CoffeScript kao alternativa za JavaScript

Pavlović, Antonio

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Antonio Pavlović

COFFESCRIPT KAO ALTERNATIVA ZA JAVASCRIPT

ZAVRŠNI RAD

Varaždin, 2021.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Antonio Pavlović

JMBAG: 0016139516

Studij: Primjena informacijske tehnologije u poslovanju

COFFESCRIPT KAO ALTERNATIVA ZA JAVASCRIPT

ZAVRŠNI RAD

Mentor:

Prof. dr. sc. Dragutin Kermek

Varaždin, rujan 2021.

Antonio Pavlović

Izjava o izvornosti

Izjavljujem da je moj završni rad izvorni rezultat mojeg rada te da se u izradi istoga nisam koristio drugim izvorima osim onima koji su u njemu navedeni. Za izradu rada su korištene etički prikladne i prihvatljive metode i tehnike rada.

Autor potvrdio prihvaćanjem odredbi u sustavu FOI-radovi

_____

Sažetak

Ovaj radi se bavi proučavanjem JavaScript-a i uspoređivanjem s njegovom alternativom, CoffeeScript-om. Na početku se objašnjavaju osnovni pojmovi i s kojim jezicima nastaju Web aplikacije. Potom se definiraju načini na koje se može pratiti aktivnost korisnika na Webu te se dotiče teme vezane za aktivne korisničke dijelove Web aplikacije. Prikazuje se sama specifičnost implementacije tih dijelova u Web aplikaciji. Nakon toga dolazi do teme JavaScript-a gdje se objašnjavaju najbitnije stvari kod tog programskog jezike. Uz objašnjeni JavaScript, počinje se opisivati CoffeeScript te se prikazuje primjer izrađene Web aplikacije koja sadržava primjenu CoffeeScripta. Na kraju se opisuju prednosti i mane rada s CoffeeScriptom nakon čega se donosi konkretan zaključak, ali i odgovor na pitanje je li CoffeeScript dobra alternativa za JavaScript?

Ključne riječi: Web aplikacija, razvoj, JavaScript, CoffeeScript, korisnički dijelovi, praćenje aktivnosti, dinamičke osobine

Sadržaj

1. Uvod

Završni rad se bavi temom CoffeeScripta kao mogućom alternativom za JavaScript. Polazna točka ove teme je objašnjavanje teorijske pozadine koja stoji iza samog CoffeeScripta, to jest JavaScripta. Stoga se u početku objašnjavaju teme poput Web aplikacija, praćenja aktivnosti korisnika putem Weba i korisnički dijelovi Web aplikacija. Nakon toga dolazi do objašnjavanja JavaScripta i to njegovog razvoja, sintakse i određenih elemenata samoga jezika. Nakon što se objasni osnovno o JavaScriptu, kreće se s CoffeeScriptom. Što se samog CoffeeScripta tiče, o njemu je najbitnije reći kako nema prevelikih razlika od JavaScripta, izuzevši sintaksu i način pisanja koda. Bitno je napomenuti kako se sam CoffeeScript kompilira u JavaScript čime se zapravo dolazi do toga da je sam CoffeeScript zapravo JavaScript. Potom se uz teorijski dio rada navodi praktični dio rada koji uključuje izradu jednostavne Web aplikacije čija je tema poslovna Web aplikacija za obrt koji se bavi uslugama popravljanja oštećenja na autolimariji i termolakiranja. Sama Web aplikacija je izrađena u PHP programskom jeziku uz dodatak kompiliranog JavaScripta, zapravo napisanog pomoću CoffeeScripta. Unutar same aplikacije se administratoru omogućuje unos radnih naloga, čitanje poruka, kreiranje korisnika, promjena statusa naloga te uređivanje galerije slika. Običan korisnik, ukoliko mu se odobri registracija može pristupiti pregledu poslanih poruka sa svojega računa, pregledu radnih naloga koji se na njega odnose i tako dalje. Nakon predstavljenog primjera Web aplikacije govori se o dobrim i lošim stranama CoffeeScripta u odnosu na JavaScript te se na kraju u zaključku odgovara na pitanje je li CoffeeScript dobra alternativa za JavaScript.

2. Web aplikacije

Otkada je Tim Berners-Lee prvi put pokrenuo ideju o svjetskoj mreži (poznatijoj kao engl. *World Wide Web*) 1989. godine, počela je nova razina u razvoju Interneta čime su se otvorila mnoga vrata za kreiranje web stranica. Kako su napisali Berners-Lee, Cailliau, Luotonen, Nielsen i Secret (1994.) „Svjetska mreža je kreirana kako bi bila bazen ljudskog znanja, koja će omogućiti suradnicima rad na udaljenim mjestima kako bi dijelili svoje ideje i sve aspekte zajedničkog projekta."

Samim time je i kreiran poseban protokol za dijeljenje Web mjesta na Internetu, a to je HTTP (engl. *Hypertext Transfer Protocol*) protokol koji je danas ima svoju sigurniju nadogradnju HTTPS (engl. *Hypertext Transfer Protocol Secure*) u svrhu dodavanja veće sigurnosti prilikom rukovanja s osjetljivim i osobnim podacima. Potrebno je spomenuti kako je bitno razlikovati Web stranicu od Web aplikacije. Web stranica predstavlja dokument koji je napisan u HTML-u (engl. *HyperText Markup Language*), dok je Web aplikacija kreirana korištenjem određenog programskog jezika kako bi poslužitelj generirao Web stranice za pojedinog korisnika koji pristupa sadržaju na Web aplikaciji. Kako bi se povećala mogućnost Interneta, kreirane su razne Web aplikacije. U početku su te Web aplikacije bile veoma jednostavne i bez dodatnih dizajnerskih stilova. Prolaskom vremena povećao se interes za razvoj Web aplikacija, a i samim time dolazi do želje da svaka Web aplikacija ima svojevrstan dizajn i prepoznatljivost na Internetu. U današnje vrijeme postoji više pristupa razvoju Web aplikacija ovisno o njihovoj namjeni. Stoga se može prepoznati više načina razvoja poput dizajniranja Web aplikacija, razvoj Web aplikacija s korisničke strane i razvoj Web Aplikacija s poslužiteljske strane.

Jazayeri (2007.) navodi kako je Web aplikacija vrsta aplikacije koja pokrenuta unutar Web preglednika, te se nalazi na Internetu. Dok drugi autor Hadley (2006.) govori kako je Web aplikacija dinamička aplikacija bazirana na HTTP-u čije su interakcije podložne strojnoj obradi. Kod Web aplikacije je slučaj da postoji generiranje Web stranica ovisno o izvođenju instrukcija u određenome programu. Na taj način se na strani poslužitelja generira Web stranica ovisno o korisniku koji zahtjeva pristup Web aplikaciji. Stoga kod Web aplikacija dolazi do dinamičkog sadržaja koji povećava funkcionalnost Web mjesta na Internetu i omogućava određenu dozu personalizacije sadržaja za pojedinog korisnika.

Conallen (1999.) piše kako arhitektura Web aplikacije nije ništa više drugačije od arhitekture dinamičke Web stranice. Samim time se očituje kako za potrebe izrade Web aplikacije nema većih razlika od kreiranja Web stranice. Naravno tijekom razvoja Web aplikacija dolazi do značajnijeg pomaka prilikom obrade podataka kako bi se u konačnici

postigla dinamičnost i generiranje Web stranica ovisno o zahtjevu koji se upućuje prema poslužitelju.

2.1. Podjela Web aplikacija

Kod Web aplikacija, postoji određena podjela istih. Makhija (n.d.) radi podjelu Web aplikacija na način da postoje:

- Statičke Web aplikacije
- Dinamičke Web aplikacije
    - Jednostranične Web aplikacije
    - Više-stranične Web aplikacije
    - Web portali
    - Animirane Web aplikacije
    - Bogate Internet aplikacije
    - Web aplikacije pogonjene JavaScriptom
    - Progresivne Web aplikacije
    - Web aplikacije e-Trgovina

Statičke Web aplikacije bi bile aplikacije koje nemaju fleksibilnost već su univerzalne i za svakog korisnika iste. Dinamičke Web aplikacije u drugu ruku, su one aplikacije koje imaju fleksibilnost i za pojedinu ulogu korisnika generiraju pripadajuću Web stranicu. Kod dinamičkih Web aplikacija postoji podjela na tematiku samih Web aplikacija. Jednostranične Web aplikacije su Web aplikacije koje su zasnovane na sadržaju koji se u cijelosti nalazi na jednoj Web stranici. Više-stranične Web aplikacije koriste više Web stranica kako bi se veća količina sadržaja raspodijelila podjednako po cijeloj aplikaciji. Samim time koriste poveznice koje vode do potrebnih Web stranica. Web portali su Web aplikacije koje najčešće koriste novinarska poduzeća poput 24 sata ili Index.hr gdje se sadržaj mijenja ovisno o pristiglim vijestima. Animirane Web aplikacije su prezentacijskog karaktera i služe za prezentiranje poduzeća, te privlačenje većeg broja korisnika. Bogate Internet aplikacije su aplikacije koje pružaju veću interaktivnost i korisničko iskustvo od standardnih Web aplikacija. Web aplikacije pogonjene JavaScriptom su aplikacije koje su dobile na popularnosti dolaskom JavaScript okvira poput Angular.js, React.js i sličnih okvira. One se najviše zasnivaju na razvoju korisničke strane Web aplikacije. Progresivne Web aplikacije su vrlo slične mobilnim aplikacijama, stoga se može

zaključiti kako je njihov razvoj i korištenje prvobitno namijenjeno mobilnim korisnicima. Na kraju se navode Web aplikacije e-Trgovine koje omogućuju upravo kupovinu putem Interneta.

Murugesan i Ginige (2005.) kategoriziraju Web aplikacije prema funkcionalnosti. Na taj način se Web aplikacije dijele na:

- Informacijske
- Interaktivne
- Transakcijske
- Orijentirane na tijek rada (engl. *Workflow*)
- Okruženja za kolaborativan rad
- Mrežne (engl. *online*) zajednice i tržnice

Informacijske Web aplikacije su tipa online novinskih portala, online knjiga, kataloga proizvoda i slične aplikacije. Interaktivne Web aplikacije su najčešće online igrice, ali mogu biti i registracijski obrasci. Transakcijske Web aplikacije se najviše orijentiraju na online kupovinu i e-bankarstvo. Web aplikacije koje su orijentirane na tijek rada su najčešće tematike online planiranja te upravljanja raznim procesima poput upravljanja inventarom, upravljanje lancem opskrbe i tako dalje. Web aplikacije koje su tipa okruženja za kolaborativan rad su Web aplikacije poput Figma-e, koja predstavlja alat za kolaborativno dizajniranje. Online zajednice i tržnice su zapravo e-aukcije, grupe za diskusiju i slične tematike.

Murugesan i Ginige (2005.) govore kako se Web aplikacije mogu kategorizirati na više načina, zbog toga što danas Web aplikacije imaju razne funkcionalnosti i različite karakteristike te same zahtjeve koje moraju ispunjavati. Te smatraju kako je njihova podjela korisna prilikom razumijevanja zahtjeva koje sama Web aplikacija mora ispunjavati te za razvoj sustava i aplikacija zasnovanih na Webu.

3. Programski jezici koji stvaraju Web aplikacije

Web aplikacije se mogu izraditi poznavanjem jezika HTML i CSS, uz dodatak JavaScript-a kako bi se poboljšala interaktivnost aplikacije. Navedeni jezici se najčešće koriste prilikom izrade statičkih Web aplikacija.

3.1. HTML, CSS i JavaScript

HTML je jezik kojeg Ragget, Le Hors i Jacobs (1999.) definiraju kao jezik koji se koristi za objavljivanje sadržaja na svjetskoj mreži, odnosno engl. *World Wide Web*. Pomoću HTML-a se mogu kreirati dokumenti koji koriste hipermedijske elemente poput: teksta, slike, zvuka, videozapisa i animacija. HTML koristi oznake za određene elemente u dokumentu. Na taj način HTML kod izgleda ovako:

```
<!DOCTYPE html>
<html lang="hr">
<head>
     <title>Primjer</title>
     <meta charset="UTF-8"/>
</head>
<body>
     <h1>Ovo je primjer HTML dokumenta</h1>
     <p>Ovo je odlomak</p>
</body>
</html>
```

Prvi red koda predstavlja preambulu odnosno definiranje vrste dokumenta, prema definiranome je dokument tipa HTML 5. Druga linija definira početak HTML dokumenta i jezik koji se koristi. Nakon toga se definira „head“ sekcija gdje se navode informacije o Web stranici. Elementi poput naslova Web stranice i meta podataka koji mogu uključivati autora, opis i slova koja se koriste na Web stranici, definiraju se u „head“ dijelu koda. Nakon toga dolazi dio koji se prikazuje korisniku na monitoru, a to je dio u „body“ sekciji, iliti tijelu Web stranice. U samome tijelu je moguće definirati elemente za naslove, podnaslove, odlomke, članke, tablice, slike i ostale medije. U primjeru je navedena oznaka „h1“ koja se koristi za najveći naslov. Nakon toga je definiran element za odlomak u kojemu se nalazi običan tekst. Glavno pravilo kako bi se kod mogao „izvršiti“ i prikazati na zaslonu korisnika, svaka oznaka elementa mora nakon „otvaranja“ biti i „zatvorena“. To je najbolje objasniti na primjeru kada oznaka „`<p>`“ ima i pripadajući „`</p>`“ koji označava u praksi kraj odlomka.

Na navedenom primjeru mogu se dodavati i CSS upute. Mikkonen i Taivalsaari (2008.) govore kako je CSS jezik tablice stilove i koristi se kako bi se opisali prezentacijski aspekti HTML dokumenta. Same CSS upute mogu se pojaviti u HTML dokumentu na tri načina. Prvi način je da se ispod „`</head>`" oznake definira „`<style></style>`" dio u kojem će se unositi CSS upute. Drugi način je u vanjskoj datoteci s ekstenzijom „.css", uz dodatak „`<link rel="stylesheet" href="primjer.css"/>`". Treći način je da se u redu gdje je oznaka za element definira „`style`" atribut s CSS uputama. Treći način je prikazan u kodu na sljedeći način:

```
…
<h1 style="text-align: center;">Ovo je primjer HTML dokumenta</h1>
…
```

U navedenom primjeru je prikazana CSS uputa koja će poravnati tekst unutar oznake na sredinu zaslona. Same CSS upute se koriste za bolju prezentaciju sadržaja na Web stranici te su neophodne za razvoj same Web aplikacije.

Mikkonen i Taivalsaari (2008.) pišu o JavaScriptu kao o kodu koji se može izvršavati i najčešće se koristi uz HTML i CSS. JavaScript uz DOM (engl. *Document Object Model*), platformu na kojoj se nalazi kolekcija objekata koji čine Web stranicu, može programski analizirati Web stranicu i promijeniti samu Web stranicu.

JavaScript se u HTML dokumentu može javiti na dva načina. Prvi način je pri dnu HTML dokumenta u oznakama „`<script></script>`" ili drugi način kada postoji eksterna datoteka s „.js" ekstenzijom te se u pri dnu HTML dokumenta postavi oznaka „`<script src="primjer.js"></script>`" gdje atribut „`src`" označuje lokaciju vanjskog JavaScript dokumenta. Sam kod napisan u JavaScriptu koristi elemente Web stranice gdje uz određene varijable može manipulirati ostalim kodom te tako stvoriti interaktivne dijelove Web stranice, pogonjene određenim događajima.

3.2. PHP i MySQL

Lerdorf, Tatroe i MacIntyre (2006.) kažu da „PHP je jednostavan, ali moćan jezik dizajniran za kreiranje HTML sadržaja." Što u konačnici znači da uz malo znanja o programiranju, programer koji razvija Web aplikacije , može kreirati dinamički sadržaj i unaprijediti Web aplikaciju u smislu funkcionalnosti. PHP je programski jezik koji se zasniva na C programskom jeziku stoga sadrži elemente poput varijabli, instrukcija, funkcija, klasa i ostale elemente. PHP je inače rekurzivni akronim koji označava engl. „*PHP: Hypertext Preprocessor*" iliti PHP: hipertekstualni pretprocesor. On se najčešće koristi zajedno s MySQL jezikom kako bi se kreirale dinamičke Web aplikacije. MySQL je sustav za upravljanje

relacijskim bazama podataka koji koristi jezik SQL pomoću kojeg se može izvršavati manipulacija nad bazama podataka, odnosno entitetima, atributima i podacima koji se nalaze u samoj bazi.

Welling i Thomson (2003.) navode kako je MySQL snažan i brz sustav za upravljanje relacijskom bazom podataka. MySQL koristi jezik za definiranje podataka (engl. *Data Definition Language*, *DDL*), jezik za upite nad podacima (engl. *Data Query Language*, *DQL*), jezik za upravljanje podacima (engl. *Data Control* Language, DCL) i jezik za manipulaciju nad podacima (engl. *Data Manipulation Language*, *DML*).

4. Praćenje aktivnosti korisnika na Webu

Praćenje aktivnosti korisnika na Webu je jedna od glavnih karakteristika na Web aplikacijama. Samo praćenje aktivnosti korisnika uključuje promatranje korisnikove interakcije s Web aplikacijom, daje uvid u njegove preferencije, način rada s aplikacijom, ponašanje i zainteresiranost korisnika na Web aplikaciji. Podaci koje se u konačnici dobe služe najviše za marketing. Postoji više metoda pomoću kojih se može pratiti aktivnost korisnika. Najčešće korištena metoda je pomoću kolačića (engl. *Cookies*). Praćenje pomoću kolačića je u zadnje vrijeme postala tema svakog spora vezanog uz privatnost na Internetu jer se njihovim korištenjem stvara pitanje vezano uz pohranu privatnih podataka korisnika, te moguća daljnja distribucija istih. Osim kolačića postoje i druge metode, odnosno tehnologije praćenja aktivnosti korisnika na Internetu.

4.1. Tehnologije praćenja

Bujlow i ostali (2015.) su kategorizirali tehnologije praćenja na samo-sesijske tehnologije, tehnologije zasnovane na pohrani, tehnologije zasnovane na predmemoriji, tehnologije bazirane na uzorkovanju i ostale tehnologije. Prema tome su Bujlow i ostali (2015.) kreirali tablicu koja prikazuje mehanizme praćenja i njihove tehnologije pomoću kojih se izvršava praćenje.

Tablica 1. Tehnologije koje se koriste za praćenje korisnika (Bujlow i ostali, 2015.)

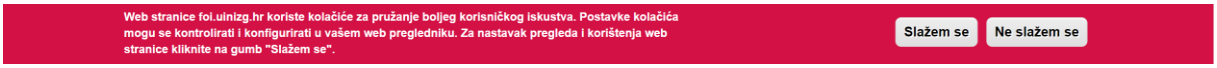
Sekcija	**Mehanizam praćenja**	**Tehnologija**
II	**Samo-sesijski**	
A	Sesijski identifikatori pohranjeni u skrivenim poljima	Web-poslužiteljska sesija
B	Eksplicitna autentifikacija pomoću Web-obrasca	Web-poslužiteljska sesija
C	„window.name“ DOM svojstvo	HTML5, JavaScript
III	**Zasnovani na pohrani**	
A	HTTP kolačići	HTTP zaglavlja, JavaScript
B	„Flash“ kolačići i Java „JNLP“ usluga dosljednosti	Flash / Java
C	„Flash“ objekt lokalne veze	Flash
D	„Silverlight“ izolirana pohrana	Silverlight
E	HTML5 Globalna, lokalna i sesijska pohrana	HTML5, JavaScript
F	Web SQL baza podataka i HTML5 indeksirana baza podataka	HTML5, JavaScript
G	Internet pretraživač (engl. *Internet Explorer*) pohrana korisničkih podataka	JavaScript
IV	**Zasnovani na predmemoriji**	
A	**Web predmemorija**	

	1	Ugrađivanje identifikatora u predmemoriranim dokumentima	HTML5, JavaScript
	2	Učitavanje testova performansi	Mjerenja na poslužiteljskoj strani, JavaScript
	3	E-oznake i zadnja promijenjena zaglavlja	HTTP zaglavlje
	B	**DNS pretraživanja**	
C		**Operacijska predmemorija**	
	1	HTTP 301 preusmjerena predmemorija	HTTP zaglavlje
	2	HTTP ovjerena (engl. *authentication*) predmemorija	HTTP zaglavlja, JavaScript
	3	HTTP strogo transportno osigurana predmemorija	HTTP zaglavlja, JavaScript
	4	TLS sesijsko nastavljanje predmemorije i TLS sesijski identifikatori	Web-poslužiteljska sesija
V		**Uzorkovanje**	
	A	Mrežno i lokacijsko uzorkovanje	IP adresa, geolokacijske tehnike zasnovane na poslužitelju, HTTP zaglavlja, HTML5, JavaScript, Flash i Java
	B	Uzorkovanje uređaja	IP adresa, TCP zaglavlja, HTTP zaglavlja, JavaScript, Flash
	C	Uzrokovanje slučaja operacijskog sustava	JavaScript, Flash, Java, ActiveX
	D	Uzorkovanje verzije Web preglednika	HTML5, JavaScript, CSS
	E	Uzorkovanje slučaja Web preglednika korištenjem platna (engl. *canvas*)	HTML5, JavaScript
	F	Uzorkovanje slučaja Web preglednika korištenjem povijesti pretraživanja na Webu	Mjerenja na strani poslužitelja, HTTP zaglavlja, JavaScript
	G	Ostale metode uzorkovanja slučajeva Web preglednika	HTTP zaglavlja, JavaScript, Flash
VI		**Ostali mehanizmi praćenja**	
	A	Zaglavlja prikvačena na odlazne HTTP zahtjeve	HTTP zaglavlja
	B	Korištenje mobilnih meta podataka	Zlonamjerni softver na pametnom uređaju (engl. *smartphone*)
	C	Vremenski napadi	HTML5, JavaScript, CSS
	D	Korištenje nesvjesne suradnje korisnika	HTML5, JavaScript, CSS, Flash
	E	Clickjacking	HTML5, JavaScript, CSS
	F	Vječni kolačići (super-kolačići) (engl. *Evercookies*)	Web-poslužiteljska sesija, HTTP zaglavlja, HTML5, JavaScript, Flash, Silverlight, Java

Prema navedenoj tablici je moguće vidjeti da postoji mnogo metoda praćenja korisnikove aktivnosti na Internetu, samim time je zaključivo da su najkorištenije tehnologije praćenja upravo pomoću tehnologija HTML5, JavaScript i HTTP zaglavlja.

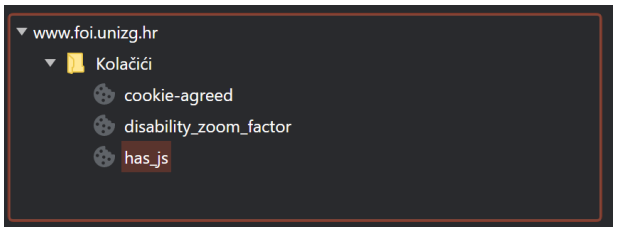
4.2. Osobine Web aplikacija u odnosu na praćenje aktivnosti

Što se tiče osobina samih Web aplikacija, one u današnje vrijeme moraju imati jasno naznačene koje aktivnosti praćenja koristi, odnosno ukoliko se to radi o pohrani podataka u kolačiće, Web aplikacija mora najprije od korisnika zatražiti zahtjev da se slaže da Web aplikacija pohranjuje njegovu aktivnost i podatke u kolačiće. Uzrok tome je zakon koji je donesen za područje cijele Europe, Opća uredba o zaštiti podatka, poznatija kao GDPR (engl. *General Dana Protection Regulation*) kojom se štite osobni podaci korisnika kod bilo kojeg davanja osobnih podatka u svrhu pohrane istih u određeni sustav. Stoga svaka Web aplikacija mora imati određenu obavijest s tipom obrasca preko kojeg će korisnik dati privolu za skupljanje podataka.



Slika 1. Izgled obavijesti o prihvaćanju skupljanja podataka u kolačiće na FOI Webu (foi.unizg.hr)

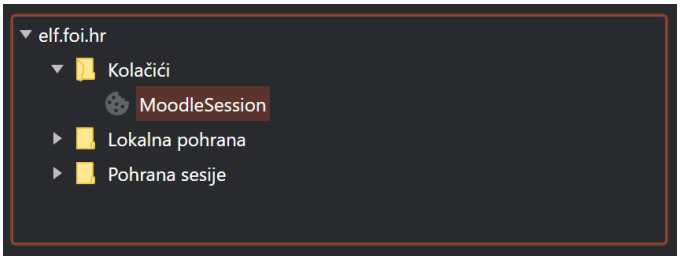
Nakon prihvaćanja zahtjeva da se mogu prikupljati podaci, odnosno korištenje kolačića, korisnikova aktivnost se počinje bilježiti u sustav. Samim time se na Web aplikaciji mogu spremati prijave koje korisnik radi i zapamtiti uneseni podaci. Uglavnom jednom prihvaćena privola generira prvi kolačić u kojemu se pohranjuje ta potvrda. Sam pregled kolačića je dostupan u postavkama Web preglednika te ih korisnik u bilo kojem trenutku može ugasiti.



Slika 2. Kolačići na stranici FOI-ja (foi.unizg.hr)

Kolačići su postavljeni da traju određeno vrijeme, stoga se ti podaci nakon nekog vremena brišu i zaustavlja se prikupljanje dokle god ponovno korisnik ne odobri prikupljanje podataka. Što se tiče kolačića, Conallen (2003.) je zapazio kako prilikom učitavanja oglasa na Web stranici dolazi do slanja HTTP zahtjeva za te oglase na poslužitelj poduzeća koje je kreiralo oglas, pošto se većina oglasa na Internetu radi kod određenih poduzeća koje su specijalizirane za oglašavanje na Webu. Prema tome prilikom učitavanja oglasa, odnosno slike oglasa, dolazi do slanja HTTP zahtjeva na vanjski poslužitelj, odnosno poslužitelj oglašivačke tvrtke. Uz „pomoć“ tog zahtjeva događa se razmjena kolačića između poduzeća koje je vlasnik

Web stranice, ali i poduzeća koje je vlasnik poslužitelja na kojem se nalazi slika, odnosno informacije o oglasu. Conallen (2003.) tada govori kako se prilikom posjeta većem broju Web stranica može dogoditi slučaj kako se na posjećenim stranicama koriste oglasi istog poduzeća za oglašavanje, prilikom čega to poduzeće ima dosta podataka za profiliranje prema stranicama koje korisnik najčešće posjećuje te tada nastoji prikazati oglase prilagođene korisniku. Kod Web aplikacija postoji i drugi način prikupljanja podataka, a to je uz pomoć sjednica (engl. *session*). Sjednice su prema Calvazara, Focardi, Squarcina i Tempesta (2017.) polutrajne informacije koje se razmjenjuju između Web preglednika i Web poslužitelja, a uključuju višestruke zahtjeve prema Web poslužitelju, ali i odgovore od Web poslužitelja. Njihova uporaba može biti prilikom autentikacije korisnika, kako korisnik ne treba više puta unositi podatke za prijavu ukoliko napusti Web aplikaciju te se kasnije vrati u istu za vrijeme trajanja Web sjednice. Za nju na Web aplikaciji nije potrebna nikakva privola jer nema trajnog zapisa podataka. Sesija se na Web aplikaciji može stvoriti na primjer tijekom prijave korisnika u sustav. Tada se bilježi u bazu podataka da se korisnik prijavio i da je stvorena sjednica.



Slika 3. Pregled sesije u sustavu ELF (elf.foi.hr)

I sjednice i kolačići se stvaraju pomoću programskih jezika. Na projektu iz kolegija „Izgradnja Web aplikacija“ na stručnom studiju „Primjena informacijske tehnologije u poslovanju“, studenti uče kako se kreiraju kolačići i sjednice u programskom jeziku PHP. Također se kolačići i sjednice mogu kreirati pomoću jezika JavaScript. Bitno je shvatiti kako kolačići tijekom kreiranja zaprimaju vrijednosti imena i trajanja, dok se u daljnjem dijelu koda Web aplikacije pohranjuju vrijednosti u same kolačiće. Što se sjednice tiče, ona ima u programskom jeziku PHP svoju superglobalnu varijablu „`$_SESSION[]`“ u koju se pohranjuju vrijednosti kao u asocijativni niz. Asocijativni niz će biti detaljnije objašnjen u daljnjem dijelu rada.

5. Aktivni korisnički dijelovi Web aplikacije

Aktivni korisnički dijelovi prema organizaciji Usability.gov (n.d.) uključuju ulazne kontrole, navigacijske komponente, informacijske komponente i kontejnere. Pod ulazne kontrole se prvotno misli na ulazna polja koja se mogu naći na obrascima poput potvrdnih okvira, radio gumba, padajućih izbornika i sličnih elemenata. Navigacijske komponente su najčešće klizači, mrvice (engl. *breadcrumbs*), tražilice, straničenje (engl. *paginacija*), oznake i ikone. Informacijske komponente uključuju generirane savjete, ikone, trake za napredak, obavijesti, okviri za poruke i modalni prozori. Dok se kod komponenata tipa kontejnera primjenjuje „harmonika“ kako bi se stvorio korisnički dio na Web aplikaciji.

5.1. Vrste korisničkih dijelova Web aplikacije

Prema podjeli koju navodi organizacija Usability.gov (n.d.) postoji osnovna podjela na ulazne kontrole koji se najčešće nalaze u obrascima, nakon toga navigacijske komponente kojima se korisnik koristi kako bi prolazio kroz sadržaj Web aplikacije, također tu se nalaze informacijske komponente koje služe za informiranje korisnika o statusu, porukama i sličnim stavkama koje se odnose na samog korisnika. Na kraju se govori o kontejnerima koji predstavljaju određeni dio Web aplikacije kod kojeg se generira sadržaj za samog korisnika te je najčešće korišteni harmonijski prikaz gdje su postavljeni tabulatori na koje korisnik može kliknuti te mu se prikaže određeni sadržaj.

5.1.1. Ulazne kontrole

Pod ulazne kontrole Usability.gov (n.d.) definira potvrdne okvire koji omogućuju da korisnik može označiti više opcija u određenom kompletu. Radio gumbi su slični kao i potvrdni okviri uz iznimku da omogućuju označavanje pojedine opcije. Padajući izbornici omogućuju korisniku izbor jedne opcije, za razliku od radio gumbi, praktičniji su jer ne zauzimaju mnogo prostora na Web aplikaciji. Okviri s popisima su također slični kao i potvrdni okviri, uz iznimku da su praktičniji za uporabu kod većeg broja odabira. Gumbi definiraju korisnikovu akciju kad bivaju pritisnuti, služe najčešće za objavljivanje navedenih ulaznih elemenata. Prebacivači (engl. *toggles*) služe korisniku kako bi se mogao prebaciti između dva stanja. To je najčešće kod korištenja kada se nešto treba prikazati ili sakriti pritiskom na gumb. Tekstna polja omogućuju korisniku unos teksta i na kraju postoji odabir datuma i vremena gdje korisnik može odabrati određeni datum i vrijeme.

5.1.2. Navigacijske komponente

Navigacijske komponente prema Usability.gov (n.d.) su tražilice pomoću kojih korisnik može pretraživati sadržaj na Web aplikaciji. Mrvice predstavljaju tekstualne oznake koje korisniku upućuju gdje se nalazi, one izgledaju ovako: „Početna/O nama/Povijest“. Straničenje je zapravo lista stranica, korisnik može odabrati na koju stranicu želi prijeći prilikom pregleda sadržaja Web aplikacije. Oznake ili engl. *Tags* predstavljaju određene ključne riječi prema kojima korisnik može filtrirati sadržaj na Web aplikaciji. Klizači su elementi pomoću kojih korisnik može filtrirati određenu vrijednost. Njihov format, odnosno tip podataka je najčešće broj. Ikone su zapravo pojednostavljene slike prema kojima se korisnik može lakše pozicionirati na željenu Web stranicu unutar Web aplikacije, također se u današnje vrijeme ikone koriste za prikaz društvenih mreža vezanih uz Web aplikaciju te služe kao poveznica na određene društvene mreže. Vrtuljak slika, engl. *image carousel*, je inače prikaz slika kojima korisnik upravlja pomoću postavljenih elemenata poput gumba koji su definirani kako bi promijenili trenutnu postavu slika.

5.1.3. Informacijske komponente

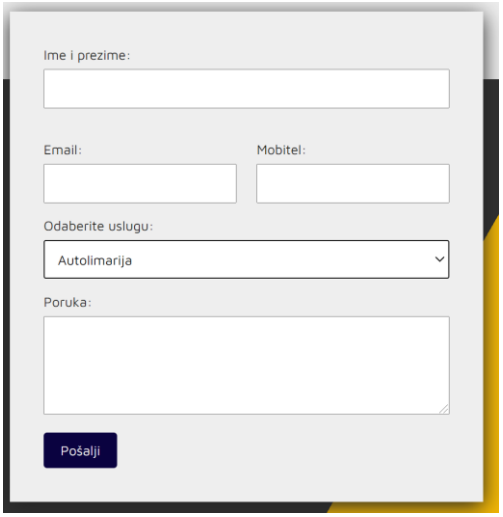
Informacijske komponente prema Usability.gov (n.d.) su najčešće obavijesti koje se generiraju za pojedinog korisnika ovisno o postavljenim vrijednostima. Trake za napredak koje su pojedinačne za svakog korisnika i prikazuju status njegovog pregledavanja sadržaja. Savjeti iliti engl. *tooltips*, su tekstualne poruke koje savjetuju korisnika prilikom pregleda sadržaja. Okviri za poruke se koriste kako bi se korisniku prikazale informacije koje su vezane uz njegovu aktivnost na Web aplikaciji. Na kraju se govori o modalnim prozorima, engl. *pop-up*, koji se otvaraju u zasebnim prozorima kada korisnik pritisne na njihov aktivator.

5.1.4. Kontejneri

Kontejneri su u biti veliki dijelovi Web aplikacije u kojima se nalazi određeni sadržaj. U slučaju kojeg navodi Usability.gov (n.d.) oni su najčešće tipa harmonike gdje korisnik vidi određene oznake na kojima se nalazi opis skrivenog sadržaja te pritiskom na te oznake korisnik može vidjeti sadržaj koji se nalazi pod određenom oznakom. Oni su odlični za preglednost sadržaja jer omogućuju da korisnik sam odabire sadržaj koji želi vidjeli.

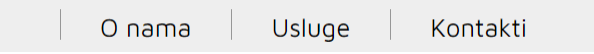
5.2. Specifičnost razvoja aktivnih korisničkih dijelova Web aplikacije

Sam razvoj aktivnih korisničkih dijelova Web aplikacije počiva na pravilnom razmještaju i dizajnu korisničkih dijelova na Web aplikaciji. Prema tome postoje određene pozicije na Web stranici gdje je najbolje smjestiti aktivne korisničke dijelove. Ulazne kontrole se najčešće smještaju unutar obrazaca jer tamo nude najširu primjenu pošto obrasci inače od korisnika očekuju da unese određene podatke i pošalje ih poslužitelju koji će ih kasnije proslijediti na bazu podataka.



Slika 4. Obrazac za kontaktiranje poduzeća (autorski rad)

Na slici iznad je vidljivo kako su na kontakt obrascu zastupljeni elementi poput polja za unos teksta, padajućeg izbornika i gumba za slanje podataka. Navedeni elementi su dostatni za normalnu interakciju i poticanje korisnika na akciju, odnosno postizanje potencijalnih klijenata poduzeću koje ima Web aplikaciju koja koristi obrasce s ulaznim kontrolama. Nakon toga se javljaju navigacijske komponente. One se najčešće smještaju ispod zaglavlja ili ispred podnožja, a predstavljaju glavnu interakciju korisnika s Web aplikacijom. Bez pravilne navigacije, korisnik nije u mogućnosti vidjeti sav sadržaj koji mu aplikacija može ponuditi, stoga je najbitnije da se ne ostave „slijepe ulice“, odnosno Web stranice koje nisu povezane navigacijom.



Slika 5. Izgled navigacije (autorski rad)

Na gornjoj slici je prikazan izgled jednostavne navigacije na jednostraničnoj Web aplikaciji. Sama navigacija je smještena ispod zaglavlja i kako korisnik prolazi Web stranicom tako navigacija ostaje fiksirana na istome mjestu na Web stranici, što je čini veoma

pristupačnom korisniku na korištenje. Informacijske komponente se mogu iskoristiti bilogdje na stranici, pozicija samog elementa ovisi o njegovom sadržaju. Ukoliko je riječ o obavještavanju tada će se element postaviti u zaglavlju kako bi ga korisnik odmah opazio. Ukoliko se radi o savjetima, oni se mogu smjestiti uz sam sadržaj, naravno uz pravilno isticanje od ostatka teksta. Kontejneri inače predstavljaju sam način na koji će se sadržaj prezentirati na Webu stoga je njihovo mjesto rezervirano ispod zaglavlja. Njihova upotreba je ključna ukoliko se želi korisnika potaknuti na čitanje sadržaja kojeg ima podosta, jer na taj način korisnik može odabrati sadržaj koji želi vidjeti čime ga se već u prvu ruku poziva na akciju i omogućuje praćenje njegove aktivnosti jer se upravo taj odabir može pohraniti u jednu od tehnika praćenja aktivnosti, a to su kolačići.

6. JavaScript

„JavaScript je skriptni jezik koji omogućuje poboljšavanje statičkih Web aplikacija pružanjem dinamičnog, personaliziranog i interaktivnog sadržaja.“ (Wilton, 2004.)

Za samo korištenje JavaScripta nije potrebna nikakva skupocjena oprema niti softver, već ga može pokrenuti svatko na bilo kojem Web pregledniku. Za programiranje u JavaScript jeziku također nema potrebe za kupovinom određenog softvera za razvoj koda jer se JavaScript može izvršavati u samom HTML dokumentu, što znači da je za programiranje u JavaScriptu dostatan alat poput Notepad-a.

6.1. Kratka povijest JavaScripta

Rauschmayer (2014.) govori kako je 1994. godine osnovano poduzeće „Netscape“ u Americi s ciljem otkrivanja potencijala Svjetske računalne mreže. Na taj način je nastao Netscape-ov Web pretraživač „Navigator“. Netscape je tada zajedno s poduzećem „Sun“ implementirao programski jezik „Java“ u Navigator. Prema tome je stvorena ideja za razvojem skriptnog jezika koji će se moći implementirati na Webu, te su odlučili stvoriti prototip koji će imati sličnu sintaksu kao i Java. U samo par dana Brendan Eich stvara prototip pod prvim imenom „Mocha“. Daljnjim razvojem dolazi do mnogih verzija samog prototipa JavaScripta, da bi on na početku prosinca 1995. godine postao JavaScript koji se koristi danas pri razvoju Web aplikacija.

Nadalje, Rauschmayer (2014.) spominje kako je Microsoft tada vodio „Web pretraživački rat“ s poduzećem Netscape te je implementirao isti jezik pod drugim imenom JScript u svoj Web pretraživač „Internet pretraživač“ (engl. *Internet Explorer*). Iz tog razloga Netscape pokreće standardizaciju jezika pod nazivom specifikacije ECMA-262 te zbog autorskih prava koje je tvrtka Sun, danas Oracle, imala za naziv JavaScript, naziva isti ECMAScript, ali se taj naziv koristi samo kada se govori o verzijama jezika, dok se za sam naziv jezika dan danas koristi naziv JavaScript.

JavaScript je do 2021. godine imao dvanaest standardizacija od kojih je zadnja prema ECMA International (2021.) završena u srpnju 2021. godine, kada je prošlo dvadeset i četiri godine od objavljivanja prve standardizacije. Jedina standardizacija koja se nije dogodila, jer je odbačena, je četvrta standardizacija.

Dolaskom JavaScripta, počinje se javljati niz tehnologija koje su vezane uz JavaScript, prema tome Rauschmayer (2014.) navodi neke projekte koji su se pojavili nakon otkrića JavaScripta, pa sve do 2014. godine:

- Dinamički HTML – 1997.
- XMLHttpRequest – 1999.
- JSON, format za razmjenu podataka zasnovan na JavaScriptu – 2001.
- Dojo Toolkit, okvir za programiranje u JavaScriptu – 2004.
- Ajax, aplikacija klase radne površine bazirane na Webu – 2005.
- Apache CouchDB, JavaScript-centrirana baza podataka
- jQuery, pomoć pri DOM manipulaciji – 2006.
- WebKit, preuzimanje mobilne Web matice (engl. *mainstream*)
- V8, dokazivanje da JavaScript može biti brz – 2008.
- Node.js, implementacija JavaScripta na poslužitelj – 2009.
- PhoneGap, pisanje izvornih aplikacija (engl. *native apps*) u HTML5 – 2009.
- Chrome OS, čineći Web preglednik operativnim sustavom – 2009.
- Windows 8, prvoklasne (engl. *first-class*) HTML5 aplikacije

6.2. Sintaksa

Što se sintakse JavaScripta tiče, ona je poprilično jednostavna. Postoje osnovne vrijednosti koje se koriste u JavaScriptu, a one su: booleove vrijednosti (engl. *booleans*) koji mogu poprimiti vrijednosti točno/netočno (engl. *true/false*), brojevi koji mogu biti cijeli ili decimalni, znakovni niz (engl. *string*), obični predmeti (engl. *plain objects*) ili polja (engl. *arrays*). Par primjera sintakse JavaScripta:

```
var x;
var y;
x = 3 + y;
//ovo je jednolinijski komentar
/* ovo je višeli-
nijski komentar */
fun(x, y);
obj.metd(1);
if (x === 0) {
     x = 111;
}
function fun(x ,y) {
```

```
    return x + y;
}
```

U navedenim primjerima je prikazano kreiranje varijabli, rad s varijablama, korištenje komentara, pozivanje funkcija, pozivanje metoda za definirane objekte, korištenje instrukcija i definiranje funkcija. O svemu navedenome će se govoriti u daljnjem dijelu rada, počevši s varijablama.

6.3. Varijable

Kada se žele kreirati varijable u JavaScriptu, one se deklariraju na sljedeći način:

```
var x; let y; const z;
```

Prema tome postoje tri načina kreiranja varijabli, prvu uključuje naredbu `var`, drugi naredbu `let`, a treći način je pomoću naredbe `const`. Razlika je u tome što je u početku korištena `var` naredba, a ostale dvije su se pojavile u kasnijim standardizacijama JavaScripta. Razlika je u tome što varijabla deklarirana `let` naredbom prema W3Schools (n.d.) ne može biti ponovno deklarirana, mora biti deklarirana prije korištenja i deklarirana varijabla ima opseg bloka. Varijable deklarirane `const` naredbom se razlikuju prema tome što ne mogu biti ponovno deklarirane, prekomandirane (engl. *reassigned*) i također imaju opseg bloka. Što se tiče imena varijable, ono mora prema W3Schools (n.d.) poštivati nekoliko pravila:

1. Imena mogu koristiti slova, brojeve, podvlačenja i znak dolara
2. Imena moraju početi slovom
3. Imena također mogu započeti s $ i _
4. Imena su osjetljiva na velika i mala slova
5. Rezervirane riječi (na primjer JavaScript ključne riječi (engl. *keywords*)) se ne mogu koristiti kao imena

Jednostavni primjer kako koristiti kreirane varijable u JavaScriptu izgleda ovako:

```
x = 2;
y = 5;
var z = x + y;
```

U navedenome primjeru će se u novostvorenoj varijabli „z“ pridružiti vrijednost zbrajanja brojeva koji su pridruženi pripadajućim varijablama „x“ i „y“. Varijabla može poprimiti vrijednosti: točno/netočno (engl. *true/false*), cijele brojeve, decimalne brojeve, predmete (engl. *objects*), nedefinirano (engl. *undefined*) i nula (engl. *null*). Bitno je napomenuti kako se prilikom definiranja varijabli koristi operator pridruživanja „=“ kojeg je potrebno razlikovati od ostalih operatora.

6.4. Operatori

U JavaScriptu postoji niz operatora koji se koriste, prema tome se mogu razlikovati operator pridruživanja, operatori uspoređivanja, aritmetički operatori, logički operatori, operatori tipa i bitni (engl. *bitwise*) operatori.

Operator pridruživanja je „=“ i on se može naći u prijašnjim primjerima prilikom deklariranja varijabli. Operatori uspoređivanja se najčešće koriste kod instrukcija uvjeta kako bi se prema određenom uvjetu izvršavale definirane naredbe. Oni mogu biti:

Tablica 2. Operatori pridruživanja (prema W3Schools, n.d.)

==	Jednaka vrijednost podataka
===	Jednaka vrijednost i jednak tip podataka
!=	Različita vrijednost
!==	Različita vrijednost i tip podataka
<	Vrijednost „manja od“
>	Vrijednost „veća od“
<=	Vrijednost „manja ili jednaka od“
>=	Vrijednost „veća ili jednaka od“
?	Ternarni (engl. *ternary*) operator

Nadalje se govori o logičkim operatorima koji su „&&“ koje predstavlja logičko „i“, „||“ koje predstavlja logičko „ili“ i „!“ koje predstavlja logičko „ne“. Potom postoje operatori tipa, „typeof“ koji vraća tip varijable te „instanceof“ koji vraća „točno“ ukoliko je objekt slučaj (engl. *instance*) tipa objekta. Te na kraju dolazi do operatora bita koji pretvaraju bilo koji brojčani operand u 32 bitni broj. Rezultat se na kraju vraća u JavaScript broj. Inače JavaScript zapisuje brojeve kao 64 bitne brojeve s pomičnim zarezom, dok se kod upotrebe operatora bita ti brojevi prebacuju u 32 bitne brojeve i nakon završetka bitne operacije, vraćaju se u prvobitan format.

Tablica 3. Operatori bita (prema W3Schools, n.d.)

Operator	Opis
&	„I“

|	„ILI“
~	„NE“
^	„Isključivi ILI“ (engl. *exclusive OR – XOR*)
<<	Zapisuje nulu i lijevi pomak
>>	Zapisuje desni pomak
>>>	Zapisuje nulu i desni pomak

6.5. Tipovi podataka

Prema zadnjem JavaScript, odnosno ECMAScript standardu postoji sveukupno osam tipova podataka. Mozilla Corporation (n.d.) govori o:

- Bool – podaci koji uključuju vrijednosti točno (engl. *true*) ili netočno (engl. *false*)
- Nula (engl. *null*) – posebna riječ koja definira ništavnu vrijednost
- Nedefinirano (engl. *undefined*) – predstavlja svojstvo najviše razine čija vrijednost nije definirana
- Broj – odnosi se na cijeli broj ili na broj s pomičnim zarezom ili točkom. Na primjer broj 3 ili decimalni broj 3.14.
- Veliki broj (engl. *BigInt*) – cijeli broj s arbitrarnom preciznosti, na primjer 314159265359n, gdje „n“ predstavlja kraj cjelobrojne vrijednosti.
- Znakovni niz (engl. *string*) – slijed znakova koji se odnose na tekstualnu vrijednost, poput „Pozdrav“.
- Simbol – predstavlja onaj tip podatka čiji su slučajevi korištenja jedinstveni i nepromjenjive
- Polja (engl. *arrays*) – liste u kojima se pohranjuju vrijednosti, mogu biti indeksirane ili asocijativne
- Objekti – odnosi se zapravo na strukturu podataka i instrukcije za rad s podacima, objekti se mogu smatrati imenovanim kontejnerima za vrijednosti.

Kako kasnije navodi Mozilla Corporation (n.d.) da je JavaScript dinamički jezik, što znači da nije potrebno definirati tip podataka varijable prilikom deklariranja same varijable. U stvarnosti to izgleda ovako:

```
var a = 314;
var rečenica = 'Ovo je rečenica!';
```

U prvome primjeru je prikazano deklariranje varijable „a“ s tipom podatka cijelog broje, dok je drugi primjer deklaracije varijable „rečenica“ koja ima tip podataka znakovnog niza. Upravo zbog toga JavaScript će se uspješno izvršiti bez greške, što nije slučaj kod C++ programskog jezika gdje je prije svega potrebno definirati tip podataka varijable. U C++ jeziku se tip podataka varijable definira na sljedeći način:

```
int a,b;
string rijec;
```

Navedeni primjer se koristi prije definiranja vrijednosti varijabli. U ovome slučaju varijable „a“ i „b“ će biti cjelobrojne, a varijabla „rijec“ će biti tipa znakovnog niza.

6.6. Funkcije i događaji

Kantor (21.06.2021.) govori da su funkcije temeljni blokovi programa, odnosno da one omogućuju pozivanje koda više puta bez potrebe za ponavljanjem. Kako bi se sama funkcija kreirala potrebno je istu deklarirati, u JavaScriptu se to radi na sljedeći način:

```
function poruka() {
     alert('Ovo je poruka!');
}
```

U navedenom kodu se deklarira funkcija s nazivom poruka koje poprima argumente unutar zagrada „()“, argumenti predstavljaju parametre koji se koriste unutar funkcije, sami argumenti tada kod pozivanja funkcije moraju biti definirani željenom vrijednosti. U navedenom primjeru nema argumenata stoga se pozivanjem navedene funkcije prikazuje poruka, odnosno upozorenje (engl. *alert*) koje će na zaslonu otvoriti prozor na kojemu će se ispisati „Ovo je poruka!“. Sintaksa koja se koristi prilikom deklaracije funkcije počinje naredbom funkcija (engl. *function*) nakon čega slijedi imenovanje same funkcije. Unutar zagrada se postavljaju argumenti funkcije nakon čega se unutar vitičastih zagrada „{}“ se unose instrukcije, odnosno naredbe koje će se izvršavati pozivanjem funkcije. Prilikom završavanja svake linije naredbe potrebno je na kraju staviti znak za parsiranje/sintaksnu analizu „;“. U konačnici deklariranje funkcije i samo pozivanje funkcije u JavaScriptu izgleda otprilike ovako:

```
function poruka () {
     alert('Ovo je poruka!');
}
poruka();
```

Zadnja linija koda prikazuje pozivanje funkcije tek nakon njezine deklaracije. Funkcija se može pozvati više puta unutar istog programa. Kod samih funkcija je bitno spomenuti kako

se tada pojavljuju pojmovi poput globalnih i lokalnih varijabli. Kantor (21.06.2021.) navodi kako su lokalne varijable one varijable koje se deklariraju i koriste samo unutar funkcije, dok su globalne varijable one koje su definirane izvan funkcija, ali funkcije koriste iste. Glavna razlika između globalnih i lokalnih varijabli je u tome što se lokalne varijable koriste samo unutar funkcija i izvan funkcija se iste ne mogu pozivati. Kod globalnih varijabli je slučaj da se one deklariraju izvan funkcija i instrukcija, prema tome mogu biti pozvane u istima te tako neće dolaziti do greške prilikom izvođenja programa. Kako to izgleda, pokazuje primjer:

```
function prikazPoruke () {
     let poruka = 'Pozdrav, moje ime je Antonio!';

     alert(poruka);
}
prikazPoruke();
alert(poruka);
```

Navedeni primjer prikazuje definiranje lokalne varijable „poruka“ unutar funkcije, pozivanje funkcije će se prikazati vrijednost unesena u varijablu poruka. Na kraju će program izbaciti grešku, zbog toga što se koristi ugrađena funkcija „alert(poruka);“ koja poziva varijablu poruka, ali ista je deklarirana unutar funkcije što znači da ne postoji na globalnoj razini programa. Što se tiče globalnih varijabli, na primjeru to izgleda ovako:

```
let ime = 'Antonio';
function prikazPoruke () {
     let poruka = 'Pozdrav, moje ime je ' + ime;
     alert(poruka);
}
alert(ime);
```

U ovome primjeru se izvan funkcije deklarira varijabla „ime“ koja se koristi unutar same funkcije na način da se vrijednost varijable „poruka“ povezuje s varijablom „ime“, prema tome će program ispisati poruku „Pozdrav, moje ime je Antonio“. Također na kraju neće doći do greške jer se koristi globalna varijabla, te će program ispisati „Antonio“.

Što se samih argumenata tiče oni se mogu objasniti ovako:

```
function prikazPoruke (ime, poruka) {
alert (ime + ': ' + poruka);
}
prikazPoruke('Antonio', 'Pozdrav!');
```

Argumenti unutar funkcije predstavljaju parametre bez vrijednosti, koji tek kada se funkcija poziva poprimaju određene vrijednosti koje se izvršavaju unutar funkcije. Prema tome će se program prikazati sljedeće rješenje: „Antonio: Pozdrav!“. Navedeni argumenti također

mogu poprimiti fiksne vrijednosti koje je potrebno definirati kod samih argumenata, na primjer bi to izgledalo ovako: `function prikazPoruke (ime, poruka = 'Pozdrav!')… .` Ostatak koda ostaje isti te daje isti rezultat. Kod funkcija se također može koristiti povratak (engl. *return*) što, kako govori Kantor (21.06.2021.), može biti bilo gdje unutar funkcije. Kada se izvršava linija koda s naredbom povratka, funkcija se zaustavlja i vrijednost se vraća u kod pozivanja. To u primjeru izgleda ovako:

```
function provjeriGodine (godine) {
 if (godine <= 18) {
     return confirm('Nemate dovoljno godina za obavljanje kupovine putem
Interneta!');
} else {
     return true;
     }
}
let godine = prompt('Koliko imate godina?', 22);

if (provjeriGodine (godine)) {
     alert('Pristup odbijen!');
} else {
     Alert('Pristup dopušten!');
}
```

U primjeru se deklarira funkcija koja provjerava godine. U samoj funkciji se nalazi instrukcija „if“ koja provjerava ispunjava li argument „godine“ definirani uvjet. Ukoliko je uvjet ispunjen, petlja vraća vrijednost potvrde, dok kada uvjet nije ispunjen, vraća se vrijednost točno. Nakon toga se globalnoj varijabli pridružuje vrijednost pomoću funkcije „prompt“ koja omogućuje unos korisnika, te se poziva funkcija unutar petlje. Ukoliko je u funkciji uvjet ispunjen, program prikazuje obavijest kako je pristup odbijen, a u drugome slučaju kako je pristup dopušten. Pošto se same funkcije vežu s akcijama, iste se mogu imenovati upotrebom glagola. Prema tome Kantor (21.06.2021.) navodi kako se koristi praksa imenovanja funkcija na način da se u verbalnom prefiksu koristi „grubi“ opis akcije koje funkcija izvršava. Na taj način ime funkcije može započeti s:

- „get“ – vraća vrijednost
- „calc“ – izračunava vrijednost
- „create“ – kreira vrijednost
- „check“ – provjerava vrijednost

Nakon funkcija, uz JavaScript se najviše vežu događaji (engl. *events*). Događaji su prema W3Schools (n.d.) „stvari“ koje se događaju unutar HTML dokumenata. To su događaji koji se vežu uz ponašanje preglednika ili akcije korisnika. Najčešće korišteni događaji su:

Tablica 4. Najčešći događaji (prema W3Schools, n.d.)

Događaj	Opis
Na promjenu (engl. *onchange*)	Promjena sadržaja na HTML dokumentu
Na pritisak (engl. *onclick*)	Korisnikov pritisak tipke miša na određeni HTML element
Na prijelaz miša preko (engl. *onmouseover*)	Korisnik je pomaknuo miš preko HTML elementa
Na prijelaz miša izvan (engl. *onmouseout*)	Korisnik je pomaknuo miš izvan HTML elementa
Na pritisak tipke (engl. *onkeydown*)	Korisnik je pritisnuo tipku na tipkovnici
Na učitavanje (engl. *onload*)	Preglednik je završio s učitavanjem stranice

Najčešća upotreba događaja se veže uz HTML element gumba (engl. button) te se to izvodi prema primjeru: <button onclick='prikaziLozinku()'>Prikaži lozinku</button>. U primjeru se pritiskom na gumb izvršava funkcija „prikaziLozinku()“ koja će prikazati lozinku prilikom prijave na Web aplikaciju.

6.7. Objekti

Objekti su, kako o tome govori Mozilla Corporation (n.d.) kolekcije svojstava i svojstvo je zapravo asocijacija između naziva/ključa i vrijednosti. Postoji slučaj kada je svojstvo objekta funkcija i tada se taj objekt naziva metodom. Objekti u JavaScriptu se najčešće vežu uz stvari iz realnog svijeta. Objekt prema tome može biti, na primjer auto, kuća, jahta i slično. Svaki objekt se veže uz pojedino svojstvo koje se definira prilikom njegove deklaracije u kodu. Svojstvo opisuje objekt, odnosno u realnom svijetu to može biti boja, dizajn, mjerna jedinica i slično. Postoje dva načina na koje se mogu kreirati objekti. Prvi način izgleda ovako:

```
var mojaJahta = new Object();
mojaJahta.marka = 'Prestige 500';
mojaJahta.boja = 'Bijela';
```

```
mojaJahta.godina = '2015';
```

U navedenome primjeru se objekt definira pomoću funkcije novi objekt (engl. *new Object*) te se svako svojstvo definira zasebno uz pravilo da se prvo stavi naziv objekta, potom dolazi interpunkcijski znak „.“ te se pridružuje vrijednost za određeno svojstvo. Drugi način kreiranja objekata izgleda ovako:

```
var mojaJahta = {
     marka = 'Prestige 500',
     boja = 'Bijela',
     godina = '2015'
}
```

Drugi način ne koristi funkciju kreiranja novog objekta već se pomoću vitičastih zagrada u JavaScriptu direktno pridružuju svojstva s vrijednostima za objekt. Svako svojstvo se odvaja zarezom. Bitno je spomenuti kako se objekti mogu smatrati asocijativnim poljima zbog toga što se svojstva mogu definirati na način da se u kodu kreira linija: `mojaJahta['marka'] = 'Prestige 500';` . Nakon što su se definirala svojstva, od ECMAScript 5 standarda postoji nekoliko načina na koje se može iterirati kroz sva svojstva objekta:

1. For…in petlje
2. Object keys
3. Objekt.getOwnPropertyNames(o)

Prije ECMAScript 5 standarda su se svojstva objekta izlistavala pomoću funkcije koja bi sadržava petlju koja bi prošla kroz sve vrijednosti svojstava, vraćala kao rezultat te vrijednosti te ih ispisivala na samome kraju. Mozilla Corporation (n.d.) također govori kako se objekti mogu kreirati i pomoću konstrukcijskih funkcija (engl. *constructor function*), što se može vidjeti na sljedećem primjeru:

```
function Jahta (marka, boja, godina) {
     this.marka = marka;
     this.boja = boja;
     this.godina = godina;
}
var mojaJahta = new Jahta ('Prestige 500', 'Bijela', '2015');
```

Ovdje je vidljivo kako se koristi „this“ kako bi se pridružile vrijednosti za objekt. Te se prema tome kreira objekt „mojaJahta“ pomoću funkcije „Jahta“. Prema tome je moguće kreirati više objekata koji koriste funkciju za kreiranje s definiranim parametrima. Također se objekti mogu kreirati pomoću metode „Object create“.

Prema Mozilla Corporation (n.d.), metoda je funkcija koja se povezuje s objektom ili se metoda može smatrati svojstvom objekta koje je zapravo funkcija. Metode se definiraju na isti način kako se i definiraju funkcije u JavaScriptu. Kako se kreiraju metode, najbolje je prikazati na primjeru:

```
var mojaJahta = {
     Jahta: function(marka, boja, godina) {
     // … kod funkcije … //
     }
}
mojaJahta.Jahta(marka, boja, godina);
```

Prvo se kreira objekt nakon čega slijedi definiranje metode kao svojstva, u primjeru je definirana metoda „Jahta“ koja bi imala vrijednost funkcije koja je navedena u prijašnjem primjeru. Nakon toga se poziva metoda na način da se prvo ispiše ime objekta, potom dolazi interpunkcijski znak „.“ pa naziv metode i na kraju parametri koji se vežu uz metodu, odnosno funkciju. Postoje unaprijed definirane metode u JavaScriptu, a to su „get“ i „set“. „Get“ je metoda koja dohvaća vrijednost određenog svojstva, a „set“ je metoda koja definira vrijednost određenog svojstva. Sama svojstva objekta se mogu i izbrisati pomoću funkcije „delete“ te to u konačnici izgleda ovako: `delete mojaJahta.godina;` . Objekte je također moguće i uspoređivati na način da se provjeravaju postoje li slične ili iste vrijednosti u dva objekta.

6.8. Kontrola toka programa

U JavaScriptu se pod kontrolom toka smatraju instrukcije koje omogućuju veću interaktivnost kod samog programa. Najčešće korištene instrukcije za ostvarivanje željene akcije na Web aplikaciji su prema W3Schools (n.d.):

- „if“
- „else“
- „else if“
- „switch“

Sintaksa za „if“ instrukciju izgleda ovako:

```
if (uvjet) {
      // blok naredbi //
}
```

Za iskaz „else“ vrijedi sljedeća sintaksa:

```
if (uvjet) {
      // blok naredbi //
} else {
      // blok naredbi //
}
```

Za iskaz „else if“ se veže sljedeća sintaksa:

```
if (uvjet#1) {
      // blok naredbi //
      } else if (uvjet#2) {
            // blok naredbi //
      } else {
            // blok naredbi //
      }
```

Na kraju se govori o „switch-u“ čija je sintaksa u JavaScriptu izvedena na sljedeći način:

```
switch (izraz) {
     case x:
            // blok naredbi //
            break;
     case y:
            // blok naredbi //
            break;
     case z:
            // blok naredbi //
            break;
     default:
            // blok naredbi //
}
```

U navedenoj sintaksi je moguće vidjeti sve elemente od kojih se sastoji instrukcija „switch“. Ona se najviše koristi u scenarijima kada imamo više od tri slučaja koji se mogu za pojedini izraz ostvariti. Imajući to na umu, definira se svaki pojedini slučaj koji predstavlja

rezultat izraza i izvršava blok naredbi vezan uz taj uvjet. Također se postavlja naredba „break“ koja izlazi iz instrukcije nakon ispunjenja uvjeta. Može se i postaviti predefinirana vrijednost u slučaju da niti jedan slučaj ne odgovara željenom ponašanju korisnika na Web aplikaciji, prema tome se definira zadani slučaj (engl. *default*) koji se izvršava neovisno o korisnikovoj interakciji na Web aplikaciji.

6.9. Petlje i iteracije

U JavaScript jeziku se mogu koristiti petlje pomoću kojih se može više puta izvoditi određeni blok koda. Prema tome postoje petlje:

- „for“
- „do…while“
- „while“
- „labeled“ izjava
- „break“ izjava
- „continue“ izjava
- „for…in“
- „for…of“.

W3Schools (n.d.) navodi sintakse za pojedine petlje. Prva je sintaksa za petlju „for“:

```
for (izjava1; izjava2; izjava3) {
     // blok naredbi //
}
```

U navedenoj sintaksi za petlju je prikazano kako se ona navodi u kodu i koji su njene izjava, odnosno uvjeti i dokle god su izjave istinite blok naredbi će se izvršavati, to jest petlja će se izvoditi. Pomoću „for“ petlje se u najbanalnijem primjeru može kreirati brojčanik koji će uvećavati vrijednost cjelobrojne varijable za 1. Nakon toga dolazi se do „do…while“ petlje. Ono što je najbitnije kod ove petlje, je to da će se ona izvesti barem jednom, ali će se izvršavati onoliko puta dok je uvjet točan. To se najbolje može objasniti pomoću njene sintakse:

```
do {
 // blok naredbi //
} while (uvjet);
```

Kao što se u sintaksi može vidjeti, blok naredbi će se izvršiti jednom i onda će se izvršavati dokle god je uvjet petlje istinit. Nešto drugačiji slučaj se javlja kod „while“ petlje. Kod

nje je situacija malo drugačija, jer se blok naredbi neće izvršiti ukoliko uvjet nije točan, odnosno istinit, što je i vidljivo prema sintaksi:

```
while (izjava) {
     // blok naredbi //
}
```

„Labeled statement“, kako ih opisuje Mozilla Corporation (n.d.), su izjave koje se mogu koristiti više puta u kodu pomoću oznake. One se mogu definirati na sljedeći jednostavan način:

```
label:
     izjava
```

Izjava „break“ i izjava „continue“ su izjave koje se koriste prilikom upravljanja s petljama. One služe za prekidanje odnosno nastavljanje izvođenja koda u petlji te je njihova namjena jednostavna. „For…in“ petlja se najčešće koristi za polja i objekte, kada se želi proći kroz sve vrijednosti polja, odnosno objekta. Njezina sintaksa je:

```
for (ključ in objekt) {
     // blok naredbi //
}
```

Za polje je ista sintaksa, jedino se mora definirati varijabla polja i ime samog polja. Potom se dolazi do „for...of“ petlje koja je slična kao i prethodna izjava s jedinom iznimkom da se koristi samo za iterabilne objekte u programu (poput polja, mapa, argumenata, objekta i slično), njezina sintaksa je slična prethodnoj, a izgleda ovako:

```
for (varijabla of iteracijskiObjekt) {
     // blok naredbi //
}
```

Ukratko, s objašnjenim poglavljima dobiva se osnova i uviđaj u logiku samog JavaScripta te se može prijeći na programiranje s istim. Samim time će se na Web aplikacijama ostvariti veća interaktivnost i omogućiti bolje korisničko iskustvo te povećati broj posjetitelja na određenoj Web aplikaciji što je i cilj prilikom izrade Web aplikacije.

7. CoffeeScript

Odmakom godina i razvojem JavaScripta, to jest objavljivanjem ECMAScript standarda, javljaju se mnogi okviri koji se baziraju na JavaScript jeziku, a samim time javlja se i mali jezik koji se kompilira u JavaScriptu, CoffeeScript. MacCaw (2012.) govori o tome kako je sama sintaksa CoffeeScripta temeljena na programskim jezicima Ruby i Python. Pomoću CoffeeScripta se nastoji olakšati implementacija interaktivnosti na Web aplikacijama, a uz to i smanjiti opseg koda koji se mora unijeti kako bi se određena funkcionalnost izvršila u aplikaciji. U konačnici CoffeeScript je zapravo JavaScript.

7.1. Povijest CoffeeScripta

Kako je nastao CoffeeScript? Prema originalnom GitHub repozitoriju autora Jeremyja Ashkenasa (04.05.2018.), vidljivo je kako je prvi puta stvoren repozitorij 13.12.2009. pod nazivom „inicijalna predaja tajanstvenog jezika“. Već 24.12. iste godine, autor Jeremy Ashkenas izbacuje prvu dokumentiranu i označenu verziju CoffeeScripta „0.1.0“. Samim time je započeo razvoj CoffeeScripta koji je već u veljači 2010. godine dobio verziju „0.5“, koja je zamijenila tada korišteni Ruby za kompiliranje sa samo-poslužiteljskom verzijom „čistog“ JavaScripta. CoffeeScript nakon prve stabilne verzije „1.0.0“, dobiva verziju „2.0.0“ koja je predstavljena 18.09.2017. godine na službenoj stranici „CoffeeScript.org“. Trenutna verzija CoffeeScripta koja je dostupna za korištenje, je verzija „2.5.1“.

7.2. Instalacija i upotreba CoffeeScripta

Instalacija je prilično jednostavna, za provođenje iste potrebno je imati preduvjet da je instalirana aplikacija Node.js, pomoću koje se mogu instalirati razni paketi ili dodaci pomoću kojih se mogu izrađivati projekti. Nakon što računalo ima Node.js verziju 6 ili veću, može započeti s instalacijom CoffeeScripta. Sama instalacija se provodi u „Windows“ naredbenom retku (engl. *Windows Command prompt*) i to na način da se unese naredba:

```
npm install –global coffeescript
```

Ova naredba će pokrenuti instalaciju CoffeeScripta pomoću Node.js programa, te će instalirati CoffeeScript globalno čime će se omogućiti dostupnost naredbi CoffeeScripta na globalnoj razini. Drugi način za instalaciju, je lokalna instalacija u mapu projekta. Na taj način će se verzija CoffeeScripta pratiti u ovisnostima (engl. *dependencies*) projekta. Prije same lokalne instalacije, se pomoću naredbe „cd“ potrebno prebaciti u mapu projekta i unijeti naredba:

```
npm install –save-dev coffeescript
```

Prema tome će se naredbe CoffeeScripta prvo tražiti u mapi projekta. Nakon uspješne instalacije je moguće koristiti „coffee“ naredbe koje mogu pokretati skripte, kompilirati „.coffee“ datoteke u „.js“. Naredba „coffee“ ima nekoliko opcija:

Tablica 5. Opcije naredbe "coffee" s opisima (prema "*CoffeeScript*“, 03.07.2020)

Opcija	Opis
-c, --compile	Kompilira „.coffee“ skriptu u „.js“ JavaScript datoteku
-t, --transpile	Prevodi CoffeeScriptov rezultat kompiliranja kroz „Babel“ prije spremanja ili pokretanja generiranog JavaScripta. Prije izvršavanja opcije potrebno je imati instaliran Babel.
-m, --map	Generira izvorne karte (engl. *source maps*) zajedno s kompiliranim JavaScript datotekama, uz to dodaje „sourceMappingURL“ direktive u datoteke.
-M, --inline-map	Slično kao i „--map“ uključuje izvornu kartu unutar JavaScript datoteke, bez potrebe stvaranja druge datoteke.
-i, --interactive	Pokreće interaktivnu sjednicu CoffeeScripta u svrhu skraćivanja isječaka (engl. *snippets*). Slično kao i pozivanje „coffee“ bez argumenata.
-o, --output [DIR]	Ispisuje sve kompilirane JavaScript datoteke u određeni direktorij. Koristi se zajedno s „--compile“ i „--watch“.
-w, --watch	Promatra promjene u datotekama, izvršava određenu naredbu dokle god se datoteka ažurira.
-p, --print	Umjesto da se JavaScript ispisuje kao datoteka, ispisuje se direktno u „stdout“.
-s, --stdio	Cijev (engl. *pipe*) u CoffeeScriptu do „STDIN“ i nazad do JavaScripta preko „STDOUT“.

	Dobro za procese pisane u drugim programskim jezicima.
-l, --literate	Izvršava kod kao doslovan CoffeeScript. Koristi se jedino kada se kod prosljeđuje direktno preko „stdio“ ili kada se koristi neka vrsta datoteke bez ekstenzije.
-e, --eval	Kompilira i ispisuje mali isječak CoffeeScripta direktno iz naredbenog retka.
-r, --require [MODULE]	„require()“ dohvaća određeni modul prije pokretanja „REPL“ ili evaluira kod s definiranom „--eval“ zastavicom (engl. *flag*).
-b, --bare	Kompilira JavaScript bez funkcije sigurnosnog omotavanja visoke razine (engl. *top-level function safety wrapper*).
--no-header	Potišče „Generirano CoffeeScriptom“ (engl. *Generated by CoffeeScript*) zaglavlje.
--nodejs	„node“ izvršna naredba ima nekoliko korisnik opcija poput „--debug“, „--debug-brk“, „--max-stack-size“ i „--expose-gc“. Ova zastavica se koristi prilikom slanja opcija direktno u „Node.js“
--ast	Generira apstraktno-sintaksno drvo čvorova (engl. *apstract syntax tree of nodes*) CoffeeScripta. Koristi se prilikom integriranja s JavaScript alatima za kreiranje.
--tokens	Umjesto izvršavanja CoffeeScripta, ispisuje tok tokena. Koristi se za otklanjanje pogrešaka (engl. *debugging*). prilikom kompiliranja.
-n, --nodes	Umjesto izvršavanja CoffeeScripta, ispisuje drvo izvršavanja (engl. *parse tree*). Koristi se za otklanjanje pogrešaka prilikom kompiliranja koda.

7.3. Sintaksa

Prema MacCaw (2012.), sintaksa CoffeeScripta je identična sintaksi JavaScripta, ali nije preuzeta u cjelosti, stoga neke ključne riječi (engl. *keywords*) koje rade u JavaScriptu, neće raditi u CoffeeScriptu. Ukoliko se programira CoffeeScript datoteka, onda se mora programirati u čistom CoffeeScriptu te se ne smije događati da se programira u drugim jezicima u istom programu. U samome CoffeeScript sintaksi ne postoji interpunkcijski znak „;“, koji je prisutan u JavaScriptu, već se oni dodavaju prilikom kompiliranja. Komentari u CoffeeScriptu se dodavaju na isti način kao i komentari u Ruby programskom jeziku i to na sljedeći način:

```
# Ovo je komentar u CoffeeScriptu
```

CoffeeScript za komentiranje koristi znak „#“ koji se koristi kako bi se komentirala pojedina linija koda. Ukoliko se želi komentirati više linija koda, onda se to radi na sljedeći način:

```
###
     Ovo je
     višelinijski komentar
###
```

U CoffeeScript sintaksi najviše dolazi do izražaja, takozvani „bijeli prostor“ (engl. *whitespace*) koji se u praksi koristi kako bi se na primjer zamijenile vitičaste zagrade „{}“ s tipkom tabulatora („TAB“). Ovakav način korištenja praznina je preuzet iz Pythonove sintakse i služi kako bi skripta bila formatirana na razuman inače se kompiliranje neće moći izvršiti.

7.4. Varijable

CoffeeScript rješava jedan problem JavaScripta, a to je deklariranje globalnih varijabli. Prema MacCaw (2012.), kada se slučajno zaboravi unijeti „var“ prilikom deklaracije varijable, ta ista varijabla postane globalna. U CoffeeScriptu je to riješeno na način da se automatski pomoću ugrađene funkcije tijekom kompiliranja sve varijable deklariraju s „var“ naredbom u JavaScriptu. To bi na primjer izgledalo ovako:

CoffeeScript:

```
mojaVarijabla = „CoffeeScript“
```

Kompilirano u JavaScript:

```
var mojaVarijabla = „CoffeeScript“;
```

Prema tome je nemoguće slučajno kreirati globalnu varijablu, kako je to slučaj bio u JavaScriptu, ali nekad je korisno kreirati globalne varijable. To je moguće na način da se kreira objekt sa određenim svojstvima te bi to izgledalo ovako:

CoffeeScript:

```
auto = this
auto.marka = „Volvo“
```

Kompilirano u JavaScript:

```
var auto;
auto = this;
auto.marka = „Volvo“;
```

Što se tiče tipova podataka, CoffeeScript koristi iste tipove podataka kao i JavaScript koji su navedeni u prijašnjem poglavlju.

7.5. Operatori

CoffeeScript sadržava mnogo operatora koji funkcioniraju kao i operatori navedeni u JavaScriptu. Prema Lee (2014.) to su: „+, -, !, /, %, <, >, <=, >=, ., &&, ||, *“. Ternarni operatori koji se koriste u JavaScriptu, u CoffeeScriptu su ili drugačiji ili ne postoje. Prvi bitan operator koji se koristi u CoffeeScriptu je operator pridruživanja „=“. On se koristi kako bi se određenoj varijabli pridružila vrijednost. To bi u kodu izgledalo ovako:

```
mojeIme = 'Antonio'
```

Nakon toga se mogu spomenuti logički operatori. Kod logičkih operatora moguće je korištenja aliasa iliti pseudonima. Prema tome logički operator „ne“ se može navesti kao „!“ ili „not“. Logički operatori u CoffeeScriptu daju rezultat točno/netočno. Nadalje se govori o operatoru „i“ i „ili“ koji u CoffeeScriptu izgledaju ovako „&&“ i „||“ ili „and“ i „or“. Također prema Lee (2014.), CoffeeScript ima netočne/lažne vrijednosti. To su inače vrijednosti „null“ (engl. *nula*), netočno, '', nedefinirano i 0. One će uvijek davati vrijednost netočno kao rezultat operacije. Sve ostale vrijednosti u CoffeeScriptu imaju vrijednost točno/istina. Nakon toga dolaze operatori jednakosti. To su operatori „==“ i „!=“, odnosno jednako i različito. Njihovi pseudonimi su „je“ (engl. *is*) i „nije“ (*engl. isnt*). U kodu bi njihova upotreba izgledala ovako:

```
mojeIme = 'Antonio'
mojePrezime = 'Pavlović'
mojeIme is mojePrezime
# false
```

U navedenom kodu je vidljivo kako su postavljene dvije varijable, te su iste uspoređene. Kako vrijednosti tih varijabli nisu iste, rezultat operacije će biti netočno. Pri uporabi pseudonima treba pripaziti „isnt“ i „is not“ nisu iste stvari u CoffeeScriptu. Prvo znači da varijable nisu iste, dok drugo znači da su varijable iste, ali druga varijabla se stavlja u negaciju. Navedeni operatori su ekvivalentni JavaScript operatorima „===“ i „!==“, strogim operatorima

uspoređivanja koji uspoređuju podatke i tipove podataka. Potom se dolazi do operatora zbrajanja i oduzimanja, vrlo jednostavno, to su operatori: „+“ i „-„. Preporuča se da se isti koriste samo s brojevima jer kod operacija sa znakovnim nizovima, dolazi do spajanja vrijednosti, prema tome:

```
5 + 5
# 10
2 + '8'
# '28'
```

Vidljivo je kako u prvom primjeru se događa zbrajanje brojeva stoga je i rezultat zbroj, ali kod drugog primjer CoffeeScript spaja znakovni niz s brojem, stoga nastaje novi znakovni niz koji je spoj dviju vrijednosti. Oduzimanje se izvodi na isti način, također samo s brojevima, ali kada se događa da se spajaju broj i znakovni niz nastaje greška „nije broj“ (engl. *not a number, NaN*). Ostali aritmetički operatori su množenje, dijeljenje i modularno dijeljenje, odnosno „*“, „/“, „%“. Sa svim aritmetičkim operatorima se preporuča da se koriste samo s brojevima u CoffeeScriptu, inače može doći do potencijalnih problema u rezultatu operacija. Operatori uspoređivanja, uz već spomenute su također i više, manje, više-jednako i manje-jednako, inače u kodu pisani kao „>“ „<“, „>=“ i „<=“.

7.6. Funkcije i objekti

Kreiranje i pozivanje funkcija u CoffeeScriptu je pojednostavljeno i smanjuje za polovicu opseg koda kojeg je inače potrebno napisati u JavaScriptu za definiranje funkcije. Prema Hoigaard (2012.) funkcija se može vrlo jednostavno pozvati te najjednostavniji primjer za definiranje funkcije u CoffeeScriptu izgleda ovako:

```
zbrajanje = (a, b) -> a + b
zbrajanje 1, 1
# 2
```

Prema primjeru definirana je funkcija koja se zove zbrajanje, argumenti funkcije su „a“ i „b“ te se oni zbrajaju. Nakon toga se poziva funkcija „zbrajanje“ i argumentima se daju vrijednosti „1“ i „1“. Funkcija se izvršava i daje rezultat, u ovome slučaju „2“. Unutar samih funkcija se mogu koristiti petlje i instrukcije kako bi se ostvario željeni rezultat koji funkcija mora izvršavati. One u suštini služe za slučajeve kada je potrebno ponovno iskoristiti određene dijelove koda koji izvršavaju određene operacije ili iteracije kako bi se postigao zamišljeni tok ili rezultat programa.

Objekti u CoffeeScriptu su također jednostavniji za kreiranje. Prema „*CoffeeScript.*“ (2020.), objekti se mogu kreirati korištenjem uvlaka (engl. *indent*) umjesto korištenja

eksplicitnih zagrada, slično se koristi u YAML-u, standardu za jednostavnu serijalizaciju podataka za sve programske jezike. Na sličan način se mogu definirati i polja, kako to izgleda u samoj praksi, vidljivo je na sljedećem primjeru:

```
more = [„Jadransko", „Sredozemno", „Crno", „modro" ]
ribe = {Riječna: „Pastrva", Morska: „Tuna"}
listaVjetrova = [
     NW, N, NE
     W, 0, E
     SW, S, SE
]
brod =
     jahta:
             ime: „Prestige 500"
             godina: 2015
     pasara:
             ime: „Ribar 480"
             godina: 2021
```

U navedenome primjeru je prikazano kako se definiraju objekti i polja. Kod objekata je zanimljivo što CoffeeScript nudi „prečac" prilikom kreiranja istih. U kodu se taj prečac izvodi na sljedeći način:

```
ime = „Prestige 500"
godina = 2015
boja = „bijela"
brod = {ime, godina, boja}
```

Program će prvo imati definirane varijable koje imaju vrijednosti vezane za željeni objekt. Nakon toga će se deklarirati objekt i pridružiti će mu se vrijednosti unutar vitičastih zagrada. Na taj način će se dobiti objekt s definiranim svojstvima. Ovaj način deklariranja objekata je odličan ukoliko su svojstva varijabilna i fiksna, odnosno ne ponavljaju se u ostalim objektima, ali se mijenjaju ovisno o ponašanju korisnika na Web aplikaciji.

7.7. Kontrola toka programa

U CoffeeScriptu je sintaksa za instrukcije i petlje za kontrolu toka programa svedena na minimum. Prema *CoffeeScript* (2020.) „if" i „else" imaju sintaksu gdje se ne koriste zagrade. CoffeeScript može kompilirati „ako" instrukcije u JavaScript izjave koristeći ternarne operatore kada je to moguće, dok u samome CoffeeScriptu nema ternarnih operatora već se koristi jednostavna naredba „ako" kao što se može vidjeti u primjeru:

CoffeeScript:

```
mojeIme = 'Antonio'
if mojeIme == 'Antonio'
     prikaziPoruku()
else
     prikaziObavijest()
```

Kompilirano u JavaScript:

```
var mojeIme;
mojeIme = 'Antonio';
if (mojeIme == 'Antonio') {
     prikaziPoruku();
} else {
     prikaziObavijest();
}
```

Primjer prikazuje deklaracije varijable „mojeIme“ i pridruživanje vrijednosti navedenoj varijabli. Nakon toga se poziva instrukcija „if“ čiji je uvjet da vrijednost varijable mora biti znakovni niz „Antonio“, ako je uvjet istinit poziva se funkcija „prikaziPoruku“, ukoliko uvjet nije istinit poziva se funkcija „prikaziObavijest“. Obje funkcije su definirane od strane programera. Kada se kod kompilira u JavaScript, vidljivo je kako CoffeeScript svojom sintaksom zaista smanjuje veličinu koda potrebnog za izvršavanje instrukcija u programu. Prema *CoffeeScript* (2020.) većina petlji se u CoffeeScriptu piše na način da se koristi obuhvaćanje (engl. *comprehension*) polja, objekata i raspona. Obuhvaćanja zamjenjuju i kompiliraju se kao „for“ petlje koje sadržavaju zaštitne klauzule i vrijednost trenutnog indeksa određenog polja. Za razliku od „for“ petlji, obuhvaćanja su izjave te kao takve mogu biti vraćene i dodijeljene. Primjer kako se definiraju petlje u CoffeeScriptu izgleda ovako:

```
brojac = (broj for broj in [10..1])
```

Kompilirano u JavaScript

```
var brojac, broj;
brojac = (function () {
     var i, rezultat;
     rezultat = [];
     for (broj = i = 10; i >= 1; broj = --i) {
            rezultat.push(broj);
     }
     return rezultat;
})();
```

Navedeni primjer će pomoću for petlje kreirati brojač koji će brojati od 10 prema 1, vrijednost brojača će se zapisivati u polje „rezultat“. CoffeeScript rješava veći dio problema koji se može vidjeti kod JavaScripta, poput deklariranja varijabli prije pridruživanja vrijednosti i slično. Također se obuhvaćanje može koristiti za iteraciju indeksa/vrijednosti kod objekata. Koristi se „od“ (engl. *of*) kako bi se označilo obuhvaćanje svojstava objekata, umjesto obuhvaćanja vrijednosti polja. Jedina petlja niže razine koju upotrebljava CoffeeScript je „while“ petlja uz razliku da u CoffeeScriptu, ona može biti izjava koja vraća polje s rezultatima svake iteracije kroz petlju. Što se tiče „switch/case“ instrukcije, ona se u CoffeeScriptu izvodi na sljedeći način:

```
switch godisnjeDoba
     when „Proljeće“ then ugodno()
     when „Ljeto“ then vruce()
     when „Jesen“ then kisovito()
     when „Zima“ then hladno()
```

Prema primjeru se promatra ponašanje varijable „godisnjeDoba“ te za svako njezino stanje se definira linija koda koja započinje s „when“ nakon čega slijedi slučaj, odnosno moguća vrijednost varijable. Nakon definiranog slučaja započinje s naredbom „then“ definiranje bloka naredbi koja će se izvršavati. U primjeru će se izvoditi određena funkcija, ovisno o stanju varijable.

7.8. Klase

Kako o klasama govori MacCaw (2012.) da su one poprilično korisne u JavaScriptu i da CoffeeScript pruža apstraktno korištenje istih. Način na koji se kreira klasa u CoffeeScriptu je veoma jednostavan i izgleda ovako:

```
class brodica
```

Vrlo jednostavno, primjer kreira klasu s imenom brodica i također kreira ime varijable rezultante koja se koristi za kreiranje slučajeva. Iza zavjesa CoffeeScript koristi konstruktorske funkcije (engl. *constructor functions*) čime se može instancirati klasa korištenjem „novi“ (engl. *new*) operatora. Definiranje konstruktora je veoma jednostavno, u kodu se samo koristi funkcija s imenom „konstruktor“ (engl. *constructor*). U kodu bi to izgledalo ovako:

```
class brodica
constructor: (ime) ->
     @ime = ime
```

U primjeru se kreira klasa s nazivom brodica te se postavlja funkcija koja se poziva prilikom svake instance varijable. U CoffeeScriptu također postoji „prečac“ kod ovog primjera, a to je da se kod definiranja konstruktora u prefiks argumenata varijable postavi znak „@“, tada

CoffeeScript automatski postavlja argumente kao svojstva instance kod konstruktora. Dodavanje svojstva instanci je slično kao i kod dodavanja svojstava kod objekata uz napomenu da se mora paziti na uvlačenje redaka prilikom definiranja istih u tijelu klase. Uz to CoffeeScript jasno definira što se događa kada se koristi debela strelica (engl. *fat arrow*) „=>“ prilikom definiranja svojstava instance. Ona će omogućiti da se metoda instance pozove u pravilnom kontekstu i da je „this“ uvijek jednak trenutnoj instanci koja se obrađuje. Potom se govori o statičkim svojstvima, oni se u CoffeeScriptu mogu povezati uz klasu na sljedeći način:

```
class brodica
     @find: (ime) ->
brodica.find(„Jahta“)
```

Primjer će kreirati klasu brodica i uz pomoć konstruktora koji mijenja naredbu „this.find“, pokrenuti funkciju s argumentom „ime“. Na kraju će se kao rezultat pohraniti vrijednost „Jahta“ u klasu brodica. Klase mogu naslijediti svojstava drugih klasa. U CoffeeScriptu to je moguće upotrebom naredbe „extend“ koja će od „roditeljske“ (engl. *parent*) klase preuzeti svojstva i dodati ih klasi „dijete“ (engl. *child*). Primjer za nasljeđivanje je sljedeći:

```
class brodica
     constructor: (@ime) ->
class camac extends brodica
```

Prema primjeru kreirati će se klasa „brodica“ sa svojstvom konstruktorske funkcija koja zaprima argument „ime“. Nakon toga će se kreirati klasa „camac“ koja nasljeđuje svojstva klase „brodica“ i samim time ta klasa postaje „dijete“ klase „brodica“.

CoffeeScript u osnovi olakšava korištenje JavaScripta u Web aplikacijam. Njegova sintaksa je lakša i pridodaje većoj preglednosti koda. Pruža lakše definiranje interakcije i olakšava početnicima implementaciju JavaScripta na željenim programima, odnosno aplikacijama. Stjecanjem osnovne logike koja stoji iza CoffeeScripta može se već započeti s implementacijom istih na Web aplikacije. Kao što je već bilo rečeno CoffeeScript je ništa drugo nego li JavaScript, uz iznimku izdvajanje samo najboljih stvari koje JavaScript nudi.

8. Primjer Web aplikacije

Kako bi se najbolje vidjela korisnost i potvrdile pozitivne, odnosno negativne stvari vezane za CoffeeScript, potrebno je izraditi Web aplikaciju. Sve se započelo s idejom da se kreira Web aplikacija s tematikom obrta za popravak oštećenja na limariji automobila koje ima i uslugu popravka i uslugu termolakiranja. Riječ je o lokalnom obrtu koje ima određene potrebe koje su morale biti realizirane tijekom razvoja Web aplikacije. Te potrebe su pretvorene u funkcionalnosti i na kraju je uz pomoć CoffeScripta ostvareno povezivanje HTML elemenata sa dizajnom te korisničkom interakcijom na Web pregledniku.

8.1. Opis ideje Web aplikacije

Sama ideja za izradu Web aplikacije je započela osmišljavanjem teme. Tema Web aplikacije je obrt koji ima usluge popravka oštećenja na limariji vozila i usluga termolakiranja. Prilikom faze postavljanja ciljeva i zahtjeva došlo je do spoznaje koje su funkcionalnosti ključne za samu Web aplikaciju. Stoga se odlučilo kako je potrebno da Web aplikacija sadržava sljedeće:

- mogućnost kontaktiranja putem obrasca
- registracija korisnika
- prijava korisnika
- pregled i uređivanje radnih naloga
- pregled statusa radnih naloga
- evidencija poruka
- slanje poruka
- upravljanje korisnicima
- upravljanje slikama u galeriji

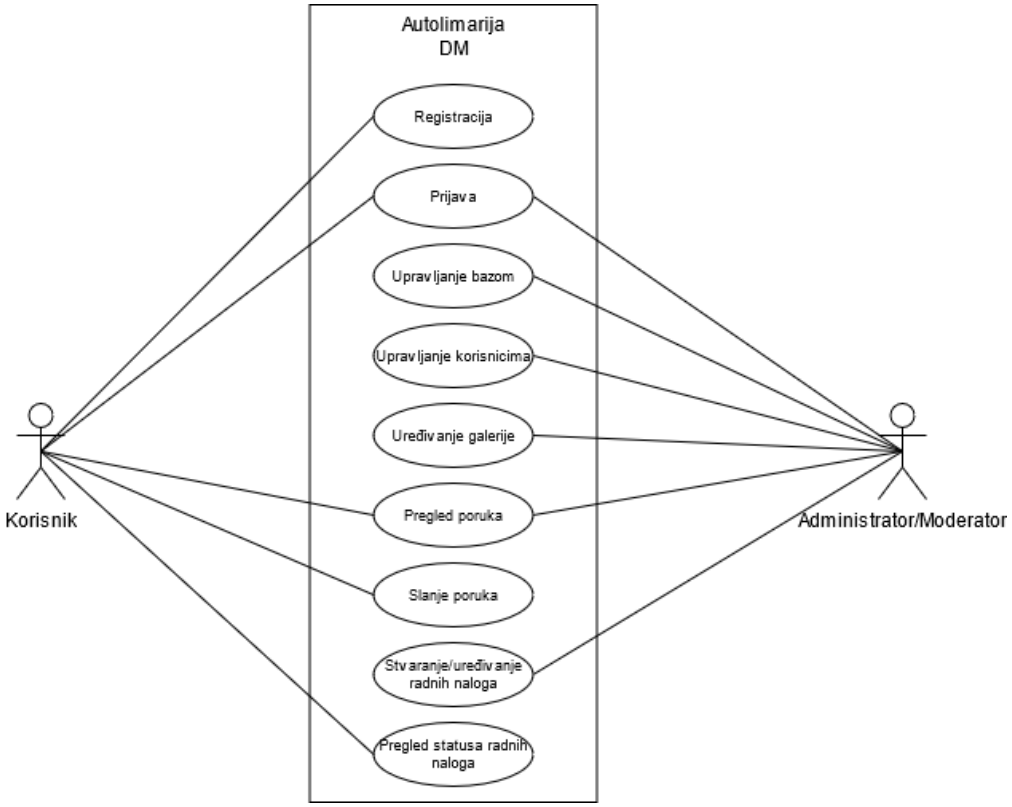
Prema navedenim funkcionalnostima, zamišljen je sljedeći scenarij rada Web aplikacije. Neregistrirani korisnik može vidjeti sadržaj na naslovnoj Web stranici koja služi za prezentacijske svrhe. On također može obaviti registraciju putem istoimenog obrasca. Nakon što kreira račun, čeka na autorizaciju koju mora obaviti administrator ili moderator putem administratorskog sučelja. Nakon što mu se odobri status korisnika, isti se može uspješno prijaviti u sustav. Potom dolazi do korisničkog sučelja gdje može vidjeti pripadajuće poruke i radne naloge koji se odnose na njega. Također može poslati novu poruku u sučelju za poruke.

Za razliku od korisničkog sučelja, administratori i moderatori imaju drugačije, administratorsko sučelje. Ono se bitno razlikuje od korisničkog sučelja jer omogućuje upravljanje i izmjenu sadržaja na samoj Web aplikaciji. Administratorsko sučelje je povezano s bazom i omogućuje veće izmjene nad podacima u bazi. Administrator, odnosno moderator, tako može pregledati sve poruke koje su poslane ili preko obrasca ili preko korisničkog sučelja. Nadalje može upravljati korisnicima, što znači da može dodavati nove korisnike, brisati postojeće korisnike ili mijenjati podatke postojećih korisnika. Potom može stvarati ili uređivati radne naloge. Nakon izrađenih radnih naloga istima je moguće mijenjati status čime se automatski mijenja status i kod korisnika. Na kraju administrator / moderator, može upravljati slikama u galeriji na način da putem obrasca unosi određene parametre vezane uz sliku u samu bazu.

Ideja je uspješno realizirana na način da su se za samo temelje i dinamičnost Web aplikacije koristili programski jezici PHP, HTML i CSS. Bitno je spomenuti kako se za CSS koristio SASS, dodatak CSS-a. SASS koristi CSS upute, ali sa svojom sintaksom i ekstenzijom „.scss“ koja se kompilira u običnu CSS datoteku. Nakon kreiranih HTML elemenata bilo je potrebno zbog dinamičnosti povezati bazu podataka koja će sadržavati entitete i atribute navedene u kasnijem poglavlju. Na kraju je dodan CoffeeScript koji je kompiliran u sam JavaScript zbog kasnije potrebe ispravljanja grešaka koja je jednostavnija kroz JavaScript kod. Kako bi Web aplikacija bila dostupna za pregled i svim korisnicima, korištena je besplatna „cloud“ platforma Heroku koja podržava PHP i koristi GIT, distribuirani sustav za upravljanje kodom, kako bi se Web aplikacija postavila na poslužitelj. Za kreiranje baze podataka na poslužitelju, korištena je besplatna usluga kreiranja baze podataka na stranici „remotemysql.com“. Nakon postavljanja i testiranja aplikacije na poslužitelju, uspješno je kreirana poveznica: https://autolimarija-dm.herokuapp.com/. Preko navedene poveznice moguće je pristupiti Web aplikaciji i koristiti sljedeće:

- Administratorsko/moderatorsko sučelje:
  - email: admin
  - lozinka: admin
- Korisničko sučelje:
  - email: korisnik
  - lozinka: 123456

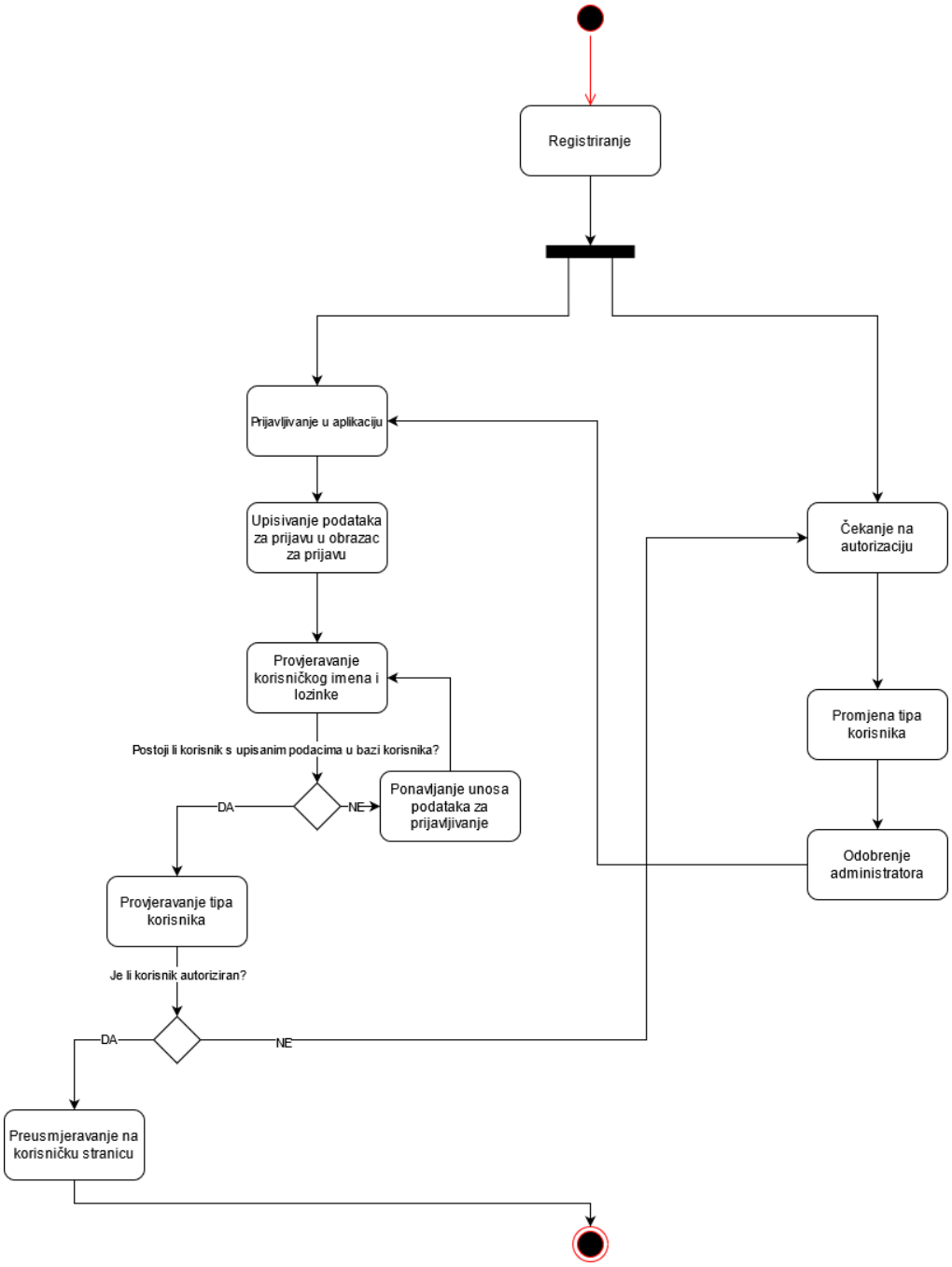
8.1.1. Dijagram slučajeva korištenja



Slika 6. Dijagram slučajeva korištenja (autorski rad)

Prema navedenome dijagramu slučajeva korištenja je vidljivo kako postoje dva tipa korisnika, običan korisnik i administrator, odnosno moderator. Običan korisnik može obaviti registraciju, prijavu u sustav, pregledavati i slati poruke i pregledavati status radnih naloga. Administrator može koristiti Web aplikaciju za prijavu u sustav. Za upravljanje bazom podataka, upravljanje korisnicima, može uređivati slike u galeriji, pregledavati poruke poslane putem obrasca na naslovnoj stranici ili kroz korisničko sučelje. Može stvarati ili uređivati radne naloge, samim time može mijenjati njihov status.

8.1.2. Dijagram slijeda aktivnosti

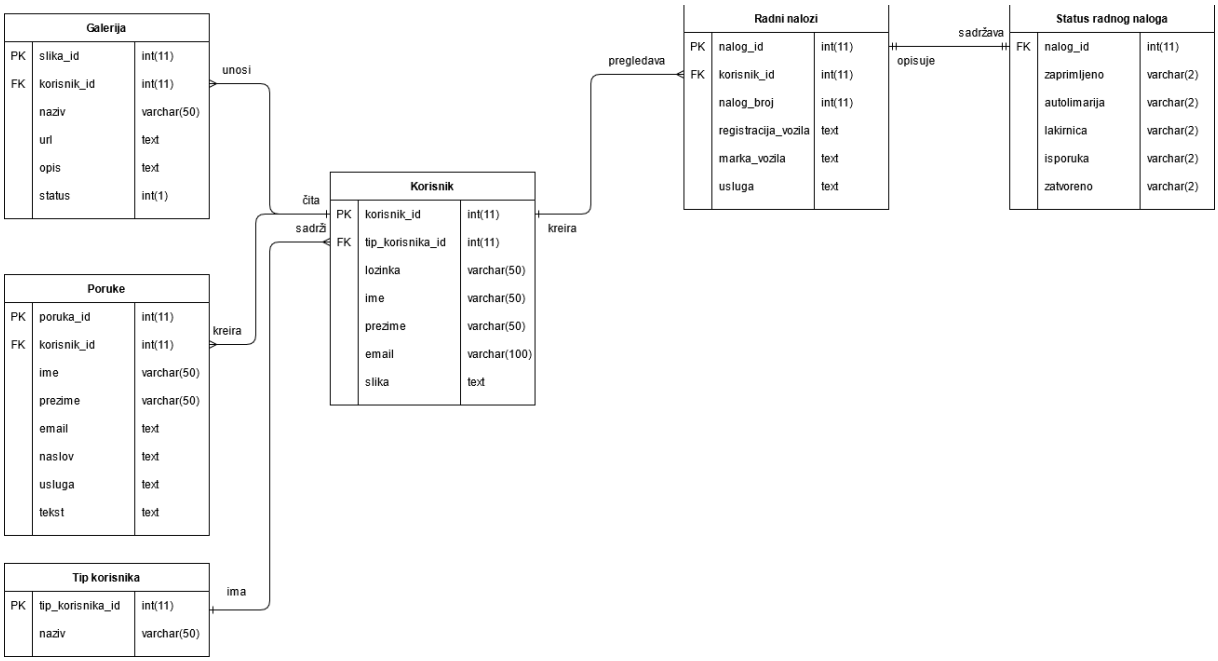


Slika 7. Dijagram slijeda aktivnosti

Dijagram slijeda aktivnosti prikazuje najvažnije aktivnosti koje se pojavljuju prilikom registracije i prijave korisnika u sam sustav. Što znači da korisnik prvo mora obaviti registraciju kako bi se prijavio u sustav, nakon čeka ima dvije mogućnosti. Mora ili čekati autorizaciju, odnosno promjenu tipa korisnika, a samim time i odobrenje administratora, ili se može pokušati prijaviti u sustav. Kod prijave obavlja aktivnost unosa podataka u obrazac za prijavu nakon čega se provjeravaju uneseni podaci. Postavlja se uvjet je li korisnik s unesenim podacima

uopće u bazi podataka. Ukoliko nije, potrebno je ponovno unijeti podatke. Ukoliko postoji korisnik u bazi podataka, tada se prelazi na sljedeću aktivnost, a to je provjeravanje tipa korisnika. Potom dolazi do novog uvjetovanja gdje se postavlja pitanje je li korisnik autoriziran. Ako je korisnik autoriziran, tada ga se preusmjerava na korisničku stranicu čime se završava slijed aktivnosti za prijavu/registraciju. Ako korisnik nije autoriziran, onda mora čekati autorizaciju, to jest, ponavlja se cijela druga kolona aktivnosti. Ovaj dijagram prikazuje aktivnosti za najvažniji dio u samoj Web aplikaciji. Postoje još aktivnosti koje se obavljaju kod ostalih funkcionalnosti, ali one su jednostavne jer sadržavaju samo unos podataka u obrazac i prijenos istih u bazu, te u drugom slučaju ažuriranje samih podataka.

8.1.3. ERA dijagram / shema baze podataka



Slika 8. ERA dijagram

ERA (engl. *Entity Relationship Attributes*) dijagram je dijagram koji će prikazivati entitete, atribute i veze između istih u bazi podataka. Stoga se u bazi podataka za kreiranu Web aplikaciju nalaze entiteti Galerija, Poruke, Tip korisnika, Korisnik, Radni nalozi, Status radnog naloga. Svaki od tih entiteta ima atribute s određenim tipom podataka koji povezuju te entitete na način da određeni primarni ključevi budu vanjski ključevi u drugim entitetima. Moguće je vidjeti kako su veze između entiteta u većini „jedan prema više“, dok je u samo jednom slučaju veza „jedan prema jedan“.

8.2. Prikaz specifičnih dijelova aplikacije i koda

Pošto se CoffeeScript isto kao i JavaScript koristi kako bi se ostvarila interaktivnost, sam opseg koda je manji nego kod koji je potreban kako bi se stvorile stranice i realizirale funkcionalnosti na Web aplikaciji. Prvi dio prikaza će obuhvatiti CoffeeScript dijelove i kod vezan za prikaze, također će se prikazati kompilirani JavaScript kod. U drugom dijelu prikaza, nalaze se ključni dijelovi koda koji služe za prikaz sadržaja i način na koje funkcionalnosti rade.

Prilikom prvog posjeta Web aplikaciji, pri samome vrhu vidljivo je zaglavlje s navigacijom. Ispod zaglavlja se pomicanjem po stranici pojavljuje traka žuta boje koja se proširuje, odnosno smanjuje vertikalnim pomakom po Web stranici.

Slika 9. Prikaz trake ispod zaglavlja i navigacije (autorski rad)

Traka je stavljena u blok s atributom klase „progress“, dok sama traka je u bloku pod atributom klase „progress__bar“ te je stilizirana na način da prvi blok ima širinu od 100%, dok drugi blok ima širinu 0%. Pomoću CoffeeScript koda:

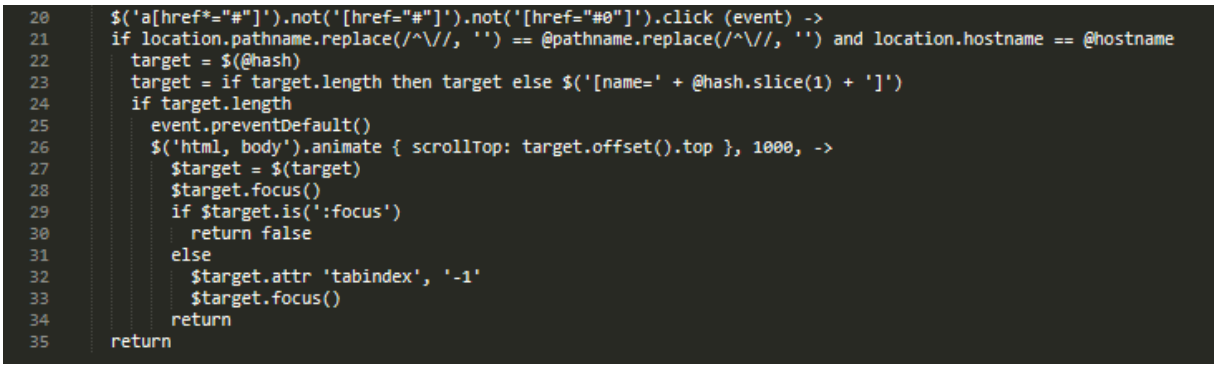
```
9   progresive = ->
10    winScroll = document.body.scrollTop or document.documentElement.scrollTop
11    height = document.documentElement.scrollHeight - (document.documentElement.clientHeight)
12    scrolled = winScroll / height * 100
13    document.getElementById('progress__bar').style.width = scrolled + '%'
14    return
```

Slika 10. Prikaz koda za traku ispod zaglavlja i navigacije (autorski rad)

Prilikom kompiliranja prikazanog koda u JavaScript dobivaju se slijedeće linije koda:

```
progresive = function() {
  var height, scrolled, winScroll;
  winScroll = document.body.scrollTop ||
document.documentElement.scrollTop;
  height = document.documentElement.scrollHeight -
document.documentElement.clientHeight;
  scrolled = winScroll / height * 100;
  document.getElementById('progress__bar').style.width = scrolled + '%';
};

window.onscroll = function() {
  progresive();
};
```

Nakon toga dolazi do sljedeće bitne funkcije koja se također može prepoznati na naslovnoj stranici, a to je lagani prijelaz prilikom vertikalnog pomaka kod pritiska na poveznice u navigaciji. Pošto ne postoji određeni element koji je prikazan na stranici već kod koristi cijelu stranicu kao okidač za pokretanje funkcije. Moguće je prikazati samo kod koji stoji iza toga, a on izgleda ovako:



```
20    $('a[href*="#"]').not('[href="#"]').not('[href="#0"]').click (event) ->
21    if location.pathname.replace(/^\//, '') == @pathname.replace(/^\//, '') and location.hostname == @hostname
22      target = $(@hash)
23      target = if target.length then target else $('[name=' + @hash.slice(1) + ']')
24      if target.length
25        event.preventDefault()
26        $('html, body').animate { scrollTop: target.offset().top }, 1000, ->
27          $target = $(target)
28          $target.focus()
29          if $target.is(':focus')
30            return false
31          else
32            $target.attr 'tabindex', '-1'
33            $target.focus()
34          return
35    return
```

Slika 11. Prikaz koda za lagane prijelaze prilikom pritiska poveznice (autorski rad)

Kada bi se navedeni CoffeeScript kod kompilirao u JavaScript, dobio bi se sljedeći rezultat:

```
$('a[href*="#"]').not('[href="#"]').not('[href="#0"]').click(function(ev
ent) {
  var target;
  if (location.pathname.replace(/^\//, '') ===
this.pathname.replace(/^\//, '') && location.hostname === this.hostname)
{
    target = $(this.hash);
    target = target.length ? target : $('[name=' + this.hash.slice(1) +
']');
    if (target.length) {
      event.preventDefault();
      $('html, body').animate({
        scrollTop: target.offset().top
      }, 1000, function() {
        var $target;
        $target = $(target);
        $target.focus();
        if ($target.is(':focus')) {
          return false;
        } else {
          $target.attr('tabindex', '-1');
          $target.focus();
```

```
            }
          });
        }
      }
    });
```

Gore navedeni kod koristi biblioteku JavaScripta pod imenom jQuery. To se može vidjeti prema prvoj liniji koda koja prikazuje određenu vrstu upita koji se postavlja prilikom traženja poveznica na naslovnoj stranici. Zadnji dio za koji se koristio CoffeeScript je dio vezan za prijavu, a to je prikaz lozinke koja se unosi. HTML već po unaprijed definiranom polju za unos lozinke, skriva znamenke koje se unose. Stoga je bilo potrebno napraviti putem JavaScripta, odnosno CoffeeScripta funkciju koja će prikazivati lozinku kao običan tekst. Prvi prikaz prikazuje polje za unos lozinke prije upotrebe funkcije:

Slika 12. Prikaz unosa lozinke u obrazac prijave - prije (autorski rad)

Dok drugi prikaz prikazuje kako izgleda polje za unos lozinke nakon pritiska na polje za „check“ listu:

Slika 13. Prikaz unosa lozinke u obrazac prijave - nakon (autorski rad)

Te je na sljedećoj slici moguće vidjeti jednostavan kod koji stoji iza navedenog prikaza:

```
1  showPassword = ->
2    lozinka = document.getElementById('pass')
3    if lozinka.type == 'password'
4      lozinka.type = 'text'
5    else
6      lozinka.type = 'password'
7    return
```

Slika 14. Prikaz koda za prikazivanje lozinke u obrascu za prijavu

Nakon čega bi njegovo kompiliranje u JavaScript davalo ovakav rezultat:

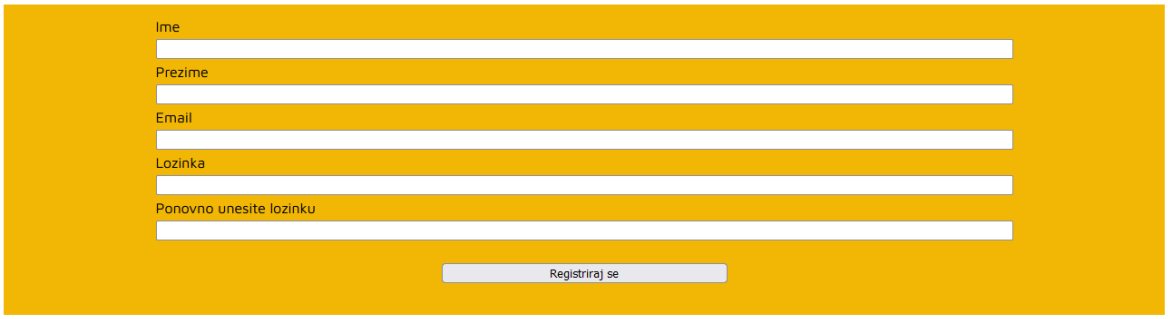
```
var showPassword;

showPassword = function() {
  var lozinka;
  lozinka = document.getElementById('pass');
  if (lozinka.type === 'password') {
    lozinka.type = 'text';
  } else {
    lozinka.type = 'password';
  }
};
```

Potom je bitno spomenuti kako dolazi do spremanja prijave. Odnosno otvaranja sjednice koja će pohraniti vrijednosti prijave dokle god je Web preglednik korisnika otvoren, ali samim time će biti prikazano i način provjere unesenih podataka. To je izvedeno na sljedeći način:

```
20      if(isset($_POST['submit'])){
21          if(isset($_POST['name']) && !empty($_POST['name']) && isset($_POST['pass']) && !empty($_POST['pass'])){
22              $email=$_POST['name'];
23              $lozinka=$_POST['pass'];
24              $sql="SELECT * FROM korisnik WHERE email='{$email}' AND lozinka='{$lozinka}'";
25              $rs=izvrsiUpit($veza, $sql);
26              if(mysqli_num_rows($rs)!=0){
27                  list($korId, $tip, $lozinka, $ime, $prezime, $email, $slika)=mysqli_fetch_array($rs);
28                  $_SESSION["aktivni_korisnik_id"] = $korId;
29                  $_SESSION["aktivni_korisnik_tip"] = $tip;
30                  $_SESSION["aktivni_korisnik_lozinka"] = $lozinka;
31                  $_SESSION["aktivni_korisnik_email"] = $email;
32                  if(isset($_SESSION["aktivni_korisnik_id"])){
33                      if($_SESSION["aktivni_korisnik_tip"]!=0 && $_SESSION["aktivni_korisnik_tip"]!=1){
34                          $_SESSION["aktivni_korisnik_ime"] = $ime . " " . $prezime;
35                      } else {
36                          $_SESSION["aktivni_korisnik_ime"] = $ime;
37                      }
38                  }
39                  if($tip==3){
40                      $greska="Vaš račun nije autoriziran. Pričekajte da admin odobri registraciju.";
41                  } else {
42                      header("location: dashboard.php");
43                      exit();
44                  }
45              } else {
46                  $greska="Email i lozinka se ne podudaraju!";
47              }
48          } else {
49              $greska="Unesite email i lozinku!";
50          }
51      }
52  ?>
```

Slika 15. Prikaz koda za provjeru i spremanje podataka prijave u sjednicu (autorski rad)

Unatoč tome bilo bi dobro prikazati i registraciju korisnika u Web aplikaciju. Ona je izvedena putem obrasca koja će slati podatke u bazu podataka te tako kreirati novog korisnika koji će morati čekati na autorizaciju. Izgled obrasca je sljedeći:



Slika 16. Prikaz obrasca za registraciju korisnika (autorski rad)

Potom slijedi prikaz koda koji prikazuje kako su poslani podaci u obrascu u bazu podataka:

```php
if(isset($_POST['submit'])) {
    if(isset($_POST['password']) && !empty($_POST['password']) && isset($_POST['password2']) && !empty($_POST['password2'])) {
        if ($_POST['password']==$_POST['password2']) {
            $sql="SELECT email FROM korisnik";
            $rs=izvrsiUpit($veza, $sql);
            while(list($email)=mysqli_fetch_array($rs)){
                if($email == $_POST['email']){
                    $greska = "korisnički račun s navedenom email adresom već postoji, molimo da unesete drugu email adresu.";
                }
            }
            if (empty($greska)){
                $korId=$_POST['novi'];
                $korTip=$_POST['tip'];
                $ime=$_POST['name'];
                $prezime=$_POST['surname'];
                $email=$_POST['email'];
                $password=$_POST['password'];
                $slika=" ";
                $sql="INSERT INTO `korisnik`(`korisnik_id`, `tip_korisnika_id`, `lozinka`, `ime`, `prezime`, `email`, `slika`) VALUES ('$korId','$korTip','$password','$ime','$prezime','$email','$slika')";
                $rs=izvrsiUpit($veza, $sql);
                $obavijest="Račun je uspješno kreiran.";
            }
        } else {
            $greska="Lozinke se ne podudaraju.";
        }
    }else {
        $greska="Unesite lozinku.";
    }
}
```

Slika 17. Prikaz koda za slanje podataka s obrasca za registraciju u bazu podataka (autorski rad)

Još slijede dva prikaza koja će pojasniti kako se podaci prikazuju iz baze podataka i kako se podaci u samoj bazi mijenjaju. Pošto postoji više funkcionalnosti, ali sve koriste sličnu logiku, prikazati će se samo dijelovi za dohvaćanje i prikazivanje poruka te izmjena, unos ili brisanje radnih naloga. Prilikom prikaza poruka, na Web aplikaciji se dobiva sljedeći prikaz:

ID korisnika	Ime	Prezime	Email	Naslov poruke	Naziv usluge	Tekst poruke	
4	Test	Test	test@email.com	Upit	lakiranje	test	Pogledaj poruku
4	Ivo	Ivić	ivo@gmail.com	Upit vezan za termin pregleda vozila	upit	Poštovani, zanima me...	Pogledaj poruku

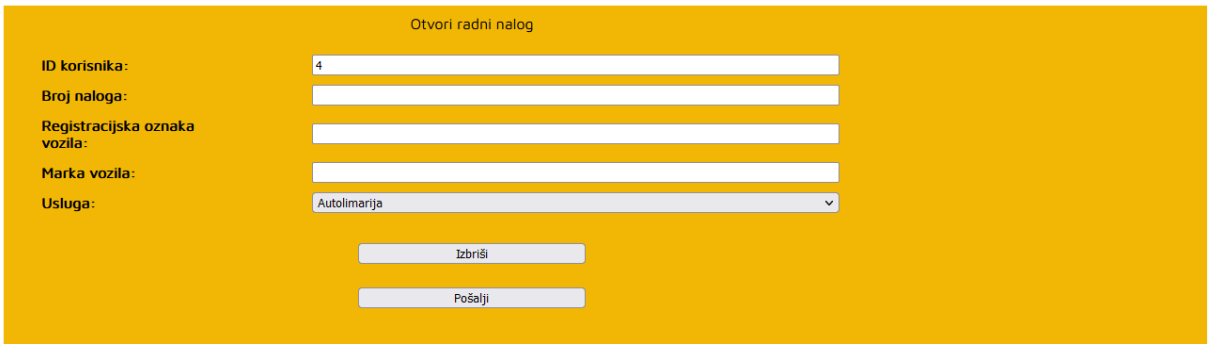
Slika 18. Prikaz pregleda poruka na administratorskim stranicama (autorski rad)

U kodu dohvaćanje i prikaz podataka je postignuto na sljedeći način:

```php
    $sql="SELECT * FROM poruke";
    $rs=izvrsiUpit($veza, $sql);
?>
<table class="dashboard__table">
    <thead>
        <tr>
            <th>ID korisnika</th>
            <th>Ime</th>
            <th>Prezime</th>
            <th>Email</th>
            <th>Naslov poruke</th>
            <th>Naziv usluge</th>
            <th>Tekst poruke</th>
            <th></th>
        </tr>
    </thead>
    <tbody>
        <?php
            if(isset($_SESSION['aktivni_korisnik_id'])) {
                if($_SESSION['aktivni_korisnik_tip']==2) {
                    $sql="SELECT * FROM poruke WHERE korisnik_id = {$_SESSION['aktivni_korisnik_id']}";
                    $rs=izvrsiUpit($veza, $sql);
                }
            }
            while(list($id, $korID, $ime, $prezime, $email, $naslov, $usluga, $tekst)=mysqli_fetch_array($rs)) {
                echo "<tr>
                    <td>$korID</td>
                    <td>$ime</td>
                    <td>$prezime</td>
                    <td>$email</td>
                    <td>$naslov</td>
                    <td>$usluga</td>
                    <td>$tekst</td>
                    <td><a href=\"poruka.php?id={$id}\">Pogledaj poruku</a></td>
                    </tr>";
            }
        ?>
    </tbody>
</table>
```

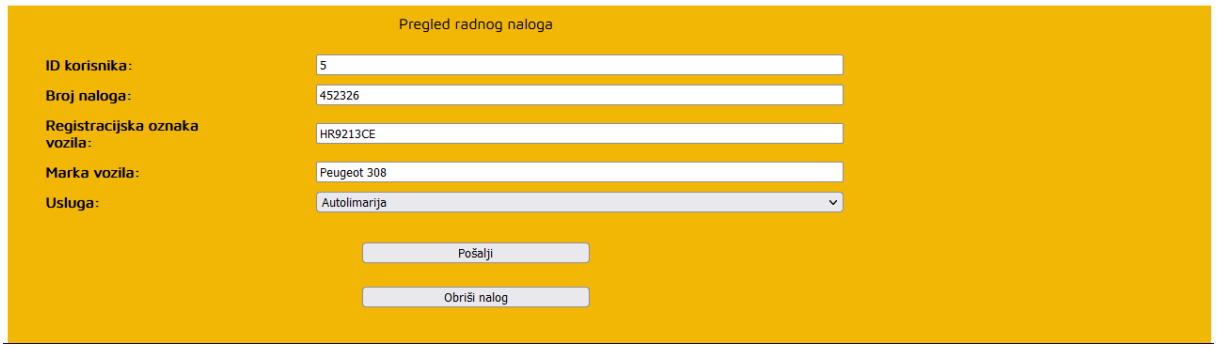
Slika 19. Prikaz koda za dohvaćanje i prikaz podatka u tablici (autorski rad)

Ako se radi o unosu podataka u bazu, onda se to izvodi pomoću obrasca za unos radnog naloga koji izgleda ovako:



Slika 20. Prikaz obrasca za unos radnog naloga (autorski rad)

Ukoliko je riječ o izmjeni podataka na radnom nalogu ili brisanju istog, onda to izgleda ovako:



Slika 21. Izmjena podataka na radnom nalogu (autorski rad)

Kod kojim se izvodi unos, promjena ili brisanje podatka o radnim nalozima je slijedeći:

```php
<?php
    $greska="";
    if(isset($_POST['submit'])){
        foreach($_POST as $key => $value){
            if(strlen($value)=='0'){
                $greska = "Sva polja za unos su obavezna";
            }
        }
        if(empty($greska)) {
            $id=$_GET['id'];
            $IDkor=$_POST['IDkor'];
            $brNal=$_POST['brNal'];
            $reg=$_POST['reg'];
            $marka=$_POST['marka'];
            $usluga=$_POST['usluga'];
            if($id==0){
                $sql="INSERT INTO `radni_nalozi`(`nalog_id`, `korisnik_id`, `nalog_broj`, `registracija_vozila`, `marka_vozila`, `usluga`) VALUES ('$id','$IDkor','$brNal','$reg','$marka','$usluga');";
            }
            else{
                $sql="UPDATE radni_nalozi SET
                korisnik_id = '$IDkor',
                nalog_broj = '$brNal',
                registracija_vozila = '$reg',
                marka_vozila = '$marka',
                usluga = '$usluga' WHERE nalog_id='$id'
                ";
            }
            izvrsiUpit($veza, $sql);
            header("Location: radni-nalozi.php");
        }
    }
    if(isset($_GET['id'])){
        $id=$_GET['id'];
        $sql="SELECT nalog_id FROM status_radnog_naloga WHERE nalog_id='$id'";
        $rs=izvrsiUpit($veza, $sql);
        list($IDnaloga)=mysqli_fetch_array($rs);
        if ($IDnaloga!=$id) {
            $sql="INSERT INTO `status_radnog_naloga`(`nalog_id`, `zaprimljeno`, `autolimarija`, `lakirnica`, `isporuka`, `zatvoreno`) VALUES ('$id','1','0','0','0','0')";
            izvrsiUpit($veza, $sql);
        }
        $sql="SELECT * FROM radni_nalozi WHERE nalog_id='$id'";
        $rezultat=izvrsiUpit($veza, $sql);
        list($id, $IDkor, $brNal, $reg, $marka, $usluga)=mysqli_fetch_array($rezultat);
    }
    else {
        $IDkor="";
        $brNal="";
        $reg="";
        $marka="";
        $usluga="";
    }
    if(isset($_POST["reset"])){
        header("location: radni-nalog.php");
    }
    if(isset($_POST['delete'])){
        $sql="DELETE FROM status_radnog_naloga WHERE nalog_id='$id'";
        izvrsiUpit($veza, $sql);
        $sql="DELETE FROM radni_nalozi WHERE nalog_id='$id'";
        izvrsiUpit($veza, $sql);
        header("location: radni-nalozi.php");
    }
?>
```

Slika 22. Prikaz koda za unos, izmjenu ili brisanje radnog naloga (autorski rad)

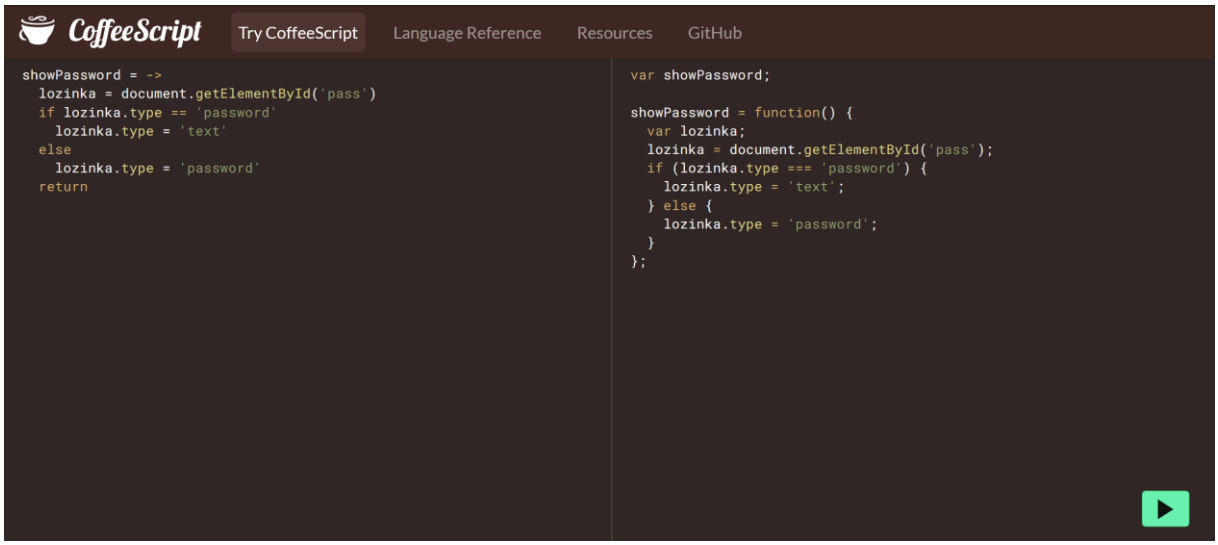
Sličnu logiku programiranja prate i ostale funkcionalnosti, stoga se neće prikazivati u daljnjem dijelu rada. Prema navedenim prikazima je moguće vidjeti kako su se realizirali bitni elementi Web aplikacije i na koji način je implementiran CoffeeScript. Čisto usporedbe radi CoffeeScript kod je pregledniji od JavaScript koda te sadrži 35 linija koda, dok JavaScript izgleda veće i sadržava 47 linija koda. Prilikom implementacije veće interaktivnosti može se pojaviti kako JavaScript ima puno veću količinu koda od CoffeeScripta, čime se otkriva navedena prednost CoffeeScripta, a to je da uz manje koda daje željenu interakciju.

9. CoffeeScript, dobra alternativa za JavaScript?

Već u spomenutim poglavljima, vidljive su prednosti i mane samog CoffeeScripta, ali tek u njegovoj stvarnoj implementaciji na Web aplikaciji, moguće je komentirati jesu li to zaista prednosti ili mane kod samog jezika. Stoga je najbolje započeti s pozitivnim stranama CoffeeScripta.

9.1. Prednosti CoffeeScritpta

Prva od bitnih prednosti je instalacija. Ona se može izvršiti preko Node.js kao paket u samome projektu ili globalno na računalo. Najbitnije je reći kako zapravo nema potrebe za instalacijom već se CoffeeScript može kompilirati putem Web preglednika. Vrlo je jednostavan za korištenje početnicima zbog svoje sintakse i nije toliko zahtjevan kao JavaScript.



Slika 23. Primjer kompiliranog koda putem web stranice CoffeeScript.org

CoffeeScript, slično kao i Python koristi prazna mjesta, odnosno uvlake u sintaksi. Na taj način se bolje grupira blok koda, čime se izostavlja upotreba zagrada što čini kod čišćim. Za početnike je CoffeeScript odlična odskočna daska za ostvarivanje interakcije na Web aplikacijama, jer se kompilira u čisti JavaScript, dok je za pisanje CoffeeScripta koda olakšana sintaksa kreirana prema programskim jezicima Ruby i Python. CoffeeScript je lagan za učenje, brži za pisanje i razvoj Web aplikacija, čitljiviji od JavaScripta, jednostavan i radi isto što i JavaScript samo uz manje potrebnog koda.

9.2. Nedostaci CoffeeScripta

Ona loša strana i glavna mana CoffeeScripta je vrijeme. Kako je vrijeme odmicalo i razvoj JavaScripta se tijekom godina mijenjao dok je CoffeeScript pokušavao kaskati za tim promjenama. JavaScript danas ima mnoge svoje biblioteke (engl. *libraries)* poput React-a,jQuery-a i sličnih, ali i programskih jezika koji su razvijeni temeljem JavaScripta, TypeScript, Dart, Kaffeine i ostalih. Zbog čega se CoffeeScript polako gubio iz upotrebe u razvoju Web aplikacija. Nakon toga dolazi do problema prilikom kompiliranje što znači da CoffeeScript ponekad koristi svoj način za kompiliranje sintakse u JavaScript što ponekad rezultira u suvišnim linijama kode zbog kojih se ne dobiva željeni rezultat. Otklanjanje pogrešaka je otežano pošto se kod kompilira u JavaScript, stoga u samome CoffeeScriptu nije olakšano otkrivanje istih, zbog čega je potrebno osnovno znanje JavaScripta. Sklon je promjenama, što znači da ukoliko se pojavi nova verzija CoffeeScripta instalirana u projektu, doći će do nefunkcionalnosti koda uslijed raznih promjena koje su dobivene novom verzijom. Za izvođenje „lakših“ stvari na Web aplikaciji, CoffeeScript pomaže pri razvoju, ali kada se pojave „teže“ stvari prilikom razvoja, tada je potrebno poznavanje HTML-a, DOM-a i CSS-a jer je JavaScript zapravo most koji povezuje spomenute jezike.

10. Zaključak

Na samome kraju završnoga rada se može dati odgovor na pitanje je li CoffeeScript dobra alternativa za JavaScript. Prije konkretnog odgovora, ukratko je potrebno reći kako je CoffeeScript u svojim počecima bio odličan odgovor za jednostavno postizanje interaktivnosti na Web aplikacijama jer je bio sve što JavaScript nije, ali je opet bio JavaScript u svojoj srži. Prolaskom godina, može se reći kako ga je vrijeme pregazilo i kako se u svijetu razvoja Web aplikacija ponudilo puno alternativa koje su korištenije od samoga CoffeeScripta. Uvidom u cijelu priču ovoga rada, vide se bitni elementi i osnove kod razvoja Web aplikacija. Daju se temelji JavaScripta te njegova povezanost, ali i razlika s CoffeeScriptom. Praktični primjer prikazuje kako se u realnoj Web aplikaciji primjenjuje CoffeeScript te kako se on zapravo kompilira u JavaScript. Kao zaključak se može reći sljedeće, a to je da CoffeeScript služi kao dobra podloga „novopečenim“ programerima kojima je potrebna preglednost i razumijevanja, odnosno čitljivost samoga koda kako bi bolje razumjeli što se to događa, dok iskusniji programeri koriste neke druge alternative JavaScripte koje drže korak s razvojem istog. Stoga da se napokon da odgovor na pitanje, CoffeeScript je jedna od dobrih alternativa za JavaScript.

Popis literature

Berners-Lee, T., Cailliau, R., Luotonen, A., Nielsen, H. F., & Secret, A. (1994). The world-wide web. Communications of the ACM, 37(8), 76-82.

Jazayeri, M. (2007, May). Some trends in web application development. In Future of Software Engineering (FOSE'07) (pp. 199-213). IEEE.

Hadley, M. J. (2006). Web application description language (WADL). Sun Microsystems. Inc.: Menlo Park, CA, USA.

Conallen, J. (1999). Modeling web application architectures with UML. Communications of the ACM, 42(10), 63-70.

Makhija, R. (n.d.). What are the types of web applications? Preuzeto 29.06.2021., s https://www.gurutechnolabs.com/types-of-web-applications/

Murugesan, S., & Ginige, A. (2005). Web engineering: Introduction and perspectives. In Web engineering: principles and techniques (pp. 1-30). IGI Global.

Raggett, D., Le Hors, A., & Jacobs, I. (1999). HTML 4.01 Specification. W3C recommendation, 24.

Mikkonen, T., & Taivalsaari, A. (2008, August). Web applications–spaghetti code for the 21st century. In 2008 Sixth international conference on software engineering research, management and applications (pp. 319-328). IEEE.

Lerdorf, R., Tatroe, K., & MacIntyre, P. (2006). Programming Php. " O'Reilly Media, Inc.".

Welling, L., & Thomson, L. (2003). PHP and MySQL Web development. Sams Publishing.

Bujlow, T., Carela-Español, V., Solé-Pareta, J., & Barlet-Ros, P. (2015). Web Tracking:

Mechanisms, Implications, and Defenses. *arXiv:1507.07872 [cs]*.

http://arxiv.org/abs/1507.07872

Usability.gov (n.d.). User Interface Elements. Preuzeto 30.06.2021., s https://www.usability.gov/how-to-and-tools/methods/user-interface-elements.html

Wilton, P. (2004). Beginning JavaScript. John Wiley & Sons.

Rauschmayer, A. (2014). Speaking JavaScript: an in-depth guide for programmers. " O'Reilly Media, Inc.".

ECMA International (2021). ECMAScript® 2022 Language Specification. Preuzeto 14.07.2021., s https://tc39.es/ecma262/

W3Schools. (n.d.). JavaScript Tutorial. Preuzeto 14.07.2021., s https://www.w3schools.com/js/default.asp

Mozilla Corporation. (n.d.). JavaScript Guide - JavaScript | MDN. Preuzeto 06.08.2021., s https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide

Kantor, I. (21.06.2021.). JavaScript Fundamentals: Functions. Preuzeto 06.08.2021., s https://javascript.info/function-basics

MacCaw, A. (2012). The little book on CoffeeScript. O'Reilly Media.

Jeremy Ashkenas (04.05.2018.) Github repozitorij: CoffeeScript. Preuzeto 09.08.2021., s https://github.com/jashkenas/coffee-script/commit/8e9d637985d2dc9b44922076ad54ffef7fa8e9c2

CoffeeScript. (03.07.2020.). Preuzeto 10.08.2021., s https://coffeescript.org/

Lee, P. (2014). CoffeeScript in action. Simon and Schuster.

Hoigaard, E. (2012). *Smooth CoffeeScript* (Interactive). https://autotelicum.github.io/Smooth-CoffeeScript/interactive/interactive-coffeescript.html

Popis slika

Popis tablica

Prilozi (1, 2, …)