UNIVERZITET U BEOGRADU
ELEKTROTEHNIČKI FAKULTET
KATEDRA ZA ELEKTRONIKU

Primena TCP/IP tehnologija u namenskim sistemima

2. JavaScript

Školska 2021/22

prof. dr Milan Ponjavić

Haris Turkmanović

Nikola Cvetković

1. Uvod

Na prethodnoj laboratorijskoj vežbi smo se upoznali sa načinom struktuiranja teksta koji pomaže nama, kao dizajnerima stranice, da razumemo sam kôd, a sa druge strane, govori *Web Browser*-u kako da interpretira našu stranicu. Nakon što smo se upoznali sa struktuiranjem teksta i koda, naučili smo kako da elementima na našoj HTML stranici dodamo stilske karakteristike (veličinu slova, boju slova, razmak između redova, *layout* stranice, itd.). Time smo obuhvatili sve ono što je neophodno da bi izvršili statički prikaz sadržaja naše stranice na internetu.

Ovakav pristup je potreban ali ne i dovoljan pri kreiranju nekih stranica. Mnoge stranice podržavaju interakciju sa korisnikom što povećava njihovu dinamiku. Na primer, neki sajtovi za gledanje filmova omogućavaju promenu pozadine usled boljeg fokusiranja na video sadržaj koji gledamo. Na *e-student* nalozima postoje određene ankete koje se popunjavaju pre prijave predmeta. Postoje razni "obrasci" koji se popunjavaju prilikom registrovanja na forume, dodavanja na mail listu, itd.

Zadatak ove laboratorijske vežbe jeste da studente upozna sa osnovama rada u *JavaScript* jeziku koji omogućava dodavanje dinamike na Web stranicu. Dakle, nakon što se kreira struktura stranice primenom HTMLa i dodaju stilovi primenom CSSa moguće je primenom *JavaScript* jezika napraviti dinamiku na našoj stranici. *JavaScript* omogućava promenu sadržaja stranice i stilova na stranici nakon učitavanja stranice što ovaj jezik čini veoma pogodnim u kreiranju različitih dinamika stranice.

Najvažnija funkcionalnost koja je implementirana na ovom jeziku jesu API (*Application Programming Interfaces*) funkcije. Zahvaljujući njima, implementiranje dinamike stranice postaje znatno lakše, intuitivnije i zanimljivije. Postoje dva tipa API funkcija: *Web Browser API* i *Ostale API funkcije*.

Web Browser API funkcije su ugrađene u *Web Browser* i omogućavaju pristup delovima radnog okruženja *Web Browsera*. Postoje različiti podtipovi ovog tipa API funkcija ali za nas je bitan jedan podtip API funkcija, a to su DOM (*Document Object Model*) API funkcije.

DOM omogućava preuređivanje HTML-a i CSS-a (kreiranje, uklanjanje delova koda, promena postojećih delova) u cilju pravljenja dinamike na *Web* Stranici. Da bismo bolje razumeli princip na osnovu koga rade ove funkcije kao i funkcionalnosti koje implementiraju, neophodno je uraditi primere date u nastavku. Nakon razumevanja principa na kome rade ove API funkcije, korišćenje *JavaScript* koda u cilju izmene sadržaja web stranice postaje znatno jednostavnije i zanimljivije.

2. JavaScript

JavaScript je skripting jezik što znači da se sastoji od niza komandi koje se ne prevode od strane komapjlera u mašinski kod, već ih interpreter (*Web Browser*) direktno tumači i sprovodi

odgovarajuću akciju u zavisnosti od navedene komande. Dakle, ove skripte se izvršavaju na strani korisnika (klijentskoj strani). Iz ovoga je jasno da se ovaj jezik direktno oslanja na karakteristike *Web Browser*-a. Kada pišemo program u ovom jeziku nije neophodno navoditi tipove podataka promenljivih (*loosly typed*). Ovaj jezik **razlikuje** velika i mala slova (*case sensitive*)

Kao što je u uvodu rečeno, ovaj programski jezik omogućava dinamiku web stranice i interakciju sa korisnikom te stranice. Neke od dodatnih mogućnosti koje pruža *JavaScript* su:

- Različita računanja i skladištenje rezultata u promenljive
- Manipulacija/upravljanje različitim delovima HTML koda
- Sprovođene akcije usled pojave nekih događaja kao što su klik miša, pritisak tastera na tastaturi, promene veličine prozora *Web Browser*-a, itd.

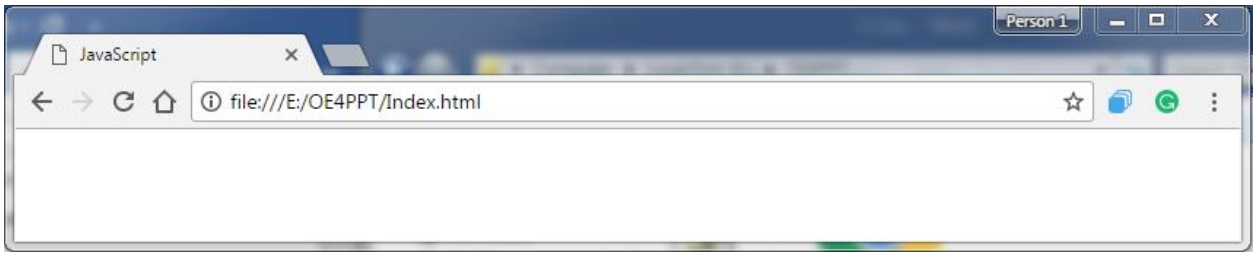
Primeri koji slede imaju za cilj da demonstriraju neke od funkcionalnosti koje pruža *JavaScript.*

2.1. Promena sadržaja klikom na dugme

Napraviti HTML stranicu index.html koja implementira različite funkcionalnosti klikom na element Button. Prilikom učitavanja stranice naslov stranice je "JavaScript". Prvi klik na dugme dovodi do promene naslova stranice u "Dodavanje paragrafa". Svaki naredni klik na dugme dovodi do dodavanja novog teksta u paragraf koji ima sadržaj "Dodat je paragraf ***i****" gde* ***i*** *predstavlja broj paragrafa na stranici. Napisati HTML kôd koji implementira traženu funkcionalnost stranice i potvrditi uspešnost napisanog koda.*

Rešenje:

Najpre je neophodno otvoriti radni direktorijum ***OE4PPT*** u koji ćemo smeštati sve fajlove neophodne za ispravno funkcionisanje naše Web stranice. Nakon toga kreiramo *index.html* fajl i nakon kreiranja standardne HTML strukture koda, koja sadrži *body* i *head* elemente, potrebno je dodati naslov stranice korišćenjem *title* elementa. Nakon kreiranja ovakve strukture i otvaranja stranice u *Web Browser*-u, dobijamo izgled stranice kao na slici 1.1

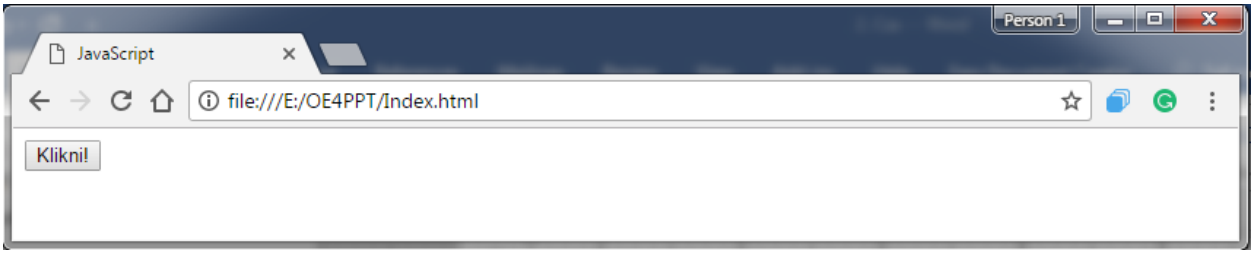


1.1 – Izgled stranice nakon kreiranja

Button je HTML element koji se na stranici vizuelno predstavlja kao deo teksta koji menja pozadinu usled klika na tekst. Nakon dodavanja ovog elementa naš kod je:

```
<!DOCTYPE html>
<html>
<head>
    <title>JavaScript</title>
</head>
<body>
    <button> Klikni! </button>
</body>
</html>
```

Nakon otvaranja stranice stranice u Web Browser-u dobijamo izgled stranice kao na slici 1.2



1.2 - Izgled stranice nakon dodavanja button elementa

Ako sada kursor miša dovedemo na dugme "Klikni!", i kliknemo na njega, videćemo da se na stranici ništa ne dešava. Dakle, sada je našoj stranici potrebno dodati traženu funkcionalnost. Ovde se završavaju mogućnosti HTML-a i sada glavnu ulogu preuzima **JavaScript (JS).**

JS kod se piše u nekom od tekstualnih editora. Pošto smo do sada koristili *Notepad++*, sada u ovom programu treba otvoriti novi fajl i sačivati ga kao *code.js*. Ekstenzija *.js* služi kao indikator *Web Browser*-u da se tu nalazi kod pisan u JavaScriptu. Nakon što smo sačuvali ovaj fajl u radni direktorijum *OE4PPT* neophodno je izvršiti povezivanje naše JS skripte sa HTML stranicom. To se postiže dodavanjem sledeće linije koda u *head* element HTML stranice:

```
    <script src="code.js" async></script>
```

Nakon dodavanja ove linije koda, naša HTML stranica i odgovarajuća JS skripta su povezane. Potrebno je napomenuti da ovo nije jedini način dodavanja JS koda u HTML stranicu. Postoji mogućnost direktnog pisanja koda unutar *script* elementa koji se može pozicionirati bilo gde na stranici ali je poželjno da bude u *head* elementu.

(Čemu služi atribut async? Pogledati snimak druge laboratorijske vežbe)

Nakon povezivanja JS skripte i HTML stranice, naš zadatak je da nađemo način da detektujemo pritisak levog tastera miša kada se kursor nalazi pozicioniran na elementu *Button* naše HTML stranice. Ovde je neophodno napraviti malu digresiju i objasniti kako zapravo funkcioniše *Web Browser*.

Na slici 1.3 je prikazan izgled jednog *Web Browser*-a (Google Chrome) sa naznačenim delovima. On se sastoji od:

1.3 - Struktura Web Browsera

- ***Window*** dela koji predstavlja deo radnog okruženja *Web Browser*-a u koji se učitava Web stranica. Za pristup ovom delu *Web Browser*-a iz *JavaScript* koda koristi se objekat *window*. Metode (funkcije) ovog objekta služe za editovanje svojstava koje su karakteristične za ovaj objekat (na primer promena veličine prozora *Web Browser*-a)
- ***Document*** deo predstavlja stranicu koja se učitava u ***Window*** deo. Za pristup delovima koda stranice iz JavaScript koda, koristi se objekat *document* koji ima svoje metode za pristup elementima stranice. Ovaj objekat se koristi da bi bilo moguće referenciranje odredjenih HTML i CSS struktura unutar učitane stranice. Pojedinačne metode ovog objekta biće objašnjavane kroz nastavak kursa.

Za interpretiranje svake učitane stranice koristi se model pod nazivom DOM (*Document Object Model*). Ovaj model podrazumeva da će svaka učitana stranica od strane *Web Browser*-a biti interpretirana kao jedna struktura koja se može modelirati *stablom* (eng. *tree structure*). Ovakav pristup omogućava da se prilikom programiranja u JS-u pristupi pojedinim strukturama stranice. U našem slučaju struktura stranice je sledeća:

- **html**
  - head
    - title
    - script
  - body
    - button

Osnovni gradivni element ovakve strukture naziva se čvor (eng. *node*). On predstavlja ulaz (eng. *entry point*) u samu strukturu. Svako stablo ima izvorišni čvor (eng. *root node*) koji predstavlja vrh hijerarhijske strukture prikazane na slici (*html*). Pored toga, jedan čvor može imati izvedeni čvor strukture (eng. *child node*). Na primer *title* element je izveden iz *head* elementa. Element koji unutar sebe ima drugi element predstavlja glavni čvor (eng *parent node*) za taj element. Na primer *body* element predstavlja glavni čvor za element *button*.

Svakom od ovih čvorova unutar HTML dokumenta se može pristupiti pomoću metoda objekta *document*. Metode koje se koriste u tu svrhu su:

- *querySelector("SelectorName");*

  Vraća referencu na prvi element u HTML kodu kome pristupamo pomoću CSS selektora *SelectorName* iz CSS fajla. Ako želimo da pristupimo svim elementima u fajlu na koje se odnosi isti CSS selektor to se postiže pomoću funkcije *querySelectorAll("SelectorName");* .

- *getElementById("IdValue");*

  Vraća referencu na element na osnivu vrednosti atributa *Id*

- *getElementsByTagName("ElementTagName")*

  Vraća niz referenci na elemente koji imaju isti naziv elementa. Na primer *getElementsByTagName("p")* vraća niz reference na sve elemente *p*

Iz priloženog možemo videti da postoje različite metode objekta *document* za referenciranje nekog elementa unutar HTML koda. Najrasprostranjenija metoda je *querySelector("SlectorName")*, pa ćemo nju koristiti kroz ostatak koda.

Sada se vraćamo na rešavanje našeg konkretnog problema. Neophodno je referencirati element *button* unutar CSS koda. To se postiže dodavanjem sledeće linije koda u naš *code.js* fajl:

```
var Button = document.querySelector('button');
```

Ispred naziva promenljive neophodno je (*or is it?*) navesti rezervisanu reč *var*. Button je naziv promenljive u koju smeštamo referencu na element *button*. *document.querySelector('button')* služi za „dohvatanje" reference na element *button* iz HTML koda učitane stranice. Nakon izvršavanja ove linije koda, promenljiva *Button* predstavlja objekat koji sadrži referencu na *button* element i sada su sva svojstva ovog elementa dostupna preko promenljive *Button.*

Nakon što smo iz HTML koda izdvojili referencu na element *button,* neophodno je detektovati klik miša na *button* element i shodno tome sprovesti odgovarajuću akciju. Kada se detektuje klik tastera miša, poziva se odgovarajući *onclik* događaj kome prethodno treba dodeliti

funkcionalnost da bi se sprovela određena akcija. Dodela funkcionalnosti ovom događaju vrši se pomoću sledećeg koda:

```
Button.onclick = function(){
    document.title="Dodavanje paragrafa";
};
```

gde *title* predstavlja osobinu objekta *document* koja se može čitati i koja se može menjati, a odnosi se na naslov stranice. Dosadašnji JS kod je:

```
var Button = document.querySelector('button');

Button.onclick = function(){
    document.title="Dodavanje paragrafa";
};
```

Sada je neophodno sačuvati naš kôd, učitati stranicu i kliknuti na dugme „Klikni!“ nakon čega se menja naslov naše stranice. Ovim smo ispunili deo zadatka, a sada je neophodno nastaviti dalje sa dodavanjem paragrafa.

Postavkom zadatka je rečeno da se prvim klikom na element *button* menja naslov stranice dok se preostalim klikovima na element *button* dodaju novi paragrafi. Zbog toga je potrebno detektovati prvi klik i preostale klikove. Postoje mnogi načini da se sprovede ovaj algoritam, ali mi ćemo ovaj algoritam sprovesti na sledeći način:

Uvodimo pomoćnu promenljivu *TitleChanged* koja nosi informaciju o tome da li je promenjen naslov. Ukoliko je naslov promenjen, vrednost ove promenljive je *true* dok je u surpotnom vrednost ove promenljive *false*. Nakon dodavanja ove promenljive i modifikacije našeg koda tako da se zadrži funkcionalnost, dobijamo sledeći kôd:

```
var Button = document.querySelector('button');
var TitleChanged = false;


Button.onclick = function(){
    if(TitleChanged == false){
        document.title="Dodavanje paragrafa";
        TitleChanged = true;
    }
    else{

    }
};
```

Sada je u *else* grani neophodno kreirati paragraf i dodeliti mu sadržaj. Kreiranje novog elementa (u ovom primeru paragrafa) se postiže pomoću *document* metode *createElelement("ElementName“).* Promena sadržaja kreiranog elementa vrši se promenom vrednosti osobine *textContent*. Linija koda kojom se postiže implementacija ove funkcionalnosti je:

```
var para = document.createElement('p');
para.textContent = "Paragraf"+(Counter++);
```

Promenljiva *Counter* predstavlja broj trenutno kreiranih paragrafa. Nakon dodavanja prethodne dve linije koda u *else* granu unutar naše funkcije i učitavanja stranice, primetićemo da sadržaj još nije vidljiv na našoj stranici. To se dešava zbog toga što naš paragraf još nije dodeljen nijednom elementu unutar fajla, već predstavlja „lebdeći" paragraf koji je nevidljiv. Dodela paragrafa nekom od postojećih elemenata vrši se pomoću metode *appendChild(Element);* metode, kojom dodeljujemo lebdeći paragraf. U našem slučaju, paragraf *p* će biti deo elementa *body*. Pošto u JS kodu nemamo referencu na element *body*, neophodno je prethodno referencirati *body* element koristeći sledeću liniju koda koju dodajemo na početak skripte:

```
var BodyEl = document.querySelector('body');
```

Nakon što smo izdvojili referencu na element *body* i smestili u promenljivu *BodyEl*, sada je neophodno primenom metode *appendChild* dodeliti ovom elementu kreirani paragraf. Nakon toga, sadržaj skripte *code.js* je sledeći:

```
var Button = document.querySelector('button');
var BodyEl = document.querySelector('body');
var TitleChanged = false;
var Counter = 0;

Button.onclick = function(){
    if(TitleChanged == false){
        document.title="Dodavanje paragrafa";
        TitleChanged = true;
    }
    else{
        var para = document.createElement('p');
        para.textContent = "Paragraf"+(Counter++);
        BodyEl.appendChild(para);
    }
};
```

Prikazani kôd realizuje zahtevanu funkcionalnost. Međutim, da bismo mogli jednoznačno referencirati kreirane paragrafe, neophodno je dodati *id* vrednost svakom kreiranom paragrafu. To se postiže pomoću osobine *id* kojoj se dodeljuje vrednost nakon što se kreira neki element. Kôd sa navedenom izmenom prikazan je u nastavku:

```
var Button = document.querySelector('button');
var BodyEl = document.querySelector('body');
var TitleChanged = false;
var Counter = 0;
Button.onclick = function(){
    if(TitleChanged == false){
        document.title="Dodavanje paragrafa";
        TitleChanged = true;
    }
```

```
    else{
        var para = document.createElement('p');
        para.textContent = "Paragraf"+Counter;
        para.id = Counter++;
        BodyEl.appendChild(para);
    }
};
```

(Nasuprot metodi *appendChild* koja se koristi u cilju dodavanja novog čvora u strukturi, postoji metoda *removeChild* koja uklanja čvor iz strukture. Na osnovu ovoga potrebno je dodati još jedan element *button* čija funkcionalnost će biti da se usled klika na njega izbriše poslednji paragraf u nizu.)

2.2. Dinamičko dodavanje elemenata

Potrebno je napraviti web stranicu koja omogućava dodavanje različitih elemenata na stranicu. Mogući elementi su: h1,h2,h3 i p.

Svakom od elemenata korisnik može da definiše:

- *proizvoljan sadržaj koji ne sme biti duži od 25 karaktera.*
- *boju pozadine pomoću # formata zadavanja boja*
- *boju slova pomoću # formata zadavanja boja*
- *da li je element vidljiv na stranici ili ne*

Ukoliko korisnik ispravno definiše zahtevane funkcionalnosti, određeni element se pojavljuje na stranici dok se u suprotnom ispisuju poruke o tome šta nije u redu. Kada se element doda na stranicu, klikom na element se vrši uklanjanje elementa sa stranice. Kreirati stranicu, napisati HTML,CSS i JS kod kojim se postiže zahtevana funkcionalnost i potvrditi uspešnost realizacije.

Rešenje:

Cilj ovog zadatka je da korisnika upozna sa osnovnim HTML elementima koji se koriste za prikupljanje informacija od korisnika kao i osnovnim algoritmima za proveru ispravnosti unetih podataka. Kroz ovaj primer biće uvedene razne funkcije koje su podržane od strane *Web Browser*-a. Takođe, pravićemo i naše funkcije, čiji je zadatak da provere ispravnost unetih podataka i da na osnovu toga sprovedu odgovarajuću akciju.

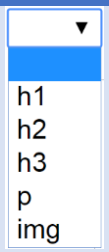
Podaci se mogu unositi u različitim formatima (kao padajuće liste, tekstualni pravougaonici, kao mogućnost selektovanja određenih funkcionalnosti, itd.). HTML ima podršku za unos različitih tipova podataka. Tako *input* predstavlja HTML element koji se koristi za unos podataka. Atribut *type* služi za definisanje različitih interfejsa za unos podataka. Neki od podržanih interfejsa za unos podataka, kao i izgled elementa na stranici, prikazani su u tabeli:

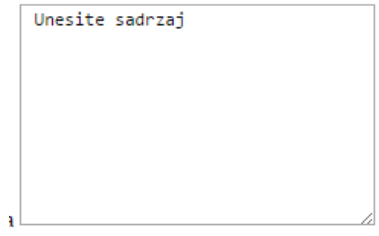
Vrednost atributa type	Kôd	Izgled elementa na stranici
text	<input type=”text”>	
email	<input type=“email“>	
password	<input type=“password“>	
url	<input type=“URL“>	
checkbox	<input type=“checkbox“ checked> <input type=“checkbox“>	 
radio	<input type=“radio“ checked> <input type=“radio“>	 

Vrednost *email* se razlikuje od *text* vrednosti po tome što *Web Browser* ima implementirane metode provere validnosti unetog sadržaja. Takođe, *password* omogućava prikrivanje unetog teksta, dok *url* podržava unos URL-a neke stranice. Svaki *input* element ima atribut *value* koji sadrži unetu vrednost u *input* element.

Pored *input* elementa, za unos za prikupljanje informacija od korisnika, koriste se i sledeći elementi:

Naziv elementa	Kod	Izgled elementa na stranici

select	`<select>`
	`  <option></option>`
	`  <option>h1</option>`
	`  <option>h2</option>`
	`  <option>h3</option>`
	`  <option>p</option>`
	`  <option>img</option>`
	`</select>`
		

textarea	`<textarea cols="30" rows="10">`
	`  Unesite sadrzaj`
	`</textarea>`
		

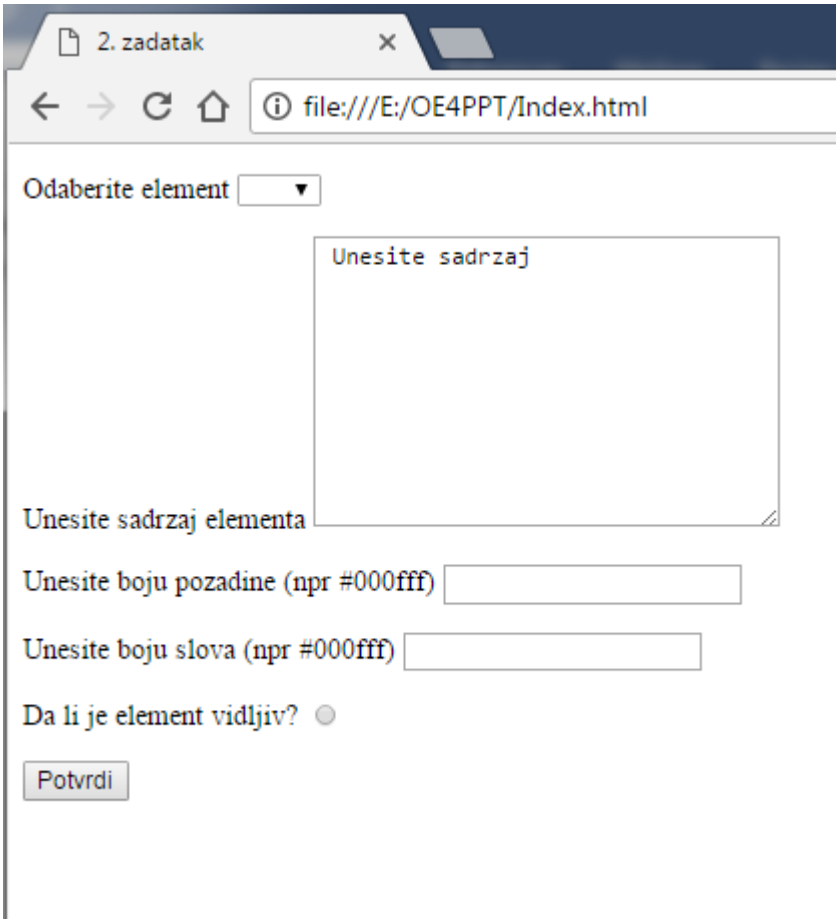
(Do sada smo naveli neke od elemenata koje ćemo koristiti u ovom primeru. Postoji još mnogo elemenata čiji se opis i način upotrebe može naći na internetu.)

Potrebno je svakom od elemenata dodeliti jedinstvenu vrednost atributa *id* kako bi bilo moguće pristupati HTML elementima iz JS koda.

Na osnovu sadržaja navedenih u tabelama potrebno je kreirati odgovarajuće elemente na našoj stranici. Nakon dodavanja elemenata na stranicu dobijamo sledeći HTML kôd:

```
<!DOCTYPE html>
<html>
<head>
    <title>2. zadatak</title>
    <script src="code.js" async></script>
</head>
<body>
    <div class="Obrazac">
        <p> Odaberite element
            <select id="Element">
                <option></option>
                <option>h1</option>
                <option>h2</option>
                <option>h3</option>
                <option>p</option>
            </select>
        </p>
        <p> Unesite sadrzaj elementa
            <textarea id="text" cols="30" rows="10"> Unesite
sadrzaj</textarea>
        </p>
        <p> Unesite boju pozadine (npr #000fff)
            <input type="text" id="BojaPozadine">
        </p>
        <p> Unesite boju slova (npr #000fff)
            <input type="text" id="BojaSlova">
        </p>
        <p> Da li je element vidljiv?
            <input type="radio"  id="Vidljivost">
        </p>
        <button id="Potvrdi"> Potvrdi </button>
    </div>
    <div class="NoviElementi">

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

Iz priloženog koda, možemo videti da je naša stranica podeljena na dva dela. Jedan deo predstavlja *div* element „Obrazac“, dok drugi deo predstavlja *div* element „NoviElementi“. Iz ovakve podele jasno se može zaključiti da nam je cilj da nove elemente stranice dodamo u *div* element „NoviElementi“. Naša stranica trenutno izgleda kao na slici 1.4

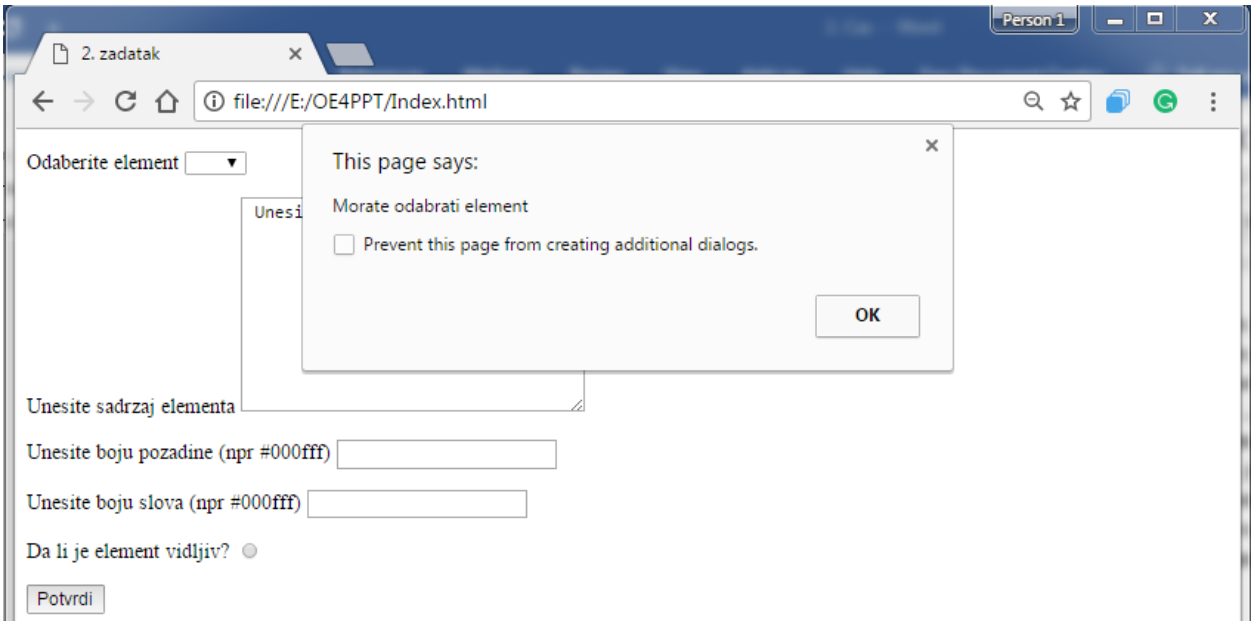


1.4 - Izgled stranice nakon dodavnja elemenata u cilju prikupljanja informacija

Sada imamo zadatak da implementiramo funkcionalnost primenom JS koda. Dakle, kada korisnik klikne na *button* element „Potvrdi“, treba proveriti pravilnost popunjenog obrasca za dodavanje novih elemenata na stranicu. Najpre treba proveriti da li je odabran neki element i, ukoliko nije, obavestiti korisnika o tome ispisivanjem odgovarajuće poruke. Da bismo mogli da vidimo da li je odabran neki element prvo treba da referenciramo element *select* koji predstavlja padajuću listu. Ovaj element referenciramo na osnovu vrednosti *id* atributa primenom metode *getElementById*. Nakon toga, ispitujemo da li je korisnik uneo neku vrednost proveravajući da li je trenutno odabrana vrednost praznog stringa. Trenutno odabrana vrednost se smešta unutar atributa *value* ovog elementa. Na osnovu ovoga, dobijamo sledeći sadržaj *code.js* fajla:

```
var Button = document.getElementById('Potvrdi');
Button.onclick = function(){
    var DropList = document.getElementById('Element');
    if( DropList.value == "" ){
        window.alert("Morate odabrati element");
    }
};
```

Za razliku od prethodnog primera, sada referencu na element *Button* dobijamo primenom druge funkcije koja referencira element na osnovu *id* vrednosti. Rezultat koji dobijamo primenom ove funkcije smeštamo u promenljivu *Button*. Usled klika na button element poziva se deo koda označen sa *function*, unutar koga pristupamo elementu *select* na osnovu *id* vrednosti „*Element*“. Dobijenu referencu smeštamo u promenljivu *DropList*. Atributu *value* pristupamo pomoću *.value* i ispitujemo da li je vrednost ovog atributa jednaka praznom stringu (“ “). Ukoliko jeste, treba obavestiti korisnika da nije dobro popunio formu. Najbolji način da se to postigne jeste pomoću *window* objekta na koji ćemo ispisati odgovarajuću poruku. Metoda *alert* ovog objekta služi za ispis odgovarajuće poruke. Sada, ukoliko otvorimo našu stranicu, ostavimo prazno polje padajuće liste i kliknemo na potvrdi dobijamo izgled kao na slici 1.5



1.5 - Izgled stranice nakon implementiranja funkcionalnosti za proveru unetih podataka u padajuću listu

Prikazani kod koji realizuje navedenu funkcionalnost se može zapisati kompaktije na sledeći način:

```
var Button = document.getElementById('Potvrdi');
if(document.getElementById('text').value.length > 25){
        window.alert("Duzina teksta je veca od 25 karaktera");
        window.alert("Morate odabrati element");
    }
};
```

vrednost atributa value je tipa string

Referenca na element

Atribut referenciranog elementa

Nakon što smo ispitali sadržaj padajuće liste, neophodno je ispitati da li sadržaj elementa ima više od 25 karaktera. To se postiže dodavanjem sledećeg koda:

```
if(document.getElementById('text').value.length > 25){
        window.alert("Duzina teksta je veca od 25 karaktera");
    }
```

Tip vrednosti atributa *value* je *string*. Svaki *string* ima osobinu *length* čija vrednost označava broj karaktera u stringu. Dakle, kada imamo neki *string*, dužinu tog stringa možemo proveriti pomoću osobine *length*. Ako nakon dodavanja ovog dela koda učitamo stranicu u *Web Browser* i unesemo sadržaj koji ima broj karaktera veći od 25 videćemo da se javlja isti tip poruke kao i u prethodnom slučaju.

Za sada smo uneli proveru da li je selektovan element i da li je dužina unetog sadržaja manja od 25. Sada treba proveriti da li uneta vrednost koja se odnosi na boje zadovoljava odgovarajući format. Ovu proveru najbolje je sprovesti u posebnoj funkciji koja ima sledeću definiciju:

```
function CheckValue(Value){
    if(Value.charAt(0) != '#'){ return false;}
    if(Value.substr(1,Value.length).length == 6){

    }
    else{
        return false;
    }
}
```

Prikazani deo koda predstavlja primer kako se definišu funkcije u JS kodu. Naime, svaka definicija funkcije počinje rezervisanom rečju *function* iza koje sledi naziv funkcije, u našem slučaju *CheckValue*. Iza naziva funkcije mogu, a i ne moraju, biti navedeni argumenti. U našem slučaju argument je *Value* koji predstavlja podatak tipa *string* čija vrednost predstavlja sadržaj *input* elementa tipa *text*. Prvi *if* uslov proverava prvi karakter *string*-a *Value*.

Objekat tip-a *string* ima metode *charAt(position)* i *substr(start,end)*. *charAt* metoda služi za referenciranje karaktera unutar stringa na poziciji *position*, dok *substr* metoda služi za izdvajanje podstringa počevši od karaktera na poziciji *start* do karaktera na poziciji *end-1*.

Dakle, ukoliko je prvi karakter u stringu *Value* različit od karaktera „#“ to je prvi znak da korisnik nije uneo dobar format i u tom slučaju se izlazi iz funkcije i kao povratna vrednost funkcije vraća se *false*. U suprotnom proveravamo da li je broj karaktera iza znaka „#“ jednak 6 i ako nije vraćamo *false* dok je u suprotnom potrebno sprovesti detaljniju proveru za svaki od 6 karaktera. To podrazumeva proveru da li je svaki karakter cifra ili jedan od slova heksadecimalnog zapisa. Kôd koji implementira traženu funckionalnost je sledeći:

```
function CheckValue(Value){
    if(Value.charAt(0) != '#'){ return false;}
    if(Value.substr(1,Value.length).length == 6){
        var HexAlph = "AaBbCcDdEeFf";
        for(i=1;i<Value.length;i++){
            if(isNaN(Value.charAt(i))){
                if(HexAlph.indexOf(Value.charAt(i))<0){
                    return false;
                }
            }
        }
        return true;
    }
    else{
        return false;
    }
}
```

Ukoliko smo utvrdili da string počinje sa karakterom „#“ i da iza njega sledi 6 karaktera, neophodno je ispitati vrednost svakog od tih 6 karaktera. To se postiže tako što u jednoj *for* petlji za svaki od karaktera proverimo da li predstavlja broj. Ukoliko je posmatrani karakter broj, prelazimo na ispitivanje sadržaja narednog karaktera u nizu. Ukoliko posmatrani karakter nije broj, ispitujemo da li pripada skupu slova iz heksadecimalnog zapisa. Provera da li je neki karakter broj vrši se pomoću funkcije *isNaN* koja vraća *true* ukoliko karakter **nije** broj dok u suprotnom vraća *false*. Provera toga da li posmatrani karakter pripada skupu heksadecimalnih karaktera vrši se na taj način što prvo definišemo neki string karaktera, a zatim, pomoću metode stringa *indexOf(“value“),* tražimo da li se u tom stringu nalazi *value*. Ukoliko se unutar stringa nalazi *value*, ova metoda vraća index prvog karaktera u nizu za koji je našla poklapanje. Ukoliko karakter ne postoji, ova funkcija vraća -1.

Nakon što smo definisali funkciju *CheckValue*, potrebno je obezbediti da se ona poziva pritiskom na element *button* što se postiže dodavanjem ove funkcije u deo koda koji obrađuje *onclick* event. Ista funkcija se primenjuje na oba *input* elementa zadužena za unos boja. Poželjno je deo koda koji proverava ispravnost unetih podataka izvojiti u posebnu funkciju koja će vraćati vrednost *true* ukoliko su zahtevani podaci uneti ispravno dok bi u suprotnom vraćala *false*. Ta funkcija se u ovom primeru zove *DataValid*. Nakon ovih izmena, *code.js* fajl ima sledeću sadržinu:

```
var Button = document.getElementById('Potvrdi');

function CheckValue(Value){
    if(Value.charAt(0) != '#'){ return false;}
    if(Value.substr(1,Value.length).length == 6){
        var HexAlph = "AaBbCcDdEeFf";
        for(i=1;i<Value.length;i++){
            if(isNaN(Value.charAt(i))){
                if(HexAlph.indexOf(Value.charAt(i))<0){
                    return false;
                }
            }
        }
        return true;
    }
```

```
    else{
        return false;
    }
}

function DataValid(){
    var BColor = document.getElementById('BojaPozadine').value;
    var TColor = document.getElementById('BojaSlova').value;
    if(document.getElementById('Element').value == "" ){
        window.alert("Morate odabrati element");
        return false;
    }
    if(document.getElementById('text').value.length > 25){
        window.alert("Duzina teksta je veca od 25 karaktera");
        return false;
    }
    if(CheckValue(BColor)!= true){
        window.alert("Format vrednosti boje pozadine nije dobar");
        return false;
    }
    if(CheckValue(TColor)!= true){
        window.alert("Format vrednosti boje slova nije dobar");
        return false;
    }
    return true;
}

Button.onclick = function(){
    if(DataValid()){
        window.alert("Uspesan unos");
    }
};
```

Nakon implementirane funkcionalnosti, koja se odnosi na proveru unetih informacija, neophodno je implementirati kreiranje odgovarajućih elemenata na osnovu unetih informacija. Elementi se kreiraju nakon klika na element *button*, što znači da je funkcionalnost za kreiranje novih elemenata potrebno implementirati u okviru funkcije koja obrađuje događaj *onclik* koji je vezan za ovaj element. Deo koda funkcije kojom se postiže zahtevana funkcionalnost je sledeći:

```
Button.onclick = function(){
    if(DataValid()){
        switch(document.getElementById('Element').value){
            case "h1":
                var h1 = document.createElement('h1');
                h1.textContent = document.getElementById('text').value;
                h1.style.color =
document.getElementById('BojaPozadine').value;
                h1.style.backgroundColor =
document.getElementById('BojaSlova').value;
                h1.hidden = !document.getElementById('Vidljivost').checked;
                document.querySelector('.NoviElementi').appendChild(h1);
            break;
            case "h2":
                var h2 = document.createElement('h2');
                h2.textContent = document.getElementById('text').value;
                h2.style.color =
document.getElementById('BojaPozadine').value;
                h2.style.backgroundColor =
document.getElementById('BojaSlova').value;
                h2.hidden = !document.getElementById('Vidljivost').checked;
                document.querySelector('.NoviElementi').appendChild(h2);
            break;
            case "h3":
                var h3 = document.createElement('h3');
                h3.textContent = document.getElementById('text').value;
                h3.style.color =
document.getElementById('BojaPozadine').value;
                h3.style.backgroundColor =
document.getElementById('BojaSlova').value;
                h3.hidden = !document.getElementById('Vidljivost').checked;
                document.querySelector('.NoviElementi').appendChild(h3);
            break;
            case "p":
                var p = document.createElement('p');
                p.textContent = document.getElementById('text').value;
                p.style.color =
document.getElementById('BojaPozadine').value;
                p.style.backgroundColor =
document.getElementById('BojaSlova').value;
                p.hidden = !document.getElementById('Vidljivost').checked;
                document.querySelector('.NoviElementi').appendChild(p);
            break;
        }
    }
};
```

Nakon što kliknemo na element *button* na HTML stranici poziva se deo koda koji obrađuje taj događaj. Unutar tog koda, pod uslovom *if* iskaza, poziva se funkcija *DataValid* koja proverava da li su podaci ispravno uneti. Ukoliko nisu, korisnik se obaveštava odgovarajućom porukom i ne ulazi se u deo *if* uslova koji treba da sprovede odgovarajuću funkcionalnost.

Unutar ovog *if* uslova nalazi se switch-case struktura koja ima za zadatak da sprovede akciju na osnovu tipa kreiranog elementa. Nezavisno od tipa kreiranog elementa unutar *case* strukture sprovode se sledeći koraci:

1. kreira se element odgovarajućeg tipa pozivom metode *createElement('ElementName')*
2. dodeljuje se sadržaj kreiranom elementu tako što se uneta vrednost iz pravougaonika sa HTML stranice dodeli atributu *textContent* kreiranog elementa
3. CSS astributu *style* promeni vrednost osobine *color* na vrednost definisanu od strane korisnika
4. CSS astributu *style* promeni vrednost osobine *backgroundColor* na vrednost definisanu od strane korisnika
5. u zavisnosti od toga da li je korisnik odabrao da želi da pravougaonik bude vidljiv ili ne, atributu *hidden* se dodeljuje odgovarajuća vrednost.
6. na kraju se *div* elementu „NoviElement", pozivom metode *appendChild*, dodeljuje kreirani element

Pošto je niz koraka potrebnih za kreiranje elemenata skoro isti, navedene korake (osim prvog) je moguće izvojiti u zasebnu funkciju *CreateElementProperties*. Na taj način je smanjen broj linija koda, a samim tim povećana njegova razumljivost. Definicija ove funkcije je:

```
function CreateElementProperties(Element){

    Element.textContent = document.getElementById('text').value;
    Element.style.color = document.getElementById('BojaPozadine').value;
    Element.style.backgroundColor =
document.getElementById('BojaSlova').value;
    Element.hidden = !document.getElementById('Vidljivost').checked;
    document.querySelector('.NoviElementi').appendChild(Element);
}
```

Pošto je postavkom zadatka rečeno da klik miša na kreirani element treba da uzrokuje uklanjanje elementa sa stranice, potrebno je svakom kreiranom elementu dodati funkciju koja obrađuje događaj „Klik miša na element". Ova funkcionalnost se postiže pomoću metode *addEventListener("EventType",FunctionName).* Dakle, kada se nad elementom desi događaj tipa *„EventType"* doći će do pozivanja funkcije *FunctionName*. U funkciju *CreateElementProperties* potrebno je dodati sledeću liniju koda:

```
    Element.addEventListener("click",RemoveElement);
```

koja definiše da se usled pojave događaja „Klik miša na element" pozove funkcija *RemoveElement*. Sada je naš zadatak da definišemo funkcionalnost ove funkcije. Kôd kojim se definiše ova funkcionalnost je :

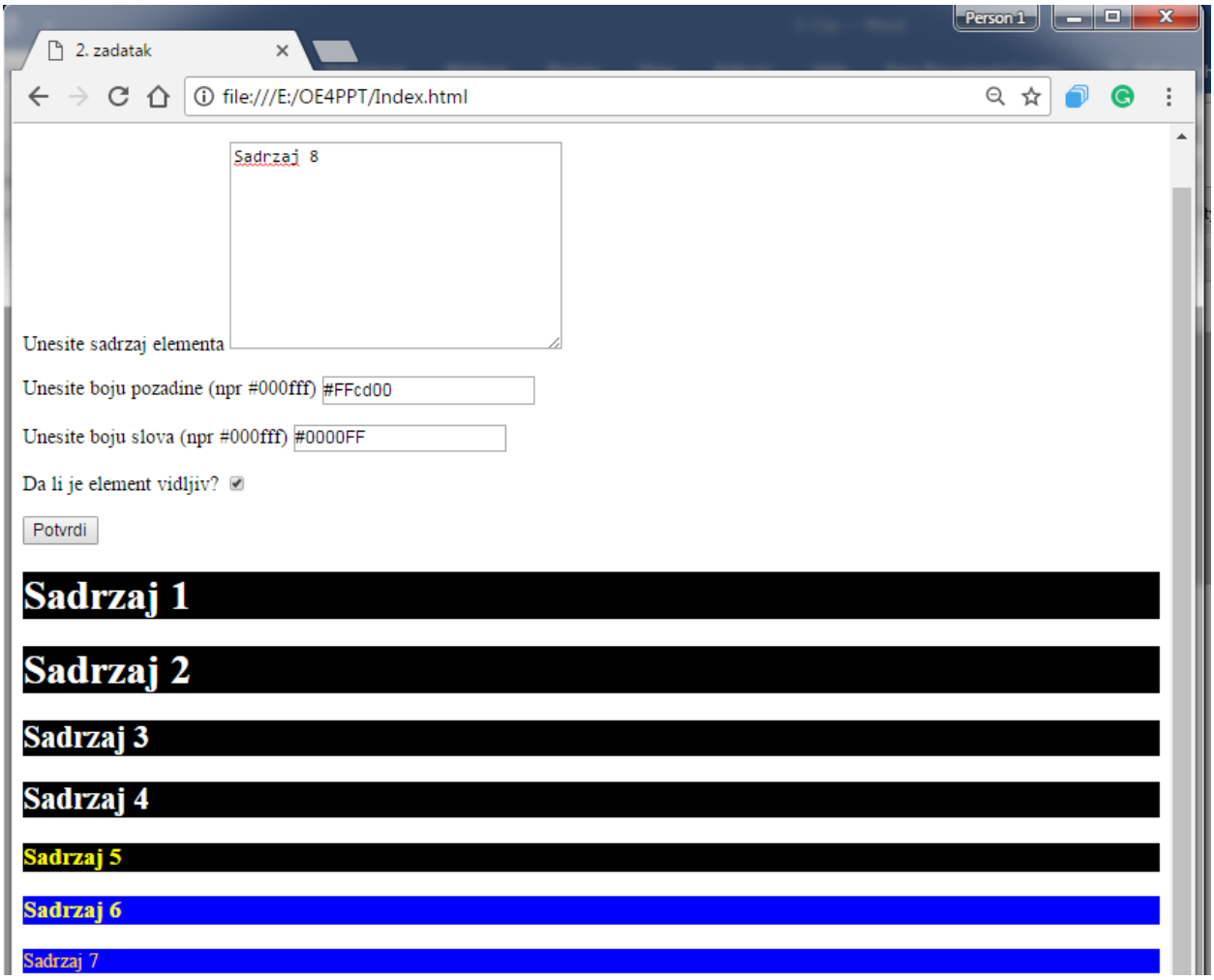
```
function RemoveElement(){
    document.querySelector('.NoviElementi').removeChild(this);
}
```

Dakle, usled klika na neki od kreiranih elemenata pozvaće se funkcija *RemoveElement* koja ima zadatak da ukloni kreirani element iz *div* elementa „NoviElement". Ovog puta, jedini način da referenciramo objekat na koji smo kliknuli mišem jeste pomoću reference *this*. Nakon ovih modifikacija sadržaj našeg *code.js* fajla je:

```
var Button = document.getElementById('Potvrdi');

function CheckValue(Value){
    if(Value.charAt(0) != '#'){ return false;}
    if(Value.substr(1,Value.length).length == 6){
        var HexAlph = "AaBbCcDdEeFf";
        for(i=1;i<Value.length;i++){
            if(isNaN(Value.charAt(i))){
                if(HexAlph.indexOf(Value.charAt(i))<0){
                    return false;
                }
            }
        }
        return true;
    }
    else{
        return false;
    }
}

function DataValid(){
    var BColor = document.getElementById('BojaPozadine').value;
    var TColor = document.getElementById('BojaSlova').value;
    if(document.getElementById('Element').value == "" ){
        window.alert("Morate odabrati element");
        return false;
    }
    if(document.getElementById('text').value.length > 25){
        window.alert("Duzina teksta je veca od 25 karaktera");
        return false;
    }
    if(CheckValue(BColor)!= true){
        window.alert("Format vrednosti boje pozadine nije dobar");
        return false;
    }
    if(CheckValue(TColor)!= true){
        window.alert("Format vrednosti boje slova nije dobar");
        return false;
    }
    return true;
}
function RemoveElement(){
    document.querySelector('.NoviElementi').removeChild(this);
}

function CreateElementProperties(Element){

    Element.textContent = document.getElementById('text').value;
    Element.style.color = document.getElementById('BojaPozadine').value;
    Element.style.backgroundColor =
document.getElementById('BojaSlova').value;
    Element.addEventListener("click",RemoveElement);
    Element.hidden = !document.getElementById('Vidljivost').checked;
    document.querySelector('.NoviElementi').appendChild(Element);
}
Button.onclick = function(){
    if(DataValid()){
```

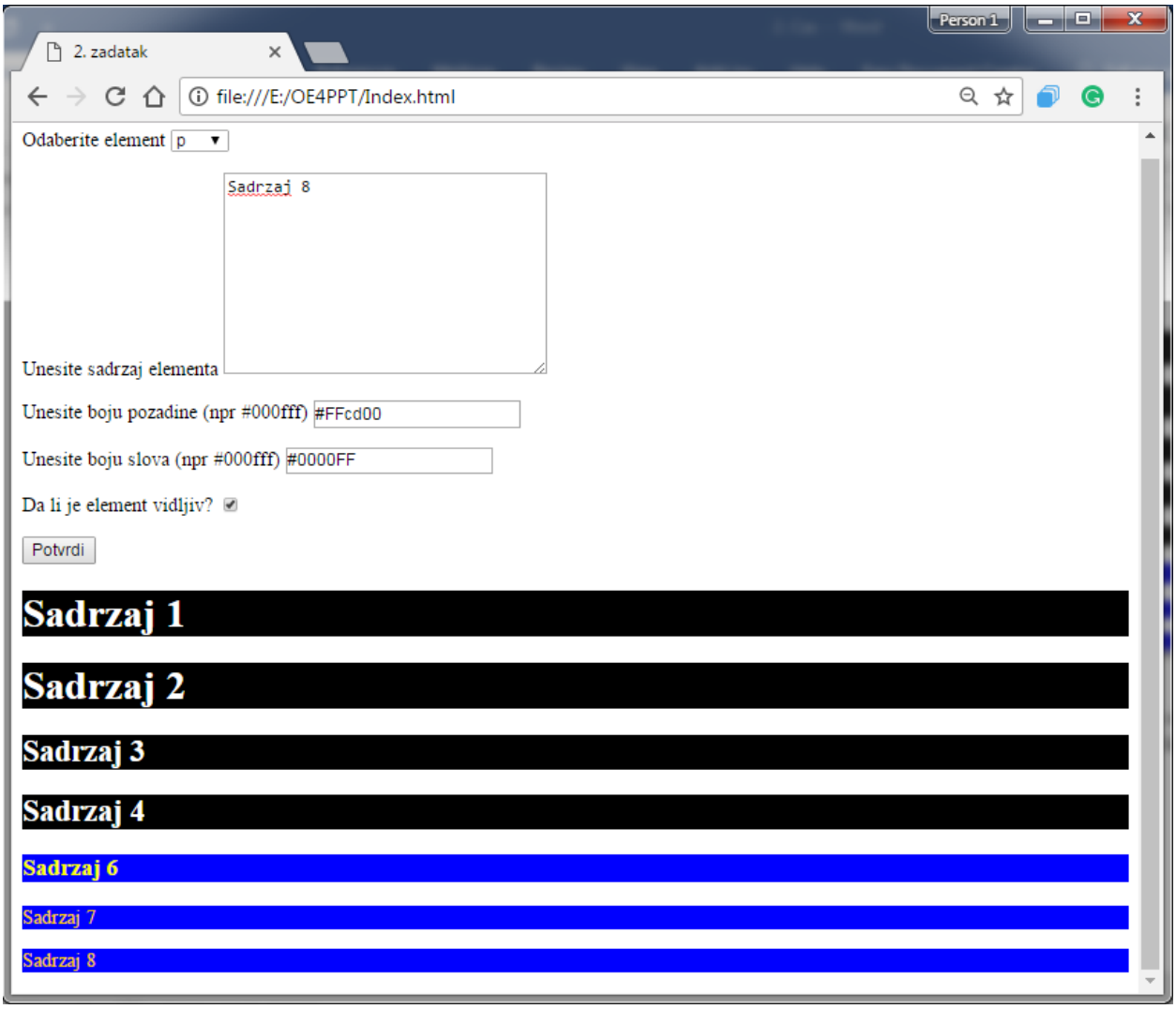
```
                var h1 = document.createElement('h1');
                CreateElementProperties(h1);
            break;
            case "h2":
                var h2 = document.createElement('h2');
                CreateElementProperties(h2);
            break;
            case "h3":
                var h3 = document.createElement('h3');
                CreateElementProperties(h3);
            break;
            case "p":
                var p = document.createElement('p');
                CreateElementProperties(p);
            break;
        }
    }
};
```

Izgled naše stranice, nakon testiranja funkcionalnosti koja obuhvata kreiranje različitih elemenata, je prikazan na slici 1.6



1.6 - Izgled stranice nakon dodavanja elemenata

Ako sada kliknemo na neki od kreiranih elemenata videćemo da će on nestati sa stranice. Na primer, ako kliknemo na element sa sadržajem „Sadržaj 5“ dobijamo izgled stranice kao na slici 1.6



1.7 - Izgled stranice nakon uklanjanja elementa sa sadržajem "Sadržaj5"

Da bi se testirala funkcionalnost polja vidljivosti, u HTML fajlu se može odabarti tip *input* elementa *checkbox* umesto *radio*, ili se u okviru funkcije *CreateElementProperties* može dodati deo koda koji će da, pri kraju funkcije, vrati *radio button* na *false* vrednost, kako bi ostala neodabrana. U slučaju *checkbox* tipa, ovo nije neophodno jer se *checkbox* može selektovati i deselektovati.

(Na osnovu do sada izloženog objašnjenja, potrebno je da student na ovus stranicu doda i mogućnost zadavanja informacije o dimenzijama elementa. Širina elementa ne sme biti veća 600px dok njegova visina ne sme biti veća od 300px.)