UNIVERZA V LJUBLJANI

FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Marija Štokelj

Uporaba SVG pri razvoju odzivnih spletnih strani

DIPLOMSKO DELO
VISOKOŠOLSKI STROKOVNI ŠTUDIJSKI PROGRAM PRVE STOPNJE
RAČUNALNIŠTVO IN INFORMATIKA

Ljubljana, 2014

UNIVERZA V LJUBLJANI

FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Marija Štokelj

Uporaba SVG pri razvoju odzivnih spletnih strani

DIPLOMSKO DELO
VISOKOŠOLSKI STROKOVNI ŠTUDIJSKI PROGRAM PRVE STOPNJE
RAČUNALNIŠTVO IN INFORMATIKA

Mentor: prof. dr. Matjaž Branko Jurič

Ljubljana, 2014

Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Tržaška 25
1001 Ljubljana, Slovenija
telefon: 01 476 84 11
01 476 83 87
faks: 01 426 46 47
01 476 87 11
www.fri.uni-lj.si
e-mail: dekanat@fri.uni-lj.si

Št. naloge: 00481 / 2013
Datum: 12.4.2013

Univerza v Ljubljani, Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Kandidat: **MARIJA ŠTOKELJ**

Naslov: **UPORABA SVG PRI RAZVOJU ODZIVNIH SPLETNIH STRANI**
USING SVG FOR RESPONSIVE WEB SITE DEVELOPMENT

Vrsta naloge: Diplomsko delo visokošolskega strokovnega študija prve stopnje

Tematika naloge:

Opišite arhitekturo odzivnega spletnega oblikovanja in opravite pregled tehnologij za izdelavo odzivnih spletnih strani. Podrobno analizirajte jezik SVG (Scalable Vector Graphics), proučite sintakso, grafične elemente in način uporabe rastrske grafike, animacij, skriptnega jezika, SVG DOM (Document Object Model) modela in CSS ter uporabe SVG v spletnih aplikacijah. Razvijte praktično spletno aplikacijo z uporabo omenjenih tehnologij s poudarkom na SVG in odzivnim oblikovanjem.

Mentor: Dekan:

prof. dr. Branko Matjaž Jurič prof. dr. Nikolaj Zimic

Fakulteta za računalništvo in informatiko • UNIVERZA V LJUBLJANI • 2

IZJAVA O AVTORSTVU DIPLOMSKEGA DELA

Spodaj podpisana Marija Štokelj, z vpisno številko 63090284, sem avtorica diplomskega dela z naslovom:

Uporaba SVG pri razvoju odzivnih spletnih strani

S svojim podpisom zagotavljam, da:

- Sem diplomsko delo izdelala samostojno pod mentorstvom prof. dr. Matjaža Branka Juriča,
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano verzijo diplomskega dela
- soglašam z javno objavo elektronske oblike diplomskega dela v zbirki »Dela FRI«.

V Ljubljani, dne 2. Februarja 2014 Podpis avtorice:

Zahvala

Najprej se zahvaljujem svoji družini, ki me že od malega spodbuja pri izobraževanju, ter mi je omogočila študij na fakulteti in mi vedno stala ob strani.

Iskrena zahvala gre mentorju prof. dr. Matjažu Branku Juriču za odlično strokovno svetovanje, vso potrpežljivost in vloženo delo pri pregledovanju naloge.

Hvala tudi vsem prijateljem in sodelavcem za razumevanje in spodbudne besede. Posebej pa se zahvaljujem še mojemu Matjažu za pomoč in podporo tudi v najbolj napornih trenutnih.

Kazalo

Povzetek 1
Abstract 3
1 Uvod 5
2 Opis in arhitektura odzivnega spletnega oblikovanja 7
2.1 Spremenljiva postavitev 8
2.2 Prilagodljive slike 9
2.3 Medijske poizvedbe 10
2.4 Pristopi k razvoju odzivnih spletnih strani 12
2.5 Tehnologije za izdelavo odzivnih spletnih strani 12
2.5.1 Izvajanje na odjemalcu 13
2.5.2 Izvajanje na strežniku 14
2.5.3 Skupna uporaba 15
3 Pregled jezika SVG in načinov uporabe na spletu 17
3.1 Sintaksa dokumentov SVG 18
3.2 Grafični elementi SVG 19
3.2.1 Vektorska grafika 19
3.2.2 Rastrska grafika 20
3.2.3 Besedilo 21
3.3 Napredna uporaba SVG 21
3.3.1 Animacije SVG 22
3.3.2 Skriptni jezik in SVG DOM 22
3.3.3 Prehodi in animacije CSS 24
3.4 Uporaba SVG na spletu 24
3.4.1 Uporaba orodij in optimizacija datotek 24
3.4.2 Vključitev dokumentov SVG v spletno stran 25
3.4.3 Knjižnice za delo s SVG 25
4 Predstavitev spletne strani 27
5 Razvoj spletne strani 33
5.1 Odzivnost spletne strani 33
5.2 Knjižnica D3 36
5.2.1 Zajem vhodnih podatkov 36

5.2.2 Umerjanje v podano območje 36
5.2.3 Vstavljanje elementov 37
5.2.4 Oblikovni slog 40
6 Testiranje in rezultati 43
7 Sklep 45
Seznam slik in tabel 47
Literatura 49

Seznam uporabljenih kratic

API	Application Programming Interface
ASV	Adobe SVG Viewer
BOM	Browser Object Model
CSS	Cascading Style Sheets
CSV	Comma-separated values
D3	Data-Driven Documents
DOM	Document Object Model
DRR	Device Description Repository
HTML	Hyper Text Markup Language
HTTP	Hypertext Transfer Protocol
IRI	Internationalized Resource Identifier
JPEG	Joint Photographic Experts Group
JSON	JavaScript Object Notation
JSP	JavaServer Pages
MIME	Multipurpose Internet Mail Extensions
PGML	Precision Graphics Markup Language
PNG	Portable Network Graphics
RESS	Responsive Web Design with Server Side
REST	Representational state transfer
SMIL	Synchronized Multimedia Integration Language
SVG	Scalable Vector Graphics
TSV	Tab-separated values
URI	Uniform Resource Identifier
VML	Vector Markup Language
W3C	World Wide Web Consortium
WURFL	Wireless Universal Resource File
XML	Extensible Markup Language
XSLT	Extensible Stylesheet Language Transformations

Povzetek

Poglavitni namen diplomskega dela je preučitev tehnike odzivnega spletnega oblikovanja in standarda SVG (*Scalable Vector Graphics*) za izdelavo vektorskih grafik, ter združitev obeh v okviru spletne strani. V uvodnem delu je predstavljena problematika zagotavljanja efektivnih, uporabniku prijaznih strani za izjemno širok nabor mobilnih naprav, ki so danes dostopne na trgu. Kot rešitev k razvoju optimalno prilagojenim spletnim vsebinam na vsaki napravi, je detajlneje predstavljeno odzivno spletno oblikovanje. Standard SVG je nadalje opisan kot zelo primeren grafični format za uporabo na odzivnih straneh, saj ne glede na intenzivnost manjšanja ali večanja, ohranja svojo prvotno ostrino. Obenem so datoteke vektorskih grafik občutno manjše v primerjavi z bitnimi slikovnimi formati, kar je v mobilnem svetu zaradi omejenih virov zelo dobrodošlo. Souporaba obeh rešitev je izvedena na odzivni spletni strani, namenjeni izdelavi diagramov v SVG jeziku, ki smo jih kreirali s pomočjo knjižnice D3 (*Data-Driven Documents*). V sklepnem delu so predstavljeni rezultati testiranja na več brskalnikih in napravah, ter podane ugotovitve o prednostih in slabostih obeh raziskanih področij.

Ključne besede:

Odzivno spletno oblikovanje, SVG, vektorske grafike, spletna stran, D3.

Abstract

The principal goal of this thesis is to examine responsive web design approach and the SVG (*Scalable Vector Graphics*) standard for vector graphics. Both topics are ultimately used on designing a web page. The opening chapter presents the difficulty of ensuring the users with effective and accessible web pages, across all kinds of mobile devices that are currently available on the market. Responsive web design is presented as a viable solution for the development of web pages, which are optimised for viewing on any device. The following section describes the SVG standard as a particularly suitable graphics file format for responsive pages, as it preserves the original sharpness of images regardless of scaling intensity. Furthermore, the size of vector graphics files is significantly smaller in comparison to raster graphics, which is a beneficial fact, since mobile devices have more limited resources. Combination of both solutions is implemented on a responsive web page for generating graphs. Those are created with SVG by using the D3 (*Data-Driven Documents*) library. The conclusion presents the results of testing the web page across multiple browsers and devices, along with our evaluation of the benefits and downsides of both of the discussed topics.

Key words:

Responsive web design, SVG, vector graphics, web page, D3.

1 Uvod

Sodobni svet družbi narekuje hiter življenjski slog. Posamezniki ki znotraj takega okolja stremijo k uspešnosti, so zato primorani v najkrajšem času pridobiti potrebno znanje in ga uporabiti. Informacijski družbi je posledično na voljo veliko število tako tiskanih, kot elektronskih medijev, daleč najbolj široko dostopni pa so spletni mediji. Z velikim porastom cenovno dostopnih mobilnih naprav, ki omogočajo uporabo interneta, je splet v zadnjih letih postal še lažje dosegljiv. Uporabniki lahko tako poiščejo želene informacije ne le kadar koli, temveč skorajda kjer koli. Oblikovanje efektivnih spletnih strani za tako velik nabor naprav, predstavlja zahteven razvijalni problem. Poleg manjše sistemske zmogljivosti, imajo posamezne naprave različno velike zaslone, ki se nadalje ločijo po kvaliteti prikazane slike. Razvijalci se pri izdelavi prilagodljivih strani poslužujejo različnih tehnik, ena od bolj uspešnih pa je odzivno spletno oblikovanje. Pristop zapoveduje načine, s katerimi formiramo enotno spletno stran, ki se v različnih okoljih samodejno umerja. To pomeni, da so posamezni elementi (npr. menijska vrstica, besedilo in slike) smiselno postavljeni in jasno razumljivi, ne da bi moral uporabnik pretirano drseti po zaslonu naprave, ter širiti in manjšati vidno polje. Rezultat so hitreje pridobljene informacije in boljša uporabniška izkušnja.

Ko najdemo iskane podatke, pa se lahko zgodi, da jih je preveč in imamo težavo z njihovo primerjavo in razumevanjem. Za lažjo predstavitev večjih količin informacij se zato pogosto poslužujemo diagramov. Eden od načinov za izdelavo diagramov na spletnih straneh je uporaba označevalnega jezika SVG (*Scalable Vector Graphics*), ali skalabilne vektorske grafike.

Za cilj diplomske naloge smo določili predstavitev uporabe SVG grafik na spletni strani, ki temelji na odzivnem spletnem oblikovanju. V začetku smo si zadali podrobneje predstaviti izvor, namen ter glavne komponente v obeh zastavljenih področjih. Nadalje je opisana sama spletna stran, katere glavni namen je izdelava diagramov. Vrsto grafa, količino vhodnih podatkov ter njihove vrednosti izbere, oziroma vnese uporabnik sam. V nadaljevanju so predstavljena orodja in tehnologije uporabljena pri izdelavi spletne strani. V zaključnem delu so predstavljeni rezultati, ter podani predlogi za morebitni nadaljnji razvoj spletne strani.

2 Opis in arhitektura odzivnega spletnega oblikovanja

Odzivno spletno oblikovanje (*Responsive Web Design*) je tehnika za prilagajanje oblike spletnih strani, tako da je vsa vsebina vedno pregledno umeščena znotraj prikazovalnega okna poljubne velikosti. Poglavitno načelo odzivnega oblikovanja je dojemanje razsežnosti spletne strani kot neodvisne od fiksnega okvirja določene naprave. Atribute kot so dolžina, višina ali velikost pisave je potrebno predstaviti s premičnimi dimenzijami, ki se samodejno prilagodijo zaslonu, tako da je vsebina vedno optimalno predstavljena.

Začetki takšnega pristopa segajo v devetdeseta leta, preden je izšel standard kaskadnih slogovnih pol CSS (*Cascading Style Sheets*). Eden od odmevnejših člankov, ki so predstavili pomembnost prilagajanja vsebine spletnih strani, je objavil John Allsopp, ki trdi [1]: »*Menim da je največja prednost spleta pogosto videna kot omejitev, kot defekt. Narava spleta je fleksibilna in mi, ki smo v vlogi oblikovalcev in razvijalcev, bi morali to fleksibilnost sprejeti, ter ustvarjati strani, ki so s svojo prilagodljivostjo dostopne vsem.«*

Pojem odzivno spletno oblikovanje, kot ga uporabljamo danes, je leta 2010 prvi uporabil spletni razvijalec Ethan Marcotte v članku spletne revije A List Apart. Kot tri osnovne gradnike pri oblikovanju odzivnih strani je združeno predstavil spremenljive mreže (*fluid grids*), prilagodljive slike (*flexible images*) in medijske poizvedbe (*media queries*) [2]. Vsak od konceptov je podrobneje predstavljen v nadaljnjih podpoglavjih.

Razlog da je odzivni dizajn postal izredno priljubljen šele v zadnjih letih, je nedvomno odraz tehnoloških trendov. Še ne dolgo nazaj smo do spleta večinoma dostopali le prek osebnih ali prenosnih računalnikov. Nato pa so na trg začele prihajati razne mobilne naprave kot so tablični računalniki, elektronski bralniki in pametni telefoni. Podatki Statističnega urada Slovenije nakazujejo, da je število uporabnikov, ki do interneta dostopajo preko mobilnih telefonov od leta 2012 do 2013 zrasla iz 20 na 32 odstotkov, tablični računalnik pa je prisoten v 14 % anketiranih gospodinjstev [3, 4]. V prid uporabi odzivnega dizajna so tudi podatki, ki jih je februarja 2013 objavil Cisco. Rezultati njihove globalne raziskave kažejo, da je samo v letu 2012 količina podatkov prenesenih prek mobilnih naprav narasla za 70 odstotkov [5]. Poleg manjših ročnih naprav, postajajo na drugi strani zasloni osebnih računalnikov večji, do spletnih vsebin pa lahko dostopamo tudi preko televizijskih sprejemnikov. Z namenom zagotoviti uporabnikom prijazne in intuitivne spletne vsebine, so se razvijalci začeli posluževati različnih metod. Dva bolj pogosta pristopa sta npr. izdelava ločene strani za mobilne naprave, do katerih uporabniki dostopajo prek druge povezave ter izvorne aplikacije (*native app*), ki jih uporabniki namestijo na mobilno napravo in v veliko primerih do nje dostopajo tudi brez internetne povezave [6].

Slabost odzivnega oblikovanja je zahtevnejša implementacija, saj moramo v eno stran znati vključiti več različnih postavitev in slogov elementov. Po drugi strani pa so njegove prednosti lažje vzdrževanje ter optimizirana podoba na vseh možnih dimenzijah zaslonov, tudi takih, ki bodo morda prišli na trg šele v prihodnosti.

2.1 Spremenljiva postavitev

Prvi korak pri razvoju odzivne spletne strani je priprava spremenljive postavitve elementov. Dimenzije vsakega razdelka strani kot so logotip, glava, navigacija, noga ipd. morajo biti v sorazmerju z dimenzijami prikazovalnega okna. To pomeni, da pri manjšanju okna posamezni deli ne segajo preko robov svojega nadrejenega elementa, temveč se ustrezno skrčijo. Primer spremenljive postavitve je prikazan na sliki 1.

Slika 1: Spremenljiva postavitev elementov [7].

Pri definiranju dimenzij posameznih razdelkov, moramo za izvedbo takega umerjanja izbrati relativno mersko enoto, kot sta odstotek ali *em*. Slednja enota predstavlja privzeto velikost pisave nadrejenega HTML elementa in jo lahko uporabljamo za izražanje vertikalnih in horizontalnih meritev. Včasih pa za ozadje spletnih strani, ali posameznih razdelkov uporabimo slike izdelane z grafičnimi orodji, na katere želimo smiselno umestiti elemente. V takih primerih moramo fiksne dimenzije pretvoriti v spremenljive. To lahko storimo tako, da v slikovnih točkah izmerimo velikost želene slike ter velikost prostora v katerega želimo vpeti element. Dimenzijo izraženo v odstotkih nato izračunamo po formuli:

$$\frac{Dimenzija\ vpetega\ razdelka}{Dimenzija\ nadrejenega\ razdelka} \times 100 = rezultat$$

Tako pridobljeno vrednost nato uporabimo v definiciji CSS podrejenega razdelka. Mnogokrat je lahko število decimalnih mest v rezultatu zelo dolgo. Če želimo na primer znotraj okvira širokega 860px umestiti besedilo, ki zaseda 644px, bi po izračunu z zgornjo enačbo dobili rezultat:

$$\frac{644}{870} \times 100 = 74{,}022988506$$

Ob prvem pogledu na tako število hitro pomislimo na okrajšavo števila, vendar je za optimalno prileganje na vseh velikostih zaslonov pomembno, da pustimo rezultat nespremenjen [2].

Ob mreži, ki se vedno prilagaja prikazovalnemu oknu, je pomembno tudi kako predstavimo vsebino posameznega razdelka. Na zelo velikem zaslonu lahko uporabimo večjo velikost besedila, večje presledke med posameznimi odstavki, ter širši rob besedila. Na manjših zaslonih pa moramo dani prostor bolje izkoristiti. Pomembno je uporabiti manjšo velikost pisave, s čimer bralcu olajšamo branje, saj zmanjšamo število prehodov v novo vrstico in mu prihranimo pretirano drsenje po zaslonu navzdol. Prav tako je smiselno minimalizirati rob besedila ter presledke med posameznimi deli besedila, vendar le do mere, ko besedilo še vedno ohrani jasnost in urejenost. Poleg omenjenih bolj intuitivnih prilagoditev teksta, pa ne smemo pozabiti tudi na privzete velikosti okvirjev blokovskih elementov v HTML. Tipično so to naslovi (npr. <*h1*> in <*h2*>) odstavek <*p*>, seznami <*ol*> in <*ul*> ter podobni elementi, katerih privzeti robovi se malenkostno razlikujejo v posameznih brskalnikih.

Ker je pri pregledovanju strani na zelo majhnem zaslonu tudi minimalen rob lahko odveč ni presenetljivo, da nekateri razvijalci v datotekah CSS privzete okvirje elementov najprej ponastavijo na ničlo vrednost. Na manj kompleksnih straneh je dovolj, da razvijalec to stori sam. Obstajajo pa tudi naprednejše inicializacije, ki zajamejo notranje in zunanje robove, poravnavo ter velikost teksta. Zelo znana je na primer Eric Meyerjeva ponastavitev, ki je praktično vzorec prosto dostopne kode CSS, namenjene razvijalcem spletnih vsebin [8].

2.2 Prilagodljive slike

Če smo ustrezno pripravili spremenljivo postavitev elementov strani, je umerjanje slik v prikazovalno polje trivialno opravilo, ki ne zahteva več kot eno vrstico kode CSS:

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

Z uporabo te deklaracije zagotovimo, da širina slik vključenih z elementom <*img*>, ne bo nikoli presegala širine svojega nadrejenega elementa. Enako nastavitev lahko uporabimo tudi za umerjanje elementov <*embed*>, <*object*> in <*video*> [9].

Za izdelavo učinkovite odzivne strani ni dovolj, da sliko le prilagodimo dimenzijam zaslona. Na majhnih mobilnih napravah lahko uporaba večjega števila slik zmanjša preglednost vsebine, zato je potrebno premisliti, ali je prikaz vseh slik sploh potreben. Pri grafikah, ki so bistvene za razumevanje vsebine moramo nadalje preudariti, kako veliko datoteko bomo ponudili uporabniku. Na osebnih računalnikih nam nalaganje spletnih strani pri povprečni internetni povezavi običajno dela dovolj hitro. Mobilne naprave imajo na drugi strani manj zmogljive procesorje, manjše delovne pomnilnike ter počasnejši prenos podatkov**.** Kljub temu uporabniki pričakujejo kratek čas nalaganja vsebine spletnih strani na vseh napravah. Pri spletnih straneh, ki vsebujejo slike, videe ali animacije je zato odzivno spletno oblikovanje smiselno nadgraditi in zagotoviti, da se te sploh ne prenašajo na ciljno napravo. Pri slikah ki jih vključujemo kot ozadje preko deklaracije CSS *background-image*, lahko za pogojno nalaganje uporabimo kar medijske poizvedbe, ki so podrobneje predstavljene v naslednjem podpoglavju. Paziti moramo le, da za vsako poizvedbo postavimo natančne mejne pogoje, tako da se slika na napravo prenese res le za ustrezno poizvedbo [10]. Ta način smo uporabili tudi na našem praktičnem primeru, saj nismo prikazovali veliko slik in tako nismo potrebovali zahtevnejšega preverjanja.

Smiselno je tudi, da bitne določene grafike (npr. v formatu PNG ali JPEG) zamenjamo z vektorskimi grafikami, kot je format SVG. Te zasedejo veliko manj prostora, obenem pa pri vseh ločljivostih ohranjajo enako kvaliteto.

2.3 Medijske poizvedbe

Kot del specifikacije CSS poznamo medijske tipe (*media types*) s katerim določimo na kateri vrsti naprave bomo uporabili določene oblikovne lastnosti. Lahko se na primer odločimo, da bomo na zaslonu prikazali manjšo velikost pisave pri tiskanju na papir pa nekoliko večjo [11]. Vse vrste naprav na katere se lahko sklicujemo z medijskimi tipi so opisane v tabeli 1:

Vrsta naprave	**Opis**
all	Vse naprave.
braille	Braillove naprave, ki informacije podajajo taktilno.
embossed	Naprave namenjene reliefnemu tiskanju.
handheld	Mobilne naprave z manj razpoložljivimi viri. Zaradi napredka v razvoju se danes manj uporablja in ga nadomešča *screen*.
print	Naprave namenjene tiskanju ter razni predogledi tiskanja.
projection	Projektorji ali celozaslonski pogled v nekaterih brskalnikih.
screen	Primarno namenjen računalnikom, ki so bili dovolj zmogljivi, da so lahko prikazali večje število slogovnih definicij. Danes so mobilne naprave sposobnejše in imajo

	običajno dovolj dobro internetno povezavo, da tudi zanje uporabimo ta tip.
speech	Sintetizatorji govora.
tty	Teleprinterji, terminali ter podobne naprave z omejeno zmožnostjo predstavitve podatkov.
tv	Televizijski sprejemniki.

Tabela 1: Vrste medijskih tipov.

Konzorcij svetovnega spleta W3C je 4. Aprila 2001 objavil delovni osnutek (*Working Draft*) o medijskih poizvedbah, ki so jih predstavili kot razširitev medijskih tipov. Namen nove tehnologije ne obsega več zgolj določitev vrst naprav, temveč tudi opredelitev ostalih lastnosti naprav in brskalnikov, ki vplivajo na prikaz spletne strani. Po tem ko so medijske poizvedbe (v ponekod manjšem in drugod večjem obsegu) dobile podporo v več spletnih brskalnikih, so bile 19. Junija 2012 razglašene za priporočilo W3C [12].

Medijske značilnosti so nabor lastnosti prikazovalne naprave, ki neposredno vplivajo na videz spletne strani. Njihova sintaksa sestoji iz dveh delov. V prvem delu z uporabo že omenjenih medijskih tipov določimo vrsto ciljne naprave, v drugem delu pa z logičnim izrazom določimo eno ali več medijskih značilnosti. Na sliki 2 je prikazana ena od medijskih poizvedb iz našega praktičnega primera, kjer naslavljamo vse tipe naprav pri katerih je minimalna širina zaslona 96 em.

```
457
458 @media all and (min-width:96em){
459   #main{
460     width:80%;
461   }
462    .izbira{
463     padding-bottom:85%;
464   }
465 }
```

Slika 2: Medijska poizvedba.

V medijski poizvedbi lahko določimo eno ali mnogo medijskih značilnosti. Če jih naštejemo več, moramo med njimi uporabiti logični operator in (*AND*). Običajno posamezni značilnosti priredimo konkretno vrednost, ni pa nujno. Z barvo lahko brez dodane vrednosti poizvemo zgolj ali ima prikazovalna naprava barvni zaslon, če pa dodamo tudi vrednost, nas zanima z najmanj koliko biti naprava zapiše posamezno barvo. Pri posamezni značilnosti lahko uporabimo tudi predloga min in max, s katerima določimo najmanjšo ali največjo dovoljeno vrednost. Vse medijske značilnosti po katerih lahko poizvedujemo so predstavljene v tabeli 2.

Ime značilnosti	**Opis**
width	Širina polja namenjenega prikazovanju vsebine na ciljni napravi (npr. osrednje polje brskalnika).
height	Višina polja namenjenega prikazovanju vsebine na ciljni napravi.

device-width	Širina celotnega prikazovalnega polja ciljne naprave (npr. zaslon računalnika).
device-width	Višina celotnega prikazovalnega polja ciljne naprave.
orientation	Orientacija naprave je pokončna (portrait), če je višina prikazovalnega polja vsebine večja od širine. V obratnem primeru je orientacija ležeča (landscape).
aspect-ratio	Razmerje med širino in višino prikazovalnega polja vsebine.
device-aspect-ratio	Razmerje med širino in višino celotnega prikazovalnega polja naprave.
color	Barvna globina, ali število bitov, ki na ciljni napravi definirajo barvo.
color-index	Določa število zapisov v barvni iskalni tabeli. Če ciljna naprava nima tabele, je vrednost značilnosti 0.
monochrome	Podana vrednost pove število bitov na slikovno točko.
resolution	Ločljivost zaslona na napravi. Vrednost podamo z enotama dpi ali dpcm.
scan	Namenjena tv medijskim tipom, ali natančneje skeniranju videa, ki je lahko prepleteno ali progresivno.
grid	Vrednost je 1, če naslavljamo naprave z vrstičnim prikazom (*grid-based*) podatkov (npr. razni terminali ali zasloni z eno fiksno velikostjo pisave). Sicer je vrednost 0.

Tabela 2: Vrste medijskih značilnosti.

2.4 Pristopi k razvoju odzivnih spletnih strani

Poleg razlike v tehniki izvedbe zahteva odziven dizajn tudi drugačen miselni pristop k razvoju. Spletno stran pogosto začnemo razvijati za večje zaslone, ter nato postopamo proti manjšim. Takemu postopku pravimo postopna degradacija. Kot pravilen pristop k odzivnemu oblikovanju pa izkušeni razvijalci diktirajo progresivno nadgrajevanje ali mobilno prioriteto (*mobile first*) [13]. Osrednji namen postopka je, da si primarno zamislimo in implementiramo postavitev za najmanjše zaslone, nato pa po korakih nadgrajujemo videz do največjega zaslona. Najpomembnejši razlogi za izbiro tega pristopa so:

- Prodaja pametnih mobilnih naprav in število uporabnikov, ki do spleta dostopajo prek njih že več let nenehno narašča.
- Prioritetni razvoj za mobilne naprave zagotavlja, da se posvetimo zgolj bistvenim komponentam spletne strani, ter ne izgubljamo časa za razvoj dodatne nepomembnih vsebin.
- Interakcija uporabnika z mobilnimi napravami, se razlikuje od uporabe namiznega računalnika. Na voljo imamo na primer upravljanje z dotikom, nagibom naprave, glasovnim izbiranjem, ugotovimo lahko točno lokacijo uporabnika,… S tem znanjem v ospredju zagotovimo, da bodo naše spletne vsebine, ne le delujoče, temveč polno izkoriščene na mobilnih napravah.

2.5 Tehnologije za izdelavo odzivnih spletnih strani

Temelj na katerem gradimo razvoj odzivnih spletnih strani je zaznavanje lastnosti odjemalčeve naprave ter prilagoditev vsebine tej napravi. To lahko storimo na več načinov, primarno pa ločimo dva načina:

- **na odjemalcu** (*client-side*): Zaznavanje lastnosti naprave in / ali prilagajanje vsebine, se izvajata na posamezni ciljni napravi, ki dostopa do strani. Najbolj uporabljeni tehnologiji pri tem sta CSS3 in JavaScript. Prednosti pristopa sta lažja in hitrejša implementacija. Po drugi strani naletimo tudi na težave. Nekatere naprave, s katerimi lahko danes dostopamo do spleta, nimajo podpore za CSS3. Tu sicer lahko uporabimo ogrodja JavaScript, ki znajo simulirati izvajanje medijskih poizvedb, a obstajajo tudi naprave, ki jezika JavaScript ne podpirajo. Da se prilagajanje vsebine strani lahko izvede na odjemalcu, jo moramo najprej prenesti nanj. To je za manj zmogljive naprave lahko zelo obremenjujoče in znatno vpliva na hitrost nalaganja strani ter s tem na uporabniško izkušnjo.
- **na strežniku** *(server-side*): Izvajanje zaznavanja naprave in vsebine poteka na strani strežnika. Pristop se velikokrat poimenuje tudi kot odzivno spletno oblikovanje s strežniškimi komponentami ali RESS (*REsponsive web design with Server Side components*) [14]. Ta način je sicer zahtevnejši, vendar je v obenem tudi veliko bolj zmogljiv. Primarno nam omogoča, da izvemo veliko več podatkov o odjemalcu (npr. znamko in model naprave, različico brskalnika, različico operacijskega sistema ipd.), kar nam omogoča veliko višji nivo prilagajanja vsebine. Nadalje lahko vse zahtevnejše spremembe podatkov (npr. pomanjšava slik in selektiven prenos podatkov) izvedemo na strežniku in jih nato v optimizirani obliki pošljemo odjemalcu, ter tako skrajšamo čas nalaganja vsebine.

V nadaljevanju so predstavljene nekatere bolj znane tehnologije obeh pristopov.

2.5.1 Izvajanje na odjemalcu

Medijske poizvedbe, s katerimi poizvedujemo o izvajalnem okolju v CSS, smo že podrobneje opisali kot enega od treh glavnih konstruktov osnovnega odzivnega oblikovanja in nadaljnji opis ni potreben. Osredotočili se bomo na JavaScript in razna ogrodja, ki temeljijo na njem ter ogrodja za izdelavo spremenljive postavitve.

2.5.1.1 JavaScript

Enostavne poizvedbe o lastnostih ciljne naprave lahko izvedemo z običajno uporabo jezika JavaScript, natančneje z uporabo objektnega modela brskalnika ali BOM. V naboru objektov so nam pri odzivnem oblikovanju lahko v pomoč predvsem *window*, *navigator* in *screen*. Vsak od teh ima nadalje lastnosti, s katerimi dobimo informacije kot so širina ali višina okna brskalnika, širina ali višina celotnega zaslona, različica brskalnika, izvajalna platforma ipd [15]. Do lastnosti naprav lahko dostopamo tudi preko nekaterih drugih ogrodij JavaScript, ki omogočajo zaznavanje značilnosti (*feature detection*). Nekatera od teh so jQuery, Modernizr, Foresight.js, Enquire.js, Has.js in Dojo. Modernizr je podrobneje opisan v nadaljevanju, saj je trenutno eno najbolj priljubljenih ogrodij. Poleg direktnega poizvedovanja imamo na voljo

tudi manjša namenska ogrodja, ki so mišljena kot alternativa medijskim poizvedbam. Trije konkretni primeri so npr. Adapt.js, Respond.js in css3-mediaqueries.js [16,17].

2.5.1.2 Modernizr

Modernizr je posebno ogrodje JavaScript namenjeno zaznavanju specifikacij sodobnih spletnih standardov kot so CSS3, HTML5 in SVG, ki jih lahko uporabimo na ciljnem brskalniku. Za zaznavanje več funkcionalnosti ogrodje najprej ustvari element, ter mu pripiše določene slogovne lastnosti. Na podlagi odgovorov, ki jih dobi od brskalnika, se ustvari objekt JavaScript, ki vsebuje nabor vrnjenih Boolovih vrednosti [18]. Z dobljenimi rezultati lahko v nadaljevanju odjemalcu selektivno ponudimo ustrezno vsebino.

2.5.1.3 Ogrodja za izdelavo spremenljive postavitve

Pripraviti odzivno postavitev je pri zahtevnih straneh lahko zamudno delo, obstajajo pa orodja, ki nam znajo pri tem pomagati. Danes jih je že zelo veliko, štiri od bolj znanih pa so npr.: Foundation, Gridset, Skeleton in Bootstrap [19]. Vsako orodje ima svoje posebnosti. Običajno imajo vsa že svoje predloge za postavitev okvirjev strani, omogočajo pa tudi personalizacijo, nekatera v večji, druga v manjši meri.

Od naštetih ogrodij je trenutno verjetno najbolj razširjen Bootstrap [20]. Razvila sta ga Twitterjeva razvijalca, temelji pa na uporabi jezikov HTML, CSS in JavaScript. Podpora odzivnemu spletnemu oblikovanju je na voljo od različice 2.0, ki omogoča da se ustvarjena postavitev elementov na podlagi lastnosti ciljne naprave samodejno prilagaja. Poleg osnovnih elementov HTML, so na voljo tudi sestavljene komponente kot so gumbi, spustni meniji, navigacijski meniji, pojavna okna, medijski tipi, itd.

2.5.2 Izvajanje na strežniku

Princip zaznavanja naprave na strežniku deluje na podlagi dveh komponent. Prva je repozitorij, ki vsebuje zbirko znanih lastnosti različnih naprav. Repozitorijem ki hranijo zbirke lastnosti naprav pravimo tudi DDR (*Device Description Repository*). Bolj znana ogrodja so 51Degrees.mobi, DetectRight, DeviceAtlas in WURFL. Za zagotavljanje točnih podatkov se morajo redno posodabljati, saj lahko sicer vračajo napačne podatke, vsa pa so v veliki meri plačljiva. Druga komponenta strežniškega zaznavanja naprav pa je aplikacijski programski vmesnik ali API (*Application Programming Interface*), s katerim pridobimo podatke za iskano napravo [21].

Repozitoriji običajno ponujajo dva načina uporabe: lokalno (*local detection*) in oblačno (*cloud detection*). WURFL na primer ponuja lokalno namestitev za različne platforme (Java, PHP in .NET). Paket vključuje zbirko podatkov v XML formatu in ustrezen API, za dostop do posameznih podatkov. Ta način zagotavlja hitrejšo zagotovitev podatkov, saj ni potrebno

pošiljati zahtev na oddaljen repozitorij. Omogočena je tudi personalizacija strukture DDR datoteke XML, kar je uporabno če želimo dodati posebne specifike, ki jih repozitorij privzeto ne vključuje. Slabost lokalnih namestitev je potreba po posodabljanju zbirke podatkov. Glede na hitro rastoči trg, DDR sistemi tedensko zagotavljajo nove datoteke, ki jih morajo vzdrževalci strani za zagotovitev stabilnega zaznavanja naprav redno prenašati.

Oblačna namestitev nudi dostop do repozitorija v oblaku, preko ključa API, ki ga pridobimo z ustvarjenim uporabniškim računom. Potrebna je namestitev oblačnega klienta WURFL, ki nato na osnovi storitve REST pošilja zahteve do repozitorija. REST je na kratko opisan v nadaljevanju. Pri tej namestitvi ni potrebno redno posodabljati lokalne zbirke podatkov, samo pošiljanje zahtev in prejemanje odgovorov pa zahteva zgolj nekaj vrstic kode.

2.5.2.1 Storitve REST

Spletne storitve REST temeljijo na protokolu HTTP in arhitekturnem stilu REST. Namenjene so izmenjavi podatkov od strežnika do odjemalca in delujejo neodvisno od platform. Storitve delujejo na konceptu izmenjave virov, ki so enostavni podatkovni objekti in se lahko prenašajo preko več formatov (npr. XML, JSON, navaden tekst in HTML). Komunikacija med odjemalcem in strežnikom poteka preko poslanih zahtev in vrnjenih odgovorov. Vsaka posamezna zahteva že vključuje vse podatke, ki jih strežnik potrebuje za izvedbo storitve. Odjemalec za pošiljanje zahtev uporablja ukaze HTTP GET, POST, PUT in DELETE, za identifikacijo posameznega vira na strežniku pa uporablja naslavljanje URI [22]. Odgovor ki ga odjemalec prejme, lahko za nadaljnjo uporabo shrani v predpomnilnik, ter tako skrajša čas prihodnje izmenjave podatkov.

2.5.3 Skupna uporaba

Implementaciji odzivnega spletnega oblikovanja na odjemalcu in strežniku nista izključujoči, temveč ju lahko po potrebi uporabimo tudi v skupni kombinaciji. Za definiranje enostavnih prelomnih točk strani lahko na strani odjemalca uporabimo JavaScript s katerim ugotovimo velikost zaslona ciljne naprave. Rezultat nato shranimo v piškotek [23]. Strežnik lahko na podlagi shranjenih podatkov nato izvede zahtevnejše obdelave nad podatki in jih v optimizirani obliki pošlje odjemalcu. Drug način kombinirane uporabe bi bila na primer uporaba ogrodja Modernizr in sistema DDR WURFL [24]. S prvim bi lahko ugotovili ali ciljna naprava podpira specifikacije CSS3, z drugim pa bi izvedeli kako zmogljiva je. Na podlagi obeh rezultatov bi nato določili obseg vsebine, ki jo bomo posredovali odjemalcu.

3 Pregled jezika SVG in načinov uporabe na spletu

SVG ali skalabilna vektorska grafika je odprti standard, ki ga uporabljamo za izdelavo dvodimenzionalnih vektorskih grafik. Omogoča nam izdelavo zelo kompleksnih grafik in profesionalnih načrtov, obenem pa je njegova struktura v osnovi zelo lahka za razumevanje, saj temelji na označevalnem jeziku XML.

Kmalu po širšem razponu spleta, se je pojavila želja po formatu za upodabljanje vektorske grafike na spletnih straneh. Leta 1996 je razvijalec Chris Lilley v okviru konzorcija svetovnega spleta izdal dokument W3C zahtev za skalabilno grafiko. Dve leti kasneje je konzorcij prejel že več predlogov za standardizacijo formata vektorske grafike, vendar so namesto sprejetja enega izmed teh ustanovili delovno skupino SVG, katere cilj je bil izdelati nov enovit odprti standard. Njihov produkt je bil jezik SVG, pri razvoju katerega so se opirali na ugotovitve iz predhodnih predložitev. Od teh je vredno omeniti predvsem VML (*Vector Markup Language*) in PGML (*Precision Graphics Markup Language*). Prvi jezik je v večjem delu zasnoval Microsoft, drugega pa Adobe [25].

Leta 2001 je SVG 1.0 postal priporočilo W3C, Adobe pa je zanj predstavil vstavek za brskalnike ASV 3 *(Adobe SVG Viewer*). Po tem se je začel standard razširjati preko spleta, vendar je pravi razpon doživel šele leta kasneje, ko so ga v svojo podporo začeli integrirati glavni spletni brskalniki. Zadnji od njih je bil Internet Explorer, ki je SVG podprl šele v različici 9 leta 2011. Eden ključnih faktorjev povečanja uporabe skalabilne vektorske grafike je bil tudi prihod HTML5, ki med svoje nove specifikacije vključuje tudi medvrstični SVG.

Trenutno aktivna različica standarda je SVG 1.1, obstajata pa tudi dve poenostavljeni različici, SVG Basic in SVG Tiny 1.2, ki sta bili razviti za uporabo na mobilnih napravah. V pripravi je že tretja različica SVG 2, katere glavni namen je dodatno izboljšati integracijo s HTML, CSS in modelom DOM. Predvideno naj bi specifikacija postala priporočilo W3C v avgustu 2014. Eden od glavnih snovalcev standarda SVG Jon Ferraiolo o nadaljnjem razvoju pravi [26]: » *W3C v kolaboraciji z delovnimi skupinami brskalnikov in organizacijo, generalizira veliko najboljših značilnosti SVG 1.0 (npr. striženje, animacijo in filtre) v CSS, tako da bodo te odlike na voljo tudi v HTML, ter čisti SVG, ki bo tako enostavnejši za uporabo (npr. odstranitev zahteve XML).* «

Največja alternativa SVG, ki se danes uporablja za izdelavo grafik na spletu je HTML5 Canvas, druge manj uporabljene možnosti pa so Flash, WebGl in VML [27]. Prednosti SVG v primerjavi s Canvas sta npr. resolucijska neodvisnost in podpora upravljanju dogodkov. Na drugi strani zna Canvas ustvarjene slike izvažati v formata PNG in JPEG, ter je primernejši za uporabo v igrah [28].

3.1 Sintaksa dokumentov SVG

Skalabilna vektorska grafika je v osnovi imensko področje jezika XML. SVG je zato lahko samostojen dokument, ali le del dokumenta XML v souporabi z drugimi imenskimi področji. Z uporabo specifikacije XSLT lahko na primer ustvarimo nov dokument SVG iz običajnega dokumenta XML, oziroma preoblikujemo obstoječo grafiko.

Dokument skalabilna vektorske grafike definiramo z elementom <*svg*>. Ta predstavlja vidno polje v kateri bomo ustvarili sliko. Glavne lastnosti risalne površine nadaljnje opredelimo z ustreznimi atributi, od katerih so najpogostejši opisani v tabeli 3.

Atribut	**Opis**
version	Uporabljena različica SVG.
x	Koordinata x. Definiramo jo samo pri elementih, ki so ugnezdeni znotraj nadrejenega elementa SVG.
x	Koordinata y. Definiramo jo samo pri elementih, ki so ugnezdeni znotraj nadrejenega elementa SVG.
width	Širina polja namenjenega grafiki, ki jo želimo ustvariti.
height	Višina polja namenjenega grafiki, ki jo želimo ustvariti.
viewBox	Opisuje dimenzije polja SVG s štirimi vrednostmi. Začetni koordinati x in y, ter širino in višino polja.
preserveAspectRatio	Običajno v souporabi z atributom *viewBox*. Določa način umerjanja grafike znotraj nadrejenega polja (npr. ohranimo originalno razmerje grafike ali grafiko razširimo samo po osi x ipd.).
contentScriptType	Določa privzeti skriptni jezik v dokumentu SVG. Če ga ne definiramo, se uporabi vrednost *application/ecmascript*.
contentStyleType	Določa privzeti slogovni jezik v dokumentu. Če ga ne določimo, se uveljavi vrednost *text/css*.

Tabela 3: Atributi elementa SVG.

Izhodiščni koordinati vidnega polja se običajno nahajata v zgornjem levem kotu. Vrednost x torej narašča od leve proti desni, vrednost y pa narašča od zgoraj navzdol. Znotraj elementa <*svg*> lahko ugnezdimo tudi istoimenske podrejene elemente s katerimi določimo novo vidno polje. To običajno storimo, ko želimo ustvariti sliko v sliki [29].

Ko imamo pripravljeno risalno površino, lahko začnemo vstavljati grafične elemente. SVG obsega delo s tremi objekti: vektorsko grafiko, rastrsko grafiko in tekstom. Ti so podrobneje opisani v naslednjem podpoglavju. Vsak grafični element ima pripadajoče atribute, ki jih W3C deli v dve skupini:

- **Splošni atributi** (*regular attributes*): določajo podatke za izris in obnašanje grafičnega elementa znotraj vidnega polja (npr. dolžina, širina, radij ali rotacija).
- **Predstavitveni atributi** (*presentation attributes*): opredelijo oblikovne lastnosti (npr. barva, poravnava teksta, prosojnost, velikost pisave ali oblika obrobe).

Grafike SVG so v osnovi sestavljene iz enostavnih geometričnih objektov, vendar lahko te med sabo združujemo v kompleksnejše objekte, ki so obravnavani kot celota. To nam omogočata elementa <*g*> in <*symbol*>. Prvega lahko definiramo kjer koli znotraj dokumenta SVG, ko želimo povezati skupino elementov. Element <*symbol*> na drugi strani uporabljamo za izdelavo grafičnih predlog, ki jih želimo v dokumentu prikazali večkrat. Vse predloge določene s <*symbol*> postavimo v nadrejeni element <*defs*>. Njegov namen je podrejene objekte definirati, ne pa tudi prikazati. Tako ustvarjeno predlogo nato uporabimo kjer koli v vidnem polju ter v poljubno mnogo kopijah. Sklic na predlogo izrazimo z elementom <*use*>. Ta poleg atributa *xlink:href,* v katerem navedemo referenco IRI (*Internationalized Resource Identifier*) na predlogo, podpira še pet splošnih atributov: koordinati *x* in *y*, širino (*width*) in višino (*height*) ter *transform*. Določimo lahko tudi dodatne predstavitvene atribute, vendar se ti uveljavijo le, če nima element iste lastnosti določene že v predlogi.

Dva posebna elementa SVG sta <*title*> in <*desc*>. Prvi je namenjen vnosu naslova posameznega objekta v grafiki in se lahko v samostojnih spletnih straneh SVG prikaže kot naslov zavihka. Običajno ga definiramo kot prvega v nadrejenem elementu. Element <*desc*> pa je namenjen daljšemu opisu objektov in ga navadno uporabimo takoj za <*title*> [30].

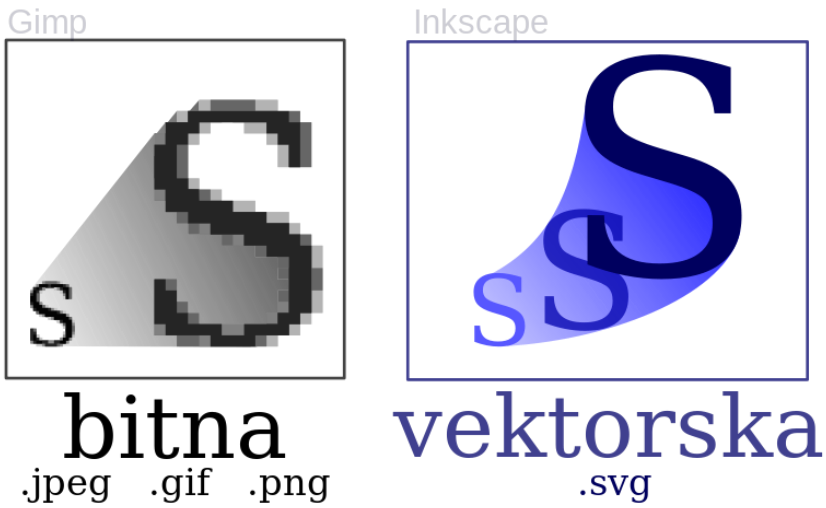
3.2 Grafični elementi SVG

Skalabilna vektorska grafika je, kot nakazuje že ime, primarno namenjena delu z vektorsko grafiko, vendar obenem omogoča tudi oblikovanje rastrskih ali bitnih grafik in besedila. Vsak od treh elementov je podrobneje opisan v nadaljevanju.

3.2.1 Vektorska grafika

Vektorska grafika je slikovni zapis kreiran z uporabo geometrijskih objektov, ki se izrisujejo na podlagi matematičnih formul. To dejstvo omogoča več prednosti pred bitno grafiko:

- Posamezen objekt je določen z lastnim izrazom, zato mu lahko v vsakem trenutku spremenimo lastnosti kot so pozicija, nivo prikaza, barva, velikost,… ne da bi pri tem neposredno vplivali na ostale objekte.
- Vektorska grafika ohranja enako kvaliteto tudi pri neskončnih (zelo velikih) povečavah, medtem ko bitne slike s povečavo izgubljajo ostrino. Primer povečave obeh vrst je prikazan na sliki 3.
- Velikost datotek je manjša, ker ni potrebno za vsako točko v grafiki definirati svojega barvnega zapisa.



Slika 3: Primerjava povečave vektorskih in bitnih grafik [31].

Enostavni geometrijski objekti, ki jih SVG uporablja kot samostojne elemente so opisani v tabeli 4.

Ime	**Opis elementa in glavnih atributov**
pravokotnik (rect)	Enostaven lik, ki mu običajno določimo koordinati x in y, ter širino in višino. Na voljo imamo tudi atributa *rx* in *ry* s katerima lahko zaoblimo kote pravokotnika.
krog (circle)	S *cx* in *cy* definiramo središče kroga, velikost radija pa določimo z atributom *r*.
elipsa (ellipse)	Tako kot pri krogu opredelimo središče s *cx* in *cy*, radij pa predstavimo z vrednostma *rx* in *ry*.
črta (line)	Ravna črta, ki ji določimo začetni in končni par koordinat x in y. Z atributom *stroke-dasharray* lahko kreiramo prekinjeno črto s poljubno dolgimi intervali.
poligon (polygon)	Definira lik s poljubno mnogo točkami, ki jih v zaporedju naštejemo z atributom *points*. Črte v liku se lahko tudi prekrivajo.
lomljenka (polyline)	Lomljena, poljubno dolga črta.
pot (path)	Zaporedje poljubno mnogo točk, ki je lahko zaključeno ali ne.

Tabela 4: Osnovni liki SVG.

3.2.2 Rastrska grafika

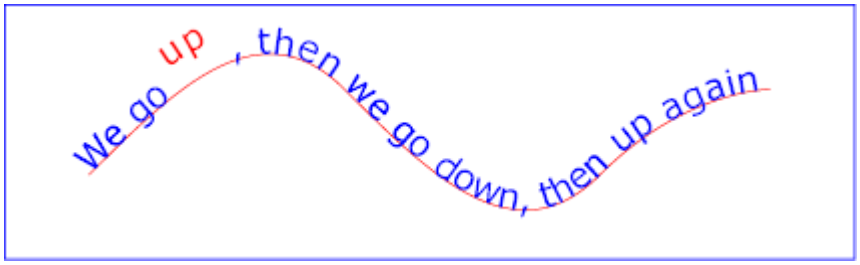
Bitne slike formatov PNG, JPEG ipd. lahko v SVG vključimo z uporabo elementa <*image*>. Vsaki sliki moramo definirati osnovne atribute o prikazovalnem polju, to so: začetni koordinati x in y, ter širina in višina prikazovalnega polja. Uporabimo lahko tudi atribut *preserveAspectRatio*, s katerim določimo v kakšnem razmerju želimo prikazati sliko, npr. v originalnem razmerju, ali v razmerju prikazovalnega polja.

Rasterskim slikam lahko z uporabo elementa <*filter*> dodamo različne grafične filtre, kot so: ostrenje, zameglitev, barvne transformacije, zaznava robov, itd. Na vsak objekt lahko dodamo več kombiniranih filtrov.

3.2.3 Besedilo

SVG za prikazovanje teksta uporablja element <*text*>. Za razliko od drugih osnovnih grafičnih elementov ima <*text*> vedno začetno in končno značko, med kateri vpišemo besedilo. Ker je besedilo zapisano kot znakovni podatek XML, ga lahko v brskalniku označimo, kopiramo ali poiščemo s kombinacijo tipk Ctrl+f. Poleg določanja osnovnih lastnosti besedila kot so npr. velikost pisave ali barva pisave, SVG omogoča tudi naprednejše oblikovanje. Pomembni so predvsem podrejeni elementi:

- **<tspan>**: Uporaba je podobna kot pri elementu <*span*> v HTML. Manjšemu delu besedila določimo drugačno obliko od ostalega teksta npr. odebeljeno, ležeče, nadpisano ali podpisano. Element je zelo uporaben, ko želimo prikazati besedilo v več vrsticah. Namesto da vsako vrstico pišemo v ločen <*text*> element, jo vstavimo v <*tspan*>, ki mu z atributom *y* pripišemo ustrezno večjo koordinato in tako simuliramo prehod v novo vrstico.
- **<tref>**: Namenjen je večkratni uporabi istega teksta preko naslavljanja IRI.
- **<textPath>**: Privzeto besedilo poteka v ravni črti od leve proti desni. Z elementom vzpostavimo povezavo med besedilom in obstoječim elementom <*path*>. Rezultat je besedilo, ki poteka po definirani poti. Enostavnejši primer je prikazan na sliki 4.



Slika 4: Teksti, ki poteka po definirani poti [32].

Dodatno ima tekst SVG tudi posebne zanimive atribute, s katerimi lahko npr. določimo orientacijo besedila, določimo teksturo s katero želimo napolniti znake ali določimo različne kote na podlagi katerih se vsak znak posebej rotira. SVG omogoča tudi izdelavo lastne pisave po meri z uporabo elementa <*font*>.

3.3 Napredna uporaba SVG

Standard SVG poleg kreiranja statičnih grafik, omogoča tudi izdelavo animacij, ki so na spletu zelo priljubljen način predstavitve podatkov. Prikazano vsebino lahko na ta način namreč naredimo bolj odzivno glede na uporabnikova dejanja ter enostavnejšo za razumevanje. Kljub temu veliko razvijalcev poudarja, da mora biti uporaba animacij omejena in se je moramo posluževati zgolj, ko želimo z njo izraziti dodaten pomen. Za izdelavo animacij v SVG poznamo tri načine: animacije SVG, uporaba skriptnega jezika, ali uporaba CSS.

3.3.1 Animacije SVG

Animacija SVG je najbolj osnoven način animacije in temelji na označevalnem jeziku SMIL (*Synchronized Multimedia Integration Language*). Poznamo pet elementov, s katerimi v imenskem področju SVG naznanimo animacijo. Vsak je na kratko predstavljen v tabeli 5.

Ime elementa	**Opis**
animate	Namenjen spreminjanju vrednosti posameznega atributa v določenem časovnem okviru.
set	Uporaben za vse vrste atributov in jim ob določenem pogoju definira novo vrednost.
animateMotion	Za elemente, ki jih želimo premikati na določeni poti. Pot definiramo z elementom <*path*>, ter se nato v animaciji sklicujemo nanjo.
animateColor	Element <*animate*> se je prvotno uporabljal samo za atribute z numeričnimi vrednostmi. Zato je bil za animacijo barv, ki imajo črkovno-številske vrednosti potreben ločen element. Danes se element opušča in nadomešča z <*animate*>.
animateTransform	Namen elementa je modifikacija vrednosti transformacijskih atributov (pozicija, rotacija in umerjanje).

Tabela 5: Elementi animacije SVG SMIL.

Element ustrezajoč tipu animacije, ki jo želimo implementirati, postavimo znotraj objekta, na katerem bomo animacijo izvedli. Za definiranje podrobnosti o dogajanju uporabimo različne atribute. Najprej povemo kateri atribut vsebovanega elementa bomo spreminjali, na primer širino pravokotnika. Nato povemo, kaj naj se z izbranim atributom zgodi, v kakšnem časovnem obdobju naj se sprememba izvede in ali se animacija ponavlja, ali izvede le enkrat. Na koncu običajno naznanimo še, kaj se zgodi z rezultatom izvedene animacije. Spremembe lahko ohranimo na zaslonu, ali pa objekt povrnemo v začetno stanje.

Animacija SVG omogoča veliko modifikacij obstoječih elementov, vendar ga v času pisanja te diplomske naloge iz podpore še vedno izključujeta Internet Explorer (tako osnovna kot mobilna verzija), ter Opera Mini [33]. Predvsem prvi predstavlja razlog, da se razvijalci velikokrat raje poslužujejo drugih načinov ustvarjanja animacij.

3.3.2 Skriptni jezik in SVG DOM

Skriptni jezik omogoča razvijalcem izdelavo naprednejših animacij, uporabnikom pa večjo interakcijo z grafikami. Poleg spreminjanja obstoječih elementov SVG lahko s skriptnim jezikom in souporabo SVG DOM v grafiko dodajamo nove, ali brišemo stare objekte. Uporabo skriptnega jezika napovemo z elementom <*script*>, čigar funkcionalnost je enaka istoimenskemu elementu v HTML. Z uporabo atributa *type* določimo še, kateri skriptni jezik bomo uporabili (npr. JavaScript). Če atributa ne definiramo, obvelja privzeta vrednost *application/ecmascript*. Izvajanje skripte za želeni element SVG sprožimo z uporabo dogodkovnih atributov. Večino dogodkov smemo uporabiti samo na določenih vrstah elementov SVG. W3C elemente ki jim lahko pripišemo dogodke, deli v naslednje skupine:

- **Grafični elementi** (*graphics element*): to so elementi, katerih rezultat je viden grafični objekt (npr. <*rect*>, <*line*> ali <*text*>).
- **Elementi z vsebino** (*container element*): sem spadajo elementi, ki lahko vsebujejo podrejene elemente (npr. <*defs*>, <*g*> ali <*svg*>).
- **Animacijski elementi** (*animation elements*): elementi ki jih uporabljamo za animacijo SVG (npr. <*animate*> ali <*animateMotion*>).

Atributi, ki jih uporabljamo za opis dogodkov, pa so razdeljeni kot:

- **SVGLoad dogodek**: tu gre samo za atribut *onload*. Uporabljamo ga lahko v večini elementov vseh treh zgoraj omenjenih skupin.
- **Grafični dogodki** (*graphical event attributes*): sproži jih uporabnik preko uporabniškega vmesnika (npr. *onclick*, *onmouseover*, *onfocusin*. Uporabni so za večino grafičnih elementov in elementov z vsebino.
- **Dogodki dokumenta** (*document event attributes*): opis stanja dokumenta SVG (npr. *onerror*, *onabort*, *onzoom*) in posledično na voljo samo za element <*svg*>.
- **Animacijski dogodki** (*animation events*): za tri tipične pojave pri animacijskih elementih: začetek (*onbegin*), konec (*onend*) in ponovitev (*onrepeat*) animacije.

Ob določenem dogodku nato z metodami SVG DOM spremenimo lastnosti grafičnih elementov, ter na ta način ustvarimo učinek animacije.

Tako kot pri jeziku XML lahko tudi dokumente SVG predstavimo z drevesno strukturo s pomočjo modela DOM. Ta omogoča popoln dostop do vseh elementov in njihovih atributov, ter dovoljuje napredno upravljanje dokumenta SVG, kot je dodajanje novih elementov ali spreminjanje in brisanje obstoječih elementov. V tabeli 6 je naštetih nekaj najpogostejših metod in lastnosti za upravljanje elementov.

Namen	**Ime**	**Opis**
Poizvedovanje o obstoječih podatkih	getElementById()	Metoda vrne element, ki ustreza identifikatorju v parametru.
	getElementByTagNameNS()	Vrne vsa vozlišča (*NodeList*), ki ustrezajo imenskemu področju in imenu v parametrih.
	getAttributeNS()	Vrne vrednost atributa, ki ustreza imenskemu področju in imenu v parametrih.
	parentNode	Vrne nadrejeno vozlišče.
Postavitev nove vrednosti atributa	setAttributeNS()	Doda nov atribut. Kot parametre podamo imensko področje ter ime in vrednost atributa.
Dodajanje novih elementov	createElementNS()	V podanem imenskem področju doda nov element.
	cloneNode()	Ustvari kopijo obstoječega vozlišča.
	appendChild()	Doda novo podrejeno vozlišče na konec trenutnega vozlišča.

Odstranitev elementa	removeChild()	Iz dokumenta odstrani podrejeno vozlišče, ki ustreza imenu v parametru.

Tabela 6: Najpogostejše metode in lastnosti DOM za upravljanje elementov.

3.3.3 Prehodi in animacije CSS

Zadnja različica kaskadnih slogovnih pol CSS3, je poleg drugih novosti predstavila tudi prehode (*transitions*) in animacije (*animations*). S prehodi lahko zgolj z uporabo CSS spremenimo slog, z animacijami pa določimo gibanje elementov SVG in HTML. Obe funkcionalnosti podpirajo vsi bolj znani brskalniki, vendar moramo za WebKit brskalnike pri vseh lastnostih vključiti še predpono -*webkit*- [34]. Prihod nove različice SVG2 obljublja še večjo integracijo s CSS, kot je trenutno mogoča.

3.4 Uporaba SVG na spletu

3.4.1 Uporaba orodij in optimizacija datotek

Skalabilne vektorske grafike lahko ustvarimo na več načinov. Eden od že omenjenih je izdelava dokumenta SVG v preprostem urejevalniku teksta. Ta možnost je pripravna za nezahtevne grafike, ali za manjše popravke obstoječih dokumentov. Pri zahtevnejših strukturah pa se običajno uporabljajo risarska orodja, ki znajo delati z vektorsko grafiko. Kot tri od najbolj poznanih, zmogljivih orodij lahko naštejemo:

- **Inkscape**: Odprtokodno orodje, ki podpira izvoz vektorskih grafik v širokem naboru formatov (nrp. Incscape SVG, optimiziran SVG, stisnjen SVG (.svgz), Postscript, PDF ipd.). Kot ena boljših prednosti pred drugimi orodji se navaja urejevalnik XML, ki omogoča strukturiran pogled direktno in urejanje grafičnih elementov [35]. Inkscape smo za izdelavo nekaterih grafik uporabili na našem praktičnem primeru.
- **Adobe Illustrator**: Licenčno orodje s privzetim izvozom v lastniški format AI. Ena njegovih od največjih prednosti pred Inkscapom je zelo zmogljiv modul za izdelavo mrež (*Gradient meshes*) in bolj sofisticiran sistem za upravljanje barv (*Pantone Matching System*) [36].
- **CorelDRAW**: Licenčno orodje zasnovano za operacijske sisteme Windows. Ponuja enako širok nabor funkcionalnosti kot Illustrator, veliko razvijalcev pa trdi, da je prijaznejši in hitrejši za uporabo. Privzeti format za izvoz grafik je lastniški CRD.

Omenjena orodja nam resda neprimerno pohitrijo izdelavo grafik, vendar je pred vključitvijo kreirane datoteke SVG vseeno priporočljivo preveriti, ali lahko kodo v dokumentu še dodatno optimiziramo. Pri podvajanju grafičnih elementov ali filtrov se včasih lahko zgodi, da ustvarimo kakšno kopijo preveč, vendar tega morda niti ne opazimo, saj je s sliko na videz vse v redu. Nadalje orodja sama v datoteko vključujejo tudi svoje metapodatke, privzete vrednosti, presledke in komentarje, ki za prikaz grafike na spletu niso relevantni. Za

avtomatsko optimizacijo kode obstaja več rešitev. V manjšem obsegu znajo to narediti že zgoraj opisana orodja. Če dobro poznamo sintakso SVG lahko kodo počistimo sami, uporabimo pa lahko tudi temu posebej namenjena orodja (to so npr. Scour, SVG Cleaner in SVGO) oziroma spletne strani. Dodatno lahko velikost dokumentov SVG zmanjšamo s stisnjenim formatom SVGZ.

3.4.2 Vključitev dokumentov SVG v spletno stran

SVG lahko v spletno stran vključimo na naslednje načine.

- **SVG kot samostojna spletna stran**: Za pravilno prikazovanje dokumentov s končnico SVG ali SVGZ moramo določiti vrsto MIME *image/svg+xml*. Povezavo do samostojne SVG strani lahko določimo tudi iz dokumenta HTML z uporabo elementa *<a>*.
- **Medvrstični SVG v HTML**: Skalabilne vektorske grafike lahko pri HTML5 vključimo direktno v telo dokumenta z elementom *<svg>*. Znotraj definiranega vidnega polja nato kreiramo želeno sliko. Ta način je primeren za enostavnejše grafike.
- **Referenca iz dokumenta HTML**: Na obstoječe datoteke SVG se lahko sklicujemo z uporabo elementov [28,37]:
    - **<img>**: Uporaben za prikaz končnih grafik SVG, ki jih ne želimo dodatno spreminjati s selektorji CCS ali skriptnim jezikom.
    - **<object>**: Izberemo ga, ko želimo grafiko nadalje urejati s skriptnim jezikom ali selektorji CSS. Je najbolj priporočan način za prikaz zunanjih datotek SVG s strani različnih virov.
    - **<embed>**: Namenjen vključevanju interaktivne vsebine. Predstavljen je bil kot nov element v HTML5, čeprav je bil v uporabi že veliko prej.
    - **<iframe>**: Element se uporablja za vključitev drugih dokumentov v HTML, zato lahko z njim vključimo tudi datoteke SVG. Privzeto prikaže vsebino znotraj manjšega okna.
- **CSS background-image**: Ko želimo grafiko uporabiti za ozadje določenega elementa spletne strani.

3.4.3 Knjižnice za delo s SVG

Namen knjižnic SVG je poenostaviti upravljanje z vektorskimi grafikami. Izbiramo lahko med raznimi knjižnicami, ki skalabilne vektorsko grafiko podpirajo v večjem ali manjšem obsegu. V nadaljevanju so opisane tri zmogljive odprtokodne rešitve JavaScript:

- **D3** (*Data-Driven Documents*): Kot že ime nakazuje je knjižnica D3 namenjena delu s podatkovnimi dokumenti, njen primarni namen pa je vizualizacija želenih podatkov.

Razvita je bila na podlagi starejše knjižnice Protovis in temelji na uporabi standardov HTML, SVG in CSS. Sintakso D3 bi lahko opisali kot združitev jezikov JavaScript in DOM. Omogoča nam dodajanje in brisanje elementov dokumenta ter tudi zelo kompleksno modifikacijo obstoječih podatkov z uporabo velikega nabora vgrajenih funkcij [38]. Knjižnico D3 smo uporabili pri izdelavi praktičnega primera, zato je podrobneje opisana v četrtem poglavju.

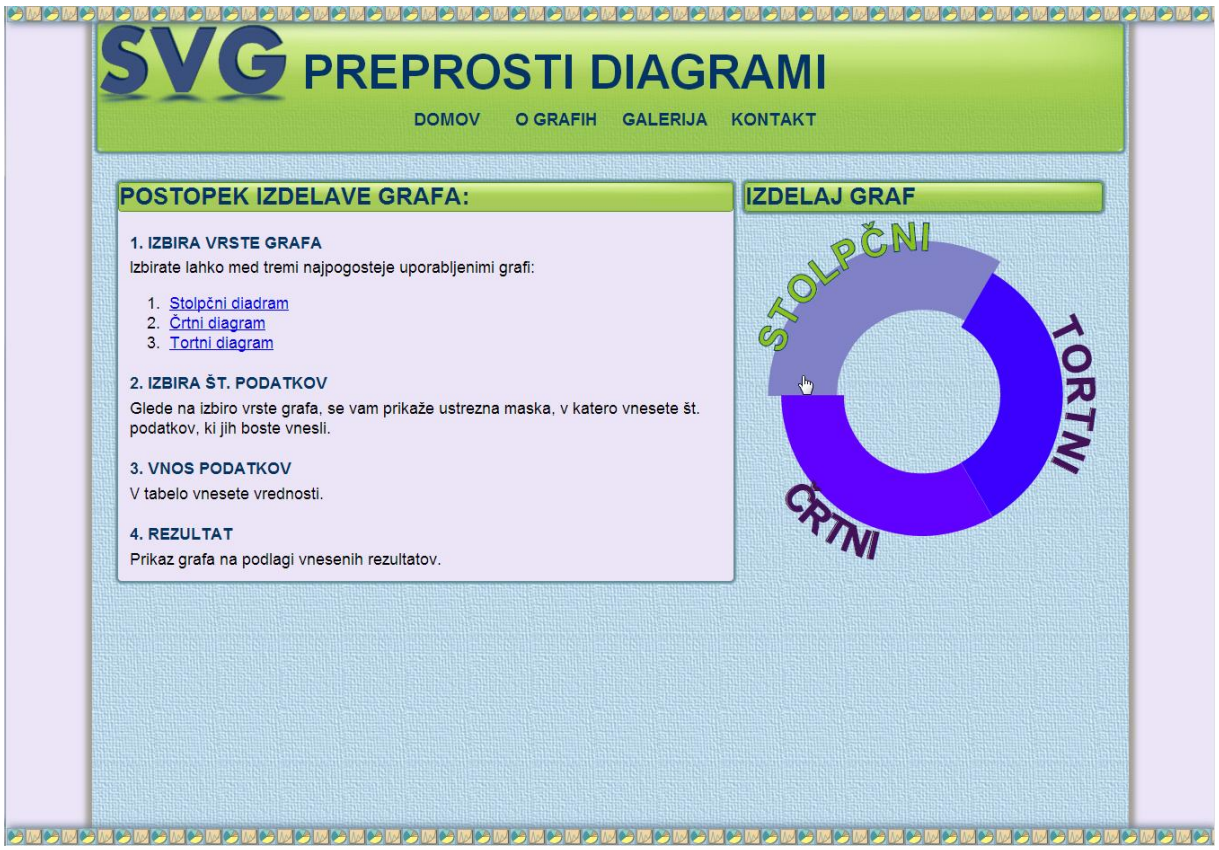
- **Raphaël**: Zaradi lahko priučljive sintakse je zelo priljubljena manjša knjižnica za prikaz vektorskih grafik. Uporablja standarda SVG in VML, slednjega primarno zaradi združljivosti s starejšimi različicami brskalnikov, ki ne podpirajo SVG [39].
- **Snap.svg**: Nova knjižnica za delo s SVG. Zasnoval jo je Adobeov razvijalec Dimitry Baranovskiy, ki je obenem tudi avtor Raphaël.js. Kot največji prednosti knjižnice sta poudarjeni možnost vključitve obstoječih zunanjih datotek SVG, ter podpora animaciji. Primeri uporabe na spletu so za enkrat še redki, vendar dober začetek za delo ponuja domača stran, kjer je predstavljenih nekaj zanimivih demonstracij animiranih grafik [40].

4 Predstavitev spletne strani

Uporabo jezika SVG na odzivnih spletnih straneh smo si zadali predstaviti s praktičnim primerom. Že na začetku smo omenili, da so diagrami zelo koristno pomagalo pri predstavitvi večje količine informacij. Zato smo se odločili, da bo ključna naloga naše spletne strani izdelava SVG diagramov. Ker je vseh vrst grafov zelo veliko, smo se omejili na tri bolj pogoste: stolpčni, črtni in tortni diagram.

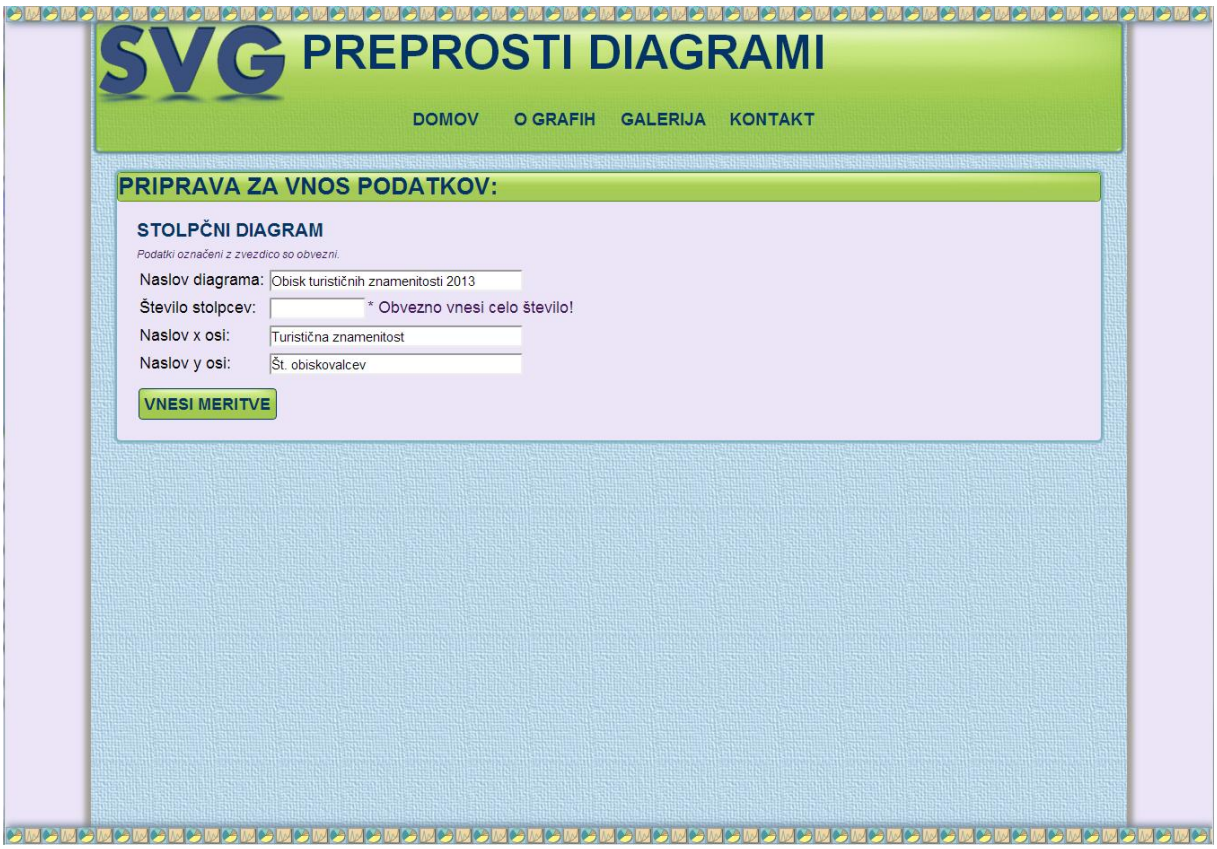
Začetna stran spletne strani vsebuje naslovni in glavni del. Na vrhu naslovne strani se nahajata logotip in naslov, pod njima pa je menijska vrstica. Prva povezava v menijski vrstici se imenuje Domov, njen namen pa je premik na začetno stran. Ostale povezave so: O grafih, Galerija, ter Kontakt. Galerija je namenjena slikovni predstavitvi različnih diagramov ter možnosti oblikovanja, ki jih lahko ustvarimo. Kontakt je zgolj informativnega namena.

Glavni del začetne strani je razdeljen na dva dela: navodila za izdelavo ter izbiro diagrama. Navodila so razdeljena v štiri točke in uporabnika informirajo kako bo potekala izdelava diagrama. Prva točka navodil predstavi tri možne vrste grafov, ki jih uporabnik lahko izbere in obenem omogoča povezavo na pregled dodatnega opisa funkcionalnosti vsakega od grafov. Na isto stran lahko pridemo tudi prek menijske vrstice z izbiro povezave O grafih. Izbira diagramov je prikazana kot obroč razdeljen na tri dele, ki predstavljajo vsak svojo vrsto diagrama. Ob postavitvi na določen odsek, se ta za jasnejši odziv poveča, ter obarva pisavo tako kot je prikazano na sliki 5.



Slika 5: Začetna stran z izbiro diagrama.

Ko uporabnik izbere želeni graf, se prikaže prva vnosna stran. Vsi možni podatki za vnos so naslov diagrama, št. podatkov, ter naslov x in naslov y osi. Slednja se pri tortnem diagramu ne pokažeta, saj ta nima osi. Edini obvezen podatek je št. podatkov, za katerega se ob potrditvi vnosa preverja, ali je celo število. Če ni, se ob vnosnem polju prikaže opozorilo, kot je vidno na sliki 6, sicer se premaknemo na naslednjo stran.

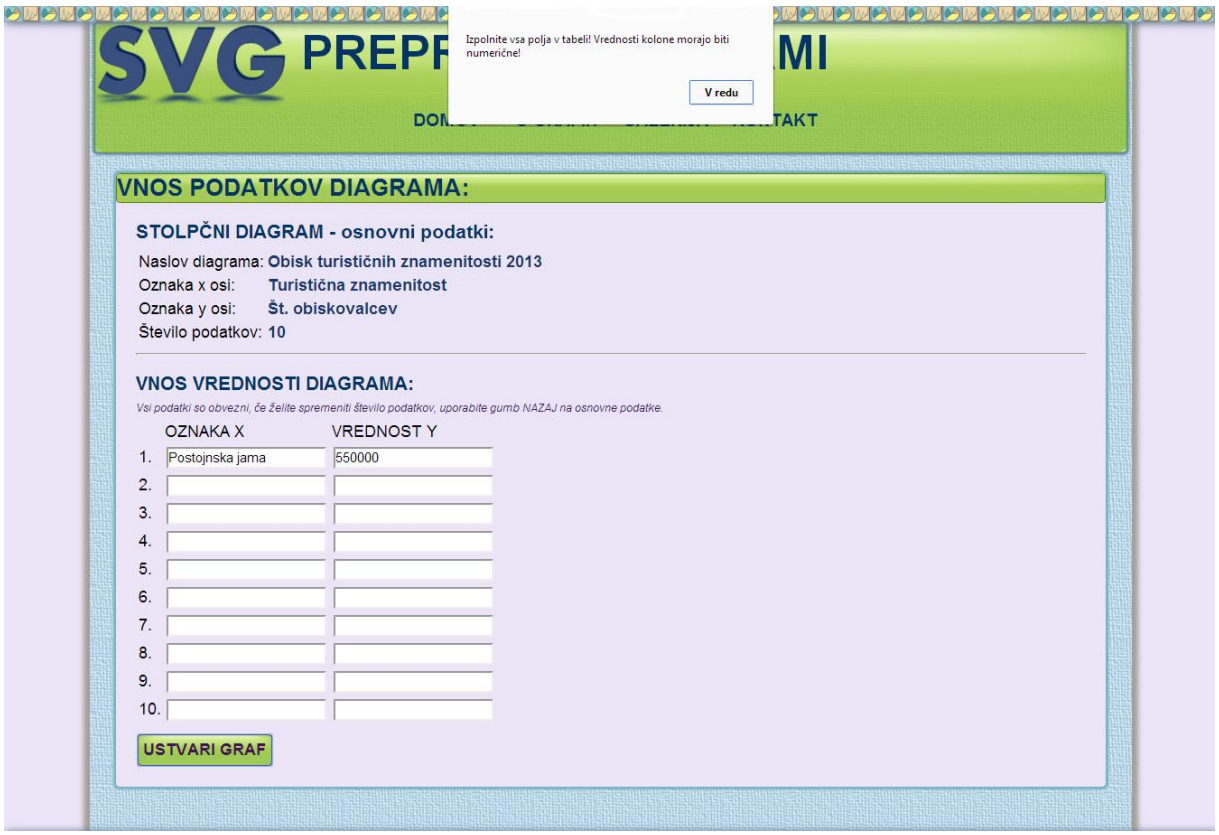


Slika 6: Prva vnosna stran podatkov, z opozorilom pri napačnem vnosu.

Druga vnosna stran na vrhu prikaže že vpisane podatke, v spodnjem delu pa je tabela za vnos vrednosti diagrama. Število vrstic tabele je enako vpisanemu številu podatkov na prvi vnosni strani. Sledi uporabnikov vnos, po katerem klikne na gumb USTVARI GRAF. V ozadju se preverijo vpisani podatki v naslednjih korakih:

- Preveri ali so vnesene vse vrednosti.
- Preglej če so vrednosti drugega stolpca numerične.
- Če je izbran diagram črtni, preveri tudi ali so vrednosti prvega stolpca numerične.

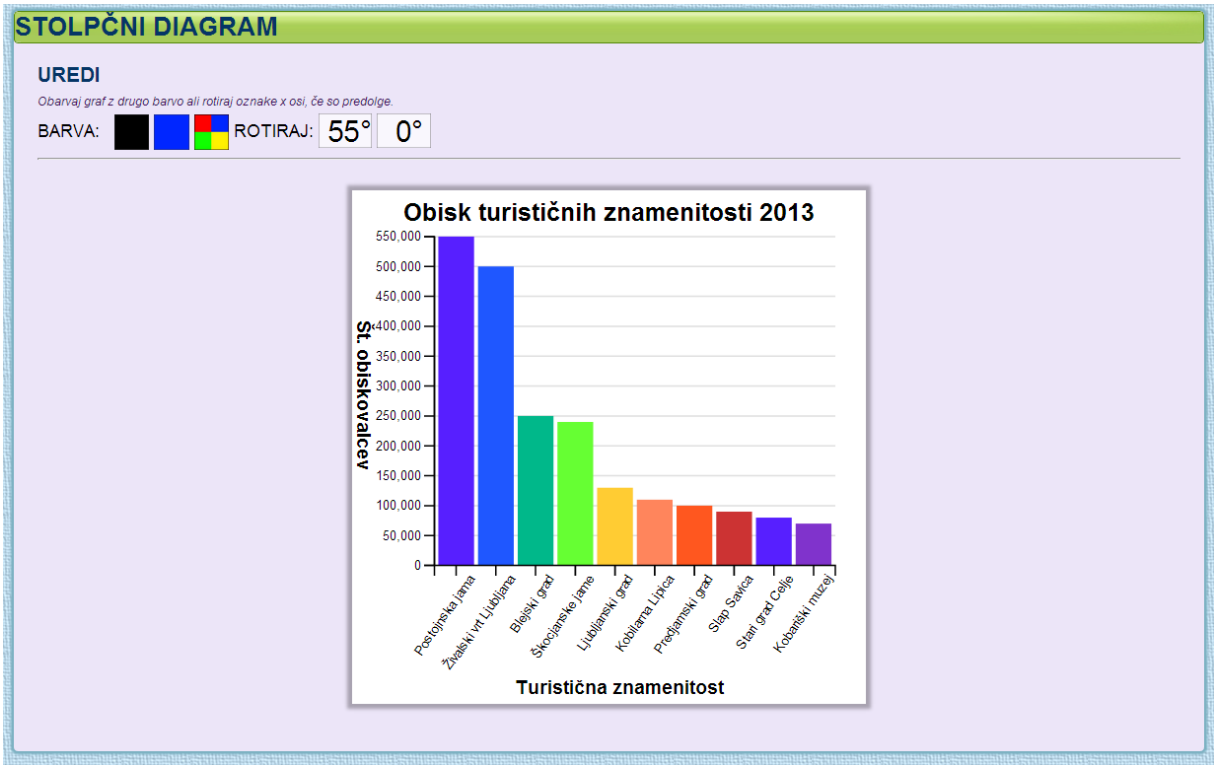
Če kateri od pogojev ni izpolnjen, se izpiše pojavno okno z opozorilom uporabniku, kot je prikazano na sliki 7. Na vnosni strani ostanemo dokler niso izpolnjeni vsi pogoji.



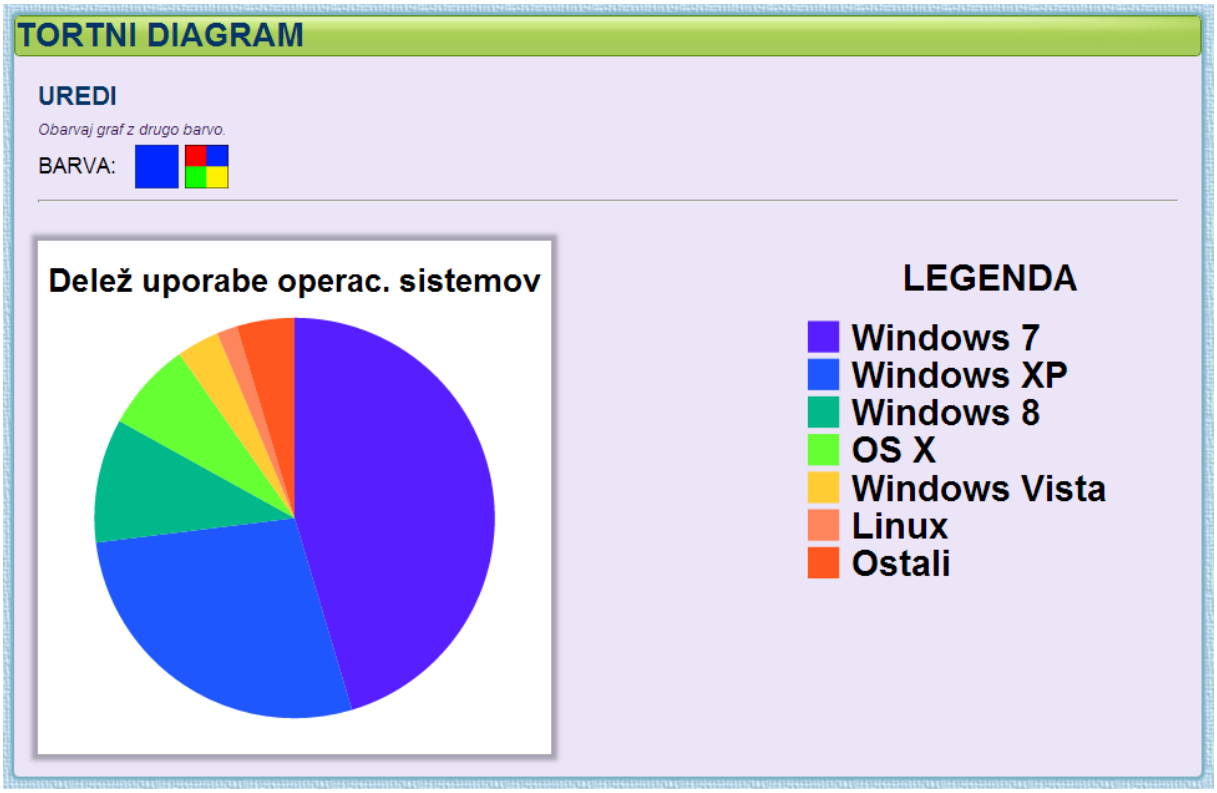
Slika 7: Druga vnosna maska podatkov, z opozorilom pri napačnem vnosu.

Zadnji korak je izdelan diagram, čigar prikaz se nekoliko razlikuje pri vsaki od treh različnih vrst:

- Stolpčni diagram poleg oznak na x in y osi vsebuje tudi rahlo mrežo, ki poteka vzporedno z y osjo. Njen namen je lažje določiti točno višino posameznega stolpca. Nadaljnje imamo nad diagramom dve dodatni možnosti urejanja. Prva je izbira barve, kjer imamo na voljo črno, modro ali večbarvno kombinacijo. Črna je obenem tudi privzeta vrednost. Druga opcija urejanja je rotiranje oznak x osi. Lahko se namreč zgodi, da uporabnik vnese tekst, ki je daljši kot širina stolpcev diagrama in se zato prekriva z drugimi oznakami. S klikom na gumb se oznake rotirajo za 55°, naslov x osi pa se pomakne nižje. Primer ustvarjenega stolpčnega diagrama prikazuje slika 8.
- Črtni diagram ima prav tako x in y os, ter mrežo ki poteka premočrtno z obema osema. Tako kot pri stolpčnem diagramu imamo možnost izbire treh barv, vendar večbarvno kombinacijo zamenja rdeča barva. Izbrana barva se uveljavi tako za črto, kot tudi za točke. Omogočena je tudi rotacija x oznak.
- Tortni diagram nima osi, ima pa legendo. Pozicija legende je odvisna od velikosti okna. Če uporabljamo napravo z večjim zaslonom, potem se prikaže desno od grafa, kot je prikazano na sliki 9. Na manjših zaslonih se prikaže pod grafom. Kot dodatno možnost imamo izbiro dveh barv v modri ali raznobarvni kombinaciji. Modra barva je primerna bolj na grafih z malo podatki. Pri veliko podatkih postane nepregledna.



Slika 8: Primer stolpčnega diagrama z rotiranimi oznakami osi x.



Slika 9: Primer tortnega diagrama z legendo.

5 Razvoj spletne strani

Izdelavo aplikacije smo se odločili implementirati z uporabo odprtokodnega razvojnega okolja Eclipse Kepler (4.3.1) za razvijalce Java EE. Predhodno smo si morali namestiti programski paket Java Development Kit 7. Za aplikacijski strežnik smo uporabili odprtokodni GlassFish 4.0, ki smo ga integrirali z okoljem Eclipse. Skalabilne vektorske grafike smo osnovali s specifikacijo SVG 1.1, saj ta za razliko od okrnjenih različic zajema celoten nabor funkcionalnosti. Podrobnosti jezika SVG smo že predstavili v drugem poglavju, zato nadaljnji opis ni potreben.

Poglavitne spletne tehnologije, ki smo jih potrebovali za izdelavo spletne strani so jezik HTML, kaskadne slogovne pole CSS, jezik JavaScript in JavaServer Pages (JSP).

Za izdelavo nekaterih vektorskih grafik (npr. logotipa, ozadja naslovne vrstice, gumbov za urejanje grafa) smo uporabili orodje Inkscape, ki smo ga že omenili v drugem poglavju. Kodo izdelanih grafik SVG smo shranili v optimizirani obliki, vendar smo pri pregledu v urejevalniku ugotovili, da tudi ta format vsebuje še veliko nepotrebnih elementov. To so bili npr. metapodatki, nepotrebna imenska področja definirana na začetku datoteke ter nekatere oblikovne lastnosti likov, ki za pravilen prikaz niso bili potrebni. Zato smo se najprej odločili preizkusiti optimizacijo grafik na spletni strani Peter Collingridgea [41], vendar smo ugotovili, da postopek zgolj odstrani identifikacije elementov in nepotreben prazen prostor. Na koncu smo se odločili še za ročno optimizacijo. Rezultati zmanjšanja velikosti datoteke pri vsaki uporabljeni datoteki, je predstavljen v tabeli 7.

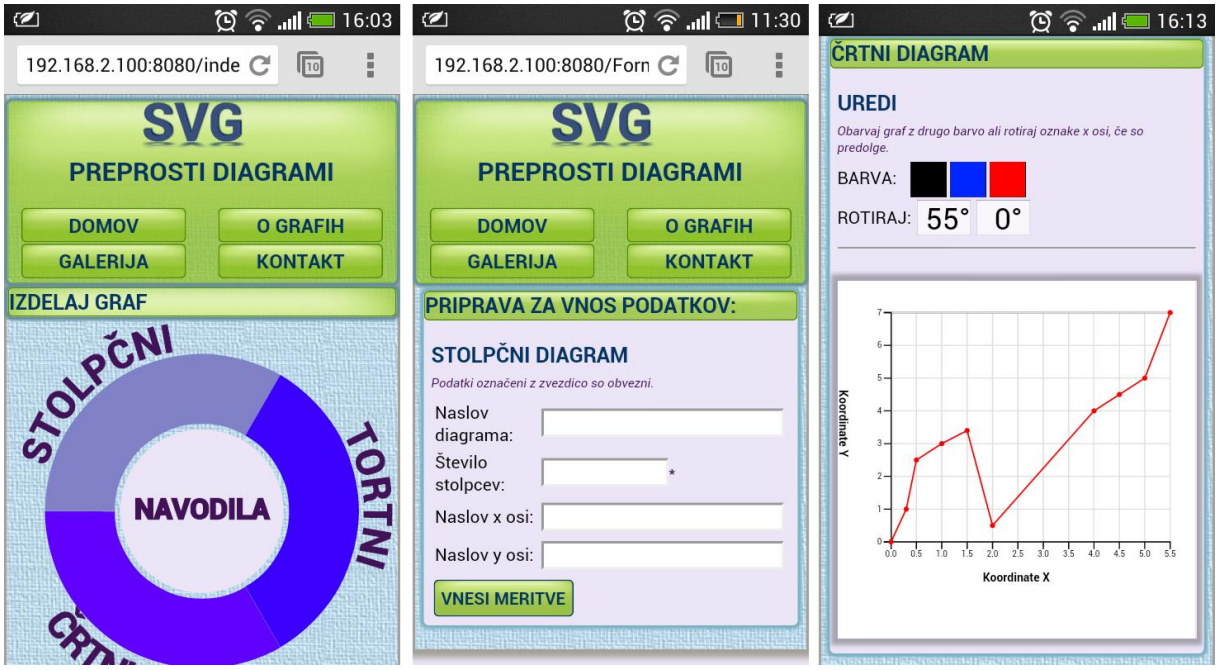
Optimizacija SVG	**Velikost datoteke (KB)**
Brez optimizacije	5,26
Inkscape optimizirani SVG	3,53
Prek spletne strani	3,37
Ročno	2,35

Tabela 7: Spreminjanje velikosti datoteke SVG pri optimizaciji.

5.1 Odzivnost spletne strani

Čeprav veliko razvijalcev odzivnih spletnih strani poudarja prednosti oblikovanja s prioriteto mobilnosti, smo začetno stran v našem praktičnem primeru najprej osnovali za velik namizni zaslon. Razlog za to odločitev je, da se nam je zdelo bolj smiselno začeti z masko, ki bo optimalna za monitor priklopljen na računalnik s katerim smo razvijali spletno stran. Za tem ko smo v osnovi izoblikovali primerno masko, smo s pristopom postopne degradacije določili velikostne razrede, v katerih naj se postavitev elementov spreminja. Po preizkušanju več kombinacij razvrstitve, smo se na koncu odločili za uporabo petih razredov. Prvi trije razredi vsebujejo večje spremembe prikaza elementov, zadnja dva pa sta zgolj razširitev tretjega razreda, ki popravita relativno velikost nekaterih elementov. Primer postavitve vsebine na

najmanjšem razredu je prikazan na sliki 10. Podrobnejše lastnosti vsakega razreda so predstavljene v tabeli 8.



Slika 10: Postavitev vsebine na najmanjših zaslonih.

Širina zaslona	**Opis postavitve**
do 35 em	Najmanjši zasloni. Povezave menija so oblikovane kot gumbi. Vsi okvirji razdelkov so minimalizirani. Pisava je maksimalno pomanjšana. Izbira grafa na začetni strani, se pojavi takoj za naslovno vrstico. Poleg izbire grafa, se na sredini pojavi še povezava do navodil za izdelavo, ki na manjših zaslonih niso vidna. Gumbi za oblikovanje končnega diagrama so v dveh vrsticah. V galeriji je prikazana ena slika čez celo širino zaslona.
od 35 do 55 em	Majhni zasloni mobilnih naprav (npr. tablični računalnik v pokončni postavitvi). Izbira grafa je še vedno na vrhu, ne prikazujemo pa več gumba do navodil, saj so ta že vidna na začetnem zaslonu. Pisava je nekoliko povečana. V galeriji sta prikazani po dve sliki v eni vrstici.
od 55 em naprej	Za srednje velike zaslone. Pisava naslovov je še nadaljnje povečana. Navodila za izdelavo in izbira grafa sta poravnana na isto višino. Gumbi za oblikovanje končnega diagrama so prikazani v eni vrstici. V galeriji so prikazane tri slike v eni vrstici.
od 66 em naprej	Večji zasloni. Povečana sta levi in desni rob od glavnega dela spletne strani. Polje za prikaz končnega diagrama je nekoliko pomanjšano.
od 96 em naprej	Dodatno povečani robovi ter pomanjšana grafika za izbiro grafa in polje končnega diagrama, saj sta bili sicer preveliki.

Tabela 8: Velikostni razredi v medijskih poizvedbah na spletni strani.

Med izvedbo postopne degradacije oblike strani smo ugotovili, da bi bilo bolje začeti razvoj v obratni smeri. Obratno sorazmerno z manjšanjem prikazovalnega polja, se je večalo število lastnosti, ki smo jih v osnovi napačno zastavili (npr. velikost različnih pisav, ter velikost gumbov). Pojavili so se tudi primeri, ko za nekatere lastnosti sploh nismo predvideli, da bodo

potrebne. Tu lahko kot dva primera izpostavimo izničenje nekaterih privzetih robov elementov, saj so na manjših zaslonih zavzeli preveč prostora, ter določitev enolične identifikacije za vsako povezavo v menijski vrstici. Slednje smo potrebovali, ko vseh elementov nismo morali več prikazati v eni vrstici in smo jih morali urejeno prelomiti v dve vrstici, ter za posamezen element določiti, kam se postavi. Na podlagi teh ugotovitev, smo strukturo datoteke CSS predelali tako, da so si velikostni razredi sledili od najmanjšega proti največjemu, ter se pri nadaljnjem razvoju skušali v celoti držati načel progresivnega nadgrajevanja, ki ga zapoveduje prioriteta mobilnosti.

Prvotne velikostne razrede smo določili na podlagi večanja in manjšanja okna brskalnika na namiznem računalniku. V realnosti pa imajo sodobne mobilne naprave lahko zelo višjo resolucijo, kot enako veliko okno namiznega brskalnika. Na visoko ločljivih zaslonih pametnih telefonov, bi tako dobili postavitev elementov, ki je bila idejno zasnovana za precej širši zaslon in ne bi bila funkcionalna. Da bi se temu izognili, smo imeli na voljo dve izbiri. Prva možnost bi bila definiranje ločenih medijskih poizvedb za mobilne naprave. Druga enostavnejša rešitev, ki smo jo aplicirali na naši spletni strani, pa je določitev atributa *viewbox* v elementu <*meta*> vseh datotek JSP. Na ta način smo specificirali, naj se širina celotne vsebine spletne strani umeri v širino ciljne naprave. Kot je vidno na sliki 11 smo začetni koordinati x in y postavili na 0, višini in širini pa smo dali enaki vrednosti kot v samostojnima atributoma. Dodatno smo uporabili atribut *preserveAspectRatio*, s katerim smo določili, naj se slika vedno umerja v središče vidnega polja.

```
1 <svg xmlns="http://www.w3.org/2000/svg"
2     id="svg2985"
3     viewbox="0 0 190.92 86.01"
4     height="86"
5     width="190.9"
6     version="1.1"
7     preserveAspectRatio="xMidYMin meet">
8
```

Slika 11: Nastavitev atributov SVG elementa za pravilno umerjanje.

Na največjo omejitev odzivnega oblikovanja strani smo naleteli pri prikazu oznak podatkov v stolpčnem in tortnem diagramu. Pri prvem je problem predstavljala širina, ki smo jo določili vsakemu stolpcu. Ker se polje za prikaz diagrama po širini vedno v celoti umeri na prikazovalno polje ima vsak podatek precej omejen prostor in je zato pri daljših oznakah stolpcev hitro prišlo do prekrivanja z drugimi oznakami. Kot rešitev smo uporabili izbirno rotacijo oznak na osi x, ki smo jo opisali v četrtem poglavju. Pri tortnem diagramu smo se prav tako znašli pred dilemo, kako predstaviti opise odsekov. Drugod so ti velikokrat postavljeni okrog diagrama, poleg odseka, ki mu pripadajo, vendar je bila ta opcija za manjše zaslone neuporabna. Zato smo sklenili opise postaviti v ločeno legendo, kot je opisano v specifikacijah spletne strani v četrtem poglavju.

5.2 Knjižnica D3

Za izdelavo diagramov smo uporabili JavaScript knjižnico D3, ki nam je zelo olajšala delo. Osnovni namen D3, ki smo ga predstavili že v drugem poglavju, je vizualizacija podatkov. Knjižnico smo v tej diplomski nalogi uporabili za manipulacijo elementov SVG, vendar bi jo po potrebi prav tako lahko izkoristili za upravljanje navadnih elementov HTML ali Canvas. V podpoglavjih so detajlno opisane funkcionalnosti D3, ki so nam najbolj pomagale pri kreiranju diagramov SVG.

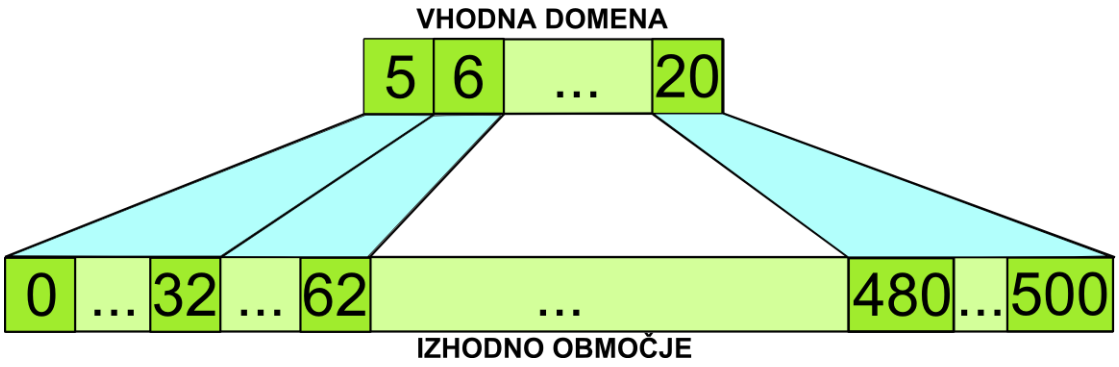
5.2.1 Zajem vhodnih podatkov

Knjižnica D3 omogoča zajem podatkov iz različnih zunanjih dokumentov (npr. CSV, TSV, JSON, XML, HTML ali iz navadnih tekstovnih datotek), kar je uporabno pri večjih količinah podatkov [42]. Za naše potrebe smo pri izdelavi diagrama potrebovali zgolj dva seznama podatkov, ki sta vsebovala podatek o vrednosti na osi x in y pri stolpčnem in črtnem diagramu, ter ime in delež odseka pri tortnem diagramu. Poleg obeh seznamov vrednosti v diagramu, smo iz spremenljivk JSP prenesli tudi naslove, da smo jih lahko vstavili v končno grafiko SVG.

5.2.2 Umerjanje v podano območje

Poseben izziv je predstavljala razporeditev podatkov v polju SVG. Ker smo želeli, da se velikost polja prilagaja velikosti zaslona smo ga namesto z atributoma *width* in *height* morali opredeliti z atributom *viewBox*, kjer smo širino in višino določili kot sto odstotni. Vse podatke smo nato v polje umestili glede na to razmerje.

Ker spletna stran dovoljuje vpis poljubnega števila podatkov, smo morali pred kreiranjem diagrama izračunati, kakšne vrednosti bo posamezen podatek zasedel. Pri stolpčnem diagramu sta to bili širina in višina posameznega stolpca, pri tortnem diagramu pa dolžina posameznega loka. Pri izračunu smo si pomagali s funkcijami D3 za umerjanje *d3.scale*, ki se ločijo na količinske (*quantitative scales*), vrstilne (*ordinal scales*) in časovne (time scales). Njihov namen je vhodni nabor podatkov (*domain*) smiselno razporediti na izhodno območje (*range*) [43]. Kot primer vzemimo nek seznam, ki zajema numerične vrednosti od 5 do 20. Za prikaz teh podatkov imamo na voljo prostor širok 500 slikovnih točk. Funkcija *d3.scale.linear()* na podlagi podanih minimalnih in maksimalnih vrednosti domene ter izhodnega območja izračuna preslikavo podatkov, kot je ponazorjeno na sliki 12. Prikaz uporabe funkcije pri črtnem diagramu, kjer smo morali izračunati umerjanje na osi x in y, je prikazan na sliki 13.



Slika 12: Ponazoritev umerjanja vhodne domene vrednosti v izhodno območje.

```
/* --- umerjanje x in y vrednosti --- */

// ustvarjen objekt shranimo v spremenljivko xMere
xMere = d3.scale.linear()
          // vhodna domena
          .domain([d3.min(x_os,function(d){return d;}), //minimum x
                   d3.max(x_os,function(d){return d;})]) //maximum x
          // izhodno območje
          .range([0,n_sirina]); // razpoložljiva širina za izris grafa

// ponovimo za y os
yMere = d3.scale.linear()
          .domain([d3.min(y_os,function(d){return d;}),
                   d3.max(y_os,function(d){return d;})])
          .range([n_visina,0]);
```

Slika 13: Umerjanje vnesenih vrednosti za izdelavo diagrama.

5.2.3 Vstavljanje elementov

Po pripravi umerjanja podatkov, je bilo potrebno izdelati glavni del diagrama. Za vsako vrsto grafa smo uporabili drugačen pristop. Najprej bomo predstavili osnovni postopek izdelave elementov SVG, v nadaljevanju pa bolj pomembne funkcije D3, ki so nam pomagale pri zahtevnejših delih.

Osnovno zaporedje po katerem dodamo nov element lahko opišemo v treh korakih. Najprej poiščemo element, znotraj katerega želimo ustvariti novega. Nato mu dodamo posamezen element ali seznam izbranih vrednosti. Na koncu moramo določiti še vsaj najosnovnejše atribute, ki definirajo posamezen element. Operatorji ki jih pri tem uporabimo, so opisani v tabeli 9.

Operator	Parametri	Opis
select	Selektor CSS ali vozlišče	Poišče prvi element v dokumentu, ki ustreza pogoju definiranem v parametru [44].
selectAll	Selektor CSS ali vozlišče	Poišče vse elemente v dokumentu, ki ustrezajo pogoju v parametru.
append	Ime elementa, ki ga dodajamo	Izbrani selekciji elementov na konec pripne element definiran v parametru. Ime elementa je lahko določimo z znakovnim nizom ali

		s funkcijo, ki vrne element v DOM.
data	Seznam vrednosti	Seznam vrednosti v parametru poveže s selekcijo elementov. Uporabimo ga, ko želimo ustvariti več elementov naenkrat.
enter	Brez	Vrne vhodno selekcijo. Če iskana vozlišča še ne obstajajo, vrne kazalec na element, v katerega se lahko dodajo.
attr	Ime in vrednost atributa	Definira posamezen atribut elementa. Vrednost atributa je lahko konstantna ali izračunana prek funkcije. Konstantne vrednosti smo večinoma določali samostojnim elementom, izračunane vrednosti pa posameznim podatkom v diagramu, ki so odvisni od celotnega seznama.

Tabela 9: Operatorji D3 za vstavljanje novih elementov.

Na sliki 14 je prikazano dodajanje dveh vrst podatkov. Prvi je samostojni element za naslov diagrama, ki ga preprosto vstavimo v korenski element <*svg*> s funkcijo *append*. Drugi primer pa ponazarja dodajanje seznama podatkov, ali konkretno vstavljanje pravokotnikov, ki bodo predstavljali stolpčni diagram. Iz primera je razvidno, da s selektorjem najprej poiščemo vse elemente <*rect*> v elementu <*g*> z identifikacijo *n_graf*. Nato selekcijo povežemo s seznamom vnesenih vrednosti, ki je shranjen v spremenljivki *y_os*, ter šele nato s funkcijo *append* za vsak podatek ustvarimo svoj stolpec. Na koncu pri nastavljanju atributov pokličemo ustrezne funkcije, ki vsakemu pravokotniku določijo lastno pozicijo.

```
147 /* --- naslov grafa --- */
148 svg.append("text")
149    .attr("x",z_sirina/2)
150    .attr("y",6)
151    .attr("width",z_sirina)
152    .attr("text-anchor","middle")
153    .text("<%=naslov%>")
154    .attr("id","g_naslov");
155
156 /* --- pozicije pravokotnikov stolpčnega diagrama --- */
157 d3.select("#n_graf")
158   .selectAll("rect")
159   .data(y_os)              // y_os je seznam vnesenih vrednosti
160   .enter().append("rect") // za vsako vrednost dodamo svoj pravokotnik
161   .attr("x", function(d,i){return xMere(x_os[i])+z_razmak+4;})
162   .attr("y", function(d){return z_razmak+yMere(d)-2;})
163   .attr("width",function(d){return xMere.rangeBand();})
164   .attr("height", function(d){return n_visina-yMere(d);} );
```

Slika 14: Dodajanje novih elementov v dokument SVG.

5.2.3.1 Odseki tortnega diagrama

SVG vse krivulje definira z elementom <*path*>, zato je to tudi glavni gradnik tortnega diagrama. Vsak kos v torti je predstavljen s svojo potjo, ki je definirana z enajstimi vrednostmi. Prvi dve vrednosti sta koordinati začetne točke na krožnici, kamor se postavimo z

ukazom *M*. Nato z ukazom *A* najavimo risanje loka. Naslednji vrednosti povesta radija x in y kroga, katerega del je odsek. Za tem določimo kot, za katerega želimo rotirati lok, ter velikost in smer rotacije. Za tortni diagram so vse tri vrednosti konstantne. Ker želimo osnovno krožnico je kot rotacije loka 0. Izrisali bomo najmanjšo možno velikost krožnice za podane koordinate, zato je tudi ta 0. Rotacija bo v smeri urinega kazalca in jo določimo z vrednostjo 1. Zadnji dve vrednosti znotraj ukaza *A* sta končni koordinati x in y na krožnici. Z ukazom *L* nadaljujemo z risanjem odseka v središče kroga, ki ima vrednosti koordinat x in y enake 0. Na koncu pa pot z ukazom *Z* povežemo do začetne točke na krožnici, ter tako dobimo zaključen odsek. Primer končnega tortnega diagrama s petimi odseki je prikazan na sliki 15. Z zeleno barvo smo označili poti, ki po zgoraj opisanem postopku definirajo posamezen odsek.

```
<g x="7" y="11" width="78" height="78" id="n_graf">
  <path id="odseki" d="M2.3879823899536357e-15,-39A39,39 0 0,1 34.45696973339489,-18.267929187291813L0,0Z"
  fill="#571FFF" transform="translate(50,54)"></path>
  <path id="odseki" d="M34.45696973339489,-18.267929187291813A39,39 0 0,1
  26.82027889528352,28.3138241850020941L0,0Z" fill="#1F57FF" transform="translate(50,54)"></path>
  <path id="odseki" d="M26.82027889528352,28.3138241850020941A39,39 0 0,1
  -26.820278895283515,28.3138241850020981L0,0Z" fill="#00B88A" transform="translate(50,54)"></path>
  <path id="odseki" d="M-26.820278895283515,28.3138241850020981A39,39 0 0,1
  -36.230092072854866,14.435388058256661L0,0Z" fill="#66FF33" transform="translate(50,54)"></path>
  <path id="odseki" d="M-36.230092072854866,14.435388058256661A39,39 0 0,1 -7.163947169860907e-15,-39L0,0Z
  fill="#FFCC33" transform="translate(50,54)"></path>
</g>
```

Slika 15: Prikaz izračunanih točk odseka končnega tortnega diagrama.

D3 ima za kreiranje odsekov kroga vgrajeno funkcijo *d3.svg.arc()*, ki smo jo uporabili na našem primeru. Funkcija za vsak podatek izračuna lok, ki ga bo določal v diagramu, ter rezultat shrani v znakovni niz. Parametri ki jih za to potrebuje, so notranji in zunanji polmer odseka ter začetni in končni kot. Za izračun slednjih dveh bi lahko uporabili še eno vgrajeno D3 funkcijo *d3.layout.pie()*, vendar smo se raje odločili izdelati lastno funkcijo JavaScript.

5.2.3.2 Osi in mreža diagramov

Pri izdelavi osi in mreže v ozadju stolpčnega in črtnega diagrama smo uporabili funkcijo *d3.svg.axis()*. Funkcija je namenjena tako izdelavi črt, kot tudi tekstovnih oznak na oseh. Ker so lahko v diagramu prikazani podatki poljubnih tipov, funkcija podpira delo z vsemi vrstami *d3.scale* umerjanj. Mrežo in osi dodajamo drugače od enostavnih elementov. Tudi tu najprej s selektorjem poiščemo element v katerega bomo os dodali, nato pa mu dodamo element <*g*>, ki smo ga v tretjem poglavju predstavili kot element za grupiranje. Glavni del osi se za tem generira ob klicu funkcije *d3.svg.axis()*. Na sliki 16 je predstavljena uporaba funkcije za izdelavo osi črtnega diagrama. Pri prvem odseku gre za os x, pri drugem pa za os y. Os x vrednosti oznak jemlje iz objekta *xMere*, ki vhodni nabor umeri na širino grafa, os y pa vrednosti vzame iz objekta *yMere*, ki definirane podatke umeri na višino grafa. Vsaki od osi smo morali z atributom *transform* določiti tudi izhodiščno pozicijo. Poleg osnovne funkcije smo uporabili več podfunkcij, s katerimi smo dodatno oblikovali oznake osi. Te so podrobneje opisane v tabeli 10, poleg ostalih najpomembnejših funkcij za oblikovanje oznak.

```
/* --- osi x in y --- */
d3.select("#n_graf").append("g")
  .attr("id","x_oznake")
  .attr("transform",
        "translate(" + (z_razmak+4) +
        "," + (n_visina + z_razmak - 2) + ")")
  .call(d3.svg.axis()
          .scale(xMere).tickSize(2,1,2).tickPadding(0.6));

d3.select("#n_graf").append("g")
  .attr("id","y_oznake")
  .attr("transform",
        "translate(" + (z_razmak+4) +
        "," + (z_razmak - 2) + ")")
  .call(d3.svg.axis()
          .scale(yMere).orient("left").tickSize(2,0,2).tickPadding(0.6));
```

Slika 16: Prikaz definiranja osi x in y.

Ime funkcije	**Opis**
scale	Pokličemo ustrezno funkcijo za umerjanje, s čimer dosežemo, da si bodo oznake ali mrežne črte sledile enakomerno in bodo prikazovale ustrezne pripadajoče vrednosti.
orient	Orientacija oznak glede na os. Možne postavitve so pod, nad, levo ali desno od osi.
ticks	Predlog števila oznak na osi. D3 za podano število preveri ali bi vse oznake imele enostavne vrednosti (npr. pri numeričnih oznakah večkratniki števila 100, ali pri datumskih oznakah posamezni meseci). Če za podano število ne dobi primernega nabora oznak, poišče najbližje število, ki ustreza tem pogojem [45].
tickSubdivide	Za določitev števila manjših oznak med glavnimi oznakami s tekstom.
tickSize	Sprejme tri vrednosti s katerimi določimo dolžino črt glavnih oznak, manjših oznak, ter dolžino končne oznake na koncu osi.
tickFormat	Namenjen določitvi formata teksta oznak (npr. decimalno število, besedilo ali datumsko polje). Pri generiranju mreže uporabimo prazna narekovaja, ter tako ustvarimo samo črte brez oznak.
tickPadding	Določa prazen prostor med tekstom in črto posamezne oznake v slikovnih točkah.

Tabela 10: Funkcije D3 uporabljene za urejanje osi diagrama.

5.2.4 Oblikovni slog

Oblikovne lastnosti elementov, ki smo jih kreirali s knjižnico D3 smo določili s selektorji CSS, ali z operatorjem *attr*, ki smo ga že podrobneje predstavili. Poleg *attr* lahko v D3 urejamo slog elementa tudi z operatorjem *style*. Ta ima zelo podobno strukturo kot istoimenski atribut v HTML. V oglatem oklepaju navedemo poljubno dolg seznam lastnosti s pripadajočimi vrednostmi, ki jih želimo uveljaviti. Njegove uporabe smo se izogibali in ga raje nadomestili s selektorji CSS.

Kot poseben primer oblikovanja na naši spletni strani lahko navedemo barvanje diagramov, kjer smo morali ustvariti seznam heksadecimalnih barvnih kod in jih razporediti po posameznih elementih. Za pravilno razvrstitev smo si tudi tu pomagali s funkcijo *d3.scale*,

vendar smo namesto že omenjene funkcije *d3.scale.linear()* uporabili vrstilno umerjanje z *d3.scale.ordinal()*. Kot vhodno domeno smo vzeli seznam števil od 0 do n, kot izhodno območje pa smo navedli seznam barvnih kod. Ko uporabnik na spletni strani izbere želeno barvo se tako pokliče funkcija, ki s selektorjem poišče vse jedrne elemente diagrama (to so stolpci, odseki in črta) in glede na indeks trenutnemu elementu priredi barvo iz seznama. Na sliki 17 je prikazana funkcija, ki prireja barve odsekov v tortnem diagramu. Omeniti gre še, da ima knjižnica D3 tudi štiri že vgrajene *d3.scale* funkcije, ki podatkom avtomatsko priredijo pred definirane barve: *d3.scale.category10()*, *d3.scale.category20()*, *d3.scale.category20b()* in *d3.scale.category20c()*. Vsako od teh smo preizkusili na naši spletni strani, vendar se nam nobena barvna kombinacija ni zdela ustrezna in smo zato ustvarili svojo.

```
46 function torta_zbarvo(evt){
47    /* ustvarili smo dve barvni lestvici: modro in raznobarvno */
48    var b_cat = d3.scale.ordinal().domain([0,1,2,3,4,5,6,7])
49                .range(["#1D0079","#3E10D5","#033EFF","#00567E",
50                        "#1488FF","#27A3BA","#67B2FF","#809AF1"]);
51    var r_cat = d3.scale.ordinal().domain([0,1,2,3,4,5,6,7,9])
52                .range(["#571FFF","#1F57FF","#00B88A","#66FF33","#FFCC33",
53                        "#FF855C","#FF571F","#CC3333","#8033CC"]);
54
55    // glede na izbrani gumb uporabimo ustrezno lestvico
56    if(evt.id == "modra"){
57      d3.select("#n_graf").selectAll("path") // obarvanje odsekov
58        .attr("fill", function(d,i){return b_cat(i);});
59      d3.select("div.legenda").selectAll("rect") // obarvanje legende
60        .attr("fill", function(d,i){return b_cat(i);});
61    }
62    if(evt.id == "razno"){
63      d3.select("#n_graf").selectAll("path")
64        .attr("fill", function(d,i){return r_cat(i);});
65      d3.select("div.legenda").selectAll("rect")
66        .attr("fill", function(d,i){return r_cat(i);});
67    }
68 }
```

Slika 17: Funkcija za barvanje elementov diagrama.

6 Testiranje in rezultati

Med začetnim pregledovanjem specifikacij SVG smo ugotovili, da se je širša podpora standarda v spletnih brskalnikih začela pojavljate šele v zadnjih nekaj letih. Zato smo postali skeptični glede obsega omogočenih funkcionalnosti SVG na posameznem brskalniku. Pri testiranju smo se zato odločili za uporabo vseh brskalnikov, ki so nam bili na voljo. To so bili: Google Chrome 32, Firefox 26, Internet Explorer 11, Opera 18 in brskalnik na Androidu 4.0 in Androidu 4.1. Spletno stran smo testirali na osebnem računalniku z diagonalo zaslona 55,9 cm, na tabličnem računalniku Samsung Galaxy 2 10.0 s 25,6 cm diagonalo zaslona, ter na pametnem telefonu HTC Desire 500 z diagonalo zaslona 10,9 cm.

V začetku razvoja spletne strani ter tudi kasneje pri manjših popravkih, smo prikaz vsebine velikokrat testirali preko manjšanja in večanja okna brskalnika na namiznem računalniku, kar pa ni realen način testiranje za različne vrste naprav z raznimi zaslonskimi ločljivostmi. Ker nismo imeli na voljo velikega števila mobilnih naprav, smo si pri testiranju odzivnosti spletne strani pomagali tudi s temu namenjenimi spletnimi aplikacijami. Preizkusili smo jih več, najustreznejša pa se nam je zdela aplikacija ProtoFluid, saj ponuja simulacijo za več priljubljenih mobilnih naprav (npr. Nexus 7, iPad ali Surface Pro) [46]. Kljub temu je bilo ključnega pomena testiranje spletne strani na dejanskih napravah. ProtoFluid sicer zna simulirati postavitev elementov na manjših napravah, v našem primeru pa je bilo enako pomembno preveriti, kako različne naprave obravnavajo grafike SVG.

Prav podpora SVG se je pri testiranju na različnih napravah in brskalnikih izkazala za najbolj problematično. Konkretni primeri, ki jih velja omeniti so:

- **Označevanje teksta**: Kot smo omenili v tretjem poglavju, je besedilo uporabljeno v elementih SVG zapisano kot navaden tekst in ga zato na spletni lahko iščemo. Vsi brskalniki na katerih smo iskanje teksta testirali, so sicer pravilno izpisali koliko zadetkov za iskani tekst obstaja, vendar ga niso vsi znali označiti. To sta naredila zgolj Internet Explorer in Firefox.
- **Ločila med točkami poti**: Pri grafiki za izbiro diagrama na začetni strani smo za definiranje lokov uporabili element <*path*>. SVG pri atributu *d*, v katerem naštevamo ukaze in točke ne definira standardnega ločila med posameznimi vrednostmi. Uporabljamo lahko vejico, presledek, ali celo pišemo ukaz in vrednost skupaj. Prvotno smo zato želeli posamezne vrednosti ločiti z vejico, kar je večina brskalnikov razumela kot pravilno sintakso. Izjema je bil Firefox, ki tako definiranih elementov ni prikazoval, zato smo morali vrednosti ločiti s presledki.
- **Skriptni jezik znotraj elementa SVG**: Ker je skriptni jezik dovoljen tudi v elementu SVG, smo želeli ta način uporabe preizkusiti pri izbiri diagrama na začetni strani. Ustvarili smo enostavno funkcijo JavaScript, ki spremeni barvo pisave in velikost

loka, na katerega se uporabnik postavi z miško, z namenom preglednejše izbire posameznega diagrama. Tudi tu je večina brskalnikov rezultat znala prikazati, izjema pa je bil brskalnik na Androidu.

- **Filtri SVG**: Ena od funkcionalnosti, ki smo jo uporabili pri ozadju naslovov in gumbov so filtri SVG. Njegovo namen smo omenili v tretjem poglavju, v našem primeru pa smo konkretno uporabili element filtra <*feGaussianBlur*> s katerim smo dosegli učinek sijaja. Pri testiranju pa smo ugotovili, da Androidov brskalnik ne zna prikazovati filtrov.
- **Umerjanje zunanjih grafik**: Grafike SVG iz zunanjih datotek smo v spletno stran vključevali z elementom <*img*>. Ker so bile to slike, ki smo jih kreirali z Inkscapom, je bilo polje SVG definirano s fiksno širino in višino. Umerjanje takih slik v manjši prostor smo želeli izvesti z dodatnim atributom *preserveAspectRatio*, s katerim smo določili razmerje v katerem naj se slika prikaže, ter s selektorjem CSS, kjer smo določili kolikšen del nadrejenega elementa <*div*> naj slika zasede (npr. 100 odstotkov v višino in 100 odstotkov v višino). Na ta način se je slika pravilno pomanjšala na vseh brskalnikih razen v Internet Explorerju. Pravilno pomanjšavo smo tam dosegli tako, da smo v vseh datotekah kreiranih z Inkscapom, obstoječe dimenzije območja SVG prestavili v atribut *viewBox*.

Umerjanje elementov spletne strani je na vseh napravah pravilno prilagojeno zaslonu, prav tako je zadovoljiv čas nalaganja strani in čas potreben za prikaz končnega diagrama. Na mobilnem telefonu so gumbi dovolj veliki, da jih lahko uporabljamo brez problemov. Prav tako menimo, da so oznake na oseh ter ostali elementi končnega diagrama, dovolj jasno predstavljeni. Nekoliko večje bi lahko bile oznake pri tabličnem računalniku, kjer imamo na razpolago dovolj prostora, da si to lahko privoščimo in bi s tem uporabniku dodatno olajšali delo.

7 Sklep

Cilj diplomskega dela je zajemal predstavitev odzivnega spletnega oblikovanja ter standarda SVG. Po podrobnejšem opisu obeh področij smo pridobljeno znanje uporabili pri izdelavi spletne strani za izdelavo diagramov. Namen spletne strani je uporabniku omogočiti izbiro več vrst grafov ter vnos poljubnega števila podatkov, na podlagi katerih se izriše končni diagram. Dodatno smo omogočili oblikovanje grafa z izbiro več barvnih naborov za obarvanje elementov in rotacijo oznak osi za primere, ko so te predolge, da bi jih prikazali vodoravno. Celotna spletna stran se zaradi odzivnega oblikovanja spremenljivo umerja v poljubno velik zaslon ciljne naprave.

Izkazalo se je, da je odziven pristop k razvoju sicer zahtevnejši in daljši, vendar končni rezultati vseeno odtehtajo začetni napor. Ustvarili smo namreč enotno stran, ki se prilega vsem zaslonom in za katero nam pri prihodnjem razvoju ali vzdrževanju ne bo potrebno spreminjati več strani. Izdelani končni diagrami dosegajo vse zastavljene funkcionalnosti, predvsem je zelo dobra enotna kvaliteta grafike, ne glede na velikost prikazovalnega polja. Tudi izris grafov je bil pri povprečni internetni povezavi na vseh testiranih napravah hiter. Kljub vsemu rezultati testiranja komponent SVG niso optimalni predvsem zaradi pomanjkljive, ali neenotne podpore na spletnih brskalnikih. Med končnim testiranjem smo ugotovili, da bi z nadaljnjim razvojem lahko spletno stran še dodatno optimizirali. Primarno bi to storili z dodatnim prilagajanjem velikosti polj na srednjih zaslonih kot jih imajo tablični računalniki, z možnostjo izvoza diagramov, razširitvijo izbora vrst diagramov ter z dodatnim oblikovanjem grafov (npr. sortiranje vrednosti, ali animiran prikaz natančnejših vrednosti posameznih elementov).

Obe predstavljeni področji v zadnjih letih znatno pridobivata na pomembnosti. Odzivno spletno oblikovanje zaradi strmega porasta uporabe mobilnih naprav, ki dostopajo do spleta, SVG pa zaradi širše pridobljene podpore v brskalnikih ter vključitvi v standard HTML5. Predvidevamo, da bo pomembnost obeh v prihodnjih letih zato še narasla, ter se bo tudi njun razvoj še nadgrajeval. Odzivno spletno oblikovanje je iz osnovnega koncepta, ki je predvideval zgolj vizualno prilagajanje, hitro prerasla v naprednejšo različico s strežniškimi komponentami RESS, ki dodatno optimizira celoten prenos vsebine od strežnika do odjemalca. Razvijalci SVG, trenutno delajo na prihodnji različici standarda 2.0, ki naj bi izšla letos. V prihodnjem razvoju je obljubljena še večja integracija z ostalimi spletnimi standardi ter poenostavitev samega jezika. Pričakujemo, da bodo tudi brskalniki v bodoče izpopolnili podporo skalabilne vektorskim grafikam, kajti obstaja še nekaj nepokritih področij, od katerih smo določene omenili tudi v jedrnem delu diplomske naloge.

Pri izdelavi spletne strani smo se v praksi seznanili s pristopom progresivnega nadgrajevanja ali mobilne prioritete. Menimo, da je ta način zelo koristen pri izdelavi odzivnih spletnih

strani, saj smo se z njegovo uporabo naučili določiti zgolj najbolj pomembne dele vsebine, ter si smernice odzivnega razvoja zastavili uspešneje, kot smo to storili pri postopni degradaciji. V celoti so nam vse raziskane tehnike razvoja utrdile prepričanje, da je zmožnost vsebinske neomejenosti spleta v primerjavi z ostalimi mediji njegova največja vrlina, ter jo moramo za izdelavo sodobnih mobilnih strani znati čim bolje uporabiti.

Seznam slik in tabel

Slika 1: Spremenljiva postavitev elementov [7]. 8
Slika 2: Medijska poizvedba. 11
Slika 3: Primerjava povečave vektorskih in bitnih grafik [31]. 20
Slika 4: Teksti, ki poteka po definirani poti [32]. 21
Slika 5: Začetna stran z izbiro diagrama. 28
Slika 6: Prva vnosna stran podatkov, z opozorilom pri napačnem vnosu. 29
Slika 7: Druga vnosna maska podatkov, z opozorilom pri napačnem vnosu. 30
Slika 8: Primer stolpčnega diagrama z rotiranimi oznakami osi x. 31
Slika 9: Primer tortnega diagrama z legendo. 31
Slika 10: Postavitev vsebine na najmanjših zaslonih. 34
Slika 11: Nastavitev atributov SVG elementa za pravilno umerjanje. 35
Slika 12: Ponazoritev umerjanja vhodne domene vrednosti v izhodno območje. 37
Slika 13: Umerjanje vnesenih vrednosti za izdelavo diagrama. 37
Slika 14: Dodajanje novih elementov v dokument SVG. 38
Slika 15: Prikaz izračunanih točk odseka končnega tortnega diagrama. 39
Slika 16: Prikaz definiranja osi x in y. 40
Slika 17: Funkcija za barvanje elementov diagrama. 41

Tabela 1: Vrste medijskih tipov. 11
Tabela 2: Vrste medijskih značilnosti. 12
Tabela 3: Atributi elementa SVG. 18
Tabela 4: Osnovni liki SVG. 20
Tabela 5: Elementi animacije SVG SMIL. 22
Tabela 6: Najpogostejše metode in lastnosti DOM za upravljanje elementov. 24
Tabela 7: Spreminjanje velikosti datoteke SVG pri optimizaciji. 33
Tabela 8: Velikostni razredi v medijskih poizvedbah na spletni strani. 34
Tabela 9: Operatorji D3 za vstavljanje novih elementov. 38
Tabela 10: Funkcije D3 uporabljene za urejanje osi diagrama. 40

Literatura

[1] (2013) J. Allsopp, A Dao of Web Design, A List Apart, 7. april 2000. Dostopno na: http://alistapart.com/article/dao

[2] E. Marcotte, Responsive Web Design, A Book Apart, New York, 2011.

[3] (2013) Uporaba informacijsko-komunikacijske tehnologije v gospodinjstvih in pri posameznikih, podrobni podatki, Slovenija, 2012 – končni podatki. Dostopno na: http://www.stat.si/novica_prikazi.aspx?id=5179

[4] (2013) Uporaba interneta v gospodinjstvih in pri posameznikih, Slovenija, 2013 - končni podatki. Dostopno na: https://www.stat.si/novica_prikazi.aspx?id=5795

[5] (2013) Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update, 2012–2017. Dostopno na: http://www.cisco.com/en/US/solutions/collateral/ns341/ns525/ns537/ns705/ns827/white_paper_c11-520862.html

[6] (2013) Alternatives to Responsive Design, Dostopno na: http://imagerystl.com/blog/12/21/Alternatives-to-Responsive-Design-Part-One-Mobile-Websites/

[7] (2013) 4 Benefits of Responsive Web Design. Dostopno na: http://www.affordablewebdesign.com/benefits-of-responsive-web-design/

[8] (2013) Eric Meyer's »Reset CSS« 2.0. Dostopno na: http://www.cssreset.com/scripts/eric-meyer-reset-css/

[9] B. Frain, Responsive Web Design with HTML5 and CSS3, Packt Publishing, Birmingham, 2012.

[10] (2013) Media Query & Asset Downloading Results. Dostopno na: http://timkadlec.com/2012/04/media-query-asset-downloading-results/

[11] (2013) B. Bos et al., Cascading Style Sheets Level 2 Revision 1 (CSS 2.1), pogl. 7 Media types, World Wide Web Consortium (W3C) priporočilo, 7. Junij 2011, Dostopno na: http://www.w3.org/TR/CSS2/media.html

[12] (2013) H. W. Lie et al., Media Queries, World Wide Web Consortium (W3C) priporočilo, 19. Junij 2012, Dostopno na: http://www.w3.org/TR/css3-mediaqueries/

[13] (2013) Mobile First. Dostopno na: http://www.lukew.com/ff/entry.asp?933=

[14] (2014) RESS: Responsive Design + Server Side Components. Dostopno na: http://www.lukew.com/ff/entry.asp?1392

[15] (2014) JavaScript Window – The Browser Object Model. Dostopno na: http://www.w3schools.com/js/js_window.asp

[16] (2014) Css3-mediaqueries-js. Dostopno na: https://code.google.com/p/css3-mediaqueries-js/

[17] (2014) 50 Useful Responsive Web Design Tools For Designers. Dostopno na: http://www.hongkiat.com/blog/rwd-tools/#testing

[18] (2014) Modernizr: Documetation. Dostopno na: http://modernizr.com/docs/#features-css

[19] (2014) Frameworks, Boilerplates And Tools For Responsive Web Design. Dostopno na: http://www.designyourway.net/blog/resources/frameworks-boilerplates-and-tools-for-responsive-web-design/

[20] (2014) Bootstrap. Dostopno na: http://getbootstrap.com/

[21] (2014) R. Cremin, L. Passani, Server-Side Device Detection: History, Benefits And How-To, Smashing Magazine, 24. september 2012. Dostopno na: http://mobile.smashingmagazine.com/2012/09/24/server-side-device-detection-history-benefits-how-to/

[22] (2014) RESTful Web services: The basis. Dostopno na: http://www.ibm.com/developerworks/webservices/library/ws-restful/

[23] (2014) J. A. Sæterås, Improve Mobile Support With Server-Side-Enhanced Responsive Design, Smashing Magazine, 9. april 2013. Dostopno na: http://mobile.smashingmagazine.com/2013/04/09/improve-mobile-support-with-server-side-enhanced-responsive-design/

[24] (2014) Getting started with RESS. Dostopno na: http://www.creativebloq.com/responsive-web-design/getting-started-ress-5122956

[25] (2014) Secret Origin of SVG. Dostopno na: http://www.w3.org/Graphics/SVG/WG/wiki/Secret_Origin_of_SVG

[26] D. Dailey, J. Frost, D. Strazzullo, Building Web Applications with SVG, O'Reilly Media, Sebastopol, 2012.

[27] (2014) Web Graphics Trends in 2013. Dostopno na: http://www.html5canvastutorials.com/articles/web-graphics-trends-in-2013/

[28] (2014) HTML Inline SVG. Dostopno na: http://www.w3schools.com/html/html5_svg.asp

[29] (2014) E. Dahlström et al., Scalable Vector Graphics (SVG) 1.1 (Second Edition), World Wide Web Consortium (W3C) priporočilo, 16. avgust 2011. Dostopno na: http://www.w3.org/TR/SVG11/

[30] J. D. Eisenberg, SVG Essentials, O'Reilly Media, Sebastopol, 2002.

[31] (2014) Scalable Vector Graphics. Dostopno na: http://sl.wikipedia.org/wiki/Scalable_Vector_Graphics

[32] (2014) The 'textPath' element. Dostopno na: http://www.w3.org/TR/SVG11/text.html

[33] (2014) Can I use SVG SMIL animation? Dostopno na: http://caniuse.com/svg-smil

[34] (2014) CSS3 Animations. Dostopno na: http://www.w3schools.com/css/css3_animations.asp

[35] (2014) Inkscape: Frequently Asked Questions. Dostopno na: http://www.inkscape.org/en/learn/faq/#Inkscape_and_other_programs

[36] (2014) Comparing Inkscape and Illustrator: What Are the Real Differences? Dostopno na: http://www.brighthub.com/multimedia/publishing/articles/73024.aspx

[37] (2014) How to Add Scalable Vector Graphics to Your Web Page. Dostopno na: http://www.sitepoint.com/add-svg-to-web-page/

[38] (2014) Data-Driven Documents. Dostopno na: http://d3js.org/

[39] (2014) Raphaël: a JavaScript API for SVG. Dostopno na: http://dev.opera.com/articles/view/raphael-a-javascript-api-for-svg/

[40] (2014) Snap.svg. Dostopno na: http://snapsvg.io/

[41] (2014) SVG Optimiser. Dostopno na: http://petercollingridge.appspot.com/svg-optimiser

[42] (2014) M. Maclean, D3 Tips and Tricks. Dostopno na: https://leanpub.com/D3-Tips-and-Tricks/read

[43] (2014) Jerome Cukier, D3: scales, and color, 11. august 2011. Dostopno na: http://www.jeromecukier.net/blog/2011/08/11/d3-scales-and-color/

[44] (2014) API Reference. Dostopno na: https://github.com/mbostock/d3/wiki/API-Reference

[45] (2014) Scot Murray, code artist, D3 Tutorials, Axes. Dostopno na: http://alignedleft.com/tutorials/d3/axes

[46] (2014) ProtoFluid, Dostopno na: http://protofluid.com/