Vizualizacija podataka:d3.js biblioteka

Stastny, Dario

Undergraduate thesis / Završni rad

2020

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Pula / Sveučilište Jurja Dobrile u Puli**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:137:252421

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-04-25**

Repository / Repozitorij:

Digital Repository Juraj Dobrila University of Pula

Sveučilište Jurja Dobrile u Puli
Fakultet informatike

DARIO STASTNY

VIZUALIZACIJA PODATAKA: D3.js BIBLIOTEKA

Završni rad

Pula, srpanj, 2020. godine

Sveučilište Jurja Dobrile u Puli
Fakultet informatike

DARIO STASTNY

VIZUALIZACIJA PODATAKA: D3.js BIBLIOTEKA

Završni rad

JMBAG: 0253005697, izvanredni student
Studijski smjer: Informatika

Predmet: Poslovni informacijski sustavi

Mentor: doc. dr. sc. Darko Etinger

Pula, srpanj, 2020. godine

IZJAVA O AKADEMSKOJ ČESTITOSTI

Ja, dolje potpisani Dario Stastny, kandidat za prvostupnika informatike ovime izjavljujem da je ovaj Završni rad rezultat isključivo mojega vlastitog rada, da se temelji na mojim istraživanjima te da se oslanja na objavljenu literaturu kao što to pokazuju korištene bilješke i bibliografija. Izjavljujem da niti jedan dio Završnog rada nije napisan na nedozvoljen način, odnosno da je prepisan iz kojega necitiranog rada, te da ikoji dio rada krši bilo čija autorska prava. Izjavljujem, također, da nijedan dio rada nije iskorišten za koji drugi rad pri bilo kojoj drugoj visokoškolskoj, znanstvenoj ili radnoj ustanovi.

Student
Dario Stastny

U Puli, srpanj, 2020. godine

IZJAVA
o korištenju autorskog djela

Ja, Dario Stastny dajem odobrenje Sveučilištu Jurja Dobrile u Puli, kao nositelju prava iskorištavanja, da moj završni rad pod nazivom „Vizualizacija podataka: D3.js biblioteka“ koristi na način da gore navedeno autorsko djelo, kao cjeloviti tekst trajno objavi u javnoj internetskoj bazi Sveučilišne knjižnice Sveučilišta Jurja Dobrile u Puli te kopira u javnu internetsku bazu završnih radova Nacionalne i sveučilišne knjižnice (stavljanje na raspolaganje javnosti), sve u skladu s Zakonom o autorskom pravu i drugim srodnim pravima i dobrom akademskom praksom, a radi promicanja otvorenoga, slobodnoga pristupa znanstvenim informacijama.
Za korištenje autorskog djela na gore navedeni način ne potražujem naknadu.

U Puli, 14. srpanj 2020. godine

Potpis
Dario Stastny

Sadržaj

1. UVOD 1
2. SVG (Scalable Vector Graphics) 2
2.1. OBLICI POMOĆU SVG-a 3
2.1.1. PRAVOKUTNIK 3
2.1.2. SVG KRUŽNICA 4
2.1.3. LINIJA 4
2.1.4. SVG PATH 5
3. RAD S OBLICIMA I ELEMENTIMA 7
3.1. SELEKTIRANJE ELEMENATA POMOĆU d3.js 7
3.2. UMETANJE ELEMENATA 8
3.3. ULANČAVANJE I ATRIBUTI 10
3.4. GRUPIRANJE 11
4. RAD S PODACIMA 12
4.1. PRIDRUŽIVANJE PODATAKA NA SVG 12
5. SKALE 20
5.1. LINEARNE SKALE 20
5.2. BANDWIDTH SKALE 22
5.2.1. MAX , MIN i EXTEND 23
5.2.2. MARGINE 23
6. FIREBASE 30
6.1. KREIRANJE FIRESTORE BAZE 30
7. AŽURIRANJE 34
7.1. AŽURIRANJE PODATAKA U STVARNOM VREMENU 36
8. TRANZICIJE 38
8.1. EVENTI 41
8.2. TOOLTIP 42
9. KRUŽNI GRAFIKON 46
9.1. TRANZICIJE POMOĆU INTERPOLACIJA I TWEENS-a 49
9.1.1. TRANZICIJA- ENTER SELECTION 49
9.1.2. TRANZICIJA- EXIT SELECTION 50
9.1.3. TRANZICIJA -AŽURIRANJE PODATAKA 51
9.1.4. LEGENDA 51
10. ZAKLJUČAK 53
11. LITERATURA 54

1. UVOD

Uz današnju tehnologiju koja je u konstantnom razvoju ili napredovanju svijet je pun informacija koje kolaju ili se dijele te danas možemo lako pristupiti informacijama s bilo kojeg mjesta u bilo koje vrijeme. Kako iz informacija ili podataka možemo puno toga saznati raspoređujemo ih i klasificiramo prema određenim karakteristikama te bi bilo poželjno da u nekim situacijama možemo i vizualizirati takve podatke. Postoje razni načini vizualizacije ali u ovom završnom radu vizualizirat ćemo podatke pomoću javascript biblioteke D3.js

Kako je d3.js javascript biblioteka moramo poznavati osnove programiranja u javascript jeziku kao što su objekti, funkcije, varijable i klase te također html osnove.

U ovom radu proći ćemo kroz osnove d3.js biblioteke kao što je izrada raznih dijagrama, SVG, tranzicije, interakcije s vizualima. Prethodno navedeno je samo je dio takve biblioteke ali rad o mogućnosti s takvom bibliotekom su neograničene.

Ko što smo naveli radit ćemo s podacima i vizualizirati ih ali moramo spremati navedene podatke u nekakvu bazu podataka a za to ćemo koristiti googleovu platformu Firebase o čemu ćemo naravno detaljnije sve opisati u daljnjim poglavljima.

U poslovnom svijetu kvalitetne, pravodobne i točne informacije su vrlo bitne i moćne pogotovo prilikom donošenja poslovnih odluka, te uz to vizualizacija takvih podataka i informacija samo nam daju još bolji uvid pri donošenju odluke ili shvaćanje podataka.

Također treba napomenuti da ćemo raditi sa tekst editorom Visual Studio Code koji je jedan od vrlo moćnih i popularnim editorima u svijetu developmenta.

Način ovog rada bit će objašnjavanje kroz izgradnju manjeg projekta tj. Na kraju ćemo imati povezane cijeline i završen manji projekt.

2. SVG (Scalable Vector Graphics)

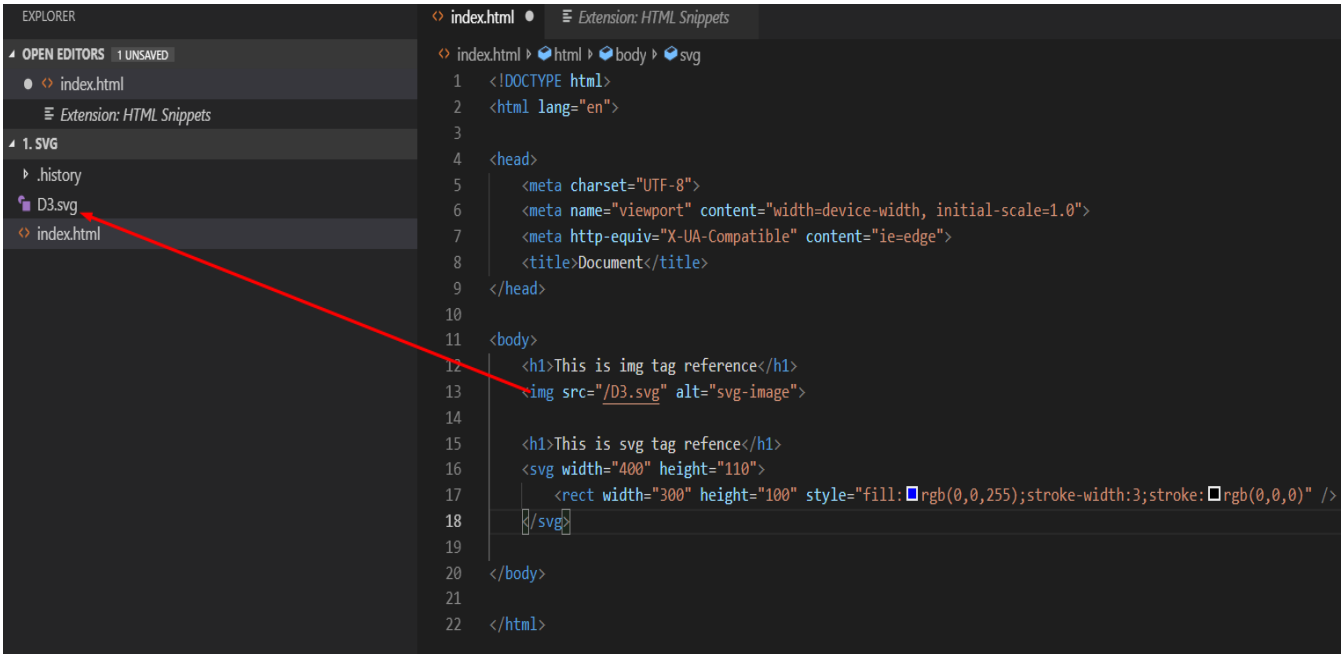
Znamo da ćemo raditi s d3.js bibliotekom i Firebase-om ali jedan od zadataka d3.js biblioteke je da stvori SVG slike koje onda možemo prikazivati u web pregledniku. Iz tog razloga treba malo upoznati se s SVG-om prije nego dublje se krene u samu biblioteku.

Kao što sama skraćenica govori SVG (scalable vector graphics) slike možemo povećavati i smanjivati bez da gube svoju kvalitetu na bilo kojoj rezoluciji ili veličini ekrana.

Još jedna kvaliteta SVG-a koju treba spomenuti je vrlo mala veličina datoteke za razliku od jpg i png formata te su zato vrlo pogodni kod izrade loga na web stranici ili prikaz dijagrama jer zbog male veličine brzo se prikazuju i ne zauzimaju prostor.

Stvoriti ili kreirati SVG možemo pomoću raznih programa kao što je Photoshop ili recimo AdobeXD te kad kreiramo SVG možemo ga jednostavno referencirati pri izradi web stranice kao što bi referencirali normalan PNG ili jpg format.

Drugi od načina je kreiranje SV G slike unutar same html strukture u editoru



Slika 1. Implementacija svg-a u html-u

Treći od načina je kreiranje pomoću d3.js biblioteke i javascripta te ćemo taj način koristiti u ovom radu. Razlog tome je što pomoću d3.js-a možemo lakše stvarati

kompleksne dijagrame te možemo napraviti interaktivni i dinamički sadržaj npr. ako želimo napraviti gumb koji će mijenjati sadržaje dijagrama.

2.1. OBLICI POMOĆU SVG-a

2.1.1. PRAVOKUTNIK

U ovom poglavlju ćemo samo prikazati kako napraviti neke od osnovnih oblika SVG-elemenata u samoj strukturi HTML dokumenta.

```
<h1>Ovo je SVG pravokutnik</h1>
<svg height="400" width="400">
   <rect x="300" y="100" fill="green" width="100" height="200"></rect>
</svg>
```

Slika 2. Pravokutnik html struktura

Ovdje vidimo da smo u strukturu html-a napravili svg oznake koje definiraju širinu i visinu područja u kojoj će biti naš element.

Zatim unutar svg oznaka označujemo naš pravokutnik sa ključnom rječju rect koja ima svoje x i y koordinate unutar svg-a , a ispunu ili boju popunjavamo ključnom rječju fill gdje smo u ovom slučaju stavili zelenu boju te visinu i širinu samog pravokutnika.

Rezultat navedenog koda je vidljiv na sljedećoj slici:

Ovo je SVG pravokutnik

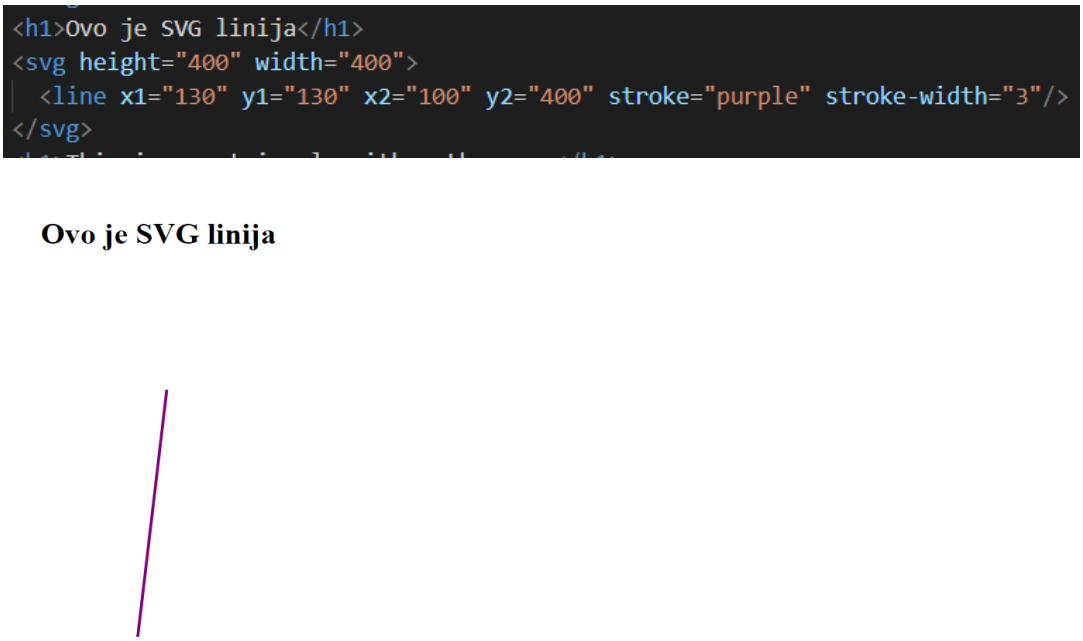
Slika 3. Pravokutnik web preglednik

2.1.2. SVG KRUŽNICA

Na donjoj slici možemo vidjeti kako smo kao i na prethodnom obilježenom području kreirali ovaj put kružnicu sa ključnom rječju circle. Koordinate kružnice su malo drugačije jer se udaljenost računa od sredine kružnice pa koordinate označavamo sa cx i cy. Ispunu smo stavili rozu a sa ključnom rječju stroke smo stavili rub oko kružnice a sa width smo mu dali određenu debljinu. Te rezultat možemo vidjeti na slici ispod.

Slika 5. Kružnica

2.1.3. LINIJA



Slika. 6 SVG linija

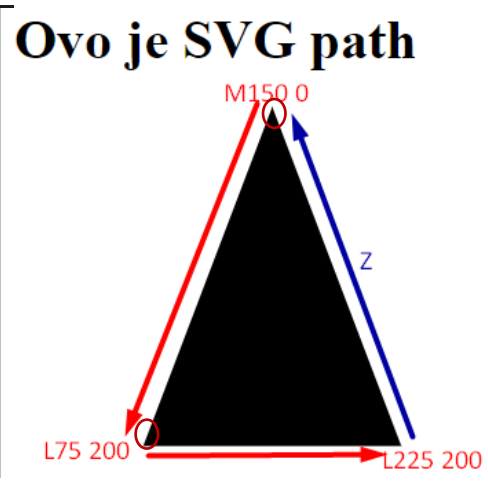
Kod kreiranja linije moramo dati x i y koordinate za početnu točku te tako i za završnu točku. Kao u koordinatnom sustavu tražimo početnu točku pomoću x1 i y1 i onda pomoću koordinata stavljamo završnu točku linije koja je x2 i y2.

Sve što je prethodno navedeno su neki od osnovnih oblika u svg-u no ako želimo kreirati ili stvarati neke od malo kompleksnijih oblika moramo koristiti neke druge opcije koje nam nudi svg.

Jedna od opcija je SVG path koji nam pomaže da crtamo po platnu proizovoljno određujući više linija i koordinata odjednom.

2.1.4. SVG PATH

```
<h1>Ovo je SVG path</h1>
<svg height="230" width="420">
  <path d="M150 0 L75 200 L225 200 Z"/>
</svg>
```

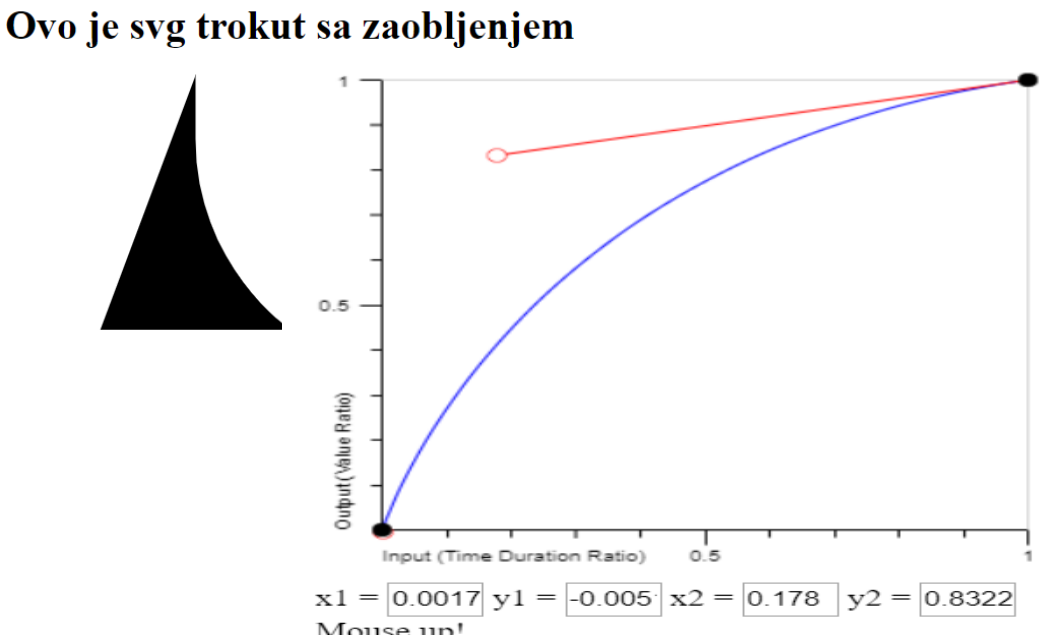


Slika 7. SVG Path

Oznaka M označava riječ move to što znači da još nismo počeli crtati niti povlačiti linije već samo se pozicioniramo na koordinate 150 i 0 tj. x i y koordinate. Oznaka L označava riječ line to što znači da povlačimo liniju do koordinate 75 i 200 te na koordinatama 225 i 200.

Oznaka Z označava close path što u biti doslovno i znači zatvaranje našeg trokuta s linijom. Prethodno navedeno nisu jedini atributi koje možemo koristiti s SVG-om. Recimo ako želimo zaobljenje možemo koristiti oznaku C ili curve koju ćemo staviti umjesto close path (Z) i unijeti 3 seta koordinata. Prvi set se sastoji od koordinata početne točke, drugi set se sastoji od kordinata gdje će se linija izvijati ili uvijati te zadnji set se sastoji od kranje točke same linije.

```
<h1>Ovo je svg trokut sa zaobljenjem</h1>
<svg height="400" width="400">
  <path d="M150 0 L75 200 L225 200 C 225 200 150 150 150 50" />
</svg>
```



Slika 8. SVG trokut sa zaobljenjem

Pored slike trokuta s iskrivljenom stranicom vidimo generator krivulje gdje vidimo 3 spomenute točke tj. početnu,završnu točku i točku izvijanja koja je obojana crvenom bojom.

Iz navedenih primjera vidimo kako možemo pomoću samog koda unutar strukture html dokumenta stvarati razne oblike u SVG formatu. No također vidimo kako bi na takav način stvaranje kompleksnijih oblika bilo vrlo teško zbog samog upisivanja linija i krivulja u html dokumentu zbog takvog razloga i također da nam se pojednostavi kreiranje kompleksnijih likova koristit ćemo d3.js biblioteku. Pomoću navedene biblioteke lakše ćemo kreirati trokute, pravokutnike, linije i pathove te razne kombinacije s kojima ćemo moći kreirati vrlo moćne vizualizacije podataka.

Prije nego što počnemo raditi kompleksnije stvari moramo uključiti d3.js biblioteku u projekt.

Slika 9. Uključivanje d3.js biblioteke

Na dnu html dokumenta smo uključili d3.js biblioteku te index.js odvojenu datoteku gdje ćemo zapisivati javascript tj. d3.js skripte.

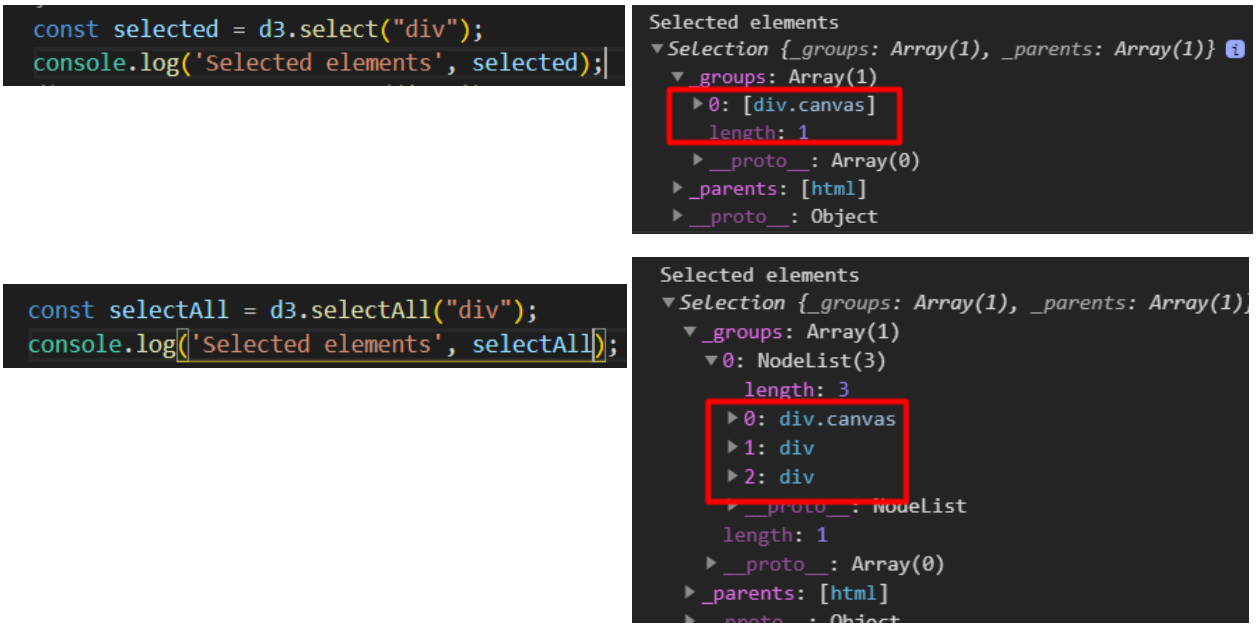
3. RAD S OBLICIMA I ELEMENTIMA

3.1. SELEKTIRANJE ELEMENATA POMOĆU d3.js

U prethodnom poglavlju smo vidjeli kako postaviti d3.js a sada ćemo demonstrirati kako sa d3.js komunicirati sa DOM-om (document object model) odnosno sa HTML dokumentom. To ćemo pokazati na način kako dodavati, mijenjati ili brisati elemente iz HTML-a. Kao i sa javascriptom možemo selektirati određene elemente te manipulirati s njima.

U javascriptu selektiramo ili obilježavamo elemente na razne načine kao što su document.querySelect koji obilježava samo prvi element ili document.querySelectAll kojim obilježavamo sve elemente s istim obilježjem.

Sa d3.js bibliotekom je vrlo slično samo što imamo ključnu riječ d3 koja označava da radimo s navedenom bibliotekom i select.



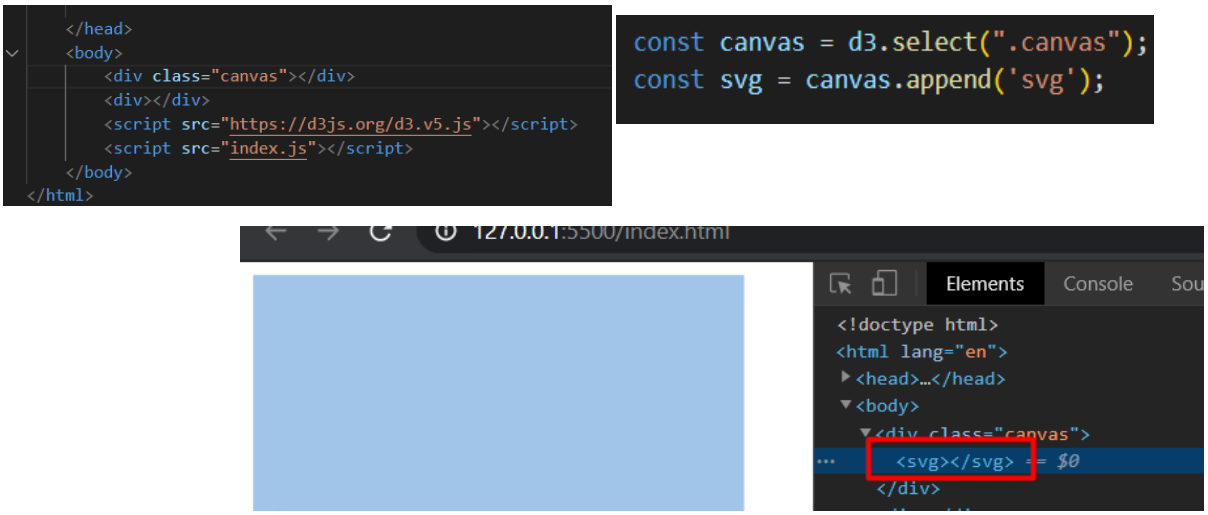
Slika 10. Selektiranje div elementa iz DOM-a

Iz prethodne slike vidimo kako selektramo element div koji se nalazi u HTML dokumentu. Također smo tu selekciju stavili u konzolu da bi vidjeli razlike između njih kad koristimo inspect u web pregledniku.

U prvoj slici vidimo kako je selektiran prvi div koji se nalazi u html dokumentu te na drugoj slici vidimo kako smo selektirali sve div elemente u HTML-u. Za razliku od čistog javascripta d3.js sve selekcije omotava sa Selection kako bismo mogli raditi kompleksnije radnje koje ćemo navesti i objasniti u slijedećim poglavljima.

3.2. UMETANJE ELEMENATA

Kao što smo već spomenuli kao i kod javascripta d3.js ima puno sličnosti jer kao što znamo da je to javascript biblioteka koja nadopunjuje čisti javascript. Tako kao i rad sa čistim ili vannilla javascriptom možemo na razne načine manipulirati elementima unutar html-a. Sada ćemo spomenuti append metodu pomoću koje možemo vrlo lako umetati na određeno mjesto svoj kreirani element.



Slika 11. Umetanje SVG elementa u DOM

Iz gornje slike vidimo da smo u html-u jednom div elementu dali klasu canvas. Nakon toga smo u javascript odvojenom dokumentu kreirali jednu varijablu tipa const jer se neće mijenjati i pomoći select metode selektirali element canvas. Zatim smo kreirali drugu varijablu koju smo nazvali svg te pomoću append metode umetnuli svg element unutar canvas elementa.

Sve što je navedeno može se vidjeti iz inspecta u konzoli web preglednika. Vidimo div s klasom canvas i unutar njega svg element.

Taj naš svg element je prazan, no u njega možemo naravno umetati još elemenata što vidimo u donjem primjeru.

Slika 12. Dodavanje raznih oblika u DOM

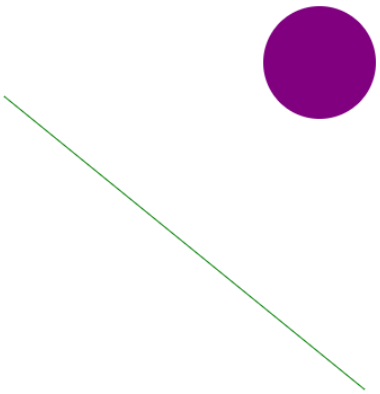
U gornjoj vidimo da smo na svg element još pomoću iste metode append umetnuli oblike kao što je linija, krug i pravokutnik i oni se sad nalaze unutar svg elementa. Pomoću navedene metode ne moramo umetati samo oblike već i sve ostale

elemente kao što su naprimjer paragrafi, linkovi itd. No kako radimo sa d3.js bibliotekom i vizualizacijom većinom ćemo se bazirati na oblike.

3.3. ULANČAVANJE I ATRIBUTI

Ulančavanje znači da na obilježeni element u našem slučaj div koji ima klasu canvas i element svg možemo vezati razne atribute koje smo vidjeli u prošli poglavljima.

```
JS index.js > ...
1   const canvas = d3.select(".canvas");
2
3   const svg = canvas.append('svg')
4     .attr ('height', 500)
5     .attr ('width', 500);
6
7   svg.append('line')
8     .attr('x1',340)
9     .attr('y1',390)
10    .attr('x2',20)
11    .attr('y2',130)
12    .attr('stroke','green');
13
14  svg.append('circle')
15    .attr('r',50)
16    .attr('cx',300)
17    .attr('cy',100)
18    .attr('fill','purple');
19
```



Slika 13. Ulančavanje atributima

Sa slike možemo vidjeti kako smo na element nadovezali razne atribute. Umjesto da stalno zapisujemo varijablu svg i dajemo joj pojedini atribut možemo kao na slici ulančavati atribute ključnom rječju attr gdje zapisujemo vrijednosti. Kao što vidimo na svg varijablu u kojoj se nalazi naš svg element stavili smo određenu širinu i visinu canvasa odnosno platna koje će sadržavati naše oblike ili elemente. Istoj varijabli smo dali atribute posebno za liniju te posebno za kružnicu.

Još jedna prednost ove biblioteke što možemo kreirati tekst u svg formatu.

```
svg.append('text')
.attr('x',20)
.attr('y',300)
.attr('fill','purple')
.text('Hello!! I am SVG text')
.style('font-style','italic');
```

Hello!! I am SVG text

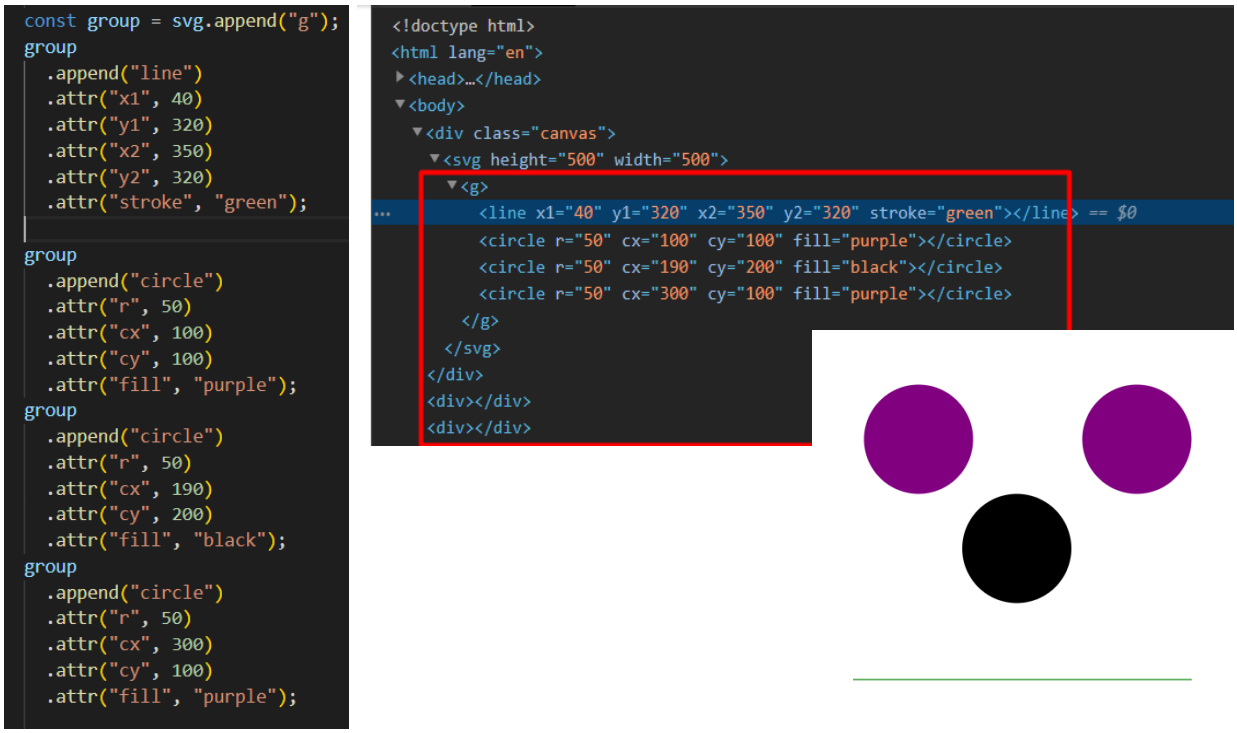
Slika 14. Dodavanje teksta i stila

Isto kao u prethodnim primjerima ulančali smo tekst ključnom rječju text gdje napravili zapis i sa atributima smo dali boju te x i y koordinate na platnu. Možemo koristiti i stilove u css-u u ovom slučaju smo tekstu dali stil italic.

3.4. GRUPIRANJE

Kad koristimo grupiranje ne kreiramo oblike kao što smo dosad kreirali npr. trokut, pravokutnik itd... već grupiramo sve elemente zajedno da možemo s njima manipulirati kao s jednom cijelinom.

Pomoću grupiranja možemo s elementima manipulirati i raditi transformacije kao što je pomicanje svih pravokutnika dijagrama u lijevo ili desno.



Slika 15. Grupiranje

Vidimo na gornjoj slici kako ćemo dva elementa napraviti da budu u grupi a tekst ćemo izostaviti iz grupe. Grupiranje elemenata je vrlo jednostavno, kreiramo varijablu group i na svg element dodajemo parametar 'g' što označava grupiranje u d3.js. Zatim smo na tu varijablu group kao i prošlim primjerima umetnili oblik sa atributima. Sada su elementi u grupi i s njima sada možemo manipulirati kao s cijelinom

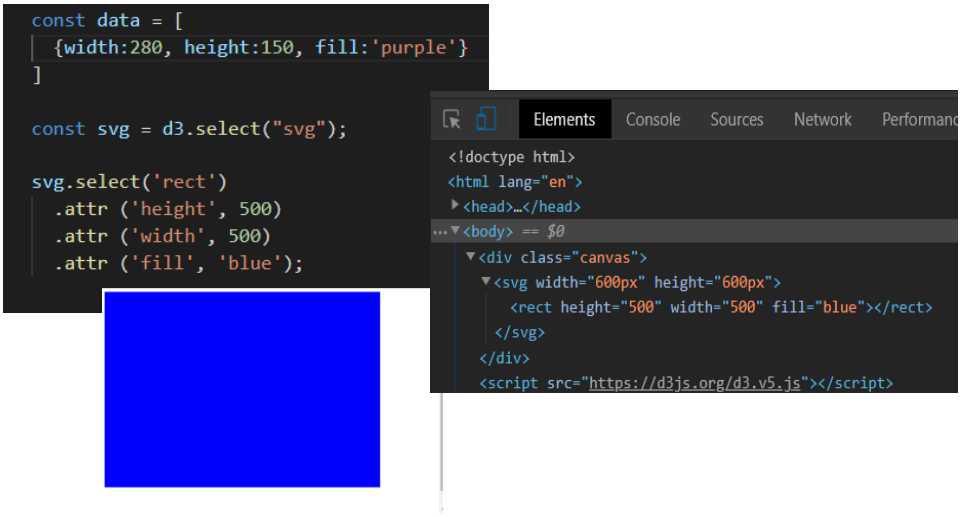
Vidimo kako i inspectoru imamo g oznaku u kojemo su kružnice i linija a tekst je van toga te sada možemo kao i što smo već napomenuli manipulirati ili micati grupirane elemente kao cjelinu. Možemo sada na grupu staviti atribut transform translate te pomaknuti elemente koji su u grupi u nekom smjeru.

4. RAD S PODACIMA

4.1. PRIDRUŽIVANJE PODATAKA NA SVG

Kako smo u uvodu vidjeli kako možemo stvoriti SVG oblike koji će nam koristiti u raznim dijagramima to nam ništa ne znači ukoliko ti oblici nisu nekako vezani za samo podatke a u ovom završnom radu se i bavimo vizualizacijom podataka. Zasad smo vidjeli da imamo već unaprijed postavljene ili 'hardcodirane' podatke koji su atributima vezani za SVG oblike, no da bi podaci pokazivali ili bili interaktivni s obzirom na podatke recimo u nekakvoj bazi podataka moramo dinamički unijeti vrijednosti u oblike kako bi davali sliku ili dijagram o samim navedenim podacima.

Za pridruživanje podataka oblicima koristi se data metoda. U početku za demonstraciju nećemo koristiti podatke iz baze neko ćemo stvoriti objekt s podacima te ćemo simulirati kao podatke iz baze.



Slika 16. Ispuna elementa sa atributom

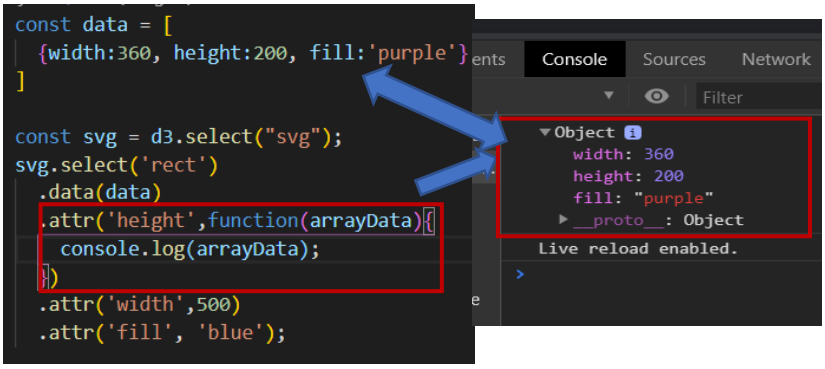
Za početak smo izmijenili HTML dokument i u jedan div klase canvas smo stavili SVG širine 600px i visine 600px i unutar jedan oblik trokuta (rect). Da bi dohvatili naš element moramo selektirati barem njegovog roditelja (parent element) te smo napravili varijablu svg koja selektira SVG element koji smo stvorili u HTML

dokumentu. Sada kada smo selektirali roditelja našeg elementa pomoću ulančavanja možemo uhvatiti naš trokut (rect) te mu dodati atribute pomoću metode attr što smo i naveli u prijašnjem poglavlju.

Stvoreni polje ili array s nazivom data sadrži objekt koji ima podatke o širini, visini i popuni boje.

Pomoću data metode možemo vidjeti kako se mijenjaju podaci u atributima sa podacima iz navedenoga polja. Varijabla rectangle sadrži selektirani pravokutnik koji ima svoje atribute. Iznad atributa smo deklarirali metodu data u koju upisujemo kao parametar naše polje s nazivom data.

No kako sada uzeti takve podatke iz polja tj. objekta rješenje je da stvorimo atribut koji se veže na data i unutar njega anonimnu funkciju (funkcija koja nema svoj nekakav naziv) koja nam vraća vrijednost podatka iz polja. Ta naša funkcija primat će kao parametar data odnosno pomoću nje imat ćemo pristup objektu sa podacima.



Slika 17.Dohvaćanje vrijednosti elementa iz polja objekata

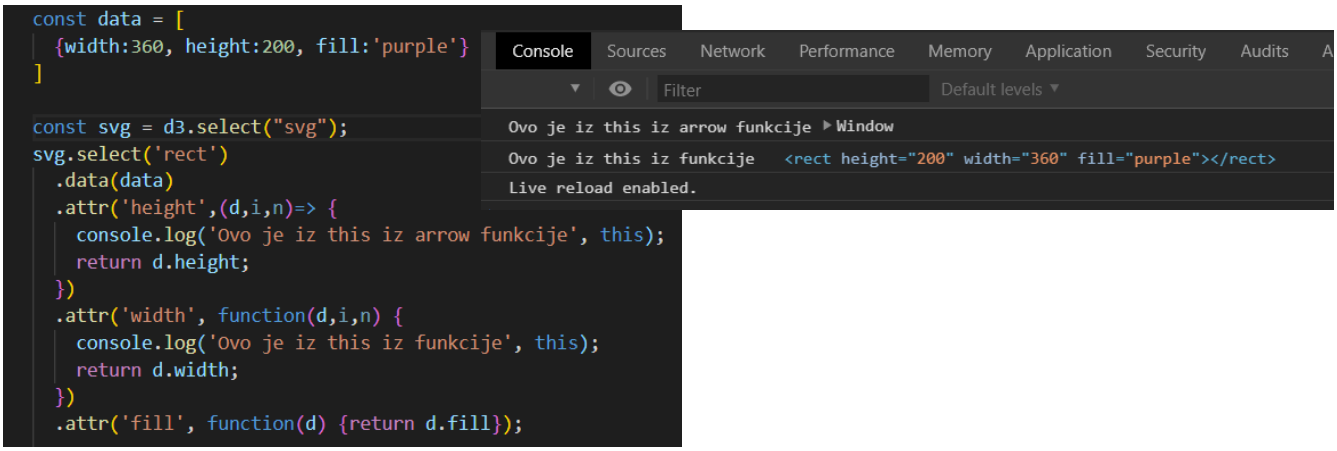
Na donjoj slici vidimo da smo dodali metodu data() te smo u atribut umjesto drugog argumenta koji je bio broj visine stavili funkciji koji prima parametar ili referencu na data() metodu te smo umjesto console.log u konzolu upisali d koji je naš parametar da vidimo što sadržimo. U inspectu u pregledniku kad odemo na console vidimo ispis objekta koji zaista drži podatke iz polja s novim podacima koje smo kreirali. Sada vidimo da možemo pristupiti podacima preko našeg parametra u funkciji koja je reference na data metodu.

Slika 18. Kreiranje pomoću data atributa

Naša funkcija ne mora primati samo jedan parametar već više njih. Recimo ako ubacimo drugi parametar npr. i on vraća indeks trenutnog elementa što je nula jer je prvi i jedini element a ako za treći parametar upišemo recimo n u ovom slučaju funkcija bi vraćala trenutni element što bi bio pravokutnik ili rect.

ES6 ES5 Funkcije

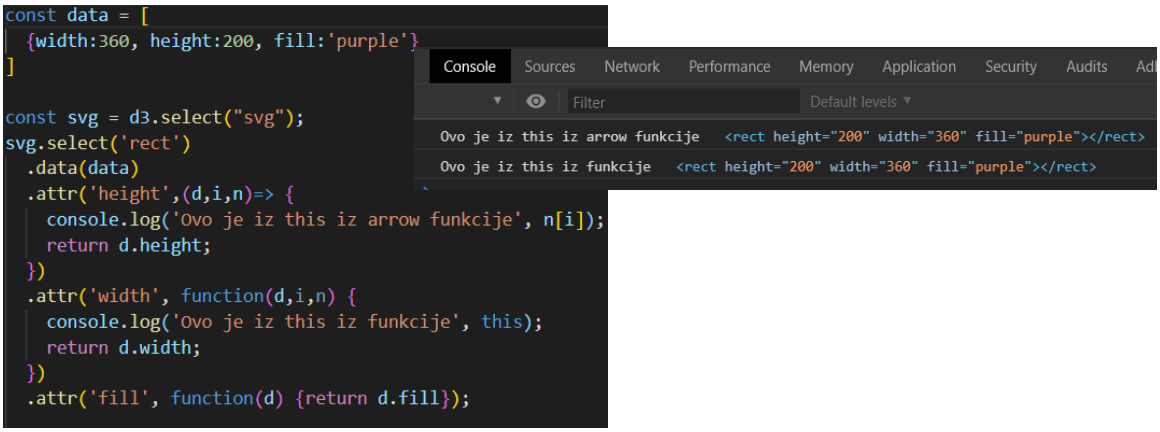
Kako znamo već duže se vrijeme za funkcije koristila ključna riječ function koja je standard u ES5. ES je kao nekakva verzija jezika javascript te svaka novija verzija se razlikuje u sintaksi. Ovo spominjemo jer ćemo raditi s funkcijama te ćemo koristiti također i arrow funkciju(ES6) koja je funkcija zapisana u drugom obliku tj. sintaksa je drugačija i ima drukčije značenje za riječ this u javascriptu koja ponekad može biti zbunjujuća.



Slika 19. This javascript

Kako vidimo iz gornje slike u console-log-u smo u arrow funkciji i običnoj funkciji napisali riječ this i u inspectu u pregledniku točno vidimo da imaju drugi kontekst.

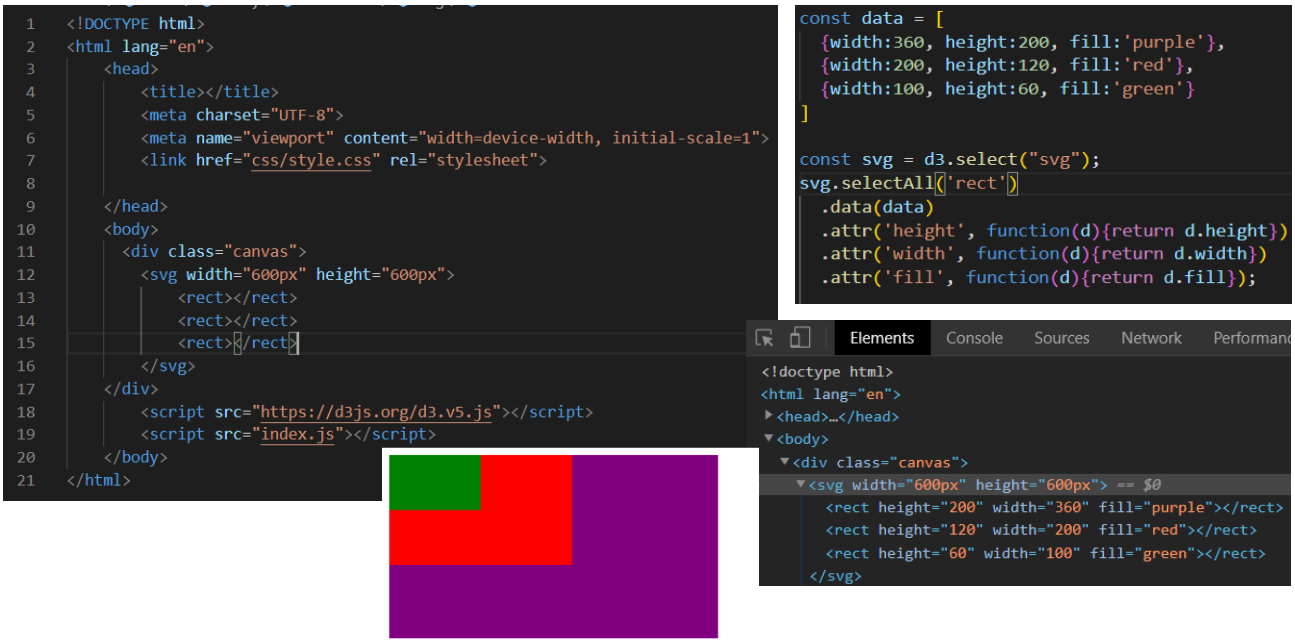
Arrow funkcija se referira na cijeli window objekt tj. cijeli prozor u pregledniku dok se funkcija ES5 sintakse odnosi na samu funkciju i i vraća samo objekt na koji se referira u ovom slučaju pravokutnik. Ako želimo da se arrow funkcija referira samo na naš objekt možemo ga dohvatiti preko parametara funkcije imamo na raspolaganju korištenje svih objekata koji se nalaze u window objektu pa tako i naš rect ili pravokutnik.



Slika. 20. ES6 i ES5 javascript sintaksa

U gornjoj slici vidmo kako smo se referirali na objekt pomoću arrow funkcije tako što smo dohvatili polje i pomoću indeksa objekt i dobili smo jednak rezultat kao i sa regularnom ES5 funkcijom. Danas se još uvijek koristi ES5 sintaksa ali ES6 postaje sve više standard u svijetu web developmenta

Iz svega navedenog smo vidjeli kako možemo odabrati jedan element no što ako želimo odabrati više elemenata i na njih primjenjivati. Recimo da želimo na četiri različita pravokutnika primjeniti različite vrijednosti trebamo svakako više podataka što nadalje znači da ćemo trebati i više objekata.

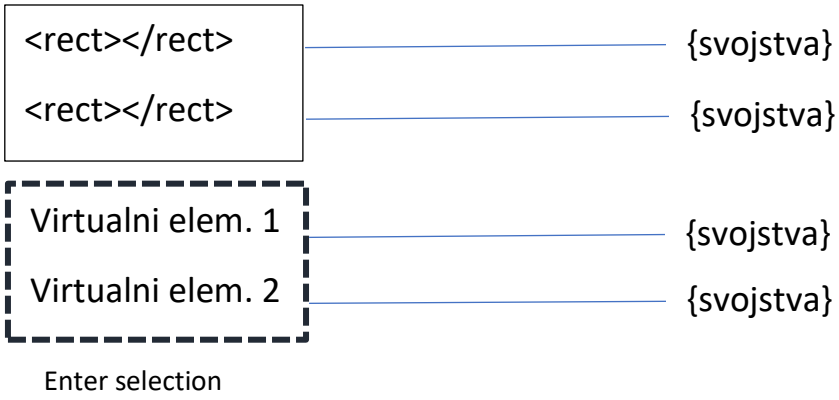


Slika 21. Kreiranje pravokutnika

Iz gornjeg primjera dalje vidimo kako smo kreirali nove objekte i svaki od njih ima drukčija svojstva. Prije smo radili samo sa metodom select no ona nam u ovom slučaju ne pomaže jer kako znamo ona odabire samo prvi element koji pronađe u DOM-u no sada trebamo više elemenata odabrati i primjeniti svojstva na njih.

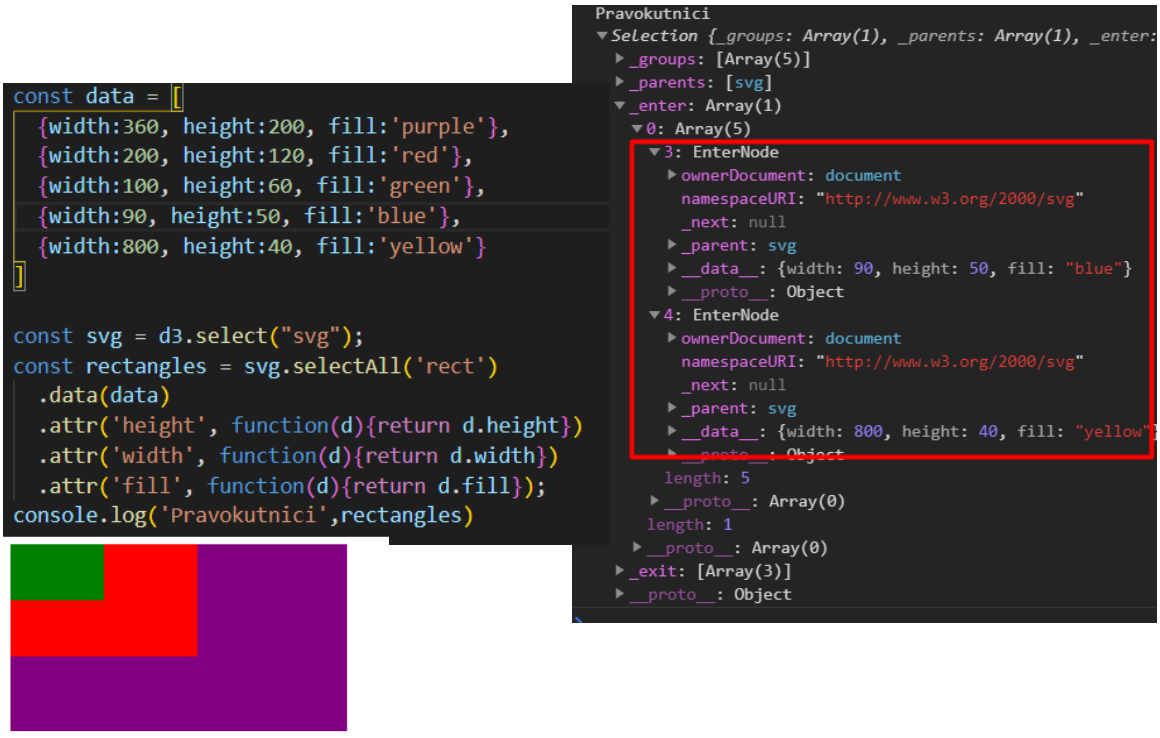
Sa SelectAll metodom odabiremo sve pravokutnike u ovom slučaju te se redom na njih primjenjuju svojstva koja su u objektima. Sve možemo detaljno vidjeti na slici u inspectu preglednika. Vidimo također da elementi idu jedan preko drugog ali to će se razjasniti u daljnjim poglavljima.

Zasad smo zbog demonstracije 'hardcode'-irali elemente tj. stavili smo da ima četiri pravokutnika i zapisali objekte koje će koristiti no to u praksi nije baš primjenjivo jer ako radimo s bazom podataka ili primamo podatke s nekog drugog izvora ne znamo možda koliko će pravokutnika biti. Zato bi selektiranje trebalo biti više dinamičnije prirode.



Slika 22. Virtualni elementi

Znamo da imamo tri pravokutnika i pet svojstava. Svaki pravokutnik prima jedno svojstvo ili kako se naziva props ali pošto ima pet svojstvava d3.js pretpostavlja da će biti još 2 elementa te automatski stvara virtualne elemente. Odabir takvih elemenata koji su virtualni odabiremo sa enter selection-om kojeg možemo zamisliti kao nekakav container za virtualne elemente preko kojeg im možemo pristupiti.



Slika 22. Enter selection u konzoli preglednika

U HTML-u smo kreirali smo tri pravokutnika a vidimo da imamo pet svojstava ili takozvana propsa te pomoću ispisa konzole u inspectu preglednika da imamo polje enter koje prikazuje prazne čvorove. Pošto imamo tri pravokutnika definirana u HTML

dokumentu a pet svojstava prikazuju se samo pravokutnici koji su definirani u DOM-u što i vidimo sa gornje slike.

```
    </head>
    <body>
      <div class="canvas">
        <svg width="600px" height="600px">
          <rect></rect>
          <rect></rect>
          <rect></rect>
        </svg>
    </div>
        <script src="https://d3js.org/d3.v5.js"></script>
        <script src="index.js"></script>
    </body>
</html>
```

```
const svg = d3.select("svg");
const rectangles = svg.selectAll('rect')
  .data(data)
  .attr('height', function(d){return d.height})
  .attr('width', function(d){return d.width})
  .attr('fill', function(d){return d.fill});
console.log('Pravokutnici',rectangles);

rectangles.enter()
  .append('rect')
  .attr('height', function(d){return d.height})
  .attr('width', function(d){return d.width})
  .attr('fill', function(d){return d.fill});
```

Slika 23. Enter Selection pravokutnici

Da bi primjenili sva svojstva i na virtualne elemente pristupamo im sa metodom enter() te pomoću append metode dodajemo vrstu elementa što je u ovom slučaju pravokutnik te im dodajemo različite atribute. U inspectu vidimo da imamo sada šest pravokutnika koje smo pomoću append metode dodali u HTML dokument te primjenjena sva svojstva te to možemo sa neke strane gledati kao nekav update za naše elemente.

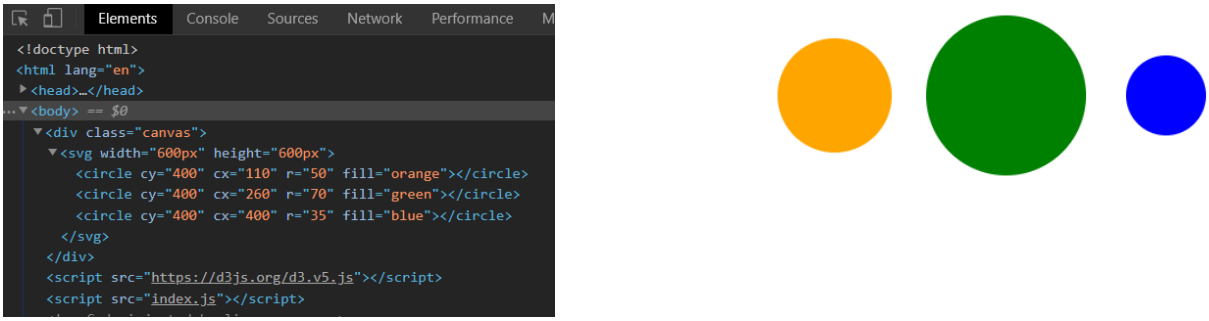
Dosad smo vidjeli kako pridruživati podatke i primjenjivati ih na razne oblike i kako koristimo lokalne podatke koje smo sami kreirali no d3.js radi s raznim vrstama podaka kao JSON, csv, vanjske baze podataka itd.

Koristiti ćemo podatke iz json dokumenta kojeg smo kreirali i nazvali circles.json te naš HTML dokument ne sadrži nikave oblike. Cilj nam je uzeti podatke iz vanjskog JSON-a umjesto da koristimo objekte kao što smo koristili u prijašnjim primjerima i na taj način vidjeti kako d3.js vrši dohvat i interakciju sa objektima iz vanjskog izvora.

Slika 24. Učitavanje iz json datoteke

Kreirali smo json file koji sadrži tri objekta koji imaju radius, distance (udaljenost) i fill(boja ispune) te ćemo te podatke dalje koristiti za prikaz oblika kružnica u pregledniku. Prvo što moramo napraviti je selektirati svg u koji ćemo umetnuti kružnice. Zatim pomoću promise funkcije ćemo učitati podatke is json-a. Promise funkcija je također dio javascripta i ukratko govoreći to je funkcija koja prvo izvršava nekakav događaj u ovom slučaju dohvaćanje json datoteke i tek kad izvrši takvo dohvaćanje izvršava se sve što je u then metodi.

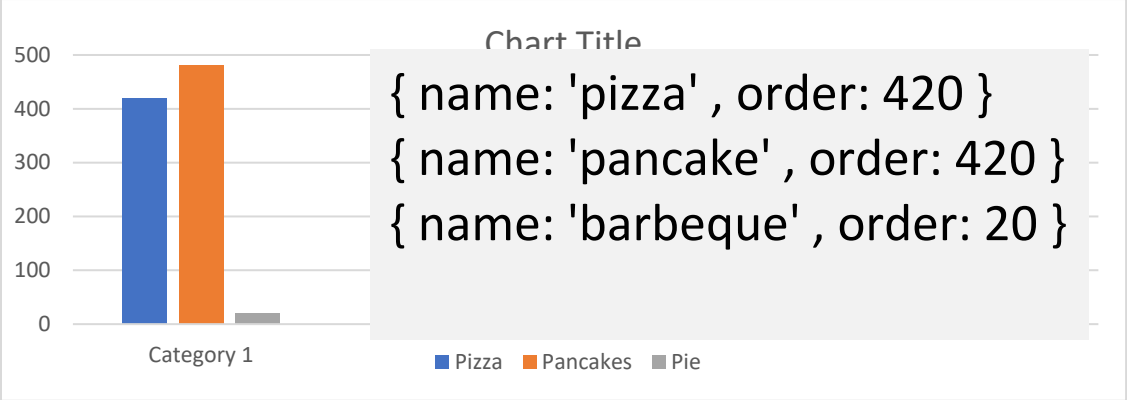
U then metodi imamo anonimnu funkciju koja prima data za parametar što je podatak iz json-a. U navedenoj funkciji inicijaliziramo varijablu circles koja sadrži selektirane elemente oblika kružnice te dodijeljujemo podatke iz JSON-a. A postoji već krug u DOM-u ažuriramo ga s novim kružnicama sa podacima iz JSON-a . Ukoliko ne postoji niti jedan krug pomoću enter selection-a pristupamo virtualnim elementima te im dodijeljujemo oblik circle te atribute.



Slika 25. Kružnice

5. SKALE

5.1. LINEARNE SKALE



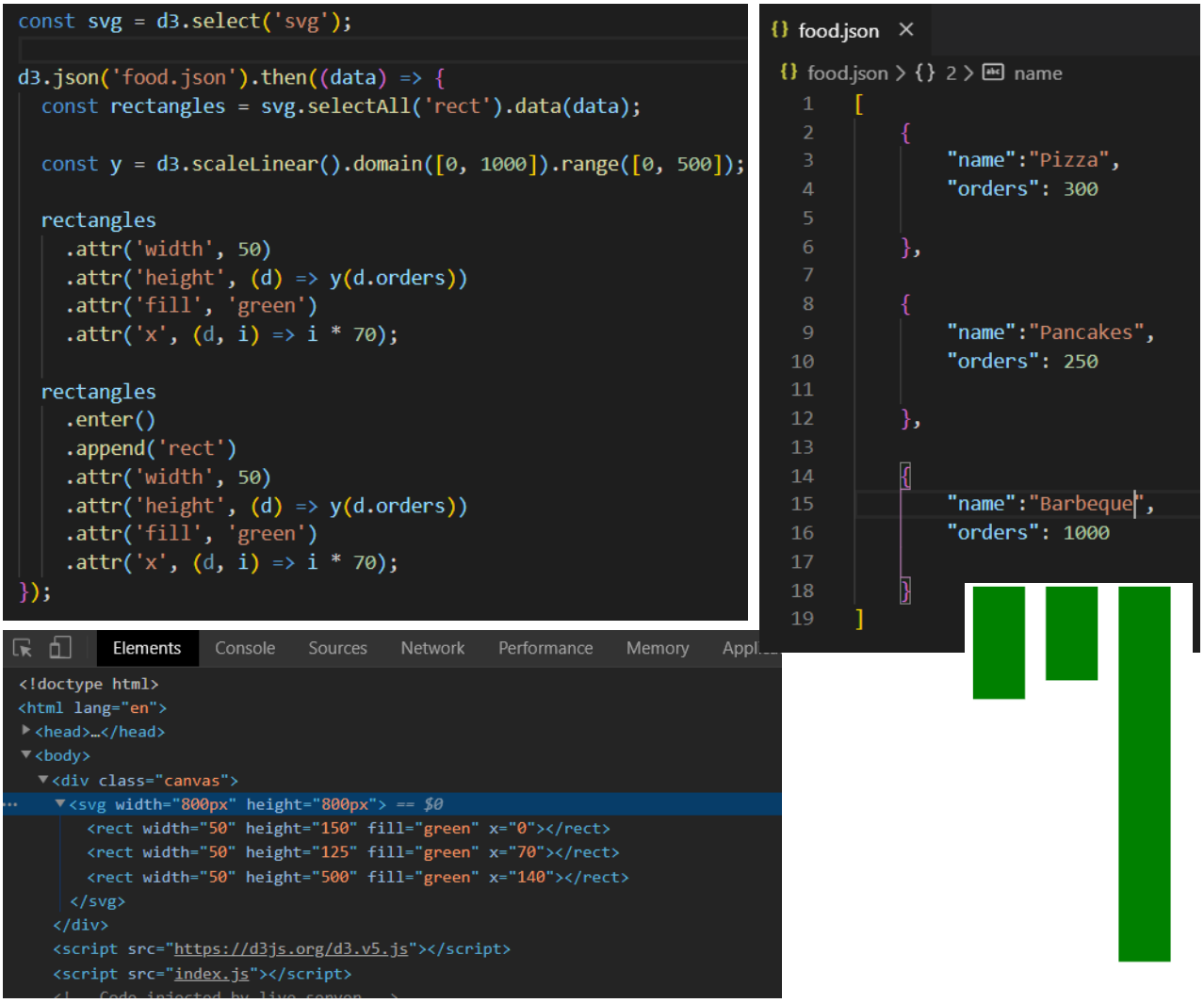
Slika 26. Prikaz linearne skale

Na gornjem primjeru imamo nekakve podatke koje se odnose na namirnice i ukupnu cijenu namirnica što je prikazano pomoću dijagrama na temelju podataka iz JSON-a. Za svaki navedeni objekt u JSON-u imamo jedan pravokutnik koji označava podatke što znači ako je npr. pizza ima order 420 i prikazan je pravokutnik od 420px te tako i za ostale narudžbe. Vidimo da je najveći limit do 500 tj. 500px. No ako recimo količina narudžbi bude preko 500 recimo 2000 pravokutnik će onda otići van dijagrama jer će biti 2000px. Umjesto da kreiramo dijagram koji ima limit do 2000 skaliramo vrijednosti da ih možemo vidjeti u razumnom mjerilu. Koristit ćemo linearno skaliranje.

Slika 27. Linearno skaliranje

Na slici imamo vrijednosti koje su maksimum kao što je 500 i upisane vrijednosti kao što je 2000. Kod linearnog skaliranja vrijednosti upisane više od limita skaliraju se ili smanjuju na vrijednosti unutar limita na primjer ako upišemo vrijednost 2000 ona će postati najviša vrijednost a to je 500. Recimo ako upišemo vrijednost 1000 ona će se

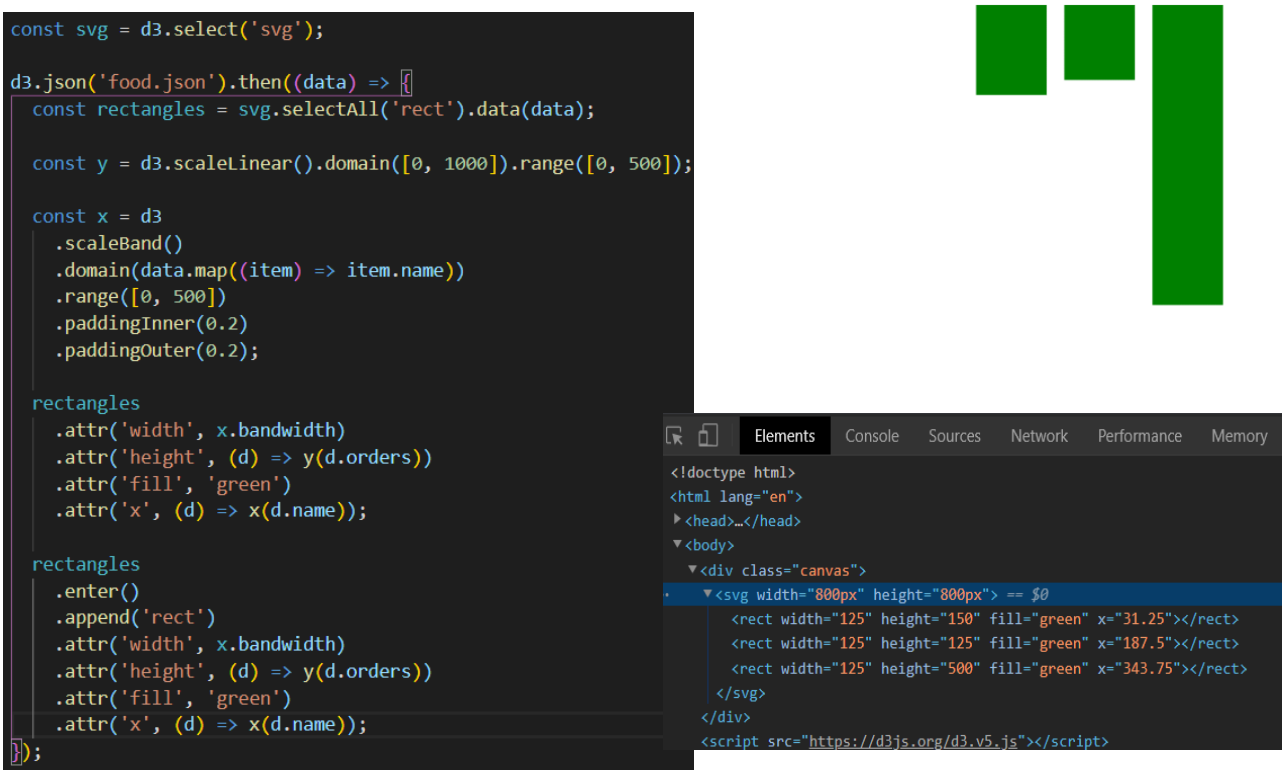
skalirati na 250. Stvari koje upisujemo kao vrijednost zove se domena što je 2000 a ono što se ispisuje na ekranu je domet u ovom slučaju 500.



Slika 28. Linearna skala

Ranije smo vidjeli kako raditi s JSON-om i s virtualnim containerima, sada ćemo proširiti kod s linearnim skaliranjem. Inicijalizirali smo varijablu y i upotrijebili metodu scaleLinear te smo dodali domenu što je vrijednost ili domet vrijednosti koje upisujemo ili povlačimo iz neke baze podataka ili u ovom slučaju JSON-a te range ili domet vrijednosti koje će ispisati na ekranu. Za domenu smo stavili minimum 0 i maksimum 1000 a za range minumum 0 a maksimum 500 te vidimo u konzoli da su vrijednosti drugačije i da visina više nije 1000 nego 500. Stoga vidimo da skaliranje radi i da su vrijednosti sada u maksimumu od 500px. Također smo u atributima dodali da razmak između svakog dodanog pravokutnika bude 70px.

5.2. BANDWIDTH SKALE



```
const svg = d3.select('svg');

d3.json('food.json').then((data) => {
  const rectangles = svg.selectAll('rect').data(data);

  const y = d3.scaleLinear().domain([0, 1000]).range([0, 500]);

  const x = d3
    .scaleBand()
    .domain(data.map((item) => item.name))
    .range([0, 500])
    .paddingInner(0.2)
    .paddingOuter(0.2);

  rectangles
    .attr('width', x.bandwidth)
    .attr('height', (d) => y(d.orders))
    .attr('fill', 'green')
    .attr('x', (d) => x(d.name));

  rectangles
    .enter()
    .append('rect')
    .attr('width', x.bandwidth)
    .attr('height', (d) => y(d.orders))
    .attr('fill', 'green')
    .attr('x', (d) => x(d.name));
});
```

```
Elements  Console  Sources  Network  Performance  Memory
<!doctype html>
<html lang="en">
▶ <head>…</head>
▼ <body>
  ▼ <div class="canvas">
    ▼ <svg width="800px" height="800px"> == $0
        <rect width="125" height="150" fill="green" x="31.25"></rect>
        <rect width="125" height="125" fill="green" x="187.5"></rect>
        <rect width="125" height="500" fill="green" x="343.75"></rect>
      </svg>
    </div>
    <script src="https://d3js.org/d3.v5.js"></script>
```

Slika 29. Bandwidth scale

Prije smo vidjeli kako se skaliraju vrijednosti na našem dijagramu i popravili smo dio što se tiče visine no što ako želimo imati puno više novih objekata u JSON-u onda bi imali i puno pravokutnika u svg containeru koji ne bi stali u takav container. Tu možemo koristiti band skaliranje koje će pomoći da bez obzira koliko novih podataka imali oni stanu u širinu našeg containera. Pošto se radi o širini stvorit ćemo varijablu x te ćemo upotrijebiti scaleBand i da dijagram bude dinamičke prirode upotrijebit ćemo map funkciju koja kao petlja prolazi kroz sve elemente u JSON-u. To radimo jer nikad ne znamo koliko ćemo podataka imati i želimo da se dijagram stvara ovisno koliko ih ima.

Domet je također 500 jer jer će se na temelju tog d3.js računati širinu pravokutnika ovisno koliko podataka imamo. Tu varijablu x ćemo iskoristiti kod atributa širine gdje ćemo dati vrijednost x.bandwith te će se elementi širiti po veličini containera. Također više nemamo vrijednost od 70 px na atributu x vrijednosti nego smo imamo ime

elementa iz JSON-a a razmake između elementa radimo pomoću metoda paddingInner i paddingOuter.

5.2.1. MAX , MIN i EXTEND

Kako vidimo naša domena ima vrijednost 1000 koju smo mi odredili ali u nekom određenom vremenu naša domena može biti premala te možemo imati veće vrijednosti. Za tu priliku možemo iskoristiti metode min(), max() i extend().

Kao što vidimo na slijedećoj slici min i max metodom prolazimo kroz podatke iz jsona s nazivom orders te nam ona vraća minimum ili maksimalnu vrijednost podataka. No extent metoda je kombinacija min i max metode jer vraća minimalnu i maksimalnu vrijednost.

Takve metode možemo uključivati u domene ili gdje nam god trebaju minimalne i maksimalne vrijednosti.

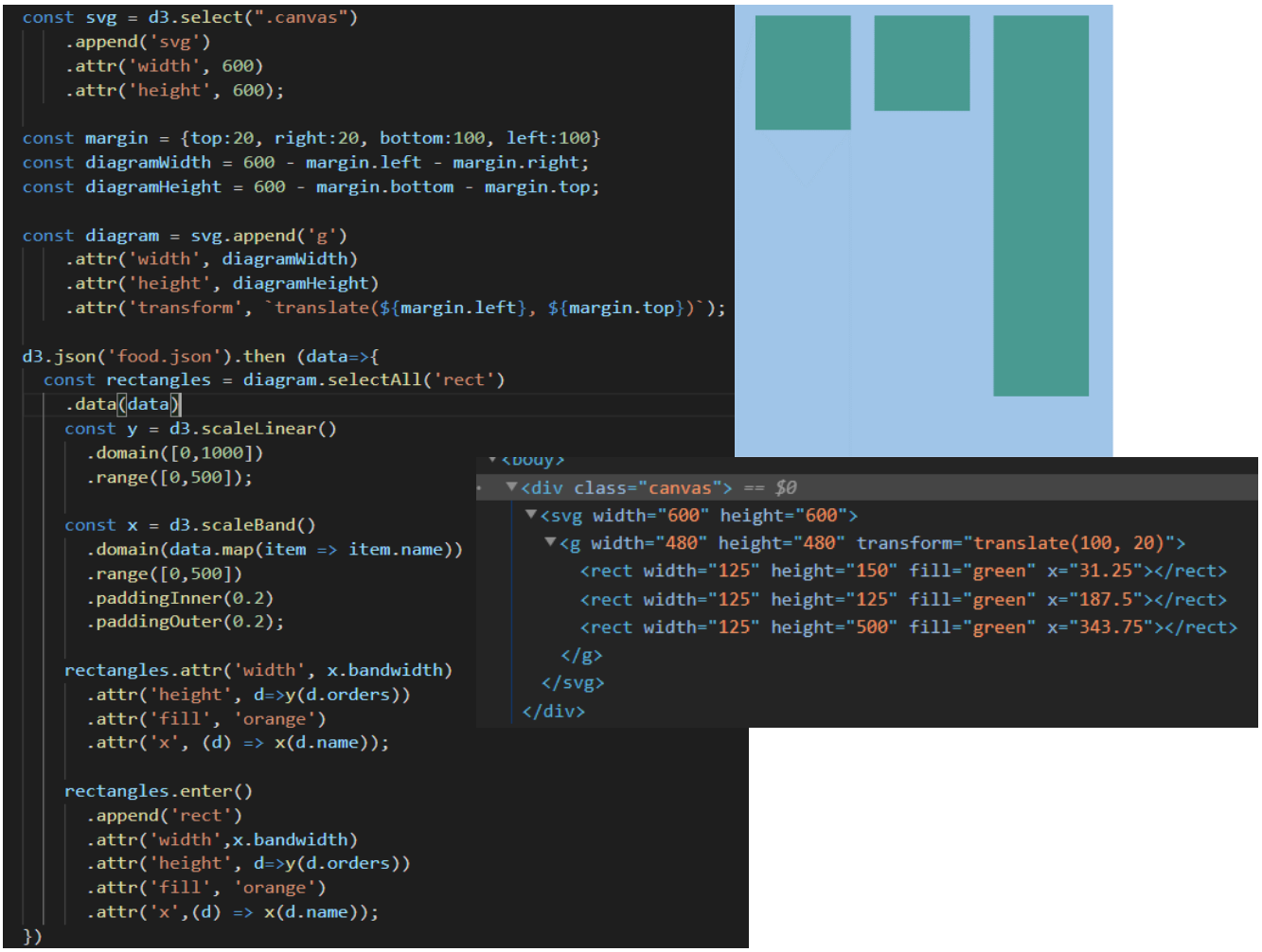
```
const min = d3.min(data, d => d.orders);
const max = d3.max(data, d => d.orders);
const extent = d3.extent(data, d => d.orders);


const y = d3.scaleLinear()
  .domain([0,min])
  .range([0,500]);
```

Slika 30. Min,max, extent

5.2.2. MARGINE

Kako imamo dijagrame u svg containeru vidimo da su uvijek pune širine i visine containera te recimo da imamo podatke ispod dijagrama koje označavaju vrijednosti ne bi mogli ih prikazati jer kako smo i napomenuli dijagram zauzima cijelo platno ili svg container.

Ovaj put u HTML-u nemamo svg container nego ćemo sve napraviti pomoću biblioteke D3.js.



Slika 31. Margine

Pomoću javascripta kreirali smo novi svg container i dali mu atribute visine i širine 600px, Nakon toga smo kreirali varijablu margin koja sadrži objekt sa vrijednostima top, right, bottom ,left.

U varijablama diagramWidth i diagramHeight ćemo od ukupne širine i visine containera oduzeti vrijednosti margina koje imamo u objektu. Zatim stvaramo varijablu diagram u koju ćemo grupirati svg element i dati mu visinu i širinu umanjeno za vrijednosti margina te pomoću transform tranlate namjestiti poziciju lijevo i desno. Pozicije upisujemo pomoću template stringsa koji su dio javascripta gdje možemo nakon znaka $ i unutar uglatih zagrada upisivati vrijednosti iz drugih izvora i varijabli.

Sada vidimo da grupirani elementi su unutar containera koji je ostao 600px umanjeni i imaju margine.Sada kada imamo margine slijedeće pri stvaranju dijagramo moramo

prikazati osi i vrijednosti na njima. Prvo kreirano varijable xGroupAxis i yGroupAxis te ih grupiramo sa dijagramom tj. sa svim elementima.

U blok javascripta gdje su nam podaci iz JSON-a deklariramo varijable xAxis i yAxis u koje spremamo metode axisBottom() i axisLeft() koje sadrže parametre x i y koje smo već deklarirali u bloku. Navedene metode ovisno o oblicima i njihovim pozicijama sadrže pozicije gdje će se nalaziti osi.

Slika 32. Dodavanje osi na dijagram

Sad moramo primjeniti stvorene oblike na grupe koje vidimo u prijašnjoj slici, to radimo na način da pozivamo metodu call() nad varijablama xAxisGroup i yAxisGroup koja kao parametre sadrži varijable u kojima smo deklarirali osi i pozicije i time je d3.js izgenerirao osi. Prikazani rezultat na slijedećoj slici je ono što smo dobili ali to još uvijek nije dobro jer osi nisu još pozicionirane kako treba.

Vidimo da dijagram nije pozicioniran kako bi trebao biti pozicioniran te da se ne vide vrijednosti niti crte koje ih odvajaju kako bi ih mogli razlikovati zato moramo kao što smo i prije iskoristiti css metodu transform:translate nad XAxisGroup koja će kao parametre imati 0 i drugi parametar visinu dijagrama. Na taj način postavljamo x os da se proteže od nulte poziciji u visini dijagrama što je na dnu.

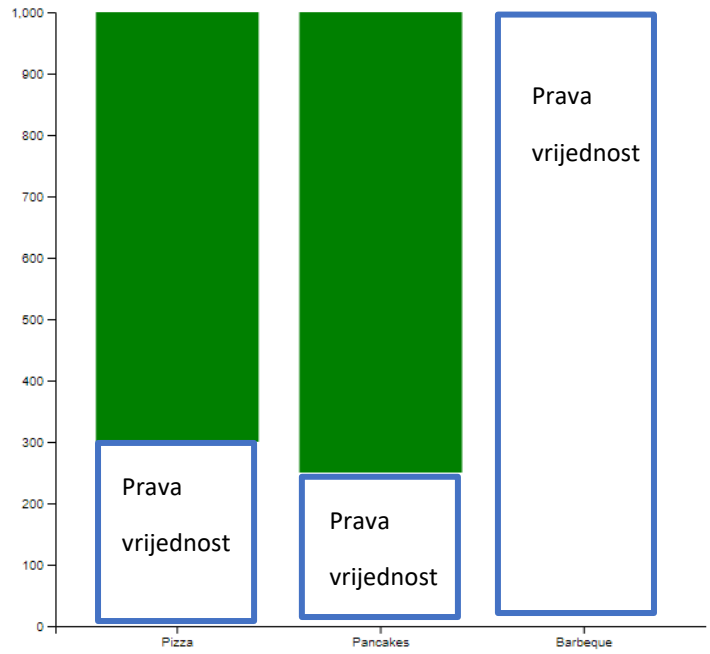
No da bi se vidjeli podaci i da oblici ne prelaze os moramo namjestiti range u varijabli y na visinu dijagrama a što je već izračunato s marginama u graphHeight varijabli.

Sada vidimo da nam je dijagram još uvijek naopačke kao i x os te ih treba popraviti što ćemo jednostavno to napraviti. X os možemo popraviti da samo gdje nam je domet tj. range zamijenimo mjesta početnoj točki i visini.

Slika 33. Podaci na x osi

Slijedeće što moramo napraviti je podesiti visinu elemenata u dijagramu tj. pravokutnike. Moramo ih okrenuti na drugu stranu kao što smo napravili i s osi. Možemo sada primijetiti da prave vrijednosti nisu obojane već su prave vrijednosti nevidljive i trebamo ih ispuniti bojom tj. obrnuti što će nam donja slika bolje pokazati. Kad koristimo d3.js sve se crta od gore prema dolje te zato radimo sve izmjene.

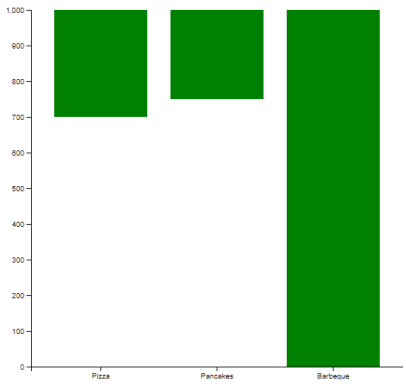
Postoje mnoge možda i jednostavnije biblioteke za kreiranje ovakvih dijagrama no samim ovim primjerima zaključujemo da prava moć ove biblioteke što imamo potpunu kontrolu nad elementima te zato možemo razne i zanimljive vizualizacije podataka.



Slika 34. Vrijednosti dijagrama

To ćemo napraviti da oduzmemo visinu osi od visine zelenog pravokutnika.

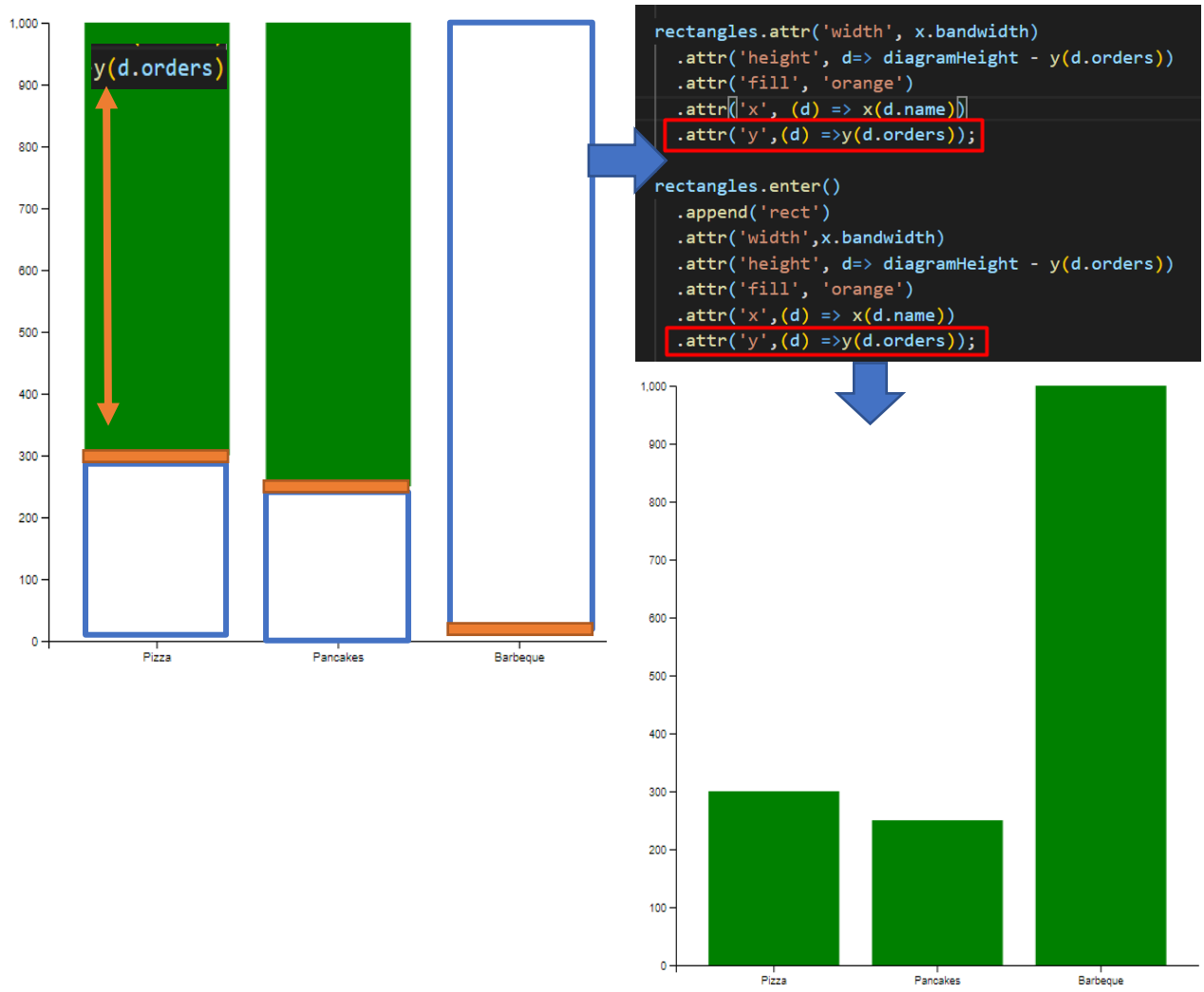
```
rectangles.attr('width', x.bandwidth)
  .attr('height', d=> diagramHeight - y(d.orders))
  .attr('fill', 'orange')
  .attr('x', (d) => x(d.name));

rectangles.enter()
  .append('rect')
  .attr('width',x.bandwidth)
  .attr('height', d=> diagramHeight - y(d.orders))
  .attr('fill', 'orange')
  .attr('x',(d) => x(d.name));
```



Slika 35. Prave vrijednosti dijagrama

Sada smo popravili veličine pravokutnika, pitali bi se zašto ne bi mogli staviti poziciju pravokutnika na drugu stranu samo do d3.js ne funkcionira na takav način već crtanje na platnu se odvija od gore prema dolje što znači da je početna pozicija prave

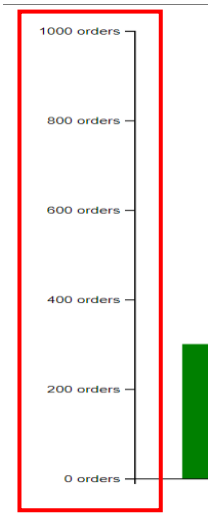
vrijednosti našeg pravokutnika će biti kraj pravokutnika koji je na y osi, te ćemo u kodu njega staviti kao početnu vrijednost y osi.



Slika 36. Pozicioniranje vrijednosti

Već vidimo da je dijagram skoro pri kraju ali morati ćemo još nekoliko stvari popraviti kao na primjer stiliziranje ili prikaz koliko će biti oznaka za podatke na osima. Da bi stilizirali ili kontrolirali količinu oznaka na osima nad samim varijabla gdje su nam x i y os koristit ćemo metodu ticks() koja kao parametar prima broj i prema tom broju d3.js će pokušati najbolje što može prikazati i rasporediti oznake na osima. Mi ćemo staviti taj broj na 4 i vidjet ćemo kako će se brojke rasporediti na slijedećoj slici. Kada to napravimo imamo samo brojke, a da bi dodali nekakvu riječ ili značenje tim brojkama koristit ćemo tickFormat() metodu gdje ćemo na našim brojkama na osi dodati riječ orders.

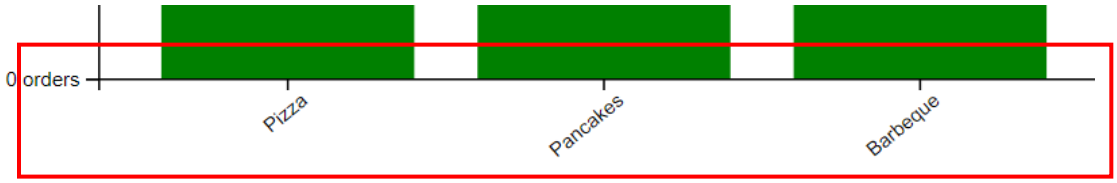
```
const xAxis = d3.axisBottom(x);
const yAxis = d3
  .axisLeft(y)
  .ticks(5)
  .tickFormat(d => d + " orders");
xGroupAxis.call(xAxis);
yGroupAxis.call(yAxis);
});
```



Slika 37.Formatiranje osi

Kada bismo na y osi imali recimo više od 3 stupca nazivi stupaca bi se ispreplitali i možda bi bilo nečitljivo zato možemo i to poboljšati tako da uzmemmo tu grupu koju smo ranije kreirali te sa selectAll metodom selektiramo sav tekst grupe te ćemo pomoću atributa transform rotirati sadržaj za 40 stupnjeva ali kako se tekst zakrenuo sada ga gornja linija prekriva i dio teksta se ne vidi. To se događa jer rotiramo tekst od njegove sredine no ako bi točku rotiranja (text-anchor) stavili na kraj teksta riješio bi se taj problem.

```
xGroupAxis.selectAll('text')
    .attr('transform','rotate(-40)')
    .attr('text-anchor','end');
```



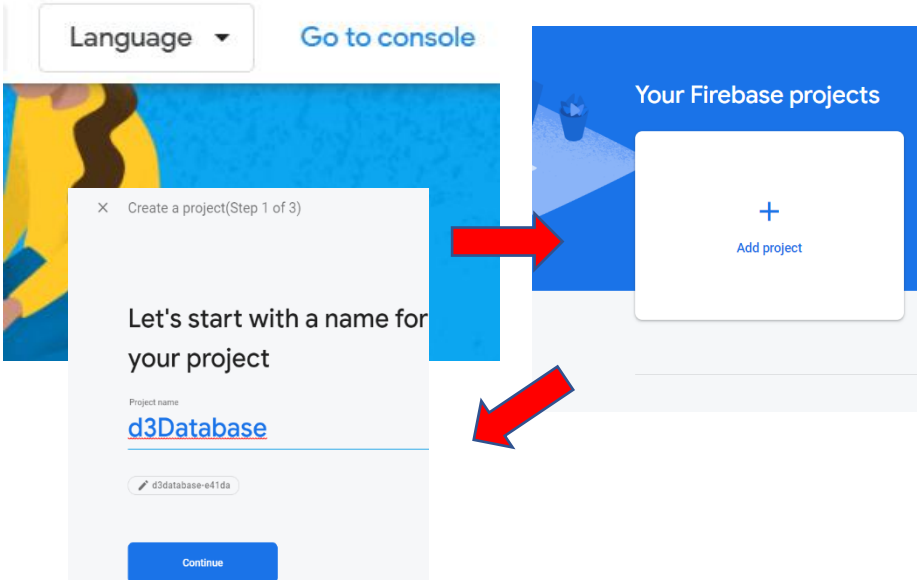
Slika 38. Stiliziranje osi

6. FIREBASE

Napravili smo dijagram i stupce namjestili po potrebi, no kada pogledamo nema baš nekakve svrhe jer smo podatke imali predefinirane u JSON file-u što nije dobro jer nam je cilj imati podatke koje ćemo dobivati iz baze podataka te će se naš dijagram vizualizirani ili prikazani na ekranu kako se mijenjaju podaci u bazi podataka. Za ovu svrhu koristit ćemo Google-ov Firebase. Firebase ima puno opcija kao što je hosting, analitika, autentifikacija itd.. Mi ćemo koristiti Cloud Firestore opciju što je NO-sql baza što znači da se u takvoj bazi podataka ne radi kao s tipičnom relacijskom bazom podataka gdje imamo stupce i redove već imamo kolekcije i dokumente. Svaka kolekcija može sadržavati documente kao npr. kolekcija korisnici će u sebi sadržavati sve dokumente o korisnicima odnosno objekte s svakim korisnikom. Takva baza podataka će nam također automatski generirati id za svaki objekt. Treba napomenuti da je Firestore baza u realnom vremenu (real time database) što znači da ako napravimo izmjene u bazi odmah će se te promjene reflektirati u pregledniku tj. automatski će se ažurirati.

6.1. KREIRANJE FIRESTORE BAZE

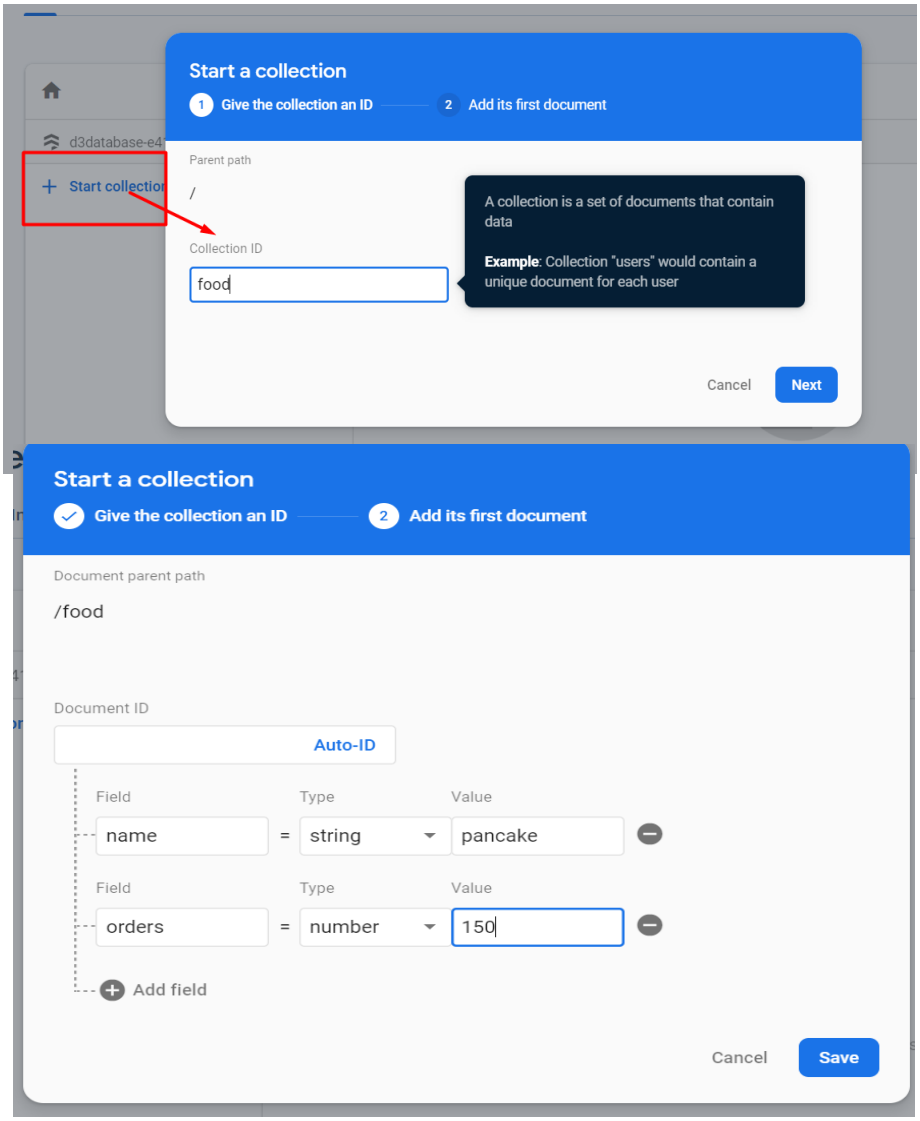
Za pristup Firebase-u moramo samo imati Google račun te je kreiranje vrlo lako. Idemo u gornjem desnom kutu u konzolu te nakon toga samo na add project i slieditи upute za kreiranje kao što je unos imena projekta.

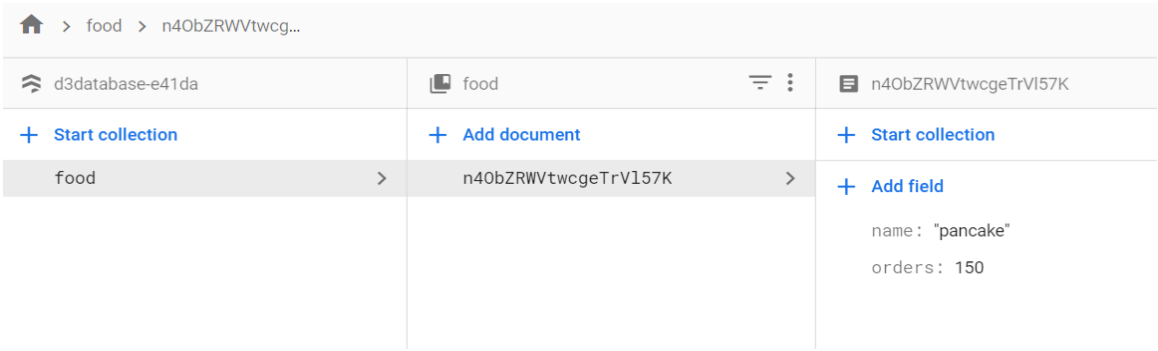


Slika 39.Kreiranje baze

Nakon kreiranja projekta firebase će nas odmah preusmjeriti na kontrolu ploču gdje imamo puno opcija. Imamo opciju koja se se zove database te klikom na nju nam se otvara ekran gdje možemo kreirati našu bazu podataka.

Bazu kreiramo u test mode-u jer onda imamo uključena sva dopuštenja te možemo čitati i pisati u bazu. Da recimo želimo deployati našu aplikaciju uključili bi production mode gdje su pravila drugačija. Zatim slijedimo daljnje upute gdje izaberemo server tj. lokaciju u našem slučaju europa te nakon toga naša je baza kreirana no još uvijek je prazna. Sada kada idemo kreirati novu kolekciju moćemo vidjeti vizualno što je kolekcija i kreirati prvi dokument te kolekcije

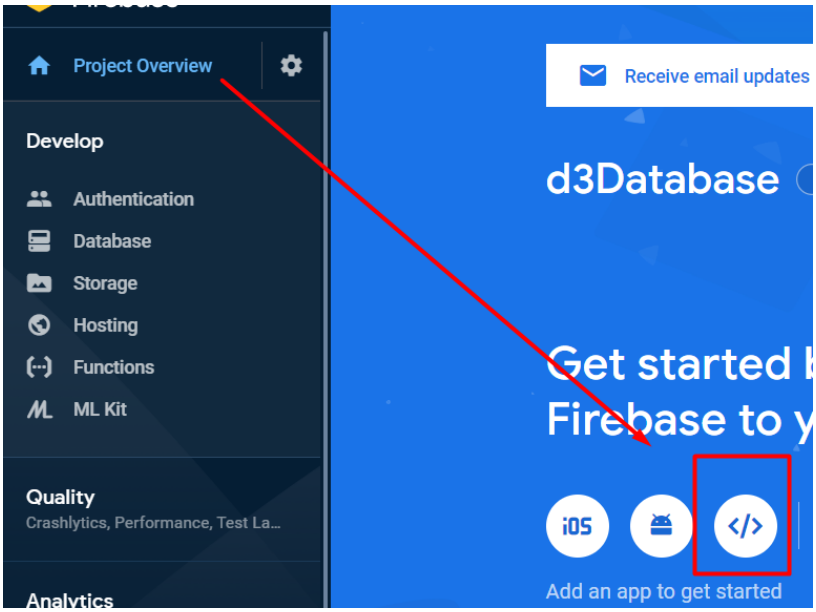




Slika 40. Kreiranje kolekcije i dokumenta

Na gornjoj slici vidimo kreiranje da nakon kreiranja naziva kolekcije kreiramo dokument koji ima svoj id te je automatski generiran te polja koja su ključ-vrijednosti tj. imamo naziv polja, tip polja i vrijednost. Nakon toga možemo vidjeti kolekciju te dukument sa izgeneriranim id-om te je polje u dokumentu koje sadrži ime te broj narudžbi. Kolekcije i dokumente možemo vrlo lako dodavati samo idemo na Add te slijedimo upute kako sto i na prvom kreiranju.

Sada trebamo spojiti našu bazu s aplikacijom



Slika 41. Gumb za konfiguraciju

U našoj konzoli kliknemo na Project overview te znak zagrada te nakon toga upišemo nadimak projekt te nam se otvori izgenirirana konfiguracija koju kopiramo u našu aplikaciju odmah iznad script oznaka u body elementu gdje smo napravili varijablu db

te tamo instanciramo našu bazu koju ćemo koristiti za komunikaciju s bazom podataka.

```
<!-- The core Firebase JS SDK is always required and must be listed first -->
<script src="https://www.gstatic.com/firebasejs/7.8.0/firebase-app.js"></script>

<!-- TODO: Add SDKs for Firebase products that you want to use
 https://firebase.google.com/docs/web/setup#available-libraries -->
 <script src="https://www.gstatic.com/firebasejs/7.8.0/firebase-firestore.js"></script>
<script>
    // Your web app's Firebase configuration
    var firebaseConfig = {
        apiKey: "AIzaSyBGwRmCsmS7hMfMqk54vKq3mH9U3aWYlSM",
        authDomain: "d3database-e41da.firebaseapp.com",
        databaseURL: "https://d3database-e41da.firebaseio.com",
        projectId: "d3database-e41da",
        storageBucket: "d3database-e41da.appspot.com",
        messagingSenderId: "872986633807",
        appId: "1:872986633807:web:5f77e89daba38e3bd72c3a"
    };
    // Initialize Firebase
    firebase.initializeApp(firebaseConfig);
    const db = firebase.firestore();
    db.settings({timestampsInSnapshot:true})
</script>
```

Slika 42. Kreiranje instance firebase-a u html-u

Sada instancu db možemo koristiti u index.js-u gdje se nalazi naš javascript tj. kreiranje dijagramu s d3.js-om

```
d3.json("food.json").then(data => {
  const rectangles = diagram.selectA
```

```
db.collection('food').get().then(res => {
  const data = [];
  res.docs.forEach(doc => {
    data.push(doc.data());
  })
```

Slika 43. Usporedba dohvaćanja podataka

Na gornjoj slici dobivamo podatke iz baze. To radimo na način da ćemo promijeniti kod i tamo gdje smo primali podatke iz JSON-a sada ćemo pomoću instance baze db dohvatiti kolekciju koja se zove food te dohvatiti podatke sa metodom get(). Ako su uspješno dohvaćeni podaci dobivamo odgovor tj. response(res). U responsu se

nalazi puno podataka koji nisu od našeg interesa, a onaj koji je dokument te njega dohvaćamo sa res.docs. No natrag dobivamo sirove podatke te zbog toga pomoću forEach petlje prolazimo kroz svaki podatak te punimo varijablu data koja je tip polje. Kako smo samo upisali jedan dokument u bazi dobivamo dijagram sa jednim dokumentom.

7. AŽURIRANJE

Spomenuli smo da je firebastore baza u realnom vremenu pa bi se ona i trebala ažurirati ako netko u bazu stavi novi podatak. No ako stavimo novi podatak u bazu ona ne prikazuje ažuriran dijagram na ekranu. Tek kada osvježimo stranicu ona se ažurira. Ideja je da slušamo promjene u bazi te ako se promjena dogodi to će se i ispisati na ekranu.

Kako smo prolazili poglavlja spominjali smo i enter selection što je virtualni element koji je vezan za podatke te reprezentira elemente koji još nisu na ekranu ili u pregledniku i tek trebaju biti ispisani.Ideja je napraviti funkciju koja će kao parametar primati podatke iz baze. Prvi korak u takvoj funkciji bi bio da ažuriramo domene, zatim bi trebali elementima pridružiti podatke i maknuti podatke koji su nepotrebni ili ne postoji više. Nakon toga potrebno je reflektirati promjene na ekranu ili klijentu (web preglednik).

Na donjoj slici možemo vidjeti funkciju za ažuriranje. Moramo prepraviti naš kod da bude više dinamičan te da prima podatke iz baze. U prvom koraku ažuriramo podatke na osima, zatim selektiramo sve pravokutnike i pridružujemo im nove podatke. Nakon ažuriranja pravokutnika ako ima elemenata koji ne trebaju biti više na ekranu i nema ih u bazi mičemo ih sa exit selection-om. Nakon toga na ekranu ili u browseru ispisujemo nove pravokutnike koji imaju svoje atribute te ponovno zovemo x i y os da se ažurira.

```
const update = (data) => {
  //1.KORAK pridruzivanje podataka pravokutnicima
  const rects = diagram.selectAll('rect').data(data);
  //2. KORAK --- uklanjanje podataka kojih nema u bazi
  rects.exit().remove();

  //3. KORAK --- Azuriranje domena
  y.domain([0, d3.max(data, (d) => d.orders)]);
  x.domain(data.map((item) => item.name));

  //4. KORAK --- Dodavanje atributa i oblikovanje elemenat koji postoje
  rects.attr('width', x.bandwidth)
    .attr('height', (d) => diagramHeight - y(d.orders))
    .attr('fill', 'green')
    .attr('x', (d) => x(d.name))
    .attr('y', (d) => y(d.orders));

  //5. KORAK --- Dodavanje atributa i oblika elemenata koji ce se tek pojaviti
  rects.enter()
    .append('rect')
    .attr('width', x.bandwidth)
    .attr('height', (d) => diagramHeight - y(d.orders))
    .attr('fill', 'green')
    .attr('x', (d) => x(d.name))
    .attr('y', (d) => y(d.orders));

  xAxisGroup.call(xAxis);
  yAxisGroup.call(yAxis);
};
```

Slika 44. Update funkcija

Kao što smo rekli malo smo preuredili kod te smo osi stavili iznad update funkcije i maknili kod koji se ne odnosi na podatke zato što smo već to napravili u update funkciji

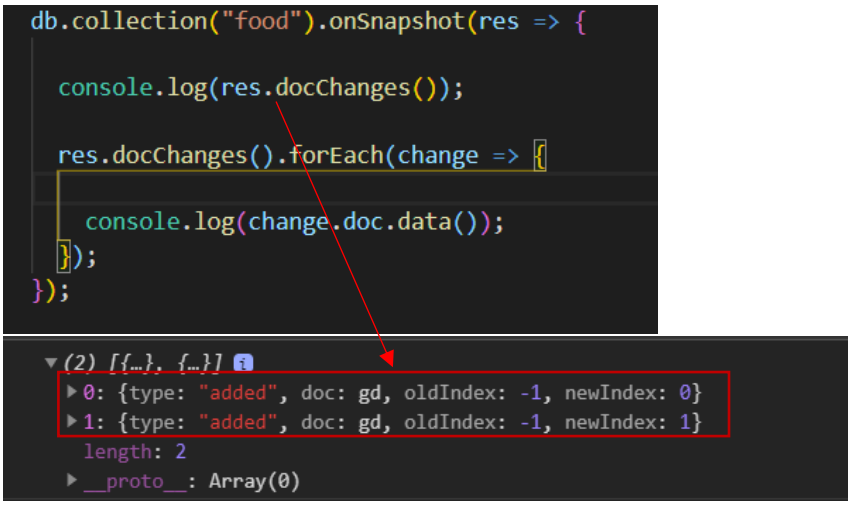
```
const x = d3
  .scaleBand()
  .range([0, 500])
  .paddingInner(0.2)
  .paddingOuter(0.2);

const xAxis = d3.axisBottom(x);
const yAxis = d3
  .axisLeft(y)
  .ticks(5)
  .tickFormat(d => d + " orders");

xGroupAxis.selectAll('text')
  .attr('transform', 'rotate(-40)')
  .attr('text-anchor', 'end');
```

Slika 45. Refaktor koda

7.1. AŽURIRANJE PODATAKA U STVARNOM VREMENU

Sada trebamo slušati promjene u bazi i podatke reflektirati u realnom vremenu reflektirati na ekran.

Na donjoj slici vidimo su da smo iz instance baze db odabrali kolekciju food i pomoću metode onSnapShot slušamo promjene iz baze. Promjene slušamo tako da nam snapshot vraća odgovor ili response(res) te pomoću njega možemo dohvatiti podatke. Promjene osluškujemo s metodom docChanges te nam ta metoda vraća niz objekata. Da bi dobili podatke iz niza moramo iterirati kroz njega a to činimo s metodom forEach koja prolazi jedan po jedan objekt. U funkciji forEach kao parametar dajemo bilo koje ime kojim ćemo pristupati objektima u ovo slučaju naziv je change. Podatke dobivamo tako da pristupamo dokumentu (doc) s metodom data().



Slika 46. Promjene dokumenta

Također možemo vidjeti da kad u konzoli upišemo res.docChanges te inspektamo u pregledniku vidimo da nam vraća spomenuti niz objekata gdje imamo i tip promjene koja nam je vrlo važna informacija. Sada vidimo da je tip „Added“ jer nismo radili s podacima pa se podrazumijeva da su dodani. Recimo da obrišemo dokument u firesbase bazi response bi nam vratio tip „Removed“, također imamo i tip „Modified“ koji nam govori da je dokument mijenjan ili ažuriran.

Slika 47. Id dokumenta u konzoli

Sada inicijaliziramo prazan niz data i objekt doc u kojoj dohvaćamo i pišemo sve podatke dokumenta ali također koristimo spread operator koji se označava sa 3 točke i služi za proširenje postojećeg objekta ili niza. U gornjem primjeru vidimo da smo dodali sve podatke iz baze ali nemamo id. Proširit ćemo postojeći objekt sa ključem id i u njega staviti id dokumenta (change.doc.id) jer nam id treba za identificiranje pojedinog objekta.

```
var data = [];

db.collection('food').onSnapshot((res) => {
    res.docChanges().forEach((change) => {
        const doc = { ...change.doc.data(), id: change.doc.id };

        switch (change.type) {
            case 'added':
                data.push(doc);
                break;
            case 'modified':
                const index = data.findIndex((item) => item.id == doc.id);
                data[index] = doc;
                break;
            case 'removed':
                data = data.filter((item) => item.id !== doc.id);
                break;
            default:
                break;
        }
    });

    update(data);
});
```

Slika 48. Dodavanje id-a

Možemo sada vidjeti da smo u switchu kao argument proslijedili tip promjene koji dobijemo od upita te smo dodali razne slučajeve i zavisno o slučaju manipuliramo s podatkom.

U slučaju added dodajemo podatke u naše polje data, u slučaju modified kreiramo konstantu index koja će sadržavati podatak koji postoji na način da tražimo tražimo jednake podatke s istim id-om u polju i podatka koji smo tražili od baze. Kada nađemo takav podatak na njegovo mjestu zamijenjujemo s novim podatkom.

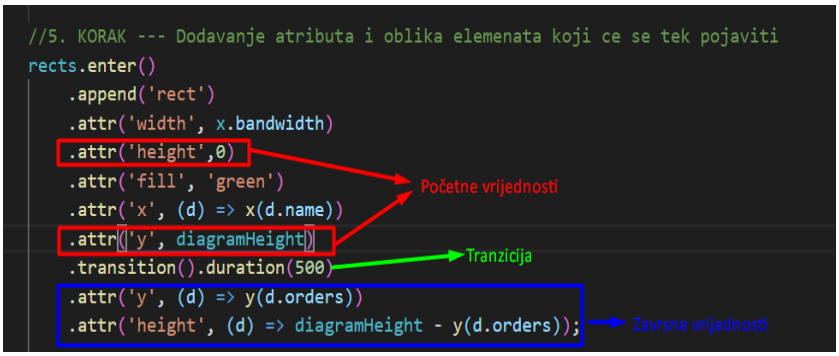
U slučaju removed u polju data samo filtriramo podatke koji nemaju jednak id te ih spremamo u data. Na kraju se poziva funkcija update koja ima za argument data što jest polje koje sadrži podatke i pomoću nje ispisujemo na ekran.

8. TRANZICIJE

U prethodnom poglavlju smo napravili da se naš dijagram ažurira u stvarnom vremenu čim promijenimo nešto u bazi podataka no možemo otići još korak dalje i napraviti da vizualizacija bude animirana te da se prilikom ažuriranja izvrši tranzicija.

Ideja je da naš dijagram to jest pravokutnici rastu on dolje prema gore tj. Od nule prema zadanoj vrijednosti. No s d3.js da bi to napravili moramo promijeniti dva svojstva a to su visina dijagrama tj pravokutnika i y poziciju koja predstavlja gornji dio pravokutnika. Ta svojstva moramo promijeniti u isto vrijeme. Y vrijednost da bude na nuli bit će cijela visina jer kako smo vidjeli u prošlim poglavljima crtamo uvijek od gore prema dolje. Znači početna vrijednost prije nego stranica se učita bit će vrijedost visine koja je 0 a y vrijednost je cijela dužina pravokutnika što je diagramHeight. A vrijednosti prema kojima želimo napraviti tranziciju y pozicije je vrijednost narudžbi(y(d.orders)) a visina je cijela visina pravokutnika umanjena za vrijednost narudžbi (diagramHeight – y(d.orders)).

Prvo što ćemo promijenit redoslijed izvršavanja u enter selectionu jer želimo čim se u čita stranica da se izvrši tranzicija. Redoslijed će biti takav da prvo se učitaju početne vrijednosti koje smo naveli a zatim tranzicija s završnim vrijednostima.



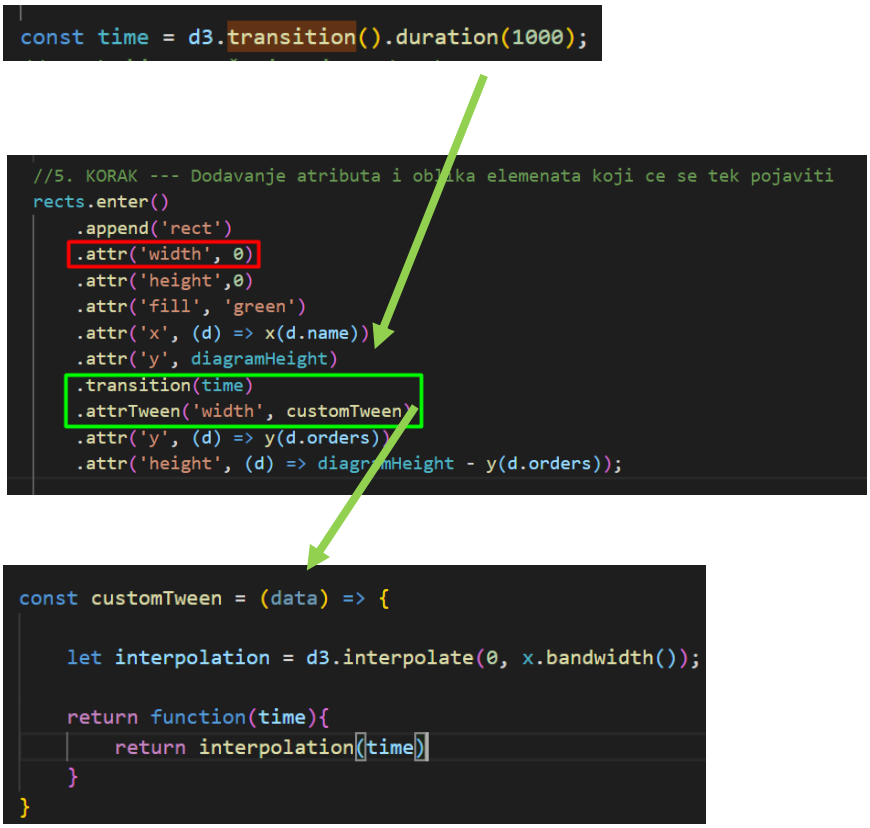
Slika 49. Dodavanje tranzicije

Na gornjoj slici vidimo da smo postavili početne vrijednosti kao što smo već naveli te nakon njih smo odmah stavili metodu transition koja se izvršava 500ms a atributi na tranziciji su završne vrijednosti pravokutnika.

Vidjeli smo kako stvoriti jedan jednostavan dijagram no ako želimo napraviti nešto kompleksnije tranzicije moramo koristiti tweens i interpolaciju. Vidjeli smo da smo već koristili tranziciju no ona u pozadini koristi tweense kako bi se izvela sama tranzicija.

Na primjeru ćemo vidjeti kako napraviti istu tranziciju širine ali ćemo koristiti spomenute tweense a ne već gotovu funkciju transition().

Prvo što vidimo na donjoj slici je da smo napravili refaktor koda gdje da ne moramo ponavljati trajanje d3 tranzicije stavili smo je u varijablu time koju ćemo koristiti na više mjesta.



Slika 50. Tranzicija sa tweenom

Definirali smo funkciju koju smo nazvali customTween te ona prima podatke. Prvo što moramo je definirati interpolaciju koja kao argument prima početnu i završnu poziciju. D3.js vraća funkciju koju smo spremili u varijablu te ćemo nju moći koristiti kako nam bude trebala.

Dalje imamo anonimnu funkciju koja za parametar prima time koje ćemo dalje dodati u funkciju interpolation. Anonimna funkcija će se stalno pozivati dok tranzicija ne bude završena.

Korištenje tweena vidimo u update funkciji nad enter elementima gdje smo širinu pravokutnika postavili na 0 jer će njegova puna širina biti kad završi tranzicija koju smo pozvali i njoj dali atribut attrTween gdje smo naveli koji atribut želimo mijenjati a za drugu argument smo pozvali našu customTween funkciju u kojoj ćemo iskoristiti time varijablu također. Sada pravokutnici kad se osvježi stranica imaju tranziciju ne samo visine već i širine.

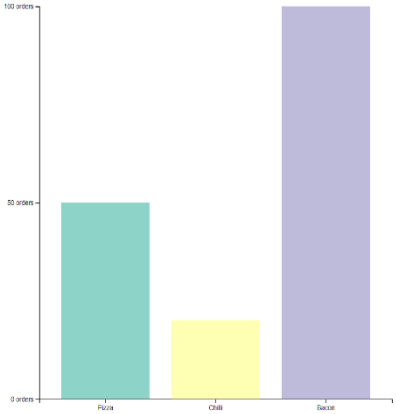
8.1. EVENTI

Sada kada smo vidjeli mogućnosti ove biblioteke ali nismo još imali interakciju korisnika sa našim dijagramom te ćemo vidjeti kako eventi nam pomažu osluškivati događaje koje korisnik stvara. Pod eventima smatramo sve ono što korisnik čini te utječe na elemente na ekranu od primjera najčešći su klik miša, prelazak miša preko elementa, pritisak tipkovnice itd...

Prije nego vidimo evente možemo još malo poboljšati dijagram a to ćemo napraviti tako da iskoristimo još jedno svojstvo d3 biblioteke a to su setovi boja koje nam navedena biblioteka pruža te ćemo svojstva dijagrama obajati u svaku zasebnu boju osvisno o svojstvu.

```
//Set boja iz d3.biblioteke
const colour = d3.scaleOrdinal(d3['schemeSet3']);
```

```
rects.enter()
    .append('rect')
    .attr('width', x.bandwidth)
    .attr('height',0)
    .attr('fill', 'green')
    .attr('x', (d) => x(d.name))
    .attr('y', diagramHeight)
    .attr('fill', d => colour(d.name))
    .transition().duration(500)
    .attr('y', (d) => y(d.orders))
    .attr('height', (d) => diagramHeight - y(d.orders));
```



Slika 51. Postavljanje seta boja

Kao što vidimo u varijablu colours smo spremili set boja pod nazivom schemeset3, a zatim smo pod atribut fill popunili svaki pravokutnik zasebno. Nasumično popunjavamo boje zavisno o imenu naših podataka iz baze jer naš colour prima polje imena te recimo on može biti colour(['pizza','chilli'] ali mi smo dinamički postavili da prima podatke s imenima iz baze jer da ponovimo naša update funkcija prima podatke iz navedene baze.

```
//Handeler functions
const handleMouseEnter = (d, i ,n) => {
    d3.select(n[i])
    .transition().duration(1000)
    .attr('fill', 'red')
}
const handleMouseLeave = (d, i ,n) => {
    d3.select(n[i])
    .transition().duration(1000)
    .attr('fill', colour(d.name))
}
const handleClick = (d,i,n) => {
    const id = d.id;
    console.log(d.id);
    db.collection('food').doc(id).delete();
}
```

```
//Events

const selectAllRects = diagram.selectAll('rect');

selectAllRects.on('mouseover', handleMouseEnter);
selectAllRects.on('mouseleave', handleMouseLeave);
selectAllRects.on('click', handleClick);
```

Slika 52. Eventi

Na gornjoj slici vidimo kako smo prvo u update funkciji u varijalblu selectAllRects selektirali sve naše pravokutnike te smo onda na tu varijablu stavili razne evente. Mouseover event događa se kad korisnik prijeđe kursorom miša preko elementa, mouse leave kad kursor napusti element dok recimo click možemo zaključit već je na klik elementa s mišom.

Uz naš event stavljamo i funkciju koja će se okinuti kada se event dogodi. U našim funkcijama kako bi pristupili trenutnom elementu jer koristimo E6 sintaksu pozivamo polje s indeksom elementa kako smo i vidjeli u početnim poglavljima.

Konvencija je da kada spremamo funkcije u varijable da ih nazivamo tako da ime označava što funkcija radi. Na mouse enter mijenjamo boju elementa tj. pravokutnika, na mouseleave vraćamo prethodnu boju dok na klik miša brišemo element iz baze te tako ga i brišemo i sa ekrana.

8.2. TOOLTIP

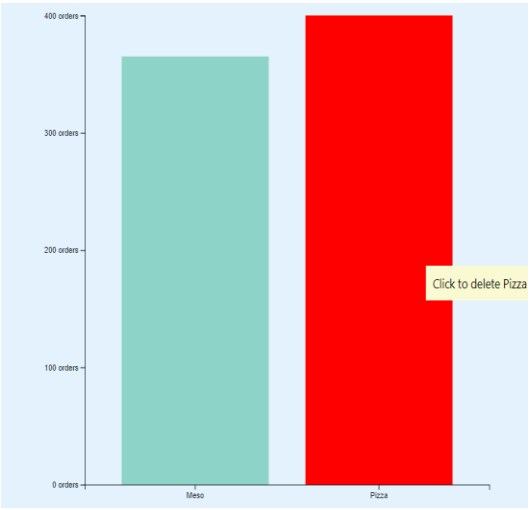
Ponekad je dobro da imamo tooltips ili nekakve upute što korisnik može raditi s dijagramom. Recimo kao što ćemo vidjeti u primjeru možemo na interakciju s dijagramom (klik miša ili bilo koji drugi event) korisniku dati poruku. Da bi koristili tooltip i već gotove funkcionalnosti morat ćemo uključiti još jednu biblioteku tip koju možemo naći na CDN-u (https://cdnjs.com/libraries/d3-tip) što je jedan od vodećih repozitorija za biblioteke.

Biblioteku uključujemo u HTML datoteku na kraju body elementa te nakon toga možemo koristiti navedenu biblioteku.

```
<script src="https://d3js.org/d3.v5.js"></script>
<script src="https://cdnjs.cloudflare.com/ajax/libs/d3-tip/0.9.1/d3-tip.min.js"></script>
<script src="https://cdnjs.cloudflare.com/ajax/libs/materialize/1.0.0/js/materialize.min.js"></script>
```

```
//tooltip
const tip = d3.tip()
    .attr('class','tip-card')
    .html(d => {
            return `<div class="name">Click to delete ${d.name}</div>`
    })
diagram.call(tip);
```

```
    selectAllRects.on('mouseover', (d,i,n)=>{
        tip.show(d,n[i]);
        d3.select('.tip-card').style('top', (d3.event.pageY +30)+'px')
            .style('left', (d3.event.pageX )+'px');

        handleMouseEnter(d,i,n);
    });

    selectAllRects.on('mouseleave', (d,i,n) =>{
        tip.hide();
        handleMouseLeave(d,i,n);
    });
```



Slika 53. Tooltip

Iz našeg koda možemo vidjeti da smo prvo kao što smo već i naveli uključili biblioteku za tootltip zatim u našem javascript datoteci kreiramo varijablu koja će sadržavati poziv tooltipa kojoj dajemo klasu tip card i HTML element. Pomoću html funkcije stvaramo novi html element i pomoću javascript template stringa dinamički stavljamo ime pravokutnika koje će dolaziti iz poziva baze. Tooltip u takvom obliku ne radi ništa nego je samo stvoren ali moramo ga pozvati na neki događaj. U našem slučaju treba biti prelaskom kursora preko pravokutnika i izlazak iz njega. Već imamo takve funkcije ali sad imamo dva poziva pa moramo napraviti refaktor naših evenata. Vidimo da na evente stavili sada anonimnu funkciju koja sadrži podatak, index i polje i već smo vidjeli u prijašnjim poglavljima da trenutnom elementu pristupamo pomoću indeksa polja. Sad u takvoj funkciji vršimo poziv toolitip-a prelaskom kursora preko elementa sa pozivom tip.show(d,n[i]) te smo dohvatili element sa klasom tip-card i preko atributa style mu namjestili poziciju da se nalazi tamo gdje je i kursor miša.Koordinate kursora miša dobili smo preko d3.event poziva(pageY,pageX) . Zatim nakon toga pozivamo našu prijašnju funkciju handleMouseEnter(). Identičan slučaj je i sa funkcijom handleMouseLeave samo tamo sakrivamo tooltip sa poziv tip.hide();

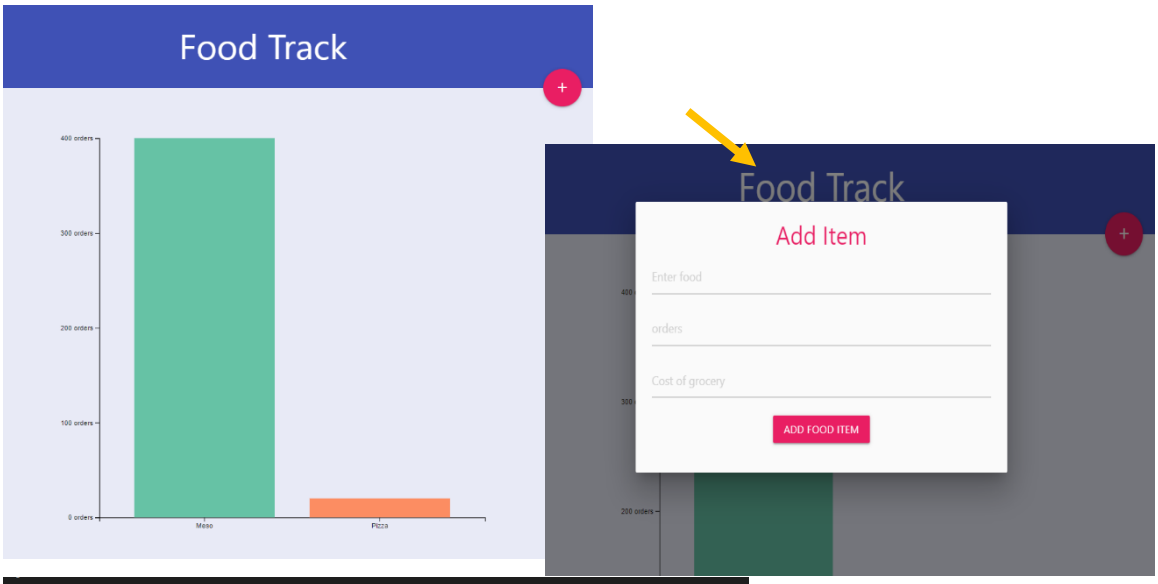
Napravili smo dijagram koji dinamički prikazuje i vizualizira podatke ali to još uvijek nije praktično jer moramo napraviti sučelje da korisnik može unositi takve podatke u bazu te da naš dijagram bude potpuno funkcionalan.

Kako završni rad nije posvećen css i stiliziranju html dokumenta koristit ćemo biblioteku koja će ubrzati taj proces te ćemo samo objasniti što smo napravili. Biblioteka koju ćemo koristiti zove se Materialize koja već posjeduje određene stilove koje ćemo uključivati dodavanjem klasa na html elemente. Materialize biblioteku uključujemo CDN linkom u head HTML dokumenta.

```
<body class="indigo lighten-5">
    <div class="indigo section grey-text" style="position: relative;">
        <h2 class="white-text center">Food Track</h2>
        <a class="btn-floating btn-large halfway-fab pink modal-trigger" href="#modal">
            <i class="material-icons">add</i>
        </a>
    </div>
    <div id="modal" class="modal">
        <div class="modal-content">
            <h4 class="pink-text center">Add Item</h4>
            <form>
                <div class="input-field">
                    <input type="text" id="name" placeholder="Enter food" />
                </div>
                <div class="input-field">
                    <input type="text" id="orders" placeholder="orders" />
                </div>
                <div class="input-field">
                    <input type="text" id="item-cost" placeholder="Cost of grocery" />
                </div>

                <div class="input-field center">
                    <button class="btn pink white-text">Add Food Item</button>
                </div>
            </form>
        </div>
    </div>
    <div class="container">
        <div class="canvas"></div>
    </div>
```

Slika 54. Unos podataka u html-u

Iz gornje slike možemo vidjeti da smo smo napravili header koji u sebi i ima link odnosno button koji otvara modal u kojoj se nalazi forma. Forma sadrži input elemente u koje ćemo upisivati vrijednosti koje ćemo slati pritiskom na element button. Također vidimo da su korištene razne klase materialize datoteke koja omogućava brzo i efikasno stiliziranje elemenata.




```
const modal = document.querySelector('.modal');
M.Modal.init(modal);

const form = document.querySelector('form');
const name = document.querySelector('#name');
const orders = document.querySelector('#orders');
const costPerItem = document.querySelector('#item-cost');
const error = document.querySelector("#error");

form.addEventListener('submit', (event) =>{
    event.preventDefault();
        let totalCost = parseInt(orders.value) * parseInt(costPerItem.value);

        const item = {
            cost:totalCost,
            name:name.value,
            orders:orders.value
        };

        db.collection('food').add(item);
        var instance = M.Modal.getInstance(modal);
        instance.close();

        form.reset();
})
```

Slika 55. Forma za unos podataka

Sada imamo modal koji sadrži formu no još uvijek moramo pomoću javascripta napraviti formu funkcionalnu. U odvojenu index.js datoteku ćemo napisati logiku koja će spremati upisane vrijednosti u bazu i s time prikazivati ih na ekranu u web pregledniku.U tom slučaju ne radimo s d3.js bibliotekom nego sa čistim javascript jezikom ali postoje određene sličnosti. Sa querySelector-om dohvaćamo naše elemente u ovom slučaju po id-u koji smo im dodijelili te onda možemo manipulirati elementima.

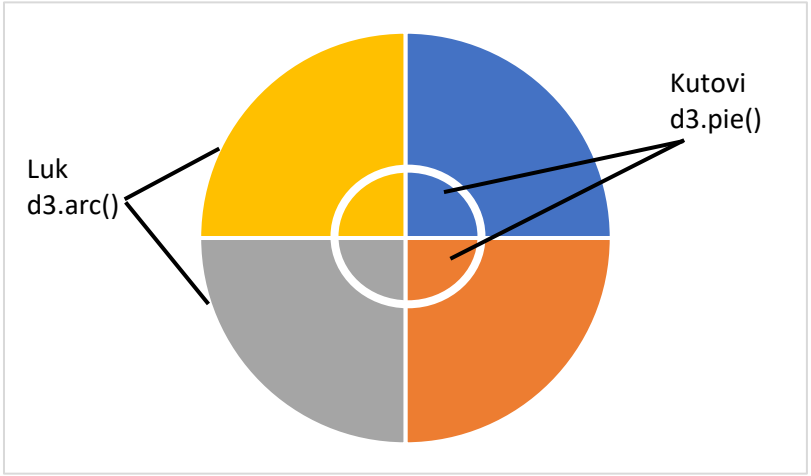
Vidimo da u razne varijable spremamo vrijednosti elemenata koje smo dohvatili tako da lakše možemo kasnije manipulirati s njima. Inicijalizaramo modal na početku te će nam to omogućiti da on bude funkcionalan na naš klik na gumb.

Na form element dodajemo event listener koji osluškuje događaja 'submit' a koji se događa kada se u formi klikne na gumb tj. button element. Također imamo anonimnu funkciju koja za parametar prima objekt našeg eventa iz tog objekta dohvaćamo funkciju preventDefault() što nam omogućuje da spriječimo uobičajeno ponašanje stranice jer inače na klik gumba stranica se osvježava a to u ovom slučaju ne želimo takvo ponašanje.

U bazi smo kreirali još jedno polje cost tipa number koje ćemo koristiti kad budemo prikazivali profit na pie chartu. Vrijednosti elemenata iz forme dohvaćamo sa javascriptom sa metodom value. U total cost varijablu spremamo umnožak narudžbi i cijenu koje pretvaramo u broj sa parseInt jer javascript bi inače to spremio u string. Nakog toga imamo objekt items sa svim tim vrijednostima koje šaljemu u bazu tj. kolekciju food te nakon uspješnog slanja zatvaramo modal i resetiramo formu sa reset().

9. KRUŽNI GRAFIKON

Do sada smo vidjeli jednostavan dijagram no ako želimo napraviti nešto kompleksnijeg oblika ne možemo koristiti samo svg oznake kao što smo koristili sa pravokutnicima nego moramo koristiti path koji smo objasnili na početnim poglavljima



Slika 57. Kružni grafikon

Veličina jednog segmenta kružnog grafikona određuje veličina kuta tj. što veći kut to će biti veći i segment grafikona. Umjesto da sve računamo što je moguće d3.js nudi jednostavnije rješenje tako da recimo uzme naše podatke i pomoću pie generatora generira različite kutove. Kada imamo kuteve moramo onda nacrtati oblike pomoću tih kutova odnosno izgenerirati pathove a za to nam služi arc generator u kojeg stavljamo podatke zajedno s već izračunatih kuteva.

U ovom poglavlju ćemo pokazati samo kako kreirati kružni grafikon jer ostalo je vrlo slično kao što je pozicioniranje,grupiranje i spremanje u bazu kao što smo napravili na stupčastom grafikonu.

```
const dims = { height: 300, width: 300, radius: 150 };
const cent = { x: (dims.width / 2 + 5), y: (dims.height / 2 + 5)};
```

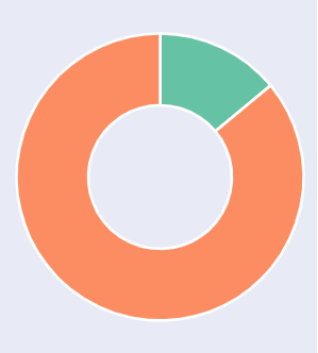
```
const pie = d3.pie()
  .sort(null)
  .value(d => d.cost);
  // vrijednosti koje kreiraju kuteve od podataka

const arcPath = d3.arc()
  .outerRadius(dims.radius)
  .innerRadius(dims.radius / 2);
```

Slika 58. Kreiranje kutova

Prvo kreiramo varijable u kojima će se nalaziti dimenzije grafikona te također koordinate za centriranje istog. Zatim pripremamo varijablu u kojoj inicijaliziramo kutove te vidimo da će vrijednost (value) primati podatke. U arcPath inicijaliziramo luk koji ima atribute unutarnjeg radijusa i vanjskog.

Kao i prije osluškujemo bazu i imamo update funkciju koja se okida kad se izvrše nekakve promjene u bazi podataka. Naša update funkcija prima podatke te prvi korak je izgenerirate kuteve pomoću pie generatora i tih podataka.

```
// pridruzivanje podataka grafikonu
const paths = graph.selectAll('path')
  .data(pie(data));

paths.enter()
  .append('path')
    .attr('class', 'arc')
    .attr('stroke', '#fff')
    .attr('stroke-width', 3)
    .attr('d', arcPath);
```

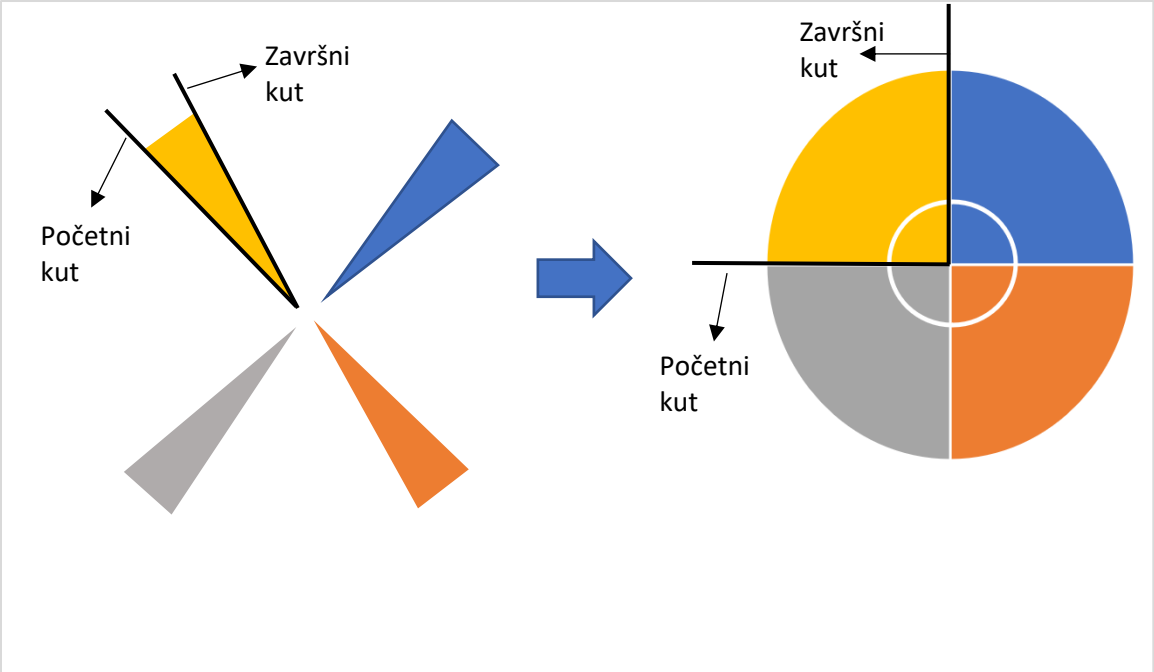


Slika 59. Pridruživanje podataka

U varijablu path selektiramo sve path elemente i u data metodi pozivamo pie generator koji generira kuteve iz podataka primljenih iz baze. Elementi ne postoje te onda stvaramo virtualne elemente sa enter selectionom te im dodijeljujemo atribute i path elemente. Lukove crtamo na ekran s d atributom koji generiramo s pozivom arcPAth funkcije koju smo inicijalizirali ranije. Boje grafikona generiramo jednako kao i ranije s ordinalnom skalom i setom boja koje nudi d3.js.

Da bi grafikon bolje izgledao kada se ažurira i da bude nekakva animacija koristiti ćemo tranzicije ali kružni grafikon je malo kompleksniji od prijašnjeg stupčastog grafikona te moramo naći način kako da izvršavamo tranziciju pathova s jednog stanja gdje je kut manji do stanja gdje je veći. Znamo također da je path samo atribut koji sadrži niz brojeva i slova pomoću kojih se crta svg te ih moramo generirati od početka pa do završetka.

Na donjoj slici vidimo početno stanje iz kojeg želimo vršiti tranziciju suprotno od kazaljke na satu do završnog stanja kada nam je grafikon potpun. Možemo zaključiti da kako vršimo tranziciju suprotno od kazaljke na satu da završni kut je uvijek isti a početni moramo mijenjati vrijednosti pathova. Naša funkcija bi bila i = d3.interpolate(d.endAngle, d.startAngle). Logika je da tijekom određenog vremenskog perioda recimo 10ms svaki put pomoću interpolacije stvaramo nove vrijednosti pathova što će se prikazivati na ekranu a tu će nam pomoći tween koji smo u prošlim poglavljima objasnili.



Slika 60. Tranzicije kuteva

9.1. TRANZICIJE POMOĆU INTERPOLACIJA I TWEENS-a

Na našem prijašnjem dijagramu smo imali jednostavniji primjer gdje smo tranziciju izvršavali samo na visini i širini pravokutnika. Na kružnom grafikonu je malo kompleksnija situacija jer imamo path atribute koji ovise o kutevima kao što smo vidjeli te zato moramo napraviti tranziciji s jednog stanja gdje je kut manji do završnog stanja gdje je kut puno veći. Znamo također da je path samo atribut koji sadrži niz brojeva i slova pomoću kojih se crta svg te ih moramo generirati od početka pa do završetka

9.1.1. TRANZICIJA- ENTER SELECTION

```
const arcTweenEnter = (d) => {
  var i = d3.interpolate(d.endAngle, d.startAngle);

  return function(t) {
    d.startAngle = i(t);
    return arcPath(d);
  };
};
```

```
paths.enter()
  .append('path')
    .attr('class', 'arc')
    .attr('stroke', '#fff')
    .attr('stroke-width', 3)
    .attr('fill', d => colour(d.data.name))
    .transition().duration(750).attrTween("d", arcTweenEnter);
```

Slika 61. Enter selection tween

Sa gornje slike vidimo tranziciju pri enter selectionu tj. koja se događa prilikom otvaranja HTML dokumenta.

Kreirali smo funkciju arcTweenEnter koja za parametar prima podatke i sadrži interpolaciju kojoj je početna pozicija završni kut a završna pozicija početni kut.

Tijekom određenog vremena zvat ćemo tu funkciju koja se nalazi u i varijabli te će svaki put vraćati vrijednost između završnog kuta i početnog kuta. U početku završni i početni kut imaju istu vrijednost.

Zatim imamo anonimnu funkciju koja za parametar prima vremenski razmak koji je između 0 i 1 i ta funkcija ažurira vrijednost početnog kuta tijekom vremena te na kraju vraća vrijednost za crtanje patha (arcPath(d)). Na gornjoj slici također možemo vidjeti da u enter selectionu imamo tranziciju sa trajanjem od 750ms te pozivamo funkciju arcTweenEnter koja se nalazi u atributu attrTween.

9.1.2. TRANZICIJA- EXIT SELECTION

Sada imamo tranziciju čim se učita html dokument ali kad brišemo neki segment kružnog grafikona želimo isto napraviti tranziciju ali suprotnu od enter selection tranzicije koju smo već pojasnili.

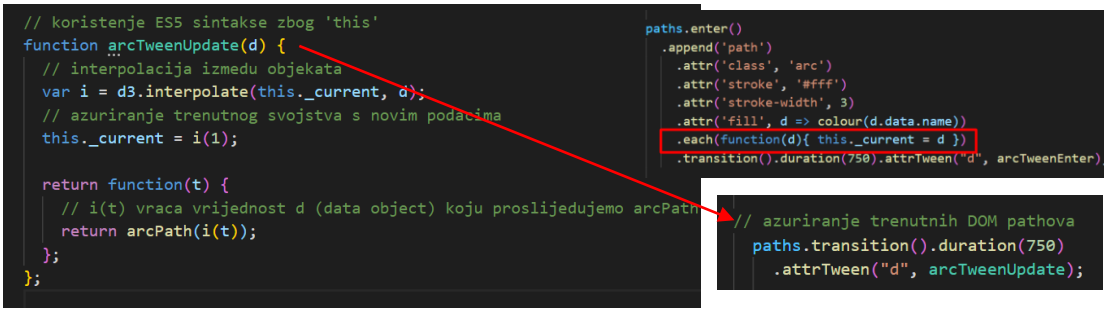
```
const arcTweenExit = (d) => {
  var i = d3.interpolate(d.startAngle, d.endAngle);

  return function(t) {
    d.startAngle = i(t);
    return arcPath(d);
  };
};
```

```
// exit selection
paths.exit()
  .transition().duration(750)
  .attrTween("d", arcTweenExit)
  .remove();
```

Slika 61. Exit selection tween

Iz navedenog zaključujemo da smo samo zamijenili u interpolaciji završni i početni kut i funkciju pozivamo na exit selection-u.

9.1.3. TRANZICIJA -AŽURIRANJE PODATAKA



Slika 62. Update tween

U ovom primjeru ćemo koristiti ES5 sintaksu javascripta kako bi mogli koristiti ključnu riječ this jer onda se odnosi na trenutni element. Prvi korak je da moramo uvijek imati ažurirane podatke za tranziciju pri ulasku na stranicu te spremljene negdje te podatke. Pomoću each metode funkcija će se okidati za svaki element zasebno. Navedena anonimna funkcija prima podatke trenutnog elementa te u njoj se referiramo na element i inicijaliziramo svojstvo current s tim podatkom što znači na početku imamo zadnje stanje iz baze.

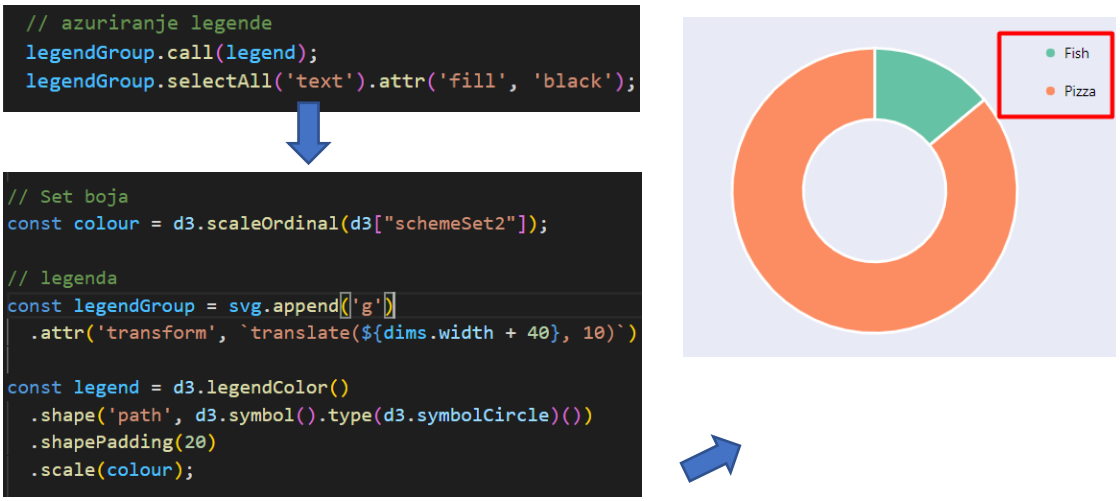
Novo stanje dobivamo kada se okine update funkcija i onda tranzicija gdje se poziva arcTweenUpdate. U tween-u vršimo vidimo da je u interpolaciji početna točka this._current što je zadnje stanje iz baze a završna točka novo stanje. I nakon toga uvijek moramo ažurirati prvotno stanje s tim novim stanjem jer će pozivom na tween to sada biti starije stanje.

9.1.4. LEGENDA

Da bi smo znali što koja boja predstavlja u kružnom grafikonu možemo napraviti dinamičku legendu koja će nam to omogućiti.

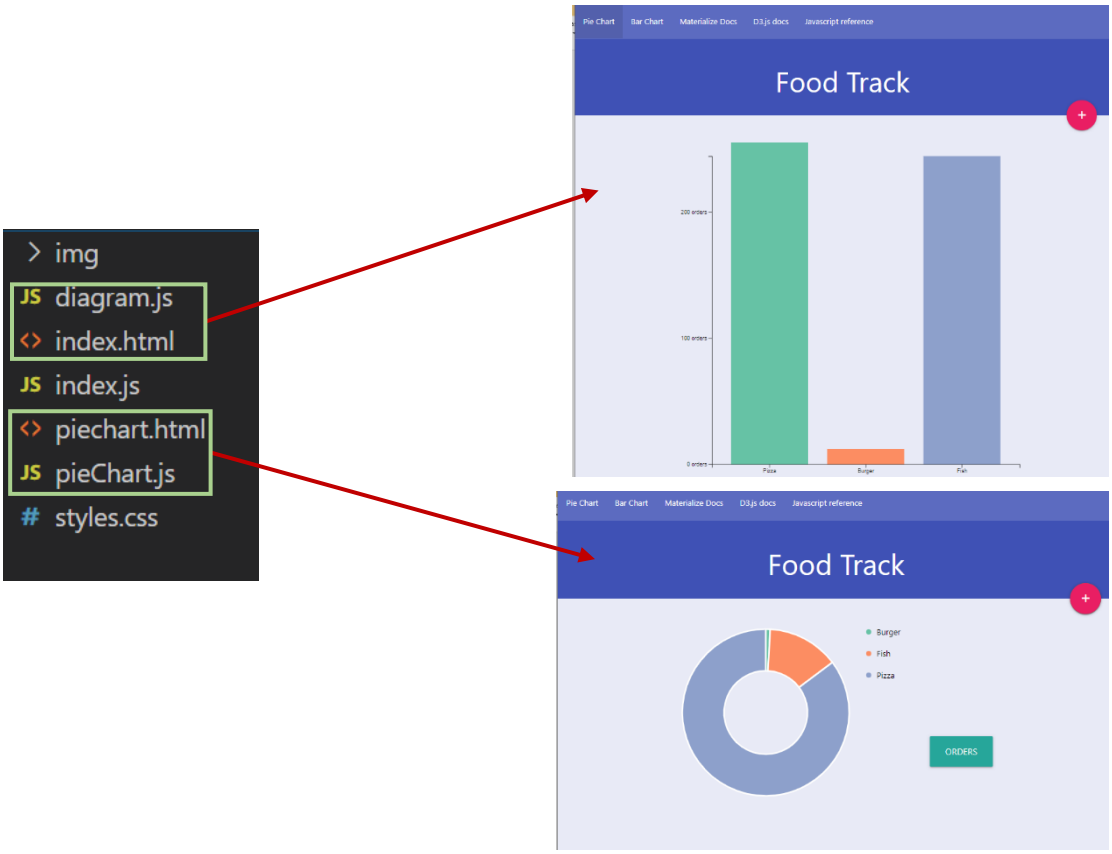
Kao što smo već vidjeli za legendu ćemo također koristiti biblioteku sa cdn-a (https://cdnjs.cloudflare.com/ajax/libs/d3-legend/2.25.6/d3-legend.min.js) koju uključujemo na dnu body elementa.

Prvo korak je pozicioniranje naše legende ali moramo je grupirati da je možemo pomicati kao cjelinu.



Slika 63. Legenda

Nakon što smo grupirali i pozicionirali legandu sa css transform translate dodajemo boje oblik i razmak. Možemo zaključiti kao i kod stupčastog grafikona u scale kao parametar stavljamo colour a što su skalirane vrijednosti. U update funkciji vršimo poziv legende.



Slika 64. Struktura projekta

10. ZAKLJUČAK

Cilj ovoga rada je da kroz objašnjenja i kreiranja manjeg projekta imamo uvid što se pomoću ovakve biblioteke može napraviti ali i da se steknu osnovna znanja o istom.

Pošto je ova biblioteka dio javascripta i pošto najčešće prikazujemo podatke u web pregledniku potrebno je savladati osnovna znanja css-a, HTML-a i javascripta.

Rad pokazuje skraćeni dio ili uvod u biblioteku no ona je mnogo opširnija i može biti mnogo kompleksnija. Također smo vidjeli da svakim dijelom grafikona možemo pristupiti i manipulirati s njim što se čini kompleksnije za učenje i pisanje koda no u završnici imamo potpunu kontrolu sa dijagramima te možemo stvarati razne zanimljive dijagrame ili grafikone.

Možemo zaključiti da ovakva biblioteka ima puno prednosti kao što su vrlo lijepe vizualizacije samim time što radimo sa svg elementima koji ne gube kvalitetu, stvaranje interaktivnih dijagrama , transformacije i tranzicije, kontrola nad virtualnim elementima, puno primjera na službenoj stranici, biblioteka je open source i ima dobru dokumentaciju.

Vizualizacija podataka je vrlo bitna tema jer pomoću takve vizualizacije se vrše razna praćenja podataka, donose vrlo bitne odluke i razmatranja. Postoje mnogo biblioteka koje se bave ovim problemom no d3,js je snažno ostavio utisak u svijetu developmenta zbog fleksibilnosti, laka je za naučiti a i open source je biblioteka.

Projekt i korišteni materijali nalaze se na githubu. Stranica projekta može se pogledati pomoću poveznice https://stastnyd.github.io./ a korištene slike i kod na

https://github.com/stastnyd/stastny-d3js.

11. LITERATURA

- D3.js službena stranica: https://d3js.org/
- SVG & D3.js Basics: https://www.youtube.com/watch?v=IpK82JaiG3A
- D3.js Practical introduction: https://www.youtube.com/watch?v=TOJ9yjvlapY
- Learning D3: https://www.youtube.com/playlist?list=PLillGF-RfqbY8Vy_G5WxXwhZx4eXI6Oea
- Tutorialspoint: https://www.tutorialspoint.com/d3js/index.html
- Javascript dokumentacija: https://developer.mozilla.org/en-US/docs/Web/JavaScript
- Materialize dokumentacija: https://materializecss.com/