Razvoj prilagodljivih korisničkih sučelja tipa WWWUI

Kolić, Teo

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Teo Kolić

RAZVOJ PRILAGODLJIVIH KORISNIČKIH SUČELJA TIPA WWWUI

ZAVRŠNI RAD

Zagreb, 2023.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Teo Kolić

Matični broj: 0016130984

Studij: ***Primjena informacijske tehnologije u poslovanju***

RAZVOJ PRILAGODLJIVIH KORISNIČKIH SUČELJA TIPA WWWUI

ZAVRŠNI RAD

Mentor:

Izv. prof. dr. sc. Mario Konecki

Zagreb, Studeni 2023.

Teo Kolić

Izjava o izvornosti

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

Autor/Autorica potvrdio/potvrdila prihvaćanjem odredbi u sustavu FOI-radovi

Sažetak

Tema ovog rada je upoznavanje sa razvojem korisničkog sučelja (UI) u Web-u. Opis promjena standarda u korisničkim sučeljima kroz vrijeme. Opis problema na koje programeri nailaze pri izradi Web korisničkih sučelja. Ukazivanje novih korisničkih potreba i navika te kako iste utječu na pristup izrade Web stranica. Važnost prilagodljivih (engl. “responsive”) korisničkih sučelja. Različite vrste pristupa za mobilne ekrane, tablete, monitore. Uloga CSS-a i njegov napredak. Cilj je upoznati alate za izradu prilagodljivih korisničkih sučelja, načine testiranja i pronalaženja grešaka (engl. Debugging) i kroz praktičan primjer prikazati implementaciju modernih alata za izradu prilagodljivih korisničkih sučelja te upoznavanje sa Angular razvojnim okvirom.

Ključne riječi: Angular; HTML; CSS; Responsive; UI;

Sadržaj

1. Uvod

Opis evolucije web korisničkih sučelja. Od dolaska interneta i prvih web stranica Web development je doživio velike transformacije i optimizacije. Prelazak sa statčnih, fiksnih sučelja na sučelja koja se prilagođavaju različitim uređajima, razlučivostima i veličinama ekrana. U ovom radu naznačit ću ključne momente u evoluciji web korisničkih sučelja kao što su CSS, medijalni upiti i pri. Opisat ću i pristup traženju grešaka pomoću razvojnih alata u pregledniku. Kroz ovaj rad izradit ću prilagodljivo korisničko sučelje koristeći moderne alate u Angular razvojnom okviru koji su danas standard u web developmentu te ću opisati implementaciju Flexbox-a, CSS Grid-a i medijalnih upita te objasniti koje probleme riješavaju.

2. Jezici

2.1. HTML i CSS

HTML (Hypertext Markup Language) je standardni jezik za prikazivanje dokumenata u Web pregledniku. Pomoću HTML-a preglednik interpretira i slaže tekst, slike i ostale audio-vizualne elemente koje vidimo na Web stranicama i aplikacijama. Za svaki HTML element postoje predefinirane zadane karakteristike i izgled. Prva javno dostupna verzija sastojala se od 18 elemenata, od kojih se 11 koristi još i danas. Inačica koja je trenutno aktualna nastala je 2014. godine te se naziva HTML5. To je unaprijeđena verzija HTML-a sa novim elementima za multimediju i novim semantičkim elementima za strukturiranje stranice. Semantički elementi uvelike olakšavaju snalaženje u strukturi dokumenta jer je odmah vidljivo što taj element u datoteci predstavlja u pregledniku. Neki od semantičkih elemenata su: „article“, „aside“, „nav“, „main“, „header“, „footer“, „audio“, „video“, „canvas“… Uz nove strukturne elemente, dodane su i nove vrste ulaznih elemenata (engl. „Input tag“) koje se automatski prikazuju u pregledniku, ovisno o tipu, kao skočni prozor ili padajući izbornik. Neki od novih ulaznih tipova su: „color“, „date“, „month“, „email“, „time“, „search“, „range“… Osim dosadašnjih događaja koje je stariji HTML pružao, sa ovom verzijom omogućeni su novi događaji: „onblur“, „onabort“, „offline“…

Osnovna struktura HTML dokumenta sastoji se se od slijedećih elemenata: „html“, „head“ i „body“. „Head“ i „body“ se nalaze unutar „html“ elementa. U „head“ element se stavljaju metapodaci (engl. „metadata“) unutar svojih tag-ova („title“, „style“, „meta“, „link“, „script“ i „base“), a kostur aplikacije (engl. „template“), odnosno onaj dio koji je vidljiv na ekranu piše se unutar „body“ elementa. Ova struktura biti će prikazana na praktičnom primjeru u ovom radu.

HTML5 radi u preglednicima na mobilnim uređajima i tablet računalima, dok starije verzije nisu imale podršku. Više nema potrebe za dodatke kao što su „Adobe Flash Reader“ za podršku multimedije. Dopušta pregledniku da direktno koristi JavaScript preko Web radnika (engl. „Web Worker API“). Omogućuje dohvaćanje lokacije korisnika preko JS GeoLocation API-a. (API – Application programming interface). HTML5 je lagan i brz, te se pokreće isključivo u Web preglednicima stoga nije potreban nikakav novi fizički uređaj ili drugi program za pokretanje. Znani problem koji se prije događao bio je vezan za nekompaktibilnost starijih preglednika sa ovom verzijom HTML-a. Bilo je potrebno testirati kod na svakom pregledniku posebno da bi se provjerila funkcionalnost i često je bilo potrebno više koda za pokriti sve

slučajeve u svim preglednicima, ali danas gotovo svi preglednici podržavaju HTML5. Kako je već spomenuto, HTML elementi imaju svoj predefiniran izgled i bez dodatnih alata nije moguće promijeniti ga. Tu u pomoć uskače CSS.

CSS – (Cascading Style Sheets) je standardni jezik za opisivanje vizualne prezentacije Web stranica i aplikacija. On određuje kako i gdje će koji HTML element biti prikazan. Pomoću CSS-a možemo promijeniti font, boje, veličinu i/ili razmak između elemenata, a moguće je i dodavanje animacija kako bi sadržaj na stranici bio privlačniji i zanimljiviji korisniku. Prvo što je potrebno znati da bi ga koristili je znati povezati ga sa HTML datotekom, a to je moguće na tri načina.

Prvi način je pisati CSS unutar HTML dokumenta u HTML oznaku. (engl. „Inline CSS“).

```
<p style=”color: red; font-size:12px;”>Lorem ipsum</p>
```

Drugi način je pisati CSS ubacivanjem „style“ oznake unutar „head“ oznake na početku HTML dokumenta.

```
<html>
      <head>
            <style type=”text/css”>
                  p{
                    color:red;
                    font-size: 2rem;
                   }
            </style>
      </head>
      .
      .
</html>
```

Treći način, povezivanje s vanjskim dokumentom, koji se najčešće koristi radi preglednosti i razdvajanja briga (engl. „Separation of concerns“). To je jedan od uzoraka dizajna kojem je cilj razdvojiti aplikaciju u više manjih cjelina od kojih se svaka brine za jedan posao i nema doticaja sa poslom druge cjeline. Vanjski CSS dokument se povezuje putem „link“ oznake unutar „head“ oznake u HTML dokumentu.

```
<html>
      <head>
            <link rel=”stylesheet” type=”text/css” href=”styles.css”>
```

```
            <link rel="stylesheet" type="text/css" href="header.css">
        </head>
        .
        .
        .
    </html>
```

Dobra je praksa razdvojiti CSS u više dokumenata tako da svaki opisuje jednu cjelinu strukture. Sve HTML elemente s istom oznakom možemo obuhvatiti jednostavno, kao u drugom načinu. Pisanjem vitičaste zagrade odmah nakon naziva oznake elementa. Ako pak želimo biti malo preciznije te obuhvatiti samo neke elemente tada unutar njihove oznake trebamo dodati ključnu riječ „class“, a unutar CSS dokumenta prije staviti prefiks „.“ koji označava da se radi o *class* selektoru. Ako je potrebno dohvatiti neki jedinstveni element, tada koristimo ključnu riječ „id“, a u CSS se dodaje prefiks „#“. Id se danas gotovo uopće ne koristi .

```
<p></p>    <div class="hidden"></div>  <button id="login-btn"></button>

p {              .hidden {                    #login-btn {
  color:red;        visibility: hidden;         border: 1px solid red;
}                  }                          }
```

Pisanje CSS koda izgleda jednostavno, ali često se dogodi da se stvari brzo zakompliciraju, što radi neispravno strukturiranog HTML koda, što radi nepoznavanja načina ponašanja i prioriteta. Najveći prioritet ima css pisan unutar html oznake (engl. inline), zatim slijede elementi sa „id“ prefiksom, onda elementi sa „class“ prefiksom, a najniži prioritet imaju selektori html oznaka. Uz to, CSS se čita linearno, od vrha prema dnu, a to znači da ako smo isti element dekorirali više puta, primijenit će se onaj stil koji je zadnji napisan. Ako želimo pregaziti (engl. „override“) prioritet nekog selektora, to možemo napraviti pomoću ključne riječi „important“ koja se piše sa „!“ ispred te se dodaje na kraj nekog stila.

```
p {
    font-size: 50px !important;
}
```

Jedan od problema u CSS-u je dohvaćanje ugniježđenih elemenata, odnosno nemogućnost pisanja ugniježđenih selektora. Da bi dohvatili element treće razine, trebamo prvo napisati selektore za prve dvije i nemamo opciju dekoriranja nijednog elementa osim onoga kojeg dohvaća zadnji selektor. Da bi dekorirali bilo koji od prva dva potrebno je napisati posebno selektor za prvi i za drugi.

```
.parent {
  background-color: red;
```

```
}

.parent.child-one {
  background-color: green;
}

.parent.child-one.child-two {
  background-color: yellow;
}
```

Ovaj način je repetativan i nepregledan. Developer je prisiljen pisati više koda i nezahvalno je raditi promjene unutar neke ugniježđene razine selektora, ali postoji rješenje i za to.

SASS (Syntactically Awesome Style Sheets) je skriptni jezik koji se sastavlja (engl. „compile“) ili interpretira u CSS. Dizajnirao ga je Hampton Catlin, a izradila Natalie Weizenbaum. SCSS je glavna sintaksa za SASS koja je izgrađena na postojećoj CSS sintaksi, koristi se zagradama i interpunkcijama istim kao i običan CSS. SCSS je unaprjeđenje za CSS isto kao što je HTML5 za HTML. Donosi nam mnogo novih mogućnosti i poboljšanja. Jednostavniji je za pisanje i pregledniji za čitanje. Isto tako omogućava nam dinamičku kalkulaciju visina i širina elemenata pomoću korištenja matematičkih operacija, ali i omogućuje ugnježđivanje selektora te dekoriranja svakog posebno unutar njihovih deklaracija, a samim time je jasnije koji je element unutar kojeg u HTML strukturi. Još jedna od novih mogućnosti koje SCSS nudi su varijable koje se označavaju sa prefiksom „$“ te ih možemo uvesti i koristiti na više mjesta u slučaju da postoji potreba za promjenom treba biti odrađena samo na jednom mjestu dok bi u običnom CSS kodu morali mijenjati na više mjesta. Uz varijable, možemo uvesti i cijeli drugi SCSS dokument pomoću ključne riječi „import“ koja se piše sa prefiksom „@“. Zbog svih prednosti koje nudi SCSS je puno stabilniji i sigurniji te lakši za održavanje.

Dohvaćanje ugniježđenih elemenata, korištenje varijabli i dinamičkih veličina u SCSS-u izgleda ovako:

```
.parent {
      background-color: red;

      .child-one {
            background-color: $greenColor;

            .child-two {
                  width: calc(100% - 50px);
            }
      }
}
```

2.2. JavaScript

Izumio ga je Brendan Eich 1995. godine, prvotno je pisan za Netscape 2, ali postao je ECMA-262 standardom 1997. (ECMA je organizacija koja stvara standarde za tehnologije) te je nastavljen razvoj za Firefox preglednik. 1999. objavljena je nova verzija ECMAScript-a – ES3 s kojom dolazi podrška za regularne izraze (engl. Regex – Regular expression) te „try“ i „catch“ ključne riječi pomoću kojih možemo manipulirati podatcima i greškama dok radimo asinkrone operacije. ES4 je preskočen pa je iduća značajna verzija bila ES5 jer je s njom 2009. godine uvedena podrška za JSON (JavaScript object notation) koji je i danas standardni format za prijenos strukture podataka u Web developmentu. Uz to, dodani su neki operatori za manipulaciju nizovima. 2015. godine izlazi ES6 koja uvodi „let“ i „const“ ključne riječi za deklariranje varijabli koje uvelike pomažu kod smanjivanja greški u pisanju koda. Od tada izlaze godišnja izdanja sa novom podrškom za manipulaciju nizovima i objektima i slično.

JavaScript se često miješa sa Java programskim jezikom, ali zapravo nemaju veze jedan s drugim. „*Java is to JavaScript as ham is to hamster*“ [2]. Korištenje korijena naziva – „Java“, je marketinški potez kako bi se JavaScript približio programerima zbog tada popularnijeg Java programskog jezika. Prvotni naziv bio je „Mocha“. U to vrijeme su se manji programi popularno nazivali „script“ te su baš ti mali programi bili prva stvar koja će se implementirati u Web i stoga današnji naziv – JavaScript.

JavaScript nam omogućuje interakciju sa korisničkim sučeljem. Umjesto da samo pregledavamo sadržaj, sada možemo i utjecati na neke elemente. Najprimitivniji primjer implementacije događaja u JavaScript-u bi bio slušanje klika miša na gumb i ispisivanje poruke korisniku. JavaScript je sam po sebi jako izložen greškama u pisanju, ali je jako modularan, odnosno pogodan prilagođavanju. Jedna operacija se može napisati na mnogo različitih načina, i različiti ljudi imaju različite preference kako nešto napisati, ali postoje najbolje prakse (engl. „Best practices“) i uzorci dizajna (engl. „Design patterns“) kojih se pametno držati radi univerzalno čitkog koda i snalaženja u projektu. U programiranju se često spominje „DRY“ akronim koji stoji za „Don't repeat yourself“ [3], a postoje još neki uzorci dizajna koji su zapravo neobavezan, ali poželjan standard za pregledan kod. Spajanjem uzoraka dizajna i kreiranje pomoćnih funkcija koje nam olakšavaju neke operacije koje se ponavljaju kroz aplikaciju stvaramo razvojni okvir (engl. „Framework“) ili programsku knjižnicu (engl. „Library“)

3. Povijest Web dizajna

3.1. Općenito

U početku, ranih 1990-ih web stranice bile su statički tekstualni dokumenti, a kasnije je omogućeno dodavanje grafika, slika, zvuka i videa. 1993. izašla je prva inačica HTML-a, ali u to vrijeme Web stranice nisu još bile toliko popularne. Prvi službeni standard HTML-a bio je HTML 2.0 koji je službeno objavljen u studenom 1995. godine. Stranice dobivaju dinamičke značajke uvođenjem JavaScript-a 1995., te je omogućena interakcija korisnika sa sadržajem. Godinu kasnije pojavio se „Flash“ koji omogućuje animacije te obilježava početak interaktivnih igara u preglednicima. Tim Berners-Lee izumio je WWW, odnosno World wide web 1989. godine te je zamišljen kao platforma za dijeljenje informacija između znanstvenika, fakulteta i instituta u svijetu.

3.2. Prva Web stranica

Prva web stranica bila je dostupna 6. Kolovoza 1991. godine. Izradio ju je isti Tim Berners i pokretala se na računalu u Europskoj organizaciji za Nuklearno istraživanje – CERN, a izradio je i prvi preglednik koji se zvao WorldWideWeb 1990. godine. To je bio prvi preglednik koji se mogao instalirati na različite vrste računala. (Cross-platform) Ta stranica bila je isključivo tekstualni prikaz sa linkovima bez ikakvih alata za dizajn. Prvi dostupni preglednik bio je Line-mode preglednik kojeg je Ovo je snimka zaslona rekreirane Web stranice iz 1991. godine.

Slika 1: Prva web stranica (Izvor: https://info.cern.ch/)

Ovdje se može vidjeti kako ta stranica izgleda u modernom pregledniku. https://info.cern.ch/hypertext/WWW/TheProject.html.

Sve web stranice iz ranog doba bile su uglavnom tekstualne zbog male brzine interneta i ograničene tehnologije. Pojavljuju se HTML elementi za zaglavlje, odlomak i linkove (<header>, <p>, <a>).

3.3. Pojava strukturiranih sučelja

U ožujku 1995. godine izašla je treća inačica HTML-a – HTML 3.0. HTML 3.0 podržavao je tablice, tekst koji se prilagođava i raspoređuje oko elementa i matematičke funkcije. 1996. godine dogodila se značajna prekretnica – Space Jam. Film koji se reklamirao na Web stranici koja je u to vrijeme bila vrhunac Web dizajna i iskoristila puni potencijal onda dostupne tehnologije. Pojava tablica u HTML-u za strukturirani raspored elemenata na ekranu i korištenje JavaScript-a za omogućivanje interakcije korisnika sa stranicom je ono što je zaintrigiralo posjetitelje ove stranice. Ta ista stranica još uvijek je dostupna na adresi https://www.spacejam.com/1996/ i nije ažurirana od devedesetih godina.

Slika 2: Space Jam Web stranica (Izvor: https://www.spacejam.com/1996/)

Klikom na planete korisnik je prebačen na druge dijelove stranice gdje je moguće pronaći igre, bojanke, postere i slično.

3.4. Fiksna sučelja

Oko 2000. godine CSS je postao popularan jer se više nije koristio samo za fontove i boje. Većina Web stranica tog doba bile su šarene i fiksne širine. Nije bilo puno različitih veličina monitora. Sav sadržaj bio je unutar oko 980 piksela jer je na taj način stranica izgledala jednako na svim ekranima. Fiksni pristup izvodio se pomoću jednog elementa obgrljena (wrapper) kojemu se definira fiksna širina i pozicionira ga se na sredinu ekrana, a svi elementi unutar njega bi se rasporedili u pikselima ili postotcima.

Slika 3: eBay 1999. (Izvor: https://www.webdesignmuseum.org/)

Kako je tehnologija s vremenom napredovala i ekrani su postajali širi i viši sučelja sa fiksnom širinom nisu bila atraktivna jer je bilo puno neiskorištenog prostora i pojavila se potreba za proširenjem sučelja van tadašnjih ograničenja. Problem je bio u tome što se sučelje nije moglo samo raširiti jer onda ne bi bilo pregledno na manjim ekranima pa su developeri bili primorani pronaći rješenje. U početku se radilo više stranica za više veličina ekrana, ali to je jako

nezahvalno za održavanje i cijena je dva puta veća jer je potrebno dizajnirati i izraditi dvije umjesto jedne stranice, odnosno sučelja. S obzirom da se to pokazalo kao neekonomično rješenje tu se pojavljuje „Fluid design / Liquid layout“. Tim pristupom nije bilo „wrapper“ elementa fiksne širine već su se elementi raširili po cijelom ekranu koristeći veličine u postotcima. Ovo rješenje je bilo relevantno sve do dolaska mobitela sa preglednicima, konkretnije Apple je 2007. godine sa svojim iPhone-om prvi omogućio prvi iskoristivi preglednik na mobitelu. Fluid design je rješavao problem prelaska s manjeg ekrana na veći, ali ekrani na mobitelima bili su premali da bi sav sadržaj bio pregledan. Mobiteli su postali sve zastupljeniji u pregledavanju Web-a, ali navigacija po tadašnjim Web stranicama nije bilo lijepo iskustvo. Od lipnja 2010. godine do lipnja 2011. godine korištenje preglednika na mobitelima je poraslo sa 2.86% na 7.06%, a u 2019. godini ta brojka je skočila na 51.11% dok je korištenje preglednika na stolnim računalima palo na 45.18%, a na tablet računalima 3.71%.

3.5. Prilagodljiva sučelja

Iako je potrebno više vremena za napraviti prilagodljivo sučelje i dalje je isplativije nego fiksno jer je potrebno je održavati samo jedan projekt, dizajnirati jedno sučelje,a stranica se poslužuje na samo jednoj adresi (engl. url) što umanjuje dodatne troškove posluživanja. Korisnik ne mora otvarati različite inačice stranice za mobitel i za računalo. Danas je neizbježno imati prilagodljivo sučelje jer korisnici žele imati mogućnost pregledati sadržaj na mobitelu, na tablet računalu i na stolnom ili prijenosnom računalu gdje svaki od navedenih uređaja ima drugu veličinu ekrana i drugu razlučivost.

2010. godine Web dizajner Ethan Marcotte uveo je koncept prilagodljivog Web designa nakon što je ukazao na probleme dotadašnjih pristupa: „*But no design, fixed or fluid, scales seamlessly beyond the context for which it was originally intended. The example design scales perfectly well as the browser window resizes, but stress points quickly appear at lower resolutions. When viewed at viewport smaller than 800×600, the illustration behind the logo quickly becomes cropped, navigation text can wrap in an unseemly manner, and the images along the bottom become too compact to appear legible. And it’s not just the lower end of the resolution spectrum that’s affected: when viewing the design on a widescreen display, the images quickly grow to unwieldy sizes, crowding out the surrounding context.“ [5]*

Još od CSS inačice 2.1 mogli smo definirati radi li se o prikazu za ekran ili prikazu za ispis.

```
<link rel="stylesheet" type="text/css" href="core.css"
  media="screen" />

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

U to vrijeme dosta medijskih tipova (engl. „Media query“) nije bilo podržano u većini preglednika tako da uglavnom nisu bili iskoristivi. CSS3 bio je prekretnica za prilagodljivost sučelja. Dostupni su nam novi medijalni upiti kojima možemo definirati za koju veličinu ekrana se određeni stilovi učitavaju.

Upit se sastoji od dva dijela:

- Medijski tip (u ovom slučaju „screen“, odnosno ekran)
- Stvarni upit u zagradi („max-device-width“) gdje se proslijedi vrijednost za veličinu (U ovom slučaju 480 piksela)

```
<link
      rel="stylesheet"
      type="text/css"
      media="screen and (max-device-width: 480px)"
      href="stilove-za-mobitele.css"
/>
```

Ovaj upit provjerava je li horizontalna razlučivost (širina) ekrana uređaja jednaka ili manja od 480 piksela i ako je učitava CSS dokument sa stilovima „stilovi-za-mobitele.css“. Moguće je provjeravata više različitih vrijednosti odjednom pomoću operatora „and“

```
<link rel="stylesheet" type="text/css"
      media="screen and (max-device-width: 480px) and (resolution: 163dpi)"
href="stilovi-za-mobitele.css" />
```

Ovo nije jedini način za uvođenje medijalnih upita u kod, naprotiv, ovo je najnepregledniji način i gotovo se ne koristi. Puno prihvaćeniji način je pisanje upita u CSS dokument.

```
      @media screen and (max-device-width: 480px) {
        .column {
          display: none;
        }
      }

@import url("stilove-za-mobitele.css") screen and (max-device-width:
480px);
```

U oba slučaja rezultat je isti, ako je upit istinit primjenit će se stilovi na HTML elemente koji su označeni u CSS dokumentu.

4. Preglednici

Prije nego krenemo u izradu praktičnog dijela treba dublje upoznati CSS3 i na koji način ga preglednici interpretiraju. Najpoznatiji i najkorišteniji preglednici su Google Chrome, Mozilla Firefox, Safari, Edge (bivši Internet explorer), Opera, ali ima ih još mnogo. Chrome, Mozilla i Internet Explorer interpretiraju CSS svojstva na 3 različita načina i radi toga njihov prikaz nije konzistentan što izaziva probleme kod prezentacije sučelja. Da bi se ovaj problem spriječio svaki preglednik je dodao prefix na eksperimentalna CSS svojstva.

```
.foi-btn {
 -moz-experimental-property: value; /* Firefox */
 -ms-experimental-property: value; /* Internet Explorer */
 -webkit-experimental-property: value; /* Chrome/Safari */
}
```

S vremenom implementacije preglednika su se promjenile, ali prefiksi su ostali i ta svojstva nisu ažurirana, radi čega proizvođači preglednika i dalje moraju podržavati ekperimentalna svojstva da se stranice koje su ih implementirale ne raspadnu. Google Chrome i Mozilla se odmiču od korištenja eksperimentalnih svojstava te ne dopuštaju korištenje svojstava koja nisu provjerena, stabilna i dostupna svima.

Čak i danas ima svojstava koja rade u jednom pregledniku, a u drugom ne, ali znatno manje nego prije. Google Chrome je najčešći odabir za preglednik u kojem će se stranica testirati i pregledavati radi toga što podržava najviše eksperimentalnih svojstava. Safari preglednik se koristi isključivo na Apple uređajima, i često je dolazilo do problema sa stranicama koje se bez problema prikazuju na Chrome-u ili Mozilla Firefox-u.

Kao developeri želimo da korisnici mogu pregledavati naš sadržaj u svakom pregledniku, a to možemo napraviti ako pratimo par jednostavnih koraka:

1. Korištenje Web standada

Web standardi su smjernice i specifikacije koje definiraju kako određene Web tehnologije rade. Smjernice zadaju organizacije poput W3C (World Wide Web Consortium) i IEFT (Internet Engineering Task Force). Ako pratimo te standarde naše stranice bit će pouzdanije, pristupačnije i konzistentnije u različitim preglednicima.

To znači da trebamo koristiti najnoviji (stabilni) HTML, CSS i JavaScript kod koji je prilagođen najnovijim standardima. Jako je važno držati stranicu/aplikaciju ažuriranom jer se u protivnom stvara „tehnološki dug“ koji je puno skuplji u smislu da treba puno manje vremena i radne

snage da se stranica ažurira sa standarda od prije pola godine nego da se ažurira nakon skoka u standardu od dvije godine. Postoje servisi kao što su „W3C Markup Validation Service“ i „W3C CSS Validation Service“ koji javljaju greške i upozorenja za naš kod.

2. Testiranje stranice na različitim preglednicima

Iako se stanje danas znatno popravilo, ne možemo pretpostaviti da će naš kod jednako raditi, odnosno imati jednak prikaz na svim preglednicima i uređajima. Prilagodljivost sučelja je samo jedno pitanje, uz to treba imati na umu da različiti preglednici imaju različite probleme koji se mogu pojaviti i bez testiranja na različitim preglednicima to ne možemo znati niti spriječiti. Postoje alati koji simuliraju različite preglednike (BrowserStack, CrossBrowserTesting) pa developeri nisu prisiljeni instalirati tri ili četiri različita preglednika, ali u današnje vrijeme memorijski prostor i performanse računala više nisu upitne pa je preporučeno imati njihove instalacije. Svaki preglednik ima svoje programerske alate (engl. „Developer tools“). preko kojih je omogućeno jednostavno traženje grešaka, mijenjanje veličine ekrana i provjera prilagođavanja te ponašanja sučelja.

3. Progresivno poboljšanje i dostojanstveni pad (engl. „Progressive enhancement and graceful degradation“)

Ovo su dvije strategije koje se međusobno nadopunjuju i omogućavaju da naša stranica jednako dobro radi na različitim uređajima i preglednicima.

„Progresivno poboljšanje“ je strategija u kojoj se prvo izradi osnovna struktura stranice koja provjereno radi na svim uređajima i preglednicima i onda se postepeno dodaju značajke, svojstva i poboljšanja za preglednike koji ih podržavaju. „Dostojanstveni pad“ je apsolutna suprotnost prvoj strategiji jer je ovdje ideja da stranica u startu ima sve značajke i svojstva koja rade na modernim preglednicima i uređajima, a naknadno se traže rješenja i optimizacije za preglednike i uređaje koji ih ne podržavaju. Korištenjem detekcije značajki (engl. „Feature detection“) možemo provjeriti je li značajka/svojstvo podržana u određenom pregledniku i naći neko obilazno rješenje u slučaju da ista nije podržana.

4. Korištenje prilagodljivog Web dizajna

Iako je prilagodljivost dizajna glavna tema ovog rada, to je samo jedan dio koji je važan za globalnu prilagodljivost stranice.

5. Korištenje Web fontova i ikona

Ako koristimo Web fontove i ikone smanjujemo mogućnost grešaka u prikazu jer Web fontovi i ikone provjereno rade i prikazuju se u različitim preglednicima i uređajima.

5. Alati

Sada kad smo upoznati sa svom problematikom u izradi modernih i prilagodljivih Web stranica, odnosno Web korisničkog sučelja treba pronaći prave alate koji olakšavaju njihovu izradu.

Za različite potrebe koriste se različiti pristupi. Postoje gotova i polu-gotova rješenja koja su u startu prilagođena i rade na različitim ekranima i uređajima. Neki od najpopularnijih alata za brzu izradu Web stranica su Wordpress i Wix. Pomoću njih možemo jednostavno i brzo imati gotovu stranicu. Dovoljno je izabrati temu koja nam odgovara izgledom i sadržava značajke koje zadovoljavaju naše potrebe. Teme čak imaju i primjerni sadržaj koji se može samo zamijeniti sa našim stvarnim sadržajem i testirane su na različitim preglednicima. Wordpress je čest izbor za izradu jednostavnih Web stranica bez nekih kompleksnih značajki, ali isto tako često klijent ima zahtjev koji Wordpress tema ne podržava i potrebno je izraditi nešto prilagođeno za potrebe klijenta.

Polu-gotova rješenja poput „Templated“ platforme na kojoj se nalazi stotine prilagodljivih predložaka HTML kostura koji su besplatni za preuzimanje. Ovo je odlična solucija za one bez ideje jer tamo mogu pronaći dizajn koji im odgovara, preuzeti taj kod i uređivati ga kako žele. Predlošci su prilagodljivi za različite veličine ekrana u startu tako da je to još jedna briga manje za developera.

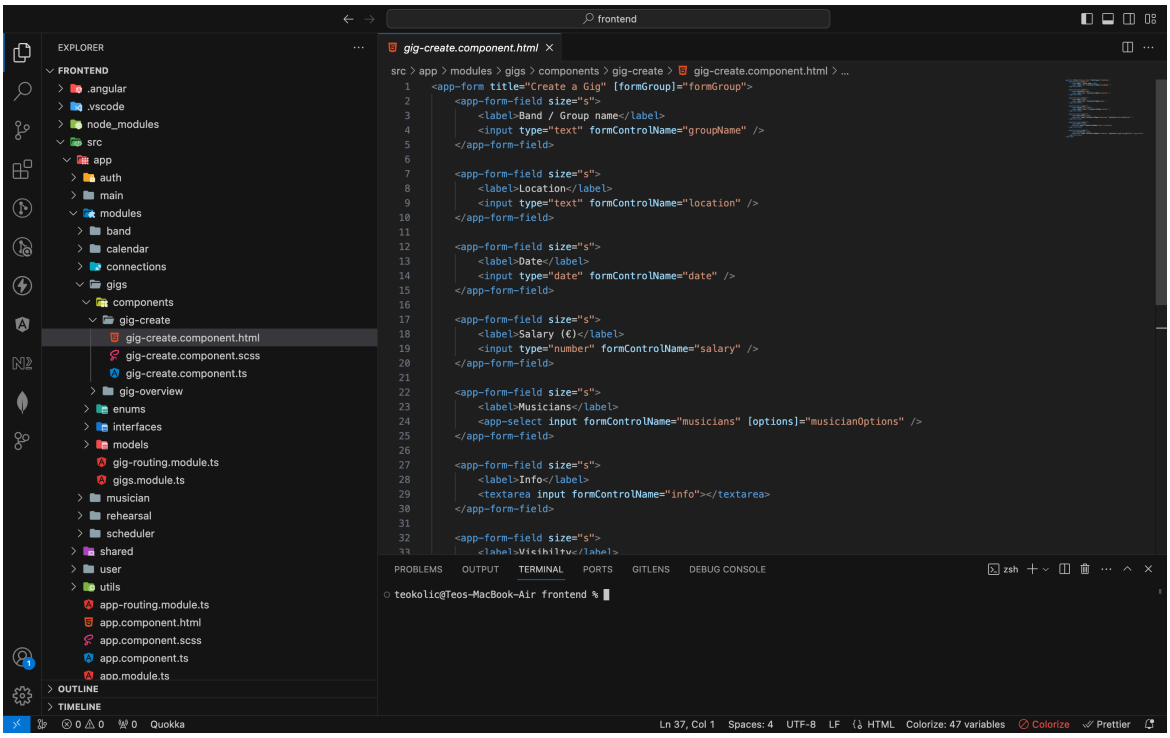
Za one koji vole imati apsolutnu kontrolu i vole sve pisati sami tu su knjižnice ili razvojni okviri pomoću kojih pišemo kod iz nule, ali nam omogućavaju korištenje nekih predefiniranih funkcionalnosti za brži razvoj. Prva i osnovna razlika između knjižnice i okvira je da knjižnicu možemo koristiti na zahtjev (engl. „On demand“), odnosno kada nam zatreba. Knjižnicu je moguće uvesti preko HTML tag-a <head> ili pomoću „import“ ili „require“ ključne riječi. Developer bira gdje i kada će koristiti knjižnicu dok je razvojni okvir je kompletan skup alata koji služi za oblikovanje i organiziranje Web stranice/aplikacije gdje mi možemo dodavati naš kod u već posloženu strukturu te on poziva naš kod kako je predviđeno unutar okvira. Danas postoji mnogo knjižnica i okvira stoga je u početku problem izabrati „pravi“, ali vodeći se popularnošću istih u svijetu, od knjižnica za Web korisnička sučelja se najviše koristi „Boostrap“, dok je u zadnje vrijeme popularan razvojni okvir „Tailwind CSS“

Uz alate za izradu stranice treba se upoznati i s alatima za testiranje i pronalaženje grešaka.

„Responsinator“ je besplatan i jednostavan alat za provjeru prilagodljivosti stranice na nekoliko različitih uređaja i orijentacija. Dovoljno je upisati adresu naše stranice i možemo vidjeti kako se stranica ponaša u različitim uvjetima. „Resizer“ je još jedan sličan alat koji nam odjednom prikazuje izgled naše stranice na više različitih veličina ekrana i svakom od njih možemo dodatno mijenjati veličinu. Uz alate za provjeru prilagodljivost korisno je poslužiti se i alatima za provjeru ponašanja stranice na različitim preglednicima. To su alati koji su spomenuti ranije – CrossBrowserTesting i BrowserStack.

6. Okolina za razvoj

Prvo što je potrebno za izradu Web stranice je tekstualni editor za pisanje koda. Tekstualni editor (IDE – Integrated development environment) je stvar izbora. Ima ih mnogo, a neki od najpopularnijih za Web development su WebStorm, Visual Studio Code, Brackets, Sublime text… Svaki od njih je dovoljan za pisanje koda, ali razlikuju se u nadogradnjama, izgledu, dodatnim mogućnostima i osobnom ukusu, odnosno navici. Za izradu ove stranice koristit će se Visual Studio Code zato što je mali u veličini, jednostavan za navigaciju i ima pregledno sučelje.

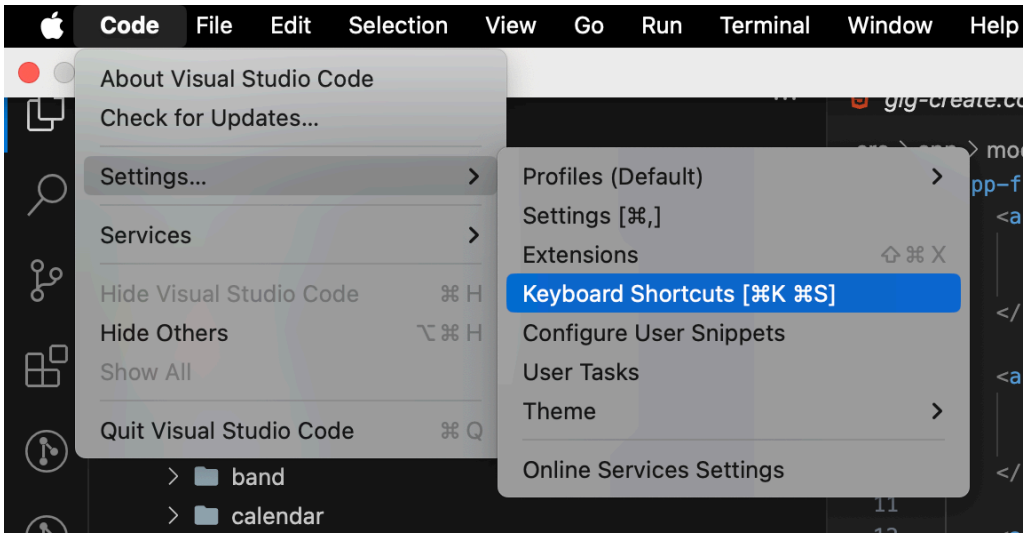


Slika 4: Visual Studio Code

S lijeve strane nalazi se alatna traka i stablasti prikaz arhitekture mapa i datoteka našeg projekta. U alatnoj traci nalaze se alati za verzioniranje, pretraživanje i dodatne nadogradnje koje smo instalirali. Glavni dio ekrana je otvorena datoteka koju uređujemo, a moguće je imati i više datoteka otvoreno odjednom što omogućuje jednostavnu usporedbu i/ili pretraživanje nekih naziva klasa ili funkcija. Dvojni prikaz možemo postići tako da prevučemo dokument iz stablastog prikaza na desnu stranu editora. Dolje se nalazi terminal u koji pišemo naredbe i u kojem dobivamo povratnu informaciju od našeg editora vezanu za ispravnost koda. Ukoliko

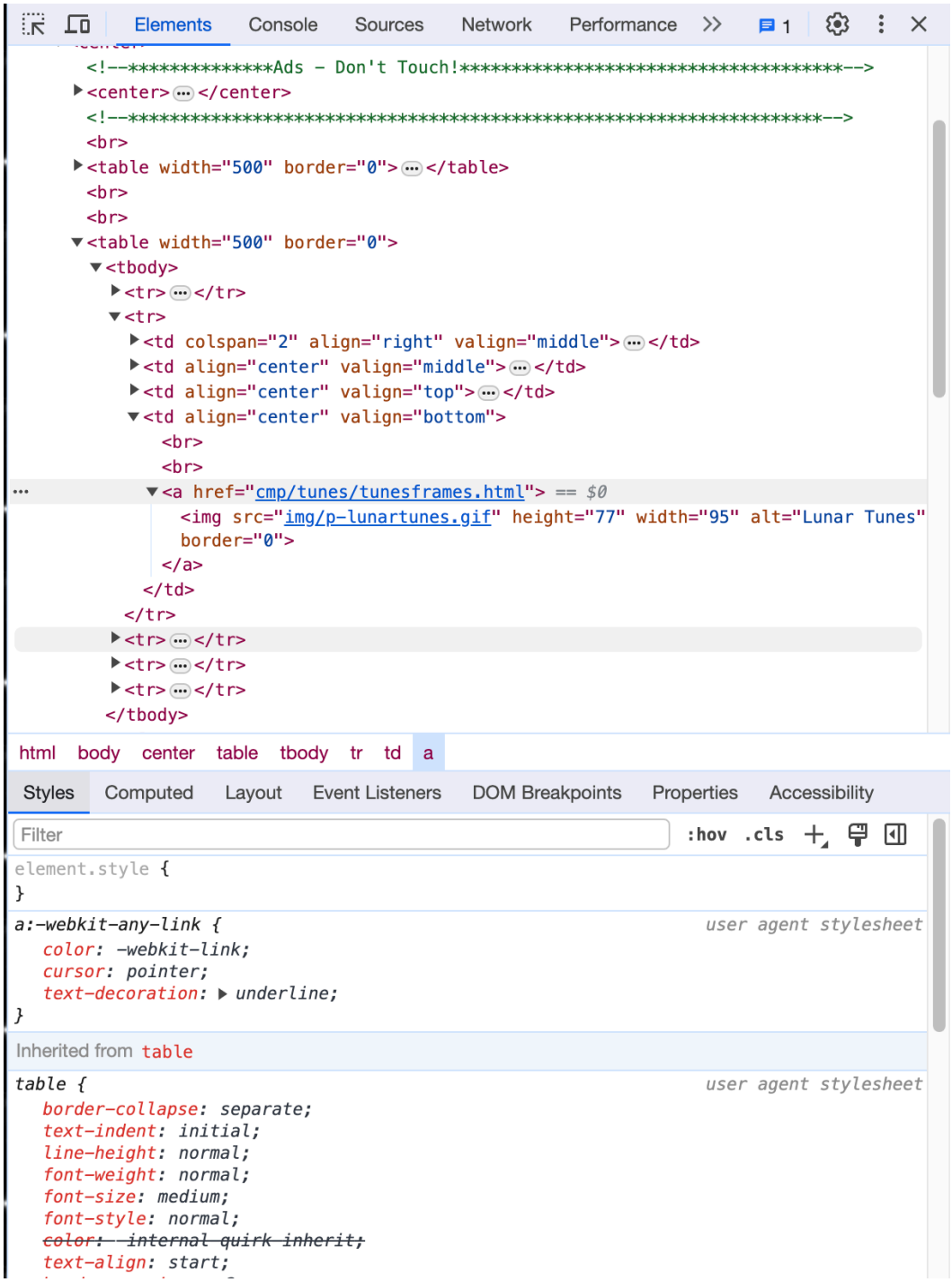
trebamo više terminala postoji mogućnost i za to na njegovoj alatnoj traci. U slučaju potrebe, terminal se može raširit preko cijelog ekrana, a to vrijedi za sve prozore unutar editora. Skroz desno nalazi se umanjeni prikaz cijele otvorene datoteke. Taj prikaz je jako koristan za nalaženje grešaka u kodu kada datoteke imaju po nekoliko stotina linija jer odmah vidimo gdje se kod crveni i klikom na taj dio umanjenog prikaza vodi nas na taj dio u kodu. Na vrhu nalazi se globalni pretraživač, a na dnu još jedna alatna traka koja ima informacije vezane za formatiranje koda i trenutnu verziju projekta. Još jedno odlično svojstvo ovog IDE-a je što je potpuno prilagodljiv korisniku pa korisnik ima mogućnost pomaknuti terminal“ alatnu traku i umanjeni prikaz bilo gdje na ekranu. Desno od globalnog pretraživača nalaze se tri gumba za otvaranje i zatvaranje prozora sa svake strane te četvrti gumb koji nam nudi prečac za preslagivanje prozora na druga mjesta.

Visual Studio Code nudi mogućnost konfiguriranja prečaca i kombinacija tipki za različite akcije što uvelike poboljšava produktivnost i ubrzava rad. Te prečace i kombinacije tipki možemo postaviti klikom na „Code“ -> „Settings“ -> „Keyboard shortcuts“



Slika 5: Keyboard shortcuts

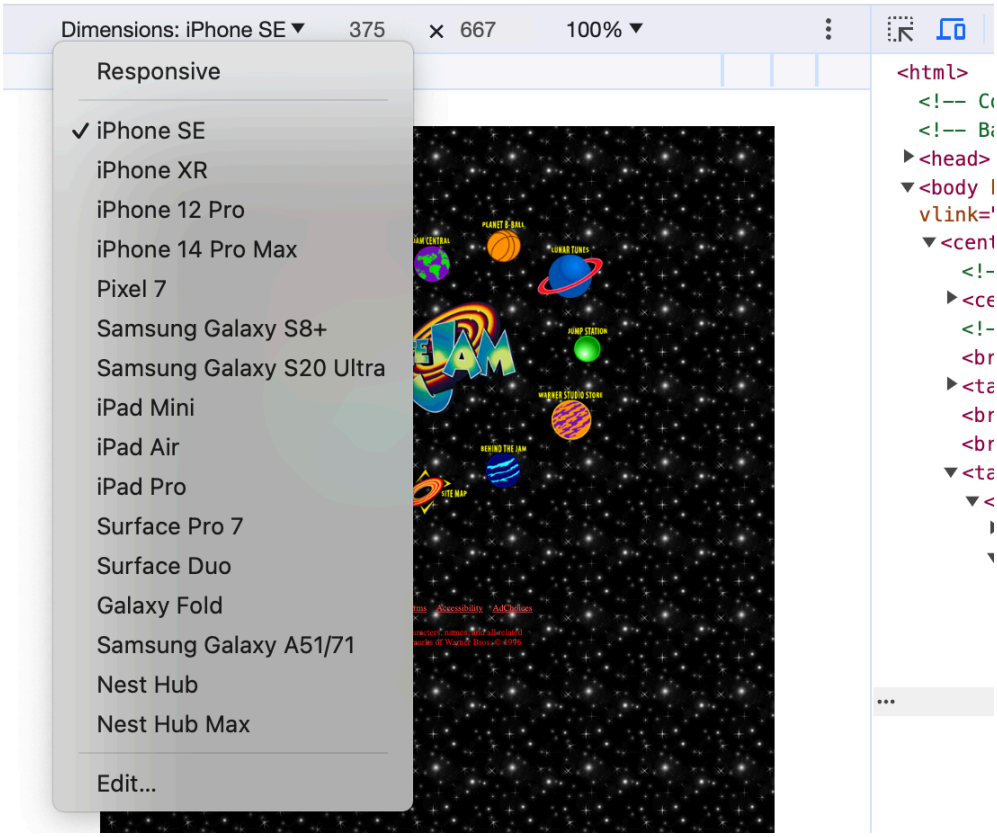
Uz tekstualni editor potreban je i preglednik. Već je spomenuto kako je korisno imati različite preglednike za testiranje, ali za vrijeme izrade stranice najčešće koristimo jedan pa kasnije provjeravamo kako se kod ponaša u drugim preglednicima. Ovu stranicu kroz izradu pregledavat ćemo u Google Chrome pregledniku radi njegovih developerskih alata koji su jako praktični. Svaki preglednik ima svoju inačicu tih alata, ali izabrali smo Chrome zbog jednostavnosti i preglednosti sučelja. Developerske alate možemo otvoriti desnim klikom bilo gdje na stranicu i klikom na „inspect“.



Slika 6: Developerski alati

Na vrhu nalazi se horizontalna alatna traka sa jako praktičnim mogućnostima. Prvi gumb služi za odabir elementa na stranici klikom miša. Kada odaberemo element u najvećem prozoru ispod alatne trake dobijemo prikaz arhitekture HTML kostura sa označenim elementom koji smo odabrali i automatski se odabire „element“ u alatnoj traci. Na taj način vrlo jednostavno možemo vidjeti gdje se traženi element unutar strukture naše stranice. Ispod prozora sa strukturom nalazi se još jedna alatna traka sa raznim značajkama, a ispod nje

nalaze se CSS stilovi koji su aplicirani na odabrani element. Vrijednosti tih stilova možemo vrlo jednostavno promijeniti u tom prozoru i vidjeti promjenu na stranici u stvarnom vremenu bez utjecaja na naš kod. Promjene napravljene na ovaj način ne ostaju spremljene, već se brišu nakon osvježavanja stranice stoga je puno praktičnije raditi provjere iz developerskih alata. Desno od gumba za odabir elementa nalazi se gumb koji otvara integrirani simulator različitih veličina ekrana. Odličan alat za provjeru prilagodljivosti ekrana. Budući da tehnologija brzo napreduje i konstantno se proizvode novi uređaji ovdje se uvijek nalaze najpopularniji uređaji tako da developer uvijek ima brz pristup potrebnim različivostima za testiranje, a uz to ima i mogućnost odabira bilo koje razlučivosti.



Slika 7: Prikaz popisa uređaja za provjeru prilagodljivosti

Ukoliko trebamo testirati stranicu na nekoj neobičnoj razlučivosti možemo ju spremiti kao predložak koji će biti spremljen u ovaj padajući izbornik. Desno od „elements“ kartice nalazi se kartica konzole (engl. „console“) koja otvara prozor u kojem nam preglednik javlja greške u našem JavaScript kodu te greške kod učitavanja dokumenata. Nakon konzole nalazi se gumb izvora (engl. „sources). Klikom na „sources“ imamo uvid u izvorni kod stranice kao što bi bio

prikazan u našem editoru. Mrežna kartica (engl. „Network“) služi za pregled upita koje naša stranica radi prema serveru. Još jedna bitna kartica je aplikacijska kartica (engl. „Application“). Klikom na tu karticu imamo pristup lokalnoj i sesijskoj pohrani podataka. Lokalna pohrana koristi se kao jedna vrsta upravljanja stanjem stranice ili aplikacije jer se podaci iz nje ne brišu nakon osvježavanja stranice. Tu možemo spremiti neke postavke ili podatke koje želimo zadržati.

Ako smo odlučili raditi stranicu bez dodatnih knjižnica ili razvojnih okvira u tom slučaju ne trebamo ništa drugo osim IDE-a i preglednika. Naša stranica biti će izrađena u Angular razvojnom okviru koji nije neizbježan za ovaj rad, ali je zbog svoje logične i pregledne arhitekture mapa i datoteka dobar primjer za organizirati i prezentirati Web stranicu. Uz to, omogućuje nam korištenje gotovih komponenti iz PrimeNG seta komponenti.

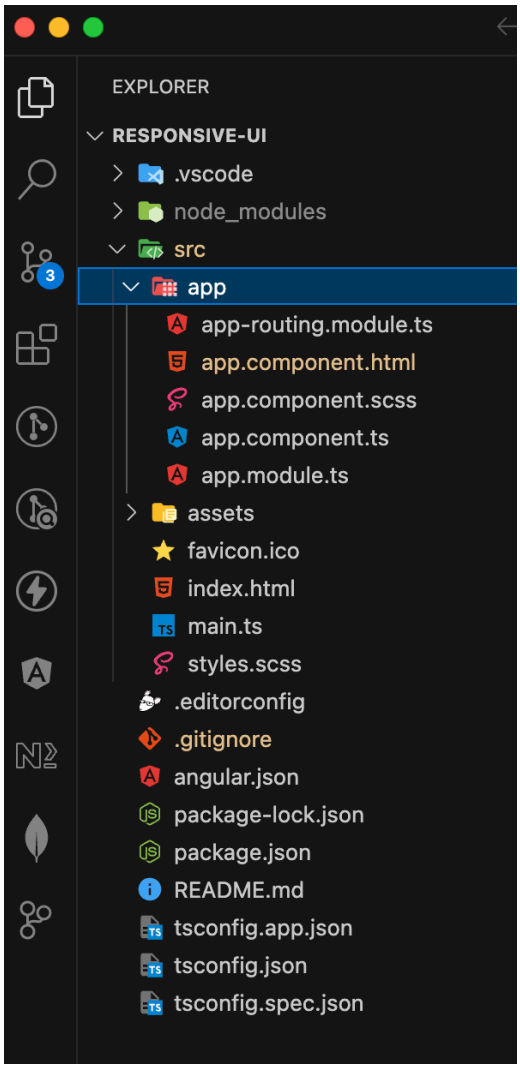
Za korištenje Angulara potrebno je instalirati Node.js i NPM (Node package manager). Node.js je otvorena JavaScript okolina za izvršavanje serverskih aplikacija. Pomoću Node.js-a omogućujemo računalu da izvršava JavaScript kod izvan preglednika. NPM je kao što samo ime kaže menadžer za Node, odnosno JavaScript pakete. Preko NPM-a možemo jednostavno instalirati ili brisati JavaScript knjižnice i druge alate potrebne za razvoj aplikacija. Angular dosta ovisi o vanjskim paketima stoga je NPM jako bitan faktor u razvoju Angular aplikacje.

Angular je razvojni okvir za jednostranične aplikacije baziran ja JavaScript jeziku, odnosno na TypeScript jeziku koji je proširenje JavaScript-a u vidu da nam nudi mogućnost deklariranja tipova podataka. U ovom radu ćemo koristiti neke Angular značajke, ali daleko od punog potencijala ovog razvojnog okvira stoga ga nećemo opisivati i učiti u dubinu za potrebe ove stranice.

Node.js možemo preuzeti na službenoj stranici: https://nodejs.org/en/download/current te uz njega dolazi i NPM pa ga nije potrebno dodatno instalirati.
Nadalje, koristeći NPM možemo instalirati Angular CLI (Command Line Interface) upisivanjem naredbe: *npm install -g @angular/cli* u terminal.
„*-g“* znači „globalno“, odnosno dostupno iz bilo koje lokacije na računalu. Angular CLI nam omogućuje jednostavno i brzo generiranje projekta, komponenti i ostalog preko terminala koristeći kratke naredbe tipa *„ng generate component menu“ ili „ng g c menu“.* Obje naredbe su identične te generiraju mapu sa tri datoteke – menu.component.ts, menu.component.html i menu.component.css (ili scss ako smo tako odabrali pri generiranju projekta).

Prvo što trebamo napraviti je generirati novi projekt, a to ćemo napraviti preko terminala. Pozicioniramo se u mapu u kojoj želimo imati projekt i napišemo:

„ng new <ime-projekta>“, odnosno u našem slučaju *„ng new responsive-ui“.* Angular CLI će nam izgenerirati osnovnu strukturu projekta i sve datoteke potrebne za pokretanje istog.



Slika 8: Struktura Angular projekta

Krenuvši s kraja, vidimo konfiguracijske datoteke koje nam za potrebe ovog rada nisu bitne pa ih nećemo opisivati. Vrlo bitna datoteka je „package.json“. Tu se nalaze svi paketi i njihove verzije koje koristimo u našem projektu. Uz pakete, tu definiramo i razne skripte za pokretanje aplikacije i slično. „styles.css“ i „index.html“ su poznati svima koji su se već susreli sa izradom Web stranica. Bitno je napomenuti da se u Angular-u CSS piše za svaku komponentu posebno dok je „styles.css“ globalna datoteka koja se aplicira na sve komponente. Unutar Index.html datoteke nalazi se*„<app-root></app-root>“,* na taj način se naša osnovna komponenta („app.component“), a preko nje i cijela aplikacija monitra u HTML dokument. U „assets“ mapu spremamo slike, ikone i slične datoteke. „app-routing.module.ts“ je datoteka u kojoj definiramo koje komponente se prikazuju na kojoj adresi i tako nam omogućava SPA (engl. „Single page application“) dizajn što znači da stalni sadržaj učitavamo samo jednom, a dinamični učitavamo po potrebi.

7. Praktični rad

7.1 Generiranje prvih komponenti

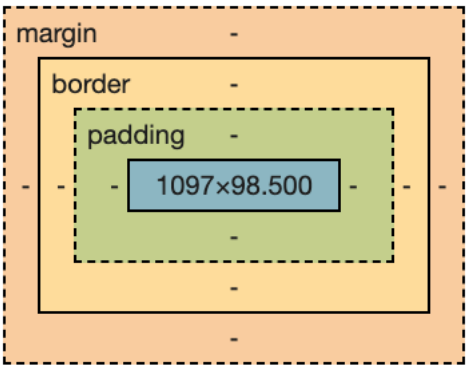
Za početak trebamo smisliti dizajn stranice i složiti osnovnu strukturu. Gotovo svaka stranica ima izbornik, zaglavlje i prostor gdje se prikazuje sadržaj. Kreiramo dvije komponente pomoću Angular CLI-a naredbama:

```
ng g c menu
ng g c header
```

Generirana Angular komponenta (.ts) izgleda ovako

```
import { Component } from '@angular/core';

@Component({
  selector: 'app-menu',
  templateUrl: './menu.component.html',
  styleUrls: ['./menu.component.scss']
})
export class MenuComponent {
}
```

„selector“ je oznaka kojom ubacujemo komponentu u drugu komponentu stoga u „app.component.html“ dodajemo „<app-menu></app-menu>“. i u pregledniku će se pojaviti osnovni tekst „menu works!“. Na isti način prikazat ćemo i zaglavlje. Sada možemo krenuti sa pisanjem kostura komponenti i dodavanjem stilova. Za početak ćemo napraviti zaglavlje koje će biti na vrhu stranice, postavit ćemo padding na 0.5rem 1rem, širina će mu biti 100% - 2rem (kompenziramo za 1rem sa svake strane) da se prilagođava širini ekrana, a visinu 4rem. „rem“ i „em“ su mjerne jedinice za veličinu koja se skalira u odnosu na baznu veličinu fonta koja je inicijalno 16 piksela. Ako za sve veličine na stranici koristimo „rem“ onda jednostavno možemo skalirati cijelu stranicu promjenom samo jedne veličine – body { font-size: 16px }. Za visinu zaglavlja postavljamo 3rem i dodajemo sjenu da se jasno vidi gdje zaglavlje prestaje. „Padding“ je prostor između sadržaja i ruba elementa u kojemu se sadržaj nalazi. U početku je lako zamjeniti „padding“ i margine, ali zato u developerskim alatima imamo legendu pomoću koje možemo provjeriti gdje se što nalazi u „CSS Box“ modelu.



Slika 9: CSS Box model

Obično se u zaglavlju nalazi logo i gumb za prijavu/odjavu. Za raspored elemenata koristit ćemo „Flexbox“. Vrlo praktičan CSS alat za prilagodljivo pozicioniranje elemenata. Prije flexbox-a trebalo je puno koda za centriranje elementa u sredinu svog roditelja, a sada je to moguće u tri linije koda. Logo će biti na lijevoj strani, a gumb za login na desnoj. Obzirom na to da će logo biti slika trebamo ga spremiti unutar drugog elementa kojem ćemo dati fiksnu širinu i visinu, a „img“ elementu unutra ćemo dati širinu 100%, a visinu „auto“. Bez roditeljskog elementa za sliku logo bi bio pune veličine. Kreiramo roditeljski element u „header“ komponenti u koji ćemo staviti dva „div“ elementa – „logo-wrapper“ i „user“. Da bi rasporedili na lijevu i desnu stranu možemo jednostavno na roditeljski element postaviti „display:flex“ i „justify-content: space-between“. Opcija „space-between“ postavlja jednaki razmak između elemenata i zauzima cijelu širinu roditeljskog elementa, a s obzirom da imamo samo dva elementa unutar njega oni će se raširiti svaki na svoj kraj. Za sada komponenta zaglavlja izgleda ovako:

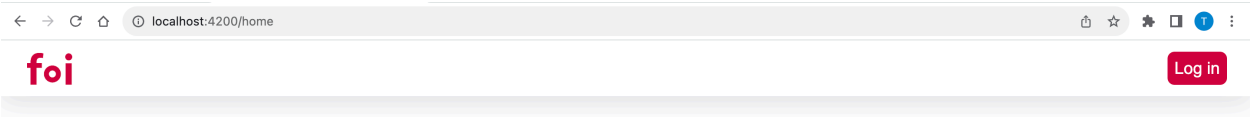
header.component.html

```
<div class="header">
  <div class="logo-wrapper">
    <img src="./../../assets/images/foi-logo.png" alt="Logo" />
  </div>
  <div class="user">
    <button>Log in</button>
  </div>
</div>
```

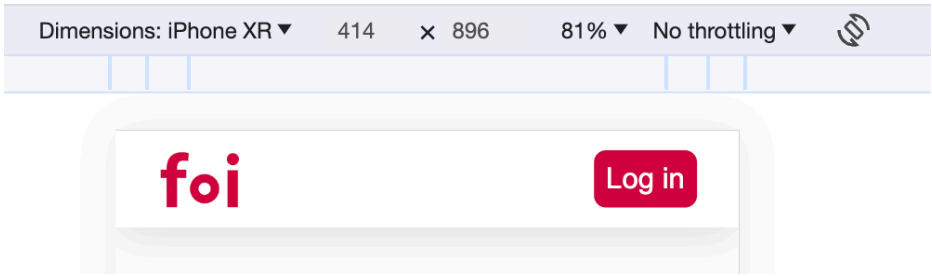
Kombinacijom Flexbox-a i širine u postotcima dobili smo zaglavlje sa prilagodljivim značajkama te nakon svega implementiranog CSS izgleda ovako:

```
.header {
  width: calc(100% - 2rem);
  height: 3rem;
  box-shadow: rgba(100, 100, 111, 0.2) 0px 7px 29px 0px;
  padding: 0.5rem 1rem;
```

```
  display: flex;
  justify-content: space-between;
  align-items: center;

  .logo-wrapper {
    width: 5rem;
    height: 3rem;

    img {
      width: 100%;
      height: auto;
    }
  }

  .user {
    button {
      font-size: 1.2rem;
      border: 0.1rem solid black;
      background-color: white;
      border-radius: 0.5rem;
      padding: 0.5rem;
    }
  }
}
```



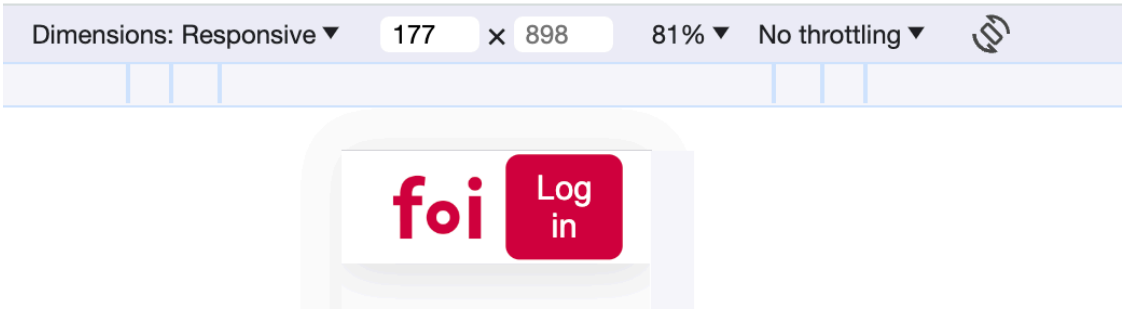
Slika 10: Zaglavlje na velikom ekranu



Slika 11: Zaglavlje na ekranu dimenzija uređaja iPhone XR

Zaglavlje za sada ima samo dva elementa te nema rizika za njihovo lomljenje, odnosno greške prilikom promjene na veličine mobilnih ekrana stoga nema potrebe za pisanje medijalnih upita za konkretne razlučivosti, ali svejedno ćemo pokriti neke rubne slučajeve. Iako više nema mobilnih uređaja s razlučivosti ispod 200 piksela, možemo provjeriti kako bi se stranica

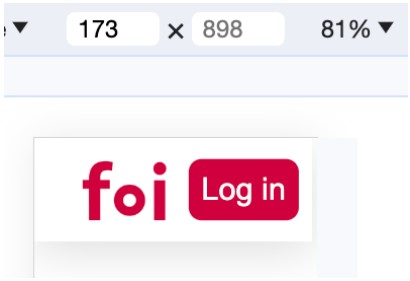
ponašala i u tim uvjetima. Kada se širina smanji na 182 piksela, logo i gumb se pomiču i mijenjaju.



Slika 12: Neželjene promjene u sučelju na niskim razlučivostima

Kako bismo riješili ovaj problem postavit ćemo fiksne minimalne širine za logo i za gumb tako da kad se širina ekrana smanji elementi neće gubiti svoju širinu. Ažurirani CSS izgleda ovako:

```
.header {
...
  .logo-wrapper {
    width: 5rem;
    height: 3rem;
    min-width: 5rem;
  }
  .user {
    min-width: 5rem;
...}
```



Slika 13: Elementi sa minimalnom širinom

Idući korak je kreirati izbornik, Većina modernih stranica ima izbornik sa lijeve strane te zauzima punu visinu ekrana do zaglavlja. Isto tako, na manjim razlučivostima, odnosno ekranima veličine mobitela taj izbornik se sakriva, a u zaglavlju se pojavljuje takozvani „Burger menu“. Tako se zove jer ikona izgledom podsjeća na burger i pritiskom na njega otvara se izbornik koji u tom slučaju zauzima punu širinu ekrana. Počet ćemo sa izradom izbornika. Postavit ćemo širinu 11rem, a za visinu ćemo koristiti dinamički pristup. Visina elemenata se ne ponaša pravilno ako je postavimo u postotcima. Ukoliko želimo zauzeti cijelu visinu ekrana moramo koristiti „vh“, odnosno „vertical height“. Ta mjerna jedinica odnosi se na cijelu visinu ekrana te ako stavimo visinu izbornika 100vh pojavit će se traka za pomicanje jer već imamo zaglavlje koje zauzima visinu veličine 4rem. Pojava trake za pomicanje u ovom slučaju bila bi pogreška u dizajnu stoga koristimo CSS-ovu „calc“ funkciju, a ona izgleda ovako:

```
height: calc(100vh - 4rem);
```

100vh će uvijek zauzimati cijelu visinu ekrana i prilagođavati se ako se ona promjeni, a 4rem je fiksna veličina koju oduzimamo od cijele visine ekrana te će na taj način izbornik uvijek zauzeti sav prostor od zaglavlja do dna ekrana bez pojave trake za pomicanje. U komponenti ćemo postaviti par generičkih poveznica koje se obično nalaze unutar izbornika. „Home“, „Gallery“, „About“ i „Contact“. Deklariramo globalnu varijablu koja će sadržavati niz MenuItem objekata te u HTML predlošku kreiramo listu i pomoću „ngFor“ direktive ispisujemo <li> elemente koji sadrže ikonu i naziv poveznice te se klikom na poveznicu učitava tražena komponenta.

menu.component.ts

```
@Component({
  selector: 'app-menu',
  templateUrl: './menu.component.html',
  styleUrls: ['./menu.component.scss'],
})
export class MenuComponent {
  menuItems: MenuItem[] = [
    { label: 'Home', icon: 'icon', route: 'home' },
    { label: 'Gallery', icon: 'icon', route: 'gallery' },
    { label: 'About', icon: 'icon', route: 'about' },
    { label: 'Contact', icon: 'icon', route: 'contact' },
  ]; }
```

menu.component.html

```
<div class="menu-wrapper">
  <ul>
    <li
      class="menu-item"
      *ngFor="let item of menuItems"
      [routerLink]="item.route"
      routerLinkActive="router-link-active"
    >
      {{ item.label }}
    </li>
  </ul>
</div>
```

menu.component.scss

```
.menu-wrapper {
  box-shadow: rgba(99, 99, 99, 0.2) 0px 2px 8px 0px;
  height: calc(100vh - 4rem);
  background-color: #cf003d;
  color: white;

  ul {
    list-style-type: none;
    padding: 0;
    margin: 0;

    li {
```

```
      padding: 1rem;
      font-size: 1.2rem;
      border-bottom: 0.05rem solid white;
      cursor: pointer;
    }
  }
}
```

7.2 Raspored komponenti

Dodali smo osnovne stilove za izbornik, ali možemo primjetiti da nismo definirali njegovu širinu. Za raspored izbornika, zaglavlja i sadržaja na ekranu koristit ćemo „CSS Grid“. Pomoću CSS Grid-a možemo podjeliti ekran na redove i stupce te odabrati koliko će ih koja komponenta zauzimati. Veličina redova i stupaca je dinamična, što znači da ju CSS kalkulira u pozadini. Kako bismo to napravili na taj način generirat ćemo još jednu komponentu „Layout“ koja će biti zadužena za raspored glavnih komponenti. Prije toga potrebno je ažurirati našu glavnu komponentu i „app-routing.module.ts“. U modulu za rute definiramo adrese za naše nove komponente. Radi Layout komponente potrebno je definirati još jednu razinu izlaza ruta (engl. „Router outlet“).

app-routing.module.ts

```
const routes: Routes = [
  {
    path: '',
    component: LayoutComponent,
    children: [
      { path: 'home', component: HomeComponent },
      { path: 'gallery', component: GalleryComponent },
      { path: 'contact', component: ContactComponent },
      { path: 'about', component: AboutComponent },
    ],
  },
  { path: '**', component: LayoutComponent },
];

@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule],
})
export class AppRoutingModule {}
```

Angular provjerava podudara li se adresa sa nizom ruta tim redom kojim su rute definirane, stoga za ulaznu rutu, odnosno kućnu adresu (http://localhost:4200) prikazujemo Layout komponentu. Da bi se komponente ispravno učitavale potrebno je u app.component.html

dodati <router-outlet></router-outlet>, to je element koji projicira komponente kada se adresa u pregledniku podudara s adresom u modulu za rute. Na primjer – Kada je adresa u pregledniku http://localhost:4200/gallery tada će se u router-outlet elementu prikazati gallery.component.ts. Unutar prvog objekta rute nalazi se svojstvo (engl. „Property“) „children“ koji je druga razina router-outlet-a što znači da ćemo u layout.component.html datoteci imati još jedan <router-outlet>. Ukratko – index.html učitava app.component.ts, app.component.ts kroz svoj router-outlet učitava layout.component.ts koji u sebi ima učitano zaglavlje, izbornik i router-outlet drugog reda koji dinamički učitava komponente Home, Gallery, About i Contact te na taj način Angular rješava pitanje Jednostraničnih aplikacija (engl. „Single page application“ – SPA). Ažurirani kod izgleda ovako:

app.component.html

```
<router-outlet></router-outlet>
```

layout.component.html

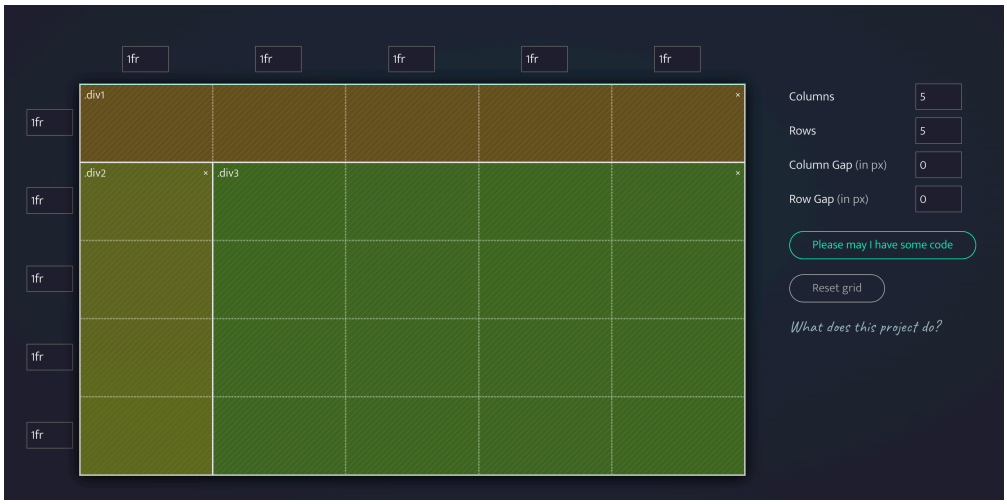
```
<div class="layout">
  <app-header></app-header>
  <app-menu></app-menu>
  <div class="content">
      <router-outlet></router-outlet>
  </div>
</div>
```

layout.component.scss

```
.layout {
  display: grid;
  grid-template-columns: repeat(7, 1fr);
  grid-template-rows: repeat(6);
  grid-column-gap: 0px;
  grid-row-gap: 0px;

  app-header {
    grid-area: 1 / 1 / 2 / 8;
  }
  app-menu {
    grid-area: 2 / 1 / 7 / 2;
  }
  .content {
    grid-area: 2 / 2 / 7 / 8;
    padding: 1rem;
  }
}
```

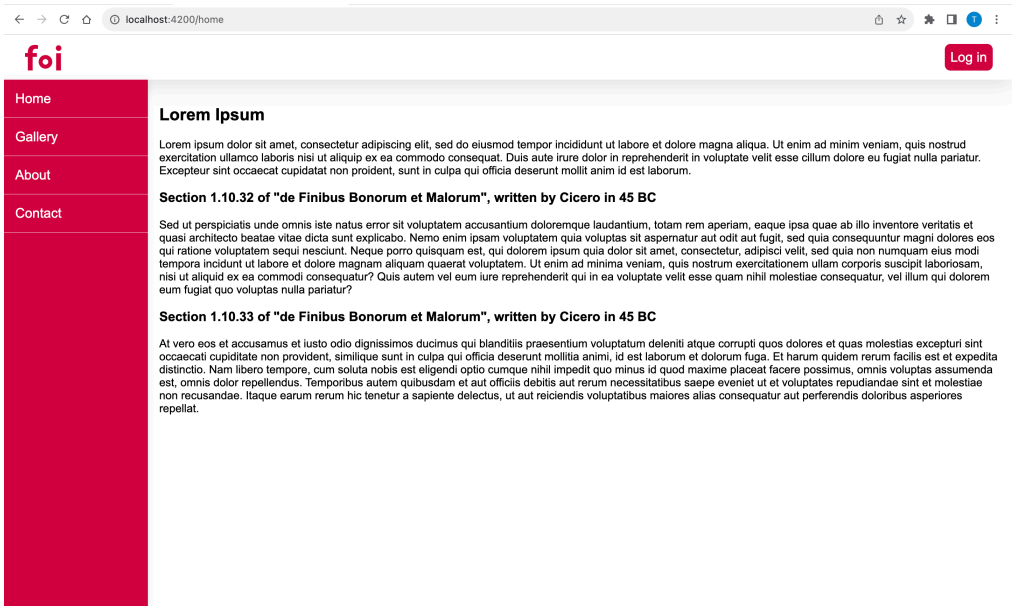
CSS grid u početku može izgledati zbunjujuće, ali zapravo je jako jednostavan i intuitivan. Na roditeljski element (layout) smo postavili display:grid kako bi CSS znao kako se treba ponašati. Za **`grid-template-columns`** postavili smo vrijednost „`repeat(7, 1fr)`“ što u praksi znači

da želimo podjeliti ekran na sedam jednakih stupaca, ako se veličina ekrana promjeni ili ako se prozor smanji stupci će se prilagoditi na način da i dalje svi imaju istu širinu. Linija ispod je isto to samo se odnosi na redove, ali u tom slučaju nemamo postavljen „1fr“ jer ne želimo da nam CSS računa visinu svakog reda iz razloga što smo postavili fiksnu visinu zaglavlja. Ispod toga možemo postaviti razmak između redova i/ili stupaca koji je u ovom slučaju 0 piksela.
Nadalje, **`grid-area`** odnosi se na stupce i redove koje element zauzima i napisan je u obliku row-start / column-start / row-end / column-end. Vrijednost za zaglavlje je: 1 / 1 / 2 / 8 što znači da zaglavlje počinje u prvom redu i prvom stupcu te da završava na početku drugog reda i na početku osmog stupca. odnosno zauzima cijeli prvi red i proteže se kroz sve stupce. Ne samo da na ovaj način možemo jednostavno poslagati elemente već imamo i alat pomoću kojega možemo odabrati broj stupaca i redova, nacrtati šablonu i dobiti generirani HTML kod i CSS stilove. Alat se nalazi na adresi https://cssgrid-generator.netlify.app/

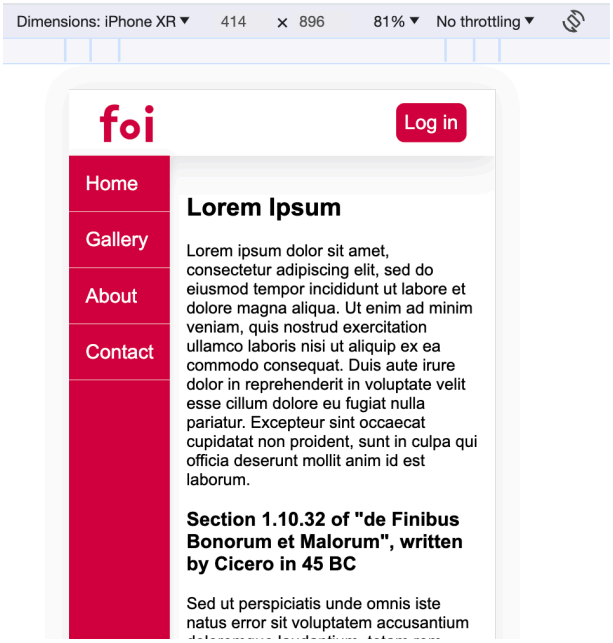


Slika 14: CSS Grid generator

Sada imamo složen kostur stranice, komponente koje se učitavaju klikom na gumbe u izborniku uvijek će biti prikazane desno od izbornika i ispod zaglavlja, odnosno unutar elementa sa „content“ CSS klasom. U layout.component.scss datoteci smo kod content elementa postavili padding jer bi se u suprotnom sadržaj naslanjao na izbornik s obzirom da nismo postavili razmak između stupaca. Nakon svih ovih promjena naša stranica izgleda ovako



Slika 15: Prikaz rasporeda osnovnih komponenti na velikom ekranu



Slika 16: Prikaz rasporeda osnovnih komponenti na mobilnom ekranu

Stranica se lijepo skalira, nema lomljenja sadržaja, ali ovaj stil izbornika nije atraktivan na mobilnom ekranu, štoviše – nepotrebno zauzima prostor na već dovoljno malom ekranu te uz sve to nije praktičan. Na malim ekranima izbornik ne treba biti stalno vidljiv, želimo iskoristiti što je više moguće prostora na ekranu za prikaz sadržaja. Obzirom na to da je „Log in“ gumb u zaglavlju isključivo estetski i nema funkcionalnost zamjenit ćemo ga logom, a na mjesto loga ćemo staviti ikonu za izbornik te želimo isključivo kada stranicu pregledavamo na mobilnim ekranima. Pod mobilne ekrane možemo kategorizirati sve ispod 500 piksela.

7.3 Medijalni upiti

Kako bismo aplicirali neke stilove isključivo u trenutcima kada je razlučivost ekrana ispod 500 piksela koristit ćemo medijalni upit koji izgleda ovako:

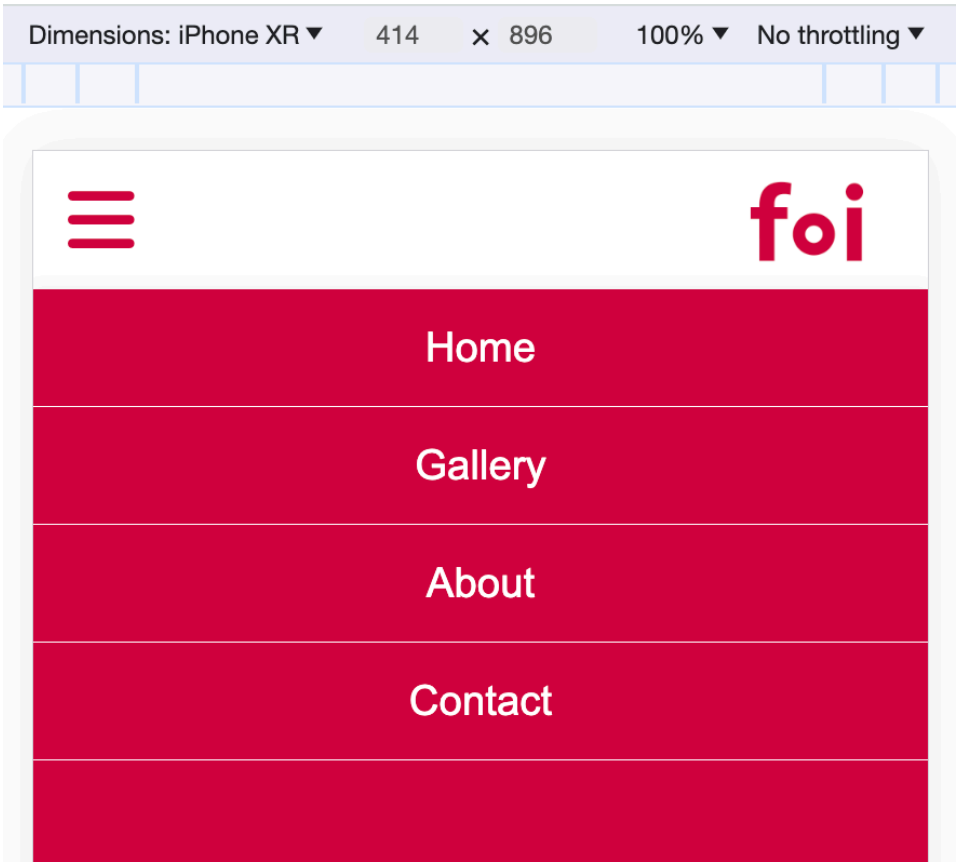
layout.component.scss

```
  @media only screen and (max-width: 500px) {
    app-menu {
      grid-area: 2 / 1 / 7 / 8;
      z-index: 1;
      text-align: center;
    }
  .content {
      grid-area: 2 / 1 / 7 / 8;
      padding: 1rem;
    }
  }
```

Ovaj dio koda će raširiti izbornik preko cijelog ekrana počevši od drugog reda jer ne želimo prekriti zaglavlje, uz to ćemo raširiti sadržaj i preko prvog stupca jer u ovom slučaju nemamo fiksni izbornik. Kako bismo bili sigurni da će izbornik biti iznad sadržaja, odnosno da sadržaj neće biti vidljiv postavili smo „z-index“ pomoću kojega možemo pomicati elemente ispred ili iza drugih elemenata. U header.component.html dodat ćemo još jedan div element u koji ćemo staviti ikonu izbornika, a ikonu ćemo dobiti preko stranice „Font Awesome“. Dovoljno je unijeti i potvrditi svoju mail adresu i imamo besplatan pristup tisućama font ikona koje možemo jednostavno implementirati u našu stranicu. U index.html datoteku ubacit ćemo <script> tag koji dobijemo na Font Awesome stranici i možemo početi dodavati ikone.

```
<div class="header">
  <div class="menu-icon">
    <i class="fa-solid fa-bars"></i>
  </div>
  <div class="logo-wrapper">
    <img src="./../../assets/images/foi-logo.png" alt="Logo" />
  </div>
  <div class="user">
    <button>Log in</button>
  </div>
</div>
```

Ovdje će nam biti potreban još jedan medijalni upit kojim ćemo sakriti logo, a prikazati ikonu izbornika na ekranima manjim od 500 piksela te nakon novih promjena naša stranica izgleda ovako na malom ekranu:



Slika 17: Promjena izbornika korištenjem medijalnih upita

7.3. Komunikacija komponenti

Sada treba dodati funkcionalnost otvaranja i zatvaranja izbornika, a da bi to postigli moramo imati komunikaciju između zaglavlja i izbornika jer se ikona, odnosno gumb izbornika nalazi u zaglavlju. U Angularu komponente mogu komunicirati s drugim komponentama na više načina, no mi ćemo koristiti servisnu komunikaciju. Pomoću servisa moguća je komunikacija između bilo koje dvije ili više komponenti unutar Angular projekta. Budući da se gumb nalazi u zaglavlju generirat ćemo header.service.ts pomoću Angular CLI naredbe „ng g s header“. U servisu ćemo deklarirati globalnu varijablu „menuOpen“ koja će biti tipa „BehaviorSubject“ koji služi za asinkronu komunikaciju. Možemo se pretplatiti na nju, slušati promjene te varijable i reagirati na njih. Bitno je napomenuti da se kod pretplate bitno i odjaviti s pretplate kako ne bi došlo do curenja memorije. Postoji i druga opcija, a to je korištenje „async“ cijevi (engl. „Pipe“) koja se sama pretplaćuje i odjavljuje pri uništenju komponente, a

točno ćemo to i napraviti. Prvo je potrebno ubaciti servis zaglavlja u layout.component.ts i header.component.ts jer je bitno da obje imaju pristup menuOpen varijabli. Servisi se ubacuju kroz konstruktor, a od prije godinu dana moguće ih je ubaciti i izvan konstruktora. U layout.component.html dodat ćemo „ngIf“ direktivu na <app-menu> element te u nju proslijediti varijablu menuOpen$ i dodati cijev „async“ za automatsku pretplatu i odjavu.

layout.component.html

```
<div class="layout">
  <app-header></app-header>
  <app-menu *ngIf="menuOpen$ | async"></app-menu>
  <div class="content"><router-outlet></router-outlet></div>
</div>
```

layout.component.ts

```
export class LayoutComponent {
  constructor(private headerService: HeaderService) {}

  get menuOpen$() {
    return this.headerService.menuOpen.asObservable();
  }
}
```

Dok smo u header.component.html dodali „click“ događaj na ikonu izbornika i u header.component.ts napravili funkciju koja se poziva na taj klik i prebacuje stanje varijable menuOpen iz istine u laž i obrnutno

header.component.html

```
<div class="header">
  <div class="menu-icon" (click)="toggleMenu()">
    <i class="fa-solid fa-bars"></i>
  </div>
  <div class="logo-wrapper">
    <img src="./../../assets/images/foi-logo.png" alt="Logo" />
  </div>
  <div class="user">
    <button>Log in</button>
  </div>
</div>
```

header.component.ts

```
export class HeaderComponent {
  constructor(private headerService: HeaderService) {}

  toggleMenu() {
    this.headerService.toggleMenu();
  }
}
```

Trenutno ponašanje izbornika čini se u redu, no ne smijemo zaboraviti automatski ga prikazati kada je veličina ekrana preko 500 piksela. Kako bismo to napravili koristit ćemo „HostListener“ direktivu koja može slušati neki događaj, a u našem slučaju taj događaj je „onResize“ te ćemo pozvati funkciju toggleMenu(true) kada širina ekrana pređe 500 piksela.

```
  @HostListener('window:resize', ['$event'])
  onResize(): void {
    if (window.innerWidth >= 500) {
      this.headerService.toggleMenu(true);
    }
  }
```

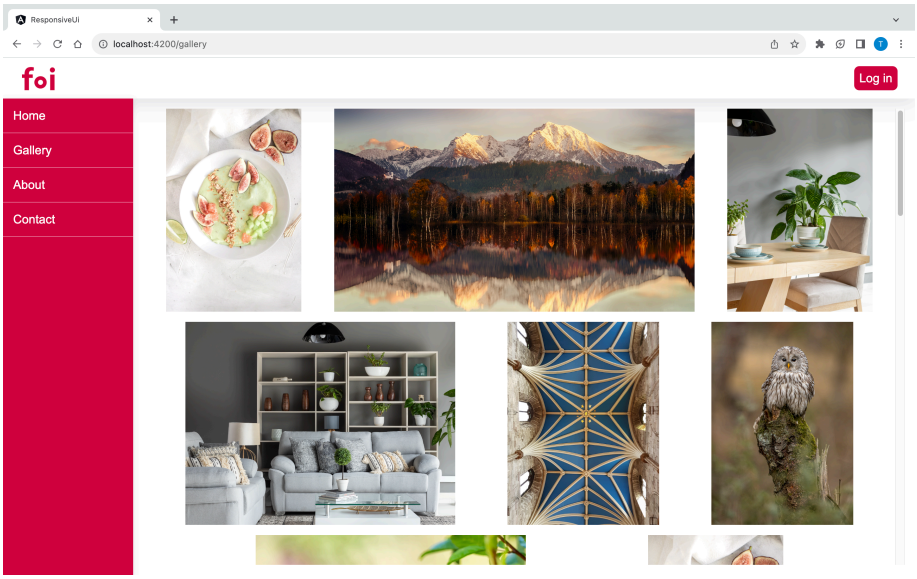
7.4. Galerija

Za kraj složit ćemo jednostavnu galeriju u kojoj će se slike raspoređivati po ekranu obzirom na njegovu širinu koristeći flexbox. Biti će dovoljno napraviti roditeljski element sa CSS klasom „gallery“ te postaviti `display:flex` kako bi koristili njegove funkcionalnosti, postaviti `flex-wrap: wrap` što govori elementima da ne moraju svi biti u istom redu već da se prebace u novi kada bi izašli van ekrana. Nadalje, trebamo postaviti fiksnu visinu i omogućiti prelijevanje elemenata (engl. „Overflow“) kako bi se pojavila traka za pomicanje i kako se slike ne bi zauzele visinu veću od ekrana. Visinu postavljamo pomoću „calc“ metode kao i prije, te postavljamo svojstvo `justify-content: space-around` kako bi se slike u svakom redu zauzele jednako mjesta, odnosno da prostor između i okolo slika u svakom redu bude jednak. Na elemente slika postavit ćemo svojstvo `flex-basis:auto` te maksimalnu fiksnu visinu od 20 rem-ova kako bi ograničili širenje slika visoke razlučivosti. Nakon svih promjena naša CSS datoteka izgleda ovako:

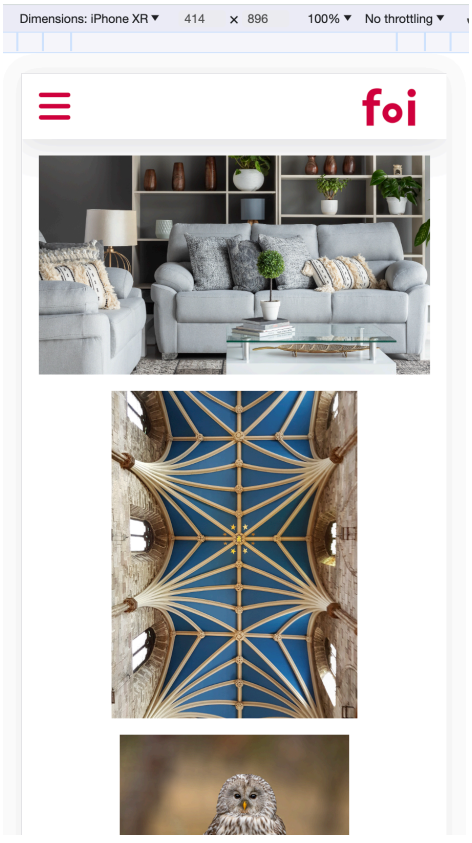
gallery.component.scss

```
.gallery {
  width: 100%;
  display: flex;
  gap: 1rem;
  flex-wrap: wrap;
  overflow: auto;
  height: calc(100vh - 6rem);
  justify-content: space-evenly;

  img {
    max-height: 20rem;
    flex-basis: auto;
  }
}
```

Slike smo ručno ubacili u HTML datoteku radi primjera prikaza, a sada ćemo pogledati kako se slike prikazuju na velikom ekranu, a kako na mobilnom ekranu.



Slika 18: Galerija na velikom ekranu



Slika 19: Galerija na malom ekranu

8. Zaključak

Na primjeru ove jednostavne Web stranice možemo lako uočiti probleme prilagodljivosti s kojima se programeri susreću pri izradi korisničkih sučelja. Isto tako, jasno je vidljivo koliko su Web tehnologije napredovale kao i alati bez kojih je danas nezamislivo izrađivati Web sučelja. U ovom radu napravili smo Web stranicu sa prilagodljivim sučeljem, prilagodljivom galerijom i dinamičkim izbornikom. Angular je korišten isključivo kao ispomoć preusmjeravanje adresa te komunikaciju između više dijelova stranice dok se sav dizajn i prilagodljivost kodirala pomoću domaćih (engl. „Native“) CSS alata. Nismo koristili gotove knjižnice tipa „Boostrap“ jer su to polu-gotova rješenja koja bi u startu prikrila probleme na koje programer početnik treba naići kako bi shvatio ćemu zapravo takve knjižnice služe. Nakon što smo vidjeli kako su izgledale prve stranice te koje su se tehnologije koristile za postizanje prilagodljivosti možemo zaključiti kako su neke od tih tehnologija relevantne još i danas. Još možemo primijetiti kako apsolutno svaka Web stranica danas mora biti pregledna na mobilnom uređaju jer iz godine u godinu raste postotak pregledavanja Weba preko mobitela. Kroz ovaj rad dotakli smo se većine problema kod razvoja Web korisničkih sučelja te smo pokazali na koji način se mogu riješiti. Naravno, ovi primjeri nisu jedini način za rješavanje tih problema, ali pristup greškama je osobna preferenca programera koju temelji na znanju, iskustvu i dokumentaciji projekta.

Popis literature

[1] Kyle Simpson, „You don't know JS.“

[2] Jeremy Keith, 2009. [Mrežno]. Available: https://adactio.com/journal/1595

[3] Andrew Hunt, David Thomas, „The Pragmatic Programmer: From Journeyman to master“

[4] Ben Frain „Responsive Web design with HTML5 and CSS“ (Fourth edition)

[5] Ethan Marcotte, 2010. [Mrežno] https://alistapart.com/article/responsive-web-design/

[6] [Mrežno] https://en.wikipedia.org/wiki/HTML#History

[7] [Mrežno] https://www.businessinsider.com/flashback-this-is-what-the-first-website-ever-looked-like-2011-6

[8] Peter Gasston, 2015. „The book of CSS3 Second edition“

Popis slika