</div>
</div>
</body>
</html>
```

## 6 Čas 8

Cilj ovog časa je demonstriranje osnovnih elemenata programskog jezika JavaScript. Takođe, čitalac će se upoznati u način izvršavanja koda napisanog u jeziku JavaScript u okviru veb pregledača, zatim će se upoznati sa modelom objekata dokumenta i funkcijama za njegovo upravljanje.

U okviru veb pregledača dostupne su nam *alatke za razvoj* (engl. *developer tools*), koje između ostalih sadrže i interaktivnu konzolu u kojoj možemo kucati JavaScript kod i videti rezultate izvršavanja.

### 6.1 Osnovne jezičke strukture

**Brojevi** Najjednostavniji tip podataka je *numerički* (engl. *number*). Svi numerički tipovi su zapisani u 64 bita. Nad numeričkim tipovima su definisani standardni binarni operatori `+`, `-`, `*`, `/`, `%`, `<`, `<=`, `>`, `>=` i unarni operator `-`, sa standardnim pravilima za prednost. Zgrade `( )` mogu da promene prednost izračunavanja. Takođe, podrazumeva se leva asocijativnost. JavaScript podržava:

- celobrojne vrednosti — `0`, `7`, `-42`, ...
- razlomljene vrednosti — `9.81`, `2.998e8` ( $= 2.998 \cdot 10^8$ ), ...
- "beskonačnosti" — `Infinity`, `-Infinity`
- `NaN` — označava da "nije broj" (skr. *Not a Number*), iako je njegov tip numerički. Ovaj rezultat možemo dobiti ako, na primer, pokušamo da izračunamo `0 / 0` ili `Infinity - Infinity`, ili bilo koju drugu nedefinisana operaciju.

**Niske** Sledeći osnovni tip su *niske* (engl. *string*). Niske se koriste za reprezentaciju teksta. Kodna shema koja se koristi je Unicode. Možemo konstruisati literal ovog tipa pomoću jednostrukih ili dvostrukih navodnika, kao i "nakošenih" navodnika (tzv. *šablon-literali* (engl. *template literals*)). Na primer:

- `'Programiranje u JavaScript-u je kul!'`
- `"Koriscenje jQuery-a je bolje!"`
- ``A tek da vidis Angular!``

Treba voditi računa o tome da kad započnemo literal niske jednim od ova tri karaktera navodnika, prvi sledeći put kada se najde na taj karakter, to će se smatrati krajem niske. Zato ne možemo direktno ugnezđavati, na primer, dvostrukе navodnike, već ih moramo označiti (engl. *escape*). Označavanje se vrši stavljanjem karaktera `\` ispred karaktera koji želimo da označimo. U narednom primeru, prva linija predstavlja neispravno, a druga linija ispravno ugnezđavanje navodnika:

- `"A onda je rekao: "Uradicu to!""`
- `"A onda je rekao: \"Uradicu to!\""`

Ako želimo da imamo višelinjske niske, to možemo uraditi na dva načina:

- Korišćenjem označenog karaktera za novi red `\n` na mestu gde želimo prelomiti u novi red. Na primer:

```
"Ovo je sve prva linija\nA odavde kreće druga linija"
```

- Korišćenjem šablon-literalata i prelamanjem tasterom za novi red na mestu gde želimo prelomiti u novi red. Na primer,

```
`Ovo je sve prva linija  
A odavde kreće druga linija`
```

Niske se mogu *nadovezivati* (engl. *concatenate*) operatorom `+`. Na primer, niska "nadovezivanje" se može dobiti izračunavanjem:

```
"na" + "do" + "vezivanje"
```

Poređenje niski pomoću standardnih operatora za poređenje `<`, `<=`, `>` i `>=` se vrši tako što se niske upoređuju leksikografski.

Šablon literali imaju još jedno zanimljivo svojstvo. U njih se mogu ugneždačavati razni drugi izrazi. Na primer, pri izračunavanju izraza

```
`Dva plus dva je jednako ${2 + 2}`
```

prvo će se izračunati vrednost izraza zapisan između `${ i }`, zatim će se njegova vrednost konvertovati u nisku, i na kraju, ta niska će biti ugnežđena na tu poziciju. Dakle, prethodni izraz će biti izračunat u izraz

```
Dva plus dva je jednako 4
```

**Bulove vrednosti** JavaScript takođe podržava izračunavanja koja proizvode Bulove vrednosti — `true` (tačno) i `false` (netačno). Binarni operatori poređenja regularno proizvode Bulove vrednosti nakon izračunavanja. Kao i u drugim programskom jezicima, Bulove vrednosti se najčešće koriste kao uslovi u naredbama grananja, petlji, i dr.

Baratanje Bulovim vrednostima se može jednostavno obaviti korišćenjem standardnih binarnih operatara `&&` (konjukcije), `||` (disjunkcije), unarnog operatatora `!` (negacije) i ternarnog operatatora `?:` (uslovni operator).

**Prazne vrednosti** Postoje dve specijalne vrednosti, `null` i `undefined`, koje se koriste da označe odsustvo "vrednosti koje ima značenje". One same predstavljaju vrednosti, ali ne nose nikakvu informaciju. Mnoga izračunavanja u jeziku JavaScript koja se proizvode vrednosti koje imaju značenje (u nastavku poglavljia ćemo se upoznati sa njima) proizvode `undefined` jednostavno zato što moraju da dobiju nekakvu vrednost. Razlika u značenjima između `null` i `undefined` dolazi iz konstrukcije jezika, ali mi ćemo ih smatrati sinonimima.

**Implicitne konverzije** JavaScript je programski jezik koji može da "proguta" razne programske izraze koji uključuju osnovne tipove i da izračuna od njih nekakve vrednosti, koje možda ne bismo očekivali. Na primer:

- Vrednost izraza `8 * null` je `0`.

- Vrednost izraza `"5" - 1` je `4`.
- Vrednost izraza `"5" + 1` je `"51"`.
- Vrednost izraza `"pet" * 2` je `Nan`.
- Vrednost izraza `!""` je `true`.

Objašnjenje za ovakvo ponašanje leži u *implicitnoj konverziji*. Naime, kada se nekom operatoru proslede tipovi koji su različiti (primeri 1 – 4) ili tip koji on ne očekuje (primer 5), JavaScript će pokušati da, nekim komplikovanim mehanizmom, konvertuje tipove tako da može da primeni operator na njih. U slučaju da to ne uspe, rezultat je `Nan`. Na primer:

- U prvom primeru, prazna vrednost `null` se konvertuje u broj `0`, pa se izračunava množenje brojeva `8 * 0`.
- U drugom primeru, vrednost niske `"5"` se konvertuje u broj `5`, pa se izračunava oduzimanje brojeva `5 - 1`.
- U trećem primeru, vrednost broja `1` se konvertuje u nisku `"1"`, pa se izračunava nadovezivanje niski `"5" + "1"`.
- U četvrtom primeru, vrednost niske `"pet"` se ne može mapirati u brojčanu vrednost na očigledan način (isto važi i za `undefined`), pa ta konverzija rezultuje vrednošću `Nan`, pa se izračunava množenje `Nan * 2`.
- U petom primeru, vrednost niske `""` se konvertuje u `false`, pa se izračunava negacija `!false`.

**Poređenje po jednakosti i nejednakosti** Jedan slučaj implicitne konverzije koji treba posebno razmotriti jeste kada se dve vrednosti porede po jednakosti ili nejednakosti. Kada se dve vrednosti istog tipa porede operatorom `==`, rezultat je jednostavno predvideti: rezultat će biti `true` ako su vrednosti iste, osim u slučaju vrednosti `Nan` (videti ispod). Međutim, kada se tipovi razlikuju, onda nije jednostavno i, štaviše, često je zburujuće utvrditi rezultat poređenja.

U slučaju poređenje vrednosti `null` i `undefined`, njihovo međusobno poređenje će proizvesti `true`. Međutim, poređenje sa nekom drugom vrednošću će proizvesti `false`. Ovo može biti korisno za testiranje da li je neka vrednost smislena ili ne.

Vratimo se na slučaj poređenja dve vrednosti različitih tipova. Šta ukoliko bismo želeli da poredimo da li je neka vrednost baš `false`? Prva ideja jeste da je poredimo pomoću operatora `==` sa vrednošću `false`. Međutim, naredni primer nam pokazuje da to neće raditi dobro:

- Vrednost izraza `false == 0` je `true`.

Dakle, nismo dobili željeni efekat. Želeli bismo da nekako "preciznije" poredimo vrednosti. U tu svrhu, u jeziku JavaScript postoji operator `==` koji pored toga što proverava da li su dve vrednosti jednake, proverava i da li su te vrednosti istog tipa! Ovim rešavamo prethodni problem, što naredni primer ilustruje:

- Vrednost izraza `false === 0` je `false`.

Poređenje dve `Nan` vrednosti će u jeziku JavaScript uvek proizvesti vrednost `false`. Smisao ovoga je da, s obzirom da `Nan` predstavlja rezultat neodređenog izračunavanja, ono ne može biti jednak nekom "drugom" neodređenom izračunavanju. Za ispitivanje da li je vrednost izraza `Nan`, može se iskoristiti funkcija `Number.isNaN`, koja kao argument prihvata neki izraz, a rezultat je `true` ukoliko taj rezultat proizvodi `Nan`, a `false` inače.

**Zamensko izračunavanje** Bulovi operatori `&&` i `||` imaju jedno zanimljivo svojstvo koje dolazi do izražaja kada se primenjuju nad vrednostima čiji tipovi nisu Bulove vrednosti:

- Operator `||` će vratiti vrednost levo od njega onda kada je tu vrednost moguće konvertovati u `true`. Inače, vraća vrednost desno od njega. Na primer,
  - Vrednost izraza `null || "korisnik"` je `"korisnik"`.
  - Vrednost izraza `"ime" || "prezime"` je `"ime"`.
- Operator `&&` će vratiti vrednost levo od njega onda kada je tu vrednost moguće konvertovati u `false`. Inače, vraća vrednost desno od njega. Na primer,
  - Vrednost izraza `"" && 42` je `""`.
  - Vrednost izraza `"ti" && "ja"` je `"ja"`.

Ovakvo ponašanje se naziva *zamensko izračunavanje* (engl. *short-circuit evaluation*). Ono se najčešće koristi u slučaju kada je potrebno iskoristiti neku vrednost, ali ukoliko ona nije dostupna, možemo koristiti neku drugu, na primer, podrazumevanu.

### 6.1.1 Programske strukture

Kao što smo videli do sada, osnovna struktura programa koja ne pravi *bočne efekte* je *izraz*. Ipak, kako ćemo mi praviti programe koji su složeniji od jednostavnog izračunavanja izraza, potrebne su nam *naredbe* koje imaju bočne efekte, kontrole toka, funkcije, i dr.

**Komentari** U programskom jeziku JavaScript, jednolinijski komentari se navode iza `//`, dok se višelinjiski komentari navode između `/*` i `*/`.

**Uvođenje promenljivih** Promenljive se uvode u *domet* (engl. *scope*) pomoću naredbe dodelje. Ona može imati oblik kao u narednom primeru:

```
let imePromenljive = 7 * 7;
```

Ključna reč `let` označava da će ova naredba dodeliti vrednost novoj promenljivoj čiji je identifikator zadat sa `imePromenljive`. Ukoliko samo deklarišemo novu promenljivu, bez defisanja njene vrednosti, a zatim zatražimo njenu vrednost, kao rezultat dobijemo `undefined`. Možemo definisati više vrednosti odvojenih zapeštom:

```
let a = 7, b = 42;
```

Pored ključne reči `let`, postoje još dve ključne reči za uvođenje promenljivih: `var` i `const`. Ključnom reči `var` se postiže isti efekat kao sa `let` (uz dodatne razlike, koje nećemo objašnjavati, te zbog toga ćemo koristiti isključivo `let`), a ključnom reči `const` uvodimo konstantne promenljive, tj. one promenljive čije vrednosti ne želimo da budu promenjene.

**Kontrole toka** JavaScript jezik podržava standardne kontrole toka, čija je forma ekvivalentna formi u programskom jeziku C:

- uslovne: `if` – `else if` – `else` i `switch`
- ponavljače: `while`, `do while` i `for`

**Funkcije** Funkcije se mogu definisati na više načina. Jedan od njih je dodeljivanje funkcije promenljivoj:

```
const imeFunkcije = function(arg1, arg2 /*, ..., argN */) {  
    // Telo funkcije  
};
```

Naravno, promenljiva ne mora biti konstantna, ali nema mnogo smisla redefinisati promenljivu koja sadrži funkciju kao vrednost. Funkcije se pozivaju navođenjem njihovih imena i vrednostima za argumente. Na primer:

```
const stepen = function(osnova, eksponent) {  
    let rezultat = 1;  
    for (let brojac = 0; brojac < eksponent; brojac++) {  
        rezultat *= osnova;  
    }  
    return rezultat;  
};  
  
stepen(2, 3); // 8
```

Postoji kraća notacija za uvođenje funkcija, čiji je oblik:

```
function imeFunkcije(arg1, arg2 /*, ..., argN */) {  
    // Telo funkcije  
}
```

Osim što je zapis kraći, ne zahteva se karakter `;` na kraju, kao kod dodeljivanja funkcije konstantnoj promenljivoj.

Postoji jedna posebna funkcija, koja se može pozvati pomoću naredbe `console.log(izraz)`. Njena uloga je da na "neki" tekstualni izlaz ispiše vrednost `izraz`. U veb pregledaču, tekstualni izlaz je JavaScript konzola.

**Anonimne funkcije** Ukoliko nije potrebno da imenujemo funkciju, već samo da se izvrši jedanput, možemo koristiti anonimne funkcije. Zapravo, već smo videli anonimne funkcije, a da to nismo ni znali! Kada definišemo funkciju putem dodeljivanja konstantnoj funkciji, izraz sa desne strane jednakosti čini jednu anonimnu funkciju, na primer:

```
function(x) {  
    return x*x;  
}
```

**Nizovi** Nizovi predstavljaju jednostavnu sekvencionalnu strukturu podataka. U JavaScript jeziku, literali nizova se zapisuju između zagrada [ i ], a vrednosti koje se skladište u nizu se razdvajaju zapetom. Na primer:

```
let nizBrojeva = [1, 2, 3, 4, 5];
let nizNiski = ["Niska1", "Niska2"];
```

Vrednostima u nizu se može pristupiti pomoću *indeksnog operatora*, koji takođe koristi zagrade [ i ], pri čemu se indeks elementa navodi između zagrada. Nizovi su indeksirani počevši od 0:

```
console.log(nizBrojeva[0]); // 1
console.log(nizNiski[1]); // Niska2
```

Ono što nije očigledno iz prethodnih primera jeste da nizovi mogu da čuvaju vrednosti različitog tipa. Na primer:

```
let niz = [1, 2, "Niska", { svojstvo: 2 }];

console.log(niz[0]); // 1
console.log(niz[1]); // 2
console.log(niz[2]); // Niska
console.log(niz[3]); // { svojstvo: 2 }
```

**Svojstva** U prethodnom primeru vidimo nešto do sada nismo videli — četvrti element (odnosno, treći element ako posmatramo sa stanovišta računara) strukture niz sadrži nešto što nazivamo literal *objekta* (engl. *object*). Objekti su još jedna struktura podataka i možemo ih posmatrati kao kolekcije nečega što nazivamo *svojstva* (engl. *properties*). Svojstva nisu ništa drugo nego vrednosti koje se čuvaju u okviru drugih objekata. Tako, iz prethodnog primera, objekat { svojstvo: 2 } ima samo jedno svojstvo koje se naziva *svojstvo* i čija je vrednost 2.

U prethodnom primeru smo videli jedan način za pristup svojstvima neke vrednosti — *tačka-notacijom*. Ona podrazumeva da navedemo vrednost, iza koje sledi karakter ., a zatim da navedemo ime svojstva. Postoji još jedan način za dohvatanje svojstava, a to je pomoću uglastih zagrada. Tako, na primer, korišćenjem vrednost.svojstvo i vrednost[svojstvo] možemo dohvatiti svojstvo od vrednost.

Elementi u nizu se čuvaju kao svojstva tog niza, te nam je sada jasno zašto mogu sadržati vrednosti različitih tipova. Međutim, oni koriste isključivo cele brojeve kao imena svojstava, te zbog toga za pristupanje elementima nizova uvek koristimo uglaste zagrade, kao što smo i pokazali.

Nad nizovima postoji definisano svojstvo length koje nam izračunava broj elemenata niza. Ono se najčešće koristi u tačka-notaciji jer unapred znamo ime tog svojstva. Naredni primer koristi ovo svojstvo da prođe kroz sve elemente niza nekiNiz i ispise njegove vrednosti na tekstualni izlaz:

```
let nekiNiz = ["Ovaj", "niz", "ima", 5, "elemenata"];
for(let i = 0; i < nekiNiz.length; i++) {
    console.log(nekiNiz[i]);
}
```

**Metodi** Nizovi i niske takođe sadrže brojna svojstva čije su vrednosti funkcije. Tako, na primer, sve niske sadrže svojstvo toUpperCase, koja vraća kopiju niske nad

kojom je pozvana u kojoj su svi karakteri pretvoreni u velika slova. Radi kompletnosti, navedimo u funkciju `toLowerCase` koja radi suprotno — transformiše velika slova u mala slova u kopiji niske nad kojom se poziva. Ovo možemo testirati narednim primerom:

```
let niska = "nIsKa";
console.log(niska.toUpperCase()); // NISKA
```

Svojstva čije su vrednosti funkcije se nazivaju *metodi* (engl. *methods*). Tako je, na primer, `toUpperCase` metod koji se nalazi u okviru tipa `string`.

## 6.2 Izvršavanje JavaScript koda u veb pregledaču

Ukoliko želimo da se neki JavaScript kod izvrši u okviru veb pregledača, potrebno je da u odgovarajućem dokumentu, a u okviru `<body>` elementa ugnezdimo element `<script>`, čiji je sadržaj upravo JavaScript kod koji se izršava. JavaScript kod ima svoj odgovarajući MIME tip `text/javascript`, koji možemo da navedemo kao vrednost `type` atributa elementa `script`. Na primer:

Kod 53: primeri/javascript1/testiranje-alert.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Testiranje alert funkcije</title>
</head>
<body>
    <h1>Testiranje alert funkcije.</h1>
    <script type="text/javascript">
        alert("Ovo je poruka!");
    </script>
</body>
</html>
```

Kod ugnezđen na ovakav način će biti izvršen čim veb pregledač pročita sadržaj elementa `<script>`. Funkcija `alert(message)` će otvoriti mali prozor u veb pregledaču i ispisati poruku `message`.

Naravno, za iole veće programske kodove, njihovo ugnezđavanje direktno u html datoteku nema nikakvog smisla. Takvi kodovi se čuvaju u datotekama sa ekstenzijom `.js`. Njih možemo uključiti u veb stranicu ponovo koristeći element `<script>`, samo što se ovoga puta umesto navođenja koda kao njegov sadržaj, koristi atribut `src`, čija je vrednost putanja do datoteke koja sadrži JavaScript kod. Prethodni primer se može razdvojiti u html i js datoteke na sledeći način:

Kod 54: primeri/javascript1/testiranje-alert-2.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Testiranje alert funkcije</title>
</head>
<body>
```

```
<h1>Testiranje alert funkcije.</h1>
<script type="text/javascript" src="testiranje-alert-2.js"></script>
</body>
</html>
```

Kod 55: primeri/javascript1/testiranje-alert-2.js  
alert("Ovo je poruka!");

Ono što primećujemo jeste da u svakom slučaju element `<script>` mora imati zatvarajuću etiketu.

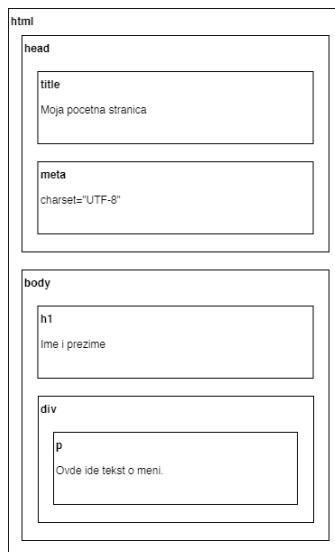
### 6.3 Rad sa DOM stablom

**Uvod u DOM stablo** Kada veb pregledač otvorí neki HTML dokument, on ga parsira i od toga pravi model tog dokumenta u memoriji. Nad tim modelom programeri mogu da izvršavaju razne operacije, koje će uporedo i osvežavati prikaz stranice u prozoru veb pregledača.

Neka nam je dat naredni HTML dokument:

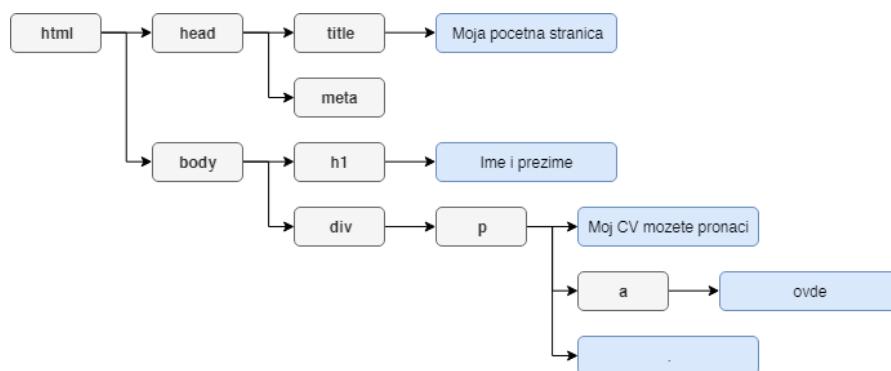
```
<!DOCTYPE html>
<html>
  <head>
    <title>Moja pocetna stranica</title>
    <meta charset="UTF-8">
  </head>
  <body>
    <h1>Ime i prezime</h1>
    <div>
      <p>Moj CV mozete pronaci <a href="cv.pdf">ovde</a>.<p>
    </div>
  </body>
</html>
```

Možemo da zamislimo strukturu HTML dokumenta kao ugnezđene kutije. Struktura HTML dokumenta iznad se vizualno može predstaviti na sledeći način:



Struktura kojom veb pregledač opisuje model HTML stranice prati ovaj grafički prikaz. Za svaku kutiju postoji objekat, sa kojim programeri mogu da interaguju da bi dohvatili njegova svojstva ili izvršavali neke metode. Ovakva reprezentacija se naziva *model objekata dokumenta* (engl. *document object model*, skr. DOM).

Preciznije, struktura podataka kojom se opisuje DOM je *stablo* (engl. *tree*). Elementi su predstavljeni čvorima stabla i oni određuju njegovu strukturu. Ugnežđeni elementi predstavljaju *naslednike* (engl. *child*) čvora koji je njihov *roditelj* (engl. *parent*). Neki čvorovi imaju *listove* (engl. *leaf*) za naslednike, što mogu biti tekstualni elementi, komentari i dr. Na primer, drvolika struktura prethodnog HTML dokumenta se može prikazati na sledeći način:



Sada ćemo se upoznati sa različitim operacijama koje možemo primeniti nad DOM stablom.

**Pronalaženje elemenata u DOM stablu** U okviru veb pregledača postoji JavaScript objekat koji se naziva `document`, koji predstavlja veb stranicu. Ako želimo da pristupimo nekom elementu na veb stranici, uvek možemo da započnemo od `document` objekta. Nad ovim dokumentom su definisani brojni metodi za pretragu elemenata.

U narednim primerima prepostavljamo da imamo HTML dokument čiji `body` element ima sledeći sadržaj:

```

<p id="prvi_pasus">Ovo je primer</p>
<p id="drugi_pasus" class="parni">rada sa DOM stablom</p>
<p id="treci_pasus">u programskom jeziku JavaScript.</p>
  
```

Prvi metod koji ćemo upoznati je `getElementById`. U pitanju je metod koji omogućava dohvatanje elementa ukoliko poznajemo njegov identifikator. Na primer:

```

let prviPasus = document.getElementById("prvi_pasus");
console.log("Pristup prvom pasusu preko identifikatora: " + prviPasus.textContent);
  
```

Ovde vidimo da možemo da koristimo svojstvo `textContent` kojim dohvativamo tekstualni sadržaj elementa.

Metod `getElementsByName` omogućava dohvatanje elemenata na osnovu njegovog imena. Kako može postojati više elemenata sa istim nazivom, rezultat rada ove funkcije je niz. Na primer:

```

let sviPasusi = document.getElementsByTagName("p");
let brojPasusa = sviPasusi.length;
  
```

```
console.log("Pristup poslednjem pasusu preko imena elementa: " + sviPasusi[  
brojPasusa-1].textContent);
```

Metod `getElementsByClassName` omogućava dohvatanje elemenata na osnovu imena klase koja im je pridružena. Kako više elemenata može imati istu klasu, rezultat je niz. Na primer:

```
let parniPasusi = document.getElementsByClassName("parni");  
console.log("Pristup srednjem pasusu preko klase: " + parniPasusi[0].  
textContent);
```

Metod `querySelector` omogućava dohvatanje elemenata na osnovu selektora koji odgovaraju CSS selektorima. Napomenimo da ova funkcija uvek dohvata samo prvi element koji zadovoljava zadati selektor. Na primer:

```
let srednjiPasus = document.querySelector(".parni");  
console.log("Pristup srednjem pasusu preko klase i querySelector funkcije: "  
+ srednjiPasus.textContent);
```

Metod `querySelectorAll` omogućava dohvatanje elemenata na osnovu selektora koji odgovaraju CSS selektorima. Za razliku od metoda `querySelector`, ova funkcija uvek dohvata sve elemente koji zadovoljavaju zadati selektor, pa se time oni mogu obrađivati kao nizovi. Na primer:

```
let sviPasusiPonovo = document.querySelectorAll("p");  
console.log("Pristup poslednjem pasusu preko querySelectorAll funkcije: " +  
sviPasusiPonovo[brojPasusa - 1].textContent);
```

Napomenimo da ovaj niz nije dinamičke prirode. Na primer, ako se u toku rada promeni neki element koji je uključen u rezultat selekcije, promene nisu vidljive na nivou niza. Ceo primer je dat narednim kodom:

Kod 56: primeri/javascript1/01\_js\_selektori.html

```
<!DOCTYPE html>  
  
<html>  
  
<head>  
    <title>JS funkcije za dohvatanje elemenata DOM-a</title>  
    <meta charset="UTF-8">  
</head>  
  
<body>  
    <p id="prvi_pasus">Ovo je primer</p>  
    <p id="drugi_pasus" class="parni">rada sa DOM stablom</p>  
    <p id="treci_pasus">u programskom jeziku JavaScript.</p>  
  
    <script type="text/javascript">  
        let prviPasus = document.getElementById("prvi_pasus");  
        console.log("Pristup prvom pasusu preko identifikatora: " + prviPasus  
            .textContent);  
  
        let sviPasusi = document.getElementsByTagName("p");  
        let brojPasusa = sviPasusi.length;  
        console.log("Pristup poslednjem pasusu preko imena elementa: " +  
            sviPasusi[brojPasusa-1].textContent);  
    </script>
```

```
let parniPasusi = document.getElementsByClassName("parni");
console.log("Pristup srednjem pasusu preko klase: " + parniPasusi[0].textContent);

let srednjiPasus = document.querySelector(".parni");
console.log("Pristup srednjem pasusu preko klase i querySelector funkcije: " + srednjiPasus.textContent);

let sviPasusiPonovo = document.querySelectorAll("p");
console.log("Pristup poslednjem pasusu preko querySelectorAll funkcije: " + sviPasusiPonovo[brojPasusa - 1].textContent);
</script>
</body>

</html>
```

**Upravljanje stilovima elemenata** Svojstvo `style` se koristi za postavljanje (i očitavanje) stilova preko `style` atributa. Na primer, možemo da postavimo boju teksta elementa tako što njegovom svojstvu `style.color` dodelimo odgovarajuću boju kao nisku. CSS svojstva koja su oblika `css-svojstvo` očigledno ne predstavljaju adekvatne JavaScript identifikatore, te se oni zamenjuju svojstvima oblika `cssSvojstvo`. Tako, na primer, CSS svojstvo `font-size` postaje `fontSize`.

Ukoliko želimo da saznamo ili da postavimo klase kojima element pripada, to možemo uraditi pristupanjem svojstvu `className`, čija je vrednost niska koja sadrži imena tih klasa. Ako želimo da dodamo više klasa odjednom, onda treba razdvojiti imena klasa belinom.

Naredni primer ilustruje korišćenje ova dva svojstva:

Kod 57: primeri/javascript1/02\_stilovi.html

```
<!DOCTYPE html>
<html>

<head>
    <title>Pridruzivanje stilova</title>
    <meta charset="UTF-8">

    <style type="text/css">
        .doterani_pasus {
            border: 2px solid green;
            font-size: 20px;
            color: orange;
            width: 100px;
        }
    </style>

</head>

<body>
    <p id="prvi_pasus">Ovo je prvi pasus.</p>
```

```
<p id="drugi_pasus">Ovo je drugi pasus.</p>

<script type="text/javascript">
    document.querySelector("#prvi_pasus").style.color = "blue";
    document.querySelector("#prvi_pasus").style.fontSize = "12px";

    document.querySelector("#drugi_pasus").className = "doterani_pasus";
</script>
</body>

</html>
```

**Očitavanje i postavljanje sadržaja** Kao što smo videli do sada, svojstvo `textContent` se koristi za čitanje i postavljanje tekstuallnog sadržaja. Nestandardizovano, ali široko podržano svojstvo je i `innerHTML` kojim se može očitati, kao i postaviti HTML sadržaj elementa.

Naredni primer ilustruje korišćenje ova dva svojstva:

Kod 58: `primeri/javascript1/03_sadrzaj.html`

```
<!DOCTYPE html>
<html>

<head>
    <title>Ocitavanje tekstuallnog sadrzaja</title>
    <meta charset="UTF-8">
</head>

<body>
    <div id="omotac">
        <p>
            Svanuo je lep i suncan dan.
        </p>
    </div>

    <script type="text/javascript">
        let pasus = document.querySelector("p");
        console.log(pasus.textContent);

        let div = document.querySelector("div");
        console.log(div.innerHTML);
    </script>
</body>

</html>
```

**Interfejs DOM modela** Iako smo videli da novi HTML sadržaj možemo dodati izmenom svojstva `innerHTML`, jedna manja ovog pristupa jeste što bismo morali prvo da generišemo nisku koja sadrži novi HTML sadržaj, a zatim da tu nisku iskoristimo kao vrednost `innerHTML` svojstva. Ovakav način programiranja nije nešto što se radi u praksi, ali na sreću po programere, postoje načini kako možemo na ispravan način da kreiramo elemente na stranici i da upravljamo njima.

Nad objektom `document` definisan je metod `createElement` koji na osnovu prosledenog argumenta koji predstavlja naziv HTML elementa, kreira novi objekat koji predstavlja upravo element čiji je to naziv. Na primer, da bismo kreirali paragraf, dovoljno je da pozovemo:

```
let pasus1 = document.createElement("p");
```

Ukoliko želimo da postavimo tekstualni sadržaj nad elementom, prvo je potrebno da napravimo novi tekstualni čvor korišćenjem metoda `document.createTextNode`, a zatim da ga dodamo kao naslednika odgovarajućem elementu pomoću metoda `appendChild` nad elementom:

```
let sadrzaj = document.createTextNode("Ovo je prvi pasus");
pasus1.appendChild(sadrzaj);
```

Ipak, novokreirani element neće biti vidljiv sve dok ga ne dodamo kao naslednika nekom roditeljskom elementu. Na primer, možemo ga dodati kao naslednika elementa `body`, koji je dostupan kao istoimeni svojstvo objekta `document`:

```
document.body.appendChild(pasus1);
```

Ono što treba napomenuti jeste da će metod `appendChild` uvek dodati element kao poslednjeg naslednika u nizu.

Ukoliko želimo da promenimo vrednost atributa nad elementom, možemo pozvati metod `setAttribute(attribute, value)`, koji postavlja vrednost atributa zadat niskom `attribute` na vrednost zadatu niskom `value`. Jedan interesantan generički atribut se naziva `title`, čija vrednost će biti prikazana kao *informaciona kutijica* (engl. *tooltip*) pozicioniranjem kursora miša nad elementom:

```
pasus1.setAttribute("title", "Pasus napravljen koriscenjem programskog
jezika JavaScript!");
```

Još jedan način za dodavanje elementa na stranici jeste korišćenjem metoda `insertBefore(newnode, existingnode)`. Ovim metodom se objekat `newnode` smesta kao naslednik elementa nad kojim se metod poziva, ali ne kao poslednji element u nizu, već ispred naslednika koji je zadat kao `existingnode`. Na primer, neka smo kreirali objekat `pasus2` koji predstavlja još jedan paragraf. Ako želimo da se on nađe ispred `pasus1`, onda je dovoljno da pozovemo:

```
document.body.insertBefore(pasus2, pasus1);
```

Brisanje naslednika se postiže pozivom metoda `removeChild`:

```
document.body.removeChild(pasus1);
```

Ceo primer je dat narednim kodom:

Kod 59: primeri/javascript1/04\_kreiranje\_elemenata.html

```
<!DOCTYPE html>
<html>

<head>
    <title>Kreiranje elemenata</title>

    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width">
```

```
<style type="text/css">
    .komentar {
        border: 1px dashed orange;
        padding: 10px;
    }
</style>
</head>

<body>
<script type="text/javascript">
    // Pravimo prvi paragraf
    let pasus1 = document.createElement("p");
    pasus1.className = "komentar";
    pasus1.setAttribute("title", "Pasus napravljen koriscenjem
        programskog jezika JavaScript!");

    let sadrzaj = document.createTextNode("Ovo je prvi pasus");
    pasus1.appendChild(sadrzaj);

    document.body.appendChild(pasus1);

    // Pravimo drugi paragraf
    let pasus2 = document.createElement("p");
    pasus2.className = "komentar";
    pasus2.setAttribute("title", "Pasus napravljen koriscenjem
        programskog jezika JavaScript!");

    sadrzaj = document.createTextNode("Ovo je drugi pasus");
    pasus2.appendChild(sadrzaj);

    document.body.insertBefore(pasus2, pasus1);

    // Pravimo treci paragraf
    let pasus3 = document.createElement("p");
    pasus3.className = "komentar";
    pasus3.setAttribute("title", "Pasus napravljen koriscenjem
        programskog jezika JavaScript!");

    sadrzaj = document.createTextNode("Ovo je treci pasus");
    pasus3.appendChild(sadrzaj);

    document.body.appendChild(pasus3);

    // Brisemo treci paragraf
    document.body.removeChild(pasus3);
</script>
</body>

</html>
```

## 6.4 Rad sa konzolom veb pregledača

Kao što smo do sada videli, u okviru veb pregledača nam je dostupna konzola za izvršavanje JavaScript kodova, kao i za ispisivanje nekakvih poruka. Objekat koji radi sa konzolom veb pregledača je `console`. Do sada smo videli korišćenje metoda `log` koji jednostavno ispisuje kontrolne poruke. Međutim, nad ovim objektom su definisani i drugi metodi:

- `error` — ispisuje poruku u vidu greške
- `warn` — ispisuje poruku u vidu upozorenja
- `info` — ispisuje poruku u vidu obaveštenja

Naredni primer ilustruje korišćenje opisanih metoda:

Kod 60: primeri/javascript1/05\_console.html

```
<!DOCTYPE html>
<html>

<head>
    <title> Rad sa konzolom </title>
    <meta charset="UTF-8">
</head>

<body>
    Pratiti ispis u konzoli pregledača.

    <script type="text/javascript">
        console.log("Zdravo svima!");
        console.error("Primer greske!");
        console.warn("Primer upozorenja");
        console.info("Primer obavestenja");
    </script>
</body>

</html>
```

## 6.5 Pridruživanje osluškivača događaja elementima

Nad raznim HTML elementima su definisani *osluškivači događaja* (engl. *event listener*). Kada je neki događaj dogodi, na primer, kada se klikne tasterom miša na neki element, može se definisati kakvu akciju treba veb pregledač da preduzme. Postoji više načina da se ovo učini.

Prvi način je da se nad elementom iskoristi atribut, čija je vrednost JavaScript kod. Ovakav način pridruživanja podrazumeva postojanje odgovarajućih atributa na nivou elementa. Neki od atributa koji postoje su:

- `onclick` definiše šta se događa kada korisnik klikne na površinu elementa
- `onchange` definiše šta se događa kada korisnik promeni vrednost, na primer, `input` elementa ili `select` elementa

- `onfocus` definiše šta se događa kada element dobije fokus (kada se korisnik pozicionira bilo mišem bilo TAB tasterom na ovo polje)
- `onblur` definiše šta se događa kada element izgubi fokus
- `onhover` definiše šta se događa kada se pređe preko površine elementa kur-sorom
- `onkeydown` definiše šta se događa kada korisnik pritisne taster na tastaturi (spust tastera)
- `onkeyup` definiše šta se događa kada korisnik pusti pritisnuti taster na tasta-turi
- ...

Drugi način je da se u kodu JavaScript pristupi odgovarajućem svojstvu elemen-ta. Ovakav način pridruživanja zahteva pristup elementu i definisanje odgovaraju-ćeg svojstva. Vrednost svojstva je anonimna funkcija sa akcijom koja treba da se dogodi.

Naredni primer ilustruje opisana dva pristupa:

Kod 61: primeri/javascript1/06\_pridruzivanje\_osluskivaca.html

```
<!DOCTYPE html>
<html>

<head>
    <title>Pridruživanje događaja elementima</title>
    <meta charset="UTF-8">

    <style type="text/css">
        .dugme {
            width: 200px;
            height: 50px;
            line-height: 50px;
            text-align: center;
            background-color: green;
            color: white;
        }

        .dugme:hover {
            cursor: pointer;
            background-color: blue;
        }

        .unos {
            border: 1px solid black;
            padding-left: 20px;
            height: 50px;
            width: 400px;
            line-height: 50px;
        }
    </style>
</head>
```

```
<body>
    <div class="dugme" onclick="prikazi_poruku();">Klik za pregled poruke</div>
    <br>
    <div id="dugme2" class="dugme">Klik za pregled poruke</div>
    <br>
    <!--
        Atributom contentEditable se omogucava da se sadrzaj elementa div
        menja,
        cime taj element takodje dobija i onblur svojstvo,
        koje podrazumevano ne sadrzi.
    -->
    <div id="dugme3" class="unos" contentEditable>Nakon sto ja izgubim fokus
        , prikazace se poruka</div>

    <script type="text/javascript">
        function prikazi_poruku() {
            window.alert("Poruka!");
        }

        let moderno_dugme = document.querySelector("#dugme2");
        moderno_dugme.onclick = function () {
            prikazi_poruku();
        };

        moderno_dugme = document.querySelector("#dugme3");
        moderno_dugme.onblur = function () {
            prikazi_poruku();
        };
    </script>
</body>

</html>
```

## 7 Čas 9

Cilj časa je osposobljavanje studenta za rad sa HTML formularima — njihovo kreiranje i obradu podataka na klijentskoj strani pomoću jezika JavaScript.

### 7.1 HTML formulari

Da bismo sakupili informacije od korisnika, na raspolaganju nam je veliki izbor HTML elemenata pomoću kojih možemo da omogućimo korisniku da unese podatke ili da odabere neke od predefinisanih podataka. Ovi elementi se nazivaju *elementi formulara*, i obuhvaćeni su u elementu koji predstavlja jedan formular — element `form`. Više o tome kako se podaci iz formulara šalju ka serveru biće reči kada budemo pričali o načinu obrađivanja podataka na serveru.

Najjednostavniji način za unos podataka jeste putem elementa `input`. Njegov prikaz zavisi od vrednosti njegovog atributa `type`:

- Za jednolinijski tekstualni unos koristi se vrednost `text`. Tekst koji se unosi može biti proizvoljan.
- Za unos lozinke koristi se vrednost `password`. Veb pregledač će najčešće prikazati karaktere zvezdice ili tačkice umesto stvarnog teksta koji se unosi.
- Za definisanje proizvoljnog dugmeta koji treba da izvrši neku akciju koristi se vrednost `button`. Vrednost atributa `onclick` definiše JavaScript kod koji će se izvršiti klikom na dugme:

```
<input type="button" value="Click Me!" onclick="alert('Hello World!')">
```

Alternativno, može se koristiti element `button`, sa sličnom sintaksom:

```
<button type="button" onclick="alert('Hello World!')>Click Me!</button>
```

- Za definisanje dugmeta koji će poslati podatke iz formulara ka serveru koristi se vrednost `submit`. Ovaj element ima smisla definisati samo u okviru elementa `form`.
- Za definisanje dugmeta koji briše sve unete podatke iz formulara koristi se vrednost `reset`.
- Za odabir tačno jednog od više ponuđenih izbora koristi se vrednost `radio`. Ovaj element ima smisla definisati samo u okviru elementa `form`. Da bi se znalo za koju opciju se vezuje koji izbor, ovim elementima je potrebno dodeliti atribut `name`. Na primer:

```
<form>
  <input type="radio" name="gender" value="male" checked> Male
  <br>
  <input type="radio" name="gender" value="female"> Female
  <br>
  <input type="radio" name="gender" value="other"> Other
</form>
```

Primetimo da svaki element ima i atribut `value` kojim se definiše izbor za opciju koja se nalazi kao vrednost atributa `name`. Ovaj element predstavlja samo jedno dugme za izbor, ali ne i prateći tekst, te je potrebno dodati opis naknadno, kao u gornjem primeru.

- Za odabir proizvoljnog broja od više ponuđenih izbora koristi se vrednost `checkbox`. Ovaj tip elementa funkcioniše na sličan način kao i prethodni tip. Razlika je u tome što ne moramo da ih vežemo za specifičnu opciju, već svaki izbor može da predstavlja svoju opciju. Na primer:

```
<form>
  <input type="checkbox" name="vehicle1" value="Bike"> I have a bike
  <br>
  <input type="checkbox" name="vehicle2" value="Car"> I have a car
</form>
```

### Zadatak 7.1:

HTML5 standard definiše još više tipova za element `input`. Istražiti definisane tipove na adresi [https://www.w3schools.com/html/html\\_form\\_input\\_types.asp](https://www.w3schools.com/html/html_form_input_types.asp) u sekciji "HTML5 Input Types" i obratiti pažnju na kompatibilnost veb pregledača za svaki od njih.

Element `input` ima i određeni broj atributa specifičnih za njega:

- Kao što smo videli, atributom `name` definišemo naziv opcije.
- Atribut `value` određuje vrednost za to polje.
- Atribut `readonly` određuje da vrednost ne bi trebalo da se menja. Važno je napomenuti da ovaj atribut nema vrednost, već samo njegovih navođenjem specifikujemo da element ima to svojstvo.
- Atribut `disabled` određuje da polje ne prima unos. Njegova vrednost neće biti poslata kao deo podataka iz formulara ka serveru. On se takođe samo navodi, poput prethodnog atributa.
- Atribut `size` definiše veličinu polja (u karakterima).
- Atribut `maxlength` definiše najveći broj karaktera koji element može da primi.

### Zadatak 7.2:

HTML5 standard definiše još više atributa za element `input`. Istražiti definisane attribute na adresi [https://www.w3schools.com/html/html\\_form\\_attributes.asp](https://www.w3schools.com/html/html_form_attributes.asp) u sekciji "HTML5 Attributes" i obratiti pažnju na kompatibilnost veb pregledača za svaki od njih.

Pored elementa `input`, HTML jezik definiše i druge elemente za unos podataka.

Element `select` predstavlja *padajuću listu* (engl. *dropdown list*) sa izborima. Atributom `name` definišemo naziv te opcije, a svaki od izbora je predstavljen elementom `option`, čiji atribut `value` definiše jedan izbor. Elementi `option` se nalaze

kao sadržaj elementa `select`. Sadržaj elementa `option` predstavlja tekst koji se prikazuje prilikom izbora opcije. Na primer:

```
<select name="cars">
    <option value="volvo">Volvo</option>
    <option value="saab">Saab</option>
    <option value="fiat">Fiat</option>
    <option value="audi">Audi</option>
</select>
```

Podrazumevano je prvi element `option` označen kao odabrani, ali to se može redefinisati dodeljivanjem atributa `selected` izboru koji želimo da bude podrazumevano označen:

```
<option value="fiat" selected>Fiat</option>
```

Ako nad elementom `select` postavimo atribut `multiple`, onda je moguće odabratи više opcija pomoću tastera CTRL ili CMD:

```
<select name="cars" multiple>
    <option value="volvo">Volvo</option>
    <option value="saab">Saab</option>
    <option value="fiat">Fiat</option>
    <option value="audi">Audi</option>
</select>
```

Ako želimo da izvršimo grupisanje opcija, to možemo uraditi pomoću elementa `optgroup`, kao u narednom primeru:

```
<select>
    <optgroup label="Swedish Cars">
        <option value="volvo">Volvo</option>
        <option value="saab">Saab</option>
    </optgroup>
    <optgroup label="German Cars">
        <option value="mercedes">Mercedes</option>
        <option value="audi">Audi</option>
    </optgroup>
</select>
```

Postoji i još jedan način za izlistavanje opcija. Moguće je da za `input` elemente definišemo predefinisane vrednosti koje mogu biti odabrane korišćenjem elementa `datalist`. Korišćenjem ovog elementa u kombinaciji sa `input` elementom (koji u tom slučaju mora da ima atribut `list` čija je vrednost identifikator elementa `datalist`), to `input` polje će dobiti opciju automatskog dopunjavanja. Navedimo da ovaj element nije dostupan u Safari pregledačima.

```
<input list="browsers">

<datalist id="browsers">
    <option value="Internet Explorer">
    <option value="Firefox">
    <option value="Chrome">
    <option value="Opera">
    <option value="Safari">
</datalist>
```

Element `textarea` definiše višelinjski unos teksta. Atributima `rows` i `cols` se definiše vidljivi broj vrsta, odnosno, kolona, redom:

```
<textarea name="message" rows="10" cols="30">
    The cat was playing in the garden.
</textarea>
```

Dodatno, veličinu ovog elementa možemo definisati i korišćenjem CSS svojstava `width` i `height`:

```
<textarea name="message" style="width:200px; height:600px">
    The cat was playing in the garden.
</textarea>
```

Za grupisanje više elemenata formulara koji su međusobno povezani u jednu celinu, možemo koristiti element `fieldset`. Ovaj element će nacrtati pravougaonik oko elemenata koje obuhvata. Da bismo anotirali ovakvu jednu celinu, možemo koristiti element `legend`, čiji je tekstualni sadržaj naslov celine koja je definisana `fieldset` elementom.

Još jedan način za semantičko obeležavanje elemenata formulara jeste upotreba elementa `label`. Ovim elementom se definiše oznaka za elemente `button`, `input`, `meter`, `output`, `progress`, `select`, ili `textarea`. Ovaj element se ne renderuje za korisnika kao drugi vizualni elementi, ali omogućava da kada korisnici kliknu na tekstualni sadržaj ovog elementa, odgovarajuća kontrola na koju se odnosi element `label` postaje aktivna. Da bismo povezali element `label` sa nekim elementom formulara, koristimo njegov atribut `for`, čija je vrednost identifikator elementa za kojeg ga vezujemo.

Naredni primer ilustruje korišćenje opisanih elemenata:

Kod 62: primeri/javascript2/01\_formulari.html

```
<!DOCTYPE html>
<html>

<head>
    <title>Rad sa formularima</title>
    <meta charset="UTF-8">

    <link rel="stylesheet" type="text/css" href="bootstrap.css">

    <style type="text/css">
        .greska {
            color: red;
            font-size: 14px;
            margin-top: 25px;
            margin-bottom: 25px;
        }

        fieldset legend {
            color: orange;
        }
    </style>
</head>
```

```
<body>
    <form method="" action="" id="f" class="container" novalidate>
        <!-- Element u okviru kojeg ce biti upisane informacije o
            eventualnim greskama -->
        <div class="greska" id="greska"> </div>

        <fieldset class="form-group">
            <legend> Lični podaci </legend>

            <div class="form-group">
                <label for="ime_prezime"> Ime i prezime*: </label>
                <input id="ime_prezime" type="text" placeholder="Pera Peric"
                    maxlength="30" name="ime_prezime" required
                    class="form-control">
            </div>

            <div class="form-group">
                <label for="datum_rodjenja"> Datum rodjenja: </label>
                <input id="datum_rodjenja" type="date" name="datum_rodjenja"
                    class="form-control">
            </div>

            <div class="form-group">
                <label for="email"> Email: </label>
                <input id="email" name="email" type="email" class="form-control">
            </div>

            <div class="form-group">
                <label for="vеб_adresa"> Veb adresa: </label>
                <input id="vеб_adresa" name="vеб_adresa" autocomplete="off"
                    type="url" class="form-control">
            </div>

            <div class="form-group">
                <label for="username"> Korisnicko ime*: </label>
                <input id="username" type="text" name="korisnicko_ime"
                    required class="form-control">
            </div>

            <div class="form-group">
                <label for="password"> Sifra*: </label>
                <input id="password" name="sifra" required type="password"
                    class="form-control">
            </div>

        </fieldset>

        <fieldset class="form-group">
            <legend> Obrazovanje </legend>

            <div class="form-group row">
                <div class="col-md-6">
```

```
<label for="fakultet" class="control-label"> Fakultet: </
    label>
<select name="fakultet" id="fakultet" class="form-control">
    <option value=""> ----- </option>
    <optgroup label="tehnički fakulteti">
        <option value="matf"> MATF </option>
        <option value="etf"> ETF </option>
        <option value="fon"> FON </option>
    </optgroup>
    <optgroup label="društvene nauke">
        <option value="f"> Filološki fakultet </option>
    </optgroup>
    <optgroup label="umetnost">
        <option value="fdu" selected="selected"> FDU </option>
        >
    </optgroup>
</select>
</div>

<div class="col-md-6">
    <label for="univerzitet" class="control-label"> Univerzitet
        : </label>
    <input list="svi_univerziteti" id="univerzitet" name="
        univerzitet" type="text" class="form-control">
    <datalist id="svi_univerziteti">
        <option value="UBG"> Univerzitet u Beogradu </option>
        <option value="UNS"> Univerzitet u Novom Sadu </option>
        <option value="UNI"> Univerzitet u Nisu </option>
        <option value="UKG"> Univerzitet u Kragujevcu </option>
    </datalist>
</div>
</div>

<div class="form-group">
    Godina studija:

    <div class="form-check form-check-inline">
        <input name="godina" id="godina_1" value="1" type="radio"
            class="form-check-input">
        <label for="godina_1" class="form-check-label">1</label>
    </div>

    <div class="form-check form-check-inline">
        <input name="godina" id="godina_2" value="2" type="radio"
            class="form-check-input">
        <label for="godina_2" class="form-check-label">2</label>
    </div>

    <div class="form-check form-check-inline">
        <input name="godina" id="godina_3" value="3" type="radio"
            class="form-check-input">
        <label for="godina_3" class="form-check-label">3</label>
    </div>
```

```
<div class="form-check form-check-inline">
    <input name="godina" id="godina_4" value="4" type="radio"
           class="form-check-input">
    <label for="godina_4" class="form-check-label">4</label>
</div>
</div>

<div class="form-group">
    Lista interesovanja:

    <div class="form-check">
        <input name="vrb" id="vrb" type="checkbox" checked class="form-check-input">
        <label for="vrb" class="form-check-label">vrb
            programiranje</label>
    </div>

    <div class="form-check">
        <input name="paradigme" id="paradigme" type="checkbox" class="form-check-input">
        <label for="paradigme" class="form-check-label">programske
            paradigme</label>
    </div>

    <div class="form-check">
        <input name="grafika" id="grafika" type="checkbox" class="form-check-input">
        <label for="grafika" class="form-check-label">računarska
            grafika</label>
    </div>

    <div class="form-check">
        <input name="algoritmi" id="algoritmi" type="checkbox" class="form-check-input">
        <label for="algoritmi" class="form-check-label">algoritmi</label>
    </div>
</div>

</fieldset>

<div class="form-group">
    <label for="napomena"> Napomena: </label>
    <br>
    <textarea cols="80" rows="5" name="napomena" id="napomena" class="form-control">Unesite napomenu...</textarea>
</div>

<div class="form-group text-center">
    <input value="posalji podatke" type="submit" class="btn btn-primary">
    <input value="obrisi podatke" type="reset" class="btn btn-danger">
</div>
```

```
>
</div>
</form>

<script type="text/javascript" src="01_formulari_obrađa.js"></script>
</body>

</html>
```

**Lični podaci**

Ime i prezime\*:

Datum rođenja:

Email:

Veb adresa:

Korisnicko ime\*:

Slifra\*:

**Obrazovanje**

Fakultet:  Univerzitet:

Godina studija:  1  2  3  4

Lista interesovanja:

veb programiranje  
 programske paradigme  
 računarska grafika  
 algoritmi

Napomena:

Unesite napomenu...

**Zadatak 7.3:**

U prethodnom primeru su je korišćena biblioteka bootstrap za stilizovanje elemenata formulara pomoću različitih klasa. Istražiti različite klase koje su korišćene u primeru na adresi <https://getbootstrap.com/docs/4.1/components/forms/>.

## 7.2 Obrada podataka u formularu

Sada ćemo videti kako možemo dohvati podatke iz formulara i testirati njihove vrednosti u odnosu na predefinisane domene. Koristićemo prethodno kreirani formular. Za početak, dohvativamo čitav formular:

```
let f = document.querySelector("#f");
```

Na nivou formulara se može definisati događaj `onsubmit` koji se izvršava nakon što korisnik klikne na dugme koje je tipa `submit`. Ukoliko je povratna vrednost funkcije `true`, podaci će biti poslati na adresu naznačenu atributom `action`, a inače neće biti poslati.

```
f.onsubmit = function()
{
    // Pomocna promenljiva
    let polje;

    // U okviru polja za gresku bice upisivane greske
    let greska = document.querySelector("#greska");

    // U nastavku kod ide ovde ...
};
```

Krenimo redom po formularu i ispitujmo svako od polja. Ukoliko naiđemo na neku neregularnost, dovoljno je da vratimo vrednost `false` u ovoj funkciji. Ime i prezime korisnika je obavezno polje. Očekuje se da dužina bude manja od 30, tj. od vrednosti atributa `maxlength`. Kada dohvativamo polje formulara, njegovi atributi su nam dostupni kao svojstva odgovarajućeg objekta. Zato možemo pristupati svojstvima `value` i `maxLength` u narednom fragmentu koda:

```
polje = document.querySelector("#ime_prezime");
let imePrezime = polje.value.trim();
let maxDuzina = polje.maxLength || 30;

if (imePrezime === "" || imePrezime.length > maxDuzina)
{
    greska.textContent = "Nekorektna vrednost u polju za ime i prezime!";
    return false;
}
```

Datum rođenja korisnika treba da bude oblika gggg-mm-dd. Metod `substr`, definisan nad niskama, vraća podnisku date niske i prihvata dva argumenta: prvi je indeks od kojeg podniska počinje, a drugi je broj karaktera, tj. dužina željenje podniske. Funkcija `Number.parseInt` konvertuje broj koji je zapisan kao niska u numeričku vrednost. Slično, dostupna je funkcija `Number.parseFloat`. Ukoliko konverzija ne uspe, rezultat je `Nan`.

```
polje = document.querySelector("#datum_rodjenja");
let datumRodjenja = polje.value;
let godina = parseInt(datumRodjenja.substr(0, 4));
let mesec = parseInt(datumRodjenja.substr(5, 2));
let dan = parseInt(datumRodjenja.substr(8, 2));

if (isNaN(dan) || isNaN(mesec) || isNaN(godina) ||
    dan < 1 || dan > 31 ||
    mesec < 1 || mesec > 12 ||
    godina < 0)
{
    greska.textContent="Nekorektna vrednost u polju za datum rodjenja!";
```

```

        return false;
    }

    if (datumRodjenja.charAt(4) != "-" || datumRodjenja.charAt(7) != "-")
    {
        greska.textContent="Datum rodjenja treba da bude u formatu gggg-mm-dd";
        return false;
    }
}

```

Email adresa korisnika treba da sadrzi karakter @ i barem jednu tačku nakon tog karaktera. Metod `indexOf` vraća prvo pojavljivanje niske koja je zadata kao argument u niski nad kojom se poziva. Ukoliko niska-argument ne postoji u dатој niski, onda funkcija vraća `-1`. Slično, metod `lastIndexOf` vraća poslednje pojavljivanje niske-argumenta.

```

polje = document.querySelector("#email");
let email = polje.value;
let manki = email.indexOf("@");
let poslednjaTackica = email.lastIndexOf(".");

if (manki === -1 || poslednjaTackica === -1 || poslednjaTackica < manki)
{
    greska.textContent="Nekorektna vrednost u polju za email adresu.";
    return false;
}

```

Url adresa korisnika treba da pocinje sa `http://`.

```

polje = document.querySelector("#vеб_adresa");
let vebAdresa = polje.value;

if (vebAdresa.substr(0, 7) != "http://")
{
    greska.textContent="Nekorektna vrednost u polju za veb adresu.";
    return false;
}

```

Korisničko ime korisnika je obavezno polje. Treba da se sastoji samo od malih i velikih slova i da bude dužine barem 5 karaktera. Konstrukcija novog, praznog niza, može se izvršiti pozivom `new Array()`. Funkcija `String.fromCharCode` konstruiše nisku koja sadrži karakter čiji je kod zadat brojem koji se prosleđuje kao argument.

```

polje = document.querySelector("#username");
let korisnickoIme = polje.value.trim();

let malaSlova = new Array();
let velikaSlova = new Array();
for (let i = 0; i < 26; ++i)
{
    malaSlova[i] = String.fromCharCode(97 + i);
    velikaSlova[i] = String.fromCharCode(65 + i);
}

if (korisnickoIme.length < 5)
{
    greska.textContent = "Korisnicko ime nije dovoljno dugo.";
}

```

```

        return false;
    }

for (let i = 0; i < korisnickoIme.length; ++i)
{
    let tekuciKarakter = korisnickoIme.charAt(i);

    if (malaSlova.indexOf(tekuciKarakter) === -1 &&
        velikaSlova.indexOf(tekuciKarakter) === -1)
    {
        greska.textContent = "Nedozvoljeni karakter u polju za korisnicko ime
                            .";
        return false;
    }
}

```

Šifra korisnika je obavezna i mora da sadrži barem dve cifre.

```

polje = document.querySelector("#password");
let sifra = polje.value.trim();

if (sifra === "")
{
    greska.textContent = "Polje za sifru je obavezno.";
    return false;
}

let brojCifara = 0;
for (let i = 0; i < sifra.length; ++i)
{
    let tekuciKarakter = sifra.charAt(i);

    if ("0123456789".indexOf(tekuciKarakter) != -1)
    {
        ++brojCifara;
    }
}

if (brojCifara < 2)
{
    greska.textContent = "Polje za sifru mora da sadrži barem dve cifre.";
    return false;
}

```

Polje za fakultet mora biti odabрано. Da bismo za element `select` dohvatali indeks vrednosti koja je odabrana, možemo nad odgovarajućim objektom dohvatiti svojstvo `selectedIndex`. Ukoliko je ova vrednost jednaka 0, nijedna opcija nije odabrana.

```

polje = document.querySelector("#fakultet");

if (polje.selectedIndex === 0)
{
    greska.textContent = "Odaberite fakultet.";
    return false;
}

```

```
}
```

Godina studija mora biti odabrana. Do sada smo sve elemente formulara dohvatali putem njihovih identifikatora. Umesto ovog pristupa, pokažimo jedan novi CSS selektor koji se češće koristi za elemente formulara, na primer, radio dugmad. Da bismo dohvatili sve `input` elemente koji imaju postavljen odgovarajući atribut, na primer, `name="godina"`, možemo koristiti selektor oblika `input[name='godina']`. Očigledno pošto ovakvih elemenata može biti više<sup>7</sup>, koristićemo metod `querySelectorAll`. Za svako radio dugme možemo ispitati da li je izabrano tako što pristupimo njegovom svojstvu `checked` koje predstavlja Bulovu vrednost.

```
let indikatorGodine = false;
polje = document.querySelectorAll("input[name='godina']");

for (let i = 0; i < polje.length; ++i)
{
    let godina = polje[i];

    if (godina.checked)
    {
        indikatorGodine = true;
        break;
    }
}

if (!indikatorGodine)
{
    greska.textContent = "Godina studija je obavezno polje.";
    return false;
}
```

Polja lista interesovanja i napomena su opciona, tako da smo ovime završili obradu podataka u ovom formularu.

#### Zadatak 7.4:

U prethodnom primeru smo rekli da su interesovanja opciona (primetimo da smo zbog toga koristili `checkbox` elemente). Istražiti kako bismo u JavaScript jeziku nametnuli ograničenje da korisnik mora da odabere makar 2 interesovanja i napisati odgovarajući kod kojim se proverava to ograničenje.

Takođe, možemo i reagovanje na događaj kojim se sadržaj polja u formularu vraćaju na podrazumevana, tako što definišemo metod `onreset`. Slično kao i `onsubmit`, i ovaj metod će obrisati sadržaj u formularu za kojim je vezan ako i samo ako metod vrati `true`. Ukoliko bismo, na primer, želeli da korisnik odabere da li želi zaista da resetuje formular, možemo redefinisati ovaj metod tako što u njemu iskoristimo funkciju `window.confirm` koja će prikazati prozor sa dugmićima "OK" i "Cancel" i tekstrom koji je prosleđen kao argument. Ako korisnik klikne na "OK", funkcija vraća `true`, a inače vraća `false`.

```
f.onreset = function()
```

<sup>7</sup> Zapravo, radio dugmad ne bi imalo smisla koristiti kada bi postojao samo jedan odabir, te ih uvek ima više od jednog.

```
{  
    let odgovor = window.confirm("Da li zelite da ponistite unos?");  
    return odgovor;  
}
```

Celo rešenje je dato narednim kodom:

```
Kod 63: primeri/javascript2/01_formulari_obrađa.js  
let f = document.querySelector("#f");  
  
f.onsubmit = function()  
{  
    // Pomocna promenljiva  
    let polje;  
  
    // U okviru polja za gresku bice upisivane greske  
    let greska = document.querySelector("#greska");  
  
    // Provera za ime i prezime  
    polje = document.querySelector("#ime_prezime");  
    let imePrezime = polje.value.trim();  
    let maxDuzina = polje.maxLength || 30;  
    if (imePrezime === "" || imePrezime.length > maxDuzina)  
    {  
        greska.textContent = "Nekorektna vrednost u polju za ime i prezime!";  
        return false;  
    }  
  
    // Provera za datum rodjenja  
    polje = document.querySelector("#datum_rodjenja");  
    let datumRodjenja = polje.value;  
    let godina = parseInt(datumRodjenja.substr(0, 4));  
    let mesec = parseInt(datumRodjenja.substr(5, 2));  
    let dan = parseInt(datumRodjenja.substr(8, 2));  
  
    if (isNaN(dan) || isNaN(mesec) || isNaN(godina) ||  
        dan < 1 || dan > 31 ||  
        mesec < 1 || mesec > 12 ||  
        godina < 0)  
    {  
        greska.textContent="Nekorektna vrednost u polju za datum rodjenja!";  
        return false;  
    }  
  
    if (datumRodjenja.charAt(4) != "-" || datumRodjenja.charAt(7) != "-")  
    {  
        greska.textContent="Datum rodjenja treba da bude u formatu gggg-mm-  
        dd";  
        return false;  
    }  
  
    // Provera za email  
    polje = document.querySelector("#email");  
    let email = polje.value;
```

```
let manki = email.indexOf("@");
let poslednjaTackica = email.lastIndexOf(".");

if (manki === -1 || poslednjaTackica === -1 || poslednjaTackica < manki
    )
{
    greska.textContent="Nekorektna vrednost u polju za email adresu.";
    return false;
}

// Provera za url adresu
polje = document.querySelector("#vеб_adresa");
let vebAdresa = polje.value;

if (vebAdresa.substr(0, 7) != "http://")
{
    greska.textContent="Nekorektna vrednost u polju za veb adresu.";
    return false;
}

// Provera za korisnicko ime
polje = document.querySelector("#username");
let korisnickoIme = polje.value.trim();

let malaSlova = new Array();
let velikaSlova = new Array();
for (let i = 0; i < 26; ++i)
{
    malaSlova[i] = String.fromCharCode(97 + i);
    velikaSlova[i] = String.fromCharCode(65 + i);
}

if (korisnickoIme.length < 5)
{
    greska.textContent = "Korisnicko ime nije dovoljno dugo.";
    return false;
}

for (let i = 0; i < korisnickoIme.length; ++i)
{
    let tekuciKarakter = korisnickoIme.charAt(i);

    if (malaSlova.indexOf(tekuciKarakter) === -1 &&
        velikaSlova.indexOf(tekuciKarakter) === -1)
    {
        greska.textContent = "Nedozvoljeni karakter u polju za korisnicko
                            ime.";
        return false;
    }
}

// Provera za sifru
polje = document.querySelector("#password");
```

```
let sifra = polje.value.trim();

if (sifra === "")
{
    greska.textContent = "Polje za sifru je obavezno.";
    return false;
}

let brojCifara = 0;
for (let i = 0; i < sifra.length; ++i)
{
    let tekuciKarakter = sifra.charAt(i);

    if ("0123456789".indexOf(tekuciKarakter) != -1)
    {
        ++brojCifara;
    }
}

if (brojCifara < 2)
{
    greska.textContent = "Polje za sifru mora da sadrzi barem dve cifre
    .";
    return false;
}

// Provera za fakultet
polje = document.querySelector("#fakultet");

if (polje.selectedIndex === 0)
{
    greska.textContent = "Odaberite fakultet.";
    return false;
}

// Provera za godinu studija
let indikatorGodine = false;
polje = document.querySelectorAll("input[name='godina']");

for (let i = 0; i < polje.length; ++i)
{
    let godina = polje[i];

    if (godina.checked)
    {
        indikatorGodine = true;
        break;
    }
}

if (!indikatorGodine)
{
    greska.textContent = "Godina studija je obavezno polje.";
```

```

        return false;
    }
};

f.onreset = function()
{
    let odgovor = window.confirm("Da li zelite da ponistite unos?");
    return odgovor;
};

```

**Zadatak 7.5:**

Napisati JavaScript kod koji bi "u hodu" proveravao da li je korisničko ime dužine više od 5 karaktera. Drugim rečima, potrebno je da, nakon što korisnik unese korisničko ime nedozvoljene dužine i pređe na naredno polje, veb pregledač prikaže poruku "Korisnicko ime mora biti duže od 5 karaktera!" funkcijom `window.alert()`.

Pomoć: Kada korisnik unese korisničko ime, a zatim pređe na naredno polje, polje za unos korisničkog imena gubi fokus (videti deo o osluškivačima).

**Zadatak 7.6:**

Napisati HTML datoteku koja sadrži formular dat na narednoj slici

Površina trougla

Unesite stranice trougla:

a:

b:

c:

p:

[Izračunaj površinu](#)

Napisati JavaScript kod koji nakon klika na dugme "Izracunaj povrsinu" izračunava i ispisuje vrednost površine trougla. U slučaju unosa nekorektnih vrednosti, treba ispisati poruku o grešci u obaveštajnom prozoru veb pregledača. Za računanje površine koristiti Heronov obrazac. Formular doterati korišćenjem Bootstrap biblioteke.

## 8 Čas 10

Cilj časa je upoznavanje studenta sa asinhronim modelom programiranja u programskom jeziku JavaScript, konceptom po-potrebi-pozivnih funkcija i kreiranja asinhronih zahteva ka udaljenim resursima. Takođe, na ovom času ćemo dati detaljan opis odabranih elemenata biblioteke jQuery, a zatim će uslediti mnogobrojni primeri koji ilustruju njihovu upotrebu.

### 8.1 Asinhrono programiranje

U *sinhronom* modelu programiranja, stvari se dešavaju sekvencionalno, jedna za drugom. Kada pozovemo funkciju koja izvršava neku akciju, ona se vraća tek kada je akcija završena i tada može da vrati neki rezultat. Ovim se program stopira za ono vreme koliko je bilo potrebno toj akciji da se završi.

*Asinhroni* model programiranja dozvoljava više stvari da se dešavaju u isto vreme. Kada pokrenemo neku akciju, program nastavlja sa radom. Onog trenutka kada akcija završi, program biva informisan o završetku akcije i dobija pristup rezultatu.

Poređenje između sinhronog i asinhronog programiranja se može jednostavno sagledati kroz naredni primer. Neka je potrebno napisati program koji dohvata dva resursa sa interneta, a zatim kombinuje rezultate. U sinhronom modelu, najjednostavniji način jeste da se zahtevi za resursima vrše jedan za drugim<sup>8</sup>. Problem ovog pristupa jeste da se drugi zahtev započinje tek onda kada se prvi završi. Ukupno vreme rada programa je najmanje zbir trajanja dohvatanja odgovora. U asinhronom modelu, moguće je poslati dva zahteva jedan za drugim, a zatim u trenutku kada su obe vrednosti dostupne, vrši se kombinovanje rezultata. Prednost ovog modela je u tome što se preklapa vreme potrebno za dohvatanje dva resursa, čime se značajno ubrzava rad programa.

#### 8.1.1 Po-potrebi-pozivne funkcije

Jedan pristup asinhronom programiranju jeste da se funkcije koje izvršavaju duge ili spore akcije konstruišu tako da prihvataju dodatni argument, *po-potrebi-pozivnu funkciju* (engl. *callback function*). U tom slučaju se akcija započinje i onoga trenutka kada se završi, poziva se po-potrebi-pozivna funkcija sa rezultatom akcije.

Kao primer ovog modela, možemo razmotriti funkciju `setTimeout`, dostupnu i u veb pregledaćima i u Node.js platformi, koja prihvata dva argumenta: po-potrebi-pozivnu funkciju i broj milisekundi — funkcija postavlja tajmer koji traje prosleđeni broj milisekundi i po isteku tajmera, poziva se po-potrebi-pozivna funkcija:

```
console.log("Start!");
setTimeout(function()
{
    console.log("Tick");
}, 1000);
```

Ako otvorimo konzolu, prvo što ćemo primetiti jeste da se ispiše:

```
Start!
```

Nakon jedne sekunde, ispis se menja u

<sup>8</sup> Postoje i konkurentni sistemi u sinhronom modelu programiranja o kojima neće biti reči u tekstu.

```
Start!
Tick
```

Vidimo da se po-potrebi-pozivna, anonimna funkcija koju smo prosledili funkciji `setTimeout` izvršila asinhrono, u ovom slučaju, tek nakon što je istekao tajmer koji je postavila funkcija `setTimeout`.

Slično, metod `setInterval` omogućava učestalo izvršavanje zadate po-potrebi-pozivne funkcije kao prvi argument u pravilnim vremenskim razmacima čija se perioda zadaje drugim argumentom. Za upravljanje ovim pozivima koristi se povratna vrednost metoda. Pre svega, pozivom metoda `clearInterval`, kojem se prosleđuje povratna vrednost metoda `setInterval`, može se prekinuti dalje ponavljanje poziva. Na primer:

```
let si = setInterval(function()
{
    console.log("Tick!");
}, 1000);

setTimeout(function()
{
    clearInterval(si);
}, 5500);
```

Rezultat ispisa:

```
Tick!
Tick!
Tick!
Tick!
Tick!
```

### 8.1.2 XMLHttpRequest objekat

Da bismo poslali asinhroni HTTP zahtev ka nekom resursu<sup>9</sup>, potrebno je kreirati objekat klase `XMLHttpRequest`, otvoriti url ka resursu, i poslati zahtev. Nakon što se transakcija završi, objekat će sadržati korisne informacije poput tela HTTP odgovora i statusnog koda. Ovaj objekat prolazi kroz razna stanja, kao što je "otvorena konekcija", "finalno stanje" i dr. Svako stanje ima svoj kod.

**Slanje zahteva** Kao što smo rekli, prvo je potrebno kreirati objekat, koji se inicijalizuje u stanje "UNSET" (kod je 0):

```
let xhr = new XMLHttpRequest();
```

Zatim je potrebno formirati HTTP zahtev pozivom metoda `open`. Njegovi argumenti su: (1) HTTP metod koji se koristi, (2) URL resursa, (3) Bulova vrednost koja označava da li se zahtev vrši sinhrono ili asinhrono (podrazumevana vrednost je `true`, što označava asinhronost). Metod ima i dodatne argumente za korisničko ime i lozinku.

```
xhr.open('GET', 'http://api.icndb.com/jokes/random');
```

<sup>9</sup>Ovakvi zahtevi se još nazivaju i *AJAX* zahtevi, skr. od *Asynchronous JavaScript And XML*.

U ovom trenutku se mogu postaviti zaglavja HTTP zahteva pomocu `setRequestHeader` metode, na primer, `xhr.setRequestHeader(imePolja, vrednost)`. Zatim je potrebno dodati po-potrebi-pozivne funkcije koje će vršiti nekakvu obradu pristiglog odgovora ili, ono što se često radi, obradu pri promeni stanja asinhronog zahteva. Odgovor zahteva je sadržan u svojstvima `response`, `responseText` ili `responseXML` u zavisnosti od tipa odgovora. U slučaju greške, svojstvo `statusText` sadrži statu-snu poruku koja odgovara HTTP statusnoj poruci. Na primer:

```
xhr.onload = function()
{
    // Proveravamo da li je odgovor servera sa vrednoscu 200 preko polja
    // zahteva status
    if (xhr.status === 200)
    {
        console.log(xhr.response);
    }
    else
    {
        console.error(xhr.statusText);
    }
};
```

Događaj `onload` se izvršava kada objekat pređe u finalno stanje "DONE" (kod je 4). Događaj `onerror` se izvršava kada dođe do greske prilikom zahteva:

```
xhr.onerror = function()
{
    console.error('Problem prilikom slanja zahteva');
};
```

Alternativa je moguća sa `onreadystatechange` osluskivacem koji će pozvati po-potrebi-pozivnu funkciju prilikom svake promene stanja:

```
xhr.onreadystatechange = function()
{
    switch(xhr.readyState)
    {
        case XMLHttpRequest.DONE:
            if (xhr.status === 200)
            {
                console.log(xhr.response);
            }
            else
            {
                console.error(xhr.statusText);
            }
    }
};
```

Konačno, potrebno je poslati HTTP zahtev, što se vrši metodom `send`. U zavisnosti od tipa zahteva, opcioni argument metode `send` predstavlja telo zahteva.

```
xhr.send();
```

Iako jednostavan, ovaj primer ilustruje važan koncept, a to je kreiranje komunikacije između klijentskih i serverskih aplikacija korišćenjem XMLHttpRequest objekta. Na ovu temu ćemo se osvrnuti još jednom, kada se budemo upoznali sa kreiranjem naših serverskih aplikacija.

## 8.2 Biblioteka jQuery

jQuery je biblioteka za JavaScript jezik čija je osnovna upotreba olakšavanje razvoja čestih funkcionalnosti u jeziku JavaScript. jQuery objedinjuje često korišćene poslove koji zahtevaju mnogo linija koda u metode koji se pozivaju jednom linijom. Takođe, jQuery pojednostavljuje mnoge komplikovane stvari iz JavaScript-a kao što su AJAX pozivi i DOM manipulacije. jQuery biblioteka sadrži sledeće:

- Metode za upravljanje DOM stablom
- Metode za upravljanje CSS stilovima
- HTML metode zasnovane na događajima
- Efekte i animacije
- Metode za kreiranje AJAX zahteva
- Druge pomoćne metode

Iako postoje mnoge biblioteke za razvoj u jeziku JavaScript, jQuery je jedna od najkorišćenijih, najpopularnijih i najviše se širi. Neke od velikih kompanija koje koriste jQuery su Google, Microsoft, IBM, Netflix, i dr. Jedna od osobina jQuery biblioteke jeste ta da će raditi isto u svim većim pregledačima, pa čak i u Internet Explorer 6.

### 8.2.1 Dodavanje jQuery-ja

Postoji više načina kako koristiti jQuery. Moguće je:

- preuzeti jQuery biblioteku sa [jQuery.com](https://jquery.com)
- umetnuti jQuery sa mreže za dostavljanje sadržaja (engl. *Content Delivery Network*, skr. CDN), poput Google CDN.

Postoje dve verzije jQuery-ja dostupne za preuzimanje:

- Production verzija — koristi se za veb sajtove koji su podignuti i rade (zato što je sve minimizovano i kompresovano)
- Development verzija — koristi se za testiranje i razvoj (ceo kod je nekomprimovan i čitljiv)

Obe verzije su dostupne za preuzimanje sa [jQuery.com](https://jquery.com).

Biblioteku jQuery čini jedinstvena JavaScript datoteka, i može se umetnuti pomoću elementa `script`, kao i svaka druga JavaScript datoteka:

```
<body>
  <script type="text/javascript" src="jquery-3.3.1.min.js"></script>
</body>
```

Ukoliko ne želimo da preuzmemo i hostujemo jQuery, moguće je umetnuti ga sa nekog CDN-a. Da bismo koristili jQuery na ovakav način, potrebno je umetnuti sledeće:

```
<body>
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.
        min.js"></script>
</body>
```

ili

```
<body>
    <script src="https://ajax.aspnetcdn.com/ajax/jQuery/jquery-3.3.1.min.js"
        ></script>
</body>
```

Prednost korišćene ovakvog načina jeste u tome što je većina korisnika verovatno već preuzela jQuery sa Google-a ili Microsoft-a prilikom pretraživanja nekog drugog veb sajta. Kao rezultat toga, jQuery će već biti učitan iz keša, što dovodi do bržeg učitavanja. Takođe, većina CDN-a će se postarati da, kada korisnik zatraži fajl sa njega, taj fajl bude preuzet sa servera koji je najbliži njima, što takođe dovodi do bržeg učitavanja.

### 8.2.2 Osnovna jQuery sintaksa

jQuery sintaksa je osmišljena za označavanje HTML elemenata i njihovo obradovanje. Osnovna sintaksa jQuery-ja jeste: `$(selektor).akcija()`:

- Korišćenjem znaka `$` pristupamo jQuery biblioteci
- `(selektor)` služi da pronađe HTML element(e)
- `akcija()` služi da se primeni na element(e)

Neki jednostavni primeri upotrebe dati su narednim primerima:

```
$(this).hide(); // sakriva trenutni element
$("p").hide(); // sakriva sve "p" elemente
$(".test").hide(); // sakriva sve elemente koji imaju atribut "class"
                  postavljen na vrednost "test"
$("#test").hide(); // sakriva sve elemente koji imaju atribut "id"
                  postavljen na vrednost "test"
```

### 8.2.3 "Dokument je spreman" događaj

S obzirom da veb pregledač prikazuju stranicu tokom parsiranja HTML dokumenta, može se desiti da želimo da pristupimo, na primer, tekstualnoj vrednosti nekog elementa, a da taj element još uvek nije parsiran i dodat u DOM stablo. Zbog toga može doći do neočekivanih grešaka. Ukoliko želimo da se osiguramo da se neki jQuery kod ne izvršava dok dokument ne završi sa učitavanjem (odnosno, dok ne bude "spreman za obradu"), možemo naš jQuery kod smestiti unutar osluškivača događaja `ready` koji se okida nad objektom `document`:

```
$(document).ready(function(){
    // jQuery metode idu ovde...
});
```

Dobra je praksa sačekati da se dokument učita pre nego što se počne sa radom na njemu. Opisani pristup takođe omogućuje da se JavaScript kod piše u zaglavlju dokumenta, bez brige o tome da li će se on ispravno izvršiti. Neki od primera gde može doći do greške ukoliko su metodi pokrenuti pre nego što je dokument učitan su:

- Pokušavanje da se sakrije element koji još nije kreiran
- Pokušavanje dobijanja veličine slike koja još nije učitana
- ...

Postoji još jedna, skraćena varijanta prethodnog koda koja daje isti rezultat, ali daje slabije čitljiv kod:

```
$(function(){
    // jQuery metode idu ovde...
});
```

#### 8.2.4 jQuery selektori

jQuery selektori omogućavaju selektovanje i upravljanje HTML elementima. Selektori traže HTML elemente na osnovu njihovih identifikatora, klase, atributa, vrednosti atributa i dr. Bazirani su na CSS selektorima, ali postoje još neki. Svi selektori u jQuery-ju počinju sa znakom dolara i zagradama: `$()`.

U nastavku dajemo spisak najkorišćenijih jQuery selektora i njihovih opisa:

- Element selektor — elemente možemo selektovati na osnovu njihovih imena. Sve `p` elemente možemo selektovati pomoću  `$("p")`.
- Selektor identifikatora — ovaj selektor koristi `id` atribut HTML elementa da bi ga pronašao. Element sa atributom `id="test"` možemo dohvatiti pomoću  `$("#test")`.
- Selektor klase — ovaj selektor koristi `class` atribut HTML elementa da bi ga pronašao. Element sa atributom `class="test"` možemo dohvatiti pomoću  `$(".test")`.
- Ostali selektori
  - Selektor `">*` selektuje sve elemente.
  - Selektor `this` selektuje element na koji se odnosi funkcija koja ga obuhvata.
  - Selektor `$(".p.intro")` selektuje sve `p` elemente sa atributom `class="intro"`.

- Selektor `$("p:first")` selektuje prvi `p` element.
- Selektor `$("ul li:first")` selektuje prvi `li` element prvog `ul` elemen-ta.
- Selektor `$("ul li:first-child")` selektuje prvi `li` element svakog `ul` elementa.
- Selektor `$("[href]")` selektuje sve elemente koje imaju postavljen atri-but `href`.
- Selektor `$("a[target='_blank']")` selektuje sve elemente `a` sa atribu-tom `target="_blank"`.
- Selektor `$("a[target!='_blank']")` selektuje sve elemente `a` koji nema-ju atribut `target="_blank"`.
- Selektor `$(":button")` selektuje sve `button` elemente, kao i `input` ele-mente sa atributom `type="button"`.
- Selektor `$("tr:even")` selektuje sve parne `tr` elemente.
- Selektor `$("tr:odd")` selektuje sve neparne `tr` elemente.
- Selektor `$("div:gt(n)")` selektuje sve elemente `div` koji su na poziciji većoj od `n`. Numeracija počinje od 0.
- Selektor `$("div:lt(n)")` selektuje sve elemente `div` koji su na poziciji manjoj od `n`. Numeracija počinje od 0.
- Selektor `$("div:eq(n)")` selektuje sve elemente `div` koji su na poziciji `n`. Numeracija počinje od 0.
- Selektor ":checked" selektuje sve elemente koji imaju selektovana polja.

Metod `is(selector)` proverava da li element nad kojim se poziva zadovoljava prosledjeni argument `selector`. Na primer, za proveru da li je checkbox dugme sa identifikatorom "odaberi" označeno, možemo koristiti:

```
if ($("#odaberi").is(":checked"))  
{  
    console.log("Polje je odabrano");  
}  
else  
{  
    console.log("Polje nije odabrano");  
}
```

### 8.2.5 jQuery metodi zasnovani na događajima

I u jQuery biblioteci su dostupni različiti događaji koje korisnik može da inicira. Neki od čestih DOM događaja su dati narednom tabelom:

Događaji miša	Događaji tastature	Događaji formulara	Događaji dokumenta/prozora
click	keypress	submit	load
dblclick	keydown	change	resize
mouseenter	keyup	focus	scroll
mouseleave		blur	unload

Većina DOM metoda sa kojima smo se upoznali ima svoj jQuery-ekvivalentan metod. Da bismo pridružili metod događaju `click`, dovoljno je proslediti metod odgovarajućoj jQuery metodi kao argument:

```
$("p").click(function(){
    // Ovde ide akcija...
});
```

Često korišćeni jQuery metodi za definisanje odgovarajućih akcija su:

- `$(document).ready(callback)` — dozvoljava da se izvrši funkcija `callback` nakon što se učita ceo dokument
- `click(callback)` — funkcija `callback` se izvršava kada korisnik klikne na element
- `dblclick(callback)` — funkcija `callback` se izvršava kada korisnik dvostruko-klikne na element
- `mouseenter(callback)` — funkcija `callback` se izvršava kada kurSOR pokaže na element
- `mouseleave(callback)` — funkcija `callback` se izvršava kada se izgubi fokus kursora sa elementa
- `mousedown(callback)` — funkcija `callback` se izvršava kada je levi klik miša pritisnut dok je kurSOR na elementu
- `mouseup(callback)` — funkcija `callback` se izvršava kada je levi klik miša podignut nakon pritiska
- `hover(callback1, callback2)` — metod koji koristi dve funkcije i kombinacija je `mouseenter()` i `mouseleave()` metoda. Funkcija `callback1` se izvršava kada kurSOR uđe na element, a funkcija `callback2` se izvršava kada kurSOR napusti element. Na primer:

```
$("#p1").hover(function()
{
    alert("You entered p1!");
},
function()
{
    alert("Bye! You now leave p1!");
});
```

- `focus(callback)` — koristi se nad elementima čiji se sadržaj može menjati, na primer, polja formulara. Funkcija `callback` se izvršava kada element dobije fokus. Na primer:

```
$( "input" ).focus(function()
{
    $(this).css("background-color", "#cccccc");
});
```

- `blur(callback)` — koristi se nad elementima čiji se sadržaj može menjati, na primer, polja formulara. Funkcija `callback` se izvršava kada se sa elementa napusti fokus. Na primer:

```
$( "input" ).blur(function()
{
    $(this).css("background-color", "#aaaaaa");
});
```

- `change(callback)` — osluškivač nad ovim događajem se može pridružiti elementima `input`, `textarea` i `select` i odnosi se na promenu vrednosti ovih polja. Svaki put kada se promeni vrednost polja, poziva je funkcija `callback`. Na primer:

```
$( "input:checkbox" ).change(function()
{
    console.log($(this).val());
});
```

Postoji jedan specijalan metod kojim je moguće pridružiti jedan ili više događaja selektovanom elementu — metod `on(eventString, callback)`. Njegov prvi argument je naziv događaja, a drugi argument je po-potrebi-pozivna funkcija koja će se izvršiti kada se taj događaj okine nad selektovanim elementom. Na primer, pridruživanje `click` događaja elementu `p` može se izvršiti na sledeći način:

```
$( "p" ).on("click", function(){
    $(this).hide();
});
```

Za pridruživanje više događaja, koristimo varijantu metoda `on(callbacksObj)`, koji kao argument prihvata objekat čija su svojstva nazivi događaja, a vrednosti tih svojstava odgovarajuće po-potrebi-pozivne funkcije koje će se izvršiti kada se okinu događaji koji su zadati svojstvima. Na primer:

```
$( "p" ).on({
    mouseenter: function(){
        $(this).css("background-color", "lightgray");
    },
    mouseleave: function(){
        $(this).css("background-color", "lightblue");
    },
    click: function(){
        $(this).css("background-color", "yellow");
    }
});
```

Da bismo ukloniti neki osluškivac sa elementa, možemo koristiti metod `off(event)`, kojem prosleđujemo naziv događaja koji osluškujemo, na primer, `"click"`.

### 8.2.6 jQuery efekti i animacije

Za sakrivanje i prikazivanje elementa, mogu se koristiti metodi

```
$(selector).hide(speed, callback);  
$(selector).show(speed, callback);
```

gde je `speed` brzina sakrivanja/prikazivanja elementa, koja može da uzima neku od predefinisanih vrednosti `"slow"` i `"fast"` ili broj milisekundi. Opcioni parametar `callback` je po-potrebi-pozivna funkcija koja se poziva nakon izvršenja metoda.

Ova dva metoda je moguće kombinovati metodom

```
$(selector).toggle(speed, callback);
```

koji uzima iste parametre kao i prethodni metodi. Ukoliko je element bio vidljiv, nakon izvršenja ovog metoda, element će biti sakriven. Ukoliko je element bio sakriven, nakon izvršenja ovog metoda, element će biti vidljiv.

Za pravljenje prelaza tokom sakrivanja i prikazivanja elementa, mogu se koristiti metodi

```
$(selector).fadeIn(speed, callback);  
$(selector).fadeOut(speed, callback);  
$(selector).fadeToggle(speed, callback);  
$(selector).fadeTo(speed, opacity, callback);
```

koji imaju iste parametre kao i prethodni metodi, sa izuzetkom poslednjeg koji ima obavezan parametar `opacity` koji uzima vrednost u pokretnom zarezu između 0 i 1. Njegov smisao je da odredi do kog nivoa se vrši prelaz.

Za pravljenje efekata "slajdera", mogu se koristiti metodi

```
$(selector).slideDown(speed, callback);  
$(selector).slideUp(speed, callback);  
$(selector).slideToggle(speed, callback);
```

jQuery metod `animate` koristi se za pravljenje korisnički definisanih animacija. TO znači da pomoću ovog metoda možemo konstruisati najrazličitije animacije. Sintaksa poziva ovog metoda glasi:

```
$(selector).animate({params}, speed, callback);
```

Obavezni parametar `params` definiše CSS svojstva koja se animiraju. Ovaj parametar uzima vrednost objekta čija su svojstva nazivi CSS svojstava koji će biti primenjeni nad elementom, a vrednosti tih svojstava su vrednosti odgovarajućih CSS svojstava — element nad kojim se metod primenjuje će na kraju animacije imati dodeljene vrednosti za data CSS svojstva. Opcioni parametar `speed` specifičkuje dužinu trajanja animacije. Opcioni parametar `callback` je po-potrebi-pozivna funkcija koja se izvršava nakon završetka animacije. Naredni primer demonstrira upotrebu ovog metoda tako što pomera `div` element udesno, sve dok nije udaljen od leve ivice za 250px:

```
$( "button" ).click(function(){
    $( "div" ).animate({left: '250px'});
});
```

Moguće je menjati više svojstava odjednom:

```
$( "button" ).click(function(){
    $( "div" ).animate({
        left: '250px',
        opacity: '0.5',
        height: '150px',
        width: '150px'
    });
});
```

Bitno je zapamtiti da ukoliko želimo da menjamo CSS svojstva koja su imenovana sa karakterom `-` – u CSS jeziku, ovde moramo da ih pišemo u kamiljoj notaciji. Tako da, na primer, pišemo `paddingLeft` umesto `padding-left`.

Vrednosti koje se animiraju su jednostavne numeričke vrednosti. To znači da, na primer, animiranje boja nije podržano u jezgru jQuery biblioteke. Ukoliko želimo da animiramo boje, potrebno je preuzeti jQuery.Color sa <https://github.com/jquery/jquery-color>. Više informacija je dostupno na <https://api.jquery.com/animate/>.

Moguće je korišćenje relativnih vrednosti korišćenjem `+=` ili `-=` ispred vrednosti pri čemu se dodaje ili oduzima nova vrednost od stare vrednosti. Na primer:

```
$( "button" ).click(function(){
    $( "div" ).animate({
        left: '250px',
        height: '+=150px',
        width: '+=150px'
    });
});
```

### 8.2.7 Korišćenje funkcionalnosti reda

Po pravilu, jQuery dolazi sa funkcionalnošću reda za animiranje. To znači da ukoliko napišemo više `animate()` poziva jedan iza drugog, jQuery napravi "interni" red od ovih poziva. Animacije se potom izvršavaju jedna po jedna. Neki od primera koji ovo ilustruju su:

```
$( "button" ).click(function(){
    var div = $( "div" );
    div.animate({height: '300px', opacity: '0.4'}, "slow");
    div.animate({width: '300px', opacity: '0.8'}, "slow");
    div.animate({height: '100px', opacity: '0.4'}, "slow");
    div.animate({width: '100px', opacity: '0.8'}, "slow");
});
i
$( "button" ).click(function(){
    var div = $( "div" );
    div.animate({left: '100px'}, "slow");
    div.animate({fontSize: '3em'}, "slow");
});
```

u kojem se prvo pomera `div` element udesno, pa tek onda se uvećava veličina teksta.

`jQuery().stop()` metod se koristi za zaustavljanje animacije pre nego što se završi. Metod `stop()` radi za sve jQuery metode koji implementiraju neki efekat, pa i za slajdere, prelaze i korisničko animiranje. Sintaksa poziva ovog metoda je

```
$(selector).stop(stopAll, goToEnd);
```

Opcioni `stopAll` parametar specifikuje da li će se animacijski red isprazniti ili ne. Po pravilu je `false`, što znači da će se samo aktivna animacija zaustaviti, a da će sve naredne animacije nastaviti sa izvršavanjem, regularno. Opcioni `goToEnd` parametar specifikuje da li će se u momentu zaustavljanja animacija dovršiti do kraja i po pravilu je `false`. Dakle, podrazumevano, metod `stop` samo zaustavlja trenutnu animaciju koja se vrši na zadatom elementu. Primer rada metoda bez parametara:

```
$("#stop").click(function(){
    $("#panel").stop();
});
```

### 8.2.8 Po-potrebi-pozivne funkcije i animacije

JavaScript naredbe se izvršavaju linija po liniju. Ipak, sa efektima, naredna linija koda se vrši iako efekat možda još nije završen. Ovo može dovesti do grešaka u radu. Da bi se ovo spričilo, koriste se po-potrebi-pozivne funkcije. Tipična sintaksa je:

```
$("button").click(function(){
    $("p").hide("slow", function(){
        alert("The paragraph is now hidden");
    });
});
```

Naredni primer ne koristi po-potrebi-pozivnu funkciju u telu funkcije-argumenta za `click`, pa će se prvoručić prikazati pre nego što je efekat sakrivanja završen, što je nekorektno:

```
$("button").click(function(){
    $("p").hide(1000);
    alert("The paragraph is now hidden");
});
```

### 8.2.9 Lančanje metoda

Do sada smo sve jQuery naredbe pisali jednu ispod druge. Ipak, postoji tehnika koja se naziva *lančanje* (engl. *chaining*), koja nam omogućava da pokrenemo više jQuery komandi, jednu za drugom, nad istim elementom (ili nad istim elementima). Na ovaj način pregledači ne moraju da traže iste elemente više od jedanput.

Da bismo ulančali akciju, potrebno je samo nadovezati novu akciju na prethodnu akciju. Sledeći primer ilustruje lančanje metoda `css`, `slideUp` i `slideDown`. Element sa identifikatorom "p1" se prvo menja u crvenu boju, pa zatim pomera naviše, pa na kraju naniže:

```
$("#p1").css("color", "red").slideUp(2000).slideDown(2000);
```

Pošto nadovezivanjem jedna linija koda može da postane veoma duga, možemo da je formatiramo kako zelimo, na primer:

```
$("#p1").css("color", "red")
    .slideUp(2000)
    .slideDown(2000);
```

što daje isti efekat.

### 8.2.10 Upravljanje DOM stablom

Jedan veoma važan aspekt jQuery biblioteke jeste upravljanje DOM stablom. Metodi koji nam ovo omogućavaju su:

- `text()` i `text(text)` — dohvata ili postavlja tekstualan sadržaj selektovanog elementa
- `html()` i `html(htmlString)` — dohvata ili postavlja HTML sadržaj selektovanog elementa
- `val()` i `val(value)` — dohvata ili postavlja vrednost elementa formulara
- `attr(attributeName)` i `attr(attributeName, value)` — dohvata ili postavlja vrednost atributa `attributeName`

Primeri korišćenja ovih metoda dati su u nastavku:

```
// Postavljanje vrednosti

$("#btn1").click(function(){
    $("#test1").text("Hello world!"); // postavlja text
});
$("#btn2").click(function(){
    $("#test2").html("<b>Hello world!</b>"); // postavlja HTML
});
$("#btn3").click(function(){
    $("#test3").val("Dolly Duck"); // postavlja vrednost polja formulara
});

// Dohvatanje vrednosti

$("#btn1").click(function(){
    alert("Text: " + $("#test").text()); // vraca text
});
$("#btn2").click(function(){
    alert("HTML: " + $("#test").html()); // vraca HTML
});
$("#btn1").click(function(){
    alert("Value: " + $("#test").val()); // vraca vrednost polja forme
});
$("button").click(function(){
    alert($("#w3s").attr("href")); // vraca vrednost atributa href
});
```

Da bismo dodali novi sadržaj, možemo koristiti metode:

- `append(content)` — dodaje sadržaj na kraj selektovanog elementa
- `prepend(content)` — dodaje sadržaj na početak selektovanog elementa
- `after(content)` — dodaje sadržaj nakon selektovanog elementa
- `before(content)` — dodaje sadržaj ispred selektovanog elementa

Sva četiri metoda mogu uzimati veći broj parametara da bi kreirali nov sadržaj.  
Na primer:

```
function appendText() {
    var txt1 = "<p>Text.</p>"; // Create element with HTML
    var txt2 = $("<p></p>").text("Text."); // Create with jQuery
    var txt3 = document.createElement("p"); // Create with DOM
    txt3.innerHTML = "Text.";
    $("p").append(txt1, txt2, txt3); // Append the new elements
}
```

ili

```
function afterText() {
    var txt1 = "<b>I </b>"; // Create element with HTML
    var txt2 = $("<i></i>").text("love "); // Create with jQuery
    var txt3 = document.createElement("b"); // Create with DOM
    txt3.innerHTML = "jQuery!";
    $("img").after(txt1, txt2, txt3); // Insert new elements after <img>
}
```

Da bismo obrisali stari sadržaj, možemo koristiti:

- `remove(content)` — briše selektovani element kao i svu njegovu decu
- `empty()` — briše svu decu selektovanog elementa

Ukoliko ipak želimo da filtriramo šta želimo da obrišemo, onda možemo proslediti parametar metodu `remove`, koji predstavlja jQuery selektor. Na primer, da bismo obrisali sve `p` elemente klase "test", možemo koristiti:

```
$( "p" ).remove( ".test" );
```

Ukoliko želimo da obrišemo još i sve `p` elemente klase "demo", možemo koristiti:

```
$( "p" ).remove( ".test, .demo" );
```

Biblioteka jQuery sadrži veliki broj metoda za prolanaženje elemenata u DOM stablu. Uglavnom se počinje od jednog elementa, pa se prolazi kroz ostale dok se ne dođe do onog koji se traži. U ovoj pretrazi, možemo prolaziti naviše (obilazeći pretke), naniže (obilazeci potomke) ili u istom nivou (obilazeći braću). Neki od metoda za prolazak kroz stablo su:

- `parent(selector)` — vraća direktnog pretka selektovanog elementa. Može se koristiti opcioni parametar `selector` za filtriranje predaka.
- `parents(selector)` — vraća sve pretke do korena (`html` elementa). Može se koristiti opcioni parametar `selector` za filtriranje predaka.

- `parentsUntil(selector, filter)` — vraća sve pretke između selektovanog elementa i elementa koji se zadaje kao argument `selector`. Može se koristiti opcionalni parametar `filter` za filtriranje predaka.
- `children(selector)` — vraća svu direktnu decu selektovanog elementa. Može se koristiti opcionalni parametar `selector` za filtriranje dece.
- `find(selector)` — vraća sve potomke do listova, filtrirane po argumentu `selector`.
- `siblings(selector)` — vraća sve rođake u stablu selektovanog elementa. Može se koristiti opcionalni parametar `selector` za filtriranje rođaka.
- `next(selector)` — vraća sledećeg rođaka selektovanog elementa. Može se koristiti opcionalni parametar `selector` za filtriranje rođaka.
- `nextAll(selector)` — vraća sve sledeće rođake selektovanog elementa. Može se koristiti opcionalni parametar `selector` za filtriranje rođaka.
- `nextUntil(selector, filter)` — vraća sve sledeće rođake između selektovanog elementa i elementa koji se zadaje kao argument `selector`. Može se koristiti opcionalni parametar `filter` za filtriranje rođaka.
- `prev()`, `prevAll()` i `prevUntil()` — imaju iste uloge kao prethodne tri metode, samo što vraćaju prethodne rođake umesto naredne.

S obzirom da ove metode vraćaju, na primer, niz svih očeva elemenata `p` (ukoliko je pozvan metod `$("p").parent()`), potrebno je ući u dalju pretragu u ovom nizu. Za to nam služe metodi:

- `first()` — vraća prvi element među selektovanim elementima.
- `last()` — vraća poslednji element među selektovanim elementima.
- `eq(index)` — vraća element koji se nalazi na poziciji u nizu čiji je indeks zadat kao argument metoda. Mogu se zadati negativni brojevi da bi se pretraga vršila od kraja niza.
- `filter(selector)` — vraća sve elemente koji imaju specifikaciju koja se zadaje kao argument metode.
- `not()` — vraća sve elemente koji nemaju specifikaciju koja se zadaje kao argument metode.

### 8.2.11 Upravljanje stilovima

jQuery poseduje nekoliko metoda za upravljanje CSS klasama. Neki od njih su:

- `addClass(className)` — dodaje jednu ili više klase selektovanom elementu
- `removeClass(className)` — uklanja jednu ili više klase selektovanom elementu

- `toggleClass(className, state)` — prebacuje između dodavanja i uklanjanja klase selektovanog elementa
- `css(propertyName, value)` — dohvata ili postavlja CSS vrednosti u okviru atributa `style`

U narednim primerima podrazumevamo korišćenje sledećih CSS pravila:

```
.important {
    font-weight: bold;
    font-size: xx-large;
}

.blue {
    color: blue;
}
```

Sledeći primjeri ilustruju upotrebu prethodno opisanih metoda:

```
// Dodavanje jeste klase nad vise elemenata
$("button").click(function(){
    $("div").addClass("important");
    $("h1, h2, p").addClass("blue");
});

// Dodavanje vise klasa jednom elementu
$("button").click(function(){
    $("#div1").addClass("important blue");
});

// Uklanjanje klase nad vise elemenata
$("button").click(function(){
    $("h1, h2, p").removeClass("blue");
});

// Dodavanje/uklanjanje klase nad vise elemenata
$("button").click(function(){
    $("h1, h2, p").toggleClass("blue");
});
```

Da bismo dohvatili vrednost specifikovanog CSS svojstva, koristimo sintaksu `css(propertyName)`. Naredni primer dohvata vrednost CSS svojstva `background-color` prvog nađenog elementa:

```
$( "p" ).css( "background-color" );
```

Da bismo postavili vrednost specifikovanog CSS svojstva, koristimo sintaksu `css(propertyName, value)`. Naredni primer postavlja vrednost CSS svojstva `background-color` svih nađenih elemenata:

```
$( "p" ).css( "background-color" , "yellow" );
```

Da bismo postavili više vrednosti specifikovanih CSS svojstava, koristimo sintaksu:

```
css({
    propertynname: value,
```

```

    propertyname: value,
    ...
});
```

Naredni primer postavlja vrednosti za CSS svojstva `background-color` i `font-size` svih nađenih elemenata:

```

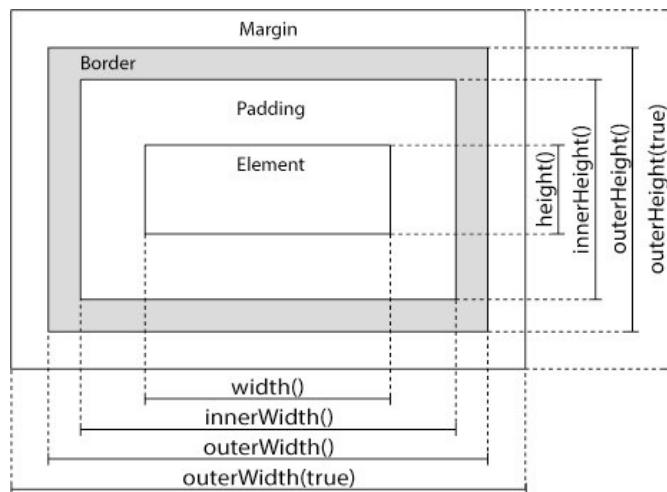
$("p").css({
  "background-color": "yellow",
  "font-size": "200%"
});
```

### 8.2.12 Upravljanje modelom kutije

Postoji nekoliko važnijih metoda za rad sa dimenzijama elementa:

- `width()` i `width(value)` — dohvata ili postavlja širinu sadržaja nad selektorom nad kojim se primenjuje.
- `height()` i `height(value)` — dohvata ili postavlja visinu sadržaja nad selektorom nad kojim se primenjuje.
- `innerWidth()` i `innerWidth(value)` — dohvata ili postavlja zbirnu širinu sadržaja i punjenja nad selektorom nad kojim se primenjuje.
- `innerHeight()` i `innerHeight(value)` — dohvata ili postavlja zbirnu visinu sadržaja i punjenja nad selektorom nad kojim se primenjuje.
- `outerWidth(includeMargin)` i `outerWidth(value)` — dohvata ili postavlja zbirnu širinu sadržaja, punjenja i ivice, kao i pojasa, ukoliko je opcioni argument `includeMargin` postavljen na `true`.
- `outerHeight(includeMargin)` i `outerHeight(value)` — dohvata ili postavlja zbirnu visinu sadržaja, punjenja i ivice, kao i pojasa, ukoliko je opcioni argument `includeMargin` postavljen na `true`.

Na narednoj slici je prikazano čime upravljaju svaki od navedenih metoda:



Da bismo prikazali upotrebu nekih od pomenutih elemenata jQuery biblioteke, pogledajmo naredne primere.

**Primer 8.1:**

Napisati program u programskom jeziku JavaScript koji izračunava broj dana do kraja tekućeg semestra od trenutka kada se veb dokument učitava. Koristiti ugradenu klasu `Date` za konstruisanje današnjeg i traženog datuma. Korišćenjem jQuery biblioteke upisati rezultat u odgovarajući element na stranici i dinamički mu dodeliti odgovarajuću klasu koja će obojiti tekst u plavo ako ima barem 15 dana do kraja, odnosno u crveno, ako ima manje od 15 dana do kraja semestra. Omogućiti da ukoliko se stranica pokrene nakon traženog datuma, ispiše se tekst "Semestar se zavrsio..." i postaviti odgovarajuća CSS svojstva tako da se tekst prikaže podebljano i u zelenoj boji.

**Rešenje:**

Kod 64: primeri/javascript3/kraj\_semestra.html

```
<!DOCTYPE html>

<html>

<head>
    <title> jQuery – broj dana do kraja semestra </title>
    <meta charset='utf-8'>

    <style type='text/css'>
        .red {
            color: red;
        }

        .blue {
            color: blue;
        }
    </style>

    <script src='js/jquery-3.3.1.min.js'></script>
    <script type='text/javascript'>
        $(document).ready(function () {
            /*
                Date objekat se moze inicializovati navodjenjem odgovarajuce
                godine, meseca i dana.
                Napomena: numeracija meseci pocinje od 0.
            */
            let kraj_semestra = new Date(2019, 0, 11);
            let danas = new Date();

            // Ocitavanje aktuelnog datuma
            let dan = danas.getDate();
            let mesec = danas.getMonth() + 1;
            let godina = danas.getFullYear();

            console.log(dan, mesec, godina);
        });
    </script>
</head>
<body>
    <div class='red'>Broj dana do kraja semestra: </div>
</body>

```

```
// Broj dana do kraja semestra
let broj_dana;

// Metod getTime() vraca broj milisekundi od epohe: 1. januar
// 1970.
broj_dana = kraj_semestra.getTime() - danas.getTime();
if (broj_dana < 0)
{
    $("#info")
        .text("Semestar se zavrsio...")
        .css({
            'font-weight': 'bold',
            'color': 'green'
        });
    return;
}

// Math.ceil je funkcija koja vrsti zaokruzivanje vrednosti na prvu
// vecu vrednost
broj_dana = Math.ceil(broj_dana / (1000 * 3600 * 24));

if (broj_dana < 15)
{
    $("#info")
        .text(broj_dana)
        .addClass('red');
}
else
{
    $("#info")
        .text(broj_dana)
        .addClass('blue');
}
});

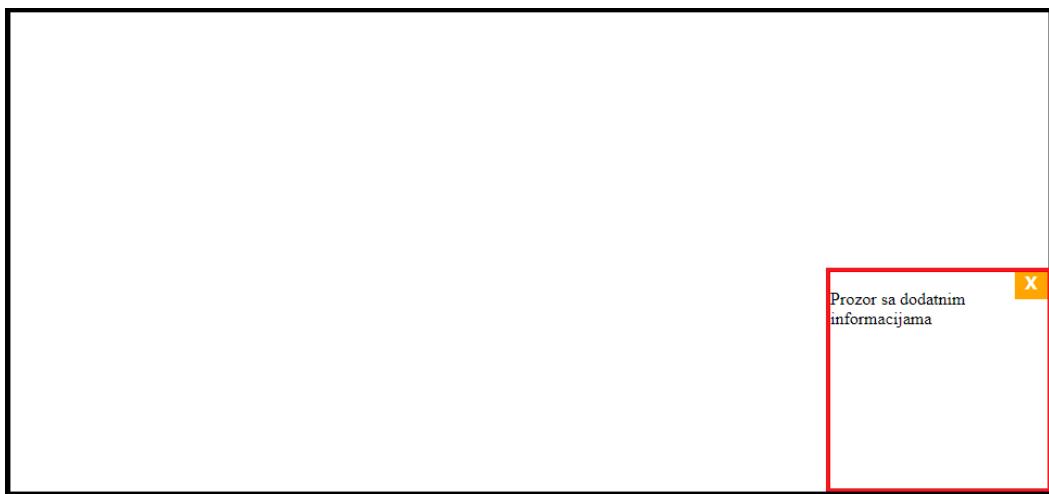
</script>
</head>

<body>
    <div id='info'>
    </div>
</body>

</html>
```

**Primer 8.2:**

Korišćenjem HTML i CSS jezika za obeležavanje i stilizovanje, konstruisati i pozicionirati informacioni prozor kao na narednoj slici:



Inicijalno je potrebno sakriti prozor. Pomoću biblioteke jQuery, od trenutka učitavanja veb dokumenta, nakon 2 sekunde je potrebno prikazati prozor efektom slajdera odozdo u trajanju od 1 sekunde. Takođe, omogućiti da se prozor sakrije efektom slajdera odozgo u trajanju od 1 sekunde klikom na "X".

**Rešenje:**

Kod 65: primeri/javascript3/help\_box.html

```
<!DOCTYPE html>

<html>

<head>
    <title> jQuery – animacije </title>
    <meta charset='utf-8'>

    <style type='text/css'>
        #help_box {
            border: 2px solid red;
            position: absolute;
            bottom: 0px;
            right: 0px;
            width: 200px;
            height: 200px;
            display: none;
        }

        .close {
            position: absolute;
            top: 0px;
            right: 0px;
        }
    </style>

```

```
height: 25px;
width: 30px;
font-weight: bold;
background-color: orange;
color: white;
font-family: Tahoma, sans-serif;
text-align: center;
}

.close:hover {
    cursor: pointer;
}
</style>
<script src='js/jquery-3.3.1.min.js'></script>
<script type='text/javascript'>
    $(document).ready(function () {
        $('#help_box').delay(2000).slideDown(1000);
        $('.close').on('click', function () {
            $('#help_box').slideUp(1000);
        });
    });
</script>
</head>

<body>
    <div id='help_box'>
        <div class='close'> X </div>
        <div>
            <p> Prozor sa dodatnim informacijama </p>
        </div>
    </div>
</body>

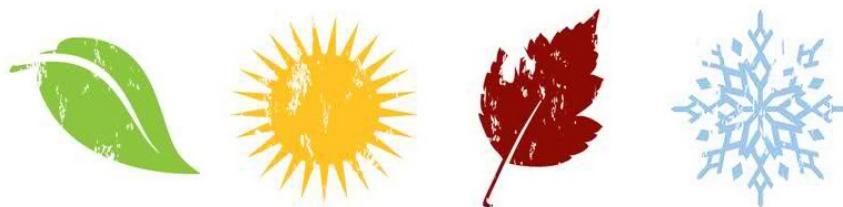
</html>
```

**Primer 8.3:**

Korisćenjem HTML i CSS jezika za obeležavanje i stilizovanje, konstruisati galeriju slika kao na narednoj slici:

[PRETHODNA](#)[NAREDNA](#)[SLIDESHOW](#)[STOP](#)

Galerija slika se sastoji od 4 slike koje predstavljaju proleće, leto, jesen i zimu, u tom poretku:



Koristeći biblioteku jQuery, omogućiti naredne funkcionalnosti:

- Klikom na dugme "PRETHODNA", napraviti prelaz između trenutne i prethodne slike u trajanju od 1 sekunde. Ukoliko je trenutna slika prva u nizu, prikazati poslednju sliku.
- Klikom na dugme "NAREDNA", napraviti prelaz između trenutne i naredne slike u trajanju od 1 sekunde. Ukoliko je trenutna slika poslednja u nizu, prikazati prvu sliku.
- Klikom na dugme "SLIDESHOW", omogućiti da se automatski vrše prelazi na naredne slike (odnosno, na prvu sliku kad se dođe do poslednje slike) u pravilnim vremenskim intervalima od 1 sekunde. Takođe, postaviti da se dugme "SLIDESHOW" oboji u zelenu boju dok je slideshow aktivran.
- Klikom na dugme "STOP", omogućiti da se slideshow prekine i da se dugme "SLIDESHOW" vrati u narandžastu boju.

### Rešenje:

Kod 66: primeri/javascript3/galerija\_jquery.html

```
<!DOCTYPE html>

<html>

<head>
    <title> jQuery – galerija </title>
    <meta charset="utf-8">
    <style type="text/css">
        .dugme {
            background-color: orange;
            color: white;
            height: 20px;
            width: 100px;
            display: inline-block;
            text-align: center;
            text-transform: uppercase;
            padding: 15px;
            font-family: sans-serif;
        }

        .dugme:hover {
            cursor: pointer;
        }
    </style>
</head>
```

```
}

#galerija {
    border: 1px solid orange;
    height: 210px;
    width: 210px;
    position: relative;
}

/*
    Slike u galeriji pozicioniramo tako da se preklapaju,
    tj. da budu jedna preko druge
*/
#galerija div {
    position: absolute;
    top: 10px;
    left: 10px;
}

#dugmad {
    position: relative;
    top: 25px;
}

.aktivnoDugme {
    background-color: forestgreen;
}
</style>
</head>

<body>
<!--
    Slike koje se nalaze u galeriji se navode unutar elementa sa
    identifikatorom "galerija".
    Slika proleca je na poziciji 0, a slika zime na poziciji 3.
-->
<div id="galerija">
    <div> </div>
    <div></div>
    <div></div>
    <div></div>
</div>

<div id="dugmad">
    <span id="prethodna" class="dugme"> prethodna </span>
    <span id="naredna" class="dugme"> naredna </span>
    <span id="slideshow" class="dugme"> slideshow </span>
    <span id="stop" class="dugme"> stop </span>
</div>

<script src="js/jquery-3.3.1.min.js"></script>
<script type="text/javascript">
    $(document).ready(function () {
```

```
// Redni broj slicice koja se trenutno vidi.  
let i = 0;  
  
// Kad se dokument ucita, sakrivamo sve slicice osim prve.  
$("#galerija div:gt(0)").hide();  
  
// Prikaz naredne slike  
const narednaSlika = function () {  
    // Sakrivamo tekucu slicicu  
    $("#galerija div:eq(" + i + ")").fadeOut(1000);  
  
    // Odredjujemo indeks sledece slicice  
    i = (i == 3) ? 0 : i + 1;  
  
    // Prikazujemo sledecu slicicu  
    $("#galerija div:eq(" + i + ")").fadeIn(1000);  
};  
  
// Prikaz prethodne slike  
const prethodnaSlika = function () {  
    // Sakrivamo tekucu slicicu  
    $("#galerija div:eq(" + i + ")").fadeOut(1000);  
  
    // Odredjujemo indeks prethodne slicice  
    i = (i == 0) ? 3 : i - 1;  
  
    // Prikazujemo prethodnu slicicu  
    $("#galerija div:eq(" + i + ")").fadeIn(1000);  
};  
  
// Dodajemo osluskivace za dugmice naredna i prethodna  
$("#naredna").click(narednaSlika);  
$("#prethodna").click(prethodnaSlika);  
  
// Promenljiva koja pamti podesavanja za setInterval  
let si;  
  
// Aktiviranje slideshow-a  
$("#slideshow").click(function () {  
    /*  
        Brisemo stari slideShow ako je postojao.  
        Ovim se sprecava nekorektno ponasanje ako korisnik vise  
        puta klikne na ovo dugme  
    */  
    window.clearInterval(si);  
  
    si = window.setInterval(function ()  
    {  
        // Specijalan slucaj kad se prelazi iz poslednje u prvu  
        // slicicu  
        if (i == 3)  
        {  
            $("#galerija div:last").fadeOut(1000);  
        }  
    }, 3000);  
});
```

```
        $("#galerija div:first").fadeIn(1000);
        i = 0;
    }
    // Ostali slucajevi
    else
    {
        $("#galerija div:eq(" + i + ")")
            .fadeOut(1000)
            .next()
            .fadeIn(1000);
        ++i;
    }

}, 1000);

// Menjamo izgled slideshow dugmeta u aktivno
$(this).addClass("aktivnoDugme");

// Onemogucujemo da korisnik klikne na dugmice naredna i
// prethodna
$("#naredna").off("click");
$("#prethodna").off("click");
});

// Prekidanje slideshow-a
$("#stop").click(function ()
{
    // Brisemo stanje za slideshow
    window.clearInterval(si);

    // Vracamo dugme za slideshow u normalno stanje
    $("#slideshow").removeClass("aktivnoDugme");

    // Aktiviramo ostale dugmice
    $("#naredna").click(narednaSlika);
    $("#prethodna").click(prethodnaSlika);
});
});
</script>
</body>

</html>
```



**Primer 8.4:**

Korišćenjem HTML i CSS jezika za obeležavanje i stilizovanje, konstruisati formular kao na narednoj slici:

Pre odabira polja "Prijavite se za vesti":

The screenshot shows a user registration form with the following fields and buttons:

- Korisničko ime \*:** An input field.
- e-mail \*:** An input field.
- Šifra \*:** An input field.
- Ponovite šifru \*:** An input field.
- Prihvati se za vesti:** A checkbox labeled with the text "Prihvati se za vesti".
- Posalji podatke** and **Reset** buttons at the bottom.

A small note at the bottom right of the form area states: "polja obeležena sa \* su obavezna".

Nakon odabira polja "Prijavite se za vesti":

The screenshot shows the same user registration form, but the "Prihvati se za vesti" checkbox is now checked. The "Oblast" dropdown menu is open, showing three options: "Dnevne", "Nedeljne", and "Mesečne". The "Nedeljne" option is selected. The entire dropdown menu is highlighted with a dotted orange border.

The "Posalji podatke" and "Reset" buttons are at the bottom, and the note "polja obeležena sa \* su obavezna" is still present at the bottom right.

Doterati formular korišćenjem Bootstrap biblioteke. Koristeći biblioteku jQuery omogućiti naredne funkcionalnosti u formularu:

- Kada element za unos korisničkog imena izgubi fokus, proveriti da li je polje prazno. Ukoliko jeste, upisati poruku o grešci u za to pripremljeno polje uz dodavanje klase "greska". Ukoliko nije, obrisati poruku o grešci iz za to predviđenog polja i ukloniti klasu "greska".
- Kada element za unos elektronske adrese izgubi fokus, proveriti da li je polje prazno. Ukoliko je polje prazno, proveriti da li posle njega postoji polje sa porukom o grešci i ako ne postoji, kreirati ga. Ukoliko postoji, ne

treba ništa preduzimati. Ukoliko polje nije prazno, proveriti da li posle njega postoji polje sa porukom o grešci i ako postoji obrisati ga. Ukoliko ne postoji, nista ne preduzimati.

- Prilikom svakog unosa preko tastature u elementu za unos lozinke, obezbediti da se vrši provera da li uneta lozinka ima više od 5 karaktera. Ukoliko lozinka nije korektna, proveriti da li posle njega postoji polje sa porukom o grešci i ako ne postoji, kreirati ga. Ukoliko postoji, ne treba ništa preduzimati. Ukoliko je lozinka korektna, proveriti da li posle njega postoji polje sa porukom o grešci i ako postoji obrisati ga. Ukoliko ne postoji, nista ne preduzimati.
- Prilikom svake promene stanja checkbox dugmeta, proveriti da li je dugme selektovano. Ako jeste, prikazati dodatna polja za odabir vesti efektom slajdera na dole u trajanju od 1 sekunde. Ukoliko nije, sakriti dodatna polja za odabir vesti efektom slajdera na gore u trajanju od 1 sekunde.
- Pri slanju podataka iz formulara:
  - Proveriti da li je korisničko ime prazno. Ako jeste, ispisati odgovarajuću poruku u delu za izveštaj, fokusirati element za unos korisničkog imena i prekinuti dalju validaciju.
  - Proveriti unos za elektronsku adresu. Polje ne sme da bude prazno i mora da bude zadovoljavajućeg formata. Ako unos nije korekstan, ispisati odgovarajuću poruku u delu za izveštaj, fokusirati element za unos elektronske pošte i prekinuti dalju validaciju.
  - Proveriti originalnu i ponovljenu šifru. Obe moraju biti unete, minimalna dozvoljena dužina je 6 i moraju se poklapati. Ako unos nije korekstan, ispisati odgovarajuću poruku u delu za izveštaj, fokusirati element za unos originalne lozinke i prekinuti dalju validaciju.
  - Proveriti da li se korisnik prijavio na vesti. Ako jeste, proveriti da li je odabrao oblast. Ako jeste, očitati vrednost dinamike isporučivanja i prikazati prozor sa porukom o odabranoj dinamici. Ako nije odabrao oblast, prikazati prozor sa odgovarajućom porukom i prekinuti dalju validaciju.

### Rešenje:

Kod 67: primeri/javascript3/provera\_formulara.html

```
<!DOCTYPE html>
<html>

<head>
    <title> Validacija i ocitavanje podataka formulara koriscenjem JQuery biblioteke </title>
    <meta charset="utf-8">

    <link rel="stylesheet" type="text/css" href="css/bootstrap.css">
    <style type="text/css">
        form {
            border: 1px solid #eee;
            width: 50%;
            margin: 50px auto;
```

```
        position: relative;
        padding: 10px;
    }

    sup {
        color: orange;
        font-size: 10px;
    }

    span {
        font-size: 10px;
    }

    span.greska {
        color: red;
    }

    #vesti {
        display: none;
        margin: auto;
        margin-top: 20px;
        border: 1px dotted orange;
        width: 80%;
        padding: 35px;
    }

    /*
        Postoji problem sa ovom Bootstrap klasom i slideDown jQuery
        metodom,
        koja se ispravlja na ovaj nacin.
    */
    .form-check-input {
        position: static;
    }
</style>
</head>

<body>

<form method="GET" action="" class="container">

    <div class="form-group">
        <label for="username">
            Korisničko ime <sup> * </sup>:
        </label>
        <input type="text" id="username" name="username" class=
            "form-control">
        <span></span>
    </div>

    <div class="form-group">
        <label for="email">
            e-mail <sup> * </sup>:
        </label>
    </div>
</form>
```

```
</label>
<input type="text" id="email" name="email" class="form-control">
<span></span>
</div>

<div class="form-group">
    <label for="sifra1">
        Šifra <sup> * </sup>:
    </label>
    <input type="password" id="sifra1" name="sifra1" class="form-control">
</div>

<div class="form-group">
    <label for="sifra2">
        Ponovite šifru <sup> * </sup>:
    </label>
    <input type="password" id="sifra2" name="sifra2" class="form-control">
</div>

<div class="form-group form-check">
    <input type="checkbox" id="prijava" name="prijava" value="p" class="form-check-input">
    <label for="prijava" class="form-check-label">
        Prijavite se za vesti
    </label>
</div>

<div id="vesti">
    <div class="form-group">
        <label for="oblast">
            Oblast:
        </label>
        <select name="oblast" id="oblast" class="form-control">
            <option value="">Odaberite oblast</option>
            <option value="ar">Arhitektura Racunara</option>
            <option value="oop">Objektno-orijentisano programiranje</option>
            <option value="web">Web Tehnologije</option>
        </select>
    </div>

    <div class="form-gorup">
        <div class="form-group-check">
            <input type="radio" name="vesti" id="vesti_d" value="d" class="form-check-input">
            <label for="vesti_d" class="form-check-label">
                Dnevne
            </label>
        </div>

        <div class="form-group-check">
```

```

<input type="radio" name="vesti" id="vesti_n" value="n"
       checked class="form-check-input">
<label for="vesti_n" class="form-check-label">
    Nedeljne
</label>
</div>

<div class="form-group-check">
    <input type="radio" name="vesti" id="vesti_m" value="m"
           class="form-check-input">
    <label for="vesti_m" class="form-check-label">
        Mesečne
    </label>
</div>
</div>
</div>

<p id="izvestaj">
</p>

<input type="submit" value="Posalji podatke" class="btn btn-primary">
<input type="reset" value="Reset" class="btn btn-danger">

<span style="position: absolute; right: 15px;"> polja obeležena sa
    <sup> * </sup> su obavezna </span>
</form>

<script type="text/javascript" src="js/jquery-3.3.1.min.js"></script>
<script type="text/javascript" src="provera_formulara.js"></script>

</body>

</html>

```

Kod 68: primeri/javascript3/provera\_formulara.js

```

$(document).ready(function () {

    // Provera za korisnicko ime
    $("input[name='username']").blur(function ()
    {
        if ($(this).val() == "")
        {
            $(this).next().addClass("greska").text("Ovo polje je obavezno!");
        }
        else
        {
            $(this).next().removeClass("greska").text("");
        }
    });

    // Provera za elektronsku adresu
    $("input[name='email']").blur(function ()
    {

```

```
if ($(this).val() == "")  
{  
    if (!$(this).next().hasClass("greska"))  
    {  
        $(this).after("<span class='greska'> Ovo polje je obavezno! </span>");  
    }  
}  
else  
{  
    $(this).next(".greska").remove();  
}  
});  
  
// Provera za lozinku  
$("input[name='sifra1']").keyup(function ()  
{  
    if ($(this).val().length <= 5)  
    {  
        if (!$(this).next().hasClass("greska"))  
        {  
            $(this).after("<span class='greska'> kratka šifra! </span>");  
        }  
    }  
    else  
{  
        $(this).next(".greska").remove();  
    }  
});  
  
// Prikazivanje/sakrivanje polja za vesti  
$("input:checkbox").change(function ()  
{  
    if ($(this).is(":checked"))  
    {  
        $("#vesti").slideDown(1000);  
    }  
    else  
{  
        $("#vesti").slideUp(1000);  
    }  
});  
  
// Validacija formulara pri slanju podataka  
$("form").submit(function ()  
{  
    // Proveravamo korisnicko ime  
    let username = $("input[name='username']").val();  
    if (username === "")  
    {  
        $("#izvestaj").text("Korisničko ime je obavezno!");  
        $("input[name='username']").focus();  
        return false;  
    }  
});
```

```
}

// Proveravamo elektronsku adresu
let email = $("input[name='email']").val();
if (email === "") {
    $("#izvestaj").text("E-mail adresa je obavezna!");
    $("input[name='email']").focus();
    return false;
}

let manki = email.indexOf("@");
let poslednjaTackica = email.lastIndexOf(".");
if (manki === -1 || poslednjaTackica === -1 || manki > poslednjaTackica) {
    $("#izvestaj").text("Uneta e-mail adresa nije korektna!");
    return false;
}

// Proveravamo sifre (originalnu i ponovljenu sifru).
let sifra_original = $("input[name='sifra1']").val();
let sifra_potvrda = $("input[name='sifra2']").val();

if (sifra_original === "" || sifra_original.length <= 5 ||
    sifra_potvrda === "" || sifra_original != sifra_potvrda) {
    $("#izvestaj").text("Ponovo popunite polja za šifru!");
    $("input[name='sifra1']").focus();
    return false;
}

// Proveravamo da li se korisnik prijavio na vesti
if ($("#input:checkbox").is(":checked"))
{
    let oblast = $("select[name='oblast']").val();
    if (oblast === "") {
        window.alert("Niste odabrali oblast za vesti!");
        return false;
    }

    let vesti = $("input:radio[name='vesti']:checked").val();
    let dinamika;
    switch (vesti)
    {
        case "d":
            dinamika = "dnevnom";
            break;
        case "n":
            dinamika = "nedeljnom";
            break;
        case "m":
```

```
        dinamika = "mesecnom";
        break;
    }
    window.alert(`Odabrali ste da primelete vesti na ${dinamika} nivou`)
    ;
}
}

// Ako je sve u redu, u polju za izvestaj se na kratko moze videti
// zahvalnica...
$("#izvestaj").text("Hvala sto ste se prijavili!");

return true;
});

$("form").on("reset", function()
{
    $("span").text("");
    // Sakrivanje polja za vesti ukoliko su bile aktivne
    if ($("#input:checkbox").is(":checked"))
    {
        $("#vesti").slideUp(1000);
    };
});
});
```

## 9 Čas 11

Cilj časa je upoznavanje studenta sa razvojem aplikacija koje se izvršavaju na serveru u okruženju Node.js. Na početku ćemo govoriti o arhitekturi serverskih aplikacija kroz *pakete* (engl. *package*) i *module* (engl. *modules*), o upravljaču paketa u ovom okruženju i demonstriraćemo kako možemo koristiti pakete koji nam dolaze pri instalaciji Node.js okruženja kroz jednostavne primere.

### 9.1 Moduli

*Modul* (engl. *module*) je deo programa koji specifikuje na koje se druge delove programa on oslanja i koje mogućnosti on daje ostalim delovima programa. Te funkcionalnosti koje su izložene "svetu" zajedno se nazivaju *interfejs* (engl. *interface*) modula. Sve ostalo se smatra privatnim za taj modul, i o tome svet ne mora da vodi računa. Korišćenjem modula se smanjuje celokupna povezanost delova programa, odnosno, tendencija da "sve zna za sve", što se smatra lošom praksom za programiranje.

Odnosi između modula se nazivaju *zavisnosti* (engl. *dependency*). Kada modul zahteva deo iz nekog drugog modula, kažemo da taj modul zavisi od drugog modula. Da bismo razdvojili module, potrebno je da svaki modul ima svoj, privatan domet.

#### 9.1.1 Paketi

*Paketi* (engl. *package*) predstavljaju delove koda koji se mogu distribuirati (kopirati i instalirati). Paket može da sadrži jedan ili više modula i ima informacije o tome od kojih drugih paketa zavisi. Paket obično sadrži i dokumentaciju. Kada se problem pronađe u paketu ili se doda nova mogućnost, paket se ažurira. Tada, programi koji zavise od paketa (koji takođe mogu biti drugi paketi) mogu da se jednostavno ažuriraju na novu verziju.

#### 9.1.2 Node upravljač paketima

Rad na ovakav način zahteva infrastrukturu — mesto za skladištenje i pronalaženje paketa, kao i jednostavan način za njihovo instaliranje i ažuriranje. U svetu JavaScript jezika, ovakva infrastruktura je obezbeđena od strane NPM (<https://npmjs.org>). NPM predstavlja dve stvari:

1. veb servis odakle je moguće preuzeti (i postaviti) pakete
2. program (koji dolazi uz Node.js okruženje za JavaScript) koji pomaže instaliranje i upravljanje paketima

U tekstu ćemo videti kako možemo instalirati različite pakete za naše programe koje budemo pisali.

#### 9.1.3 CommonJS

Ovaj način predstavlja verovatno najrasprostranjeniji način za kreiranje JavaScript modula. Sistem Node.js koristi upravo ovaj način, a takođe i većina paketa na NPM sistemu, te ćemo mu se zbog toga i posvetiti.

Glavni koncept u CommonJS modulima jeste funkcija `require`. Kada pozovemo ovu funkciju sa imenom modula od kojeg zavisi naš modul, funkcija se postara da je modul učitan i vraća nam njegov interfejs. Pošto učitavanje postavlja omotač nad

kodom modula u funkciji, moduli automatski dobijaju svoj lokalni domet. Sve što oni treba da urade jeste da `require` pozivima dohvataju pakete od kojih zavise, a svoj interfejs stave u objekat koji se vezuje za `exports`. Zanimljiva stavka CommonJS modula je da, iako sistem modula kreira prazan objekat-interfejs za nas (koji je vezan za promenljivu `exports`), mi ga možemo zameniti bilo kojom vrednošću tako što pregazimo vrednost `module.exports` i time definišemo šta je to što naš modul isporučuje korisnicima. Tako možemo imati mnoge "skrivene" funkcije koje predstavljaju implementaciju, a koje korisnici neće videti (jer to za njih i nije značajno) tako što ih jednostavno ne izvezemo kroz promenljivu `module.exports`. Ovo je urađeno u velikom broju postojećih modula da bi se exportovala jedna vrednost umesto objekta-interfejsa.

Kada putanja za učitavanje paketa nije relativna, Node.js će pogledati instalirane pakete i potražiće u njima paket sa prosleđenim imenom. Instaliranje paketa u korenom direktorijumu projekta se može izvršiti komandom

```
$ npm install imePaketa
```

Kroz primere koji slede videćemo razne upotrebe CommonJS načina za kreiranje i uvoženje modula.

Da bismo kreirali novi Node.js projekat, koji će u našem slučaju predstavljati novu serversku aplikaciju, potrebno je da pokrenemo komandu

```
$ npm init
```

Komanda je interaktivnog tipa i od nas će tražiti podatke tipa "Kako se zove datoteka od koje se pokreće aplikacija?", "Koji je naziv projekta?", "Koja je licenca projekta?" i mnoge druge. Mi ćemo sve naše serverske aplikacije kreirati od `server.js` datoteke. Ova komanda će kreirati `package.json` datoteku koja sadrži unete vrednosti. Naravno, moguće je jednostavno izmeniti ovu datoteku ukoliko odlučimo da ažuriramo neku vrednost. Ono što je glavna prednost inicijalizacije novog projekta jeste što sada možemo da instaliramo pakete od kojih naša aplikacija zavisi, i te zavisnosti će biti zapamćene u `package.json` datoteci. To znači da možemo proslediti naš izvorni kod projekta zajedno sa tom datotekom drugoj osobi, koja može podesiti projekat jednostavnom komandom

```
$ npm install
```

kojom će se pogledati od kojih sve paketa zavisi naš projekat i oni će se instalirati. Svi paketi koji se instaliraju se čuvaju pod korenim direktorijumom projekta u direktorijumu `node_modules`. Ovaj direktorijum nije potrebno niti poželjno proslediti uz izvorni kod (ili postaviti na sistem za kontrolu verzija, ukoliko se takav sistem koristi).

## 9.2 Kreiranje serverskih aplikacija

Osnovni paket koji ćemo koristiti za kreiranje serverskih aplikacija je paket `http`<sup>10</sup>. On nam nudi mnogobrojne metode za rad, a mi ćemo koristiti metod `createServer` kojim se kreira nova instanca veb servera. Dakle, da bismo učitali paket `http` i kreirali instancu veb servera, možemo uraditi:

```
const http = require("http");
const server = http.createServer();
```

<sup>10</sup>Dokumentacija za ovaj paket je dostupna na adresi <https://nodejs.org/docs/latest-v10.x/api/http.html>

Ono što je potrebno uraditi jeste registrovati funkcije kojima će server odrediti na određene događaje, kao što je na primer, događaj koji se okine kada pristigne novi HTTP zahtev. Da bismo postavili osluškivač, možemo koristiti metode `on(eventName, listener)`<sup>11</sup> i `once(eventName, listener)`<sup>12</sup>. Razlika između ovih metoda je u tome što prvi metod će izvršiti po-potrebi-pozivnu funkciju `listener` svaki put kada je događaj predstavljen niskom `eventName` okinut, a drugi metod će samo jednom. Tako, na primer, za osluškivanje novih zahteva možemo koristiti događaj "request"<sup>13</sup> koji će proslediti po-potrebi-pozivnoj funkciji dva objekta: prvi predstavlja HTTP zahtev od klijenta, a drugi predstavlja HTTP odgovor od servera.

```
server.on("request", function(req, res)
{
    // Odgovaramo na zahtev od korisnika ovde...
})
```

Da bismo zapravo rekli serveru da treba da se pokrene, moramo pozvati metod `listen([port[, host[, backlog]]][, callback])`<sup>14</sup>, kojog je dovoljno proslediti broj porta na koji se pokreće serverska aplikacija.

```
const port = 3000;
server.listen(port);
```

Ukoliko želimo da izvršimo neku akciju kada je server spreman, možemo registrovati po-potrebi-pozivnu funkciju nad događajem "listening"<sup>15</sup>:

```
server.once("listening", function()
{
    console.log(`Go to http://localhost:${port}`);
});
```

Vratimo se na događaj "request" i objekte koje on prosleđuje, a sa kojima mi možemo da radimo: HTTP zahtev<sup>16</sup> i HTTP odgovor<sup>17</sup>. Nad ovim objektima su definisana razna svojstva i metodi koje možemo koristiti. Na primer, nad HTTP zahtevom imamo definisano:

- Svojstvo `headers` sadrži objekat čija su svojstva zaglavljia HTTP zahteva
- Svojstvo `method` sadrži nisku sa HTTP metodom
- Svojstvo `url` sadrži nisku sa url-om u HTTP zahtevu

Nad HTTP odgovorom imamo definisano:

- Metod `setHeader(name, value)` postavlja vrednost `value` za jedno zaglavje čiji je naziv `name`. Ako zaglavje već ima definisanu vrednost, biće upotrebљena nova vrednost umesto nje. Potrebno ga je pozvati pre metoda `writeHead`.

<sup>11</sup>[Dokumentacija](#)

<sup>12</sup>[Dokumentacija](#)

<sup>13</sup>[Dokumentacija](#)

<sup>14</sup>[Dokumentacija](#)

<sup>15</sup>[Dokumentacija](#)

<sup>16</sup>[Dokumentacija](#)

<sup>17</sup>[Dokumentacija](#)

- Metod `writeHead(statusCode[, statusMessage][, headers])` šalje HTTP zaglavlj. Možemo mu proslediti statusnu poruku i objekat čija su svojstva zaglavlj odgovora. Ako prosledimo zaglavlj koja su ranije postavljena pomoću metoda `setHeader`, onda će vrednosti prosleđene ovom metodu imati prioritet. Metod mora biti pozvan tačno jednom i to pre poziva metoda `write` i `end`.
- Metod `write(chunk[, encoding][, callback])` postavlja telo odgovora. Metod se može pozvati više puta svaki put dodajući nastavak tela.
- Metod `end([data][, encoding][, callback])` kojim se završava obrada HTTP zahteva i šalje se HTTP odgovor. Metod signalizira da su sva zaglavlj i telo poslati, te se mora pozvati nad svakim odgovorom.

Da bismo pokrenuli našu serversku aplikaciju, potrebno je da se u terminalu pozicioniramo u koreni direktorijum projekta i ukucamo komandu

```
$ nodemon server.js
```

Nodemon (<https://nodemon.io/>) predstavlja alat kojom se mogu lakše restartovati pokrenuti serveri prilikom izmena u izvornom kodu. Ovaj alat prati da li je došlo do izmene nekog izvornog fajla i ako jeste, onda će automatski ugasiti server, primeniti izmene i ponovo ga pokrenuti, što je korisno ako testiramo rad servera i pravimo izmene u isto vreme. Da bismo ga instalirali, potrebno je da izvršimo komandu

```
$ npm install -g nodemon
```

### Primer 9.1:

Napisati serversku aplikaciju koja se pokreće na portu 3000 i koja treba da opslužuje naredne HTTP zahteve:

- Bilo koji zahtev nad `http://localhost:3000/*`
  - Potrebno je da se vrati tekst "Hello world!" u telu odgovora. Tip odgovora je `text/plain`, a statusni kod je 200.

### Rešenje:

Kod 69: primeri/nodejs1/001/server.js

```
const http = require("http");

const server = http.createServer();
server.on("request", function(req, res)
{
    res.writeHead(200, {
        "content-type": "text/plain"
    });

    res.write("Hello World!");
    res.end();
});
```

```
const port = 3000;

server.listen(port);
server.once("listening", function()
{
    console.log(`Hello World server listening on port ${port}`);
    console.log(`Go to http://localhost:${port}`);
});
```

Prethodni primer ilustruje jednostavnost kreiranja i podizanja serverskih aplikacija. Naravno, što su zahtevi naših zadataka veći, to je i kod kojim se ti zadaci rešavaju veći. Međutim, ono što je pozitivna karakteristika Node.js razvojnog okruženja jeste da se aplikacije jednostavno skaliraju, kao i da je za korišćenje novih paketa dovoljno da razumemo API koji ti paketi nude. Pogledajmo naredni primer.

### Primer 9.2:

Napisati serversku aplikaciju koja se pokreće na portu 3000 i koja treba da opslužuje naredne HTTP zahteve:

- GET zahtev nad `http://localhost:3000/*`
  - Izračunati skalarni proizvod vektora  $x = (x_1, x_2, x_3)$  i  $y = (y_1, y_2, y_3)$  koji se prosleđuju serveru preko svojih koordinata.
- OPTIONS zahtev nad `http://localhost:3000/*`
  - Server odgovara da su dostupni metodi OPTIONS i GET.

Ispravno implementirati CORS mehanizam zaštite. Zatim, kreirati formular na klijentskoj strani, stilizovan pomoću biblioteke Bootstrap, i pomoću biblioteke jQuery izvršiti odgovarajuće validacije i poslati asinhroni zahtev ka serverskoj aplikaciji sa odgovarajućim podacima.

**Rešenje:** Kao što vidimo, u odnosu na prethodni primer, naši zahtevi su se značajno usložnili. Međutim, nakon ovog primera, razumećemo kako da ispravno povežemo našu klijentsku aplikaciju koja se pokreće iz veb pregledača sa serverskom aplikacijom koja se izvršava kao Node.js aplikacija. Za početak, krenimo od implementiranja serverske strane aplikacije, čiji će kostur biti nešto izmenjeno rešenje iz prethodnog primera:

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

const server = http.createServer();
server.on('request', function(req, res)
{
    let teloOdgovora = {};
    let statusniKod = 200;

    // Obrada zahteva ide ovde...

    res.writeHead(statusniKod, {
        'Access-Control-Allow-Origin': '*',
        'Access-Control-Allow-Headers': 'Content-Type'
    });
});
```

```

    res.write(JSON.stringify(teloOdgovora));
    res.end();
});

const port = 3000;
server.listen(port);
server.once('listening', function()
{
    console.log(`http://localhost:${port}`);
});

```

Ono što vidimo da je drugačije u odnosu na prethodni primer jeste što smo izdvojili statusni kod i telo odgovora u posebne promenljive, koje ćemo izmeniti u zavisnosti od toga na koji zahtev treba da odgovorimo, kao i na šta je to što ćemo vratiti kao odgovor.

Dodatno, vidimo da u startu postavljamo zaglavla Access-Control-Allow-Origin, kao i Access-Control-Allow-Headers. Ovo je u vezi za nečim što zovemo *deljenje resursa među različitim izvorima* (engl. *Cross-Origin Resource Sharing*, skr. CORS). U pitanju je mehanizam koji koristi dodatna HTTP zaglavla u odgovoru da naznači veb pregledaču da dopusti da veb aplikacija koja se izvršava sa jednog izvora (domena) ima pravo da koristi resurse sa servera koji se nalazi na drugom izvoru (domenu). Veb aplikacija kreira *HTTP zahtev među različitim izvorima* (engl. *cross-origin HTTP request*) kada zahteva resurs koji ima drugačiji izvor u odnosu na njegov izvor. Pod "izvorom" podrazumevamo domen, protokol i port. Da bismo ispravno implementirali obradu CORS zahteva, potrebno je da naša serverska aplikacija naznači sa kojih izvora CORS zahtevi mogu biti prihvaćeni, što se kontroliše zaglavljem Access-Control-Allow-Origin. Ukoliko je njegova vrednost \*, onda su svi izvori dostupni. Inače, mogu se samo neki izvori smatrati kao validni, na primer, `http://foo.example` i tada bismo vratile zaglavlj:

`Access-Control-Allow-Origin: http://foo.example`

Slično, možemo koristiti zaglavje Access-Control-Allow-Headers da bismo specifikovali koja zaglavla klijent može da pošalje ka serveru (pored nekih podrazumevanih koje veb pregledač automatski dodaje, kao što su Accept i User-Agent). U primeru smo postavili da je dozvoljeno zaglavje još i Content-Type. Ovime smo uspešno implementirali CORS mehanizam, te možemo preći na obradu zahteva.

S obzirom da je potrebno da obradimo samo zahteve GET i OPTIONS, potrebno je da izdvojimo informaciju o metodu iz HTTP zahteva, a zatim da obezbedimo da se samo ti metodi obrađuju:

```

let method = req.method;

if (method === "GET")
{
    // Obrada GET zahteva ide ovde...
}
else if (method === "OPTIONS")
{
    // Obrada OPTIONS zahteva ide ovde...
}
else
{
    statusniKod = 405;
    res.setHeader('Access-Control-Allow-Methods', 'OPTIONS, GET');
}

```

```
}
```

Vidimo da se u slučaju slanja nepodržanog metoda u zahtevu vraća statusni kod 405, kao i da se postavlja zaglavlj Access-Control-Allow-Methods, čija je vrednost spisak metoda koje server podržava.

Da bi se sprečilo da klijent šalje nepodržane metode, često klijentske aplikacije prvo pošalju OPTIONS metod ka serveru pre nego što pošalju sam zahtev. HTTP metod OPTIONS služi da se od servera dobije informacija o tome koji su HTTP metodi implementirani na serveru. Potrebno je da server pošalje ovu informaciju kroz zaglavlj Access-Control-Allow-Methods:

```
else if (method === "OPTIONS")
{
    statusniKod = 200;
    res.setHeader('Access-Control-Allow-Methods', 'OPTIONS, GET');
}
```

Predimo sada na glavni deo naše serverske aplikacije: implementiranje obrade GET zahteva. Kada se podaci od klijenta šalju putem zahteva GET, pravilo je da ti podaci budu zapisani kao deo URL-a ka kojem se zahtev šalje, i to u vidu dela URL-a koji se naziva *upit* (engl. *query*). Upit se od ostatka dela URL-a odvaja karakterom ? iza kojeg slede podaci zapisani u formatu *ime=vrednost* odvojeni karakterom &, na primer:

```
field1=value1&field2=value2&field3=value3...
```

S obzirom da mi očekujemo koordinate vektora  $x = (x_1, x_2, x_3)$  i  $y = (y_1, y_2, y_3)$ , očekujemo da će klijent poslati GET zahtev ka URL-u:

```
http://localhost:3000?x1=0&x2=0&x3=0&y1=0&y2=0&y3=0
```

Naravno, nule u gornjem URL-u će biti zamenjene vrednostima koje klijent zista bude posao ka serveru. Da bismo dohvatali URL zahteva klijenta, potrebno je da pristupimo svojstvu url objekta zahteva req. Sada je potrebno da izvučemo informacije iz ovog URL-a, što se može postići upotrebom paketa url koji dolazi instaliran uz Node.js. Funkcijom url.parse<sup>18</sup> možemo parsirati URL nisku, čime dobijamo objekat koji sadrži razne informacije o parsiranom URL-u. Nama je korisno svojstvo query, čija je vrednost objekat koji ima svojstva koji odgovaraju prosleđenim podacima:

```
const url = require('url');

// ...

if (method === "GET")
{
    let urlString = req.url;
    let urlObj = url.parse(urlString, true);
    let getPodaci = urlObj.query;
    let x = [getPodaci.x1, getPodaci.x2, getPodaci.x3];
    let y = [getPodaci.y1, getPodaci.y2, getPodaci.y3];

    // Ostatak obrade GET zahteva ide ovde...
}
```

<sup>18</sup>[Dokumentacija](#)

Primetimo da je potrebno da prosledimo kao drugi argument funkcije `url.parse` vrednost `true` da bi se parsirao i upit kao deo URL-a (podrazumevano neće biti parsiran).

Sada kada smo dohvatali podatke, kreirajmo funkciju koja će izračunati skalarni proizvod. Zapravo, kreiraćemo svoj CommonJS modul, da bismo ispoštovali arhitekturu Node.js okruženja:

```
// U okviru datoteke 'skalarni-proizvod.js'

function skalarni_proizvod(x, y)
{
    let sp = 0;
    for (let i = 0; i < x.length; ++i)
    {
        sp += x[i] * y[i];
    }

    return sp;
}

module.exports.skalarni_proizvod = skalarni_proizvod;
```

Vidimo da modul eksportuje jednu funkciju, koju možemo dohvatiti njegovim uključivanjem u `server.js` datoteku:

```
const {skalarni_proizvod} = require('./skalarni-proizvod');
```

Sada možemo da koristimo funkciju `skalarni_proizvod` iz našeg modula:

```
if (method === "GET")
{
    // ...

    let rezultat = skalarni_proizvod(x, y);

    res.setHeader('Content-Type', 'application/json');
    teloOdgovora = {rezultat};
}
```

Primetimo da smo morali da stavimo vitičaste zagrade oko naziva funkcije. To je zato što ćemo pozivom funkcije `require` dohvatiti objekat koji sadrži tu funkciju. Mogli smo napisati i

```
const spPaket = require('./skalarni-proizvod');
```

čime bismo onda umesto poziva `skalarni_proizvod(x, y)` morali da koristimo `spPaket.skalarni_proizvod(x, y)`.

Takođe, primećujemo još jednu novu sintaksu koju možemo da koristimo. Kada napišemo `{rezultat}`, kreiraće se objekat koji ima svojstvo `rezultat`, a čija vrednost će biti jednaka vrednosti istoimene promenljive u istom dometu. Drugim rečima, izraz `{rezultat}` je skraćeni zapis izraza `{rezultat: rezultat}`.

Ceo izvorni kod implementacije na serveru dat je u nastavku:

Kod 70: primeri/nodejs1/002/server.js

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

```
const url = require('url');
const {skalarni_proizvod} = require('./skalarni-proizvod');

const server = http.createServer();
server.on('request', function(req, res)
{
    let teloOdgovora = {};
    let statusniKod = 200;

    let method = req.method;

    if (method === "GET")
    {
        let urlString = req.url;
        let urlObj = url.parse(urlString, true);
        let getPodaci = urlObj.query;
        let x = [getPodaci.x1, getPodaci.x2, getPodaci.x3];
        let y = [getPodaci.y1, getPodaci.y2, getPodaci.y3];

        let rezultat = skalarni_proizvod(x, y);

        res.setHeader('Content-Type', 'application/json');
        teloOdgovora = {rezultat};
    }
    else if (method === "OPTIONS")
    {
        res.setHeader('Access-Control-Allow-Methods', 'OPTIONS, GET');
    }
    else
    {
        statusniKod = 405;
        res.setHeader('Access-Control-Allow-Methods', 'OPTIONS, GET');
    }

    res.writeHead(statusniKod, {
        'Access-Control-Allow-Origin': '*',
        'Access-Control-Allow-Headers': 'Content-Type'
    });
    res.write(JSON.stringify(teloOdgovora));
    res.end();
});

const port = 3000;
server.listen(port);
server.once('listening', function()
{
    console.log(`http://localhost:${port}`);
});
```

Kod 71: primeri/nodejs1/002/skalarni-proizvod.js

```
function skalarni_proizvod(x, y)
{
    let sp = 0;
```

```

for (let i = 0; i < x.length; ++i)
{
    sp += x[i] * y[i];
}

return sp;
}

module.exports.skalarni_proizvod = skalarni_proizvod;

```

Pozabavimo se sada implementacijom na klijentskoj strani. Da bismo prosledili podatke iz veb pregledača našoj serverskoj aplikaciji, možemo da napravimo formular kao na slici:

**Vektor x**

x:

**Vektor y**

y:

**Skalarni proizvod**

Rezultat:

**Izracunaj**

Validaciju podataka ćemo raditi na klijentskoj strani, slično kao i na prethodnim časovima:

```

$("#f").submit(function()
{
    // Brisanje rezultata od proslog izracunavanja
    $("#rezultat").val("");

    // Dohvatanje vrednosti i validacija
    let x1 = Number.parseInt($("#x1").val());
    let x2 = Number.parseInt($("#x2").val());
    let x3 = Number.parseInt($("#x3").val());
    let y1 = Number.parseInt($("#y1").val());
    let y2 = Number.parseInt($("#y2").val());
    let y3 = Number.parseInt($("#y3").val());

    if (isNaN(x1) || isNaN(x2) || isNaN(x3) || isNaN(y1) || isNaN(y2) ||
        isNaN(y3))
    {
        $("#greska").text("Treba da unesete brojeve!");
        return false;
    }
})

```

```

}

let podaci = {x1, x2, x3, y1, y2, y3};

// Slanje asinhronog zahteva ide ovde...

// Cistimo polje za gresku
$("#greska").text("");

// Sprecavanje da veb pregledac prosledi podatke na stranicu zadatu u
// atributu action
return false;
}

```

Videli smo kako je moguće proslediti asinhroni zahtev ka serveru korišćenjem objekta klase `XMLHttpRequest`. Međutim, uverili smo se da je čak i na jednostavnim primerima rad sa ovim objektom veoma neugodan i kod ne izgleda lepo. Na našu sreću, biblioteka jQuery definiše svoju funkciju za slanje asinhronih poziva: `$.ajax(url [, settings])`. Argumenti ove funkcije su:

1. `url` — Niska koja sadrži putanju na koju je potrebno poslati asinhroni zahtev.
2. `settings` — Objekat kojim se vrši konfiguracija asinhronog zahteva. Neke od vrednosti koje možemo podešavati su:
  - `method` — Niska koja sadrži HTTP metod zahteva.
  - `data` — Objekat koji definiše podatke koji se šalju u zahtevu. Ukoliko je za `method` izabran `GET` zahtev, podaci će biti prosleđeni kao upit u URL-u. Inače, možemo birati u kom formatu ćemo poslati podatke — na primer, ukoliko želimo da podaci budu poslati u JSON formatu, potrebno ih je serializovati tako što se proslede funkciji `JSON.stringify()` i potrebno je postaviti vrednost `contentType` na '`application/json`' (videti ispod).
  - `contentType` — Niska kojom se definiše MIME tip podataka koji se šalje.
  - `success` — Po-potrebi-pozivna funkcija koja će se izvršiti ukoliko zahtev prođe uspešno. Funkcija može primiti tri argumenta:
    - (a) `data` — Telo odgovora od servera.
    - (b)  `textStatus` — Statusna poruka koja je dobijena od servera.
    - (c) `jqXHR` — Specijalan jQuery objekat koji predstavlja nadskup `XMLHttpRequest` klase<sup>19</sup>.
  - `error` — Po-potrebi-pozivna funkcija koja će se izvršiti ukoliko zahtev prođe neuspešno. Funkcija može primiti tri argumenta:
    - (a) `jqXHR` — Specijalan jQuery objekat koji predstavlja natklasu `XMLHttpRequest` klase<sup>20</sup>. To znači da možemo da koristimo ista svojstva i metode kao nad običnim `XMLHttpRequest` objektom, kao što smo to radili u sekciji 8.1.2.

<sup>19</sup>[Dokumentacija](#)

<sup>20</sup>[Dokumentacija](#)

- (b) `textStatus` — Statusna poruka koja je dobijena od servera. Može biti neka od: `"timeout"`, `"error"`, `"abort"` ili `"parsererror"`.
- (c) `errorThrown` — Objekat koji predstavlja grešku.

Slanje asinhronog zahteva sada možemo uraditi na sledeći način:

```
$.ajax("http://localhost:3000", {
    method: "GET",
    data: podaci,
    success: function(data, textStatus, jqXHR)
    {
        if (data.hasOwnProperty("rezultat"))
        {
            $("#rezultat").val(data.rezultat);
        }
    },
    error: function(jqXHR, textStatus, errorThrown)
    {
        $("#greska").text("Postoji greska sa zahtevom: " + textStatus);
    }
});
```

Primetimo da smo nad objektom koji predstavlja telo odgovora od servera pozvali metod `hasOwnProperty` koji do sada nismo videli. Ovaj metod služi da proveri da li objekat nad kojim se poziva sadrži svojstvo koje se zadaje kao argument metoda, pre nego što mu zapravo pristupimo.

Ono što treba da zapamtimo kod asinhronih zahteva kreiranih na ovaj način jeste da će uvek biti izvršena ili funkcija `success` ili funkcija `error`, nikada obe.

Ceo izvorni kod implementacije na klijentu dat je u nastavku:

Kod 72: primeri/nodejs1/002/skalarni\_proizvod.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Skalarni proizvod</title>

    <link rel="stylesheet" type="text/css" href="bootstrap.css">

    <style>
        #greska {
            color: red;
        }
    </style>
</head>
<body>
    <div class="container text-center">
        <form id="f" method="GET" action="" novalidate>
            <fieldset class="form-group">
                <legend>Vektor x</legend>
                <div class="form-group">
                    <label for="x1">x:</label>
```

```
<input type="text" id="x1" name="x1">
<input type="text" id="x2" name="x2">
<input type="text" id="x3" name="x3">
</div>
</fieldset>
<fieldset class="form-group">
<legend>Vektor y</legend>
<div class="form-group">
<label for="y1">y:</label>
<input type="text" id="y1" name="y1">
<input type="text" id="y2" name="y2">
<input type="text" id="y3" name="y3">
</div>
</fieldset>
<div id="greska"></div>
<fieldset class="form-group">
<legend>Skalarni proizvod</legend>
<div class="form-group">
<label for="rezultat">Rezultat:</label>
<input type="text" id="rezultat" name="rezultat">
</div>
</fieldset>
<div class="form-group">
<input type="submit" value="Izračunaj" class="btn btn-primary">
</div>
</form>
</div>

<script type="text/javascript" src="jquery-3.3.1.min.js"></script>
<script type="text/javascript">
$("#f").submit(function()
{
    // Brisanje rezultata od proslog izacunavanja
    $("#rezultat").val("");

    // Dohvatanje vrednosti i validacija
    let x1 = Number.parseInt($("#x1").val());
    let x2 = Number.parseInt($("#x2").val());
    let x3 = Number.parseInt($("#x3").val());
    let y1 = Number.parseInt($("#y1").val());
    let y2 = Number.parseInt($("#y2").val());
    let y3 = Number.parseInt($("#y3").val());

    if (isNaN(x1) || isNaN(x2) || isNaN(x3) || isNaN(y1) || isNaN(y2)
        || isNaN(y3))
    {
        $("#greska").text("Treba da unesete brojeve!");
        return false;
    }

    let podaci = {x1, x2, x3, y1, y2, y3};
}
```

```
// Dohvatanje rezultata.  
// Primetimo da za slanje podataka koristimo GET metod,  
// tako da nije potrebno da ih serijalizujemo i postavimo  
// content type svojstvo,  
// zato sto se ti podaci ugnezdjavaju u URL zahteva.  
$.ajax("http://localhost:3000", {  
    method: "GET",  
    data: podaci,  
    success: function(data, textStatus, jqXHR)  
    {  
        console.log(data);  
  
        if (data.hasOwnProperty("rezultat"))  
        {  
            $("#rezultat").val(data.rezultat);  
        }  
    },  
    error: function(jqXHR, textStatus, errorThrown)  
    {  
        $("#greska").text("Postoji greska sa zahtevom: " +  
            textStatus);  
    }  
});  
  
// Cistimo polje za gresku  
$("#greska").text("");  
  
// Sprecavanje da veb pregledac prosledi podatke na stranicu  
// zadatu u atributu action  
return false;  
});  
</script>  
</body>  
</html>
```

**Zadatak 9.1:**

Napisati serversku aplikaciju koja se pokreće na portu 3000 i koja treba da opslužuje naredne HTTP zahteve:

- GET zahtev nad `http://localhost:3000/*`
  - Konstruisati Captcha<sup>21</sup> nisku koja se može sastojati od cifara, velikih i malih slova. Serveru se prosleđuje niz Bulovih vrednosti koji odgovaraju svakom od izbora, kao i broj koji definiše dužinu niske koja se generiše. Ukoliko se prosledi dužina koja je veća od broja karaktera koji su dostupni, dovoljno je da se vrati niska najveće moguće dužine (na primer, ukoliko su odabrani samo brojevi, a zahteva se niska dužine 20, u redu je samo poslati nisku, na primer, "5396247018").
- OPTIONS zahtev nad `http://localhost:3000/*`
  - Server odgovara da su dostupni metodi OPTIONS i GET.

Ispravno implementirati CORS mehanizam zaštite. Zatim, kreirati formular na klijentskoj strani pregledača, stilizovan pomoću biblioteke Bootstrap, i pomoću biblioteke jQuery izvršiti odgovarajuće validacije i poslati asinhroni zahtev ka serverskoj aplikaciji sa odgovarajućim podacima iz formulara. Dozvoljeno je korišćenje paketa `range`<sup>22</sup> i `shuffle-array`<sup>23</sup>. Primer izgleda formulara dat je na narednoj slici:

Captcha programčić

- Mala slova
- Velika slova
- Cifre

Dužina niske:

Generiši

### Rešenje:

Implementacija na serverskoj strani:

Kod 73: zadaci/006/server.js

```
const http = require('http');
const url = require('url');
const range = require('range');
const shuffle = require('shuffle-array');

const server = http.createServer();

server.on('request', function(req, res) {
    let teloOdgovora = {};
    let statusniKod = 200;

    let method = req.method;

    if (method === 'GET')
    {
        // Izdvajanje podataka iz zahteva
        let urlString = req.url;
        let urlObj = url.parse(urlString, true);
        let data = urlObj.query;
        // Primetimo da ukoliko je vrednost podatka niz,
        // onda se na ime te vrednosti dodaju karakteri "[]",
        // te ne mozemo koristiti tacka-notaciju da dohvativamo taj niz
        let boxes = data["check[]"];
        let n = data.n;

        // Pravljenje nizova brojeva, malih i velikih slova
        let numbers = range.range(0, 10);
        let lower = new Array(26);
        let upper = new Array(26);
```

<sup>21</sup><https://en.wikipedia.org/wiki/CAPTCHA>

<sup>22</sup>Dokumentacija

<sup>23</sup><https://www.npmjs.com/package/shuffle-array>

```
for(let i = 0; i < 26; ++i)
{
    lower[i] = String.fromCharCode(97 + i);
    upper[i] = String.fromCharCode(65 + i);
}

// Niz koji sadrzi sve nizove koje smo generisali.
// Primetimo da indeksi nizova moraju da odgovaraju indeksima
// odgovarajucim vrednostima u nizu boxes.
// Dakle, "all" je niz nizova!
let all = new Array();
all.push(lower);
all.push(upper);
all.push(numbers);

// Niz koji sadrzi samo dozvoljene karaktere
let allowed = new Array();

for(let i = 0; i < boxes.length; ++i)
{
    // Ako je odgovarajuce checkbox polje bilo ukljuceno
    if(boxes[i] === 'true')
    {
        // Spoji prethodni niz sa nizom koji odgovara tom checkbox
        // polju
        // i sacuvaj taj niz za narednu iteraciju.
        // Dakle, "allowed" je niz karaktera!
        allowed = allowed.concat(all[i]);
    }
}

// Nasumicno promesaj niz dozvoljenih karaktera i sacuvaj ga
allowed = shuffle(allowed);

// Prvo izdvoji prvih n karaktera u promesanom nizu
let captcha = allowed.slice(0, n);
// A zatim spoji te izdvojene karaktere u nisku,
// pri cemu ce se spajanje vrsiti po praznoj niski.
captcha = captcha.join("");

res.setHeader('Content-Type', 'application/json');
teloOdgovora = {captcha};
}

else if (method == 'OPTIONS')
{
    res.setHeader('Allow', 'OPTIONS, GET');
}
else
{
    statusniKod = 405;
    res.setHeader('Allow', 'OPTIONS, GET');
}
```

```
res.writeHead(statusniKod, {
  'Access-Control-Allow-Origin': '*',
  'Access-Control-Allow-Headers': 'Content-Type'
});
res.write(JSON.stringify(teloOdgovora));
res.end();
});

const port = 3000;
server.listen(port);

server.once('listening', function() {
  console.log(`http://localhost:${port}`);
});
```

Implementacija na klijentskoj strani:

Kod 74: zadaci/006/captcha.html

```
<!DOCTYPE html>
<html>

<head>
  <meta charset="UTF-8">
  <title>Captcha</title>

  <link rel="stylesheet" type="text/css" href="bootstrap.css">
</head>

<body>
  <div class="container">
    <form id="forma" action="" method="GET">
      <fieldset class="form-group">
        <legend>Captcha programčić</legend>

        <div class="form-group">
          <div class="form-group-check">
            <input value="mala_slova" name="mala_slova" type="checkbox" id="mala_slova" class="form-check-input">
            <label for="mala_slova" class="form-check-label">Mala slova</label>
          </div>

          <div class="form-group-check">
            <input value="velika_slova" name="velika_slova" type="checkbox" id="velika_slova" class="form-check-input">
            <label for="velika_slova" class="form-check-label">Velika slova</label>
          </div>

        <div class="form-group-check">
```

```
<input value="cifre" name="cifre" type="checkbox" id="cifre" class="form-check-input">
<label for="cifre" class="form-check-label">Cifre</label>
</div>

<div class="form-group">
    <label for="n">Dužina niske:</label>
    <input id="n" name="n" type="text" class="form-control">
</div>

<div class="form-group">
    <input value="Generiši" class="btn btn-primary" id="dugme" name="submit" type="submit">
</div>
</div>
</fieldset>
</form>

<div id="rezultat">
</div>
</div>

<script type="text/javascript" src="jquery-3.3.1.min.js"></script>
<script type="text/javascript">
let f = $("#forma");
let rez = $("#rezultat");

f.submit(function () {
    // Niz u kojem ćemo cuvati sta je odabранo u checkbox-ovima
    let boxes = new Array(3);
    // Niz checkbox-ova
    let checkboxes = $("input:checkbox");

    for (let i = 0; i < checkboxes.length; ++i)
    {
        // Da bismo mogli da koristimo jQuery metode nad objektima,
        // potrebno je da ih "uokvirimo" izmedju "$(" i ")"
        let x = $(checkboxes[i]);

        if (x.is(":checked"))
        {
            boxes[i] = true;
        }
        else
        {
            boxes[i] = false;
        }
    }

    // Parsiramo vrednost za duzinu niske
    let n = $("#n");
    n = Number.parseInt(n.val());
})
```

```
if (Number.isNaN(n))
{
    rez.text("Unesite ispravnu vrednost za duzinu niske!");
    return false;
}

// Podaci koje saljemo serveru.
// Primetimo da ovde saljemo niz boxes sa imenom "check".
// Na serveru ce ovaj niz biti vidljiv pod imenom "check[]"
// ako podatke parsiramo kroz url.parse funkciju.
let podaci = { "check": boxes, n };

$.ajax("http://localhost:3000", {
    method: "GET",
    data: podaci,
    success: function (data)
    {
        if (data.hasOwnProperty("captcha"))
        {
            rez.text(data.captcha);
        }
    },
    error: function ()
    {
        alert("Neuspelo generisanje captcha niske!");
    }
});

rez.text("");

return false;
});
</script>
</body>

</html>
```

## 10 Čas 12

Cilj ovog časa je upoznanje sa *rutiranjem* (engl. *routing*) zahteva, pravljenjem naših paketa, a sve u cilju konstruisanja *programerskog interfejsa aplikacije* (engl. *application programming interface*, skr. *API*) za jednostavnu veb prodavnici kroz pakete kao što su Express.js, body-parser, i dr.

### 10.1 Razvojno okruženje Express.js

Sada kada smo naučili kako da parsiramo URL putanju i da odredimo koji HTTP metod je klijent koristio da komunicira sa serverom, u stanju smo da pišemo najrazličitije serverske aplikacije. Ono što bismo brzo shvatili jeste da čim želimo da našoj aplikaciji dodamo novu funkcionalnost, veličina našeg programskog koda značajno raste. Takođe, ništa nas ne sprečava da razmišljamo o arhitekturi izvornog koda, što opet nije dobro.

Da bismo ubrzali razvoj naših serverskih aplikacija, na raspolaganju su nam razna razvojna okruženja, od kojih ćemo se mi upoznati sa Express.js<sup>24</sup>. Ovo okruženje predstavlja minimalističko i fleksibilno razvojno okruženje koje je dovoljno moćno da nam olakša razvoj raznih veb aplikacija. Ono za šta ćemo ga mi koristiti jeste za pravljenje API-ja koji će se jednostavno proširivati sa svakim novim zahtevom.

Instalacija Express.js razvojnog okruženja je jednostavna:

```
$ npm install express
```

#### Primer 10.1:

Napisati serversku aplikaciju koja se pokreće na portu 3000 i koja treba da opslužuje naredne HTTP zahteve:

- GET zahtev nad `http://localhost:3000/products`
  - Vratiti statusni kod 200 sa porukom `"GET /products"`
- POST zahtev nad `http://localhost:3000/products`
  - Kreirati objekat koji ima svojstva `name` i `price`, čije se vrednostiочitavaju iz tela zahteva. Telo se prosleđuje u JSON formatu. Vratiti statusni kod 201 sa porukom `"POST /products"` i vratiti novokreirani objekat.
- GET zahtev nad `http://localhost:3000/products` sa parametrom `productId`
  - Očitati parametar `productId` i ukoliko je parametar jednak vrednosti `"special"`, vratiti poruku `"Otkrili ste specijalni ID!"`. U ostalim slučajevima vratiti poruku ``Prosledjen je ID: ${productId}``. Vratiti statusni kod 200 u oba slučaja.
- PATCH zahtev nad `http://localhost:3000/products` sa parametrom `productId`
  - Vratiti statusni kod 200 sa porukom `"Azuriran je proizvod"`

<sup>24</sup><https://expressjs.com/>

- DELETE zahtev nad `http://localhost:3000/products` sa parametrom `productId`
  - Vratiti statusni kod 200 sa porukom "Obrisani je proizvod"
- OPTIONS zahtev nad `http://localhost:3000/*`
  - Server odgovara da su dostupni metodi OPTIONS, GET, POST, PATCH, DELETE.

Koristiti razvojno okruženje Express.js za rutiranje zahteva. Ispravno implementirati CORS mehanizam zaštite. Zatim, kreirati formulare, stilizovane pomoću biblioteke Bootstrap, kojima se mogu testirati zahtevi na klijentskoj strani i pomoću biblioteke jQuery izvršiti odgovarajuće validacije i poslati asinhroni zahtev ka serverskoj aplikaciji sa odgovarajućim podacima.

**Rešenje:** Rešenje ovog zadatka ćemo uraditi pristupom odozgo-nadole, odnosno, prvo ćemo pisati najapstraktniji kod, pa ćemo onda ulaziti u detalje. Dakle, prvo što je potrebno jeste da kreiramo `server.js` datoteku i kojoj ćemo instancirati server i pokrenuti ga:

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

const app = require('./app');

const port = process.env.PORT || 3000;

const server = http.createServer(app);

server.listen(port);
server.once('listening', function() {
    console.log(`Listening on port ${port}`);
});
```

Odavde već vidimo da ćemo napraviti modul `app` koji će služiti kao osluškivač za zahteve ka našoj serverskoj aplikaciji. Kreiranje zasebnog modula ima smisla s obzirom da imamo veliki broj zahteva koji treba da opslužimo. U ovom slučaju je dovoljno proslediti modul metodu `createServer`.

*Promenljivama okruženja* (engl. *environment variable*) možemo pristupati kroz objekat `process.env` čija svojstva sadrže vrednosti tih promenljivih. U slučaju da nam nije dostupna promenljiva `PORT`, koristićemo port 3000, kao i do sada. Ukoliko koristimo aplikaciju nodemon za pokretanje servera, onda možemo kreirati datoteku `modemon.json` u kojoj možemo da konfigurišemo pokretanje ovog programa, na primer, možemo postaviti promenljivu okruženja `PORT`:

```
{
  "env": {
    "PORT": "3000"
  }
}
```

Predimo na kreiranje `app` modula. S obzirom da koristimo Express.js razvojno okruženje, potrebno je da ga imamo uključeno.

```
const express = require('express');
```

Kreiranje instance Express.js aplikacije se vrši na jednostavan način — pozivanjem modula kao funkcije:

```
const app = express();
```

S obzirom da će naša aplikacija koristiti tela HTTP zahteva (u nekim zahtevima), bilo bi dobro da naš zahtev prvo "prođe" kroz module za parsiranje tela. Takvih modula ima mnogo, a mi ćemo koristiti paket `body-parser`, koji se jednostavno instalira:

```
$ npm install body-parser
```

Da bismo "provukli" HTTP zahtev kroz `body-parser`, potrebno je da nad našom Express.js aplikacijom pozovemo metod `use([path,] callback [, callback...])`<sup>25</sup>. Ovim metodom se registruje *srednji sloj* (engl. *middleware*) koji će obraditi zahtev kada se putanja zahteva poklopi sa promenljivom `path`. Ukoliko nije specifikovana putanja, onda će svaki zahtev proći kroz registrovani srednji sloj. Srednji sloj ne predstavlja ništa drugo do funkciju koja može da uradi nekakvu vrstu obrade nad zahtevom koji prima. Paket `body-parser` definiše više funkcija srednjeg sloja, kao što su `urlencoded`, koji će ispravno kodirati URL za prenos preko interneta i `json`, koji će parsirati telo u formatu JSON. Argument ovih funkcija je objekat sa specifikacijama parsiranja:

```
app.use(bodyParser.urlencoded({
  extended: false
}));
app.use(bodyParser.json());
```

Prva stvar koju je potrebno uraditi jeste napisati srednji sloj koji će ispravno obraditi CORS greške. U Express.js razvojnom okruženju, ovo je jednostavno uraditi:

```
app.use(function(req, res, next) {
  res.header('Access-Control-Allow-Origin', '*');
  res.header('Access-Control-Allow-Headers', 'Content-Type');

  if (req.method === 'OPTIONS')
  {
    res.header('Access-Control-Allow-Methods', 'OPTIONS, GET, POST,
              PATCH, DELETE');
    return res.status(200).json({});
  }

  next();
});
```

Ovde vidimo da prihvatomamo sve izvora i sva zaglavla. Dodatno, već na ovom mestu je sasvim korektno ispitati da li je poslat OPTIONS zahtev i odgovoriti na njega. Postavljanje zaglavla se vrši metodom `header(field [, value])`<sup>26</sup>. Postavljanje statusnog koda se može uraditi metodom `status(code)`<sup>27</sup> koji je moguće ulančati, na primer, metodom `json([body])`<sup>28</sup>, koji će postaviti telo odgovora u JSON formatu.

<sup>25</sup>[Dokumentacija](#)

<sup>26</sup>[Dokumentacija](#)

<sup>27</sup>[Dokumentacija](#)

<sup>28</sup>[Dokumentacija](#)

Ono što vidimo jeste da srednji slojevi mogu biti funkcije od tri argumenta. Prvi predstavlja objekat HTTP zahteva<sup>29</sup>, drugi predstavlja objekat HTTP odgovora<sup>30</sup>, a treći predstavlja naredni srednji sloj koji može biti pozvan ukoliko želimo da se tekući zahtev prosledi na dalju obradu (narednom registrovanom srednjem sloju). Bitno je zapamtiti da je potrebno da naši srednji slojevi ili pozivaju naredni srednji sloj ili da izađu iz njega. U slučaju da je poslat OPTIONS zahtev, nema smisla dalje obrađivati taj zahtev, te je dovoljno da postavimo zaglavla i prekinemo dalju obradu (tako što izađemo iz funkcije). Sa druge strane, ukoliko zahtev nije bio OPTIONS, to znači da ga je potrebno dodatno obraditi nekim od narednih srednjih slojeva, te zbog toga pozivamo funkciju `next`, koja će za nas uraditi upravo to.

Sada je potrebno da definišemo srednje slojeve koji će vršiti obradu zahteva. S obzirom da sve preostale zahteve možemo smestiti pod putanjom `/products`, ima smisla napraviti odvojen modul. Pošto je ovaj modul deo našeg API-ja, smestićemo ga u direktorijumu `./api/routes`:

```
const productRoutes = require('./api/routes/products');
```

Sada možemo da ga koristimo kao srednji sloj:

```
app.use('/products', productRoutes);
```

Međutim, još uvek nismo završili sa `app.js` datotekom. Zašto? Odgovor leži u tome da nismo specifikovali šta će se desiti ako klijent pošalje zahtev na nevalidnu putanju, na primer, `http://localhost:3000/emptyStore`. Drugim rečima, ako nije dan od prethodna dva srednja sloja nije uspeo da obradi zahtev, onda je potrebno napisati srednji sloj kojim ćemo reći da je došlo do greške u zahtevu:

```
app.use(function(req, res, next) {
    const error = new Error('Zahtev nije podrzan od servera');
    error.status = 405;

    next(error);
});
```

Pomoću `new Error([message])` možemo napraviti novu grešku sa opcionom porukom koja će biti ispisana. Takođe, dodajemo statusni kod 405 kao deo greške. Iz ovog koda takođe vidimo da se poziva naredni srednji sloj sa greškom kao prvim argumentom.

Zašto je potrebno napraviti još jedan srednji sloj kad smo sve greške obradili? Zapravo, postoje neke greške koje se mogu pojaviti, a da nisu obrađene trenutnim kodom. Na primer, može da se desi da postoji deo koda koji nismo ispravno iskodirali ili, kada budemo koristili bazu podataka, da postoji neki problem u radu sa bazom, što će dovesti do pojavljivanja greške. Za sve takve situacije, definišemo finalni srednji sloj, koji je funkcija sa 4 argumenta, od kojih je prvi greška koja se javila, a preostali argumenti su standardni argumenti srednjih slojeva:

```
app.use(function(error, req, res, next) {
    res.status(error.status || 500).json({
        greska: {
            poruka: error.message
        }
    });
});
```

<sup>29</sup>[Dokumentacija](#)

<sup>30</sup>[Dokumentacija](#)

Konačno, potrebno je da izvezemo našu Express.js aplikaciju kao modul i time smo završili kreiranje modula app:

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

Predimo sada na kreiranje modula za proizvode. Kao i do sada, biće nam potreban express modul. Da bismo kreirali ovaj modul, nije potrebno da pravimo celu Express.js aplikaciju kao što smo to radili u modulu app. Cilj ovog modula jeste da definiše odgovarajuće funkcije za svaki od zahteva pod putanjom /products. Drugim rečima, modul treba da vrši *rutiranje* (engl. *routing*) zahteva na osnovu putanje, odnosno, preusmeravanje zahteva na ispravnu funkciju za obradu. Za rutiranje možemo da koristimo funkciju `express.Routing()` koja kreira i vraća novi objekat za rutiranje:

```
const express = require('express');

const router = express.Router();
```

Nad ovim objektom su definisani razni metodi čiji nazivi odgovaraju HTTP metodima, uz napomenu da imena samih metoda sadrže isključivo mala slova. To znači da ako želimo da definišemo funkciju za obradu GET zahteva nad putanjom /products, dovoljno je da napišemo:

```
router.get('/', function(req, res, next) {
    // Obrada GET zahteva nad /products
});
```

Ono što je možda zbumujuće jeste zašto koristimo kao putanju nisku "/" umesto niske "/products". Prisetimo se da smo u modulu app koristili modul products na sledeći način:

```
app.use('/products', productRoutes);
```

Time smo rekli da svi zahtevi koji počinju na putanji /products bivaju preusmerni na modul products. Zbog toga, kada zahtev stigne do modula products, on neće imati prefiks /products jer se on na tom mestu podrazumeva.

Implementacija obrade ovog zahteva je jednostavna — potrebno je vratiti statusni kod 200 i odgovarajuću poruku:

```
router.get('/', function(req, res, next) {
    res.status(200).json({
        poruka: 'GET /products'
    });
});
```

Primetimo da nismo stavili naredbu `return` ispred `res`. To je zato što ukoliko tokom izvršavanja funkcije makar jednom postavimo statusni kod i telo odgovora, onda kada se funkcija završi, biće poslatko to što je bilo postavljeno. Naravno, možemo koristiti i naredbu `return` ukoliko to želimo.

Da bismo dobili informaciju o telu HTTP zahteva, na raspolažanju nam je svojstvo `body` objekta HTTP zahteva. Nazivi njegovih svojstava odgovaraju nazivima prosledjenih podataka ka serveru. Napomenimo da ukoliko nismo prvo bitno "pruvukli" HTTP zahtev kroz modul `body-parser`, vrednost ovog svojstva bi bila `null`. Sada smo u stanju da napišemo obradu POST zahteva nad putanjom /products:

```
router.post('/', function(req, res, next) {
    const kreiraniProizvod = {
```

```

        name: req.body.name,
        price: req.body.price
    });

    res.status(201).json({
        poruka: 'POST /products',
        kreiraniProizvod
    });
});
}
);

```

Ostalo je da još vidimo kako možemo da parametrizujemo zahteve. Da bismo rutirali parametrizovane putanje, dovoljno je da na kraj putanje nadovežemo karakter `:` koji je praćen nazivom parametra koji će nam biti dostupan u kodu. Na primer, ukoliko se putanja `/:productId` iskoristi u modulu `products` na način kao što su to ostale putanje u tom modulu korišćene, onda ako bismo prosledili zahtev ka URL-u:

`http://localhost:3000/products/2145621`

u odgovarajućoj funkciji za obradu zahteva bismo imali parametar `productId` čija bi vrednost bila `2145621`. Parametri zahteva su dostupni kroz svojstvo `params` objekta HTTP zahteva. Sada smo u stanju da napišemo funkcije za obradu ostalih zahteva:

```

router.get('/:productId', function(req, res, next) {
    const productId = req.params.productId;

    if (productId === 'special')
    {
        res.status(200).json({
            poruka: 'Otkrili ste specijalni ID!',
            id: productId
        });
    }
    else
    {
        res.status(200).json({
            poruka: 'Prosledjen je ID: ' + productId
        });
    }
});

router.patch('/:productId', function(req, res, next) {
    res.status(200).json({
        poruka: 'Azuriran je proizvod'
    });
});

router.delete('/:productId', function(req, res, next) {
    res.status(200).json({
        poruka: 'Obrisani je proizvod'
    });
});

```

Konačno, modul je potrebno eksportovati:

```
module.exports = router;
```

Ceo izvorni kod implementacije na serveru dat je u nastavku:

Kod 75: primeri/nodejs2/001/nodemon.json

```
{  
  "env":  
  {  
    "PORT": "3000"  
  }  
}
```

Kod 76: primeri/nodejs2/001/server.js

```
const http = require('http');  
  
const app = require('./app');  
  
// Mozemo dodeliti port kroz promenljivu okruzenja,  
// a u slucaju da nije dostupna,  
// bice postavljena vrednost 3000  
const port = process.env.PORT || 3000;  
  
const server = http.createServer(app);  
  
server.listen(port);  
server.once('listening', function() {  
  console.log(`Listening on port ${port}`);  
});
```

Kod 77: primeri/nodejs2/001/app.js

```
const express = require('express');  
  
// Paket za parsiranje tela zahteva  
const bodyParser = require('body-parser');  
  
// Kreiramo module za svaku putanju  
const productRoutes = require('./api/routes/products');  
  
// Nasa aplikacija ce u osnovi biti express aplikacija  
const app = express();  
  
// Dodajemo mogucnosti body-parser paketa  
app.use(bodyParser.urlencoded({  
  extended: false // Zelimo samo jednostavne url encoded podatke da  
  // parsiramo  
}));  
app.use(bodyParser.json({  
  // Ovde takodje mozemo konfigurisati  
  // sta sve moze da se parsira  
}));  
  
// Da bismo resili CORS greske,  
// moramo da pre obrade zahteva
```

```
// dodamo odgovarajuca zaglavlja
app.use(function(req, res, next) {
    res.header('Access-Control-Allow-Origin', '*');
    res.header('Access-Control-Allow-Headers', 'Content-Type');

    if (req.method === 'OPTIONS')
    {
        res.header('Access-Control-Allow-Methods', 'OPTIONS, GET, POST,
PATCH, DELETE');

        return res.status(200).json({});
    }

    // Ovaj poziv ce "proslediti" zahtev narednom middleware-u
    next();
});

// Za sve putanje koje pocinju sa "/products",
// zelimo da koristimo productRoutes middleware
// koji smo mi kreirali
app.use('/products', productRoutes);

// Ako je na ovom mestu HTTP zahtev prosao,
// to znaci da nije uhvacen od strane gornjih 'use' poziva.
// To znaci da je klijent poslao zahtev koji nismo definisali,
// pa mozemo da mu vratimo, na primer, 405 statusni kod
app.use(function(req, res, next) {
    const error = new Error('Zahtev nije podrzan od servera');
    error.status = 405;

    // Sada moramo da prosledimo nasu kreiranu gresku,
    // da bi se prikazao umesto podrazumevanog,
    // pa pozivamo naredni middleware,
    // sto je u ovom slucaju funkcija ispod
    next(error);
});

// Dodajemo jos jedan nivo hvatanja gresaka,
// za sve ostale greske, tipa:
// 1. greske u kodu,
// 2. greske u bazi podataka.
// 3. ...
app.use(function(error, req, res, next) {
    res.status(error.status || 500).json({
        greska: {
            poruka: error.message
        }
    });
});

module.exports = app;
```

Kod 78: primeri/nodejs2/001/api/routes/products.js

```
const express = require('express');

const router = express.Router();

// Ne stavljamo /products,
// zato sto ce vec imati taj prefiks
// kad dodje do ove putanje,
// zbog app middlewear-a
router.get('/', function(req, res, next) {
    res.status(200).json({
        poruka: 'GET /products'
    });
});

router.post('/', function(req, res, next) {
    const kreiraniProizvod = {
        name: req.body.name,
        price: req.body.price
    };

    res.status(201).json({
        poruka: 'POST /products',
        kreiraniProizvod
    });
});

// Individualni

router.get('/:productId', function(req, res, next) {
    const productId = req.params.productId;

    if (productId === 'special')
    {
        res.status(200).json({
            poruka: 'Otkrili ste specijalni ID!',
            id: productId
        });
    }
    else
    {
        res.status(200).json({
            poruka: 'Prosledjen je ID: ' + productId
        });
    }
});

router.patch('/:productId', function(req, res, next) {
    res.status(200).json({
        poruka: 'Azuriran je proizvod'
    });
});

router.delete('/:productId', function(req, res, next) {
```

```
res.status(200).json({
  poruka: 'Obrisan je proizvod'
});
});

module.exports = router;
```

Klijentski deo rešenja ne bi trebalo da bude teško uraditi, te ćemo samo dati rešenje. Napomenimo da je neophodno da sve podatke koje šaljemo kao vrednost svojstva `data` prilikom korišćenja jQuery metoda `$.ajax` (osim kad šaljemo GET zahtev) potrebno serijalizovati korišćenjem funkcije `JSON.stringify` i postaviti vrednost `contentType` na `'application/json'`.

Kod 79: primeri/nodejs2/001/klijent.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Testiranje rada servera sa proizvodima</title>

  <link rel="stylesheet" type="text/css" href="bootstrap.css">
</head>
<body>
  <div class="container">
    <h1>Testiranje rada servera</h1>
    <div class="row">
      <div class="col-md-6 mt-3">
        <h3>GET /products[:productId]</h3>
        <form id="form1">
          <div class="form-group">
            <label for="productId1">Identifikator proizvoda:</label>
            <input type="text" id="productId1" name="productId1"
                   class="form-control" title="Unesite identifikator
                   proizvoda ili ostavite prazno da dohvate sve
                   proizvode">
          </div>

          <input type="submit" value="Posalji podatke" class="btn
              btn-primary">
        </form>
        <div id="rezultat1"></div>
      </div>
      <div class="col-md-6 mt-3">
        <h3>POST /products</h3>
        <form id="form2">
          <div class="form-group">
            <label for="name">Naziv proizvoda:</label>
            <input type="text" id="name" name="name" class=
                   "form-control">
          </div>

          <div class="form-group">
```

```

        <label for="price">Cena proizvoda:</label>
        <input type="text" id="price" name="price" class="form-control">
    </div>

    <input type="submit" value="Posalji podatke" class="btn btn-primary">
</form>
<div id="rezultat2"></div>
</div>
<div class="row">
    <div class="col-md-6 mt-3">
        <h3>PATCH /products:productId</h3>
        <form id="form3">
            <div class="form-group">
                <label for="productId3">Identifikator proizvoda:</label>
                <input type="text" id="productId3" name="productId3" class="form-control">
            </div>

            <input type="submit" value="Posalji podatke" class="btn btn-primary">
        </form>
        <div id="rezultat3"></div>
    </div>
    <div class="col-md-6 mt-3">
        <h3>DELETE /products:productId</h3>
        <form id="form4">
            <div class="form-group">
                <label for="productId4">Identifikator proizvoda:</label>
                <input type="text" id="productId4" name="productId4" class="form-control">
            </div>

            <input type="submit" value="Posalji podatke" class="btn btn-primary">
        </form>
        <div id="rezultat4"></div>
    </div>
</div>
</div>
<script type="text/javascript" src="jquery-3.3.1.min.js"></script>
<script type="text/javascript" src="klijent.js"></script>
</body>
</html>
```

Kod 80: primeri/nodejs2/001/klijent.js

```

$(document).ready(function() {
    $("#form1").submit(function()
    {
        let productId = $("#productId1").val();
```

```
if (productId === "")  
{  
    $.ajax("http://localhost:3000/products", {  
        method: "GET",  
        success: function(data) {  
            if(data.hasOwnProperty("poruka")) {  
                $("#rezultat1").text(data.poruka).css({color: "green"});  
            }  
        },  
        error: function() {  
            $("#rezultat1").text("Greska pri dohvatanju proizvoda!").  
                css({color: "red"});  
        }  
    });  
}  
else  
{  
    $.ajax(`http://localhost:3000/products/${productId}`, {  
        method: "GET",  
        success: function(data) {  
            if(data.hasOwnProperty("poruka")) {  
                $("#rezultat1").text(data.poruka).css({color: "green"});  
            }  
        },  
        error: function() {  
            $("#rezultat1").text("Greska pri dohvatanju proizvoda!").  
                css({color: "red"});  
        }  
    });  
}  
  
$("#rezultat1").text("");  
  
return false;  
});  
  
$("#form2").submit(function()  
{  
    let name = $("#name").val();  
    let price = Number.parseFloat($("#price").val());  
  
    if (name === "")  
    {  
        $("#rezultat2").text("Niste prosledili naziv proizvoda").css({  
            color: "red"});  
        return false;  
    }  
  
    if (Number.isNaN(price))  
    {  
        $("#rezultat2").text("Niste prosledili ispravnu cenu proizvoda").  
            css({color: "red"});  
        return false;  
    }  
});
```

```
}

$.ajax("http://localhost:3000/products", {
    method: "POST",
    data: JSON.stringify({
        name, price
    }),
    contentType: 'application/json',
    success: function(data) {
        if(data.hasOwnProperty("kreiraniProizvod")) {
            $("#rezultat2")
                .text(`Kreirali ste proizvod:
${JSON.stringify(data.kreiraniProizvod)}`)
                .css({color: "green"});
        }
    },
    error: function() {
        $("#rezultat2").text("Greska pri postavljanju proizvoda!").css(
            {color: "red"});
    }
});

return false;
});

$("#form3").submit(function()
{
    let productId = $("#productId3").val();

    if (productId === "")
    {
        $("#rezultat3").text("Niste prosledili identifikator proizvoda").
            css({color: "red"});
        return false;
    }

    $.ajax(`http://localhost:3000/products/${productId}`, {
        method: "PATCH",
        success: function(data) {
            if(data.hasOwnProperty("poruka")) {
                $("#rezultat3").text(data.poruka).css({color: "green"});
            }
        },
        error: function() {
            $("#rezultat3").text("Greska pri menjaju proizvoda!").css({
                color: "red"});
        }
    });

    $("#rezultat3").text("");
    return false;
});
});
```

```
$("#form4").submit(function()
{
    let productId = $("#productId4").val();

    if (productId === "")
    {
        $("#rezultat4").text("Niste prosledili identifikator proizvoda").
            css({color: "red"});
        return false;
    }

    $.ajax(`http://localhost:3000/products/${productId}`, {
        method: "DELETE",
        success: function(data) {
            if(data.hasOwnProperty("poruka")) {
                $("#rezultat4").text(data.poruka).css({color: "green"});
            }
        },
        error: function() {
            $("#rezultat4").text("Greska pri brisanju proizvoda!").css({
                color: "red"});
        }
    });

    $("#rezultat4").text("");
    return false;
});
});
```

**Zadatak 10.1:**

Dopuniti kod prethodnog primera tako da server opsluži i naredne zahteve:

- GET zahtev nad `http://localhost:3000/orders`
  - Vratiti statusni kod 200 sa porukom `"Narudzbine su pristigle!"`
- POST zahtev nad `http://localhost:3000/orders`
  - Kreirati objekat koji ima svojstva `productID` i `quantity`, čije se vrednosti očitavaju iz tela zahteva. Telo se prosleđuje u JSON formatu. Vratiti statusni kod 201 sa porukom `"Narudzba je upisana"` i vratiti novokreirani objekat.
- GET zahtev nad `http://localhost:3000/orders` sa parametrom `orderID`
  - Očitati parametar `orderID` i vratiti statusni kod 200 uz poruku ``Narudzba ${orderID} je pristigla``.
- DELETE zahtev nad `http://localhost:3000/orders` sa parametrom `orderID`
  - Očitati parametar `orderID` i vratiti statusni kod 200 uz poruku ``Narudzba ${orderID} je obrisana``.

**Rešenje:**

Implementacija na serverskoj strani:

Izmene u kodu datoteke app.js iz prethodnog primera:

```
const express = require('express');
const bodyParser = require('body-parser');

const productRoutes = require('./api/routes/products');
// Dodata je naredna linija koda
const orderRoutes = require('./api/routes/orders');

/* ... */

app.use('/products', productRoutes);
// Dodata je naredna linija koda
app.use('/orders', orderRoutes);

/* ... */
```

Kod 81: zadaci/007/api/routes/orders.js

```
const express = require('express');
const router = express.Router();

router.get('/', function(req, res, next) {
    res.status(200).json({
        poruka: 'Narudzbine su pristigle!'
    });
});

router.post('/', function(req, res, next) {
    const order = {
        productID: req.body.productID,
        quantity: req.body.quantity
    };

    res.status(201).json({
        poruka: 'Narudzba je upisana',
        order: order
    });
});

router.get('/:orderID', function(req, res, next) {
    const orderID = req.params.orderID;

    res.status(200).json({
        poruka: 'Narudzba ' + orderID + ' je pristigla'
    });
});

router.delete('/:orderID', function(req, res, next) {
    const orderID = req.params.orderID;
```

```
res.status(200).json({  
    poruka: 'Narudzbina ' + orderID + ' je obrisana'  
});  
  
module.exports = router;
```

## 11 Čas 13

Cilj ovog časa je upoznavanje studenata sa kreiranjem serverskih aplikacija koji komuniciraju sa MongoDB bazom podataka. Nakon ovog časa, student bi trebalo da razume način funkcionisanja MongoDB baze podataka, kao i da samostalno kreira modele podataka, i da upravlja njima kroz četiri osnovne operacije: pravljenje novih podataka, kao i čitanje, menjanje i brisanje postojećih podataka (engl. Create, Read, Update, Delete, skr. *CRUD*).

### 11.1 MongoDB

MongoDB je NoSQL baza dokumenata. Podaci se čuvaju kao par ključ-vrednost gde je vrednost dokument. Dokument se čuva u JSON ili XML formatu. Bazu čine kolekcije dokumenata. Svaki dokument može biti različit, sa proizvoljenim brojem polja, veličine i sadržaja.

Sledeći pojmovi su bitni za razumevanje MongoDB:

- **\_id** — Ovo je polje obavezno za svaki dokument u MongoDB bazi. Predstavlja jedinstvenu vrednost po kojoj razlikujemo dokumente u bazi. Pošto je polje obavezno, ukoliko pokušamo da napravimo novi dokument bez njega, biće automatski dodato.
- **Kolekcija** — Predstavlja grupisane dokumente. Kolekcija postoji unutar jedne baze. Kao što smo već napomenuli, kolekcije nemaju definisanu strukturu, svaki dokument može biti različit.
- **Kursor** — Pokazivač na rezultujući skup našeg upita. Klijenti mogu iterirati kroz ovaj skup kako bi dobili rezultate.
- **Baza podataka** — Skladište za kolekcije. Svaka baza ima svoj skup datoteka.
- **Dokument** — Jedan zapis u kolekciji. Sastoji se od naziva polja i vrednosti.
- **Polje** — Par (*ime, vrednost*) jednog dokumenta. Dokument može imati 0 ili više polja.
- **JSON** — Notacija za predstavljanje strukturiranih podataka u čitljivom formatu.

#### 11.1.1 Alat MongoDB Compass

MongoDB Compass predstavlja korisnički interfejs za MongoDB. Pomoću njega ćemo lako vizuelizovati naše baze. Alat prikazuje sve baze koje smo napravili na serveru i nudi mogućnost za pravljenje novih, kao i za modifikovanje postojećih. Takođe, možemo izvršavati upite nad konkretnom bazom i pretraživati postojeće dokumente.

Kada se alat pokrene, otvara se prozor kao na slici 1 koji nam omogućava da se povežemo sa serverom gde se nalazi naša baza. Pošto ćemo sve raditi lokalno, povezujemo se na localhost, što je i podrazumevana vrednost pa nisu potrebna nikakva dodatna podešavanja i možemo pritisnuti zeleno dugme za konekciju.

Connect to Host

---

Hostname

Port

SRV Record

---

Authentication

---

Replica Set Name

Read Preference

---

SSL

---

SSH Tunnel

---

Favorite Name (i)

**CONNECT**

Slika 1: Konekcija sa serverom

Kada se uspešno povežemo na server otvara se prozor u kom su prikazane osnovne informacije o svim bazama skladištenim na serveru. Klikom na dugme CREATE DATABASE možemo napraviti novu bazu. Otvara se novi prozor, kao na slici 3, koji omogućava da napravimo novu bazu sa imenom koji ne sme sadržati beline. Pored toga, baza uvek mora da sadrži bar jednu kolekciju da bi postojala na serveru zbog čega se zahteva i ime jedne kolekcije prilikom kreiranja baze. Pritisakom na dugme CREATE DATABASE napravljen je nova baza i otvara se prozor u kom je prikazan sadržaj novokreirane baze, kao na slici 4.

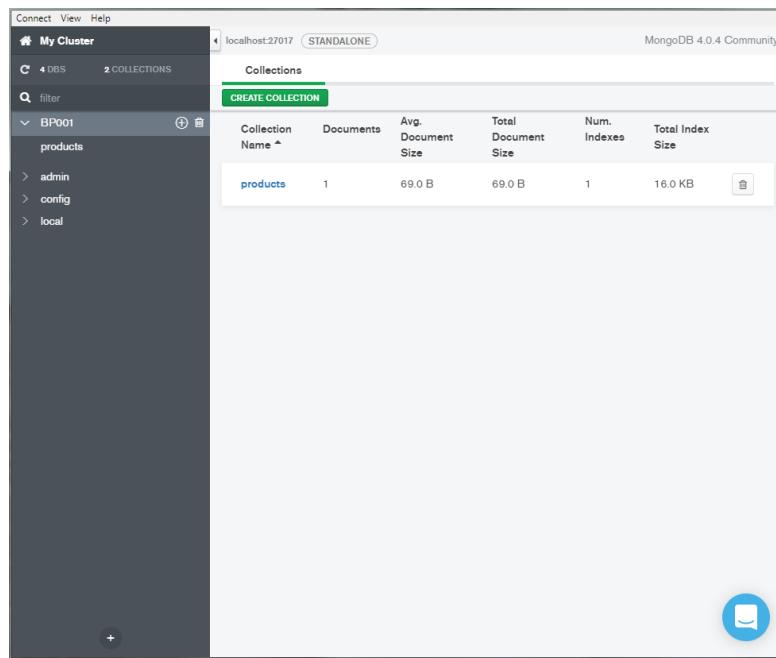
The screenshot shows the MongoDB Compass application interface. On the left, there's a sidebar titled 'My Cluster' with a 'filter' search bar. It displays '3 DBS' and '1 COLLECTIONS'. Below this are three database entries: 'admin', 'config', and 'local'. The main area is titled 'Databases' and 'Performance'. A green 'CREATE DATABASE' button is at the top. Below it is a table with columns: Database Name, Storage Size, Collections, and Indexes. The data for the three databases is as follows:

Database Name	Storage Size	Collections	Indexes
admin	16.0KB	0	1
config	36.0KB	0	2
local	36.0KB	1	1

Slika 2: Baze podataka na serveru

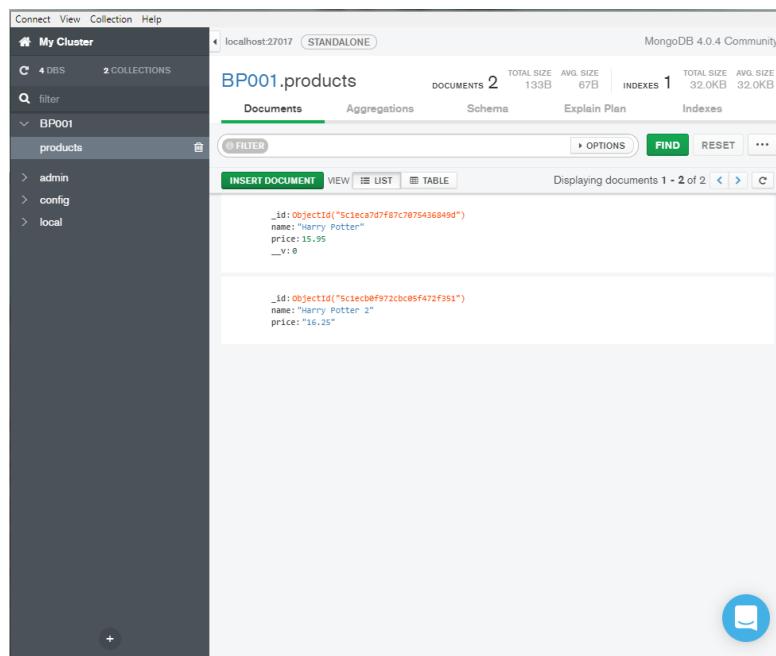
This screenshot shows the 'Create Database' dialog box. At the top, it says 'Create Database'. The first field is 'Database Name', which is empty. Below it is 'Collection Name', also empty. There are two checkboxes: 'Capped Collection' and 'Use Custom Collation', neither of which is checked. A note at the bottom states: 'Before MongoDB can save your new database, a collection name must also be specified at the time of creation.' followed by a link 'More Information'. At the bottom right are 'CANCEL' and 'CREATE DATABASE' buttons.

Slika 3: Kreiranje nove baze podataka



Slika 4: Kreiranje nove baze podataka

Sada možemo da dodajemo nove kolekcije u bazu ili da menjamo postojeće. Klikom na dugme CREATE COLLECTION možemo napraviti novu kolekciju. Klikom na naziv kolekcije otvara se prozor u kom su prikazani osnovni podaci o kolekciji i njen sadržaj, kao na slici 5. Možemo da izaberemo format u kom će podaci biti prikazani. To može biti format liste, kao na slici, ili tabelarni prikaz.



Slika 5: Kreiranje nove baze podataka

Dugme `INSERT DOCUMENT` omogućava da dodamo novi dokument u kolekciju. Otvara se prozor kao na slici 6. Za svaki novi dokument generiše se jedinstvena vrednost za `_id`. Možemo dodati proizvoljan broj polja. Prelaskom mišem preko jednog reda prikazuje se dugme za brisanje tog polja (prvo s leva, X), zatim dugme za dodavanje novog polja ispod trenutnog (drugo s leva, +), i dugme za odabir tipa polja (prvo s desna, strelica na dole koja otvara padajući meni). Dokument nije moguće kreirati sve dok postoje greške u unosu. Kada su svi podaci ispravno uneti, može se pritisnuti dugme `INSERT` i novi dokument je dodat u kolekciju.



Slika 6: Kreiranje nove baze podataka

Prelaskom miša preko nekog dokumenta pojavljuju se dugmići sa desne strane. Prvo služi sa ažuriranje dokumenta. Za svako polje prikazuju se dugmići kao i prilikom kreiranja, sa dodatkom kružne strelice, ukoliko je neko polje izmenjeno, koje omogućava da se sve vrednosti vrati na stare za odabrano polje. Dozvoljeno je menjati nazive, vrednosti i tipove polja, brisati i dodavati nova.

Sledeće dugme služi za kopiranje dokumenta. Zatim se nalazi dugme za kloniranje dokumenta koje otvara isti prozor kao i dugme za pravljenje novog dokumenta, s tim da novi dokument sadrži sva polja sa istim vrednostima kao i onaj koji kloniramo (osim polje `_id` koje je jedinstveno za svaki dokument). Kloniranom dokumentu možemo dodavanja novih polja, brisati i menjati postojećih.

Na kraju se nalazi i dugme za brisanje koje zahteva potvrdu pre nego što se dokument zaista obriše iz kolekcije.

Iznad dela sa podacima, nalazi se tekstualno polje u koje možemo uneti upit i lako pronaći podatke u kolekciji. Na ovaj način možemo izvršiti upite koje prosleđujemo funkciji `find()` o čemu će biti reči dalje u tekstu.

### 11.1.2 Upiti

Prepostavimo da imamo kolekciju `studenti` u kojoj se čuvaju dokumenti sa podacima o pojedinačnim studentima. Da bismo dohvatali podatke iz baze moramo napisati upit koji specifikuje kakvi podaci su nam potrebni. Upite pišemo kao objekte.

Ako bismo želeli da dohvativamo podatke o svim studentima u kolekciji, koristili bismo prazan upit, odnosno

```
{}
```

Često nije potrebno da dohvativamo sve podatke, već neke specifične, odnosno, sa određenim vrednostima za neka polja. U tom slučaju, upit je objekat sa svojstvima, koja odgovaraju poljima u dokumentima, i vrednostima koje tražimo

```
{ svojstvo1 : vrednost1, svojstvo2: vrednost2, ... }
```

Na primer, ako bismo iz kolekcije `student` želeli da izvučemo podatke o svim studentima sa imenom "Jovan", koristili bismo upit

```
{ime : "Jovan"}
```

Ako su nam potrebni još precizniji detalji o studentima, možemo dodati još svojstava u objekat. Na primer, ako bismo želeli sve studente koji se zovu "Jovan", i imaju prosek jednak 8.5, napisali bismo upit

```
{
  ime : "Jovan",
  prosek: "8.5"
}
```

Na ovaj način dobijamo poređenje vrednosti po jednakosti. Nekada će nam biti potrebno da pronađemo dokumente sa vrednostima koje su manje ili veće od zadate, ili koje su u nekom intervalu, itd. Definisana su posebna svojstva koja možemo pisati u upitu koja se tiču ovog problema:

- \$gt — traži vrednosti veće od zadate
- \$gte — traži vrednosti veće ili jednake zadatoj
- \$lt — traži vrednosti majne od zadate
- \$lte — traži vrednosti manje ili jednake zadatoj
- \$ne — traži vrednosti koje nisu jednake zadatoj
- \$eq — traži vrednosti jednake zadatoj
- \$in — traži vrednosti jednake nekoj iz zadatog niza vrednosti
- \$nin — traži vrednosti nisu jednake nijednoj iz zadatog niza vrednosti

Ako želimo studente sa prosekom većim od 8.5, pisali bismo upit

```
{
  prosek : {
    $gt : "8.5"
  }
}
```

Studente sa prosekom iz intervala [8.0, 9.0] možemo dohvatiti upitom

```
{
  prosek : {
    $gte : "8.0",
    $lte : "9.0"
  }
}
```

Ukoliko želimo da dohvatimo studente I i R smerova, koristili bismo upit

```
{
  smer : {
    $in : ['I', 'R']
  }
}
```

Pored toga, možemo koristiti i svojstva koja imaju ulogu logičkih operatora:

- \$and — vraća sve dokumente koji su ispunili uslove oba upita

- `$or` — vraća sve dokumente koji su ispunili uslove bar jedan od upita
- `$not` — vraća sve dokumente koji nisu ispunili uslove upita
- `$nor` — vraća sve dokumente koji nisu ispunili uslove nijednog upita

Vrednosti ovih svojstava su nizovi koji predstavljaju logičke jedinice koje se povezuju odgovarajućim logičkim operatorom. Na primer, ako želimo da dohvativamo sve studente čiji je prosek veći od 8.0 i čija je godina upisa 2018, koristili bismo upit

```
{
  $and: [
    { prosek: { $gt: 8.0 } },
    { godinaStudija: 2018 }
  ]
}
```

Prethodni upit predstavlja konjunkciju, i on se jednostavnije zapisuje navođenjem zapete između uslova poređenja:

```
{
  prosek: { $gt: 8.0 },
  godinaStudija: 2018
}
```

Ukoliko imamo uslove koji su nešto kompleksniji, onda moramo koristiti `$and` eksplicitno. Na primer, ako je potrebno izdvojiti informacije o studentima čiji je prosek jednak 9.0 ili 10.0 i čija je godina studija 2017 ili 2018, onda možemo napisati upit:

```
{
  $and: [
    $or: [ { prosek: 9.0 }, { prosek: 10.0 } ],
    $or: [ { godinaStudija: 2017 }, { godinaStudija: 2018 } ]
  ]
}
```

## 11.2 Razvojno okruženje Mongoose.js

Iako je moguće komunicirati iz Node.js aplikacije ka MongoDB bazi podataka pomoću zvaničnog MongoDB drafvera koji se razvija od strane razvijača MongoDB baze podataka<sup>31</sup>, mi ćemo u našim aplikacijama koristiti razvojno okruženje Mongoose.js<sup>32</sup>. Razlog za ovu odluku jeste što nam Mongoose.js omogućava da pišemo kod koji se poprilično dobro uklapa u ono što već znamo. Drugim rečima, na ovom času ćemo moći da upotrebimo kod koji smo napisali na prethodnom času bez i jedne konceptualne promene. Predimo na naredni primer.

---

<sup>31</sup>Više informacija se može pronaći na zvaničnoj stranici projekta [ovde](#).

<sup>32</sup>Zvanična veb prezentacija projekta se nalazi [ovde](#), dok je dokumentaciju moguće pronaći [ovde](#).

**Primer 11.1:**

Dopuniti aplikaciju iz primera 10.1 tako da se umesto odgovarajućih HTTP zahteva opslužuju naredni zahtevi:

- GET zahtev nad `http://localhost:3000/products`
  - Vratiti statusni kod 200 sa listom svih dokumenata iz baze podataka koji odgovaraju proizvodima sortirane rastuće po imenu i opadajuće po ceni
- POST zahtev nad `http://localhost:3000/products`
  - Kreirati novi proizvod koji ima svojstva `name` i `price`, čije se vrednosti očitavaju iz tela zahteva. Objekat konstruisati na osnovu odgovarajućeg modela. Telo se prosleđuje u JSON formatu. Po uspešnom upisu u bazu podataka, vratiti statusni kod 201 sa porukom "Proizvod je uspesno kreiran" i vratiti novokreirani objekat. Za neuspešan upis vratiti statusni kod 500 i grešku.
- GET zahtev nad `http://localhost:3000/products` sa parametrom `productId`
  - Očitati parametar `productId` i pronaći proizvod u bazi podataka čiji je identifikator jednak prosleđenom parametru. Ukoliko je podatak pronađen, vratiti statusni kod 200 i rezultat. U suprotnom, proslediti klijentu statusni kod 404 sa porukom "Ne postoji traženi proizvod"
- PATCH zahtev nad `http://localhost:3000/products` sa parametrom `productId`
  - Očitati parametar `productId` i proveriti šta je od parametara `name` i `price` klijent prosledio u telu zahteva. Izmeniti prosleđene podatke za proizvod čiji je identifikator jednak `productId`. Ukoliko je podatak uspešno izmenjen, obvestiti klijenta o uspešnosti porukom i statusnim kodom 200. U suprotnom, vratiti statusni kod 500 i za poruku iskoristiti grešku.
- DELETE zahtev nad `http://localhost:3000/products` sa parametrom `productId`
  - Očitati parametar `productId` i obrisati proizvod čiji je identifikator jednak `productId`. Ukoliko je podatak uspešno obrisan, obvestiti klijenta o uspešnosti porukom i statusnim kodom 200. U suprotnom, vratiti statusni kod 500 i za poruku iskoristiti grešku.
- GET zahtev nad `http://localhost:3000/orders`
  - Vratiti statusni kod 200 sa ukupnim brojem narudžbina i listom narudžbina. Postaviti da se u svakoj porudžbini prikaže samo identifikator proizvoda, ne i njegova količina.
- POST zahtev nad `http://localhost:3000/orders`
  - Kreirati novu porudžbinu koja ima svojstva `product` i `quantity`, čije se vrednosti očitavaju iz tela zahteva. Objekat konstruisati pomoću

odgovarajućeg modela. Telo se prosleđuje u JSON formatu. Po uspešnom upisu u bazu podataka, vratiti statusni kod 201 sa porukom "Narudzbina je upisana" i vratiti novokreirani objekat. Za neuspešan upis vratiti statusni kod 500 i grešku.

- GET zahtev nad `http://localhost:3000/orders` sa parametrom `orderID`
  - Očitati parametar `orderID` i pronaći narudžbine u bazi podataka čiji je identifikator jednak prosledenom parametru. Ukoliko je podatak pronađen, vratiti statusni kod 200 i rezultat. Rezultat treba da sadrži i podatke o samom proizvodu. U suprotnom, proslediti klijentu statusni kod 404 sa porukom "Ne postoji tražena porudzbina"
- DELETE zahtev nad `http://localhost:3000/orders` sa parametrom `orderID`
  - Očitati parametar `orderID` i obrisati narudžbinu čiji je identifikator jednak `orderID`. Ukoliko je podatak uspešno obrisan, obavestiti klijenta o uspešnosti porukom i statusnim kodom 200. U suprotnom, vratiti statusni kod 500 i za poruku iskoristiti grešku.

Implementirati odgovarajuće sheme i modele. Koristiti razvojno okruženje Mongoose.js za komunikaciju sa MongoDB bazom podataka i implementaciju odgovarajućih CRUD operacija nad definisanim modelima. Ispravno obraditi sve greške u komunikaciji sa bazom podataka.

**Rešenje:** Kao što smo rekli, za početni kod koristimo implementaciju iz primera 10.1. Datoteka `server.js` ostaje netaknuta. Datoteka `app.js` ima jednostavnu izmenu — potrebno je da uvezemo paket '`mongoose`' i da se povežemo na bazu podataka. Pre samog uvoženja, moramo da se osiguramo da je paket instaliran, što možemo učiniti pozivom

```
$ npm install mongoose
```

Sada ga možemo uvesti u našu aplikaciju:

```
const mongoose = require('mongoose');
```

**Konekcija na sistem za upravljanje bazom podataka** Da bismo kreirali globalnu konekciju ka bazi, potrebno je da pozovemo funkciju `mongoose.connect(uri [, options])`<sup>33</sup>. Prvi argument je URL ka bazi podataka. Očekuje se da ovaj argument bude u formatu

```
mongodb://user:pass@localhost:port/database
```

Nisu sve opcije obavezne, na primer, mi nećemo eksplicitno upisivati korisničko ime i lozinku. Drugi argument čine opcije za povezivanje. Potrebno je proslediti objekat sa svojstvom `useNewUrlParser` čija je vrednost `true`.

```
mongoose.connect("mongodb://127.0.0.1:27017/BP001", {
  useNewUrlParser: true
});
```

Time smo završili sa izmenama u `app.js` datoteci. Predimo sada na implementiranje obrade zahteva.

<sup>33</sup> Dokumentacija

**Sheme i modeli** Da bismo to uradili, moramo da razumemo kako Mongoose.js funkcione. Potrebno je da kreiramo *shemu* (engl. *scheme*) za objekte sa kojima radimo. Shema nije nista drugo do opis svega onoga što naši objekti sadrže u sebi, odnosno, onoga što će biti sačuvano u bazi podataka. Kada kreiramo nove objekte ili čitamo postojeće iz baze, uvek ćemo specifikovati koju shemu želimo da koristimo. Kada je shema definisana, onda se od nje konstruišu *modeli* (engl. *model*), kojima se pored informacija o objektima dodaju razne metode za njihovo upravljanje. Razvojno okruženje Mongoose.js definiše pregršt metoda za rad sa modelima i mi ćemo upotrebiti neke od njih da bismo implementirali rešenja za naše zahteve. Sve modele ćemo čuvati na nivou direktorijuma ./api/models.

**Proizvodi** Prvo definišimo model za naše proizvode u datoteci product.js. Pomoću funkcije `mongoose.Schema([definition][, options])`<sup>34</sup> pravimo novu shemu. Prvi argument ove funkcije je objekat, čija su svojstva opisi sheme. Mi želimo da naši proizvodi imaju naziv i cenu, ali je potrebno da imaju i jedinstveni identifikator kako bi mogli da ih identifikujemo<sup>35</sup>:

```
const mongoose = require('mongoose');

const productSchema = mongoose.Schema({
  _id: mongoose.Schema.Types.ObjectId,
  name: String,
  price: Number
});
```

Kao što vidimo, objekat koji definiše shemu ima svojstva čiji nazivi odgovaraju poljima u MongoDB bazi podataka, dok su vrednosti ovih svojstava tipovi polja. Više o tipovima koje postoje i konstrukciji shema se može pronaći [ovde](#). Napomenimo samo da je `mongoose.Schema.Types.ObjectId` specijalan tip za identifikatore.

Kreiranje modela na osnovu sheme se jednostavno postiže pozivom funkcije `mongoose.model(name [, schema][, collection][, skipInit])`<sup>36</sup>. Prvi argument ove funkcije je niska kojom se definiše naziv modela. Uobičajena je konvencija da se imena modela pišu početnim velikim slovom. Drugi argument ove funkcije je shema na osnovu koje se konstruiše model. Dakle, kreiranje modela proizvoda se može postići pozivom (koji ga ujedno i eksportuje):

```
module.exports = mongoose.model("Product", productSchema);
```

Predimo sada na implementaciju zahteva za proizvode u datoteci products.js u direktorijumu ./api/routes. Prvo što je potrebno jeste da uključimo paket `mongoose` i prethodno definisani model:

```
const mongoose = require('mongoose');
const Product = require('../models/product');
```

Rutiranje ostaje isto, jedina razlika jeste u definiciji registrovanih funkcija. Dohvatavanje dokumenata iz MongoDB baze podataka je moguće izvesti pozivom statičke funkcije `find(conditions [, projection][, options][, callback])`<sup>37</sup>. Prvi argument ove funkcije je objekat kojim se definišu kriterijumi koji dokumenti moraju da ispune da bi bili odabrani. Drugim rečima, prvi argument je upit ka bazi

<sup>34</sup> [Dokumentacija](#)

<sup>35</sup> Slično kao što osobe identifikujemo po JMBG broju.

<sup>36</sup> [Dokumentacija](#)

<sup>37</sup> [Dokumentacija](#)

podataka. Poslednji argument je funkcija koja će se pozvati kada se akcija završi, bilo uspešno ili neuspešno. Sada kod možemo napisati na sledeći način:

```
router.get('/', function(req, res, next) {
  Product.find({}, function(err, result)
  {
    if (err)
    {
      return res.status(500).json({reason: err});
    }

    res.status(200).json({
      rezultat: result
    });
  });
});
```

Kako bismo sortirali dokumente, možemo koristiti funkciju `sort(arg)`<sup>38</sup> koju pozivamo nad rezultatom funkcije `find`. Funkcija kao argument prima jedan objekat u kome definišemo po kojim poljima želimo da sortiramo rezultujuće dokumente. Ako želimo da sortiramo podatke po polju `name` rastuće, prosledili bismo objekat `{ name : 1 }`, a za opadajući poredak iskoristili bismo vrednost `-1`. Moguće je postaviti više od jednog kriterijuma za sortiranje, pri čemu se dodaju nova stvojstva objektu sa odgovarajućim vrednostima, kao u narednom kodu:

```
router.get('/', function(req, res, next) {
  Product.find({}, function(err, result)
  {
    if (err)
    {
      return res.status(500).json({reason: err});
    }

    res.status(200).json({
      rezultat: result
    });
  }).sort({name : 1, price : -1});
});
```

Kreiranje novih instanci i njihovo čuvanje je takođe jednostavno. Sve što je potrebno da kreiramo novu instancu jeste da pozovemo konstruktor funkciju koja odgovara nazivu modela i prosledimo joj objekat sa odgovarajućim vrednostima za polja modela:

```
const product = new Product({
  _id: new mongoose.Types.ObjectId(),
  name: req.body.name,
  price: req.body.price
});
```

Kada je instanca kreirana, onda nad njom možemo pozvati metod `save([options][, fn])`<sup>39</sup> koja će funkciji `fn` proslediti ili grešku ili objekat koji je uspešno upisan u bazu podataka:

<sup>38</sup>[Dokumentacija](#)

<sup>39</sup>[Dokumentacija](#)

```

product.save(function(err, result) {
  if (err)
  {
    return res.status(500).json({reason: err});
  }

  res.status(201).json({
    poruka: 'Proizvod je uspesno kreiran',
    kreiraniProizvod: result
  });
});

```

Da bismo pronašli instancu u bazi podataka sa određenim identifikatorom, možemo iskoristiti staticku funkciju `findById(id [, projection][, options][, callback])`<sup>40</sup> nad modelom, čiji je prvi argument identifikator po kojem se traži podatak, a poslednji argument funkcija koja će dobiti grešku ili pronađeni podatak:

```

router.get('/:productId', function(req, res, next) {
  const productId = req.params.productId;

  Product.findById(productId, function(err, result)
  {
    // Provera da li je doslo do greske
    if (err)
    {
      return res.status(500).json({reason: err});
    }

    // Provera da li je pronađen objekat
    if (result)
    {
      return res.status(200).json({rezultat: result});
    }

    // Ukoliko nije pronađen objekat
    return res.status(404).json({poruka: "Ne postoji traženi proizvod"});
  });
});

```

Možemo postaviti i drugi argument, `projection`, kako bismo dobili samo polja čije vrednosti su nam potrebne. Ovo se radi na isti način kao za funkciju `find`.

Za ažuriranje postojećih podataka u bazi možemo koristiti staticku funkciju `updateOne(conditions, doc [, options][, callback])`<sup>41</sup> nad modelom, koji ažuriра jedan podatak u bazi. Njen prvi argument je kriterijum za pronađenje podatka. Njen drugi argument je opis kako treba izmeniti pronađeni podatak. Na primer, mogli bismo ovaj metod pozvati na sledeći način:

```

Product.updateOne(
  {_id: productId}, // Trazimo podatak po identifikatoru
  { $set:      // Ovo je specijalna sintaksa koju MongoDB razume
    // a kojom kazemo da zelimo da postavimo vrednosti
  }

```

<sup>40</sup>[Dokumentacija](#)

<sup>41</sup>[Dokumentacija](#)

```

        // Ovde idu parovi kljuc-vrednost
        // kojima se opisuje sta treba da se menja
        name: req.body.newName,
        price: req.body.newPrice
    }
},
function(err, raw)
{
    // Uraditi proveru za gresku
}
));

```

Problem sa ovim pristupom jeste što se očekuje da ćemo uvek menjati i ime i cenu, što ne mora nužno da bude slučaj. Umesto toga, možemo da tražimo od klijenta da nam pošalje listu svega što je potrebno da se menja i novih vrednosti, a mi ćemo od te liste napraviti objekat koji ćemo staviti kao vrednost za `$set`. Na primer, možemo zahtevati da korisnik pošalje telo u formatu:

```
[
{
    "nazivPolja": "name",
    "novaVrednost": "Harry Potter 2"
},
{
    "nazivPolja": "price",
    "novaVrednost": "15.95"
}
]
```

Očigledno, gornjim sadržajem tela zahteva kažemo da želimo da promenimo i naziv i cenu, ali ako bismo uklonili jedno od njih, na primer:

```
[
{
    "nazivPolja": "name",
    "novaVrednost": "Harry Potter 2"
}
]
```

onda bismo rekli da želimo da promenimo samo naziv. Naravno, identifikator objekta koji menjamo se nalazi u parametru zahteva, tako da njega ne moramo da prosleđujemo u telu zahteva.

Dakle, sada možemo da iteriramo kroz telo (pošto se šalje kao niz) i da pamtimo sve što smo pronašli u nekom objektu:

```
const updateOptions = {};
for (let i = 0; i < req.body.length; ++i)
{
    let option = req.body[i];
    updateOptions[option.nazivPolja] = option.novaVrednost;
}
```

Sada možemo da ažuriramo podatak u bazi podataka narednim kodom:

```
Product.updateOne(
    {_id: productId},
    { $set: updateOptions },
```

```

function(err, raw)
{
  if (err)
  {
    return res.status(500).json({message: err});
  }

  res.status(200).json({message: "Successfully updated"});
}
);

```

Za brisanje jednog podatka iz baze podataka možemo koristiti statičku funkciju `deleteOne(conditions [, callback])`. Njen prvi argument je objekat sa kriterijumima koje objekat mora da zadovolji, a drugi je funkcija koja dobija objekat greške:

```

router.delete('/:productId', function(req, res, next) {
  const productId = req.params.productId;

  Product.deleteOne({_id: productId}, function(err)
  {
    if (err)
    {
      return res.status(500).json({poruka: err});
    }

    res.status(200).json({poruka: "Proizvod je uspesno uklonjen"});
  });
});

```

**Narudžbine** Pređimo sada na obradu zahteva za narudžbine. Definicija sheme je data u datoteci `./api/models/order.js` i izgleda:

```

const mongoose = require('mongoose');

const orderSchema = mongoose.Schema({
  _id: mongoose.Schema.Types.ObjectId,
  product: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'Product',
    required: true
  },
  quantity: {
    type: Number,
    default: 1
  }
});

module.exports = mongoose.model('Order', orderSchema);

```

U ovoj shemi vidimo dosta nove sintakse. Za početak, primetimo da ukoliko je potrebno navesti dodatne opcije za neko polje u bazi, umesto samo navođenja tipa polja kao što smo to radili u shemi za proizvode, možemo proslediti objekat sa detaljnijom konfiguracijom. U toj konfiguraciji možemo podesiti nekoliko vrednosti:

- `type` — tip polja u kolekciji.
- `ref` — naziv modela na koji se polje odnosi. Ova vrednost se najčešće koristi kada u jednoj kolekciji želimo da sačuvamo informaciju o identifikatoru podataka iz neke druge kolekcije. U ovom primeru, u modelu Order želimo da čuvamo informacije o identifikatorima modela Product.
- `required` — Bulova vrednost koja definiše da li je polje obavezno ili ne. Doći će do greške ukoliko se vrednost ne postavi.
- `default` — vrednost koja će biti iskorišćena ukoliko se ne prosledi druga vrednost za ovo polje.

Implementacija obrade zahteva za narudžbine će biti u datoteci `orders.js` u direktorijumu `./api/routes/`.

Funkcija za čitanje iz baze vraća cele dokumente koji ispunjavaju uslove upita. Mi možemo da izdvojimo podskup polja iz dokumenta koji su nam potrebni navođenjem vrednosti za drugi, opcioni, argument funkcije `find — projection`. Ovaj argument je objekat kojim definišemo koja polja želimo da koristimo. Za svako polje koje želimo u rezultatu dodajemo svojstvo sa istim nazivom kao polje, a vrednost postavljamo na 1 (što označava `true`), ili ako ne želimo neko polje, vrednost možemo postaviti na 0 (što označava `false`). Na primer, ako nakon upita prosledimo objekat `{ name: 1 }`, biće nam vraćeni svi dokumenti koji zadovoljavaju uslove upita, a moći ćemo da pročitamo samo polje `name`. Celokupan kod je zadan sa:

```
router.get('/', function(req, res, next) {
  Order.find({}, {product : 1}, function(err, result)
  {
    if (err)
    {
      return res.status(500).json({poruka: err});
    }

    res.status(200).json({
      count: result.length,
      orders: result
    });
  });
});
```

Kada imamo povezane kolekcije, kao što su u ovom primeru Orders i Products — u Orders čuvamo identifikator proizvoda koji je naručen, može nam biti od koristi da iz baze dohvativimo i informacije o proizvodu na osnovu identifikatora. Na rezultat funkcije `find` i `findById` možemo nadovezati poziv funkcije `populate(path [, select][, model][, match][, options])42` koja će umesto identifikatora prosleđen kroz parametar `path` ubaciti objekat sa podacima o proizvodu koji tražimo. U pozadini se vrši pretraga kolekcije koja je zadata kao vrednost svojstva `ref` u shemi. Dakle, sve što treba da prosledimo funkciji jesu nazivi polja koje treba dopuniti razdvojeni blanko karakterom:

```
router.get('/:orderId', function(req, res, next) {
  const orderId = req.params.orderId;
```

<sup>42</sup> Dokumentacija

```

Order.findById(orderID, function(err, result)
{
    if (err)
    {
        return res.status(500).json({poruka: err});
    }

    if (result)
    {
        return res.status(200).json(result);
    }

    res.status(404).json({poruka: "Ne postoji trazena narudzbina"});
}).populate('product');
});
```

Zahtevi za POST i DELETE nad narudžbinama sada ne bi trebalo da budu zahtevni za implementiranje, te se ostavljaju čitaocu za vežbu. Ceo izvorni kod implementacije na serveru dat je u nastavku:

Kod 82: primeri/nodejs3/001/server.js

```

const http = require('http');
const app = require('./app');

const server = http.createServer(app);

const port = process.env.PORT || 3000;
server.listen(port);

server.once('listening', function()
{
    console.log(`http://localhost:${port}`);
});
```

Kod 83: primeri/nodejs3/001/app.js

```

const express = require('express');
const bodyParser = require('body-parser');
const mongoose = require('mongoose');

const productRoutes = require('./api/routes/products');
const orderRoutes = require('./api/routes/orders');

const app = express();

mongoose.connect("mongodb://127.0.0.1:27017/BP001", {
    useNewUrlParser: true
});

app.use(bodyParser.urlencoded({
    extended: false
}));
app.use(bodyParser.json());
```

```
app.use(function(req, res, next) {
    res.header('Access-Control-Allow-Origin', '*');
    res.header('Access-Control-Allow-Headers', 'Content-Type');

    if (req.method === 'OPTIONS')
    {
        res.header('Access-Control-Allow-Methods', 'OPTIONS, GET, POST,
                  PATCH, DELETE');

        return res.status(200).json({});
    }

    next();
});

app.use('/products', productRoutes);
app.use('/orders', orderRoutes);

app.use(function(req, res, next) {
    const error = new Error('Zahtev nije podrzan od servera');
    error.status = 405;

    next(error);
});

app.use(function(error, req, res, next) {
    res.status(error.status || 500).json({
        greska: {
            poruka: error.message
        }
    });
});

module.exports = app;
```

Kod 84: primeri/nodejs3/001/api/models/product.js

```
// U ovom fajlu definisemo model koji ce imati nasi proizvodi
const mongoose = require('mongoose');

const productSchema = mongoose.Schema({
    _id: mongoose.Schema.Types.ObjectId,
    name: String,
    price: Number
});

module.exports = mongoose.model("Product", productSchema);
```

Kod 85: primeri/nodejs3/001/api/routes/products.js

```
const express = require('express');
const mongoose = require('mongoose');
```

```
const router = express.Router();

const Product = require('../models/product');

router.get('/', function(req, res, next) {
  Product.find({}, function(err, result)
  {
    if (err)
    {
      return res.status(500).json({reason: err});
    }

    res.status(200).json({
      rezultat: result
    });
  }).sort({name : 1, price : -1});
});

router.post('/', function(req, res, next) {
  // Pravimo novu instancu Product modela
  // na osnovu prosledjenih parametara
  const product = new Product({
    _id: new mongoose.Types.ObjectId(),
    name: req.body.name,
    price: req.body.price
  });

  product.save(function(err, result) {
    if (err)
    {
      return res.status(500).json({reason: err});
    }

    res.status(201).json({
      poruka: 'Proizvod je uspesno kreiran',
      kreiraniProizvod: result
    });
  });
});

// Individualni

router.get('/:productId', function(req, res, next) {
  const productId = req.params.productId;

  Product.findById(productId, function(err, result)
  {
    // Provera da li je doslo do greske
    if (err)
    {
      return res.status(500).json({reason: err});
    }
  });
});
```

```
// Provera da li je pronadjen objekat
if (result)
{
    return res.status(200).json(result);
}

// Ukoliko nije pronadjen objekat
return res.status(404).json({poruka: "Ne postoji trazeni proizvod"});
});

router.patch('/:productId', function(req, res, next) {
    const productId = req.params.productId;

    const updateOptions = {};
    for (let i = 0; i < req.body.length; ++i)
    {
        let option = req.body[i];
        updateOptions[option.nazivPolja] = option.novaVrednost;
    }

    Product.updateOne(
        {_id: productId},
        { $set: updateOptions },
        function(err, raw)
        {
            if (err)
            {
                return res.status(500).json({poruka: err});
            }

            res.status(200).json({poruka: "Proizvod je uspesno izmenjen"});
        }
    );
});

router.delete('/:productId', function(req, res, next) {
    const productId = req.params.productId;

    Product.deleteOne({_id: productId}, function(err)
    {
        if (err)
        {
            return res.status(500).json({poruka: err});
        }

        res.status(200).json({poruka: "Proizvod je uspesno uklonjen"});
    });
});

module.exports = router;
```

Kod 86: primeri/nodejs3/001/api/models/order.js

```
// U ovom fajlu definisemo model koji ce imati nase porudzbine
const mongoose = require('mongoose');

const orderSchema = mongoose.Schema({
    _id: mongoose.Schema.Types.ObjectId,
    product: {
        type: mongoose.Schema.Types.ObjectId,
        // ref je naziv modela sa kojim se povezuje ovaj model
        ref: "Product",
        required: true
    },
    quantity: {
        type: Number,
        default: 1
    }
});

module.exports = mongoose.model("Order", orderSchema);
```

Kod 87: primeri/nodejs3/001/api/routes/orders.js

```
const express = require('express');
const mongoose = require('mongoose');

const Order = require('../models/order');
const Product = require('../models/product');

const router = express.Router();

router.get('/', function(req, res, next) {
    Order.find({}, {product : 1}, function(err, result)
    {
        if (err)
        {
            return res.status(500).json({poruka: err});
        }

        res.status(200).json({
            count: result.length,
            orders: result
        });
    });
});

router.post('/', function(req, res, next) {
    const order = new Order({
        _id: new mongoose.Types.ObjectId(),
        quantity: req.body.quantity,
        product: req.body.productId
    });

    order.save(function(err, result)
```

```

{
  if (err)
  {
    return res.status(500).json({poruka: err});
  }

  res.status(201).json(result);
});
});

router.get('/:orderID', function(req, res, next) {
  const orderID = req.params.orderID;

  Order.findById(orderID, function(err, result)
  {
    if (err)
    {
      return res.status(500).json({poruka: err});
    }

    if (result)
    {
      return res.status(200).json(result);
    }

    res.status(404).json({poruka: "Ne postoji trazena narudzbina"});
  }).populate('product');
});

router.delete('/:orderID', function(req, res, next) {
  const orderID = req.params.orderID;

  Order.deleteOne({_id: orderID}, function(err)
  {
    if (err)
    {
      return res.status(500).json({poruka: err});
    }

    res.status(200).json({poruka: "Narudzbina je uspesno uklonjena"});
  });
});

module.exports = router;

```

Na klijentu se (konceptualno) nije mnogo toga promenilo. Izgled aplikacije je nešto ažuriran, dodata su polja za izmenu podataka pri PATCH zahtevu za proizvode i ažurirali smo validaciju da obuhvati i nova polja formulara, kao i slanje zahteva da ispravno prikaže odgovor od servera. Takođe, dodati su formulari za testiranje zahteva za proizvode, sa odgovarajućim validacijama i slanjem asinhronih zahteva. Ceo izvorni kod implementacije na klijentu dat je u nastavku:

Kod 88: primeri/nodejs3/001/client.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Testiranje rada servera sa proizvodima</title>

    <link rel="stylesheet" type="text/css" href="bootstrap.css">

    <style>
        .proizvod {
            border: 2px solid lightskyblue;
            display: inline-block;
            vertical-align: top;
            width: 250px;
            height: 150px;
            margin-right: 10px;
            margin-bottom: 10px;
            padding: 10px;
        }

        .narudzbinia {
            border: 2px solid darkslateblue;
            display: inline-block;
            vertical-align: top;
            width: 350px;
            height: 150px;
            margin-right: 10px;
            margin-bottom: 10px;
            padding: 10px;
        }

        .odabraniProizvod, .odabranaNarudzbinia {
            border-style: dashed;
        }
    </style>
</head>
<body>
    <div class="container-fluid">
        <h1 class="display-1 bg-secondary text-light">Prodavnica</h1>

        <h1 class="display-4 bg-light">Proizvodi</h1>

        <div class="row">
            <div class="col-md-4 mt-3">
                <form id="form1">
                    <div class="form-group">
                        <label for="productId1">Identifikator proizvoda:</label>
                        <input type="text" id="productId1" name="productId1"
                               class="form-control" title="Unesite identifikator
                               proizvoda ili ostavite prazno da dohvivate sve
                               proizvode">
                    </div>
                </form>
            </div>
        </div>
    </div>
</body>
```

```
<input type="submit" value="Dohvati proizvod(e)" class="btn  
    btn-primary">  
</form>  
<div id="greska1"></div>  
</div>  
  
<div class="col-md-8 mt-3">  
    <div id="listaProizvoda"></div>  
</div>  
</div>  
<div class="row">  
    <div class="col-md-4 mt-3">  
        <h3>Dodajte novi proizvod</h3>  
        <form id="form2">  
            <div class="form-group">  
                <label for="name">Naziv proizvoda:</label>  
                <input type="text" id="name" name="name" class="form-control">  
            </div>  
  
            <div class="form-group">  
                <label for="price">Cena proizvoda:</label>  
                <input type="text" id="price" name="price" class="form-control">  
            </div>  
  
            <input type="submit" value="Kreiraj novi proizvod" class="btn  
                btn-primary">  
        </form>  
  
        <div id="greska2"></div>  
    </div>  
    <div class="col-md-8 mt-3">  
        <div id="kreiraniProizvod"></div>  
    </div>  
</div>  
<div class="row">  
    <div class="col-md-4 mt-3">  
        <h3>Izmenite postojeći proizvod</h3>  
        <form id="form3">  
            <div class="form-group">  
                <label for="productId3">Identifikator proizvoda:</label>  
                <input type="text" id="productId3" name="productId3"  
                    class="form-control">  
            </div>  
  
            <div class="form-group">  
                <label for="name2">Naziv proizvoda:</label>  
                <input type="text" id="name2" name="name" class="form-control">  
            </div>
```

```
<div class="form-group">
    <label for="price2">Cena proizvoda:</label>
    <input type="text" id="price2" name="price" class="form-control">
</div>

    <input type="submit" value="Posalji podatke" class="btn btn-primary">
</form>
    <div id="greska3"></div>
</div>
<div class="col-md-4 mt-3">
    <h3>Obrišite postojeći proizvod</h3>
    <form id="form4">
        <div class="form-group">
            <label for="productId4">Identifikator proizvoda:</label>
            <input type="text" id="productId4" name="productId4" class="form-control">
        </div>

        <input type="submit" value="Posalji podatke" class="btn btn-primary">
    </form>
    <div id="greska4"></div>
</div>
<div class="col-md-4"></div>
</div>

<h1 class="display-4 bg-light mt-3">Narudžbine</h1>

<div class="row">
    <div class="col-md-4 mt-3">
        <form id="form5">
            <div class="form-group">
                <label for="orderId1">Identifikator narudžbine:</label>
                <input type="text" id="orderId1" name="orderId1" class="form-control" title="Unesite identifikator proizvoda ili ostavite prazno da dohvate sve proizvode">
            </div>

            <input type="submit" value="Dohvati narudžbinu/e" class="btn btn-primary">
        </form>
        <div id="greska5"></div>
    </div>
    <div class="col-md-8 mt-3">
        <div id="listaNarudzbina"></div>
    </div>
</div>

<div class="row">
    <div class="col-md-4 mt-3">
        <h3>Naručite proizvod</h3>
```

```

<form id="form6">
    <div class="form-group">
        <label for="productId5">Identifikator proizvoda:</label>
        <input type="text" id="productId5" name="productId5"
               class="form-control">
    </div>

    <div class="form-group">
        <label for="quantity">Kolicina:</label>
        <input type="text" id="quantity" name="quantity" class="form-control">
    </div>

    <input type="submit" value="Posalji podatke" class="btn btn-primary">
</form>
<div id="greska6"></div>
</div>
<div class="col-md-8 mt-3">
    <div id="kreiranaNarudzbinा"></div>
</div>
</div>

<div class="row">
    <div class="col-md-4 mt-3">
        <h3>Odustanite od porudžbine</h3>
        <form id="form7">
            <div class="form-group">
                <label for="orderId2">Identifikator narudzbine:</label>
                <input type="text" id="orderId2" name="orderId2" class="form-control">
            </div>

            <input type="submit" value="Posalji podatke" class="btn btn-primary">
        </form>
        <div id="greska7"></div>
    </div>
    <div class="col-md-8 mt-3"></div>
</div>
</div>
<script type="text/javascript" src="jquery-3.3.1.min.js"></script>
<script type="text/javascript" src="client_products.js"></script>
<script type="text/javascript" src="client_orders.js"></script>
</body>
</html>

```

Kod 89: primeri/nodejs3/001/client\_products.js

```

$(document).ready(function() {
    $("#form1").submit(function()
    {
        // Element u kojem ce biti ispisani bilo svi postojeći proizvodi,
        // bilo samo onaj ciji je identifikator prosledjen
    })
});

```

```
let listaProizvoda = $("#listaProizvoda");
// Element u kojem ce biti ispisana greska
let greska1 = $("#greska1");

let productId = $("#productId1").val();

if (productId === "") {
    $.ajax("http://localhost:3000/products", {
        method: "GET",
        success: function(data) {
            if(data.hasOwnProperty("rezultat")) {
                let sviProizvodi = data.rezultat;

                if (sviProizvodi.length === 0)
                {
                    listaProizvoda.html("<p>Nema proizvoda!</p>");
                    return;
                }

                for (let i = 0; i < sviProizvodi.length; ++i)
                {
                    let proizvod = sviProizvodi[i];
                    let noviProizvod = kreirajNoviProizvod(proizvod,
                        false);
                    listaProizvoda.append(noviProizvod);
                }
            }
        },
        error: function() {
            greska1.text("Greska pri dohvatanju liste proizvoda!").css
                ({color: "red"});
        }
    });
}
else
{
    $.ajax(`http://localhost:3000/products/${productId}`, {
        method: "GET",
        success: function(data) {
            let noviProizvod = kreirajNoviProizvod(data, true);
            listaProizvoda.append(noviProizvod);
        },
        error: function() {
            greska1.text("Greska pri dohvatanju trazenog proizvoda!").
                css({color: "red"});
        }
    });
}

// Cistimo elemente od rezultata poziva od prethodnog puta
listaProizvoda.html("");
greska1.text("");
```

```
        return false;
    });

    $("#form2").submit(function()
{
    // Element u kojem ce biti prikazan kreirani proizvod
    let kreiraniProizvod = $("#kreiraniProizvod");
    let greska2 = $("#greska2");

    let name = $("#name").val();
    let price = Number.parseFloat($("#price").val());

    if (name === "")
    {
        greska2.text("Niste prosledili naziv proizvoda").css({color: "red"});
        return false;
    }

    if (Number.isNaN(price))
    {
        greska2.text("Niste prosledili ispravnu cenu proizvoda").css({
            color: "red"});
        return false;
    }

    $.ajax("http://localhost:3000/products", {
        method: "POST",
        data: JSON.stringify({
            name, price
        }),
        contentType: 'application/json',
        success: function(data) {
            if(data.hasOwnProperty("kreiraniProizvod")) {
                let noviProizvod = kreirajNoviProizvod(data.
                    kreiraniProizvod, true);
                kreiraniProizvod.append(noviProizvod);
            }
        },
        error: function() {
            greska2.text("Greska pri postavljanju proizvoda!").css({color:
                "red"});
        }
    });

    kreiraniProizvod.html("");
    greska2.text("");

    return false;
});

$("#form3").submit(function()
```

```
{  
    let greska3 = $("#greska3");  
  
    let productId = $("#productId3").val();  
  
    if (productId === "")  
    {  
        greska3.text("Niste prosledili identifikator proizvoda").css({  
            color: "red"});  
        return false;  
    }  
  
    let name = $("#name2").val();  
    let price = $("#price2").val();  
    let updateOptions = [];  
  
    if (name !== "")  
    {  
        updateOptions.push({  
            nazivPolja: "name",  
            novaVrednost: name  
        });  
    }  
  
    if (price !== "")  
    {  
        updateOptions.push({  
            nazivPolja: "price",  
            novaVrednost: price  
        });  
    }  
  
    $.ajax(`http://localhost:3000/products/${productId}` , {  
        method: "PATCH",  
        data: JSON.stringify(updateOptions),  
        contentType: 'application/json',  
        success: function(data) {  
            if(data.hasOwnProperty("poruka")) {  
                greska3.text(data.poruka).css({color: "green"});  
            }  
        },  
        error: function() {  
            greska3.text("Greska pri menjaju proizvoda!").css({color: "red"});  
        }  
    });  
  
    greska3.text("");  
  
    return false;  
});  
  
$("#form4").submit(function()
```

```
{  
    let greska4 = $("#greska4");  
  
    let productId = $("#productId4").val();  
  
    if (productId === "")  
    {  
        greska4.text("Niste prosledili identifikator proizvoda").css({  
            color: "red"});  
        return false;  
    }  
  
    $.ajax(`http://localhost:3000/products/${productId}` , {  
        method: "DELETE",  
        success: function(data) {  
            if(data.hasOwnProperty("poruka")) {  
                greska4.text(data.poruka).css({color: "green"});  
            }  
        },  
        error: function() {  
            greska4.text("Greska pri brisanju proizvoda!").css({color: "red"});  
        }  
    });  
  
    greska4.text("");  
    return false;  
});  
});  
  
// Pomocna funkcija za kreiranje novog <div> elementa  
function kreirajNoviProizvod(podaciOProizvodu, odabran)  
{  
    let naziv = podaciOProizvodu.name;  
    let cena = podaciOProizvodu.price;  
    let sku = podaciOProizvodu._id;  
  
    let noviProizvod = $(document.createElement("div"));  
    noviProizvod.addClass("proizvod");  
  
    if (odabran === true)  
    {  
        noviProizvod.addClass("odabraniProizvod");  
    }  
  
    noviProizvod.html(`  
        <h4>${naziv}</h4>  
        <h5>Cena: ${cena}</h5>  
        <p>${sku}</p>  
    `);  
  
    return noviProizvod;  
}
```

## Kod 90: primeri/nodejs3/001/client\_orders.js

```
$(document).ready(function() {

    $("#form5").submit(function()
    {
        let listaNarudzbina = $("#listaNarudzbina");
        let greska5 = $("#greska5");

        let orderId = $("#orderId1").val();

        if (orderId === "")
        {
            $.ajax("http://localhost:3000/orders", {
                method: "GET",
                success: function(data) {
                    let count = data.count;
                    let sveNarudzbine = data.orders;

                    if (count === 0)
                    {
                        listaNarudzbina.html("<p>Nema narudzbina!</p>");
                        return;
                    }

                    for (let i = 0; i < sveNarudzbine.length; ++i)
                    {
                        let tekucaNarudzbina = sveNarudzbine[i];
                        let novaNarudzbina =
                            kreirajNovuNarudzbinuBezPodatakaOProizvodu(
                                tekucaNarudzbina, false);
                        listaNarudzbina.append(novaNarudzbina);
                    }
                },
                error: function() {
                    greska5.text("Greska pri dohvatanju svih narudzbina!").css
                        ({color: "red"});
                }
            });
        }
        else
        {
            $.ajax(`http://localhost:3000/orders/${orderId}`, {
                method: "GET",
                success: function(data) {
                    let novaNarudzbina =
                        kreirajNovuNarudzbinuSaPodacimaOProizvodu(data, true);
                    listaNarudzbina.append(novaNarudzbina);
                },
                error: function(jqXHR) {
                    // Objekat jqXHR ima svojstvo status kojim dobijamo
                    // statusni kod od servera
                    if (jqXHR.status === 404)
```

```
{  
    // Objekat jqXHR ima svojstvo responseText kojim  
    // dobijamo sam odgovor od servera.  
    // Posto mi znamo da se odgovor salje kao JSON niska,  
    // onda mozemo da konvertujemo tu nisku u JavaScript  
    // objekat  
    // tako sto pozovemo funkciju JSON.parse, kojoj se  
    // prosledjuje JSON niska.  
    // Dakle, JSON.parse je inverzna funkcija funkcije JSON.  
    // stringify (vazi i obrnuto).  
    let odgovorOdServera = JSON.parse(jqXHR.responseText);  
    greska5.text(odgovorOdServera.poruka).css({color: "red"  
        "});  
    }  
    else  
    {  
        greska5.text("Greska pri dohvatanju narudzbine!").css({  
            color: "red"});  
    }  
}  
});  
  
listaNarudzbina.html("");  
greska5.text("");  
  
return false;  
});  
  
$("#form6").submit(function()  
{  
    let kreiranaNarudzbina = $("#kreiranaNarudzbina");  
    let greska6 = $("#greska6");  
  
    let productId = $("#productId5").val();  
    let quantity = Number.parseInt($("#quantity").val());  
  
    if (productId === "")  
    {  
        greska6.text("Niste prosledili identifikator proizvoda").css({  
            color: "red"});  
        return false;  
    }  
  
    if (Number.isNaN(price))  
    {  
        greska6.text("Niste prosledili ispravnu kolicinu proizvoda").css({  
            color: "red"});  
        return false;  
    }  
  
    $.ajax("http://localhost:3000/orders", {  
        method: "POST",  
    }
```

```
        data: JSON.stringify({
            productId, quantity
        }),
        contentType: 'application/json',
        success: function(data) {
            let novaNarudzbina =
                kreirajNovuNarudzbinuBezPodatakaOProizvodu(data, true);
            kreiranaNarudzbina.append(novaNarudzbina);
        },
        error: function() {
            greska6.text("Greska pri postavljanju narudzbine!").css({color
                : "red"});
        }
    });

    kreiranaNarudzbina.html('');
    greska6.text('');

    return false;
});

$("#form7").submit(function()
{
    let greska7 = $("#greska7");

    let orderId = $("#orderId2").val();

    if (orderId === "")
    {
        greska7.text("Niste prosledili identifikator narudzbine").css({
            color: "red"});
        return false;
    }

    $.ajax(`http://localhost:3000/orders/${orderId}` , {
        method: "DELETE",
        success: function(data) {
            if(data.hasOwnProperty("poruka")) {
                greska7.text(data.poruka).css({color: "green"});
            }
        },
        error: function() {
            greska7.text("Greska pri brisanju proizvoda!").css({color: "red"});
        }
    });
    greska7.text("");

    return false;
});
});
```

```
// Pomocne funkcije za kreiranje novog <div> elementa

function kreirajNovuNarudzbinuBezPodatakaOProizvodu(podaciONarudzbini,
    odabrana)
{
    let orderID = podaciONarudzbini._id;
    let productID = podaciONarudzbini.product;

    let novaNarudzbinu = $(document.createElement("div"));
    novaNarudzbinu.addClass("narudzbinu");

    if (odabrana === true)
    {
        novaNarudzbinu.addClass("odabranaNarudzbinu");
    }

    novaNarudzbinu.html(` 
        <h4>ID proizvoda: ${productID}</h4>
        <p>${orderID}</p>
    `);

    return novaNarudzbinu;
}

function kreirajNovuNarudzbinuSaPodacimaOProizvodu(podaciONarudzbini,
    odabrana)
{
    let orderID = podaciONarudzbini._id;
    let kolicina = podaciONarudzbini.quantity;
    let proizvod = podaciONarudzbini.product;

    let novaNarudzbinu = $(document.createElement("div"));
    novaNarudzbinu.addClass("narudzbinu");

    if (odabrana === true)
    {
        novaNarudzbinu.addClass("odabranaNarudzbinu");
    }

    novaNarudzbinu.html(` 
        <h5>Proizvod: ${proizvod.name}</h5>
        <h5>Cena: ${proizvod.price}</h5>
        <p>Količina: ${kolicina}</p>
        <p>${orderID}</p>
    `);

    return novaNarudzbinu;
}
```

**Zadatak 11.1:**

Jedan problem koji prethodna implementacija ima jeste što možemo da proslidimo zahtevu za kreiranje nove porudžbine identifikator proizvoda koji ne postoji u bazi. Time dobijamo invalidnu situaciju ukoliko pokušamo da dohvati informacije o proizvodu koji ne postoji. Prepraviti implementaciju tako da se pri unosu nove porudžbine prvo proverava da li postoji proizvod sa zadatim identifikatorom.

**Rešenje:** Ono što je dovoljno uraditi jeste u datoteci `./api/routes/orders.js` prvo uvesti `Product` model:

```
const Product = require('../models/product');
```

a zatim ispraviti implementaciju za rutiranje POST zahteva nad putanjom `/orders` na sledeći način:

```
router.post('/', function(req, res, next) {
  Product.findById(req.body.productId, function (err, result) {
    if (err)
    {
      return res.status(500).json({poruka: err});
    }

    if (result === null)
    {
      return res.status(404).json({poruka: "Ne mozemo kreirati
          porudzbinu za nepostojeci proizvod"});
    }

    const order = new Order({
      _id: new mongoose.Types.ObjectId(),
      quantity: req.body.quantity,
      product: req.body.productId
    });

    order.save(function(err, result)
    {
      if (err)
      {
        return res.status(500).json({poruka: err});
      }

      res.status(201).json(result);
    });
  });
});
```



## Dodatak

### Lista kodova

1	primeri/uvod_u_html/osnovna_struktura.html . . . . .	7
2	zadaci/001/strana.html . . . . .	12
3	zadaci/002/strana.html . . . . .	18
4	zadaci/002/stil.css . . . . .	19
5	primeri/model_kutije/box_model_inline.html . . . . .	21
6	primeri/model_kutije/block_sirina_visina.html . . . . .	21
7	primeri/model_kutije/block_padding.html . . . . .	22
8	primeri/model_kutije/border_radius.html . . . . .	25
9	primeri/model_kutije/block_margin.html . . . . .	26
10	primeri/model_kutije/margin_auto.html . . . . .	27
11	primeri/model_kutije/inline_margin.html . . . . .	28
12	zadaci/003/strana.html . . . . .	30
13	zadaci/003/stil.css . . . . .	31
14	primeri/pozicioniranje/inline-block.html . . . . .	39
15	zadaci/004/relativno.html . . . . .	41
16	zadaci/004/relativno.css . . . . .	41
17	zadaci/004/apsolutno.css . . . . .	42
18	primeri/float/clear.html . . . . .	44
19	zadaci/004/float.css . . . . .	46
20	primeri/float/parent.html . . . . .	47
21	primeri/pozadina/background_image.html . . . . .	49
22	primeri/pozadina/background_repeat-default.html . . . . .	50
23	primeri/pozadina/background_repeat-repeatax.html . . . . .	51
24	primeri/pozadina/background_position.html . . . . .	52
25	primeri/pozadina/background_size.html . . . . .	53
26	primeri/pozadina/linear_gradient.html . . . . .	55
27	primeri/pozadina/radial_gradient.html . . . . .	56
28	primeri/rad_sa_slikama/img.html . . . . .	58
29	primeri/rad_sa_slikama/img_override.html . . . . .	59
30	primeri/rad_sa_slikama/img_override.css . . . . .	59
31	primeri/rad_sa_slikama/figure.html . . . . .	60
32	primeri/tabele/table.html . . . . .	62
33	primeri/tabele/table_collapsed.html . . . . .	63
34	primeri/tabele/table_misc.html . . . . .	65
35	primeri/tabele/table_span.html . . . . .	66
36	primeri/tabele/table_stilovi.html . . . . .	67
37	primeri/liste/nenumerisana.html . . . . .	70
38	primeri/liste/numerisana.html . . . . .	71
39	primeri/liste/pozicioniranje.html . . . . .	72
40	primeri/liste/nth-child.html . . . . .	73
41	primeri/liste/ugnjezdene.html . . . . .	74
42	primeri/liste/definiciona.html . . . . .	75
43	primeri/veze/veze.html . . . . .	78
44	primeri/veze/veze.css . . . . .	78
45	primeri/pseudoklasa/pseudoklase.html . . . . .	80
46	primeri/responsive/viewport.html . . . . .	83
47	primeri/responsive/pozicioniranje.html . . . . .	85
48	primeri/responsive/min_max_width.html . . . . .	87

49	primeri/responsive/small_devices.css . . . . .	88
50	zadaci/005/media.html . . . . .	88
51	primeri/bootstrap/posicioniranje.html . . . . .	90
52	primeri/bootstrap/margine.html . . . . .	91
53	primeri/javascript1/testiranje-alert.html . . . . .	100
54	primeri/javascript1/testiranje-alert-2.html . . . . .	100
55	primeri/javascript1/testiranje-alert-2.js . . . . .	101
56	primeri/javascript1/01_js_selektori.html . . . . .	103
57	primeri/javascript1/02_stilovi.html . . . . .	104
58	primeri/javascript1/03_sadrzaj.html . . . . .	105
59	primeri/javascript1/04_kreiranje_elemenata.html . . . . .	106
60	primeri/javascript1/05_console.html . . . . .	108
61	primeri/javascript1/06_pridruzivanje_osluskivaca.html . . . . .	109
62	primeri/javascript2/01_formulari.html . . . . .	114
63	primeri/javascript2/01_formulari_obrađa.js . . . . .	123
64	primeri/javascript3/kraj_semestra.html . . . . .	144
65	primeri/javascript3/help_box.html . . . . .	146
66	primeri/javascript3/galerija_jquery.html . . . . .	148
67	primeri/javascript3/provera_formulara.html . . . . .	154
68	primeri/javascript3/provera_formulara.js . . . . .	157
69	primeri/nodejs1/001/server.js . . . . .	164
70	primeri/nodejs1/002/server.js . . . . .	168
71	primeri/nodejs1/002/skalarni_proizvod.js . . . . .	169
72	primeri/nodejs1/002/skalarni_proizvod.html . . . . .	172
73	zadaci/006/server.js . . . . .	175
74	zadaci/006/captcha.html . . . . .	177
75	primeri/nodejs2/001/nodemon.json . . . . .	186
76	primeri/nodejs2/001/server.js . . . . .	186
77	primeri/nodejs2/001/app.js . . . . .	186
78	primeri/nodejs2/001/api/routes/products.js . . . . .	187
79	primeri/nodejs2/001/klijent.html . . . . .	189
80	primeri/nodejs2/001/klijent.js . . . . .	190
81	zadaci/007/api/routes/orders.js . . . . .	194
82	primeri/nodejs3/001/server.js . . . . .	211
83	primeri/nodejs3/001/app.js . . . . .	211
84	primeri/nodejs3/001/api/models/product.js . . . . .	212
85	primeri/nodejs3/001/api/routes/products.js . . . . .	212
86	primeri/nodejs3/001/api/models/order.js . . . . .	215
87	primeri/nodejs3/001/api/routes/orders.js . . . . .	215
88	primeri/nodejs3/001/client.html . . . . .	216
89	primeri/nodejs3/001/client_products.js . . . . .	220
90	primeri/nodejs3/001/client_orders.js . . . . .	225