



VELEUČILIŠTE U RIJECI

RAZVOJ INTERAKTIVNIH WEB-APLIKACIJA

Vlatka Davidović

Rijeka, 2023.

VLATKA DAVIDOVIĆ, viši pred.

RAZVOJ INTERAKTIVNIH WEB-APLIKACIJA

Nakladnik: Veleučilište u Rijeci,
Trpimirova 2/V, Rijeka

Za nakladnika: dr. sc. Saša Hirnig, prof. struč. stud.

Recenzenti:

dr. sc. Marin Kaluža, prof. struč. stud., Veleučilište u Rijeci
dr. sc. Maja Gligora Marković, znan. sur.,
Sveučilište u Rijeci, Medicinski fakultet

Lektorica:

doc. dr. sc. Sanja Grakalić Plenković

Tisak: Veleučilište u Rijeci

Naklada: *online* izdanje

Nastavna skripta Razvoj interaktivnih *web*-aplikacija intelektualno je vlasništvo, neotuđivo, zakonom zaštićeno i mora se poštovati. Nijedan dio ove publikacije ne smije se preslikavati, umnožavati ili na bilo koji drugi način reproducirati, uključujući *web*-distribuciju i sustave za pretraživanje te skladištenje podataka, bez pisanoga dopuštenja izdavača.

ISBN: 978-953-8286-07-0

Povjerenstvo za izdavačku djelatnost Veleučilišta u Rijeci odobrilo je izdavanje ove nastavne skripte (KLASA: 003-09/23-01/09, URBROJ: 2170-57-01-23-13/MJG).

Razvoj interaktivnih *web*-aplikacija

Predgovor

Ova nastavna skripta namijenjena je studentima stručnog studija Informatika na Veleučilištu u Rijeci koji slušaju predmet Razvoj interaktivnih *web*-aplikacija. U skripti su pokrivena teme koje prate sadržaj predmeta i ishode učenja.

Kroz predmet studenti usvajaju koncepte i pojmove vezane uz arhitekturu *web*-aplikacije, rad *web*-klijenta i *web*-poslužitelja te protokole i formate za komunikaciju i razmjenu podataka između klijenata i poslužitelja. Napravljena je raščlana klijentskog i poslužiteljskog dijela *web*-aplikacije, te se ovdje naglasak stavlja na razvoj klijentskog dijela *web*-aplikacije. U razvoju klijentskog dijela *web*-aplikacije bitno je poznavati jezike za formiranje strukture i dizajna grafičkih elemenata te programiranje interaktivnih dijelova. Da bi se ubrzao razvoj *web*-aplikacije koriste se softverski razvojni okviri. Opisane su karakteristike softverskih razvojnih okvira za izradu klijentskih *web*-aplikacija, komponente klijentskog softverskog razvojnog okvira Quasar.js, dodavanje interakcija i dinamičko povezivanje s aplikacijskim programskim sučeljem s poslužiteljske strane. U skripti su napravljeni i objašnjeni primjeri programskog koda klijentske *web*-aplikacije. Prikazano je i opisano povezivanje s aplikacijskim programskim sučeljima i korištenje metoda prijenosa podataka koje pozivaju operacije kreiranja, čitanja, ažuriranja i brisanja podataka.

U pripremi za razvoj *web*-aplikacije odabire se i postavlja radno okruženje i repozitorij za čuvanje verzija aplikacije. Gotova aplikacija se konfigurira i prebacuje u izvršni oblik za produkcijsko okruženje.

Uz aplikaciju, izlazni proizvodi ovog predmeta su tehnička i korisnička dokumentacija, koje nisu pokrivena skriptom.

Primjeri koji su u skripti napisani mogu se iskopirati i testirati, s obzirom na to da je ovo izdanje skripte napravljeno u digitalnom obliku. Potrebno je pripaziti kod kopiranja dijelova programskog

koda na nevidljive znakove koji se mogu pojaviti u tekstu, a koji bi mogli raditi greške kod pokretanja programa.

Iako se *web* i internet danas koriste kao sinonimi, u tekstu se umjesto termina internetski preglednik ciljano koristi termin *web*-preglednik, kako bi se naglasilo da je *web* jedna od usluga na internetu. Smatram da je to bitno razlikovati u kontekstu ovog kolegija, za buduće razvojne inženjere.

Zahvalila bih recenzentima na konstruktivnim kritikama prilikom kreiranja ove skripte.

Autorica

Sadržaj

1	Arhitektura <i>web</i> -aplikacije	6
1.1	<i>Web</i> -klijent. <i>Web</i> -poslužitelj	7
1.2	Formati za prijenos podataka	11
1.3	Troslojna arhitektura <i>web</i> -aplikacija.....	13
2	Priprema radnog okruženja za razvoj <i>web</i> -aplikacija.....	16
2.1	Instalacija i konfiguracija alata za razvoj <i>web</i> -aplikacije.....	16
2.2	Instalacija alata za čuvanje verzija aplikacije	17
2.3	Instalacija i konfiguracija softverskog razvojnog okvira za razvoj klijentskog dijela <i>web</i> -aplikacije	19
2.4	Rad s upraviteljem paketa	21
2.5	Instalacija Quasar razvojnog okvira i kreiranje projekta.	22
2.5.1	Rad na novom projektu.....	22
2.5.2	Rad na postojećem projektu.....	23
3	Repozitoriji i alati za čuvanje verzija softvera	26
3.1	Kreiranje repozitorija	26
3.2	Osnovne naredbe u radu s repozitorijem.....	27
3.3	Rad s udaljenim repozitorijem	29
3.4	Rad s granama repozitorija u timskom okruženju.....	29
4	Programski jezici i softverski razvojni okviri za razvoj klijentskog dijela <i>web</i> -aplikacije ...	32
4.1	Osnovni koncepti klijentskih <i>web</i> -tehnologija.....	32
4.1.1	HTML	32
4.1.2	CSS	39
4.1.3	DOM	43

4.1.4	JavaScript.....	45
4.1.5	AJAX	58
4.2	Karakteristike softverskih razvojnih okvira za realizaciju klijentskog dijela <i>web</i> - aplikacija.	60
4.2.1	Razvojni okviri Quasar i Vue.js.....	61
4.2.2	Responzivni dizajn.....	61
4.2.3	Single page application.....	62
4.3	Struktura dokumenta u softverskom razvojnom okviru.....	62
4.4	Životni ciklus aplikacija u softverskom razvojnom okviru.....	63
5	Interaktivna <i>web</i> -aplikacija.....	68
5.1	Raspored elemenata u softverskom razvojnom okviru	68
5.1.1	Layout	68
5.1.2	Page.....	71
5.1.3	Router.....	71
5.2	Komponente i njihov razmještaj	73
5.3	Oblikovanje <i>web</i> -dokumenta. Grafički elementi	76
5.4	Dinamizacija <i>web</i> -dokumenta. Dodavanje interaktivnosti na grafičke elemente	78
5.4.1	Korištenje direktiva.....	79
5.4.2	Događaji.....	83
5.4.3	Funkcije.....	88
5.5	Asinkrona komunikacija sa serverskom stranom.....	89
5.6	Povezivanje aplikacije s aplikacijskim programskim sučeljem. Izvršavanje CRUD operacija	91
5.6.1	Dohvat podataka iz postojećih aplikacijskih programskih sučelja	92

5.6.2	Kreiranje aplikacijskog programskog sučelja	103
5.6.3	Kreiranje aplikacije s CRUD operacijama.....	108
6	Razvojno i produkcijsko okruženje	118
7	LITERATURA	120
8	POPIS SLIKA.....	123

1 Arhitektura *web*-aplikacije

Web-aplikacija je vrsta aplikacije koja za rad koristi *web*-tehnologije, kojoj korisnici mogu pristupiti putem *web*-preglednika. U tom slučaju *web*-preglednik se koristi kao klijentska aplikacija i, osim *web*-preglednika, korisnici ne moraju razmišljati o instalaciji drugih aplikacija i dodataka.

Za rad takve aplikacije koriste se *web*-tehnologije koje prikazuju podatke korisniku u obliku koji je „ugodan“ te postoji cijela grana u informatici koja se bavi korisničkim iskustvom u radu s takvim aplikacijama – UX (engl. *User eXperience*). Podaci se pohranjuju na poslužiteljima, a za prijenos podataka se koristi HTTP/HTTPS protokol.

Web-aplikacija se sastoji od klijentskog dijela (engl. *client side*) i poslužiteljskog dijela (engl. *server side*); ta se podjela odnosi na to gdje se koji dio aplikacije izvršava. Klijentski dio *web*-aplikacije se prikazuje i izvršava unutar *web*-preglednika. Korisnik je u interakciji s klijentskim dijelom aplikacije, a taj dio komunicira s poslužiteljskim dijelom aplikacije. Gledano prema tome što se prikazuje krajnjem korisniku, *web*-aplikaciju dijelimo na prednji dio (engl. *frontend*) koji korisnik vidi i pozadinski dio (engl. *backend*) koji ne vidi.

Klijentski dio aplikacije čine osnovne tehnologije: HyperText Markup Language HTML, Cascading Style Sheets CSS i JavaScript. HTML i CSS su jezici koji služe za oblikovanje izgleda dokumenta koji *web*-preglednik interpretira i prikazuje, a JavaScript je programski jezik koji omogućuje dinamičnu interakciju korisnika s korisničkim sučeljem (engl. *user interface*) *web*-aplikacije.

Poslužiteljski dio aplikacije izvršava se na poslužitelju, a može biti razvijen u bilo kojem programskom jeziku koji poslužitelj podržava (PHP, Python, JavaScript, Java...). Uobičajeno klijentski dio aplikacije direktno komunicira samo s poslužiteljskim dijelom aplikacije. Poslužiteljski dio aplikacije obično radi CRUD (engl. *create, read, update, delete*) operacije s podacima iz baze podataka, pristupa datotekama, *web*-servisima ili drugim aplikacijama.

Primjeri *web*-aplikacija su brojni: *online* uređivanje sadržaja, Facebook, Google, MS Office 365, *online* rezervacije, novinski portali, *web*-sjedišta različitih organizacija koja se temelje na nekom sustavu za upravljanje sadržajem, usluge u oblaku (engl. *cloud services*) i druge. Uredska poslovanja su se također prebacila na *web*. Trenutno postoji vrlo malo sadržaja na *webu* koji nije dohvaćen, dinamički generiran, renderiran i prikazan putem *web*-preglednika. Pri tom bilo koje dinamičko generiranje/renderiranje podataka mora izvesti *web*-aplikacija.

1.1 *Web*-klijent. *Web*-poslužitelj

Web se temelji na distribuiranoj klijentsko-poslužiteljskoj arhitekturi. U ovom tipu arhitekture jedan ili više klijenata pristupaju poslužitelju i šalju zahtjeve. Poslužitelj upravlja resursima i isporučuje resurse na zahtjev klijenata. Komunikacija između klijenta i poslužitelja definirana je komunikacijskim protokolom.

Na *webu*, klijenti su *web*-preglednici, poslužitelji su *web*-poslužitelji, a komunikacija se odvija putem HTTP/HTTPS protokola.

Web-poslužitelji udomljavaju (engl. *hosting*) dokumente različitih tipova koje mogu isporučiti na zahtjev klijenta ili mogu dinamički generirati dokumente s potrebnim podacima koji će se, isto na zahtjev, isporučiti klijentu. *Web*-poslužitelji su obično pokrenuti na računalu koje je stalno dostupno na internetu i ima IP adresu i neko simboličko ime, te im klijenti mogu pristupiti u bilo koje doba s bilo koje lokacije, ako postavke poslužitelja dopuštaju pristup s korisnikove IP adrese. No također se mogu pokrenuti i na bilo kojem računalu. Radi potrebe razvoja *web*-sjedišta i/ili aplikacije programeri mogu pokrenuti *web*-poslužitelj na vlastitom računalu i pristupati mu lokalno. U tom slučaju *web*-poslužitelj ima simboličko ime localhost, većinom s adresom 127.0.0.1. U tom slučaju klijent (*web*-preglednik) pristupa poslužitelju prema lokalnoj adresi.

Osim adrese *web*-poslužitelja, korisno je znati i broj priključka (engl. *port*) na kojem *web*-poslužitelj očekuje komunikaciju. Standardno je definiran na broju 80 (HTTP) ili 443 (HTTPS), no mogući su i drugi brojevi (8080, 8081, 8843). Oni se definiraju kod konfiguriranja *web*-poslužitelja.

Da bi *web*-klijent mogao dohvatiti neki resurs, odnosno poslati zahtjev za dohvatom nekog resursa (*web*-stranice, slike, nekog dokumenta ili podataka u formatiranom ili neformatiranom obliku), mora znati njegovu adresu – URL (engl. *Uniform Resources Locator*) (Felke-Morris, 2017).

URL je adresa resursa na internetu koja se sastoji od nekoliko komponenti:

protokol://poslužitelj.poddomena.domena/putanja_do_resursa_na_poslužitelju

pri čemu se na *webu* koristi:

- *protokol* – HTTP ili HTTPS
- *poslužitelj.poddomena.domena* – jedinstvena oznaka poslužitelja na internetu (npr. www.veleri.hr) koje se može pisati i kao IP adresa
- *putanja_do_resursa_na_poslužitelju* – resursi se nalaze u folderu kojem taj poslužitelj ima pristup

Ponekad se u URL ubacuje i broj priključka:

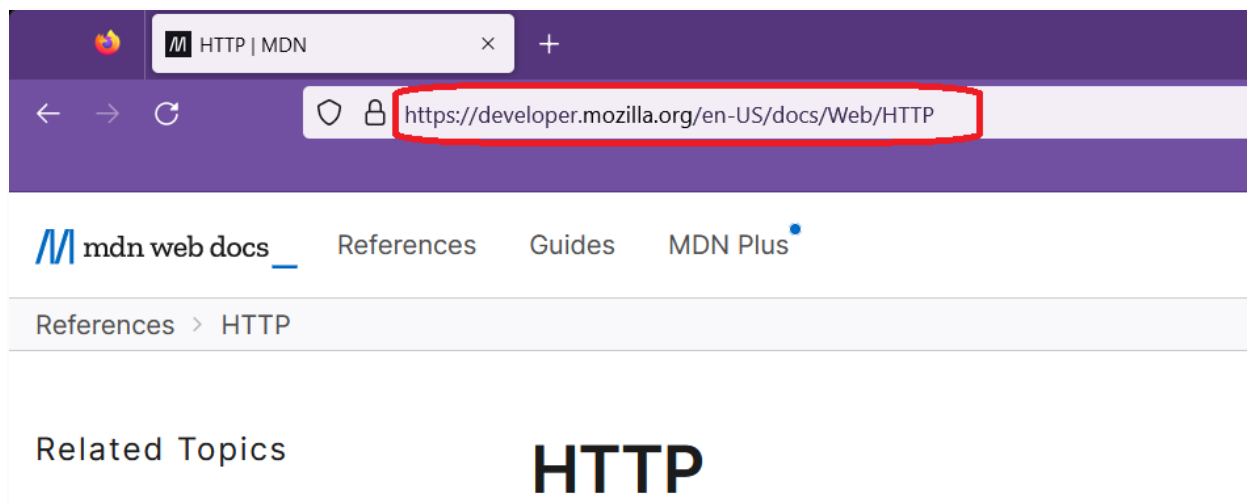
protokol://poslužitelj.poddomena.domena:broj_prikljucka/putanja_do_resursa_na_poslužitelju

U navedenim URL primjerima namjerno su izbačeni hrvatski dijakritički znakovi i razmaci i dobra je praksa ne koristiti ih u nazivima datoteka i foldera.

Protokoli za komunikaciju

Na internetu se koristi HTTP (engl. *HyperText Transfer Protocol*) za komunikaciju između klijenta i poslužitelja, odnosno HTTPS (engl. *HyperText Transfer Protocol Secure*) koji koristeći SSL/TLS (engl. *Secure Socket Layer/Transport Layer Security*) osigurava kriptiranu komunikaciju. Ovi su protokoli ugrađeni u *web*-poslužitelje i *web*-preglednike.

Komunikacija između *web*-preglednika i *web*-poslužitelja počinje klijentskim zahtjevom prema *web*-poslužitelju. Obično to znači unos adrese resursa (URL) u *web*-preglednik (Slika 1). Isto takav klijentski zahtjev se događa prilikom bilo kojeg klika na poveznicu na nekoj *web*-stranici.



Slika 1. Unos adrese resursa (URL) u web-preglednik (Izvor: autorica)

Klijentski HTTP zahtjev (engl. *request*) sadrži:

- Protokol i verziju protokola (HTTP/1.1 ili HTTP 2.0)
- HTTP metodu (GET, POST, PUT, DELETE, HEAD..)
- URL prema traženom resursu
- Zaglavlje s dodatnim informacijama o *web*-klijentu (User-Agent, Accept, Content-Type...)
- Ponekad tijelo metode (obično podaci uz POST zahtjeve).

Kad poslužitelj dobije zahtjev, on šalje HTTP odgovor (engl. *response*):

- Protokol i verziju protokola
- Kod i tekst statusa (200 OK, 10x informacije, 30x redirekcije, 40x greške s klijentske strane, 50x greške s poslužiteljske strane)¹
- Zaglavlje s dodatnim informacijama s poslužiteljske strane (Content-Type, Date, Keep-Alive, Server,...)

¹ <https://www.rfc-editor.org/rfc/rfc9110.html#name-overview-of-status-codes>

- Tijelo s traženim dokumentom

HTTP/1.1 je tekstualna i otvorena verzija protokola koji podatke šalje u čitljivom formatu. HTTP/2 radi performansi i optimizacije ima dodatan sloj mehanizma koji podatke dijeli i sažima u manje binarne okvire (engl. *frames*) (*HTTP Messages*, 2022.)

Pregled komunikacije klijenta i poslužitelja moguće je dobiti preko bilo kojeg *web*-preglednika. Obično u *web*-pregledniku postoji opcija Developer tools s mogućim pregledom mrežnog prometa (komunikacije između preglednika i poslužitelja), pregledom DOM i CSS strukture HTML dokumenta, ispisom JavaScript dijelova skripti na konzolni izlaz, praćenje performansi, debugger i slično. Na slici 2 prikazano je praćenje mrežnog prometa kod poziva i učitavanja dokumenata u *web*-preglednik.



Slika 2. Alat za razvijatelje aplikacija (Developer Tools) u *web*-pregledniku Mozilla Firefox (Izvor: autorica)

HTTP protokol izvorno je zamišljen za prijenos HTML (engl. *HyperText Markup Language*) dokumenata. Hipertekst je zapravo tekst koji služi kao poveznica na neki drugi resurs koji se nalazi negdje na *webu*, ne nužno na istoj lokaciji kao i dokument koji smo učitali.

HTML se sastoji od oznaka (engl. *tag*) koje *web*-preglednik može interpretirati i prikazati. Dokumenti pisani u HTML-u prikazuju sadržaj između HTML oznaka u *web*-pregledniku, tako da je on ljudima vizualno prihvatljiv, čitljiv i ugodan. HTML ima ograničen skup oznaka i namijenjen je definiranju strukture dokumenta i njegovom prikazu u *web*-pregledniku.

1.2 Formati za prijenos podataka

Klijentska *web*-aplikacija dinamički dohvaća podatke s poslužitelja. Za razmjenu podataka između klijenta i poslužitelja na *webu* često se koriste JSON i XML formati datoteka. Oba sustandardizirana formata su tekstualna i neovisna o programskim jezicima.

JSON (engl. *JavaScript Object Notation*) je standardan format zapisa za spremanje i razmjenu podataka, temeljen na sintaksi pisanja JavaScript objekta. Tekstualan je i jezično neovisan format zapisa koji se može koristiti u bilo kojem programskom jeziku, a sadrži parove *key: value*. Postoji nekoliko pravila za pisanje ovog formata datoteke:

- Ključevi (engl. *keys*) moraju biti zatvoreni unutar dvostrukih navodnika
- Parovi ključ - vrijednost (engl. *key-value*) moraju biti odvojeni dvotočkom
- Može biti više parova ključ - vrijednost, odvojenih zarezima
- Ovaj zapis ne dopušta komentare (Adhikary, 2021)

Primjer JSON zapisa (na istim podacima kao u prethodnom primjeru):

```
{
  "location" : "Rijeka",
  "date" : "18.11.2022",
  "forecasts" : [
    {
      "time" : "9:00",
      "forecast" : "sunny",
      "temp" : 10,
      "wind" : {
        "speed" : "1 m/s",
        "description" : "Light air from north-northeast "
      }
    }
  ]
}
```

Razmjena podataka u ovim formatima je uobičajena između međusobno neovisnih klijentskih i poslužiteljskih aplikacija, te je samo bitno kakve podatke poslužiteljska aplikacija može isporučiti i kakve podatke joj klijentska aplikacija treba pripremiti za unos.

XML (engl. *EXtensible Markup Language*) je označni jezik (engl. *markup language*) za pisanje XML dokumenata koji ima sljedeća sintakсна pravila (IBM *XML Syntax Rules*, 2022):

- XML elementi ili oznake su primarni gradivni blokovi dokumenata
- XML elementi nisu predefimirani
- Osjetljivi su na velika i mala slova (engl. *case sensitive*)
- XML elementi mogu sadržavati parove atribut="vrijednost", npr.

```
<line color="red">Thick line</line>
```

- Elementi moraju biti pravilno strukturirani, odnosno ugniježđeni, tako da se ranije otvoreni elementi kasnije zatvaraju
- Svaki otvoreni element mora biti i zatvoren
- Dokument mora imati *root* element
- Komentari su dozvoljeni u obliku `<!-- komentar -->`

Primjer XML zapisa:

```
<?xml version="1.0" encoding="UTF-8"?>
<root>
  <location> Rijeka </location>
  <date> 18.11.2022 </date>
  <forecasts>
    <time> 9:00 </time>
    <forecast> sunny </forecast>
    <temp> 10 </temp>
    <wind>
      <speed>1 m/s </speed>
      <description> Light air from north-northeast </description>
```

```
</wind>
</forecasts>
</root>
```

1.3 Troslojna arhitektura *web*-aplikacija

Arhitektura *web*-aplikacije općenito se temelji na troslojnom modelu:

- Prezentacijski sloj
- Poslovni ili aplikacijski sloj
- Podatkovni sloj.

Prezentacijski sloj je zadužen za prikaz i interakciju korisnika s aplikacijom. U *web*-arhitekturi radi se o sloju koji *web*-preglednik prikazuje korisniku. Tamo se nalaze i grafički elementi preko kojih korisnik može biti u interakciji s ostatkom aplikacije – bilo da se radi o elementima za prikaz i/ili elementima za unos, poveznicama i ostalim elementima. Ovaj sloj sadrži HTML i CSS elemente koji formatiraju, strukturiraju i definiraju dizajn dokumenta koji se prikazuje te JavaScript koji se koristi za dinamično renderiranje stranice.

Poslovni sloj je zadužen za osnovnu funkcionalnost aplikacije i sadrži poslovnu logiku aplikacije. On povezuje prezentacijski i podatkovni sloj, s obzirom na to da prezentacijski sloj ne može direktno pristupiti podatkovnom sloju. U poslovnoj logici se mogu koristiti JavaScript, PHP, Python, Java, C#, GoLang, Ruby, Kotlin, Scala i ostali programski jezici, odnosno njihove pripadne tehnologije i razvojne okoline.

Podatkovni sloj je vezan uz dohvat podataka s nekog izvorišta podataka (npr. baza podataka, datoteka, API...).

Ovako složena troslojna arhitektura ima tri glavne komponente:

- *web*-preglednik - klijentska komponenta koja je zadužena za prikaz i interakciju s korisnikom;

- *web*-poslužitelj – na njemu se nalaze *web*-stranice i ostali dokumenti koje poslužitelj dostavlja *web*-pregledniku na zahtjev. Ti dokumenti mogu se dostaviti takvi kakvi jesu, u izvornom obliku. Ako se u tom obliku dostavljaju HTML dokumenti, onda govorimo o statičnim *web*-stranicama. Međutim, *web*-poslužitelj može *web*-pregledniku dostaviti i dinamički generirane dokumente ili dijelove dokumenata ovisno o zahtjevu koji dolazi s klijentske strane, a koje će onda klijent prezentirati na ispravan način;
- poslužitelj baze podataka ili neki izvor podataka.

Ovaj općeniti model ima različite implementacije u praksi, ali postoje dvije bitne karakteristike ovih slojeva: prva je da se iz prezentacijskog sloja do podatkovnog može doći samo preko aplikacijskog sloja, a druga je da se aplikacijski sloj može dodatno raščlaniti, čime dobivamo višeslojnu arhitekturu.

Prezentacija krajnjem korisniku pripada razvoju klijentskog dijela *web*-aplikacije. Dohvat i spremanje podataka, odnosno općenito CRUD (engl. *create, read, update, delete*) operacije na podacima pripada poslužiteljskom dijelu *web*-aplikacije.

Za razvoj potpune *web*-aplikacije kombinira se razvoj klijentskog i poslužiteljskog dijela (Banga, 2022). Korištenje razvojnih okvira (engl. *frameworks*) olakšava razvoj *web*-aplikacija. Pri tom postoje poslužiteljski razvojni okviri (engl. *server-side frameworks, backend frameworks*) i klijentski razvojni okviri (engl. *client-side frameworks, frontend frameworks*).

Web-aplikacija vezana je uz dinamičko renderiranje HTML dokumenata – ovisno o zahtjevu *web*-klijenta (preglednika) *web*-aplikacija dobiva zahtjev, kreira sadržaj i šalje ga kao odgovor klijentu. Ovisno o načinu dohvata, kreiranja i prikaza sadržaja, postoji renderiranje s poslužiteljske strane (engl. *server-side rendering (SSR)*) i renderiranje s klijentske strane (engl. *client-side rendering (CSR)*). Renderiranje s poslužiteljske strane znači da se na zahtjev klijenta sav sadržaj prema *web*-pregledniku (klijentu) priprema na poslužiteljskoj strani i gotov šalje klijentu. Ovo znači više opterećenja na poslužiteljskoj strani, ovisno o broju klijenata te učestalosti zahtjeva za renderiranjem. Renderiranje s klijentske strane znači da se dio sadržaja dohvaća s poslužiteljske strane (obično su to samo podaci), a JavaScript s klijentske strane zaslužan je za dinamičko

postavljanje tog sadržaja u postojeći. Današnje *web*-aplikacije baziraju se na jednom od načina renderiranja, te se onda, ovisno o odabiru, odabiru prikladne tehnologije koje će osigurati razvoj potpune (engl. *full stack*) *web*-aplikacije.

Na klijentskoj strani JavaScript obavlja većinu aplikacijskog posla: spajanje na poslužiteljski dio aplikacije, dohvat podataka, obradu podataka na željeni način i prezentaciju podataka. Klijentski razvojni okviri Vue.js/Quasar, React, Ionic, Angular bazirani su na JavaScriptu.

Poslužiteljski dio *web*-aplikacije može se kreirati u bilo kojem programskom jeziku koji je podržan i može se izvršiti na poslužiteljskoj strani: PHP, Python, Java, JavaScript, Perl...

Poslužiteljski razvojni okviri (Laravel, CodeIgniter, Django, Flask, Node.js...) razvijaju *web*-aplikacije koje cijelu aplikacijsku logiku odrađuju s poslužiteljske strane i rade renderiranje dokumenata s poslužiteljske strane ili se u njima izrađuju *web*-servisi s aplikacijskim programskim sučeljima (API).

2 Priprema radnog okruženja za razvoj *web*-aplikacija

U ovom poglavlju prikazana je instalacija i konfiguracija alata za razvoj klijentskog dijela *web*-aplikacije. Pripremljeni su sljedeći alati u radnom okruženju:

- VisualStudio Code (VS Code) koristi se za pisanje programskog koda
- Lokalni i udaljeni Git repozitorij za čuvanje verzija aplikacije
- Node i Quasar – JavaScript razvojni okvir za izradu klijentskog dijela *web*-aplikacije.

2.1 Instalacija i konfiguracija alata za razvoj *web*-aplikacije

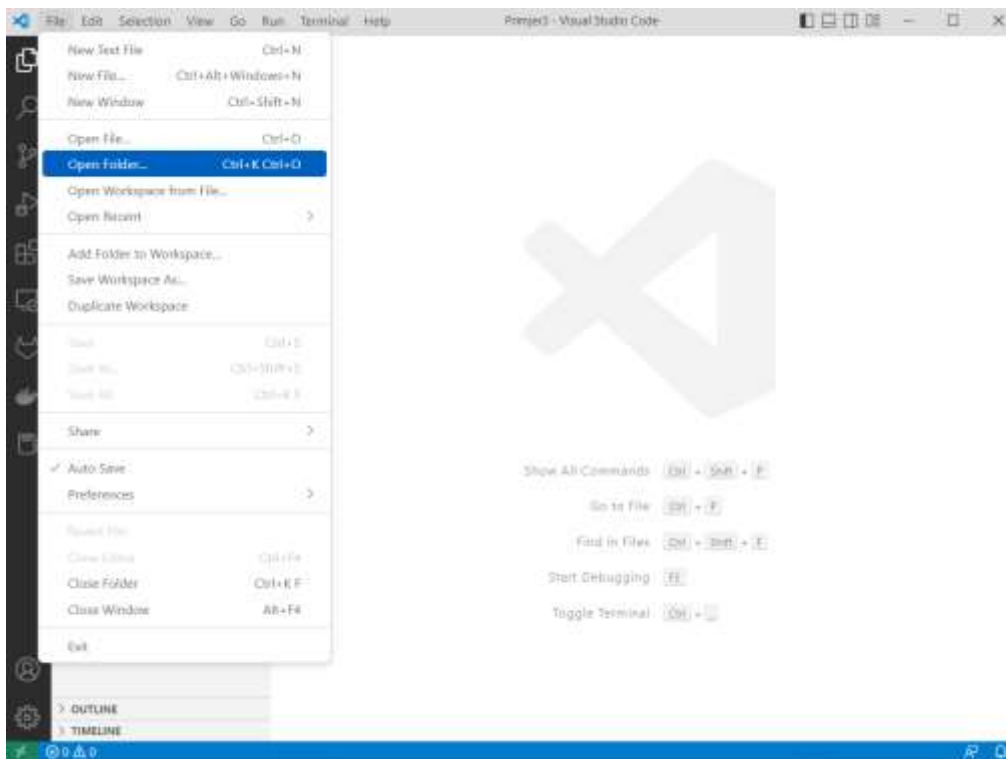
Za pisanje programskog koda obično se koristi IDE (engl. *Integrated Development Environment*) koji ima funkcionalnosti koje programeru olakšavaju razvoj aplikacije: integrirane dijelove za pregled i pokretanje aplikacije, označavanje i formatiranje različitih dijelova programskog koda, integrirane alate za otkrivanje grešaka, mogućnost jednostavne instalacije dodatnih biblioteka, podršku za više programskih jezika, mogućnost poziva vanjskih naredbi i ostalo (Microsoft, 2021). VS Code je trenutno jedan od popularnijih uređivača programskog koda, besplatan i dostupan za macOS, Linux i Windows. Preuzeti se može verzija koja se automatski instalira ili zapakirana verzija u .zip formatu, koja se može raspakirati na bilo kojem računalu i koristiti bez dodatne instalacije.

Posljednja verzija može se preuzeti na:

<https://code.visualstudio.com/download>

Prije pokretanja novog projekta treba kreirati novi folder te ga zatim otvoriti u VSCode IDE-u. Na slici 3 prikazana je opcija otvaranja foldera unutar VSCode alata.

Unutar alata može se uključiti opcija Auto Save koja automatski sprema sve izmjene napravljene u programskom kodu.



Slika 3. Visual Studio Code editor - opcija otvaranja foldera (Izvor: autorica)

Pregled svih datoteka u folderu nalazi se s lijeve strane.

Otvaranjem novog terminala, s donje strane prozora (slika 4) pojavljuje se dio koji se može koristiti za izvršavanje naredbi u komandnoj liniji.

Unutar terminal prozora ponuđene su dvije opcije: Command Prompt i PowerShell.

2.2 Instalacija alata za čuvanje verzija aplikacije

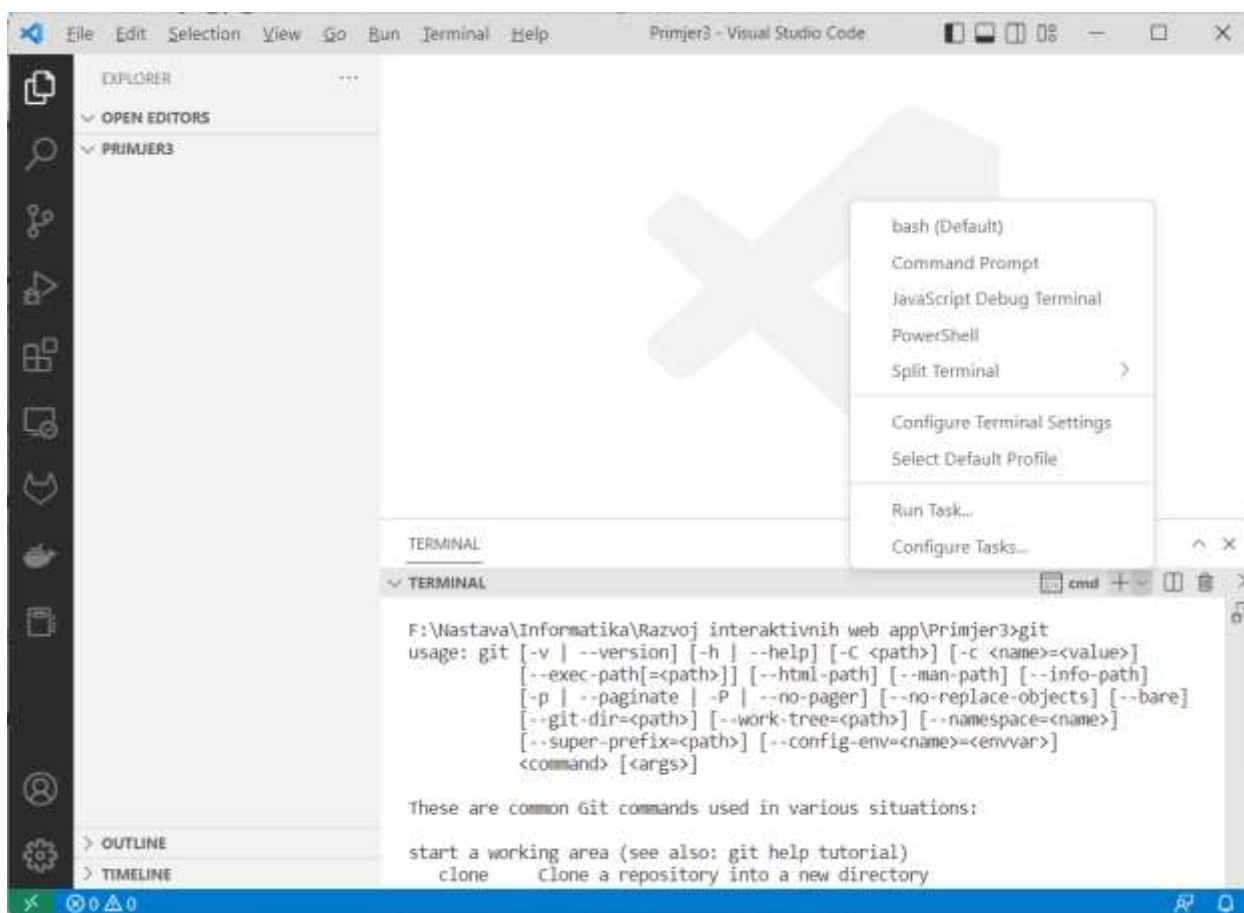
Čuvanje različitih verzija aplikacije moguće je korištenjem nekog od sustava za kontrolu i upravljanje verzijama programskog koda (engl. *version control system*). Trenutno se najčešće koristi Git (engl. *Global Information Tracker*). To je distribuirani sustav otvorenog koda (engl. *open source*) koji omogućuje praćenje promjena u programskom kodu kroz određeni vremenski

period te upravljanje i rad s različitim verzijama (Chacon & Straub, 2014). Može se koristiti na lokalnom računalu, a preuzeti se može s poveznice:

<https://git-scm.com/downloads>

Na stranici se može odabrati automatska instalacija ili prenosiva verzija koja se samo raspakira na računalo.

S Git sustavom se radi preko naredbenog retka, odnosno CLI (engl. *command-line interface*). Git instalacija dolazi s *bash* jezikom (engl. *Bourne Again SHell*) za naredbeni redak, pa ako se kod instalacije nudi kao opcija, trebalo bi je izabrati.



Slika 4. Popis programskih jezika za naredbeni redak i izvršavanje naredbe git u terminalu (Izvor: autorica)

Testiranje instalacije radi se upisivanjem **git** naredbe u terminal VS Code alata. Ako je sve prošlo u redu, Git bi trebao izlistati popis svojih naredbi koje se mogu pozvati iz naredbenog retka (slika 4). Također se nakon restarta VS Code alata može provjeriti postoji li nakon otvaranja terminala *bash* opcija. Na slici 4 otvoren je skočni prozor s popisom dostupnih programskih jezika za naredbeni redak.

Git se koristi na različitim platformama koje nude uslugu udomljavanja (engl. *hosting*) centralnih repozitorija, te obično nude uz to i druge usluge poput praćenja i upravljanja projektima i zadacima, praćenja i prijave problema i slično. Poznatije platforme su: GitLab (GitLab Inc.), GitHub (Microsoft), BitBucket (Atlassian), SourceForge (Slashdot Media), Launchpad (Canonical), AWS CodeCommit (Amazon).

Za korištenje GitHub udaljenog repozitorija potrebno je imati kreiran korisnički račun. Nakon toga se na GitHubu može kreirati repozitorij. Kad se pripremi lokalno razvojno okruženje, treba kopirati poveznicu s udaljenog repozitorija koja će ga klonirati u lokalni repozitorij:

```
git@github.com:korisnik/repozitorij.git
```

Kloniranje udaljenog repozitorija na lokalni repozitorij opisuje se u poglavlju *Repozitoriji i alati za uvanje verzija softvera*.

2.3 Instalacija i konfiguracija softverskog razvojnog okvira za razvoj klijentskog dijela *web*-aplikacije

Instalacija softverskog razvojnog okvira Quasar.js počinje preuzimanjem Node.js sa stranice:

<https://nodejs.org/en/download/>

Preuzeti se može verzija koja se automatski instalira na željenom OS-u ili zapakirana verzija u .zip formatu koja se može raspakirati na bilo kojem računalu i koristiti bez dodatne instalacije.

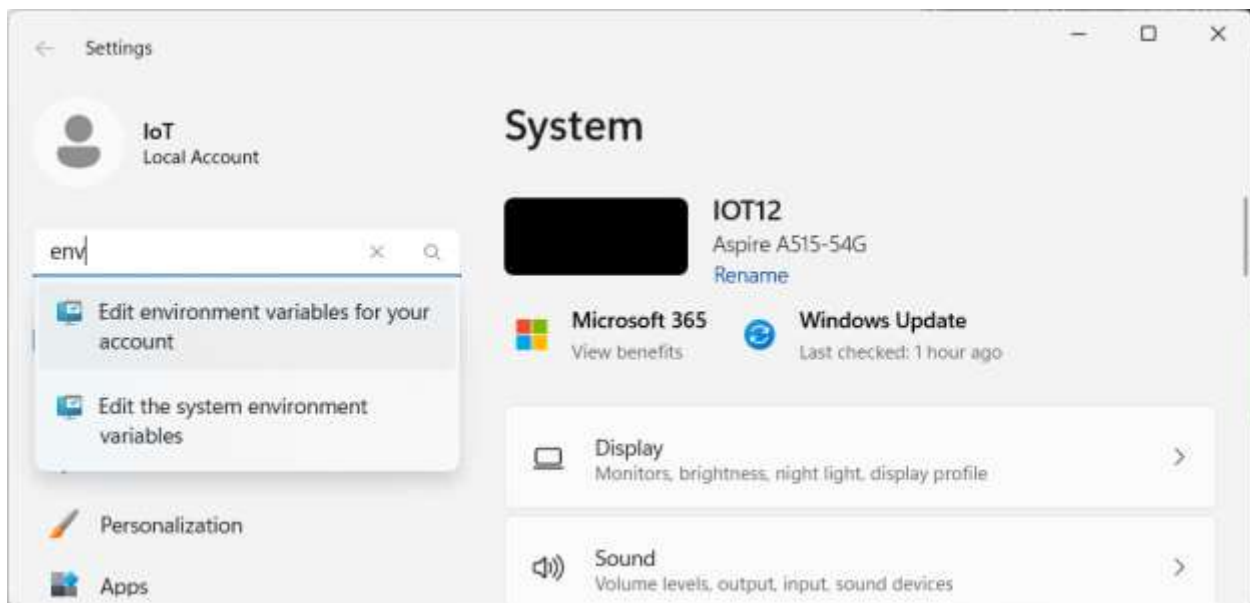
Nakon što se Node.js preuzme i instalira ili raspakira, treba provjeriti ako je smješten u PATH varijablu. Unutar VS Code terminala treba upisati **node**. Ako se dobije poruka kao na slici 5, znači

da se node može koristiti bez dodatnih aktivnosti. Sa Ctrl + C se izlazi iz Node konzole. U protivnom, treba definirati putanju do node.exe programa.

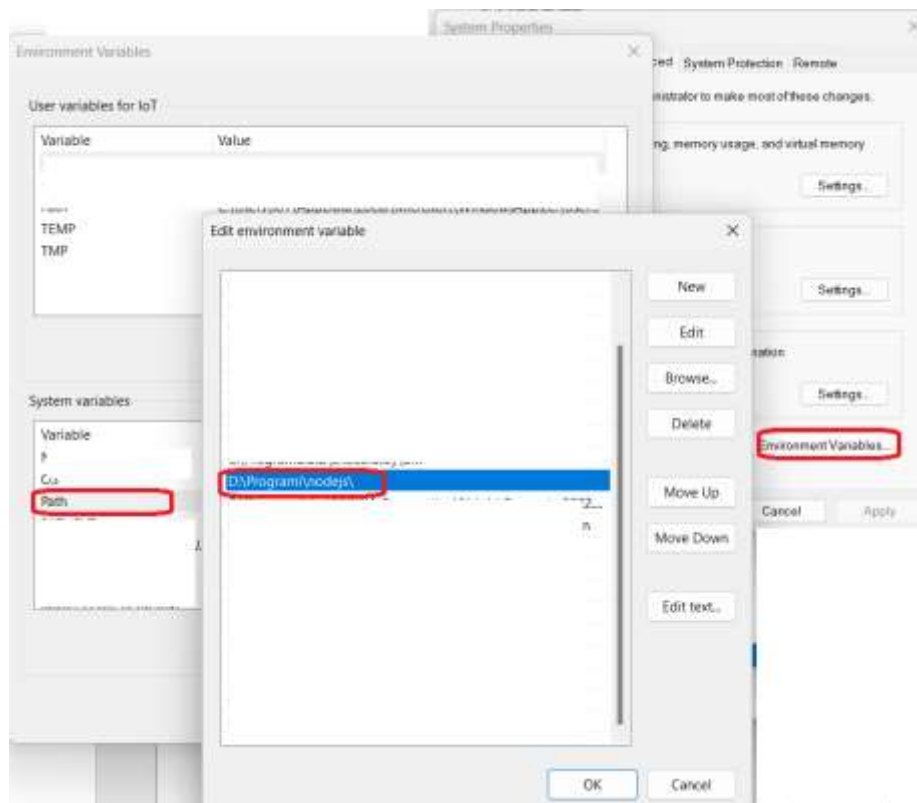


Slika 5. Pokretanje naredbe node u terminalu Visual Studio Code alata (Izvor: autorica)

Do node.exe se definira sistemska varijabla (engl. *environment variable*) PATH, odnosno treba kopirati cijelu putanju do datoteke node.exe, otvoriti Settings (na Windows OS-u, prikazano na slici 6), potražiti sistemske varijable sustava i unutar Path varijable ubaciti putanju do node.exe (slika 7).



Slika 6. Traženje system environment varijable u Windows OS-u (Izvor: autorica)



Slika 7. Uređivanje sistemske varijable Path (Izvor: autorica)

Nakon toga, treba ponovno pokrenuti VS Code.

2.4 Rad s upraviteljem paketa

Node.js dolazi u paketu s programom npm (engl. *Node Package Manager*). Program npm je upravitelj paketa za Node, odnosno instalira, ažurira ili briše sve potrebne JavaScript biblioteke i razvojne okvire, vodeći računa o njihovim međuovisnostima (engl. *dependency*) i verzijama (*Node.js V17.9.1 documentation, 2022*).

S upraviteljem paketa se radi kroz komandnu liniju (terminal), te se unosom naredbe **npm** dobije popis svih naredbi koje se koriste uz njega:

npm

Usage: npm <command>

where <command> is one of:

access, adduser, audit, bin, bugs, c, cache, ci, cit, clean-install,
clean-install-test, completion, config, create, ddp, dedupe, deprecate,
dist-tag, docs, doctor, edit, explore, fund, get, help, help-search,
hook, i, init, install, install-ci-test, install-test, it, link, list,
ln, login, logout, ls, org, outdated, owner, pack, ping, prefix, profile,
prune, publish, rb, rebuild, repo, restart, root, run, run-script, s, se,
search, set, shrinkwrap, star, stars, start, stop, t, team, test, token,
tst, un, uninstall, unpublish, unstar, up, update, v, version, view,
whoami

npm <command> -h quick help on <command>

npm -l display full usage info

npm help <term> search for help on <term>

npm help npm involved overview

Instalacija se radi korištenjem naredbe

```
npm install naziv_paketa
```

a popis svih trenutno instaliranih paketa:

```
npm list
```

2.5 Instalacija Quasar razvojnog okvira i kreiranje projekta.

2.5.1 *Rad na novom projektu*

Za projekt u Quasar razvojnem okviru potrebno je kreirati folder i pozicionirati se na njega.

Quasar razvojni okvir zahtijeva postojanje npm (Node Package Manager) koji dolazi s Node.js paketom.

Ako se radi **novi projekt** u Quasaru, prvo se radi instalacija Quasar razvojnog okvira unutar Visual Studio Code terminala:

```
npm i -g @quasar/cli
```

a zatim inicijalizacija (početno postavljanje) projekta

```
npm init quasar
```

Prilikom inicijalizacije u koracima se odabiru:

```
ESLint  
Axios  
Vue-i18n
```

Ostalo se uglavnom ostavlja sve predloženo (samo Enter za dalje). ESLint je alat za analizu JavaScript programskog koda koji nalazi problematične dijelove u kodu. Axios je JavaScript biblioteka koja radi s HTTP zahtjevima i odgovorima na asinkron način. Vue-i18n je dio Vue.js razvojnog okvira koji podržava internacionalizaciju (*i18n* je skraćenica koja sadrži prvo i početno slovo naziva *internationalization*, a između njih je još 18 znakova).

Za pokretanje Quasara treba se pozicionirati unutar Quasar projekta:

```
cd quasar-project
```

Quasar se pokreće naredbom

```
quasar dev
```

Nakon pokretanja treba otvoriti *web*-preglednik i unijeti URL.

2.5.2 T c f " p c " r q u v q l g g o " r t q l g m v w

Ako se radi na **postojećem projektu** (dohvaćenom npr. s GitHuba), potrebno je napraviti instalaciju modula Node paketa. Oni se nalaze u `node_modules` folderu. Obično se taj folder ne stavlja na GitHub, nego samo `package.json` datoteka, koja sadrži popis aktualnih Node.js paketa

koje treba instalirati prije bilo kakvog rada na projektu. Prvo se treba pozicionirati unutar Quasar projekta. Naredbe se izvršavaju unutar Visual Studio Code terminala,

```
cd quasar-project
```

a zatim se radi instalacija:

```
node install
```

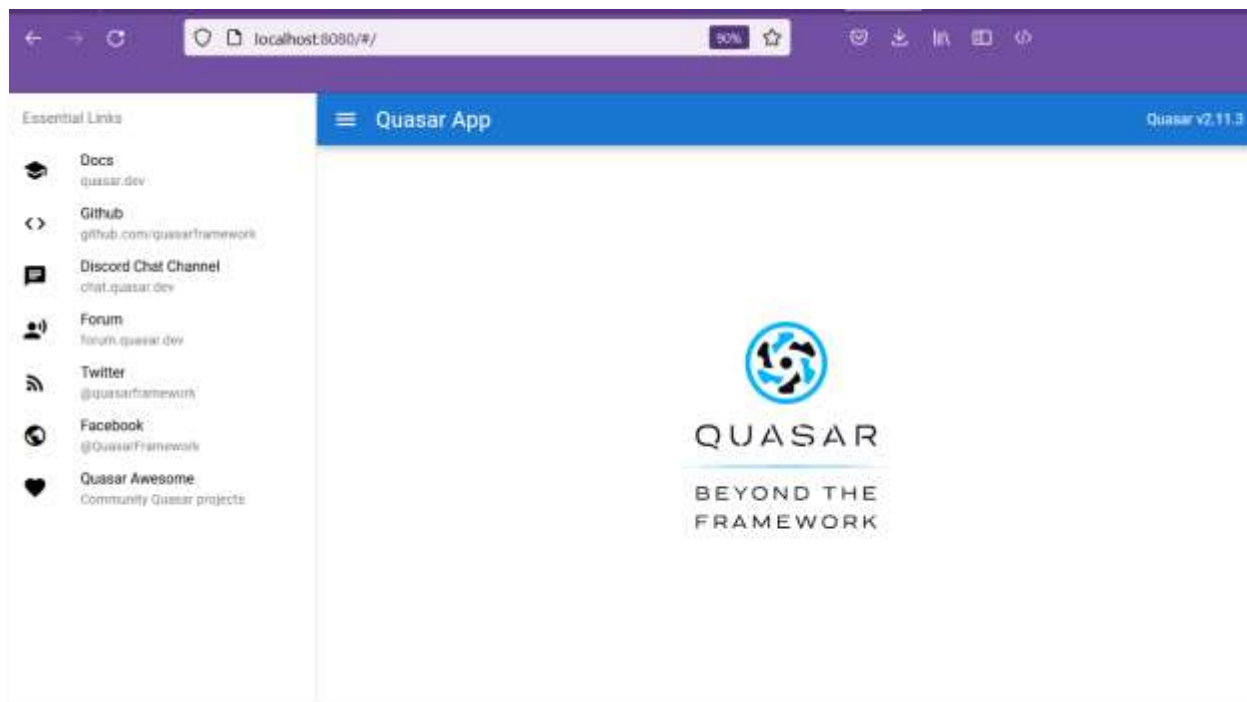
Quasar se pokreće naredbom:

```
quasar dev
```

Nakon pokretanja treba otvoriti *web*-preglednik i unijeti adresu:

```
http://localhost:8080/#/
```

Otvora se početna stranica Quasar razvojnog okvira (slika 8).



Slika 8. Po etna stranica Quasar razvojnog okvira (Izvor: autorica)

3 Repozitoriji i alati za čuvanje verzija softvera

Sustav za kontrolu verzija izvornog programskog koda (engl. *version control or source control system*) definira način čuvanja, praćenja promjena i rada s različitim verzijama izvornog programskog koda. Kad se neki dio programskog koda doda ili izmijeni, sustav kreira novu verziju programskog koda. Sustav čuva i prati sve izmjene koje su se dogodile između svih prethodnih i trenutne verzije.

Sustavi za kontrolu verzija dijele se na lokalne, centralizirane i distribuirane.

Lokalni sustavi za kontrolu verzija čuvaju sve verzije programskog koda na lokalnom računalu.

Centralizirani sustavi za kontrolu verzija rade tako da se sav kod postavlja na centralni poslužitelj. Svaki član tima može napraviti promjenu i prilagodbu izvornog programskog koda koji razvija i postaviti ga direktno na poslužitelj.

Distribuirani sustavi za kontrolu verzija su decentralizirani. Sav programski kod je smješten na centralnom poslužitelju, ali također svaki član tima ima i svoj lokalni repozitorij. Kod preuzimanja repozitorija klijenti kompletno kloniraju centralni repozitorij, zajedno s povijesnim podacima, čime se radi i *backup* podataka.

3.1 Kreiranje repozitorija

Repozitorij je mjesto na kojem se čuvaju različite verzije projekta, a obično obuhvaća folder i datoteke unutar njega. Kad se kreira novi projekt u novom folderu, Git repozitorij se može kreirati (inicijalizirati) kroz naredbeni redak naredbom

```
git init
```

Unutra se kreira folder `.git` u kojem se sprema sva povijest izmjena.

`.gitignore` je datoteka u koju se smještaju lokacije datoteka za koje ne želimo pratiti povijest verzija.

Git sprema stanja datoteka. U repozitoriju datoteke mogu biti u nekoliko stanja:

- *untracked* – datoteke koje se ne prate
- *tracked* – datoteka je u stanju praćenja, potvrđena i nema izmjene od zadnje potvrde
- *modified* – datoteka je modificirana, a da bi bila potvrđena (*commit*) mora biti u stanju pripreme (*staged*)
- *staged* – datoteka je u međuprostoru, u stanju pripreme za potvrdu (*commit*) ali još nije potvrđena.

Jednom kad je repozitorij inicijaliziran, mogu se u njega dodavati datoteke s:

```
git add *  
git commit -a -m 'Initial commit.'
```

Naredba *git add ** dodaje sve datoteke iz postojećeg foldera u repozitorij (* zamjenjuje sve nazive datoteka).

Naredba *git commit -a* stavlja sve izmjene u stanje *stage*, a *-m* radi potvrdu izmjena uz dodavanje poruke. Obično je poruka vezana uz objašnjenje neke napravljene izmjene (Chacon & Straub, 2014).

3.2 Osnovne naredbe u radu s repozitorijem

U lokalnom repozitoriju:

git status - vraća status datoteka

git add datoteka - dodaje datoteku u *staged* stanje i pripremljena je za *commit*

git commit -m „ O p i s v e r p o t v r d u j e ” potvrđuje novu verziju izmjena u repozitoriju i dodaje opis verzije

Projekt inicijalno ima jednu *master* granu (engl. *branch*). Ako se radi na izmjenama programskog koda za koje ne želimo da narušavaju dosadašnje stanje, radimo novu granu. Može se raditi o novoj funkcionalnosti koju razvijamo i želimo je spojiti s master granom tek kad bude gotova.

Kreiranje nove grane:

```
git checkout -b new_branch
```

Pozicioniranje na željenu granu (npr. *new_branch*):

```
git switch new_branch
```

ili

```
git checkout new_branch
```

Nakon što se na granu dodaju željene izmjene, radi se stavljanje datoteka u staged stanje

```
git add *
```

Potvrda izmjena za lokalni repozitorij, s tim da se u opis stavlja opis napravljenih izmjena:

```
git commit -m "Opis izmjena"
```

Kad je neka funkcionalnost dovršena, ako se želi napraviti spajanje tih izmjena s glavnom granom, prvo se radi pozicioniranje na glavnu granu:

```
git checkout master
```

i zatim se dodaju izmjene s *new_branch* grane:

```
git merge new_branch
```

Grana s izmjenama više ne treba postojati, pa se može obrisati:

```
git branch -d new_branch
```

3.3 Rad s udaljenim repozitorijem

Rad s udaljenim repozitorijem podrazumijeva korisnički račun s kojim se onda spaja na udaljeni repozitorij. Na GitHubu se može kreirati repozitorij, te se kopirati poveznica koja će ga klonirati u lokalni repozitorij:

```
git@github.com:korisnik/repozitorij.git
```

Kloniranje udaljenog repozitorija na lokalni (Chacon & Straub, 2014):

```
git clone git@github.com:korisnik/repozitorij.git
```

Sve izmjene rade se unutar lokalnog repozitorija te se potvrđuju. Kad se one trebaju prebaciti s lokalnog repozitorija na udaljeni radi se:

```
git push
```

Ova naredba sinkronizira potvrđene izmjene na lokalnom repozitoriju s udaljenim repozitorijem.

Ako treba s udaljenog repozitorija u lokalni dohvatiti izmjene, ažuriranje lokalnog repozitorija radi se naredbama:

```
git fetch
```

i

```
git pull
```

git fetch dohvaća samo datoteku koja sadrži popis posljednjih izmjena, a *git pull* dohvaća sve izmijenjene datoteke (GitHub About Git, n.d.).

3.4 Rad s granama repozitorija u timskom okruženju

Za rad u timskom okruženju potrebno je definirati tijek rada na Git sustavu (engl. *Git workflow*), s obzirom na to da se paralelno radi na razvoju aplikacije. Tijek rada nije specificiran nekim

standardom, nego većinom ovisi o dogovoru unutar tima i pojedinoj situaciji. Najčešće se dogovara ili zadaje prije početka rada na projektu.

Jedan od načina na koji se može definirati tijek rada i grananje je postojanje sljedećih grana:

- Main ili master – odražava stanje produkcije
- Staging – testiranje prije produkcije
- Development - su zadnje izmjene u razvoju, s glavnim integracijama programskog koda.

Mogu se definirati i *feature* grane na kojima neki tim radi na nekoj funkcionalnosti koja se spaja s development granom kad je gotova.

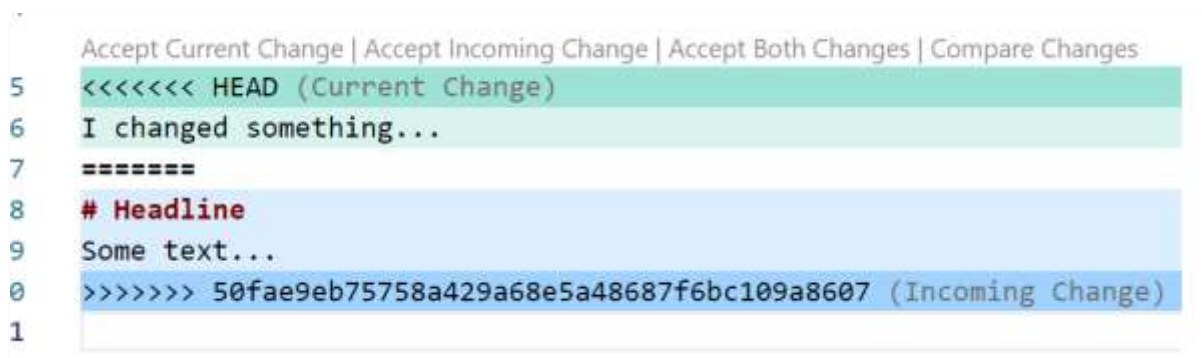
Ako se radi na development grani, prvo se lokalno dohvate sve izmjene koje su u međuvremenu napravljene na udaljenom repozitoriju (pull) na development grani:

```
git pull development development
```

Zatim se lokalno spajaju vlastite izmjene s dohvaćenim:

```
git add *  
git commit -m "Opis novih izmjena"
```

Ako se pojave konflikti (primjer na slici 9) treba ih riješiti.



```
Accept Current Change | Accept Incoming Change | Accept Both Changes | Compare Changes  
5 <<<<<<< HEAD (Current Change)  
6 I changed something...  
7 =====  
8 # Headline  
9 Some text...  
0 >>>>>> 50fae9eb75758a429a68e5a48687f6bc109a8607 (Incoming Change)  
1
```

Slika 9. Prikaz konfliktne situacije prilikom dohvata izmjena s git pull naredbom (Izvor: autorica)

Na istoj datoteci mogu se pojaviti izmjene od nekog drugog člana tima, pa treba pregledati izmjene i provjeriti nisu li u konfliktu s vlastitim programskim kodom. Izmjene se mogu prihvatiti, modificirati ili odbaciti.

Nakon rješavanja konflikta, datoteke se mogu ponovno postaviti u *staged* stanje, nakon čega se potvrđuju izmjene:

```
git add *  
git commit -m "Opis novih izmjena"
```

Redovito bi trebalo raditi dohvat izmjena iz *development* grane iz udaljenog repozitorija u lokalni.

Postavljanje svih izmjena na udaljeni repozitorij:

```
git push origin development
```

4 Programski jezici i softverski razvojni okviri za razvoj klijentskog dijela *web*-aplikacije

4.1 Osnovni koncepti klijentskih *web*-tehnologija

4.1.1 HTML

HTML (engl. *HyperText Markup Language*) označni jezik za opisivanje strukture *web*-dokumenta koji se prikazuje unutar *web*-preglednika. Hipertekst ili poveznica je pri tom označeni tekst koji se povezuje s resursom koji se nalazi javno dostupan na bilo kojoj lokaciji na internetu. Osnova HTML jezika je predefimirani skup oznaka (engl. *tag*), između kojih je ubačen sadržaj dokumenta. *Web*-preglednici interpretiraju i prikazuju HTML dokument.

Osnovna struktura HTML dokumenta je:

```
<!DOCTYPE html>
<html>
  <head>
    <title> Naslov dokumenta </title>
  </head>
  <body>
    S a d r ž a j   v i d l i c i   u   w e b - p r e g l e d n i c i m a .
  </body>
</html>
```

HTML 5 je dio W3C standarda koji definira da se HTML oznake pišu na sljedeći način:

- Oznake se pišu malim slovima te svaka oznaka ima početak `< >` i završetak `</ >`. Na primjer:

```
<div> Tekst </div>
```

- Oznake su ugniježdene. To znači, da se prethodno otvorena oznaka kasnije zatvara. Na primjer:

```
<div>Tekst <b> je podebljan i <u>podcrtan</u> </b> </div>
```

daje rezultat:

Tekst **je podebljan i podcrtan**

- Oznaka na početku dokumenta

```
<!DOCTYPE html>
```

označava da je dokument pisan u HTML 5 standardu

- Oznake

```
<html> </html>
```

su obavezne na početku i kraju HTML dokumenta

- Zaglavlje

```
<head></head>
```

HTML dokumenta šalje dodatne informacije *web*-pregledniku.

- Unutar oznaka

```
<body></body>
```

nalazi se sadržaj koji se prikazuje u *web*-pregledniku krajnjim korisnicima.

- Zatvaranje HTML oznaka je obavezno, osim u nekoliko slučajeva, poput

```
<hr>, <br>, <img>, <button>
```

Ove HTML oznake ne opisuju dio dokumenta, pa nemaju zatvarajuće dijelove.

Od ostalih osnovnih HTML oznaka mogu se izdvojiti:

- oznake za naslove koje definiraju naslove (engl. *heading*) dokumenta

<h1></h1>... <h6></h6>

- oznake za odlomke koje definiraju odlomke (engl. *division, paragraph*) dokumenta

<div></div>, <p></p>

- oznaka za dio teksta unutar odlomka dokumenta

- oznake za formatiranje teksta:

 oznaka koja razlama tekst u novi red (engl. *brake*)

 oznaka koja podebljava (engl. *bold*) dio teksta

 oznaka koja naglašava (engl. *strong*) dio teksta

<i></i> oznaka koja ukošava (engl. *italic*) dio teksta

<u></u> oznaka koja podcrtava (engl. *underline*) dio teksta

<hr> oznaka koja iscrtava ravnu liniju (engl. *horizontal rule*)

- oznaka za postavljanje slike (engl. *image*) u dokumentu:

- oznaka za postavljanje sidra (engl. *anchor*):

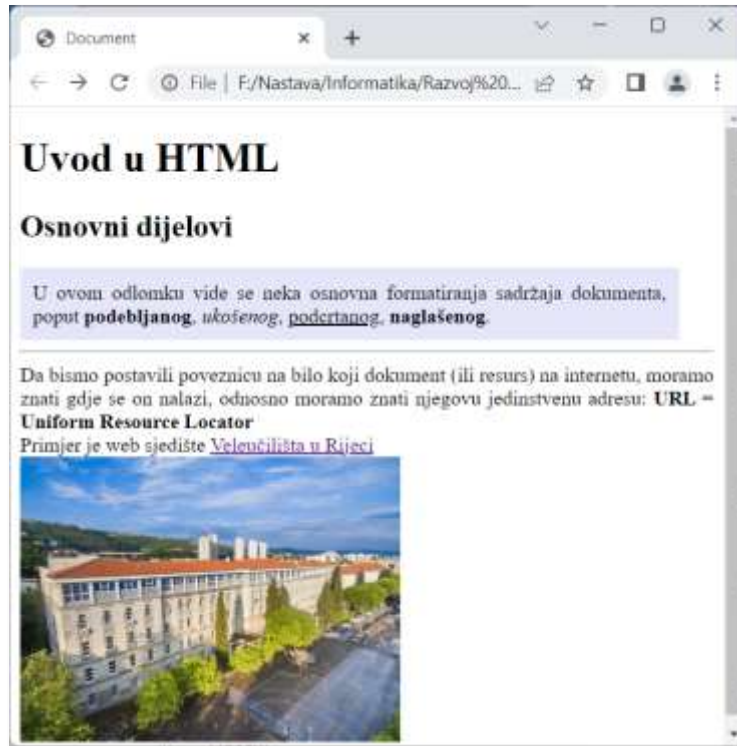
<a>

Uz oznaku <a> vezuje se atribut href koji označava poveznicu ili hiperpoveznicu (engl. *hyperlink*) koja vodi do drugog resursa na internetu (engl. *anchor for hypertext reference*).

Primjer jednostavnog HTML dokumenta:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-
    scale=1.0">
  <title>Document</title>
</head>
<body>
  <h1>Uvod u HTML</h1>
  <h2>Osnovni dijelovi</h2>
  <div style="background-color: lavender; inline-size: 500px;
    padding: 10px; text-align: justify;">
    U ovom odlomku vide se neka osnovna formatiranja
    dokumenta, poput <b>podebljanog</b>, <i>ukošenog</i>,
    <u>podcrtanog</u>, <strong>naglašenog</strong>. </div>
  <hr>
  <div style="text-align: justify;">Da bismo postavili poveznicu na
    bilo koji dokument (ili resurs) na internetu, moramo znati gdje se
    on nalazi, odnosno moramo znati njegovu jedinstvenu adresu: <b>
    URL = Uniform Resource Locator </b> <br> Primjer je <b>web-sjedište</b>
    <a href="http://www.veleri.hr/"> Veleučilišta u
  </div>
  <div> 
  </div>
</body>
</html>
```

Dokument se sprema s nastavkom .html ili .htm. Izgled napravljenog dokumenta – prikaz u *web*-pregledniku dan je na slici 10.



Slika 10. Izgled jednostavnog HTML dokumenta prikazanog u web-pregledniku (Izvor: autorica)

Uz svaku HTML oznaku mogu se pisati atributi. Atributi obično dolaze u paru s vrijednostima: atribut="vrijednost", pri čemu se vrijednosti upisuju unutar jednostrukih ili dvostrukih navodnika.

Osnovni atributi koji se mogu postaviti uz bilo koju HTML oznaku su: alt, title, id, class, style

alt alternativni tekst koji ide obično uz sliku kad slika ne može biti prikazana

title dodatni tekst koji se pojavljuje kao oblačić iznad elementa

style atribut koji se koristi za stiliziranje elementa CSS-om

`id`, `class` atributi koji se koriste za stiliziranje (CSS) ili dinamizaciju (Javascript) elementa.

Atribut koji se koristi uz sidro (oznaka `<a>`):

`href` definira se URL (*web-adresa*) poveznice ili hiperpoveznicu (engl. *hyperlink*) koja vodi do drugog resursa na internetu (engl. *anchor for hypertext reference*).

Atributi koji se koriste uz sliku:

`src` definira URL (*web-adresa*) slike

`width` definira se širina prikazane slike.

URL ili *web-adresa* je jedinstvena adresa lokacije nekog resursa (HTML datoteke, slike, generiranog zapisa...) na internetu. Može se koristiti kao apsolutna ili relativna adresa.

Relativna adresa resursa odnosi se na lokaciju resursa lokalno, u odnosu na trenutnu datoteku (Morris, 2017).

Od dodatnih HTML elemenata koji rade neku interakciju s korisnicima su: elementi za unos teksta, odabir opcija te gumbi za potvrdu.

Uglavnom se radi o HTML elementima `<input>` i `<button>`, s tim da `input` može imati više mogućih izgleda, ovisno o atributu `type` koji se uz njega nalazi.

`<input type="text">` daje polje za unos jedne linije teksta

`<input type="radio">` daje mogućnost odabira (radio button)

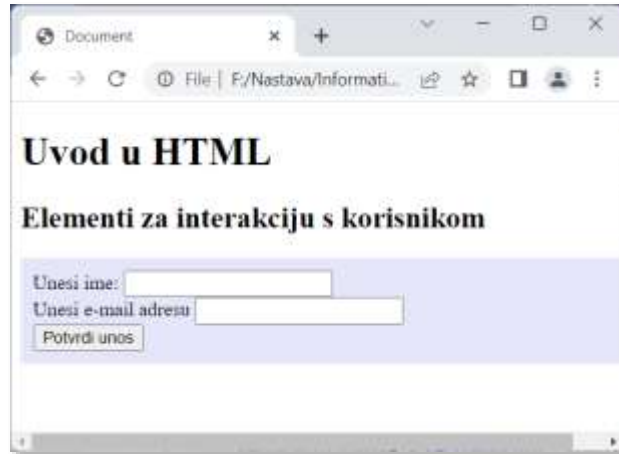
`<input type="submit">` daje gumb za potvrdu

`<button>` običan gumb

Primjer elemenata za unos teksta:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-
    scale=1.0">
  <title>Document</title>
</head>
<body>
  <h1>Uvod u HTML</h1>
  <h2>Elementi za interakciju s korisnikom</h2>
  <div style="background-color: lavender; inline-size: 500px;
    padding: 10px; text-align: justify;">
  <div>
    <label>Unesi ime:</label> <input type="text" name="name"
      id="name">
  </div>
  <div>
    <label>Unesi e-mail adresu</label> <input type="email"
      name="email" id="email">
  </div>
  <div>
    <input type="button" value="Potvrđi unos">
  </div>
</div>
</body>
</html>
```

Izlazni rezultat prikazan je na slici 11.



Slika 11. Prikaz HTML elemenata za unos teksta (Izvor: autorica)

Interakcija s korisnicima zahtjeva praćenje onog što korisnici unose ili kliknu, pa s ovim komponentama obično dolazi i obrada događaja kroz Javascript.

4.1.2 CSS

CSS (engl. *Cascading Style Sheets*) je skup stilskih oznaka kojima se grafički dizajniraju HTML elementi. CSS nudi dizajn, odnosno grafičko oblikovanje *web*-dokumenta koji je odvojen od njegovog sadržaja i strukture. Navedenim odvajanjem dizajna od sadržaja dobivena je veća fleksibilnost u primjeni istog dizajna na više dokumenata strukturiranih u HTML-u, kao i mogućnost razvoja HTML i CSS standarda odvojeno. Trenutno se brže razvija CSS standard, što kao posljedicu ima uvođenje više različitih stilskih elemenata.

CSS se sastoji od naziva stila (selektora) te jednog ili više atributa s vrijednostima. Primjer:

```
.naslov {  
    color: red;  
}
```

naslov je naziv stila, *color* je atribut, *red* je vrijednost.

CSS se može primijeniti na HTML elemente na tri načina:

1. CSS stilovi se izdvajaju u zaseban dokument koji se može pozvati kao vanjski dokument u pojedinim HTML dokumentima, te se tako ujednačeni dizajn može primijeniti na cijelo *web*-središte. Pozivanje vanjskog CSS dokumenta radi se unutar zaglavlja HTML dokumenta.

Unutar `<head>` elementa stavlja se poveznica na vanjski CSS dokument:

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

2. CSS stilovi se mogu pisati unutar HTML dokumenta – na početku dokumenta unutar `<head></head>` oznaka i primjenjivi su samo na taj dokument. Postavljaju se unutar `<style></style>` elementa.
3. CSS stilovi se pišu unutar pojedinog HTML elementa da bi stilizirali samo taj element. Unutar pojedinog HTML elementa koristi se atribut `style`. Ovaj način pisanja predstavljen je unutar poglavlja o HTML-u.

U prva dva slučaja, kad stilove ne ubacujemo direktno u element, nazive stilova (selektora) pišemo i pozivamo na jedan od načina (Gupta, 2022):

- kao jednostavne selektore – imaju naziv HTML elementa (u primjeru je dan element *table*)
- kao klasne selektore – imaju proizvoljan naziv, a pozivaju se `class` atributom. Unutar HTML dokumenta poziva se `class="tabl_vrem_grad"`, a u CSS dokumentu element je definiran kao `.tabl_vrem_grad`
- kao id selektore – imaju proizvoljan naziv, a pozivaju se `id` atributom. Unutar HTML dokumenta poziva se kao `id="t"`, a u CSS dokumentu element je `#t { }`.

Primjer CSS datoteke stil.css:

```
table {
  border:solid 1px;
  text-align: center;
  margin: 10px;
  padding: 10px;
  background-color: azure;
  inline-size: 500px;
}
.tabl_vrem_grad tr {
  border:solid 1px;
}
.tabl_vrem_grad th {
  border:solid 1px;
  background-color: cadetblue;
  margin: 20px;
}
.tabl_vrem_grad td {
  background-color: lightblue;
}
```

Primjer poziva vanjske CSS datoteke u HTML dokumentu:

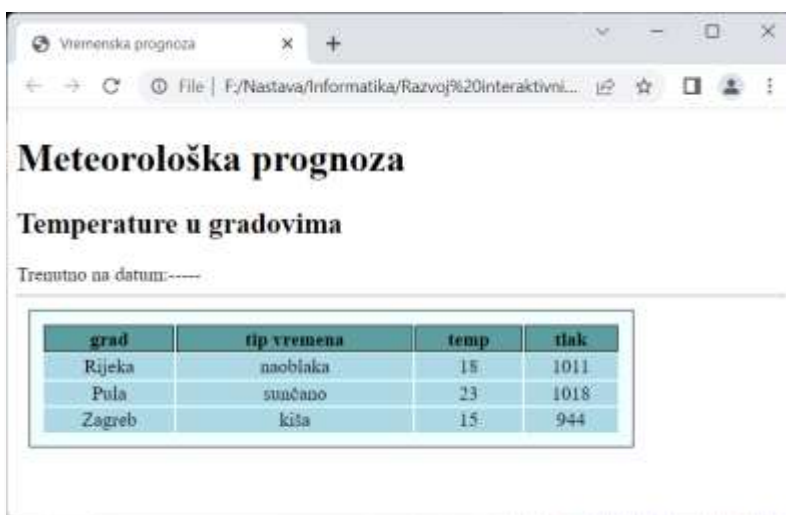
```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title> Vremenska prognoza </title>
    <link rel="stylesheet" type="text/css" href="stil.css">
  </head>
  <body>
    <h1 title = " Aktualne temperature " > Meteorološka p
    <h2>Temperature u gradovima</h2>
```

```

<div> Trenutno na datum:-----</div>
<hr style="color: darkblue;">
<table title="Gradovi" class="tabl_vrem_grad">
  <tr><th>grad</th><th>tip vremena</th>
  <th>temp</th><th>tlak</th></thead></tr>
  <tr><td>Rijeka</td><td>naoblaka</td><td>18</td>
    <td>1011</td></tr>
  <tr><td>Pula</td><td>sunčano</td><td>23</td><td>1018</td></tr>
  <tr><td>Zagreb</td><td>kiša</td><td>15</td><td>944</td></tr>
</table>
</body>
</html>

```

Izgled napravljenog dokumenta – prikaz u *web*-pregledniku dan je na slici 12.



Slika 12. Prikaz HTML dokumenta s jednostavnim CSS stilovima (Izvor: autorica)

CSS je standardni stilski jezik, no pojavljuju se i njegova proširenja poput SASS (engl. *Syntactically Awesome Style Sheets*), SCSS (engl. *Sassy Cascading Style Sheets*), Less (engl. *Leaner Style Sheets*). Ova proširenja dolaze s vlastitim skriptnim jezicima koji imaju mogućnost korištenja varijabli, funkcija i pravila. Na taj način pojednostavljuju i olakšavaju pisanje stilova i

strukturiranje stilskih dokumenata. Predprocesiranjem se prevode u standardni CSS jezik koji *web*-preglednik razumije.

4.1.3 DOM

DOM (engl. *Document Object Model*) je standardni objektni model za HTML dokumente koji HTML dokument opisuje kao skup objekata kojima se može dinamički pristupiti i mijenjati ih. Ovaj model promatra strukturu HTML dokumenta u hijerarhijskom obliku, pri čemu se hijerarhija gleda kao obrnuto stablo s korijenom na vrhu i čvorovima na rubovima. Elementi u DOM stablu su:

- roditeljski čvorovi (engl. *parents*)
- djeca (engl. *children*)
- braća/sestre (engl. *siblings*).

Hijerarhija u DOM strukturi definira da se korijenski čvor (root) nalazi na vrhu stabla, svaki čvor ima jednog roditelja, čvor može imati više djece, braća/sestre imaju zajedničkog roditelja.

Unutar HTML dokumenta definirani su sljedeći dijelovi DOM modela:

Sam dokument je jedan čvor - čvor dokumenta (document)

Svaki HTML element je jedan čvor - čvor elementa (node)

Svaki HTML atribut je isto zaseban čvor - čvor atributa (attribute)

Svaki sadržaj (tekst) u HTML elementu je zaseban čvor - tekstualni čvor (text).

Komentari su čvorovi komentara.

Na slici 13 je prikazan izgled DOM model za sljedeći HTML dokument:

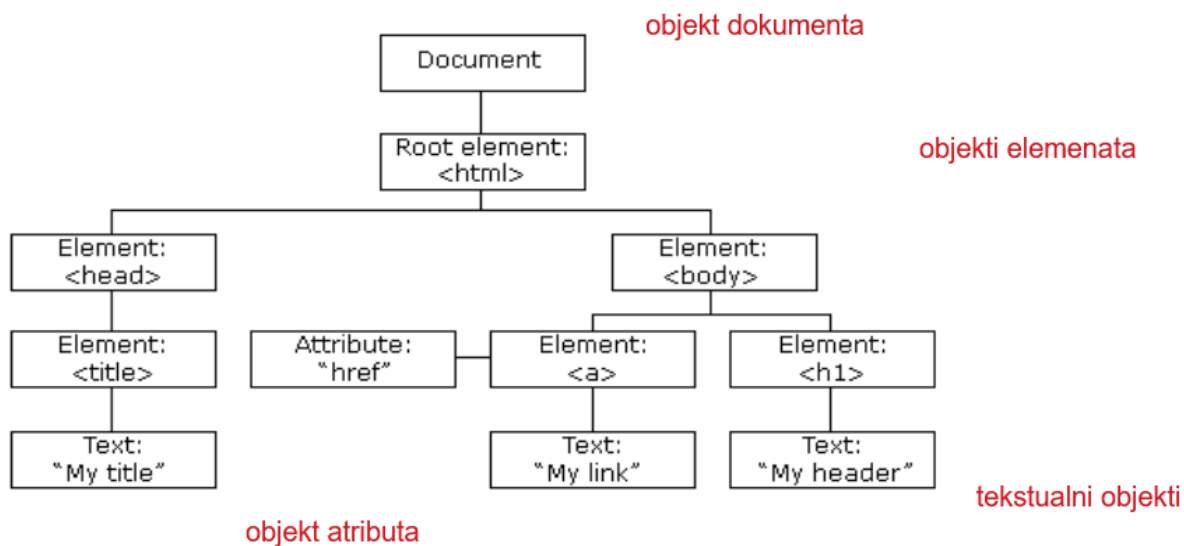
```
<html>
  <head>
    <title>My title</title>
```

```

</head>
<body>
  <h1>My header</h1>
  <a href = " " > M y   l i n k < / a >
</body>
</html>

```

Svaki čvor DOM-a je jedan objekt, pa se HTML elementi definiraju kao objekti (W3Schools *HTML DOM Documents*, n.d.).



Slika 13. Prikaz hijerarhije DOM objekata za HTML dokument

(Izvor: W3Schools *HTML DOM Documents*, n.d.)

DOM omogućuje da se pojedinim HTML elementima pristupa preko JavaScripta i da ih se može dinamički mijenjati. Hijerarhijski oblik znači da se, primjerice, brisanjem jednog elementa brišu i svi elementi ispod njega, odnosno mijenjanjem nekog svojstva izmjena utječe na sve elemente ispod.

JavaScript može koristeći DOM model definirati slušanje događaja na svakom elementu (objektu) i mijenjati: HTML elemente, HTML attribute i CSS stilove.

4.1.4 *JavaScript*

JavaScript je skriptni programski jezik koji se na klijentskoj strani interpretira unutar *web*-preglednika. Zaslužan je za responzivan dizajn, osvježavanje dijela *web*-stranice bez ponovnog učitavanja, provjeru korisničkih unosa, dinamičko kreiranje novih HTML elemenata te procese koji se odvijaju u pozadini, poput komunikacije s poslužiteljem, dohvata i slanja podataka. Odvija se nesmetano za korisnika, tako da korisnik ne zna što se u pozadini događa. Unutar *web*-preglednika korisnik može isključiti izvršavanje JavaScript programa.

Web-preglednik se danas sve češće koristi kao dobra zamjena za klasične desktop aplikacije. JavaScript je *web*-pregledniku dodao dinamične funkcionalnosti koje osiguravaju nesmetan i kontinuiran rad za korisnika. Iz tog razloga JavaScript je postao široko popularan. Da bi se pojednostavio i olakšao rad s JavaScriptom, najčešće korištene funkcionalnosti su uobličene u funkcije koje su ukomponirane u JavaScript biblioteke i/ili razvojna okruženja: JQuery, Bootstrap, Angular, Ionic, React, Vue.js, Quasar. Najčešće se radi o funkcionalnostima vezanima uz kontrolu unosa/ispisa podataka i interakciju s korisnikom, prilagodbu *web*-sučelja različitim uređajima i rezolucijama, jednostavnije korištenje grafičkih elemenata, dojam komunikacije sa serverskom stranom i bazom podataka u realnom vremenu bez čekanja. Često se navedene funkcionalnosti vežu i uz CSS te time poboljšavaju izgled aplikacije i korisnički doživljaj (engl. *user experience*) kod interakcije s korisnikom.

Jedan od nedostataka JavaScripta su biblioteke i razvojna okruženja koja jako olakšavaju razvoj klijentskih aplikacija, no obično su i ta rješenja glomazna - dolaze u velikim paketima s velikim brojem datoteka. Renderiranje s klijentske strane podrazumijeva da se aplikacije i biblioteke učitavaju na lokalno računalo i izvršavaju unutar *web*-preglednika. Današnja brza računala s puno memorijskog kapaciteta i procesorske snage lako mogu učitavati i izvršavati JavaScript biblioteke. No, treba pripaziti na to ako se izvođenje takvih JavaScript aplikacija radi na računalima s manje hardverske snage i prostora.

Ovdje je opisan mali dio funkcionalnosti *vanilla* JavaScripta koji se koristi u kombinaciji s DOM elementima da bi dinamizirao *web*-dokument.

Osnovne karakteristike JavaScripta:

- JavaScript je skriptni jezik koji podržava objektno-orientirano i funkcijsko programiranje.
- Sintaksa mu je slična Javi i C-u.
- Interpreter je, što znači da postoji JavaScript engine.
- Osjetljiv je na velika i mala slova ("case sensitive").
- Naredbe su odvojene sa ;
- Blok naredbi se nalazi unutar zagrada: { }
- Komentari su // jednolinijski ili /* višelinijski */
- Varijabla može sadržavati vrijednost bilo kojeg tipa podatka
- Ako varijabli nije dodijeljena neka vrijednost, ona se vodi kao nedefinirana (undefined)
- JavaScript kod se ubacuje u HTML, unutar oznaka: <script> </script>

Deklaracija i inicijalizacija varijabli

Varijable se deklariraju s *const*, *let* ili *var*. U pravilu se prvo preporučuje korištenje *const* gdje god se inicijalizacija radi na početku i vrijednost se neće mijenjati, zatim *let* koja ima blokovski doseg.

const – deklaracija varijable koja se koristi kod početne inicijalizacije, gdje želimo biti sigurni da će varijabla sadržavati samo vrijednosti koje na početku definira, a koje se više neće mijenjati. Ova deklaracija onemogućava promjenu vrijednosti varijable unutar bloka u kojem je deklarirana (tj. identifikator konstantno ima dodijeljenu istu vrijednost).

let – deklaracija varijable koja se može mijenjati, a kojoj je doseg do kraja bloka u kojem je definirana. Izvan tog bloka nemoguće joj je pristupiti. Može imati i globalni doseg.

Vrijednosti koje se pridružuju varijablama mogu biti različitog tipa, primjerice brojevi (engl. *numbers*), znakovni nizovi (engl. *string*), logičke vrijednosti (engl. *boolean*), objekti (engl. *objects*) te liste (engl. *array*). Varijable kod JavaScripta su dinamične, što znači da se njihov tip mijenja u ovisnosti o tome kakva im je vrijednost pridružena.

Ako je varijabla deklarirana, ali nema dodijeljenu vrijednost, onda je ona *undefined*.

Provjera tipa varijable:

```
typeof instance === "undefined"
```

Kod JavaScripta varijable mogu biti deklarirane globalno, lokalno ili blokovski (takav im je i doseg). Vrijednost globalne varijable može se dohvatiti s bilo kojeg mjesta u programu, vrijednost lokalne može se dohvatiti samo unutar funkcije u kojoj je deklarirana, a vrijednost blokovske varijable može se dohvatiti samo unutar pojedinačnoga programskog bloka (omeđen vitičastim zagradama) u kojem je deklarirana.

Funkcije

Dio programskog koda koji bi se trebao izvršiti više puta i po određenom pozivu može se staviti u zaseban blok, odnosno funkciju. Kod JavaScripta funkcija se definira koristeći ključnu riječ *function* nakon koje se navodi njezin naziv i zagrade za navođenje parametara funkcije odvojenih zarezom.

Funkcija bez parametara:

```
function ispisiTekst() {  
  let tekst = "Tekst";  
  document.write(tekst);  
}
```

Funkcija s parametrima:

```
function izracun(x,y) {  
  let z = Math.sqrt(x)+Math.sqrt(y);  
  document.write("Izračun (" + x + " , " + y + ") je " + z);  
}
```

Izvođenje funkcije završava se krajem bloka ili upotrebom izjave **return** te se nastavlja izvođenje glavnoga dijela programskog koda. Pritom funkcija obično daje povratnu vrijednost (engl. *return*

value) koja je rezultat njezina izvođenja. Izjavom `return` zaustavlja se izvođenje funkcije i vraća se njezina vrijednost.

Funkcija s povratnom vrijednosti:

```
function kvadrat(x) {  
    var y = x*x;  
    return y;  
}
```

Funkcije je potrebno je pozvati (engl. *invoke*, *call*) navođenjem njezina naziva i parametara. Ako funkcija ima povratnu vrijednost, rezultat izvršavanja funkcije može se spremiti u varijablu.

```
const z = kvadrat(8);
```

Algoritamske strukture

if... else if... else: Izvršava se onaj blok programskog koda koji zadovoljava uvjet. Ako je prvi uvjet (if...) zadovoljen (True), onda se izvršava prvi blok {} i nakon toga se izlazi iz provjere, ako nije zadovoljen (False), prelazi se na provjeru drugog uvjeta (... else if...). Ako postoji else, on se izvršava bez provjere ako raniji uvjeti nisu bili zadovoljeni.

```
<script>  
function max (x, y) {  
    if (number < 0) {  
        estimate = "Negative number";  
    } else if (number > 0) {  
        estimate = "Positive number";  
    } else {  
        estimate = "Number equal to 0";  
    }  
}  
</script>
```

for... : Ponavljanje ima tri dijela: početna vrijednost, uvjet za provjeru i operacija koja se izvodi kod svakog prolaza. Ponavljanje se izvršava od neke početne vrijednosti, dok se nakon svakog prolaza odradi operacija i provjerava uvjet. Ponavljanje se radi sve dok je uvjet True.

```
function ispisiSve(abeceda){
    for (var i=0; i<abeceda.length; i++){
        document.write(slovo abecede je "+abeceda[i]+"<br>");
    }
}
var ab = [' A', ' B', ' C', ' D', ' E'];
ispisiSve(ab);
```

while... : Ponavljanje u while bloku se radi sve dok je ispitni uvjet True. U ispitnom uvjetu se nalazi variabla koja se u programskom bloku mijenja.

```
function ispisiSve2(abeceda){
    var i=0;
    while (i<abeceda.length){
        document.write((i+1)+". slovo abecede ... je "+abeceda[i]+
            "<br />");
        i++;
    }
}
```

forEach... : Ovo ponavljanje vezano je uz niz podataka. Funkciji forEach() prosljeđuje se druga funkcija koja će raditi s pojedinim elementom niza.

```
var txt = "";
var numbers = [45, 4, 9, 16, 25];
numbers.forEach(myFunction);
function myFunction(value, index, array) {
    txt = txt + value + "<br>";
}
```

JavaScript je objektni programski jezik. Objekti se mogu kreirati na više načina, a jedan od načina kreiranja JavaScript objekata je prema JSON formatu zapisa: naziv: vrijednost

Primjer:

```
var osoba = {  
  ime: "Pero",  
  prezime: "Perić"  
};
```

Kreiranje niza objekata:

```
var osobe = [  
  {  
    ime: "Ana",  
    prezime: "Anić"  
  }, {  
    ime: "Marko",  
    prezime: "Markić"  
  }  
];
```

Pristup drugom objektu:

```
document.write(osobe[1].ime+" "+osobe[1].prezime);
```

odnosno pristup svim objektima niza korištenjem `forEach()` funkcije:

```
osobe.forEach(osoba => {  
  document.write(osoba.ime+ " "+osoba.prezime);  
});
```

Funkcija `forEach()` iz niza `osobe` uzima po jedan element i prosljeđuje ga kao parametar `osoba` funkciji koja zatim radi ispis u postojeći HTML dokument.

Unutar HTML dokumenta JavaScript pristupa DOM objektima. Za to se koriste specifične JavaScript funkcije.

Funkcija

```
getElementById(var id)
```

vraća jedan objekt koji ima navedeni id. Primjer:

```
<h1 id="prvi">Naslov</h1>
<div>Neki tekst</div>
<script>
  let elem = document.getElementById("prvi");
  elem.innerHTML="Drugi naslov";
  elem.style.borderStyle="outset";
  elem.style.borderColor="blue";
  elem.style.borderWidth="5px";
</script>
```

U navedenom primjeru varijabli elem pridružuje se objekt: <h1 id="prvi">Naslov</h1>. Nakon toga se tom objektu mijenja sadržaj (elem.innerHTML) u "Drugi naslov". Zatim mu se mijenja i CSS dizajn (elem.style): dodaje mu se plavi okvir širine 5 piksela.

Funkcija

```
getElementsByTagName(var tagname)
```

vraća niz objekata koji imaju određen naziv HTML oznake. Primjer:

```
<div>Prvi dio teksta</div>
<div>Drugi dio teksta</div>
<script>
  var elem = document.getElementsByTagName("div");
  for (let i=0; i< elem.length; i++) {
    elem[i].style.background="pink";
  }
</script>
```

```
}  
</script>
```

Funkcija

```
getElementsByClassName (var classname)
```

vraća niz objekata sa specificiranim class atributom. Primjer:

```
<div class="highlight">Naslov</div>  
<span class="plain">O b i č a n t e k s t</span>  
<div class="highlight">Podnaslov</div>  
<script>  
  var hlElements = document.getElementsByClassName("highlight");  
</script>
```

Atribut innerHTML mijenja sadržaj DOM objekta:

```
objekt.innerHTML= "Zamijenjeni tekst";
```

Promjena CSS stila DOM objekta:

```
objekt.style.color = "red";  
objekt.style.backgroundColor="lavender";
```

Javascriptom se na DOM elementima mogu dodati tzv. slušači događaja:

```
element.addEventListener (događaj, funkcija, pr
```

Događaji su inicirani od strane korisnika i mogu biti:

- pokrenuti mišem (engl. *mouse events*): onclick, ondblclick, onmousedown, onmouseup, onmouseout, onmousemove, onmouseover, onmouseenter, onmouseleave
- pokrenuti korištenjem tipkovnice (engl. *keyboard events*): onkeydown, onkeyup, onkeypress

- pokrenuti učitavanjem dokumenata (engl. *document events*): onload, onunload, onresize, onscroll, onabort, onbeforeunload, onhashchange, onpageshow, onpagehide
- događaji na elementima obrasca (engl. *form events*): onfocus, onfocusin, onfocusout, onchange, oninput, onselect, onsearch, onreset, onsubmit, onblur

Osim mijenjanja sadržaja i CSS stilova unutar elemenata, JavaScript preko DOM modela može na *web*-dokumentu kreirati nove elemente. Ovi su dinamički kreirani elementi vidljivi samo prilikom renderiranja dokumenta u *web*-pregledniku.

U primjeru je dan dio HTML i JavaScript programskog koda koji unosom podataka korisnika i klikom na gumb Potvrdi dinamički kreira nove elemente u HTML-u. Izgled aplikacije je dan na slici 14.

```

<label for="">Ime:</label> <input id="ime" type="text" /></div>
<label for="">Prezime:</label> <input id="prezime" type="text" />
<input type="submit" value="Potvrdi"
  onclick="spremiPodatke()" /></div>
Uneseni podaci su: <div id="podaci"></div>
<script>
function spremljPodatke(){
  let ime= document.getElementById("ime").value;
  let prezime = document.getElementById("prezime").value;
  let divElemPodaci = document.getElementById("podaci");
  let divElemIme = document.createElement("DIV");
  let tekstIme= document.createTextNode("Ime: "+ime);
  let divElemPrezime = document.createElement("DIV");
  let tekstPrezime= document.createTextNode("Prezime: "+prezime);
  divElemPodaci.appendChild(divElemIme);
  divElemIme.appendChild(tekstIme);
  divElemPodaci.appendChild(divElemPrezime);
  divElemPrezime.appendChild(tekstIme);
}
</script>

```

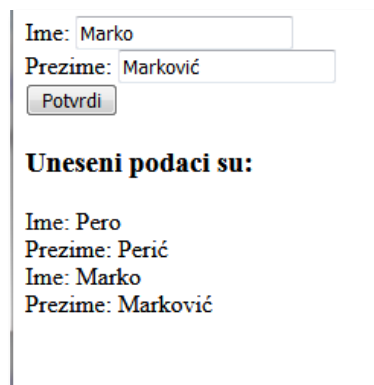
Unutar HTML dokumenta kreirana su polja za unos imena i prezimena (`<input type="text">`), gumb (`<input type="submit">`) i prazan `div` element s `id="podaci"`.

Klikom na gumb *Potvrdi* poziva se JavaScript funkcija *spremiPodatke()*. Funkcija poziva objekt `document.getElementById(„ime“)` – HTML element koji ima `id="ime"`, te iz njega funkcijom `.value()` izvlači uneseni tekst i sprema ga u varijablu `ime`. Isto je i za prezime.

Objekt `document.getElementById("podaci")` sprema se u varijablu `divElemPodaci`.

Nakon toga kreiraju se dva `<div>` elementa (čvora) i dva tekstualna elementa (čvora). Elementi `<div>` dodaju se (engl. *append*) u `divElemPodaci`, a u njih se dodaju tekstualni čvorovi s imenom i prezimenom.

Na ovaj način se elementi mogu neograničeno kreirati.



The image shows a web form with two input fields: 'Ime: Marko' and 'Prezime: Marković'. Below the fields is a 'Potvrdi' button. Underneath the button, there is a section titled 'Uneseni podaci su:' which displays the entered data: 'Ime: Pero', 'Prezime: Perić', 'Ime: Marko', and 'Prezime: Marković'.

Slika 14. Aplikacija koja dinami ki dodaje nove HTML elemente (Izvor: autorica)

Asinkrono izvršavanje

Sinkronizirano izvršavanje programskog koda je slijedno izvršavanje kod kojeg se naredbe izvršavaju jedna iza druge i druga ne počinje dok se prva ne izvrši. JavaScript se izvršava asinkrono, što znači da se istovremeno može izvršavati više akcija. To znači da se dio programskog koda može izvršavati bez čekanja, dok će drugi dio koda čekati neku informaciju.

Asinkrono izvršavanje najčešće se radi *callback* funkcijama, *promise* objektima i *async/await* pozivima.

Callback funkcija je funkcija koja se drugoj funkciji prosljeđuje kao argument, odnosno parametar. Ona se poziva unutar vanjske funkcije kako bi završila neku akciju. Callback funkcije izvršavaju se u pozadini, unutar logike originalne funkcije. Nakon što su asinkroni podaci vraćeni (riješeni), vanjska funkcija izvršava callback funkciju. Primjer je `setTimeout()` funkcija koja izvršava callback funkciju nakon 2000 milisekundi (ms). Umjesto `setTimeout` može se uzeti bilo koja akcija koja se ne mora odmah riješiti. U poglavlju Asinkrona komunikacija s poslužiteljskom stranom opisan je razlog i način korištenja asinkronih funkcija u *web*-aplikacijama.

```
function waitingForMusic(music) {
  setTimeout(() => {
    console.log(play(music))
  }, 2000);
}

function play(value) {
  return "play "+value;
}

waitingForMusic("Main theme");
console.log("setting music...")
console.log('end setting');
```

U navedenom primjeru prvo se poziva funkcija `waitingForMusic()`, no dok ona izvršava *callback* funkciju u kojoj se nalazi `console.log(play(music))`, ujedno se izvršavaju ostale naredbe: `console.log(„setting music“)` i `console.log(„end setting“)`.

Za rješavanje kompleksnijih situacija koriste se *promise* objekti. *Promise* objekt reprezentira rezultate izvršavanja asinkrone operacije. Objekt ima dvije *callback* funkcije: *resolve* i *reject*. On na temelju asinkrone akcije poziva jednu od njih. Ako je asinkrona akcija bila uspješna, poziva se *resolve callback* funkcija unutar `then()` metode, odnosno, u suprotnom, kad se pojavi greška, poziva se *reject callback* funkcija unutar `catch()` metode. (Morgan, 2018)

JavaScript *promise* objekt može biti u 3 stanja:

- na čekanju (engl. *pending*) - inicijalno stanje kad je asinkrona operacija u tijeku
- ispunjena (engl. *resolved*) kad je operacija uspješno završena

- odbačen (engl. *rejected*) kad operacija ima grešku ili nije uspješno izvršena.

Kad je *promise* objekt pozvan, onda je u stanju čekanja, a funkcija koja ga je pozvala nastavlja se izvršavati. Iz čekanja, kad su uvjeti zadovoljeni, ide u rješavanje i ispunjeno ili odbačeno stanje te poziva odgovarajuću *callback* funkciju za to.

Promise ima 3 glavne metode: then, catch i finally.

then se poziva kad je operacija koju izvodi *promise* objekt uspješno završila (engl. *resolve*);

catch se poziva u slučaju greške (engl. *reject*);

finally se poziva na kraju vezanih uvjeta, bez obzira na uspješnost ili neuspješnost izvođenja ranijih operacija.

Dakle, *resolve* se poziva preko *then* metode, a *reject* preko *catch* u slučaju greške. U slučaju poziva jedne metode (*resolve* ili *reject*) druga metoda se ne poziva.

U primjeru

```
let musicOn = true;
const musicSetup = new Promise((resolve, reject) => {
  if (musicOn)
    resolve({
      play: 'Playing... ',
    })
  else
    reject({
      play: 'Music is off',
    })
});

function getMusicSetup(musicTheme) {
  musicSetup
    .then(music => {
```

```

        return music.play+ musicTheme;
    })
    .then(music => {
        console.log(music+' ... on repeat');
    })
    .catch(e => {
        console.log('fail '+e.play)
    });
}
getMusicSetup("my favorite music")

```

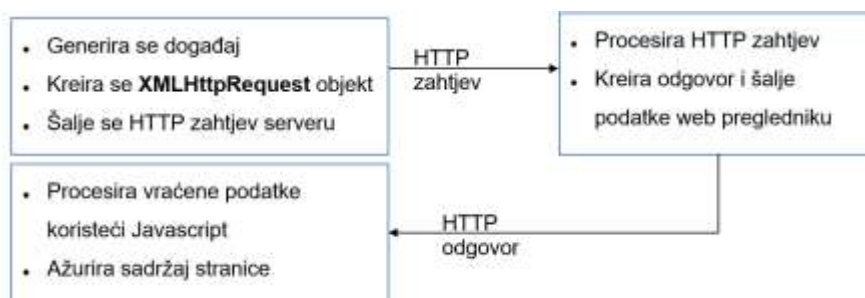
kreiran je *promise* objekt `musicSetup`. On izvršava `resolve()` metodu u slučaju da je varijabla `musicOn` postavljena na `true`. U suprotnom izvršava `reject()` metodu. *Promise* objekt se poziva iz `getMusicSetup()` funkcije kojoj se prosljeđuje varijabla `musicTheme`. U slučaju da se u *promise* objektu pozove `resolve()` metoda, izvršava se prva `then` operacija. S obzirom na to da prva operacija vraća `music.play+musicTheme`, poziva se i druga `then` operacija koja radi ispis na konzolu (`console.log`). Ako `return` u prvoj `then` operaciji zamijenimo s ispisom (bez `return`), druga se `then` operacija neće izvršiti. U slučaju da `musicSetup` objekt izvrši `reject()` metodu, funkcija `getMusicSetup()` izvršava `catch()` dio s ispisom na konzolu.

Async/await se koriste zajedno kod poziva funkcija koje imaju programski kod koji mora čekati da se jedan dio izvrši i vrati rezultat, kako bi drugi dio koda nastavio s izvođenjem. Funkcija zbog čekanja ne blokira izvršavanje cijelog programa. Riječ *async* stavlja se ispred imena funkcije koja vraća *promise* objekt koji može biti uspješno riješen (engl. *resolved*) ili odbačen (engl. *rejected*) ako se pojavi greška. Ispred funkcije koja čeka na odgovor stavlja se riječ *await*. Kad odgovor dođe, funkcija nastavlja izvođenje. Primjeri takvih funkcija nalaze se u poglavlju Povezivanje aplikacije s aplikacijskim programskih sučeljem.

4.1.5 AJAX

Kod klijentsko-poslužiteljske komunikacije klijent šalje zahtjev poslužitelju za nekom stranicom ili informacijom, a poslužitelj šalje odgovor. Kod sinkronizirane komunikacije klijent čeka odgovor i kad ga dobije obično je to nova stranica s traženom informacijom koja se učitava u *web*-preglednik. Kod bilo koje interakcije korisnika sa stranicom koja treba konstantno dohvaćati podatke s poslužitelja klijent bi trebao čekati novo učitavanje stranice. Da bi se to izbjeglo, koristi se AJAX.

AJAX (engl. *Asynchronous JavaScript and XML*) je skup tehnologija HTML, DOM, JavaScript koje osiguravaju asinkronu komunikaciju s poslužiteljskom stranom i prikaz novog sadržaja na *web*-stranici ili dijelu *web*-stranice, bez ponovnog učitavanja cijele stranice. Prilikom komunikacije s poslužiteljskom stranom razmjenjuju se male količine podataka, *web*-stranica se normalno izvršava, a kad dođe odgovor od strane poslužitelja osvježava se dio sadržaja dokumenta. U međuvremenu korisnik ima osjećaj normalnog rada i ne zna da se u pozadini odvija komunikacija s poslužiteljem. Od tehnologija koje sudjeluju u asinkronoj komunikaciji JavaScript je zadužen za osiguravanje komunikacije s poslužiteljskom stranom i dinamičko ažuriranje dijela stranice, a DOM osigurava pristup HTML elementima koji se mogu osvježiti novim informacijama s poslužitelja. JavaScript pristupa HTML elementima preko DOM modela.



Slika 15. AJAX komunikacija između klijenta i poslužitelja (Izvor: autorica)

Dio JavaScript programskog koda koji radi komunikaciju s poslužiteljskom stranom:

```
var httpRequest = new XMLHttpRequest();
    httpRequest.onreadystatechange = function() {
        if(httpRequest.readyState == 4) {
            var tekst = httpRequest.responseText;
            document.getElementById("tekst").innerHTML=tekst;
        }
    }
    httpRequest.open("GET", "provjeri.php?ime="+ime+"&prezime="+prezime,
        true);
    httpRequest.send();
```

XMLHttpRequest objekt je Javascript objekt koji otvara komunikacija prema serveru, te se šalju podaci skripti na serverskoj strani (slika 15). *onreadystatechange* je rukovatelj događaja koji se aktivira kod bilo koje promjene stanja, a *readyState* je varijabla koja definira trenutno stanje XMLHttpRequest objekta.

Stanja mogu biti:

0 - zahtjev nije inicijaliziran (javlja se nakon kreiranja XMLHttpRequest objekta, ali prije poziva `open()` metode)

1 - zahtjev je postavljen (nakon poziv `open()`)

2 - zahtjev je poslan (nakon poziva `send()`)

3 - zahtjev se procesira (nakon uspostavljanja veze sa serverom)

4 - zahtjev je kompletiran (nakon što je vraćen odgovor).

Obično se čeka promjena stanja objekta u vrijednost 4. To znači da je zahtjev kompletiran, došao je odgovor sa servera i može se izvršiti akcija na nekom DOM elementu (W3Schools *AJAX Introduction*, n.d.).

4.2 Karakteristike softverskih razvojnih okvira za realizaciju klijentskog dijela *web*-aplikacija.

Za brži i jednostavniji razvoj *web*-aplikacija danas se najčešće koriste softverski razvojni okviri (engl. *software frameworks*), dizajnirani kako bi pomogli razvojnim programerima u bržem kreiranju aplikacija. Razvojni okvir je aplikacija koja služi za lakši razvoj drugih aplikacija specifične namjene. Ima predefinirane skupove komponenti (modula) koji su napravljeni generički, na višoj razini apstrakcije, temeljem najbolje prakse za određenu vrstu aplikacija (*web*, desktop, mobilne, poslovne, igre). Navedene komponente su zajedničke svim aplikacijama istog tipa, a onda ih programeri mogu koristiti i prilagoditi vlastitoj aplikaciji.

Web-aplikacije razvijaju se koristeći razvojne okvire za vidljivo korisničko sučelje koje pripada klijentskoj strani (engl. *frontend frameworks*) te razvojne okvire za pozadinski dio aplikacije koji nije vidljiv krajnjem korisniku, a izvršavaju se na poslužitelju (engl. *backend frameworks*).

Klijentski razvojni okviri nude slaganje gotovih interaktivnih grafičkih komponenti koje su sastavljene od HTML-a, CSS-a i JavaScripta. HTML strukturira dokument koji se prikazuje krajnjem korisniku. CSS se koristi za dizajn dokumenta: definiranje rasporeda i izgleda komponenti. JavaScript osigurava dinamizaciju *web*-stranice i koristi se za interakciju korisnika s aplikacijom te responzivan dizajn. Sve navedene tehnologije imaju za cilj da korisniku osiguraju brzo interaktivno korisničko sučelje, koje je prilagođeno korisnicima (engl. *user-friendly*). Trenutačno popularni klijentski razvojni okviri bazirani na JavaScriptu su: Quasar, Vue.js, Angular, React, Ionic.

Poslužiteljski razvojni okviri osiguravaju izgradnju dijela aplikacije koji će odrađivati pozadinski dio aplikacije, a to se odnosi na dohvat podataka iz baze ili drugih aplikacija i slanje podataka klijentskoj strani na zahtjev, autentikaciju i provjeru dozvola pristupa korisnicima, validaciju podataka i spremanje podataka u bazu, izračune, renderiranje dokumenata i slično. Razvojni okviri nude pripremljene generičke blokove programskog koda za najčešće tražena rješenja (primjer autentikacije), te ih programeri zatim koriste i prilagođavaju za vlastite potrebe. Primjeri razvojnih okvira s poslužiteljske strane su Laravel, Yii, CodeIgniter pisani u PHP programskom jeziku, Django pisan u Pythonu, te Node.js s Expressom pisan u JavaScriptu.

4.2.1 *Razvojni okviri Quasar i Vue.js*

Quasar i Vue.js su razvojni okviri za razvoj klijentskog dijela (engl. *frontend*) *web*-aplikacija. Quasar se naslanja na razvojni okvir Vue.js i nudi brži i jednostavniji razvoj *web*-aplikacija. Vue.js je JavaScript razvojni okvir izgrađen na HTML-u, CSS-u i JavaScriptu i omogućuje deklarativno i komponentno programiranje. Komponentno programiranje vezano je uz kreiranje komponenti uz definiranje sadržaja, izgleda i logike svake komponente. Komponente koriste SFC (*Single file component*) format koji se zapisuje u „*.vue“ datoteku. SFC u datoteku smješta strukturu (HTML), izgled (CSS) i logiku (JavaScript) za pojedinu komponentu. Deklarativno renderiranje radi prema HTML predlošku u kojem se nalaze podaci koji se mogu dinamički mijenjati. JavaScript čuva i prati stanja objekata. U trenutku promijene tih stanja, stranica se automatski renderira i prikazuje nove izmjene. Kod deklarativnog renderiranja ne treba eksplicitno pisati programski kod kojim se ažurira DOM, nego se reaktivnost dobiva praćenjem JavaScript stanja i efikasnim ažuriranjem DOM elemenata u trenutku kad se izmjena dogodi.

4.2.2 *Responzivni dizajn*

Responzivni *web*-dizajn (engl. *responsive web-design*) je tip dizajna koji se automatski prilagođava uređaju na kojem se prikazuje. Ovisno o uređaju (desktop računala, tableti, pametni telefoni), dizajn *web*-aplikacije se automatski podešava po visini i širini uređaja, neki sadržaji se proširuju, neki povećavaju radi bolje preglednosti, a neki skrivaju. U kreiranju responzivnog dizajna sudjeluju HTML, CSS i ponekad JavaScript. Postoje gotovi razvojni okviri i biblioteke koje se koriste u postizanju responzivnog dizajna, a to su W3.CSS, Bootstrap, JQuery.

Za pravilnu distribuciju elemenata i responzivni dizajn u Vue.js razvojnom okviru postoji ugrađen CSS razmještaj Flexbox koji dijeli prozor na 12 jednakih stupaca. Komponente se raspoređuju u stupce, a jedna komponenta može se proširiti na više stupaca (Quasar *Introduction to Flexbox*, n.d.). Na taj način komponentama se osigurava da se dinamički prošire i povećaju na većem ekranu, odnosno smanje na manjem ekranu. Komponentama se u tom slučaju ne mora definirati fiksna veličina koja bi mogla zauzimati previše mjesta na manjem ekranu, odnosno premalo mjesta na većem ekranu. Komponente su fleksibilne, čime se osigurava responzivni dizajn.

4.2.3 *Single page application*

SPA (engl. *Single Page Application*) je *web*-aplikacija koja se s klijentske strane učitava kao jedna *web*-stranica s mehanizmom dinamičkog učitavanja sadržaja i prikaza na vidljivom dijelu stranice. Time se prema krajnjem korisniku osigurava kontinuitet u pregledu sadržaja, bez da se novi sadržaj dohvaća ponovnim učitavanjem cijele stranice u *web*-preglednik. Ponekad se za veliku količinu teksta koji se treba prikazati ugrađuje mehanizam beskonačnog pomicanja teksta kako bi se osiguralo da se ne dohvate svi podaci odjednom nego da se dohvati i prikaže dio potrebnih podataka.

4.3 **Struktura dokumenta u softverskom razvojnom okviru**

U Quasar/Vue.js razvojnim okvirima koriste se datoteke s „*.vue“ ekstenzijom. Te datoteke koriste Single File Components (SFC) format koji u istom dokumentu sadrži predložak za definiranje strukture stranice i razmještaj komponenti (HTML), dodatni dizajn tih komponenti korištenjem stilova (npr. CSS/SASS/SCSS/Less) i logiku za dinamizaciju komponenti na stranici (JavaScript).

Datoteke se sastoje od tri dijela (Quasar *How to use vue*, n.d.):

- `template`
- `script`
- `style`.

Template je dio u `.vue` datoteci omeđen s `<template>` `</template>` oznakama i postavlja strukturu grafičkog sučelja koja će se prikazati unutar *web*-preglednika. U tom dijelu se nalaze posložene grafičke komponente koje će se koristiti za interakciju s korisnikom: labele, liste, gumbi, polja za unos teksta, slike, tablice i slično. Grafičke komponente mogu se grupirati unutar kontejnera ili spremnika (engl. *containers*). Spremnici su grafičke komponente koje se mogu smjestiti unutar nekog drugog spremnika. Početnim spremnikom smatra se *web*-preglednik. Svaki spremnik ima definiran raspored elemenata (engl. *layout*). On omogućava dinamički razmještaj grafičkih komponenti unutar spremnika.

Script je dio u datoteci omeđen sa `<script>` `</script>`, a pripada logičkom dijelu aplikacije pisanom u JavaScript programskom kodu.

Style je dio u `.vue` datoteci omeđen sa `<style>` `</style>`, a pripada definiranju CSS stila za navedenu komponentu.

Primjer:

```
<template>
  <!-- you define your Vue template here -->
</template>

<script>
// This is where your Javascript goes to define your Vue component,
// which can be a Layout, a Page or your own component

export default {
  //
}
</script>

<style>
/* This is where your CSS goes */
</style>
```

4.4 Životni ciklus aplikacija u softverskom razvojnom okviru.

S obzirom na to da je Quasar nadograđen na razvojni okvir Vue.js, ima sva svojstva Vue životnog ciklusa i dizajna.

Dizajn aplikacije slijedi MVVM (engl. *Model-View-ViewModel*) uzorak, gdje Vue aplikacija pripada u ViewModel koji povezuje View (prikaz) i Model (podatke), te ih sinkronizira. Svaki put kad se dogodi promjena u podacima Vue instanca ažurira prikaz. Vue aplikacija sastoji se od jednog korijenskog (engl. *root*) objekta, te ostalih ugniježđenih komponenti koje su složene u

stablastu strukturu, počevši od glavne *root* komponente, a koje se mogu više puta koristiti (engl. *reuse*).

Sve komponente zajedno s korijenskom komponentom su objekti (instance). Svaka Vue aplikacija prolazi kroz niz koraka:

- kreiranje instance aplikacije
- inicijalizacija životnog ciklusa za instancu
- postavljanje podataka (kroz *data*) i inicijalizacija praćenja reaktivnosti
- kompajliranje predloška (engl. *template*)
- kreiranje i postavljanje elemenata u DOM
- praćenje promjene podataka i ažuriranje DOM-a kad se podaci promjene
- uništavanje komponenti i slušača na kraju životnog ciklusa.

Verzije 2 i 3 razvojnog okvira Vue (migracija s verzije 2 u 3 je aktualna u trenutku pisanja ove skripte) dosta se razlikuju u načinu smještanja komponenti. Vue 2 se naslanja na Options API, a Vue 3 na Composition API. Composition API grupira dijelove programskog koda unutar `setup()` metode. Logika aplikacije je grupirana u funkcije unutar kojih se smješta programski kod koji radi s više komponenti. Pri tom se komponente rade tako da su malene i jednostavne.

Option API specificira logiku za pojedinu komponentu, pa su komponente kompleksnije. Za to se koristi `options` objekt u Vue konstruktoru. Aplikacija započinje životni ciklus kreiranjem nove korijenske instance kojoj se prosljeđuje *options* objekt:

```
var vm = new Vue({
  // options
})
```

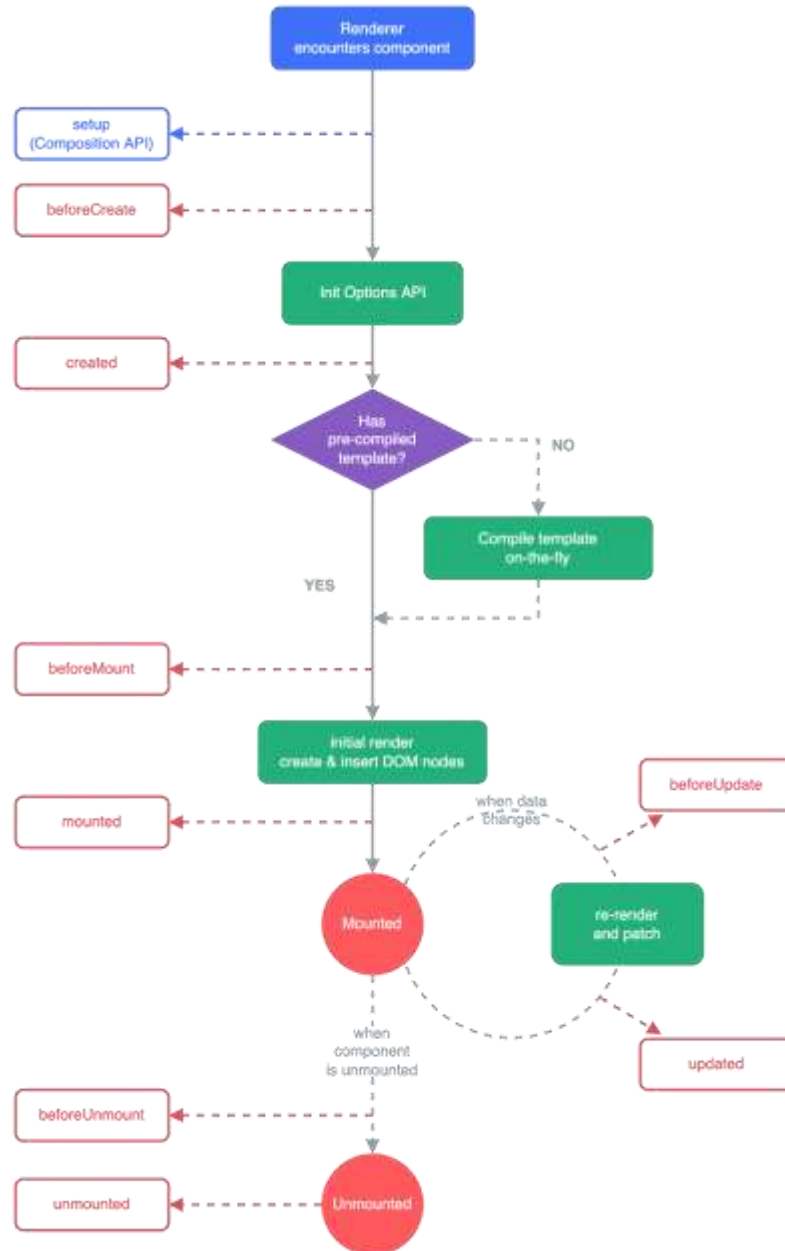
Pod *options* se mogu postavljati u Vue instancu svojstva opisana u *data* objektu, koji je povezan s reaktivnim Vue sustavom - kad god se neko svojstvo promijeni, pogled se automatski ažurira. Sva svojstva za koja želimo da se prate moraju se prijaviti prije nego je instanca kreirana. Pod *options* se također mogu upisati:

- *props* - koji sadrži listu atributa s vrijednostima
- *computed* - funkcije koje se odnose na jednostavne operacije
- *methods* - funkcije koje su vezane uz obradu događaja (*event handlers*)
- *watch* - funkcije koje zahtijevaju asinkrone operacije ili one koje zauzimaju više resursa za rad s podacima
- *emits* - događaji koje emitira komponenta.

Options također sadrži mogućnost povezivanja funkcija sa životnim ciklusom objekta (engl. *lifecycle hooks*). Te funkcije se izvršavaju u određenoj fazi životnog ciklusa, po redoslijedu (Vue.js *Options: Lifecycle*, n.d.):

- `beforeCreate`
- `created`
- `beforeMount`
- `mounted`
- `beforeUpdate`
- `updated`
- `beforeUnmount`
- `unmounted`.

Funkcija `beforeCreate` izvršava se kad komponente još nisu inicijalizirane. Funkcije definirane u `created` izvršit će se nakon što su komponente kreirane, ali nisu renderirane. Funkcije u `beforeMount` izvršit će se netom prije prvog renderiranja komponenti, a u `mounted` će se izvršiti



Slika 16. Životni ciklus Quasar/Vue.js aplikacija
(Izvor: *Vue.js. Lifecycle Hooks* (20. 12. 2022.))

nakon postavljanja komponenti koje mogu pristupiti ostalim DOM elementima. Funkcije u `beforeUpdate` izvršavaju se netom poslije promjene u komponenti, ali prije renderiranja, a u

updated se izvršavaju nakon svake promijene podataka i renderiraju komponente. Funkcije u beforeUnmount izvršavaju se netom prije uništavanja komponenti, a u unmounted nakon što su elementi uništeni, pri čemu se miču svi slušači događaja vezani uz njih i sve vezane *child* instance.

5 Interaktivna *web*-aplikacija

5.1 Raspored elemenata u softverskom razvojnom okviru

Unutar Quasar projekta u `src` folderu nalaze se početni dokumenti koji se učitavaju s prvim pokretanjem *web*-aplikacije:

```
src/layouts/MainLayout.vue
```

```
src/pages/IndexPage.vue
```

```
src/router/routes.js
```

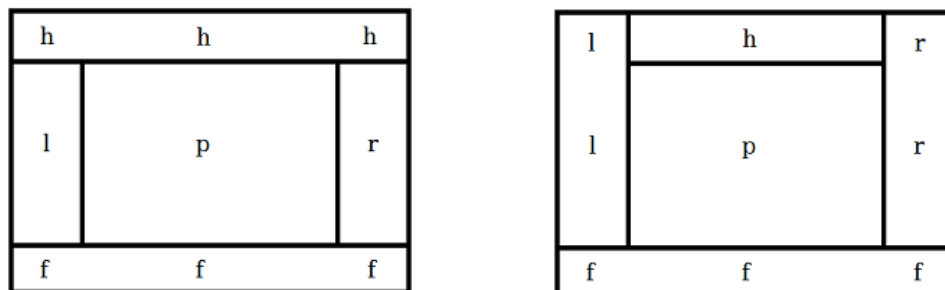
5.1.1 *Layout*

Nakon kreiranja Quasar projekta unutar `src/layouts` nalazi se dokument `MainLayout.vue`. Unutar tog dokumenta `QLayout` komponenta definira razmještaj zaglavlja, podnožja, navigacijskih traka te lijeve i desne trake.

`QLayout` dijeli ekran na matricu veličine 3x3. Prvi redak matrice pripada zaglavlju (h/H), drugi središnjem dijelu ekrana (p), a treći pripada podnožju (f/F). Također su definirani lijevi (l/L), središnji (p) i desni stupac (r/R).

Cijeli izgled matrice definira se kroz atribut `view`.

```
<q-layout view='hhh lpr fff'>
```



Slika 17. Dva moguća rasporeda elemenata u Quasar razvojnom okviru.

Lijevo: 'hhh lpr fff' Desno: 'lhr lpr fff' (Izvor: autorica)

Na slici 17 u lijevom primjeru, zaglavlje (h) se proteže kroz gornje tri ćelije (hhh). Središnji dio ekrana je definiran kroz lijevi dio (l), središnji (p) i desni dio (r). Podnožje (f) se proteže kroz donje tri ćelije (fff). U desnom primjeru slike 17 gornji dio ekrana se sastoji od lijevog dijela, zaglavlja i desnog dijela (lhr). Ostalo je isto kao i prethodnom primjeru.

Velikim slovima (H, F, L, R) označava se fiksni dio ekrana. Ovo znači da se, ako se pojavi više sadržaja na samoj stranici, ono što je označeno kao fiksni dio ekrana neće pomicati.

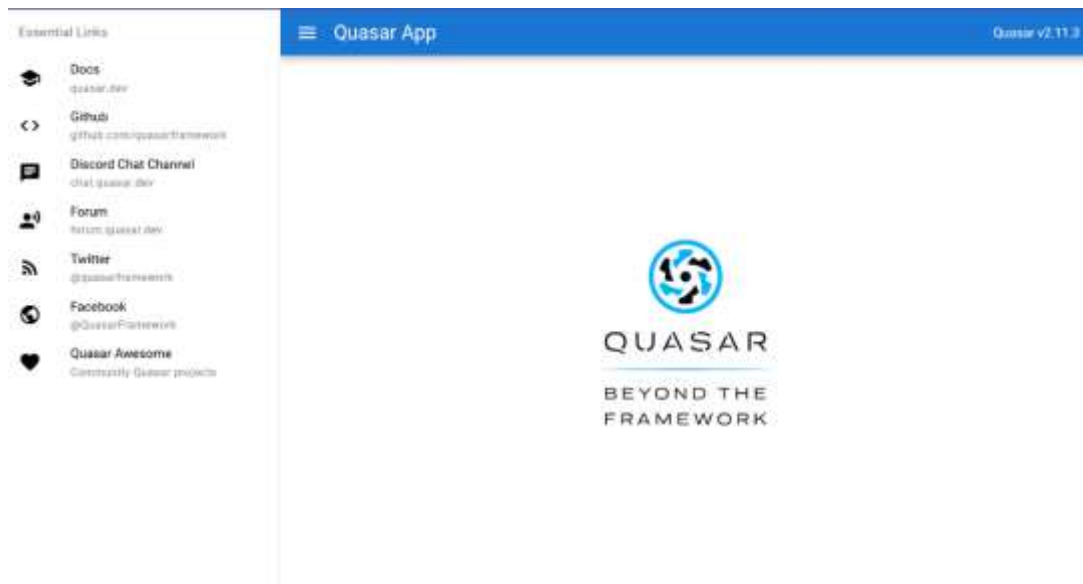
Početni prozor u Quasar razvojnom okviru je postavljen s rasporedom:

```
<q-layout view="lHh Lpr lFf">
```

Izgled je prikazan na Slici 18.

Dokumenti koji se nalaze u *src/layouts* definiraju osnovni raspored kontejnera na *web*-stranici.

Središnji dio ekrana unutar *layout* dokumenta definiran je komponentom `QPageContainer`. `QPageContainer` je kontejner za komponentu `QPage`. Unutar `<router-view />` oznake, ovisno o odabiru kroz navigaciju, prikazat će se neki od *page* dokumenata koji su smješteni unutar mape *pages*.



Slika 18. Raspored na po etnom prozoru u Quasar razvojnom okviru (Izvor: autorica)

```
<template>
  <q-layout>
    ...
    <q-page-container>
      <router-view></router-view>
    </q-page-container>
    ...
  </q-layout>
</template>
```

Može se definirati jedan *layout* dokument (npr. *MainLayout.vue*) u koji će se ubacivati različiti *page* dokumenti. Na taj način se središnji dio ekrana može iskoristiti za dinamički prikaz komponenti i sadržaja koji se nalaze u *src/pages* dokumentima.

Ako želimo napraviti novi *layout* dokument koji će sadržavati raspored kontejnera, možemo ga kreirati iz naredbenog retka:

```
quasar new layout UsersLayout
```


5.1.2 Page

Dokumenti koji se nalaze u `src/pages` definiraju središnji dio stranice. Inicijalno se kreira **IndexPage.vue** koji se nalazi u `src/pages` folderu.

Unutar Page dokumenta smješta se `QPage` komponenta koja definira sadržaj stranice, te se unutar nje mogu smjestiti i ostale komponente (`QInput`, `QTable`, `QButton...`).

```
<template>
  <q-page>
    ...page content...
  </q-page>
</template>
```

Ako želimo napraviti novi *page* dokument koji će sadržavati središnji dio stranice, možemo ga kreirati iz naredbenog retka:

```
quasar new page Users
```

Ono što povezuje *layout* i različite *page* stranice su putanje, odnosno rute između njih. One su definirane u datoteci `router/routes.js`.

Da bi se prikazala pojedina stranica unutar *router-view*aa, potrebno je rute prema stranicama definirati u datoteci `router/routes.js`

5.1.3 Router

Datoteka `routes.js` sastoji se od mapiranja URL putanja prema `.vue` datotekama. Kad se u *web*-pregledniku želi učitati određena datoteka, upisuje se njen URL, koji mora biti povezan s lokacijom te datoteke (*Quasar Routing with Layouts and Pages*, n.d.).

Datoteka `routes.js` sastoji se od:

- učitavanja `.vue` datoteka

```
import Layout from 'layouts/Layout'
```

```
import Page from 'pages/User'
```

- definiranja naziva osnovnog dijela URL-a

```
path: '/'
```

- naziva .vue datoteke koja definira raspored elemenata

```
component: 'layouts/MainLayout.vue'
```

- definiranja naziva podstranica koji se naslanjaju na osnovni dio URL-a.

```
children:  
  path: '/users'  
  component: 'pages/UsersPage.vue'  
  
const routes = [  
  {  
    path: '/',  
    component: () => import('layouts/MainLayout.vue'),  
    children: [  
      { path: '', component: () => import('pages/IndexPage.vue') },  
      { path: '/users', component: () => import('pages/UsersPage.vue') }  
    ]  
  },  
]
```

U osnovnom dijelu URL-a nalazi se:

```
http://naziv_servera:8080/
```

Uz ovaj URL povezana je putanja /, te datoteke *layout/MainLayout.vue* i *pages/IndexPage.vue*.

Podstranica osnovne stranice je:

```
http://naziv_servera:8080/#/users
```

Ovaj URL učitava datoteke *layout/MainLayout.vue* i *pages/UsersPage.vue*.

Na lokalnom server poziv stranice *UsersPage.vue* može se napraviti preko URL-a:

```
http://localhost:8080/#/users
```

5.2 Komponente i njihov razmještaj

Dobar korisnički dizajn podrazumijeva da će se korisnik lako snaći na *web*-sjedištu, a to se između ostalog postiže ujednačenim izgledom različitih dokumenata te elementima koji ostaju prisutni za vrijeme rada s aplikacijom. Obično se radi o zaglavlju, podnožju te navigacijskim elementima koji olakšavaju snalaženje krajnjem korisniku.

Unutar razvojnog okvira Quasar razmještaj definira se u *.vue* dokumentu koji se nalazi unutar quasar projekta u folderu *src/layouts*.

Struktura ekrana odnosno razmještaj komponenti na ekranu definiran je komponentom *QLayout*.

U zaglavlju se nalazi komponenta *QHeader*, u podnožju *QFooter*, navigacijske trake su definirane komponentom *QRouteTab*, a lijeve i desne trake s *QDrawer*. Središnji dio ekrana definiran je komponentom *QPageContainer* unutar koje se nalazi *QPage*. *QPage* definira koji sadržaj će se prikazivati na središnjem dijelu ekrana. Ako se definira dinamički sadržaj, onda se u *QPageContainer* komponentu ubacuju rute do pojedinih datoteka, što je prikazano u prethodnom poglavlju.

Komponenta *QHeader* označava zaglavlje dokumenta koje se pojavljuje na vrhu stranice, a *QFooter* označava podnožje dokumenta koje se pojavljuje na dnu stranice.

Uz ove komponente obično se pojavljuju i komponente *QToolbar* i *QtoolbarTitle*.

QToolbar obično grupira više komponenti zajedno i raspoređuje ih horizontalno. Obično se postavlja logotip poduzeća/projekta, glavni izbornik, komponente za pretraživanje, poveznice na dodatne informacije i slično.

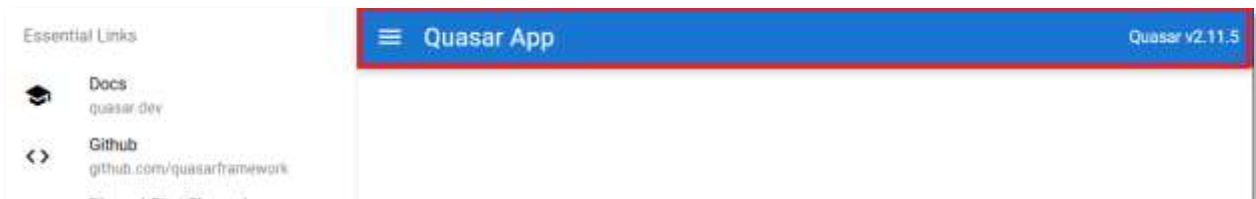
Unutar *QToolbar* komponente može se koristiti *QToolbarTitle* za dodati naslov *toolbara*.

Primjer QToolbar komponente koja se nalazi u layout/MainLayout.vue datoteci, odmah nakon kreiranja Quasar projekta:

```
<q-header elevated>
  <q-toolbar>
    <q-btn
      flat
      dense
      round
      icon="menu"
      aria-label="Menu"
      @click="toggleLeftDrawer"
    />

    <q-toolbar-title>
      Quasar App
    </q-toolbar-title>

    <div>Quasar v{{ $q.version }}</div>
  </q-toolbar>
</q-header>
```



Slika 19. Izgled komponente QToolbar (Izvor: autor)

QDrawer je komponenta koja se smješta uz lijevu i/ili desnu stranu ekrana i koristi se uz QLayout. Postoji nekoliko atributa koji se koriste uz ovu komponentu:

`show-if-above="true"` - prikazuje traku prilikom inicijalnog iscrtavanja (engl. *rendering*) ekrana

`bordered` - prikazuje okvir

`v-model="leftDrawerOpen"` - smješta traku na lijevu stranu ekrana.

Obično se unutar ove komponente smještaju `QList` i `QItem` komponente.

Primjer `QDrawer` komponente koja se nalazi u `layout/MainLayout.vue` datoteci, odmah nakon kreiranja Quasar projekta:

```
<q-drawer
  v-model="leftDrawerOpen"
  show-if-above
  bordered
>
  <q-list>
    <q-item-label
      headerr
    >
      Essential Links
    </q-item-label>
    <q-item>
      <q-item-section>
        <q-item-label></q-item-label>
      </q-item-section>
    </q-item>
    <EssentialAction
      v-for="link in essentialLinks"
      :key="link.title"
      v-bind="link"
    />
  </q-list>
</q-drawer>
```

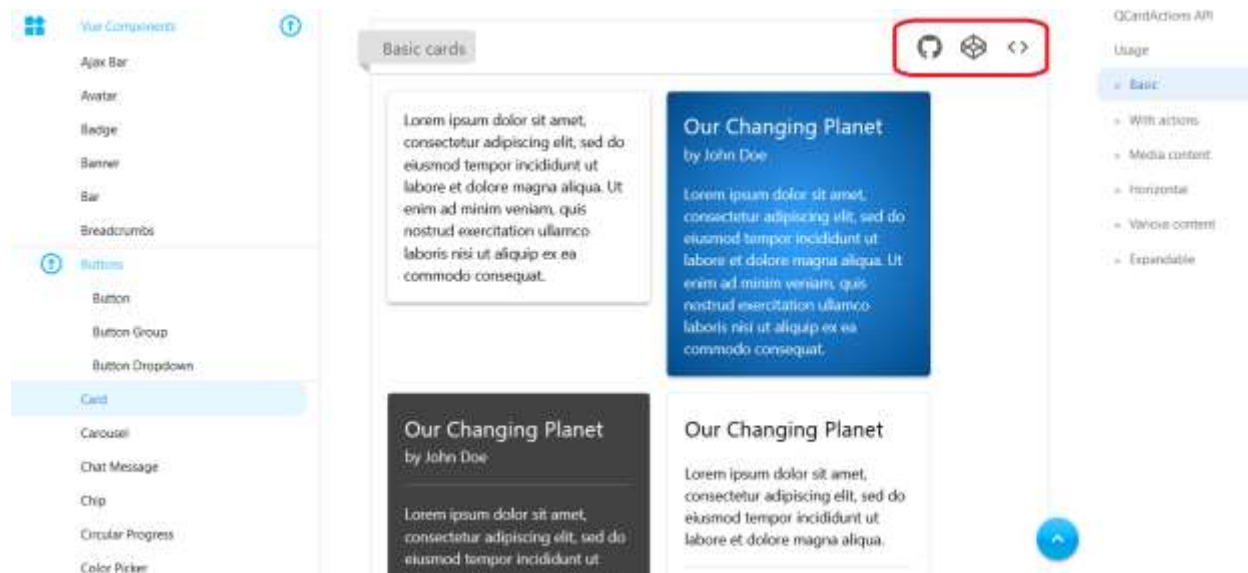
U primjeru je pozvana i komponenta EssentialLink koja je definirana kao datoteka components/EssentialLink.vue. Na slici 19, uz označenu komponentu QToolbar, prikazana je i komponenta QDrawer koja se nalazi s lijeve strane. Unutar nje iscrtane su ikone, nazivi i poveznice iz EssentialLink komponente.

5.3 Oblikovanje *web*-dokumenta. Grafički elementi

Web-stranica se sastoji od različitih grafičkih komponenti i kontejnera. Kontejneri su komponente koje unutar svojih okvira mogu smještati druge komponente.

U Quasar razvojnom okviru postoji velik broj gotovih grafičkih komponenti koje se mogu naći na službenim stranicama <https://quasar.dev/vue-components/>

Za svaku komponentu postoji opis, način korištenja. Brojni su primjeri kako neku komponentu uklopiti u osnovnu stranicu i kako je kombinirati sa skriptnim i stilskim dijelom.



Slika 20. Dio službene Quasar dokumentacije s primjerima komponenata

(Izvor: (Quasar Card n.d.)

Kad se nađe željena komponenta, uz svaki primjer dane su u gornjem desnom kutu tri opcije:

- poveznica na programski kod na GitHubu
- poveznica na Codepen
- otvaranje izvornog koda.

Na GitHubu su dostupni svi primjeri i mogu se preuzeti za isprobavanje. Poveznica vodi na programski kod aktualne komponente.

U Codepen *online editoru* može se postojeći programski kod komponente uređivati i rezultati se vide odmah na komponenti.

Izvorni kod učitava četiri dijela: Template, Script, Style i All.

U *template* dijelu nalazi se kod koji se može kopirati u *template* dio na glavnoj *page* stranici.

U *script* dijelu nalazi se dio JavaScript koda koji se kopira u `<script></script>` dio iste stranice.

Ako postoji *style* dio, vjerojatno je na komponenti definiran dodatni CSS stil bez kojeg komponenta neće izgledati isto kao na primjeru. *Style* dio kopira se u `<style></style>` na istoj stranici.

All daje prikaz svih dijelova u dokumentu.

Quasar i Vue podržavaju kreiranje novih globalnih komponenti koje se mogu koristiti u aplikaciji: *mixin*, *plugin* i *components*.


```
<p v-html = " x L o g " > < / p >
```

U gore navedenim primjerima dinamički se ubacuje vrijednost varijable *xLog*. U prvom primjeru kao običan tekstualni podatak za prikaz, a u drugom kao direktiva koja može biti renderirana.

5.4.1 *Mq t k – v g p l g " f k t g m v k x c*

Direktive su specijalni Vue atributi koji se ubacuju unutar HTML elemenata i reagiraju reaktivno na renderiranje DOM-a kad se promijeni vrijednost podataka u varijabli s kojom su povezani.

Vue dopušta umetanje HTML koda s JavaScript izrazima i Vue direktivama. Logika iz predloška ubacuje se u HTML bez potrebe da se apstrahira negdje drugdje u projektu. Vue direktive se referiraju kao „syntax sugar“ jer ne mijenjaju kako kod radi, nego način korištenja koda (Camden *et al.*, 2020).

Sve Vue direktive imaju prefiks *v-*.

v-text – služi za umetanje teksta unutar HTML elementa. Pred definira se statično mjesto za umetanje teksta i dodaje se *v-text* atribut. DOM automatski zamjenjuje tekst novim, kad novi tekst dođe. Ima isto značenje kao i interpolacija (umetanje) kroz dvostruke vitičaste zagrade.

Primjer:

```
<h2>{{ head }}</h2>  
<h3 v-text="head"></h3>
```

U `<script>` dijelu

```
export default {  
  data () {  
    return {  
      ...  
      head: 'Naslov',  
      ...  
    }  
  }  
}
```

```
}  
}
```



Slika 22. Izgled teksta u komponenti definiranog kroz v-text direktivu (Izvor: autorica)

v-once – koristi se s u kombinaciji s drugim atributima, a onemogućava reaktivnost elementa nakon što se jednom renderira tekst.

v-html – za razliku od ostalih v-direktiva koje unutar HTML elementa renderiraju isključivo tekst, ova direktiva može renderirati HTML. Odnosno, omogućuje ubacivanje drugih HTML elemenata u DOM.

Primjer:

```
<p v-text="htmlText"></p>  
<p v-html="htmlText"></p>
```

U `<script>` dijelu

```
export default {  
  data () {  
    return {  
    ...  
    htmlText: '<p><b>Formatirani</b> tekst</p>',  
    ...  
    }  
  }  
}
```

```
<p><b>Formatirani</b> tekst</p>
```

Formatirani tekst

Slika 23. HTML kod uba en kroz v-html direktivu (Izvor: autorica)

v-bind – povezuje reaktivne podatke s HTML atributima i prosljeđuje ih koristeći svojstvo :attr. Može se pisati v-bind:attr ili samo :attr. Primjer:

```
<a
  :href="link.url"
  :target="link.target">
  {{link.text}}
</a>
```

U <script> dijelu upisuje se:

```
export default {
  data () {
    return {
      ...
      link: {
        url: 'http://www.google.com',
        target: '_blank',
        text: 'Google'
      },
      ...
    }
  }
}
```

U primjeru se :href povezuje s link.url koji vodi na URL resursa, u ovom slučaju www.google.com. Atribut :target se povezuje s link.target, koji kaže kako će se resurs otvoriti u prozoru. U ovom slučaju se otvara u novom prozoru. Umetanje teksta s link.text prikazuje tekst „Google“ na ekranu.



Slika 24. Izgled teksta uz primjer proslje ivanja podataka kroz v-bind direktivu (Izvor: autorica)

U primjeru v-bind direktiva se koristi kako bi reaktivno ažurirala HTML atribut.

v-if – kontrolira blok elemenata, a često se koristi za micanje HTML elemenata iz DOM stabla. Može se koristiti uz v-else-if i v-else.

v-show - kontrolira vidljiv dio bloka HTML elemenata, odnosno može se iskoristiti za skrivanje HTML elemenata u DOM stablu.

v-for – koristi se za ponavljanje odnosno iteraciju nad elementima. Dodano je v-bind svojstvo :key koje jedinstveno identificira element.

Primjer:

```
<p v-for="item in days" :key="item">
  {{ item }}
</p>
```

U <script> dijelu upisuje se:

```
export default {
  data () {
    return {
      ...
      days: ['Ponedjeljak', 'Utorak', 'Srijeda', 'Četvrtak',
        'Subota', 'Nedjelja'],
      ...
    }
  }
}
```



Slika 25. Izgled ispisa elemenata niza iz primjera koji koristi *v-for* direktivu (Izvor: autorica)

v-model – je model komponente odnosno sadrži vrijednosti unesene u komponentu. Povezuje se s unosom teksta.

S *v*-direktivama osigurano je povezivanje DOM komponenti s podacima, a da bi se osigurala reaktivnost komponente, podaci se nalaze u skriptnom dijelu unutar `data()` funkcije. Moguće je dodati podatke i izvan `data()` funkcije, no oni neće biti reaktivni.

5.4.2 *F q i c c l k*

Događaji su akcije koje se pojavljuju u sustavu, tijekom interakcije korisnika s korisničkim sučeljem. Postoje tri bitna dijela koja definiramo kod događaja:

- događaj (engl. *event*) koji želimo pratiti – npr. klik na gumb, smanjivanje ekrana, upisivanje teksta i slično
- grafička komponenta na kojoj ćemo pratiti događaj – na komponentu postavimo slušač događaja (engl. *event listener*) i pratimo korisničku interakciju s komponentom
- obrada događaja (engl. *event handler*) – dio programskog koda koji će se pozvati nakon što se događaj dogodi.

Kod povezivanja događaja s Quasar komponentom mogu se koristiti direktive (npr. *v-on*) ili skraćene naredbe (`@`). Primjer:

```
<q-btn color="primary" label="Show all" v-on:click="showAllQuotes()" />
```

ili:

```
<q-btn color="primary" label="Show all" @click="showAllQuotes()" />
```

U primjeru je prikazan slušač događaja za klik na gumb, povezan s reaktivnom `v-on` direktivom. Nakon klika na gumb poziva se metoda `showAllQuotes()` koja se nalazi unutar `methods` opcije u skriptnom dijelu datoteke.

U drugom primjeru se klikom na gumb „New item“ uvećava broj stavki za 1, a klikom na „Delete items“ se broj postavlja na 0. Ispis broja stavki radi se unutar `QCardSection` komponente. U `<script>` dijelu unutar `data()` funkcije postavljena je varijabla `items`, koja će se unutar `{{ items }}` automatski ažurirati.

```
<q-card>
  <q-card-section>
    Number of items: {{ items }}
  </q-card-section>
</q-card>
<q-btn v-on:click="items++">New item</q-btn>
<q-btn v-on:click="items=0">Delete items</q-btn>

<script>
export default {
  data () {
    return {
      items: 0
    }
  }
}
</script>
```

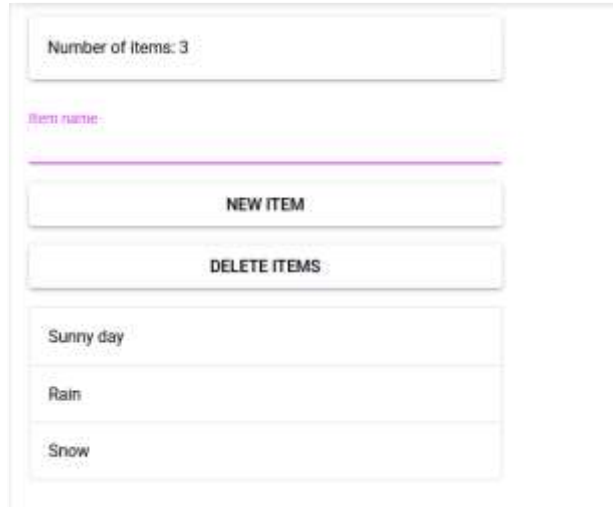


Slika 26. Izgled komponente iz primjera vezane uz v-on direktivu (Izvor: autorica)

U primjeru koji slijedi proširit ćemo dodavanje stavki s komponentama QInput, QList i QItem.

QInput postavlja polje za unos teksta, u ovom slučaju naziva stavke. QList, QItem i QItemSection postavljaju listu u kojoj će se dinamički dodavati nazivi stavki.

```
<div class="q-gutter-y-md column" style="max-width: 400px">
  <q-card>
    <q-card-section>
      Number of items: {{ items }}
    </q-card-section>
  </q-card>
  <q-input color="purple-12" v-model="text" label="Item name">
</q-input>
  <q-btn v-on:click="addNewItem()">New item</q-btn>
  <q-btn v-on:click="deleteItems();items=0">Delete items</q-btn>
  <q-list bordered separator>
    <q-item v-for="itemL in itemList" :key="itemL">
      <q-item-section>{{ itemL }}</q-item-section>
    </q-item>
  </q-list>
</div>
```



Slika 27. Primjer izgleda dinami kog dodavanja stavki (Izvor: autorica)

Komponenta `QInput` vezana je uz `v-model="text"`. Gumbi „New item“ i „Delete item“ pozivaju funkcije `addNewItem()` i `deleteItem()` u skriptnom dijelu dokumenta. Unutar `QList` definirane su `QItem` komponente koje se dinamički kreiraju prema elementima u `itemList` varijabli niza. Petlja `v-for` osigurava da se napravi `QItem` komponenta za svaki element `itemL` iz niza `itemList`.

```
<script>
import { ref } from 'vue'
export default {
  data () {
    return {
      items: 0,
      itemList: []
    }
  },
  setup () {
    return {
      text: ref('')
    }
  },
  methods: {
```



```

    addNewItem () {
      if (this.text !== '') {
        this.itemList.push(this.text)
        this.items++
        this.text = ''
      }
    },
    deleteItems () {
      this.text = ''
      this.itemList = []
    }
  },
  mounted () {
    this.addNewItem()
    this.deleteItems()
  }
}
</script>

```

Unutar skriptnog dijela radi se ubacivanje reference *import { ref }* iz Vue modela. Atribut *ref* dopušta direktan pristup DOM elementima i kreiranje reaktivnog stanja. Unutar *setup()* metode postavljeno je stanje za varijablu *text* koja je povezana s *QInput* komponentom (*v-model="text"*). Varijabla *text* sadrži tekst koji korisnik upisuje u polje za unos teksta.

Dodatne reaktivne varijable su *items* i *itemList*. Varijabla *items* povećava se s brojem klikova na gumb „New item“, odnosno to se događa u funkciji *addNewItem()*. Postavlja se na 0 prilikom klika na „Delete items“. Varijabla *itemList* je niz koji je inicijalno prazan *itemList: []*, a funkcija *push()* dodaje novu vrijednost u niz.

Metoda *addNewItem()* provjerava ako je varijabla *text* prazna, te ako nije, dodaje vrijednost u niz *itemList*.

```

this.itemList.push(this.text)

```

Funkcija `mounted()` je dio životnog ciklusa Quasar i Vue razvojnih okvira, koja se izvršava nakon učitavanja komponenti.

Dodatno se može dodati slušanje događaja kod pritiska na Enter (točnije otpuštanja tipke) na komponentu za unos teksta:

```
<q-input color="purple-12" v-on:keyup.enter="addNewItem()"
  v-model="text" label="Item name">
```

5.4.3 Funkcije

Sve funkcije pišu se u `<script>` i `</script>` dijelu `.vue` dokumenata.

U Quasaru postoje funkcije koje su dio njegovog životnog ciklusa, poput: `beforeCreated()`, `created()`, `beforeMount()`, `mounted()`, `beforeUpdate()`, `updated()`, `beforeUnmount()` i `unmounted()`. One su opisane u poglavlju Životni ciklus aplikacija u softverskom razvojnom okviru.

Funkcija `default()` vraća podrazumijevane vrijednosti koje postavljamo na početku.

Unutar funkcije `data()` nalaze se varijable čiji podaci se prate i sa svakom promjenom podataka ažuriraju se i komponente koje ih prikazuju.

Unutar objekta `methods` nalaze se funkcije koje se pozivaju iz komponenti (`v-direktive`).

Svojstvo `computed` sadrži specijalne funkcije koje vraćaju izračunate vrijednosti ili rezultate nekih logičkih operacija koje se mogu koristiti kao tip podataka koji će se reaktivno ažurirati.

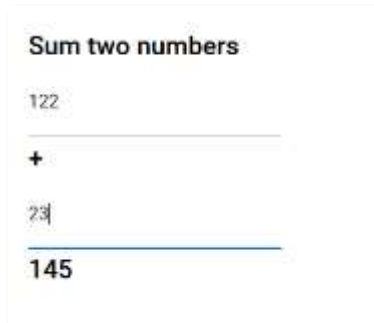
U primjeru koji slijedi koristi se svojstvo `computed` za unos i izračun zbroja dva brojeva:

```
<div class="text-h6">Sum two numbers</div>
<q-input style="max-width: 200px" v-model="firstNum"
  placeholder="First number" />
<div class="text-h6">+</div>
<q-input style="max-width: 200px" v-model="secondNum"
  placeholder="Second number" />
<div class="text-h6">{{ sum }}</div>
```

```

<script>
export default {
  data () {
    return {
      firstNum: '',
      secondNum: ''
    }
  },
  computed: {
    sum () {
      if (this.firstNum === '' || this.secondNum === '') return 0
      else return parseInt(this.firstNum) +
        parseInt(this.secondNum)
    }
  }
}
</script>

```



Slika 28. *QInput* komponente i korištenje *v-model* direktive (Izvor: autorica)

5.5 Asinkrona komunikacija sa serverskom stranom

Asinkrone funkcije u JavaScriptu koriste se kako bi se izbjeglo dugo čekanje u aplikaciji prilikom izvršavanja neke operacije, prilikom čega se sve ostale operacije zaustavljaju i aplikacija je u stanju čekanja dok se trenutna operacija ne izvrši, a tek nakon vraćanja rezultata aplikacija nastavlja

izvršavanje. Ponekad se za vrijeme izvršavanja te jedne operacije vrijeme čekanja na rezultat može bolje iskoristiti, pa se asinkronim funkcijama osigurava da se izvršavanje odvija u pozadini, dok mi možemo izvoditi druge radnje dok ne dođe rezultat izvršavanja. Obično se dugačka operacija poziva asinkronom funkcijom. Funkcija započinje operaciju i vraća kontrolu tako da aplikacija može reagirati na druge događaje. Kad je operacija gotova, funkcija nas obavještava o rezultatu.

Jedan od primjera implementacije asinkronih funkcija je u slučaju klijentsko-serverske komunikacije. U poglavlju o klijentskim *web*-tehnologijama opisan je primjer takve komunikacije korištenjem AJAX tehnologije.

Klijent pošalje upit na poslužiteljsku stranu i zatim je slobodan za izvršavanje drugih operacija sve dok ne dođe odgovor s poslužiteljske strane. Nakon dolaska odgovora od poslužitelja odgovor se prikazuje korisniku ili dalje obrađuje.

U Quasar/Vue razvojnom okruženju najčešće se koristi JavaScript biblioteka Axios (<https://axios-http.com/docs/intro>) koja služi za asinkronu komunikaciju s poslužiteljem (Vue.js *Using Axios to Consume APIs*, n.d.). Ima Promise-based API, što znači da se može koristiti s `async` i `await`.

Instalacija Axios biblioteke:

```
npm install axios
```

Axios se ugrađuje u projekt prilikom samog kreiranja Quasar projekta (potrebno ga je odabrati iz opcija), a lokacija na koju se smješta je `src/boot` folder. Također se dodaje u konfiguracijsku datoteku `quasar.config.js`, unutar `boot` dijela:

```
boot: [  
  'i18n',  
  'axios'  
],
```

Axios se koristi kod komunikacije Quasar komponenti s bilo kojim poslužiteljem, te je moguće povezati ga s API-jem na nekom poslužitelju i odraditi osnovne CRUD operacije na poslužitelju

(ovisno o dozvolama i specifikacijama *web*-servisa na poslužitelju) ili ga povezati putem HTTP protokola s nekom drugom poslužiteljskom *web*-aplikacijom.

Kod korištenja, u `<script>` dijelu `.vue` datoteka radi se import biblioteke:

```
import axios from 'axios';
```

5.6 Povezivanje aplikacije s aplikacijskim programskim sučeljem. Izvršavanje CRUD operacija

Aplikacijsko programsko sučelje (engl. *API = Application Programming Interface*) je priključna točka prema nekoj usluzi koja definira skup pravila za njeno korištenje. Pravila su vezana uz lokaciju usluge (URI), protokol koji se treba koristiti, format naredbi i podataka koje se mogu proslijediti API-ju ili se dohvaćaju s njega.

Uobičajeno se koristi REST API i HTTP protokol za komunikaciju klijenta s poslužiteljem. Na poslužiteljskoj strani nalazi se REST API aplikacija kojoj pošaljemo zahtjev (engl. *HTTP request*) s podacima ili bez njih, a ona onda prihvaća zahtjev i podatke te vraća prikladan odgovor. Zahtjevi mogu tražiti da se dohvate neki podaci (metoda GET) kao, primjerice, popis voznog reda za neku liniju, da se spreme novi podaci (metoda POST) kao, primjerice, dodavanje člana za pretplatu na listu novosti, da se ažuriraju (metoda PUT) ili brišu (metoda DELETE) neki postojeći podaci. Što će se točno dogoditi s podacima, to se definira u REST API aplikaciji.

HTTP metode POST, GET, PUT, DELETE dobro se mapiraju s CRUD operacijama: CREATE, READ, UPDATE, DELETE koje se onda izvršavaju na poslužiteljskoj strani i uobičajeno izvršavaju operacije nad trajno pohranjenim podacima.

Mapiranje CRUD operacija u aplikaciji radi se HTTP metodama:

CREATE = HTTP POST

READ = HTTP GET

UPDATE = HTTP PUT

DELETE = HTTP DELETE.

Da bi se operacije mogle izvršiti, pozivamo pojedinu HTTP metodu i po potrebi prosljeđujemo podatke, spajajući se na priključnu točku.

5.6.1 Dohvat podataka iz `rqv hqclrgn kkm c e k l u m k j " r t q i t c o u`

Na internetu se mogu naći API usluge koje se mogu koristiti neograničeno bez ikakve naknade. To su obično testni API-ji ili oni koji se mogu koristiti neograničeno ili ograničeno uz poseban ključ API KEY koji se dobije prilikom registracije, te API-ji kojima se može pristupiti uz plaćanje.

Primjer 1:

Od postojećih besplatnih API usluga koje se mogu koristiti u svrhu testiranja za praktičan primjer uzet je Random joke API koji svakim pozivom vraća jednu kratku doskočicu.

URL: https://official-joke-api.appspot.com/random_joke

API vraća JSON format podataka:

```
{
  "type": "general",
  "setup": "What do vegetarian zombies eat?",
  "punchline": "Grrrrrainnnnnssss.",
  "id": 192
}
```

U Quasaru je kreirana stranica JokesPage.vue u kojoj su dodane komponente QCard i QCardSection. Unutar QCardSection dijelova ispisuje se tekst dohvaćen iz API-ja, koji je u JSON formatu.

JSON format ima dijelove „name“:“value“ pri čemu treba dohvatiti „name“ dio, a ispisuje se ono što se nalazi u „value“ dijelu.

JSON podatak se nakon dohvata zabilježi u varijablu *joke*, te se s *joke.setup* i *joke.punchline* osigurava ispis vrijednosti („*value*“).



Slika 29. Izgled dijela stranice *JokesPage.vue* (Izvor: autorica)

U `<template>` dijelu stranice koristi se dinamičko umetanje podataka `{{ joke.setup }}` i `{{ joke.punchline }}`:

```
<q-card class="text-light-blue-3 bg-black">
  <q-card-section>
    <div class="text-h6">Random jokes</div>
  </q-card-section>
</q-card>
<q-card>
  <q-card-section>
    <div class="text-subtitle1"> {{ joke.setup }} </div>
  </q-card-section>
</q-card>
<q-card>
  <q-card-section>
    <div class="text-weight-bolder"> {{ joke.punchline }}</div>
  </q-card-section>
</q-card>
```

U `<script>` dijelu poziva se *axios* biblioteka.

```
<script>
import axios from 'axios'
```

```

export default ({
  data () {
    return {
      joke: {}
    }
  },
  methods: {
    async loadJoke () {
      await axios.get
        ('https://official-joke-api.appspot.com/random_joke')
        .then(res => {
          const result = res.data
          console.log(result)
          this.joke = result
        })
        .catch(error => {
          console.error(error)
        })
    }
  },
  mounted () {
    this.loadJoke()
  }
})
</script>

```

Da bi se s udaljene stranice učitao API, koristi se AJAX, odnosno biblioteka *axios*. Podaci s API-ja dohvaćaju se asinkrono. Node.js podržava asinkrono izvršavanje programskog koda. To znači da dok jedna funkcija kod izvršavanja može čekati na povratnu informaciju, neki se drugi dio programa može dalje izvršavati. Kad dođe povratna informacija funkciji, ona nastavlja s izvršavanjem. Može postojati više akcija koje ne čekaju jedna drugu i nije moguće predvidjeti

redosljed izvršavanja. Ovo se posebno odnosi na AJAX zahtjeve koji se izvršavaju zasebno u posebnoj niti (engl. *thread*), inače bi glavna nit bila blokirana dok ne stigne odgovor.

U slučaju navedenog primjera, funkcija *loadJoke()* ima *async* ključnu riječ koja znači da je funkcija asinkrona i da vraća takozvani *promise* objekt. *Promise* je JavaScript objekt koji sadrži uparene dijelove koda koji nešto izvršavaju i one dijelove koje ih potražuju, te obavezu da će isporučiti zahtijevane dijelove čim se izvrše. Asinkroni dohvat podataka radi s takozvanim *promise* objektima koji prate ako se asinkroni događaj dogodio ili nije, te što se dogodilo nakon toga. *Promise* objekt može biti u 2 stanja: *pending* (inicijalno stanje prije događaja), *resolved* (nakon što je operacija uspješno izvršena) i *rejected* (ako je operacija imala grešku za vrijeme izvršavanja). Ako je *promise* objekt uspješno izvršen, koristi se *.then()*, a u slučaju greške, izvršava se *catch()* (Flanagan, 2020).

Unutar funkcije *loadJoke()* poziva se funkcija *axios.get()* s ključnom riječi *await*. Ključna riječ *await* znači da će se čekati na rezultat izvršavanja funkcije *axios.get()* i kad dođe rezultat, nastavlja se s izvršavanjem *.then()* ili *.catch()*.

Unutar *.then()* rezultat dohvata s API-ja bilježi se u varijablu *res*, iz koje se čitaju podaci (*res.data*) i zapisuju u varijablu *joke*. Operator *this* je obično vezan uz trenutnu Quasar/Vue.js instancu koja se izvršava i definira način pristupa varijabli unutar `<script>` dijela datoteke.

```
.then(res => {
  const result = res.data
  this.joke = result
})
```

Rezultat izvršavanja se bilježi u varijabli *joke*: *this.joke = result*, a kako je varijabla *joke* deklarirana unutar *data()* funkcije, ona je reaktivna, pa će se njenom izmjenom automatski ažurirati i cijela stranica.

Primjer 2:

API na stranici <https://type.fit/api/quotes> isporučuje u JSON formatu čuvene izreke različitih autora. Zapis sadrži preko 1000 izreka.

Primjer:

```
[
  {
    "text": "Genius is one percent inspiration and ninety-nine percent perspiration.",
    "author": "Thomas Edison"
  },
  {
    "text": "You can observe a lot just by watching.",
    "author": "Yogi Berra"
  },
  ...
]
```

U Quasaru je kreiran primjer u kojem se dohvaća s API-ja ovaj zapis. Definirana se 3 gumba: Show all, Show one random i Show ten quotes. Odabirom prvog gumba prikazuju se sve izreke, odabirom drugog prikazuje se jedna slučajnom odabirom, a odabirom trećeg prikazuje se deset slučajno odabranih izreka.

Uz komponente QCard, QCardSection i QBtn, korištena je i komponenta QAjaxBar, koja se pojavljuje na dnu stranice i prikazuje se dok se stranica učitava.

Gumb „Show all“ prilikom klika poziva funkciju `showAllQuotes(): @click="showAllQuotes()"`

Funkcija `showAllQuotes()` postavlja `this.showCard = 'all'`. Kod prikaza izreka, ako je `showCard` postavljen na `all` (`showCard=='all'`) `v-show` je `True` i tada se prikazuje ovdje naveden dio:

```
<div v-show="showCard=='all'">
  <q-card
```

```

v-for="(quote,n) in quotes" :key="n">
<q-card-section >
  <div class="text-h6"> {{ quote.text }}</div>
  <div class="text-subtitle2">by {{ quote.author }}</div>
</q-card-section>
<q-separator dark />
</q-card>
</div>

```

Varijabla *quotes* sadrži cijeli JSON zapis u kojem su zabilježene sve izreke. Direktiva *v-for* prolazi kroz *quotes* i svakim prolazom kroz petlju čita pojedinu izreku i bilježi je u varijablu *quote*, iz koje onda izvlači tekst (*quote.text*) i autora (*quote.author*).

Na sličan način su riješeni pozivi i prikazi za funkcije *showRandomQuote()* i *showTenQuotes()*, osim što se dodatno radi odabir samo jednog zapisa koji se bilježi u varijablu *rndQuote*:

```

const rnd = Math.floor(Math.random() * this.quotes.length)
this.rndQuote = this.quotes[rnd]

```

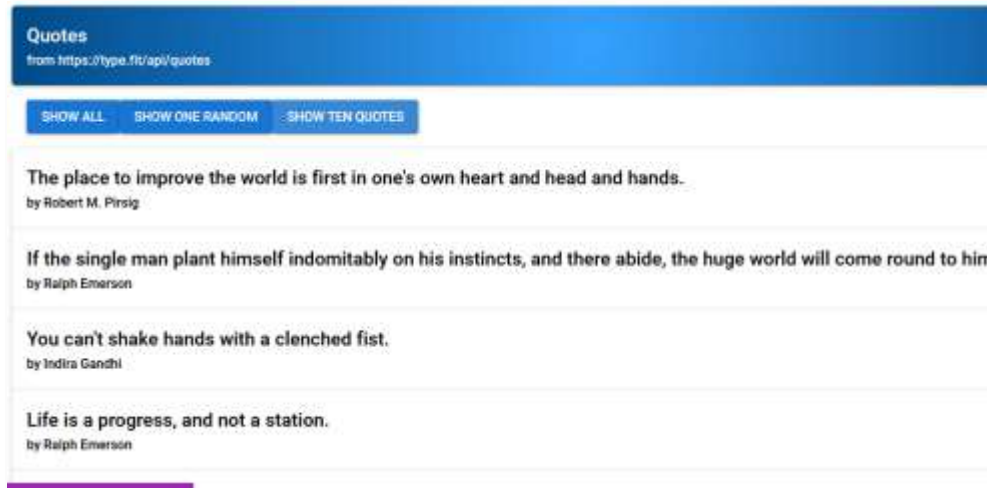
odnosno deset nasumično odabranih zapisa koji se bilježe u niz *tenQuotes*:

```

this.tenQuotes = []
while (i < 10) {
  rnd = Math.floor(Math.random() * this.quotes.length)
  this.tenQuotes.push(this.quotes[rnd])
  i++
}

```

U popisu metoda nalazi se asinkrona funkcija *loadQuotes()* koja se spaja na API i dohvaća JSON zapis u varijablu *quotes*. Funkcija se poziva unutar *created()* dijela, odnosno kod kreiranja komponenti prilikom učitavanja stranice.



Slika 30. Izgled dijela stranice QuotesPage.vue (Izvor: autorica)

```

<template>
  <q-page padding>
    <q-card class="my-card text-white"
      style="background: radial-gradient(circle, #35a2ff 0%,
        #014a88 100%)">
      <q-card-section>
        <div class="text-h6">Quotes</div>
        <div class="text-subtitle2"> from https://type.fit/api/quotes
        </div>
      </q-card-section>
    </q-card>
    <div class="q-pa-md">
      <q-ajax-bar
        ref="bar"
        position="bottom"
        color="accent"
        size="10px"
        skip-hijack
      />
      <q-btn color="primary" label="Show all"
        @click="showAllQuotes()" />

```

```

    <q-btn color="primary" label="Show one random"
      @click="showRandomQuote()" />
    <q-btn color="primary" label="Show ten quotes"
      @click="showTenQuotes()" />
  </div>
  <!-- content -->
  <div v-show="showCard=='all'">
    <q-card
      v-for="(quote,n) in quotes" :key="n">
      <q-card-section >
        <div class="text-h6"> {{ quote.text }}</div>
        <div class="text-subtitle2">by {{ quote.author }}</div>
      </q-card-section>
      <q-separator dark />
    </q-card>
  </div>
  <div v-show="showCard=='one'">
    <q-card>
      <q-card-section >
        <div class="text-h6"> {{ rndQuote.text }}</div>
        <div class="text-subtitle2">by {{ rndQuote.author }}</div>
      </q-card-section>
      <q-separator dark />
    </q-card>
  </div>
  <div v-show="showCard=='ten'">
    <q-card
      v-for="(quote,n) in tenQuotes" :key="n">
      <q-card-section >
        <div class="text-h6"> {{ quote.text }}</div>
        <div class="text-subtitle2">by {{ quote.author }}</div>
      </q-card-section>
      <q-separator dark />
    </q-card>
  </div>

```

```
    </q-card>
  </div>
</q-page>
</template>
<script>
import axios from 'axios'
import { ref } from 'vue'

const bar = ref(null)

export default ({
  data () {
    return {
      quotes: [],
      rndQuote: {},
      tenQuotes: [],
      showCard: ''
    }
  },
  methods: {
    async loadQuotes () {
      await axios.get('https://type.fit/api/quotes')
        .then(response => {
          this.quotes = response.data
        })
        .catch(error => {
          console.error(error)
        })
    },
    showAllQuotes () {
      const barRef = bar.value
      barRef.start()
    }
  }
})
```

```

setTimeout(() => {
  this.showCard = 'all'
  if (barRef) {
    barRef.stop()
  }
}, Math.random() * 3000 + 1000)
},

showRandomQuote () {
  const barRef = bar.value
  barRef.start()
  setTimeout(() => {
    console.log(this.quotes.length)
    const rnd = Math.floor(Math.random() * this.quotes.length)
    console.log(rnd)
    this.rndQuote = this.quotes[rnd]
    this.showCard = 'one'

    if (barRef) {
      barRef.stop()
    }
  }, Math.random() * 3000 + 1000)
},

showTenQuotes () {
  const barRef = bar.value
  barRef.start()
  setTimeout(() => {
    let i = 0
    let rnd = 0
    this.tenQuotes = []
    while (i < 10) {
      rnd = Math.floor(Math.random() * this.quotes.length)
      this.tenQuotes.push(this.quotes[rnd])

```

```

        i++
    }
    this.showCard = 'ten'

    if (barRef) {
        barRef.stop()
    }
    }, Math.random() * 3000 + 1000)
}
},
created () {
    this.loadQuotes()
},
setup () {
    return {
        bar
    }
}
})
</script>

```

U `<script>` dijelu, u funkciji `loadQuotes()` poziva se *axios* biblioteka za asinkroni dohvat podataka s <https://type.fit/api/quotes> API-ja.

```

async loadQuotes () {
    await axios.get('https://type.fit/api/quotes')
        .then(response => {
            this.quotes = res.data
        })
}

```

Odgovor s poslužitelja dolazi u varijablu *response*, iz koje se podaci pročitaju atributom *data* i oni se spremaju u varijablu *quotes*.

QAjaxBar komponenta prikazuje statusnu traku kod učitavanja stranice. Komponenta je kreirana u <template> dijelu:

```
<q-ajax-bar
  ref="bar"
  position="bottom"
  color="accent"
  size="10px"
  skip-hijack
/>
```

U skriptnom dijelu definirana je referenca *bar* na QAjaxBar komponentu :

```
import { ref } from 'vue'
const bar = ref(null)
```

Ta referenca koristi se za pristup komponenti preko DOM modela. Poziva se kod poziva funkcija `showAllQuotes()`, `showRandomQuote()` i `showTenQuotes` tako da se na početku funkcija poziva `start()` komponente i statusna traka pokazuje početak učitavanja. Na kraju izvođenja tih funkcija poziva se `stop()`, čime se statusna traka učitava do kraja. Dodano je praćenje vremena `setTimeout()` koje je povezano s funkcijom postavljanja varijabli za dohvat jednog podatka ili deset podataka slučajnim odabirom te prikazom tih podataka.

5.6.2 Kreiranje *crnkme klumqi "rtqitcoumqi" uwg*

U prethodnom poglavlju prikazan je dohvat podataka, odnosno operacija čitanja (READ) iz postojećih aplikacijskih programskih sučelja koja to dopuštaju. No za testiranje unosa, izmjene i brisanja najjednostavnije i najsigurnije je lokalno pokrenuti primjer aplikacije, koja će osigurati aplikacijsko programsko sučelje na koje će se klijentska aplikacija moći jednostavno spojiti. U ovom je primjeru korišten lokalni MySQL poslužitelj, te je na njemu kreirana baza s nazivom `contacts`. Kreirana je jedna tablica u toj bazi: `contact`. Struktura te tablice je:

id – atribut tipa `int`, `autoincrement`, primarni ključ

name – atribut tipa varchar (20)

surname – atribut tipa varchar (30)

phone – atribut tipa varchar (20).

Definirane su API konekcijske točke za spajanje:

HTTP metode	URI	Akcije
GET	/getContacts	get all contacts
GET	/getContact/:id	get contact by id
POST	/addContact	add new contact
PUT	/updateContact/:id	update contact by id
DELETE	/deleteContact/:id	remove contact by id

Potrebno je instalirati *Express* razvojni okvir, *mysql* biblioteku za rad s MySQL bazom, te *cors* dio koji osigurava pristup poslužiteljskoj aplikaciji s određenih domena (u primjeru je stavljen origin: '*' za sve domene).

```
npm install express mysql cors --save
```

Poslužiteljski dio aplikacije u Node.js/Express razvojnom okviru:

```
var express = require('express');
var app = express();
var cors = require('cors')
var bodyParser = require('body-parser');
const dbConfig = require("../config/db.config.js");
var mysql = require('mysql');

app.use(bodyParser.json());
```

```

app.use(bodyParser.urlencoded({
  extended: false
}));
app.use(cors({origin: '*'}))

// connection configurations
var dbConn = mysql.createConnection({
  host: dbConfig.HOST,
  user: dbConfig.USER,
  password: dbConfig.PASSWORD,
  database: dbConfig.DB
});

// connect to database
dbConn.connect();

app.get('/getContact', function (request, response) {
  dbConn.query('SELECT * FROM contact', function (error, results,
fields) {
    if (error) throw error;
    return response.send({data: results, message:
      'list of all contacts.' });
  });
});

app.get('/getContact/:id', function (request, response) {
  let contact_id = request.params.id;
  if (!contact_id) {
    return response.status(400).send({ error: true, message:
      'Please provide contact_id' });
  }
  dbConn.query('SELECT * FROM contact where id=?', contact_id,
    function (error, results, fields) {

```

```

        if (error) throw error;
        return response.send({ data: results[0], message:
            'one contact is found.' });
    });
});

app.post('/addContact', function (request, response) {
    const data = request.body;
    contact = [[data.name, data.surname, data.phone]]
    dbConn.query('INSERT INTO contact (name, surname, phone) VALUES ?',
        [contact], function (error, results, fields) {
            if (error) throw error;
            return response.send({ error: false, data: results, message:
                'contact is added.' });
        });
});

app.put('/updateContact/:id', function (request, response) {
    let contact_id = request.params.id;
    if (!contact_id) {
        return response.status(400).send({ error: true, message:
            'Please provide contact_id' });
    }
    const data = request.body;
    dbConn.query("UPDATE contact SET name=?, surname=?, phone=?
        WHERE id = ?", [data.name, data.surname, data.phone, contact_id],
        function (error, results) {
            if (error) throw error;
            return response.send({ error: false, data: results, message:
                'contact is updated.' });
        });
});
});

```

```

app.delete('/deleteContact/:id', function (request, response) {
  let contact_id = request.params.id;
  if (!contact_id) {
    return response.status(400).send({ error: true, message:
      'Please provide contact_id' });
  }
  dbConn.query("DELETE FROM contact WHERE id = ?", [contact_id],
  function (error, results) {
    if (error) throw error;
    return response.send({ error: false, data: results, message:
      'contact is deleted.' });
  });
});
// set port
app.listen(3000, function () {
  console.log('Node app is running on port 3000');
});
module.exports = app;

```

Datoteka db.config.js:

```

module.exports = {
  HOST: "",
  USER: "",
  PASSWORD: "",
  DB: ""
};

```

U db.config datoteku treba unijeti točne podatke za pristup bazi.

Poslužiteljska aplikacija se pokreće iz zasebne komandne linije:

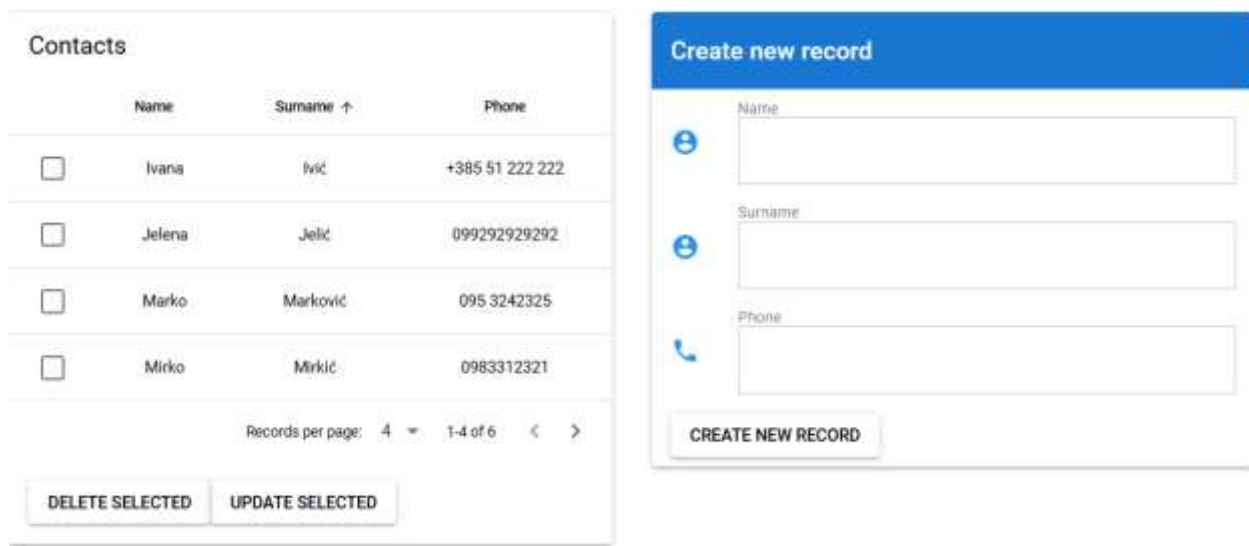
```
node indeks.js
```

Poslužitelj javlja da je aplikacija pokrenuta na priključku 3000.

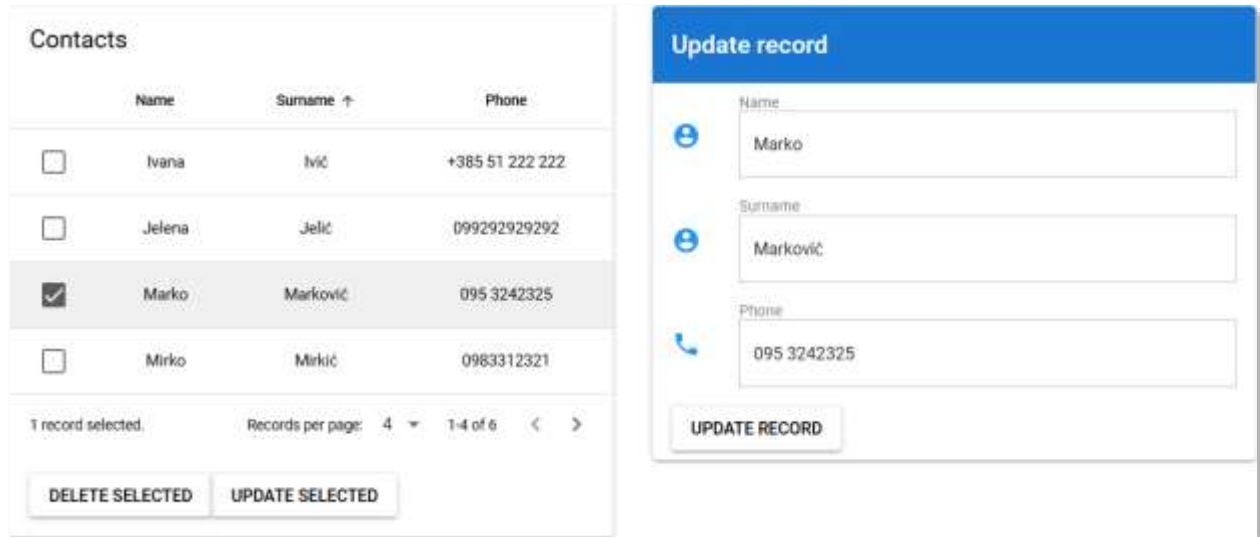
5.6.3 Kreiranje aplikacije s CRUD operacijama

Prethodno kreiran REST API iskoristit će se kao poslužiteljska aplikacija za rad s podacima. Klijentski dio *web*-aplikacije će koristiti pristupne točke API-ja za slanje i dohvat podataka. Pritom je dovoljno da klijentska aplikacija ima informaciju na koji URL se treba spojiti, koja HTTP metoda se koristi za prijenos te u kojem formatu podaci moraju biti za unos, odnosno koji format podataka aplikacija može očekivati od poslužitelja.

Za primjer je kreirana CRUD aplikacija za prikaz, kreiranje, ažuriranje i brisanje kontakata iz baze podataka kojoj se pristupa preko aplikacijskog sučelja ranije kreirane poslužiteljske aplikacije. Da bi se ova aplikacija mogla ispravno izvršavati, treba pokrenuti ranije napravljen API na poslužiteljskoj strani.



Slika 31. Gotova Quasar aplikacija s prikazom i unosom podataka (Izvor: autorica)



Slika 32. Gotova Quasar aplikacija s odabirom i ažuriranjem podataka (Izvor: autorica)

U nastavku je postavljen kompletan programski kod s CRUD operacijama.

```

<template>
  <q-page padding>
    <!-- content -->
    <div class="q-pa-md row">
      <div class="q-pa-md">
        <q-card class="my-card">
          <q-table
            title="Contacts"
            :rows="contacts"
            :columns="columns"
            row-key="id"
            :pagination="initialPagination"
            selection="single"
            v-model:selected="selected"
          />
          <q-card-section>
            <q-btn @click="deleteContact(selected)">Delete selected
          </q-btn>
        </q-card>
      </div>
    </div>
  </template>

```

```

    <q-btn @click="readContact(selected)">Update selected
  </q-btn>
</q-card-section>
</q-card>
</div>
<div class="q-pa-md">
  <q-card class="my-card">
    <q-card-section class="bg-primary text-white">
      <div class="text-h6">{{ labelContact }}</div>
    </q-card-section>

    <q-list>
      <q-item clickable>
        <q-item-section avatar>
          <q-icon color="blue" name="account_circle" />
        </q-item-section>
        <q-item-section>
          <q-item-label caption>Name</q-item-label>
          <q-input square outlined v-model="textName" />
        </q-item-section>
      </q-item>

      <q-item clickable>
        <q-item-section avatar>
          <q-icon color="blue" name="account_circle" />
        </q-item-section>
        <q-item-section>
          <q-item-label caption>Surname</q-item-label>
          <q-input square outlined v-model="textSurname" />
        </q-item-section>
      </q-item>

      <q-item clickable>

```



```

        <q-item-section avatar>
            <q-icon color="blue" name="phone" />
        </q-item-section>
        <q-item-section>
            <q-item-label caption>Phone</q-item-label>
            <q-input square outlined v-model="textPhone" />
        </q-item-section>
    </q-item>
</q-list>
<q-item>
    <q-btn @click="(isNew)? createContact(): updateContact()"
        v-model="newc">{{ labelContact }}</q-btn>
</q-item>
</q-card>
</div>
</div>
</q-page>
</template>

<script>
import { ref } from 'vue'
import axios from 'axios'
export default ({
  data () {
    const selected = ref([])
    return {
      textName: '',
      textSurname: '',
      textPhone: '',
      id: 0,
      labelContact: 'Create new record',
      isNew: true,
      selected,

```

```

contacts: [],
columns: [
  { name: 'name', align: 'center', label: 'Name',
    field: contact => contact.name, sortable: true },
  { name: 'surname', align: 'center', label: 'Surname',
    field: contact => contact.surname, sortable: true },
  { name: 'phone', align: 'center', label: 'Phone',
    field: contact => contact.phone, sortable: false }
]
}
},

setup () {
  return {
    initialPagination: {
      sortBy: 'surname',
      descending: false,
      page: 1,
      rowsPerPage: 4
    }
  }
},

methods: {
  async loadContacts () {
    await axios.get('http://localhost:3000/getContact/')
    .then(result => {
      console.log(result.data)
      this.contacts = result.data.data
    })
    .catch(error => {
      console.error(error)
    })
  },

```

```

async deleteContact (selected) {
  console.log(JSON.stringify(selected))
  console.log(selected[0].id)
  const id = selected[0].id
  await axios.delete('http://localhost:3000/deleteContact/' + id)
    .then(result => {
      console.log(result.data.data)
    })
    .catch(error => {
      console.error(error)
    })
  this.loadContacts()
},
async createContact () {
  const formData = { name: this.textName, surname: this.textSurname,
    phone: this.textPhone }
  console.log(JSON.parse(JSON.stringify(formData)))
  await axios.post('http://localhost:3000/addContact/', formData)
    .then(result => {
      console.log(result.data.data)
    })
    .catch(error => {
      console.error(error)
    })
  this.loadContacts()
  this.resetContact()
},
async updateContact () {
  const formData = { name: this.textName, surname: this.textSurname,
    phone: this.textPhone }
  console.log(JSON.parse(JSON.stringify(formData)))
  await axios.put('http://localhost:3000/updateContact/' +

```

```

    this.id, formData)
    .then(result => {
        console.log(result.data.data)
    })
    .catch(error => {
        console.error(error)
    })
    this.loadContacts()
    this.resetContact()
},
resetContact () {
    this.textName = ''
    this.textSurname = ''
    this.textPhone = ''
    this.id = 0
    this.labelContact = 'Create new record'
    this.isnew = true
},
readContact (selected) {
    const id = selected[0].id
    const name = selected[0].name
    const surname = selected[0].surname
    const phone = selected[0].phone
    this.id = id
    this.textName = name
    this.textSurname = surname
    this.textPhone = phone
    this.labelContact = 'Update record'
    this.isnew = false
}
},
created () {
    this.loadContacts()
}

```

```
})  
</script>
```

Za prikaz kontakata, u <template> dijelu, kreirana je tablica komponentom QTable te je ona popunjena podacima kroz povezivanje (v-bind ili :) s varijablama rows i columns. Varijabla rows sadrži listu kontakata „contacts“ koji se popunjavaju u skriptnom dijelu. Varijabla row-key služi za jedinstveno definiranje objekta koji se zapisuje u red tablice, kako bi se osiguralo ažuriranje i brisanje. Definirano je da se u slučaju većeg broja prikaza postavlja mogućnost listanja stranica (:pagination) te mogućnost odabira retka kroz :selected. Za reaktivno praćenje stranica i odabira postavljene su v-direktive koje se aktiviraju u slučaju bilo kakvih izmjena i događaja.

```
<q-table  
  title="Contacts"  
  :rows="contacts"  
  :columns="columns"  
  row-key="id"  
  :pagination="initialPagination"  
  selection="single"  
  v-model:selected="selected"  
>
```

U <script> dijelu funkcija loadContacts() poziva API vraćeni rezultati (*results.data.data*) se smještaju u varijablu contacts.

```
async loadContacts () {  
  await axios.get('http://localhost:3000/getContact/')  
    .then(result => {  
      this.contacts = result.data.data  
    })  
}
```

Ova funkcija se inicijalno poziva iz funkcije created() koja se automatski izvršava kao dio životnog ciklusa Quasar aplikacije, a zatim i iz funkcija createContact(), updateContact() i deleteContact(), čime se dobiva ažuriranje varijable contacts, a time i cijele tablice.

Kod unosa novog kontakta ili ažuriranja postojećeg kontakta, iskorištene su komponente QInput, te su povezane v-model direktivama s varijablama textName, textSurname, textPhone. Dodana je varijabla isNew koja prati ako se radi o unosu novog kontakta ili ažuriranju postojećeg kontakta. U slučaju unosa novog kontakta, varijabla isNew ima vrijednost True, pa se kod klika na gumb poziva funkcija createContact(). U slučaju ažuriranja kontakta poziva se funkcija updateContact()

```
<q-btn @click="(isNew)? createContact(): updateContact()"
  v-model="newc">{{ labelContact }}
```

U skriptnom dijelu u funkciji createContact() kreira se JSON objekt formData koji će se proslijediti API-ju.

```
async createContact () {
  const formData = {
    name: this.textName,
    surname: this.textSurname,
    phone: this.textPhone
  }
  await axios.post('http://localhost:3000/addContact/', formData)
```

Na sličan način je napravljeno ažuriranje, samo što se kod ažuriranja mora slati i id kontakta kojeg se ažurira.

```
async updateContact () {
  const formData = {
    name: this.textName,
    surname: this.textSurname,
    phone: this.textPhone
  }
  await axios.put('http://localhost:3000/updateContact/' + this.id,
  formData)
```

Za brisanje je dovoljno proslijediti id kontakta kojeg se briše:

```
async deleteContact (selected) {  
  const id = selected[0].id  
  await axios.delete('http://localhost:3000/deleteContact/' + id)
```

6 Razvojno i produkcijsko okruženje

Razvoj aplikacije je kompleksan zadatak koji je bitno odraditi u razvojnom okruženju (engl. *development environment*). U takvom se okruženju aplikacija može razvijati, testirati i otklanjati greške prije nego bude puštena na korištenje, u produkciju.

Kako se razvoj softvera odvija u etapama, uobičajeno je imati više okruženja koje bi odgovarale pojedinim fazama implementacije. Uz razvojno okruženje na kojem se aktivno radi na novim funkcionalnostima aplikacije, može postojati i testno okruženje na kojem se rade testiranja gotove aplikacije, staging okruženje u kojem se osigurava da sve nove izmjene rade s već isporučenom aplikacijom u produkciji te produkcijsko okruženje u kojem se *web*-aplikacija izvršava (*Difference between development, stage, and production*, 2019).

Ovakva podjela okruženja može se primijeniti i na Git-sustav za kontrolu verzija programskog koda. Na Git-sustavu se u tom slučaju obično kreiraju tri grane: *development*, *staging* i *main* (*production*).

Development grana se brzo razvija i čuva sve novonapravljene izmjene. Prije prebacivanja izmjena u produkciju, one se spajaju sa *staging* granom, kako bi se provjerilo utječu li nove izmjene na programski kod na strani produkcije. Tek nakon toga izmjene mogu ići u produkciju.

Udaljeni Git-sustavi za upravljanje repozitorijima poput GitHuba ili GitLaba imaju uključenu mogućnost korištenja automatizirane CI/CD (engl. *Continuous Integration/Continuous Delivery*) prakse. CI omogućuje brzu integraciju novog koda s postojećim, uz automatizirane i ručne provjere kvalitete programskog koda. CD osigurava automatsku isporuku u produkcijsko okruženje, jednom kad je programski kod provjeren (Camden *et al.*, 2020).

U Quasar okruženju pokretanje aplikacije odvija se naredbom

```
quasar dev
```

što označava pokretanje u razvojnom okruženju (Quasar *General deployment*, n.d.).

Jednom kad je aplikacija razvijena radi se

```
quasar build
```

Ova naredba kreira dist folder u kojem se nalaze sve datoteke koje se mogu postaviti na produkcijski *web*-poslužitelj.

7 LITERATURA

- Adhikary, T. (2021) *JSON for beginners – JavaScript Object Notation Explained in Plain English*, *freeCodeCamp.org*. freeCodeCamp.org. [mrežno]. Dostupno na: <https://www.freecodecamp.org/news/what-is-json-a-json-file-example/> (Pristupljeno: 15. 11. 2022.)
- Banga, S. (2022) *What is web-application architecture? components, models, and types*, *Hackr.io*. [mrežno]. Dostupno na: <https://hackr.io/blog/web-application-architecture-definition-models-types-and-more> (Pristupljeno: 19. 1. 2023.)
- Camden, R. *et al.* (2020) *Front-end development projects with Vue.js: Learn to build scalable web-applications and dynamic user interfaces with Vue 2*. Birmingham, UK: Packt Publishing Ltd, pp. 99–107, 581-585.
- Chacon, S. and Straub, B. (2014) *Pro Git book*. Berkeley, CA: Apress. [mrežno]. Dostupno na: <https://git-scm.com/book/en/v2> (Pristupljeno: 8. 1. 2023.)
- Felke-Morris, T. (2017) *Web-development and design foundations with HTML5*. 8th edn. Boston, USA: Pearson.
- Flanagan, D. (2020) *JavaScript: The definitive guide*. Boston: O'Reilly Media.
- GitHub Docs, *About Git* (n.d.). [mrežno]. Dostupno na: <https://docs.github.com/en/get-started/using-git/about-git> (Pristupljeno: 18. 11. 2022.)
- Gupta, D. (2022) *CSS introduction*. GeeksforGeeks. [mrežno]. Dostupno na: <https://www.geeksforgeeks.org/css-introduction/> (Pristupljeno: 20. 12. 2022.)
- IBM (2022) *XML Syntax Rules*. Dostupno na: <https://www.ibm.com/docs/en/scbn?topic=syntax-xml-rules> (Pristupljeno: 15. 11. 2022.)

HTTP Messages (2022). MDN Web-Docs. Mozilla. [mrežno]. Dostupno na:
<https://developer.mozilla.org/en-US/docs/Web/HTTP/Messages> (Pristupljeno: 12. 1. 2023.)

Morgan, J. (2018) *Simplifying JavaScript: Writing Modern JavaScript with ES5, ES6, and Beyond*, The Pragmatic Programmers, LLC.

Visual Studio Code (2021) *Documentation for Visual Studio Code*. Microsoft. [mrežno].
Dostupno na: <https://code.visualstudio.com/docs> (Pristupljeno: 19. 1. 2023.)

Difference between development, stage, and production (2019), *DEV Community*. DEV Community. [mrežno]. Dostupno na: <https://dev.to/flippedcoding/difference-between-development-stage-and-production-d0p> (Pristupljeno: 20. 1. 2023.)

Node.js V17.9.1 documentation (n.d.) *Index | Node.js v17.9.1 Documentation*. [mrežno].
Dostupno na: <https://nodejs.org/docs/latest-v17.x/api/> (Pristupljeno: 19. 1. 2023.)

Quasar Framework (n.d.) *Card*. [mrežno]. Dostupno na: <https://quasar.dev/vue-components/card>
(Pristupljeno: 21. 12. 2022.)

Quasar Framework (n.d.) *Introduction to Flexbox*. [mrežno]. Dostupno na:
<https://quasar.dev/layout/grid/introduction-to-flexbox> (Pristupljeno: 21. 12. 2022.)

Quasar Framework (n.d.) *How to use vue*. [mrežno]. Dostupno na: <https://quasar.dev/start/how-to-use-vue> (Pristupljeno: 21. 12. 2022.)

Quasar Framework (n.d.) *Routing with Layouts and Pages*. [mrežno]. Dostupno na:
<https://quasar.dev/layout/routing-with-layouts-and-pages> (Pristupljeno: 21. 12. 2022.)

Quasar Framework (n.d.) *General deployment*. [mrežno]. Dostupno na:
<https://quasar.dev/quasar-cli-vite/developing-spa/deploying#general-deployment>
(Pristupljeno: 12. 1. 2023.)

Vue.js (n.d.) *Using Axios to Consume APIs*, [mrežno]. Dostupno na:
<https://v2.vuejs.org/v2/cookbook/using-axios-to-consume-apis.html> (Pristupljeno: 8. 1. 2023.)

Vue.js (n.d.) *Lifecycle Hooks*, [mrežno]. Dostupno na
<https://vuejs.org/guide/essentials/lifecycle.html> (Pristupljeno: 20. 12. 2022.)

Vue.js (n.d.) *Options: Lifecycle*, [mrežno]. Dostupno na <https://vuejs.org/api/options-lifecycle.html> (Pristupljeno: 12. 1. 2023.)

W3Schools.com (n.d.) *AJAX Introduction*. [mrežno]. Dostupno na:
https://www.w3schools.com/js/js_ajax_intro.asp (Pristupljeno: 8. 1. 2023.)

W3Schools.com (n.d.) *HTML DOM Documents*. [mrežno]. Dostupno na:
https://www.w3schools.com/jsref/dom_obj_document.asp (Pristupljeno: 8. 1. 2023.)

8 POPIS SLIKA

Slika 1. Unos adrese resursa (URL) u web-preglednik (Izvor: autorica)	9
Slika 2. Alat za razvijatelje aplikacija (Developer Tools) u web-pregledniku Mozilla Firefox (Izvor: autorica)	10
Slika 3. Visual Studio Code editor - opcija otvaranja foldera (Izvor: autorica)	17
Slika 4. Popis programskih jezika za naredbeni redak i izvršavanje naredbe git u terminalu (Izvor: autorica)	18
Slika 5. Pokretanje naredbe node u terminalu Visual Studio Code alata (Izvor: autorica)	20
Slika 6. Traženje system environment varijable u Windows OS-u (Izvor: autorica)	20
Slika 7. Uređivanje sistemske varijable Path (Izvor: autorica).....	21
Slika 8. Početna stranica Quasar razvojnog okvira (Izvor: autorica).....	25
Slika 9. Prikaz konfliktne situacije prilikom dohvata izmjena s git pull naredbom (Izvor: autorica)	30
Slika 10. Izgled jednostavnog HTML dokumenta prikazanog u web-pregledniku (Izvor: autorica)	36
Slika 11. Prikaz HTML elemenata za unos teksta (Izvor: autorica)	39
Slika 12. Prikaz HTML dokumenta s jednostavnim CSS stilovima (Izvor: autorica).....	42
Slika 13. Prikaz hijerarhije DOM objekata za HTML dokument (Izvor: W3Schools HTML DOM Documents, n.d.)	44
Slika 14. Aplikacija koja dinamički dodaje nove HTML elemente (Izvor: autorica)	54

Slika 15. AJAX komunikacija između klijenta i poslužitelja (Izvor: autorica).....	58
Slika 16. Životni ciklus Quasar/Vue.js aplikacija (Izvor: Vue.js. Lifecycle Hooks (20. 12. 2022.))	66
Slika 17. Dva moguća rasporeda elemenata u Quasar razvojnom okviru. Lijevo: ' hhh lpr fff' Desno: 'lhr lpr fff' (Izvor: autorica).....	68
Slika 18. Raspored na početnom prozoru u Quasar razvojnom okviru (Izvor: autorica)	70
Slika 19 Izgled komponente QToolbar (Izvor: autorica).....	74
Slika 20. Dio službene Quasar dokumentacije s primjerima komponenata (Izvor: (Quasar Card n.d.)	76
Slika 21 Primjer programskog koda za komponentu u Quasar dokumentaciji na webu -(Izvor: Quasar Card n.d.)	78
Slika 22. Izgled teksta u komponenti definiranog kroz v-text direktivu (Izvor: autorica)	80
Slika 23. HTML kod ubačen kroz v-html direktivu (Izvor: autorica)	81
Slika 24. Izgled teksta uz primjer prosljeđivanja podataka kroz v-bind direktivu (Izvor: autorica)	82
Slika 25. Izgled ispisa elemenata niza iz primjera koji koristi v-for direktivu (Izvor: autorica)..	83
Slika 26. Izgled komponente iz primjera vezane uz v-on direktivu (Izvor: autorica)	85
Slika 27 Primjer izgleda dinamičkog dodavanja stavki (Izvor: autorica).....	86
Slika 28. QInput komponente i korištenje v-model direktive (Izvor: autorica).....	89
Slika 29. Izgled dijela stranice JokesPage.vue (Izvor: autorica)	93
Slika 30. Izgled dijela stranice QuotesPage.vue (Izvor: autorica)	98

Slika 31. Gotova Quasar aplikacija s prikazom i unosom podataka (Izvor: autorica)..... 108

Slika 32. Gotova Quasar aplikacija s odabirom i ažuriranjem podataka (Izvor: autorica)..... 109